RSX-11M
FORTRAN-IV User's Guide

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

RSX-11M Version 1

Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation · maynard, massachusetts

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

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

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

ASSOCIATED DOCUMENTS

Refer to User's Guide to RSX-11M Manuals, DEC-11-OMUGA-A-D.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

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

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

CONTENTS

PREFACE

This document provides information necessary to compile, task build, execute, and debug a FORTRAN program under the RSX-11M operating system. Chapter one describes the operational procedures. Chapter two provides information about the Object Time System (OTS). This system is a collection of routines selectively linked to the user's program which perform certain arithmetic input/output, and system dependent service operations. It also detects and reports run-time error conditions. Chapter three describes system dependent information not included in the PDP-11 FORTRAN Language Reference Manual. The Appendices provide reference information about internal data representations, system subroutines, error diagnostics, and compatibility of FORTRAN-IV with other PDP-11 FORTRAN processors.

This manual should be used only after some knowledge of the FORTRAN Language, as implemented on the PDP-11, has been acquired. The associated document which may be used for this purpose is titled PDP-11 FORTRAN Reference Manual (DEC-11-LFLRA-A-D). The user should also be familiar with the RSX-11M Operating System as described in the RSX-11M Operations Procedure Manual (DEC-11-OMOGA-A-D).

NOTE

Subroutines are provided in the FORTRAN library to enable real-time process control, process I/O, and RSX-11M system directives to be performed by means of FORTRAN. These routines and their ISA-standard calls are fully described in the RSX-11M Executive Reference Manual and the RSX-11M I/O Drivers Manual.

DOCUMENTATION CONVENTIONS

All RSX-11M executive and system program command lines are terminated by a RETURN. Since this is a non-printing character, at certain places in the text the notation <CR> represents the RETURN key.

Some special keyboard characters require that the CTRL (control) key be pressed simultaneously with a second character. These characters are denoted by ↑ (up arrow); e.g., ↑Z (CTRL Z).

In all examples, text printed by the RSX-11M executive or system program is underlined, text typed by the user is not underlined.

CHAPTER 1

OPERATING PROCEDURES

1.1 USING THE FORTRAN-IV SYSTEM

Three steps are required to transform a FORTRAN source program into an executing Task. These steps are:

1. Compilation
2. Task Build (or Linkage)
3. Initiation of Execution

Compilation is accomplished by invoking the FORTRAN-IV Compiler. Next, the Task Builder is used to construct a task image. Finally, task execution is initiated by using the appropriate Monitor Console Routine (MCR) commands. Each step in this procedure involves several required and optional command inputs and produces an output to be used in the next step. Other optional outputs are also possible.

A diagram depicting this process is given in Figure 1-1. The remainder of this chapter discusses each step of the process in detail and gives some simple examples.

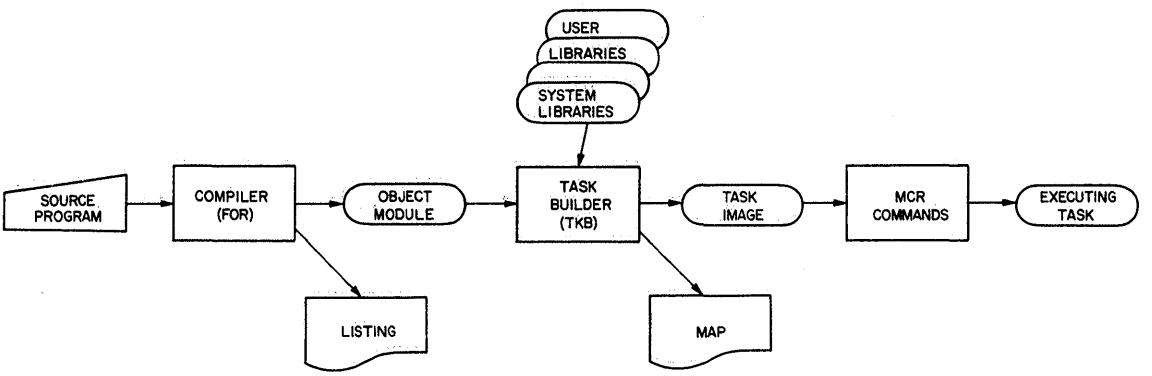


Figure 1-1
Preparing a FORTRAN Source for Execution

1.1.1 Command String

System programs running under RSX-11M usually require a command string to specify such things as input files and various options. A standard command string has one of the following forms:

```
output-files-list=input-files-list
                    or
@command-file
```

Both output-files-list and input-files-list are strings consisting of file specifications separated by commas. Each file specification selects a file to be used as an output or input file by the system program. A @command-file is an indirect command file which contains multiple command strings.

Any number of file specifiers is possible, the actual number being determined by the system program which will use the file command string.

1.1.2 File Specifications

Each file specifier (whether input or output) has the following format:

```
dev:[g,o]filename.type;version/switch1.../switchn
```

where:

dev: = The physical device unit on which the volume containing the desired file is mounted, e.g., DK0:. The name consists of two ASCII characters followed by an optional one or two digit (octal) unit number and a colon. The default value is SY:, the system disk. Table 1-1 depicts the legal device list.

[g,o] = The User Identification Code (UIC) associated with the user file directory containing the desired file. This consists of a group number and a user number. The default value of UIC is the identification code under which the system program is running, usually [200,200].

filename = The name of the file. In RSX-11M, a filename can be up to nine alphanumeric characters in length. Filename and type are always separated by a period (.).

type = A means of distinguishing among forms of one file. System programs default the file type to an appropriate standard type, (e.g., FTN) so the typical user will not need to explicitly specify it. File type and version are always separated by a semicolon (;).

version = An octal number used to differentiate among versions of a file. When a file is first created using the editor, it is assigned a version number of 1. If the file is subsequently opened for editing, the file system retains the original file for backup and creates a new file with the same filename and type, but with a version number of 2. Version is in the range 0-77777.

/switch = A one or two character ASCII name identifying the switch option. The switch itself may

have three forms. If the switch designator, for example, is SW. Then:

```
/SW sets the switch action;
/-SW, negates the switch action, and
/NOSW also negates the switch action.
```

in addition the switch identifier may be followed by any number of values. The permitted values are ASCII strings, octal numbers, and decimal numbers. The default for a value is octal. Decimal values must be terminated by a decimal point. Values preceded by a number sign (#) are octal; the octal option is included for explicit documentation purposes. Any numeric value may be preceded with a + or - sign; if the number sign (#) is used, the + or - must precede it. The following are valid switch specifications.

```
/SW:27:MAP:29.
/-SW
/NOSW:NOSWITCH:-#50
```

Although the filename command string has a standard syntax, the interpretation of the string and permissible options is program dependent.

Table 1-1
RSX-11M Devices

PERIPHERAL DEVICES	DEVICE-UNIT
Analog to Digital Converter (AD01-D)	ADnn:
Analog to Digital Converter (AFC11)	AFnn:
Card Reader (CR11)	CRnn:
Cassette (TA11)	CTnn:
DECtape (TC11)	DTnn:
Disk (RP04)	DBnn:
(RF11)	DFnn:
(RK11)	DKnn:
(RP03)	DPnn:
(RS03/RS04)	DSnn:
Laboratory Peripheral System (LPS11)	LSnn:
Line Printer (LP11/LS11/LV11)	LPnn:
Magtape (TU16)	MMnn:
Magtape (TM11)	MTnn:
Synchronous Line Interface (DP-11)	XPnn:
(DU-11)	XUnn:
Asynchronous Line Interface (DL11-E)	XLnn:
Terminal (DL11/DH11/DJ11)	TTnn:
Universal Digital Controller (UDC11)	UDnn:
PSEUDO DEVICES	
Console Listing	CL:
Console Output	CO:
Pseudo Input Terminal	TI:
System Default Device	SY:

1.2 USING THE FORTRAN-IV COMPILER

The FORTRAN-IV Compiler is an RSX-11M system program initiated via the FOR MCR-command. It produces relocatable object modules from FORTRAN source programs.

To invoke the Compiler enter the following MCR command. (All characters typed by the system are underlined).

```
↑C
MCR>FOR<CR>
```

The FORTRAN-IV Compiler then prints "FOR>" to indicate that it is ready to accept a command string.

A standard command string as described in Section 1.1.1 is used to specify input and output files to the Compiler. None, one or two output files may be specified. The first output file is the object module file and the second is the listing file. Only one input file may be specified. This is the FORTRAN source file. It may contain one or more FORTRAN program units (Main Program and/or Subprograms). File types default as shown in Table 1-2.

Table 1-2
FORTRAN Compiler Default File Types

File	Default Type
Object	OBJ
Listing	LST
Source	FTN
Command	CMD

Up to two levels of indirect command files are permitted.

An example FORTRAN command sequence is shown below:

```
>FOR
FOR>OBJECT,LIST=FILE1
```

This command string directs the Compiler to take the source file FILE1.FTN from the system device and output the files LIST.LST and OBJECT.OBJ to the system device. When the compilation is complete the Compiler again prompts with "FOR>". At this point another command string may be entered or ↑Z may be typed to cause the Compiler to terminate execution.

An alternate way to specify a single command line is to enter the command string on the line containing the MCR command. For example the following performs the same function as the example above:

```
>FOR OBJECT,LIST=FILE1
```

Only a single compilation per FOR command may be specified in this manner. Upon completion of the compilation the compiler exits to the RSX-11M Executive which will prompt with ">".

Either of the output files can be omitted by omitting the file specification from the command string. For example:

```
>FOR
FOR>FILE1=FILE1
```

produces FILE1.OBJ on the system device but no listing file.

```
FOR>,LP:=FILE1
```

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

1.2.1 Compiler Switches

The FORTRAN Compiler command strings may contain switch options on the input and output file specifications. The switches are as follows:

Switch	Description
/LI:n	Specifies the listing options. The argument n is encoded as follows:

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

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

```
/LI:7 or /LI:ALL or /LI
```

requests a source listing, a storage map listing, and a generated code listing. If this switch is omitted the default list option is /LI:3, source and storage map. (See section 1.2.2.) If no listing output is specified the following is assumed:

```
TI:LIST.LST/-LI
```

/SP	Automatically spool listing file. The default (/SP) is to spool.
/DE	Compile lines with a D in column one. These lines are treated as comment lines by default (/-DE) (see section 1.6).
/EX	Read a full 80 columns of each record in the source file. Only the first 72 columns are read by default (/-EX).
/ID	Print FORTRAN identification and version number. The default (/-ID) causes the identification and version number not to be printed.
/OP	Enable the Common Subexpression (CSE) Optimizer. In general the CSE optimizer will make the program run faster. However, the size of the program may be different than with no optimization (/-OP). The default is to optimize (/OP).

/SN — Include internal sequence numbers (ISN). The /-SN option reduces storage requirements for generated code and slightly increases execution speed but disables line number information during Traceback. The default (/SN) is to use ISNs.

/I4 — Two word default allocation for integer variables. Normally, single storage words will be the default allocation for integer variables not given an explicit length specification (i.e., INTEGER*2 or INTEGER*4). Only one word is used for computation. /-I4 is the default.

/VA — Enable vectoring of arrays (see section 2.5). The default (/VA) is to vector arrays.

/WR — Enable compiler warning diagnostics. The default (/WR) is to issue warning diagnostics.

Switch default summary:

```
/LI:3/SP/-DE/-EX/-ID/OP/SN/-I4/VA/WR
```

1.2.2 List Formats

There are three optional sections that may be included in the list file. By default the source program and the storage map are included. The generated code may also be included. Any combination of these sections can be requested by using switches in the Compiler command string (see section 1.2.1). A description of each section is given below. Figure 1-2 provides a sample of the information included in each section.

1.2.2.1 Source Listing - The source program unit is listed in this section as it appeared in the input file. Internal sequence numbers are added by the compiler for easy reference. Note that internal sequence numbers are not always incremental. For example the statement following a logical IF will have an internal sequence number 2 greater than that of the IF. The IF statement has internally been assigned 2 sequence numbers: one for the comparison and one for the associated statement. Comments, uncompiled statements, and continuation lines do not receive internal sequence numbers.

1.2.2.2 Storage Map Listing - This section includes a list of all symbolic names referenced in the program unit. A location offset from the base of the program unit (subject to relocation during task building) is given for all local symbols. There is also a description of the symbolic name including usage, data type, and in the case of COMMON blocks and array names, the defined size.

NOTE

Blank COMMON is described as COMMON BLOCK / / in the storage map, but is

located on a Task Builder map under the Program Section named .$$$$.

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

```
FORTRAN IV       GENERATED CODE

ISN #0006
000066  LSN$     #000006
000072  MOI$M$   000003
000076  DII$I$   #000002
000102  CPI$$M   000064
000106  CFI$
000110  ADF$I$   #040640
000114  MOF$$M   000030

ISN #0007
000120  ISN$
000122  MOF$M$   000030
000126  DIF$I$   #040400
000132  ADF$M$   000020
000136  CDF$
000140  MOD$$M   000044

ISN #0008
000144  ISN$
000146  REL$     000024
000152  REL$     000030
000156  CAL$     #000002 CMPLX=#000000
000164  MOD$RM   000034

ISN #0009
000170  ISN$
000172  MOI$M$   000064
000176  CDI$
000200  ADD$M$   000044
000204  MOF$I$   #040640
000210  CDF$
000212  ADD$$$
000214  MOD$$M   000044

ISN #0010
000220  ISN$
000222  REL$     000016
000226  REL$     000010
000232  IFW$
000234  REL$     000034
000240  TVC$
000242  MOF$I$   #041040
000246  CDF$
000250  MUD$M$   000044
000254  MOD$$M   000054
000260  REL$     000054
000264  TVD$
000266  EOL$

ISN #0012
000270  LSN$     #000014
000274  STP$

ISN #0013
000276  ISN$
000300  RET$
```

```
FORTRAN IV      M01-01  SOURCE LISTING

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. + 3.14159682516
0008            IMAG   =  CMPLX(REAL, 3.21)
0009            DBLE   =  DBLE + INT/2 + 5.
0010            PRINT 10,IMAG,DBLE + 10.
0011      10    FORMAT(1X,3F12.6)
0012            STOP
0013            END
```

```
FORTRAN IV      STORAGE MAP

INT     000006  INTEGER*2 VARIABLE
REAL    000030  REAL*4    VARIABLE
IMAG    000034  COMPLEX*8 VARIABLE
DBLE    000044  REAL*8    VARIABLE
CMPLX   000000  COMPLEX*8 PROCEDURE
```

Figure 1-2
Sample Listings

1.2.3 Compiler Memory Requirements

The FORTRAN Compiler runs in a minimum partition of 7K words. If run in a larger partition it uses the extra space for program and symbol storage. If available storage is exceeded during the compilation process, a FATAL ERROR T is issued.

1.3 USING THE TASK BUILDER TO LINK FORTRAN PROGRAMS

The Task Builder links relocatable object modules together to create a task image. The object modules may come from user specified input files, user libraries, or system libraries. References to global symbols defined in one module and referenced in other modules are resolved. The system object module library (SY:[1,1]SYSLIB.OLB) is automatically searched to resolve any remaining undefined symbols.

References to resident common blocks and shared libraries are also resolved. Thus, the task image produced is ready to be run under the RSX-11M executive. The Task Builder also allows the building of tasks with overlay structures.

For a more complete description of the Task Builder refer to the Task Builder reference manual (DEC-11-OMTBA-A-D).

The Task Builder is a system program invoked by the TKB MCR command. The user initiates it by typing

```
>TKB
```

The Task Builder then prints "TKB>" to indicate that it is ready to accept a command string. Alternatively the command string may be typed on the same line as the TKB MCR command. The command string should be a standard string as described in Section 1.1.1.

The first output file specifies the task image file. A second file may be specified if a memory allocation map is desired. A third file may be specified to contain the program sections and global symbol definitions in relocatable object module format. This file is described in the RSX-11M Task Builder Reference Manual. The input files contain the object modules to be linked. Additional lines of input file specifications may also be entered. After all input files have been typed, the user should type a line consisting only of "//" (unnecessary if command string is specified as part of MCR command). The Task Builder will then build the task image. The default file types used by the Task Builder are shown in Table 1-3.

Standard indirect command file specifications are also acceptable to the Task Builder.

Table 1-3
Task Builder Default File Types

File	Default Type
Task Image	TSK
Map	MAP
Input	OBJ
Library	OLB
Overlay Description	ODL
Command	CMD

The following simple example builds a task image for the object file OBJECT.OBJ created by the first example in Section 1.2.

```
MCR>TKB                          or   >TKB TASK,LP:=OBJECT
TKB>TASK,LP:=OBJECT
TKB>//
```

This creates the task image file TASK.TSK on the system device and lists the memory allocation map on the line printer. Any references in OBJECT.OBJ to FORTRAN OTS routines are resolved automatically because the OTS resides in the system object module library.

1.3.1 Task Builder Switch Options for FORTRAN Programs

A FORTRAN programmer should be aware of several Task Builder switch options. Some of these are necessary for correct linking of FORTRAN programs. Others select options which some users may find useful. Detailed descriptions of these switches and other Task Builder switches may be found in the Task Builder Reference Manual.

The /EA switch must be used on the task image file if the task uses the KE11 Extended Arithmetic Element.

The /FP switch must be used on the task image file if the task uses the PDP-11/45 Floating Point Processor.

An abbreviated form of memory allocation map may be selected by specifying /SH on the map file. See the Task Builder Reference Manual for a description of both the long and short map formats.

The /LB and /MP switches when specified with an input file indicate an object module library and overlay description file, respectively. These two switches will be discussed in more detail in the following sections.

The /SP switch may be used to cause automatic spooling of the Task Builder map file.

1.3.2 Task Builder Options for FORTRAN Programs

The Task Builder allows numerous keyword options to be specified in addition to the switches described above. Several of these are of particular interest to the FORTRAN user.

If keyword options are to be specified,the user must not specify the command string with the MCR command and must terminate his command input with a line consisting of a single slash, "/", instead of a double slash as described previously. This causes the Task Builder to solicit option information by printing:

```
ENTER OPTIONS:
TKB>
```

At this point Task Builder options may be entered, one option per line. After each option is entered the Task Builder prompts with "TKB>" indicating that it is ready to accept the next option. After specifying the desired options a single slash, "/", should be entered to indicate no more options. The Task Builder then proceeds to build the task and produce the requested output files. When it has finished it again types "TKB>" and is ready to accept a new command string. To exit from the Task Builder type ↑Z (Control Z).

The Task Builder options particularly relevant to FORTRAN programmers are described below.

1.3.2.1 ACTFIL - The number of files that may be simultaneously open by a task is defaulted to 4. Buffer space is allocated in the task image for this many files. When multibuffering is used via a call to FDBSET, the number of buffers needed by the FORTRAN task will be greater than the number of files. The number of buffers may be made smaller to conserve memory or larger to allow more files to be open simultaneously by means of the Task Builder option:

```
ACTFIL = n
```

where n is the decimal number of buffers desired. (An attempt to open a file when space is not available may cause a fatal error at run-time.)

1.3.2.2 COMMON - If a shared common block is to be referenced by the FORTRAN program this intention must be declared by specifying the following option:

```
COMMON = name:access
```

where name is the name associated with the resident common block and access is either RO for Read Only (meaningful for mapped systems only) or RW for Read/Write. The FORTRAN common with the same name is used to reference the data in the resident common.

1.3.2.3 LIBR - If a shared library is to be referenced by the FORTRAN program the following option must be used:

```
LIBR= name:access
```

where name is the library's name and access is either RO for Read Only (meaningful for mapped systems only) or RW for Read/Write. Libraries are discussed in more detail in Section 1.3.3.

1.3.2.4 MAXBUF - The maximum record size that can be handled by the FORTRAN object time system for input/output is defaulted to 132 (decimal) bytes. This may be increased by specifying the Task Builder option:

```
MAXBUF = n
```

where n is the number of bytes. The default generally is adequate for formatted input/output. If direct access operations are performed using records larger than 132 bytes, the user must employ this option to specify the size of the largest record which will be handled.

1.3.2.5 TASK - The task image being built may be given an explicit task name by specifying the following option:

```
TASK = name
```

where name may be from one to six alphanumeric characters the first of which must be alphabetic. If no task name is specified at task build time, it may be explicitly specified at install time, or the Task Builder will use the first six characters of the task image file name as the task name.

1.3.2.6 UIC - The default UIC under which the task will execute may be set by specifying the option:

```
UIC = [g,o]
```

where g,o is a valid group and owner number combination. If this option is not specified the default UIC is [200,200].

1.3.2.7 UNITS - The default number of logical units available to the FORTRAN program is 6, that is, logical units 1 through 6 inclusive. This number may be set explicitly smaller or larger at task build time by specifying the option:

```
UNITS = n
```

where n is the number of logical units desired.

The default device and file name associated with a logical unit number is discussed in Section 3.4. If a smaller number of units than the default is desired, a UNITS command must be given before any ASG commands.

1.3.2.8 FMTBUF - The default size (32 words) of the buffer used for object time format compilation may be changed by use of the following Task Builder option:

```
FMTBUF = n
```

where n is the decimal size, in words, of the object time format compilation buffer. The total size needed for format compilation is approximately equal to the number of characters in the largest object time format used by the program.

1.3.2.9 ASG - Logical unit numbers may be assigned to physical device units by use of the following option:

```
ASG = dev1:n1:n2:...,dev2:m1:m2:...,...
```

where each dev is a physical device unit name and each n and m is a valid logical unit number. The default device assignments would appear as:

```
ASG = SY:1:2:3:4,TI:5,CL:6
```

1.3.2.10 PAR - A task may be built to execute in a specific partition by use of the following option:

```
PAR = pname[:start:length]
```

where:

pname = is the name of the partition for which this task is being built. In unmapped systems, the task must run in this partition.

start = the octal starting address of the partition. This address must be on a 32-word boundary if unmapped, 4K boundary if mapped.

length = the octal length of the partition in bytes. The specified length must be divisible by 64.

The optional argument's start and length must be specified when the task is being built on a system that does not have the partition named by the pname parameter.

Normally a task is built to execute from the default partition GEN. When PAR is not specified the default assumed is:

```
PAR=GEN
```

1.3.3 FORTRAN Library Usage

An RSX-11M Library consists of a collection of object modules. Two kinds of libraries exist, shared and relocatable. The Task Builder is used to include modules from relocatable libraries in a task image.

1.3.3.1 Relocatable Libraries - Relocatable libraries are stored in files on a file structured volume such as a disk. Object modules from

relocatable libraries are copied into the task image of each task referencing the module. A relocatable library may be specified as an input file to the Task Builder. Such a file specification must include the /LB switch to indicate that the file is a library file. When a library specification is encountered, those modules in the library which contain definitions of any currently undefined global symbols are included in the task image.

1.3.3.2 Shared Libraries - Shared libraries are located in main memory and a single copy of each library is utilized by all referencing tasks. Access to a shared library is gained by using the LIBR option as described in Section 1.3.2.3. Shared libraries are built by the user with the Task Builder; they must contain shareable (reentrant) code.

1.3.3.3 System Libraries - Each RSX-11M system has a system relocatable library. The system relocatable library, SY:[1,1] SYSLIB.OLB is automatically searched by the Task Builder if any undefined global references are left after processing all user specified input files. If the definition of one of these undefined global symbols is found, the appropriate object module is included in the task. The FORTRAN OTS is included in the system object library and hence is loaded automatically with FORTRAN programs.

1.3.3.4 User Libraries - The user can construct his own relocatable libraries of assembly language and FORTRAN routines by using the RSX-11M Librarian. These libraries are accessed by using the /LB switch as described in preceding sections.

If MATRIXLIB.OLB is a relocatable library containing matrix manipulation routines and PROG.OBJ is the object file of a compiled FORTRAN program which calls the matrix routines, the following command string might be given to the Task Builder:

```
TKB>PROG,LP:=PROG.OBJ,MATRIXLIB.OLB/LB
TKB>//
```

1.3.4 Overlays

The overlay facility supplied by RSX-11M allows large programs to be executed in relatively small partitions of main memory. The overlay system is virtually invisible to the FORTRAN programmer. All he must do is specify the overall overlay structure, indicate which subprograms are to reside in various parts of the structure and specify, using the overlay description language (ODL), which routines are to be autoloadable. Overlay loading may be performed automatically (autoload) or via explicit load requests (manual load). A complete description of manual loading can be found in the Task Builder Reference Manual.

The overlay structure is specified as a tree structure in terms of the Task Builder's Overlay Description Language (ODL). If an overlay structure is to be built then only one input file can be specified to

the Task Builder. This file must contain the appropriate ODL statements. The specification of this file in a command string must be followed by the /MP switch to identify it as an ODL file.

Simple overlay structures may be constructed using only two ODL statements, .ROOT and .END. The following short examples demonstrate how to build overlays. For a more detailed explanation of this facility refer to the Task Builder Reference Manual.

Suppose a FORTRAN program consists of a main program (MAIN.OBJ) which performs input and output and calls three subroutines, one which does pre-processing of the data (PRE.OBJ), one which performs the primary processing function of the program (PROC.OBJ), and one which does post-processing of the data (POST.OBJ). The following ODL statements specify an overlay structure which has a resident portion which consists of the main program and three overlays which share the same memory locations. Each overlay contains a single subroutine. Figure 1-4 is a diagram of the overlay structure. The ODL statements to create this structure, are as follows;

```
.ROOT       MAIN-*(PRE,PROC,POST)
.END
```

The .ROOT statement is used to declare the tree structure. The .END statement indicates the end of the ODL statements. The names specify object file names (default file type is OBJ). Commas separate descriptions of overlay segments which occupy the same storage. Parentheses are used to group these descriptions. Dashes separate descriptions of modules which are concatenated. If automatic loading of overlays is desired, an asterisk must precede the name of each file which contains a subprogram invoked from a point in the overlay structure which is closer to the root of the structure than the subprogram. If all files within a pair of parentheses are to be automatically loaded, a single asterisk placed in front of the opening parenthesis may be used instead. This indicates that the segment containing the module will be automatically loaded whenever a call is made to a subroutine in the file.

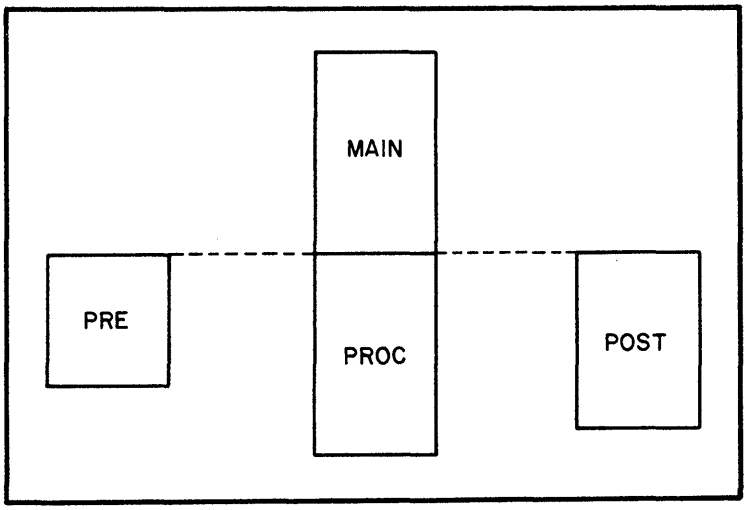


Figure 1-3
Simple Overlay Structure

A path of an overlay structure is any route from the root of the structure which follows a series of branches to an outermost segment of the tree. Figure 1-4 has only three short paths, MAIN-PRE, MAIN-PROC, and MAIN-POST. A program in one overlay segment may call a subprogram in another segment if and only if the two segments occur on a common path. Thus, MAIN may call PRE, PROC or POST, but the three subroutines cannot call each other.

A more complex structure is given in Figure 1-5 and is specified by the ODL statements

```
.ROOT       A-B-*(C,D-(E,F,G))

.END
```

The paths in this structure are A-B-C, A-B-D-E, A-B-D-F, and A-B-D-G. A possible sequence of calls is the following:

A calls G, G calls B, B calls D

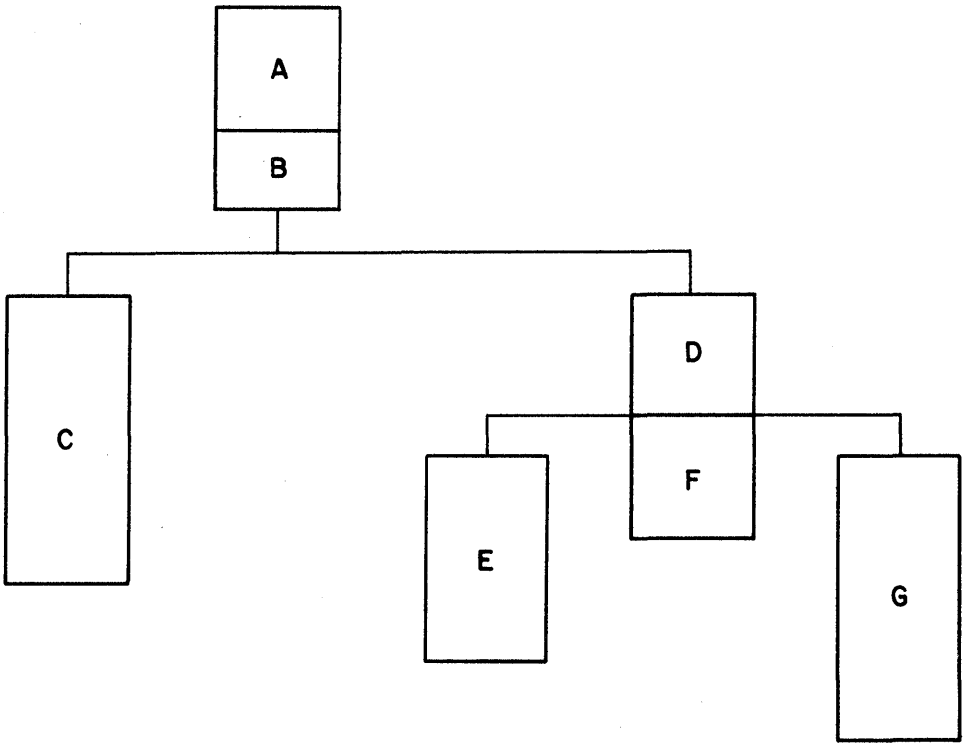


Figure 1-4
Overlay Structure

1.4 USING MCR TO INITIATE TASK EXECUTION

Once a task image has been built, MCR may be utilized to start the execution of that task. First, the task must be installed in the system by typing

```
>INS  Filespec
```

where Filespec is a specification of the file containing the task image. The default device is the system disk and the default file type is TSK. The task is installed in the partition to which it is bound. In an unmapped system it may be installed in this partition only. In a mapped system the /PAR = parname keyword may be used to install the task in any partition of sufficient size. An error will result if the partition is larger than the checkpoint area in which the task was built. The default task priority is 50. The task name may be specified by using the /TASK=name switch.

The execution of an installed task is initiated by typing the RUN MCR command:

```
>RUN tsk
```

where tsk is the task name which was specified when the task was installed or, if none was given then, the name specified when the task was built, or if none was given then, the first six characters of the task image filename.

A task may be terminated prior to its normal termination by typing the ABORT MCR command:

```
>ABO  tsk
```

Execution of a task may be suspended by a FORTRAN pause statement, or ended by a STOP statement. When this occurs, the Object Time System will type a line with the task name, the statement which caused the execution halt, and the contents of the display (text following STOP or PAUSE). To continue execution after a pause, type in the RESUME MCR command, which takes the form:

```
>RES tsk
```

A task which terminates as a result of a CALL EXIT statement will not produce any output indicating it is terminating.

1.5 EXAMPLES

The following sequence might be used to compile, link and execute a FORTRAN task consisting of:

a. the FORTRAN main program MAIN.FTN,

b. the FORTRAN subroutine SUBR1.FTN,

c. several FORTRAN subroutines in the file UTILITY.FTN,

d. some subroutines in the object module library MATLIB.OLB, and

e. the resident common block named PARM.

```
>FOR
FOR>MAIN,MAIN=MAIN
FOR>SUBR1,SUBR1=SUBR1
FOR>UTILITY,UTILITY=UTILITY
```

```
FOR>↑Z
>TKB
TKB>TSKIMAGE=MAIN,SUBR1,UTILITY,MATLIB.OLB/LB
TKB>/
ENTER OPTIONS:
TKB>COMMON=PARM:RO
TKB>TASK=FSYS
TKB>//
>INS TSKIMAGE
>RUN FSYS
```

The listing files produced by the Compiler, MAIN.LST, SUBR1.LST and UTILITY.LST, are automatically printed by the line printer spooler task and are deleted after printing. All files reside or are created on the system disk.

The preceding procedure could be accomplished through the use of indirect command files. Suppose the file COMPILE.CMD contains the following:

```
MAIN,MAIN=MAIN
SUBR1,SUBR1=SUBR1
UTILITY,UTILITY=UTILITY
```

and the file TASKBLD.CMD contains the following:

```
TSKIMAGE=MAIN,SUBR1,UTILITY,MATLIB.OLB/LB
/
COMMON=PARM:RO
```

The following is then equivalent to the previous example.

```
>FOR @COMPILE
>TKB
TKB>@TASKBLD
TKB>TASK=FSYS
TKB>//
>INS TSKIMAGE
>RUN FSYS
```

1.6 DEBUGGING A FORTRAN PROGRAM

The RSX-11M debugging program, ODT, usually cannot be effectively used with a FORTRAN program due to the nature of the object code generated by the FORTRAN Compiler (see section 2.2).

However, in addition to the FORTRAN OTS error diagnostics which include the Traceback Feature (see section 2.5), there is another debugging tool available to the FORTRAN programmer. Placing a D in column one of a FORTRAN statement allows that statement to be conditionally compiled. These statements are considered comment lines by the FORTRAN Compiler unless the /DE switch is used in the Compiler command string. In this case the lines are compiled as regular FORTRAN statements. Liberal use of PAUSE statements and selective variable printout can provide the user with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program development stages and later treated as comment lines.

CHAPTER 2

FORTRAN-IV OPERATING ENVIRONMENT

2.1 FORTRAN-IV OBJECT TIME SYSTEM

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

1. Math routines, including the FORTRAN library functions and other arithmetic routines (e.g., floating point routines),
2. Miscellaneous utility routines (ASSIGN, DATE, ERRSET, etc.),
3. Routines which handle various types of FORTRAN I/O,
4. Error handling routines which process arithmetic errors, I/O errors, and system errors,
5. Miscellaneous routines required by the compiled code,
6. Process I/O routines (AFC, UDC, etc),
7. Laboratory Peripheral Routines (LPS), and
8. RSX-11M executive directives.

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

2.2 OBJECT CODE

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

These common sequences of PDP-11 instructions are contained in a library known as the Object Time System. The FORTRAN Compiler selects

a certain combination of these instruction sequences to implement a FORTRAN program. During program execution, these sequences are threaded together and effect the desired result.

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

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

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

For example, the following FORTRAN program:

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

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

```
FORTRAN          STORAGE MAP

NAME       OFFSET     ATTRIBUTES

RARRAY     000006     REAL*4          ARRAY (10,10)
I          000626     INTEGER*2       VARIABLE
```

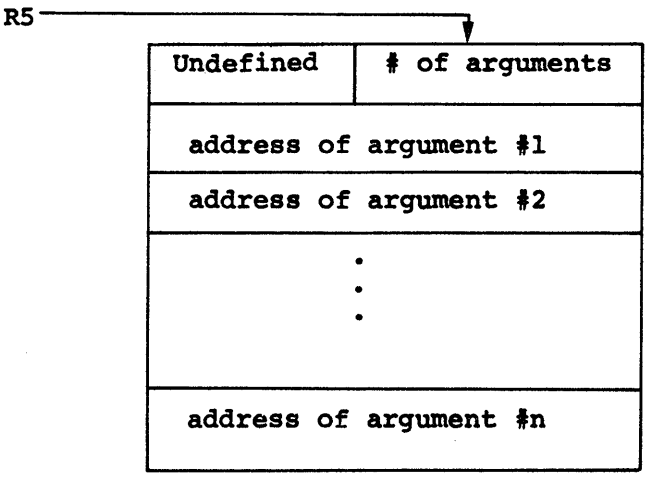
```
J          000630     INTEGER*2     VARIABLE
N          000632     INTEGER*2     VARIABLE
A          000634     REAL*4        VARIABLE

FORTRAN                                  GENERATED CODE

ISN #0002

000640  ICI$M  000626                    ; INCREMENT THE INTEGER WHOSE ADDRESS
                                         ; IS 000626 (I)

ISN #0003

000644  MOI$MS  000626                    ; MOVE VALUE OF INTEGER I ONTO STACK
000650  ADI$IS  #000144                   ; ADD 100 TO VALUE ON TOP OF STACK
000654  MOI$MS  000632                    ; MOVE VALUE OF INTEGER N ONTO STACK
000660  MUI$MS  000632                    ; AND SQUARE IT (MULTIPLY BY ITSELF)
000664  MUI$SS                            ; MULTIPLY (I+100) AND (N**2)
000666  MOI$SM  000630                    ; STORE VALUE ON TOP OF STACK INTO J

ISN #0004

000672  MOF$IM  #040400 000634           ; MOVE AN IMMEDIATE FLOATING CONSTANT
                                         ; (2.0) TO A

ISN #0005

000700  MOF$MM  000006  000012           ; MOVE RARRAY (1,1) TO RARRAY (2,1)
000706  ADF$MM  000634  000012           ; AND ADD A TO RARRAY (2,1)

ISN #0006

000714  RET$                             ; RETURN TO RSX-11M
                                         ; (EXIT FROM PROGRAM)
```

2.3 SUBROUTINE LINKAGE

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

```
JSR     PC,routine
```

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



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

Control is returned to the calling program via the instruction:

```
RTS     PC
```

An assembly language subroutine to find the sum of any number of integers using the following call:

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

might look like the following:

```
          .TITLE    IADDER
IADD::    MOV       (R5)+,R0        ;GET # OF ARGUMENTS
          CLR       R1              ;PREPARE WORKING REG.
          DECB      R0              ;CALCULATE # 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
          .END
```

2.4 SUBPROGRAM REGISTER USAGE

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

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

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

Integer and Logical functions - result in R0

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

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

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

2.5 VECTORED ARRAYS

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

Multidimensional arrays, which are actually stored sequentially in memory, require certain address calculations to determine the location of individual elements of the array. Typically, a mapping function is used to perform this calculation. For example to locate the element LIST(1,2,3) in an array dimensioned LIST(4,5,6) a function equivalent to the following may be used. This function identifies a location as an offset from the origin of the array storage.

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

where si = subscript i
di = dimension i

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

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

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

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

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

The Compiler /-VA switch can be used to suppress all array vectoring.

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

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

1. Arrays are in the same program unit
2. For the i^{th} dimension vector to be shared by the arrays, dimensions to the left of the ith dimension must be equivalent in each array.

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

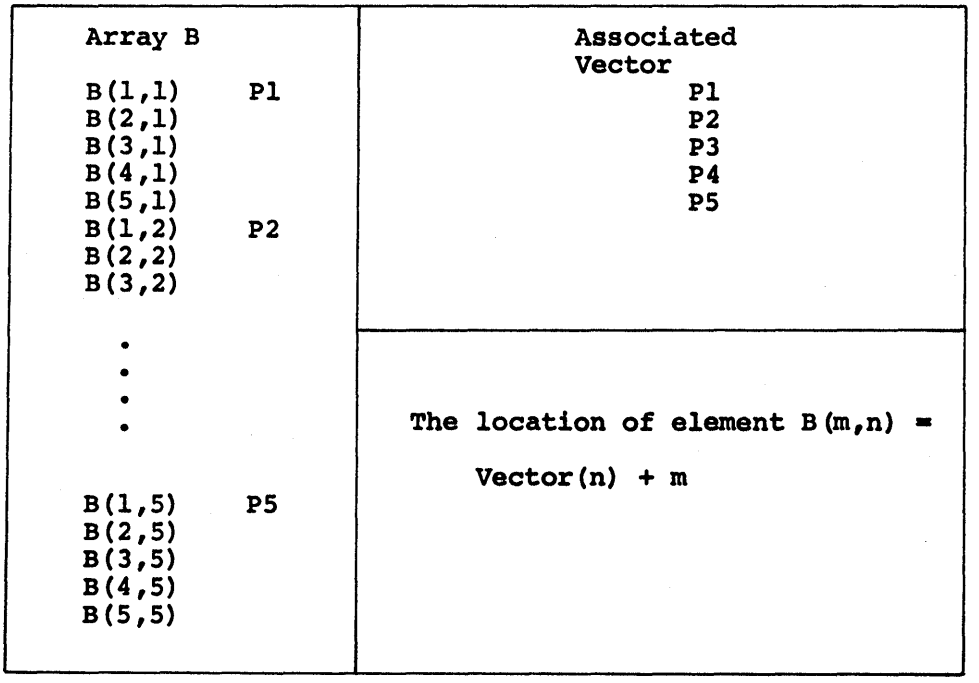


Figure 2-2
Array Vectoring

CHAPTER 3

RSX-11M FORTRAN-IV SPECIFIC CHARACTERISTICS

This chapter deals with information specific to RSX-11M FORTRAN-IV that is omitted from or relaxes restrictions included in the PDP-11 FORTRAN Language Reference Manual.

It should be noted that deviations from FORTRAN syntax requirements outlined in the PDP-11 FORTRAN Language Reference Manual, even if acceptable in RSX-11M FORTRAN, decrease the portability of the program, and may prohibit successful execution on another PDP-11 system.

3.1 VARIABLE NAMES

RSX-11M FORTRAN allows variable names to extend past six characters in length. However, only the first six characters are significant and should be unique among all variable names in the program unit. A warning diagnostic is given for each variable name which exceeds six characters in length. The diagnostic will be suppressed if (/-WR) is included in the compiler command string.

3.2 INITIALIZATION OF COMMON VARIABLES

RSX-11M FORTRAN allows any variables in COMMON, including blank COMMON, to be initialized in any program unit by use of the DATA statement.

3.3 CONTINUATION LINES

RSX-11M FORTRAN does not place any limits on the number of continuation lines that a statement may contain.

3.4 DEFAULT LOGICAL UNIT - DEVICE/FILE ASSIGNMENTS

Listed in table 3-1 are the default logical unit - device and filename assignments. The default device assignments may be changed at task build time via the ASG keyword option or prior to execution via the REASSIGN MCR command. For example the command:

```
>REA tsk 3 LP:
```

connects logical unit 3 to the physical device unit line printer in the task named tsk. The device and/or filename assignments may be changed at execution time by use of the ASSIGN library routine (see section B.2). The default filename conventions hold for logical units not listed below, i.e., unit number 12 will have a default filename of FOR012.DAT.

Table 3-1
FORTRAN Logical Device Assignments

Logical Unit Number	Default Device	Default Filename
1	System disk, SY:	FOR001.DAT
2	System disk, SY:	FOR002.DAT
3	System disk, SY:	FOR003.DAT
4	System disk, SY:	FOR004.DAT
5	Requestor terminal,TI:	FOR005.DAT
6	Console listing, CL:	FOR006.DAT

Although any combination of valid logical unit numbers may be used, there is an imposed maximum number of units which may be simultaneously active. By default, four file structured logical units may be concurrently active, i.e., four files may be open simultaneously. The number may be changed by use of the ACTFIL Task Builder option (see section 1.3.2.1). It should be noted that logical unit numbers are allocated consecutively. For example, if logical units 2 and 17 are used, units 1 through 17 will be allocated.

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. Assigning logical unit number 1 to the terminal, for example, in both cases causes input to come from the terminal.

The ACCEPT, TYPE, and PRINT statements also have similar functional analogies. Assigning devices to logical units 5, 5, and 6 affects respectively the ACCEPT, TYPE, and PRINT statements.

3.5 STATEMENT ORDERING RESTRICTIONS

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

1. In a Subprogram, the first non-comment line must be a FUNCTION, SUBROUTINE or BLOCK DATA statement.

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

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

However, if the statement ordering requirements as outlined in the PDP-11 FORTRAN Language Reference Manual are not followed, a warning diagnostic will be included with the source listing. A /-WR specified in the Compiler command will surpress the diagnostic.

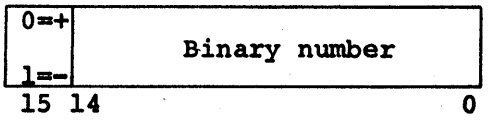
3.6 FCS/OTS FILE OPEN CONVENTIONS

A file or device is opened for I/O activity (if no file or device is already open on that logical unit) by the execution of a READ or a WRITE statement. The type of File Control Services (FCS) open operation invoked depends upon the type of file being opened. The file may be explicitly specified as "OLD" or "NEW" in a call to FDBSET; these specifications are implicit in the use of the READ and WRITE statements. When no explicit specification is made, and a READ is the first I/O operation performed on a unit, "OLD" is assumed. If a WRITE is the first I/O operations, then "NEW" is assumed.

APPENDIX A

FORTRAN DATA REPRESENTATION

A.1 INTEGER FORMAT



Integers are stored in a two's complement representation. If the /I4 compiler switch (see section 1.2.1) is used, an integer is assigned two words, although only the high-order word (i.e., the word having the lower address) is significant. By default, integers will be assigned to a single storage word. Explicit length integer specifications (INTEGER*2 and INTEGER*4) will always take precedence over the setting of the /I4 switch. Integer constants must lie in the range -32768 to +32767. For example:

$+22 = 00026_8$
$-7 = 17771_8$

A.2 FLOATING-POINT FORMATS

The exponent for both 2-word and 4-word floating-point formats is stored in excess 128 (200_8) notation. Binary exponents from -128 to +127 are represented by the binary equivalents of 0 through 255 (0 through 377_8). 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 it is redundant (this is called "hidden bit normalization"). This bit is assumed to be a 1 unless the exponent is 0 (corresponding to 2^{-128}) in which case it is assumed to be 0. The value 0 is represented by two or four words of zeros. For example, +1.0 would be represented by:

```
40200
0
```

in the 2-word format, or:

```
40200
0
0
0
```

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

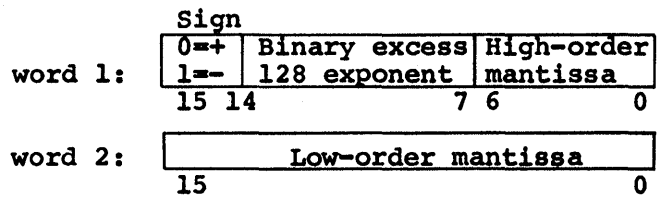
```
140640
0
```

in the 2-word format, or:

```
140640
0
0
0
```

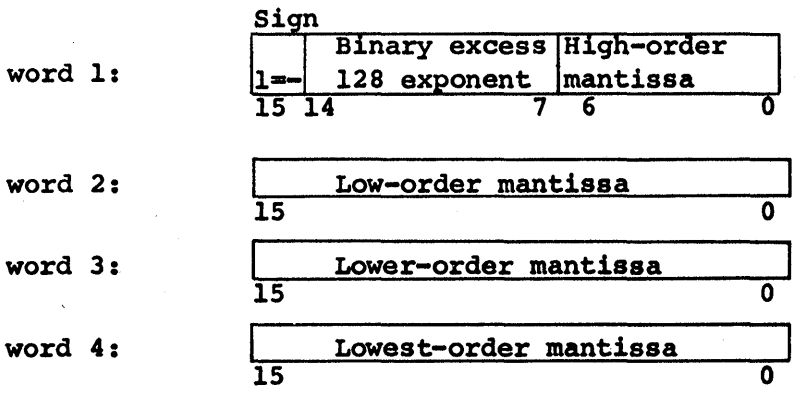
in the 4-word format.

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



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

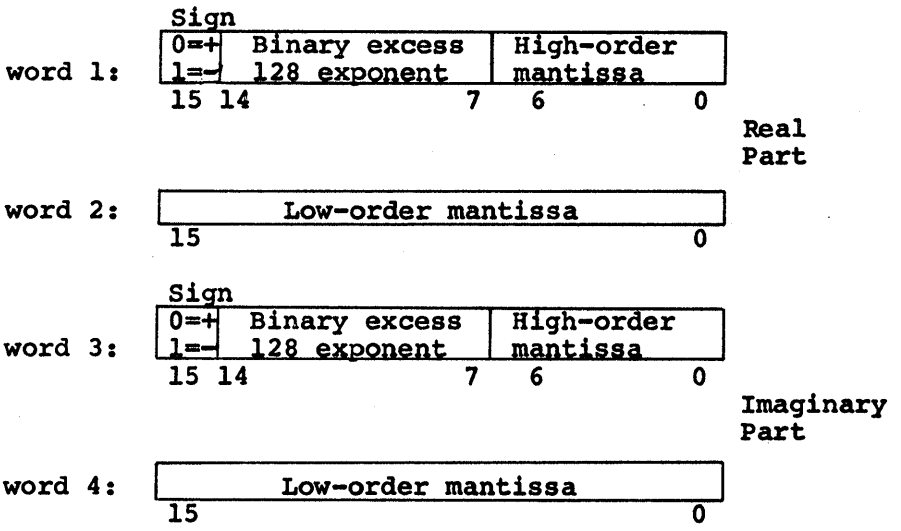
A.2.2 Double Precision Format (4-Word Floating Point)



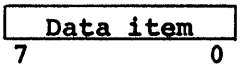
The effective precision is 56 bits or approximately 17 decimal digits of accuracy. The magnitude range lies between .29 X 10^{-38} and .17 X 10^{39}.

APPENDIX A. DATA REPRESENTATION

A.2.3 Complex Format

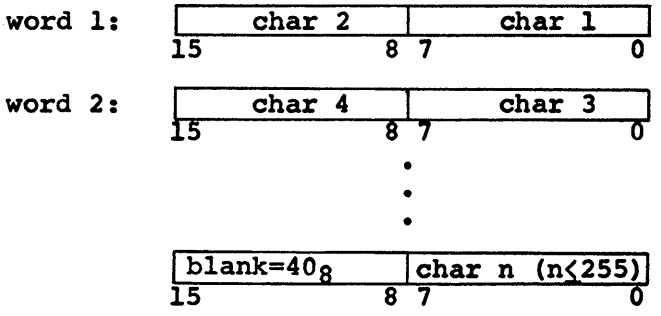


A.3 LOGICAL*1



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.

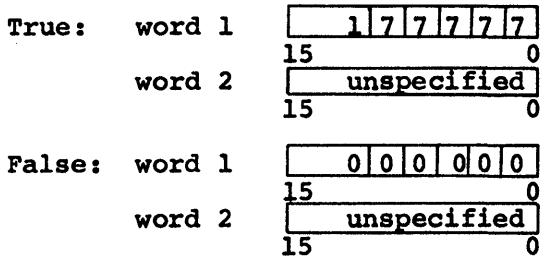
A.4 HOLLERITH FORMAT



APPENDIX A. DATA REPRESENTATION

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

A.5 LOGICAL FORMAT



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

A.6 RADIX-50 FORMAT

Radix-50 character set

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

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

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

APPENDIX A. DATA REPRESENTATION

Single Char. or First Char.	Second Character	Third Character
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
132500	002210	000035
0 135600	0 002260	0 000036
1 140700	1 002330	1 000037
2 144000	2 002400	2 000040
3 147100	3 002450	3 000041
4 152200	4 002520	4 000042
5 155300	5 002570	5 000043
6 160400	6 002640	6 000044
7 163500	7 002710	7 000045
8 166600	8 002760	8 000046
9 171700	9 003030	9 000047

APPENDIX B

LIBRARY SUBROUTINES

B.1 LIBRARY SUBROUTINE SUMMARY

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

ASSIGN	Allows specification at run-time of filename or device and filename to be associated with a FORTRAN logical unit number.
CLOSE	Allows the file on a specified logical unit to be closed.
DATE	Returns a 9-byte string containing the ASCII representation of the current date.
IDATE	Returns three integer values representing the current month, day and year.
ERRSET	Allows the user to specify the action to be taken on detection of certain errors.
ERRSNS	Allows the user to obtain information about the most recently detected error condition.
ERRTST	Allows monitoring of certain error types during program execution.
EXIT	Terminates the execution of a program and returns control to the RSX-11M executive.
USEREX	Allows specification of a routine to be invoked as part of program termination.
FDBSET	Allows specification of special I/O options to be associated with a logical unit.
RAD50	Performs conversion of up to six character Hollerith strings and returns the result as a function value.
IRAD50	Performs conversion of Hollerith strings to Radix-50 representation.
R50ASC	Converts Radix-50 strings to Hollerith strings.

RANDU, RAN	Returns a random real number with a uniform distribution between 0 and 1.
SECNDS	Provides system time of day or elapsed time as a floating point value in seconds.
TIME	Returns an 8-byte string containing the ASCII representation of the current time in hours, minutes and seconds.

B.2 ASSIGN

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

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

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 to the ASSIGN routine follows:

n	logical unit number expressed as an integer constant, variable, or expression.
name	Hollerith or literal string containing any standard RSX-11M device/filename specification. If the device is not specified, then the device remains unchanged from the default assignments, or the MCR REASSIGN command. If a filename is not specified, the default names as described in section 3.4 are used.
icnt	specifies the number of characters in the string 'name'. If 'icnt' is zero, the string 'name' will be processed until the first null character is encountered.

An argument is allowed after icnt to be compatible with previous forms of CALL ASSIGN.

For example, in the following program:

```
CALL ASSIGN(3,'TT:')
WRITE(3,-) ....
CALL CLOSE(3)
WRITE(3,-) ....
```

both WRITE operations will occur on the terminal. To cause the second WRITE to revert back to SY:, another CALL ASSIGN must be used, explicitly setting SY: to unit 3.

B.3 CLOSE

The CLOSE subroutine allows the currently open file on a logical unit to be closed. The form of the call is:

```
CALL CLOSE (n)
```

where n is an integer value specifying the logical unit. When the close is completed the buffers and FDB associated with the file are again available for use.

B.4 DATE

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

```
CALL DATE (array)
```

where array is an array capable of holding a 9-byte string. The array specification in the call may be expressed as the array name alone:

```
CALL DATE (a)
```

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

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

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

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

```
dd-mmm-yy
```

where:

dd is the 2-digit date

-

mmm is the 3-letter month specification

-

yy is the last two digits of the year

For example:

```
15-NOV-75
```

In the case where the array is a real array, 4-1/2 words are used to contain the data string with the remaining array storage being untouched. Therefore, the date string is stored in the first nine bytes in the elements a(i), a(i+1), and a(i+2). The last three bytes of a(i+2) are untouched and should be filled with blanks by the user if he intends to print the date with a 3A4 format.

B.5 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, 1975 the values of the integer variables upon return would be:

```
i = 3
j = 19
k = 75
```

B.6 ERRSET

The ERRSET subroutine allows specification of the action to be taken when an error is detected by the OTS. The error action to be taken is specified individually for each error, independent of other errors. The general form of the call is:

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

where

number — is an integer value specifying the error number to which the following parameters apply.

contin — is a Logical value specifying whether or not to continue after an error. .TRUE. means continue, .FALSE. means exit if this error occurs.

count — is a Logical value specifying whether to count this error against the task maximum error limit. .TRUE. means count, .FALSE. means don't count it.

type — is a logical value specifying the type of continuation to perform. .TRUE. means that an ERR = transfer is to be taken if available. If this action is indicated and an ERR = keyword is not specified in the I/O statement, then the task will exit when the error occurs. .FALSE. means

return to the routine that detected the error for default error recovery.

log — is a logical value specifying whether to produce an error message for this error. .TRUE. means produce message, .FALSE. means don't produce a message.

maxlim — is a positive integer value used to set the task's maximum error limit. The default value is set to 31 at task initialization.

Consult section C.2, OTS Error Processing, for a complete description of the allowed values and meanings of these arguments.

Null arguments are permitted for all but the first argument and cause no change in the current state of that control code.

B.7 ERRSNS

The ERRSNS subroutine allows the user to obtain information about the most recent error that has occurred during program execution. The general form of the call is:

```
CALL ERRSNS (num, fcserr, fcserl, iunit)
```

where

num — is an INTEGER*2 variable or array element into which will be stored the most recent error number.

(A zero will be returned if no error has occurred.)

If the last error occurred as a result of an error indication from File Control Services, then the next three parameters will receive selected values from the file descriptor block (FDB). Otherwise, values of zero will be returned. Consult the I/O Operations Reference Manual (DEC-11-OXFSA-A-D) for definitions of the interpretation of the FCS error return codes.

fcserr — is an INTEGER*2 variable or array element into which will be stored the F.ERR field of the FDB.

fcserl — is an INTEGER*2 variable or array element into which will be stored the F.ERR+1 field of the FDB

iunit — is an INTEGER*2 variable or array element into which will be stored the logical unit number.

From zero to four arguments may be specified. Specifying zero arguments serves to clear ERRSNS from previous calls. To determine if an error occurs in a given section of a program, the following technique is suggested;

1. Call ERRSNS immediately prior to the segment in order to clear any previous error data:
2. Execute the section

3. Call ERRSNS again and branch on a non-zero argument to error analysis code.

For example:

```
      CALL ERRSNS
      CALL ASSIGN (1,'NAME.DAT')
      CALL FDBSET (1,'OLD','SHARE')
      CALL ERRSNS (IERR)
      IF (IERR.NE.0) GO TO 100
```

B.8 ERRTST

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

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

where i is the error number and the value of j is returned as:

```
      j=1 if an error number i has occurred
      j=2 if an error number i has not occurred
```

The sequence:

```
      CALL ERRTST (43,J)
      GO TO (10,20),J
20    CONTINUE
```

transfers control to statement 10 if an error 43 has occurred. See section C.2.3. for a discussion of error codes and messages.

The ERRTST routine also resets to 0 the error flag for that error class (but not the error count used by ERRSET). For example:

```
      CALL ERRTST (I,J)
      CALL ERRTST (I,J) .
```

The second call is guaranteed to return J=2. The ERRTST subroutine is independent of the ERRSET subroutine; neither directly influences the other except that ERRSET can cause execution to terminate.

B.9 EXIT

A call to the EXIT subroutine, in the form:

```
      CALL EXIT
```

terminates execution. It causes all files to be closed and the issuing task to be terminated.

B.10 USEREX

USEREX is a subroutine which allows specification of a routine to be invoked as part of program termination. This allows clean up operations in non-FORTRAN routines. The form of the subroutine call is:

```
CALL USEREX (name)
```

Where 'name' is the routine which will be called at exit time and should appear in an EXTERNAL statement somewhere in the program unit. Control is transferred with a JSR PC,name instruction after all procedures required for FORTRAN program termination have been completed. The transfer of control takes place instead of the normal exit to the RSX-11M executive. The routine specified by USEREX may perform the exit procedure itself or return with an RTS PC.

B.11 FDBSET

The FDBSET subroutine permits specification of special input/output options available through File Control Services under RSX-11M.

```
CALL FDBSET (unit, mode, share, numbuf, initsz, extend)
```

where

unit — is an integer value specifying the logical unit to which the subsequent arguments apply

mode — is one of the following literals specifying the type of access to be used (only the first letter is checked for validity):

'READONLY'	For read only access
'NEW'	For creating a new file
'OLD'	For accessing an existing file
'APPEND'	For appending to an existing file (meaningful for sequential files only)
'UNKNOWN'	When it is not known whether a file exists: this option searches for it, opens it if it already exists, or creates the file if it does not.
'MODIFY'	Open an old file for updating. The file can not be extended.
'ISUP'	Open a new file (for output) but inhibit superseding of the previous version. This file will replace the current version.

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 the first I/O operation performed on that unit, 'OLD' is assumed.

share — is the literal 'SHARE' indicating the FCS shared access bit should be set.

numbuf*	is an integer value indicating the number of internal buffers to be used for multibuffered input/output. If this argument is omitted, one internal buffer is used. The ACTFIL option in the Task Builder command sequence is used to extend the buffer area.
initsz	is an integer value indicating the initial allocation, in disk blocks, of a new file.
extend	is an integer value specifying the number of blocks by which to extend a file.

FDBSET may only be called prior to opening the unit specified in the first argument. CALL FDBSET, CALL ASSIGN, and the DEFINEFILE statement may be used together.

The unit number argument is required. All other arguments may be null or missing to indicate no specification for that argument.

B.12 RAD50

The RAD50 function subprogram provides a simplified way to encode RSX-11M task names in Radix-50 notation. The form of the call is

```
RAD50 (name)
```

where

name is the variable name or array element corresponding to an ASCII string.

Note that the RAD50 function may be used as an argument to an RSX-11M system directive subroutine, e.g.,

```
DATA A/'JOE'/

CALL REQUES (RAD50(A),...)
```

THE RAD50 function is equivalent to the following FORTRAN function subprogram:

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

B.13 IRAD50

The IRAD50 subprogram performs conversions between Hollerith (text) strings and Radix-50 representation. Radix-50 representation is

---

*Multi-buffering is not implemented in the current release of RSX-11M. It will be included in a subsequent release.

required by the ISA Process Control and System Directives Subroutines for the specification of task names within the RSX-11M system. (See the PDP-11 FORTRAN Language Manual for details of the Radix-50 representation.)

IRAD50 may be called as a FUNCTION subprogram if the return value is desired, or as a SUBROUTINE subprogram if no return value is desired. The form of the call is

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

or

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

where

icnt	is the (integer) maximum number of characters to convert.
input	is an ASCII (Hollerith) text string to be converted to Radix-50.
output	is the location for storing the results of the conversion.
n	is the number of characters actually converted.

Three characters of text are packed into each word of output. The number of output words modified is computed by the expression (in integer mode)

```
(ICNT+2)/3
```

Thus if a count of four is specified, two words of output will be written even if only a 1-character input string is given as an argument.

Scanning of input characters will terminate on the first non-radix-50 character encountered in the input string.

B.14 R50ASC

The R50ASC subprogram provides decoding of Radix-50 encoded values into ASCII strings. The form of the call is:

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

where

icnt	is the number of output characters to be converted.
input	is the variable or array containing the encoded input. Note that (icnt+2)/3 words will be read for conversion.

output is the variable or array into which icnt characters (bytes) will be placed.

If the undefined Radix-50 code is detected, or the Radix-50 word exceeds the maximum value 174777 (octal) then question mark(s) ("?") will be placed in the output.

B.15 RANDU,RAN

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

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

where i1 and i2 are previously defined integer variables and x is the real variable name in which is returned a random number between 0 and 1. i1 and i2 should be initially set to 0. I1 and i2 are updated to a new generator base during each call. Resetting i1 and i2 to 0 repeats the random number sequence. The values of i1 and i2 have a special form; only 0 or saved values of i1 and i2 should be stored in these variables.

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

```
x = RAN (i1, i2)
```

is the functional form of the random number generator.

B.16 SECNDS

The SECNDS FUNCTION subprogram returns the system time as a single precision floating point value less the value of its single argument. For example:

```
TIM=SECNDS (0.)
```

will return the number of seconds since midnight, that is, the current time of day. It may be called with a non-zero argument for performing elapsed time computations as in

```
C         START OF TIMED SEQUENCE
          T1 = SECNDS(0.)
C
C         CODE TO BE TIMED
C
          DELTA = SECNDS(T1)
```

where DELTA will give the elapsed time.

The value of SECNDS is accurate to not less than the resolution of the system clock: 0.0166... seconds for a 60-cycle clock and 0.02 seconds for a 50-cycle clock. Systems with a programmable clock have resolution not less than the clock tick as set at SYSGEN.

With 24 bits of precision for real values, this representation is accurate to the clock tick for values up to about two days in duration.

B.17 TIME

The TIME subroutine allows the user to access the current system time as an ASCII String. Its form is as follows:

```
CALL TIME (a)
```

The TIME call returns the time as an 8-byte (8-character, including colons) ASCII string of the form:

```
hh:mm:ss
```

where

```
hh is the two-digit hour indication
mm is the two-digit minute indication
ss is the two-digit second indication
```

For example:

```
10:45:23
```

A 24-hour clock is used. This quantity is typically output with a 2A4 format specification.

APPENDIX C

FORTRAN ERROR DIAGNOSTICS

C.1 COMPILER ERROR DIAGNOSTICS

The RSX-11M FORTRAN Compiler, while reading and processing the FORTRAN source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, missing or illegal statement parameters.

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

Some of the most common causes of syntax errors, however, are typing mistakes. A typing mistake can sometimes cause the Compiler to give very misleading error diagnostics. The user should be careful of the following common typing mistakes:

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

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

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

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

If any errors were detected in a compilation, the message:

```
ERRORS DETECTED: n
```

will be printed on the initiating terminal; n is the number of errors, not including warnings, detected by the compiler.

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

APPENDIX C. ERROR DIAGNOSTICS

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

```
FORTRAN IV      M01-01  SOURCE LISTING                    PAGE 001

0001            DOUBLE PRECISION DBLE DBLE2          ! MISSING COMMA
0002            DATA INT/100/DBLE/500,/              ! MISSING COMMA
0003            DBLE2  =  INT/2 + 5,*DBLE
0004            WRITE,(6,10) DBLE,DBLE               ! EXTRA COMMA
0005     10     FORMAT(1X,2F12,6)
0006     10     STOP                              ! LABEL DEFINED PREVIOUSLY
***** M
0007            INTEGER INT                       ! NOT STANDARD ORDERING
       C        END                               ! NO END STATEMENT
***** E
```

```
FORTRAN IV      DIAGNOSTICS

IN LINE 0002 MSG #030   EXTRA CHARACTERS AT END OF STATEMENT
IN LINE 0004 MSG #049   SYNTAX ERROR
[ WARNING ]  MSG #091   VARIABLE "DBLEDB" NAME EXCEEDS 6 CHARACTERS
[ WARNING ]  MSG #093   NON-STANDARD STATEMENT ORDERING
```

APPENDIX C. ERROR DIAGNOSTICS

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

Some of the easily recognizable FORTRAN syntax errors are detected by the initial phase of the Compiler. These errors are reported in the source program listing. They are not reported if a source listing is not requested.

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

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

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

INITIAL PHASE ERROR DIAGNOSTICS

Code Letter	Description
B	Columns 1-5 of continuation line are not blank. Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1.
C	Illegal continuation. Comments cannot be continued and the first line of any program unit cannot be a continuation line.
E	Missing END statement. An END statement is supplied by the Compiler if end-of-file is encountered.
H	Hollerith string or quoted literal string longer than 255 characters or longer than the remainder of the statement.
I	Non-FORTRAN character used. The line contains a character that is not in the FORTRAN character set and is not used in a Hollerith string or comment line.
K	Illegal statement label definition. Illegal (non-numeric) character in statement label.
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.
M	Multiply defined label.
P	Statement contains unbalanced parentheses.
S	Syntax error. Multiple equal signs, etc. Statement not of the general FORTRAN statement form.
U	Statement could not be identified as a legal FORTRAN statement.

C.1.2 Errors Reported by Secondary Phases of the Compiler

Those Compiler error diagnostics not reported by the initial phase of the Compiler will 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 reference the statement to which they apply by using the internal sequence numbers assigned by the Compiler.

The general form of the diagnostic is:

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

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

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

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

SECONDARY PHASE ERROR DIAGNOSTICS

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****
: All arrays must be dimensioned with integer constants except as specified in the Language Reference Manual.

ARRAY **** HAS TOO MANY DIMENSIONS
: An array can have up to seven dimensions.

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

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

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

DEFECTIVE DOTTED KEYWORD
: A dotted relational operator was not recognized. Also, possible misuse of decimal point.

DO TERMINATOR **** PRECEDES DO STATEMENT
: The statement specified as the terminator of a DO loop must come after the DO statement.

EXPECTING LEFT PARENTHESES AFTER ****
: An array name or Function name reference is not followed by a left parenthesis.

EXTRA CHARACTERS AT END OF STATEMENT
All the necessary information for a syntactically correct FORTRAN statement has been found on this line, but more information exists. Possibly due to inadvertent continuation signal on next line, or a missing comma.

FLOATING CONSTANT TOO SMALL
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.

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

ILLEGAL DO TERMINATOR STATEMENT ****
A DO statement terminator must not be a GO TO, arithmetic IF, RETURN, or DO statement or logical IF containing one of these statements.

ILLEGAL STATEMENT ON LOGICAL IF
The statement contained in a logical IF must not be another logical IF or DO statement.

ILLEGAL TYPE FOR OPERATOR
An illegal variable type has been used with an exponentiation or logical operator.

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.

INTEGER OVERFLOW
An integer constant or expression value must not fall outside the range -32767 to +32767.

INVALID COMPLEX CONSTANT
A complex constant has been improperly formed.

INVALID DIMENSIONS FOR ARRAY
An attempt was made while dimensioning an array to explicitly specify zero as one of the dimensions.

INVALID DO TERMINATOR ORDERING AT LABEL ****
Do loops are incorrectly nested.

INVALID EQUIVALENCE
Illegal equivalence, or equivalence that is contradictory to a previous equivalence.

INVALID FORMAT SPECIFIER
A format specifier is not the label of a FORMAT statement or an array name.

INVALID IMPLICIT RANGE SPECIFIER
Illegal implicit range specifier, i.e., non-alphabetic specifier, or specifier range is in reverse alphabetic order.

INVALID LOGICAL UNIT
A logical unit reference must be 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.

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.

INVALID RADIX50 CONSTANT
Illegal character detected in a RADIX50 constant.

INVALID RECORD FORMAT
The third parenthetical argument in a DEFINE FILE statement must be the single character U.

INVALID STATEMENT IN BLOCK DATA
It is illegal to have any executable or FORMAT statements in a BLOCK DATA Subprogram.

INVALID STATEMENT LABEL REFERENCE
Reference has been made to a statement number that is of illegal construction. GO TO 999999 is illegal since the statement number is too long.

INVALID SUBROUTINE OR FUNCTION NAME
A name used in a CALL statement or function reference is not valid. Example: use of an array name in a CALL statement routine name reference.

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

INVALID TYPE SPECIFIER
An unrecognizable data type was used.

INVALID USAGE OF FUNCTION OR SUBROUTINE NAME
A function name cannot appear in a DIMENSION, COMMON, DATA, OR EQUIVALENCE statement.

INVALID VARIABLE NAME
A variable name contains an illegal character.

LABEL ON DECLARATIVE STATEMENT
It is illegal to place a label on a declarative statement.

MISSING ASSIGNMENT OPERATOR
The first operator seen in an arithmetic assignment statement was not an equal sign (=). Example: I+J=K.

MISSING COMMA
The comma delimiter was expected but was not found. See the section of the FORTRAN Reference Manual that describes the general form of the statement in question.

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

MISSING LABEL
Expecting a statement label but one was not found. Example: ASSIGN J TO I. A valid statement label reference should precede 'TO' but does not.

MISSING RIGHT PARENTHESIS
Expecting a right parenthesis but one was not found. Example: READ(5,100,). The first non-blank character after the format reference should be a right parenthesis but is not.

MISSING QUOTATION MARK
In a FIND statement, the logical unit number and record number must be separated by a single quotation mark.

MISSING VARIABLE
Expecting a variable, but one was not found. Example: ASSIGN 100 TO 1. A variable name should follow the 'TO' but one does not.

MISSING VARIABLE OR CONSTANT
Looking for an operand (variable or constant) but found a delimiter (comma, parenthesis, etc.). Example: WRITE(). A unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead.

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.

MULTIPLE DECLARATION FOR VARIABLE ****
A variable cannot appear in more than one data type declaration statement or dimensioning statement. Subsequent declarations are ignored.

NUMBER IN FORMAT STATEMENT NOT IN RANGE
An integer constant in a FORMAT statement is greater than 255 or is zero.

PARENTHESES NESTED TOO DEEPLY
Group repeats in a FORMAT statement have been nested too deeply.

P-SCALE FACTOR NOT IN RANGE -127 TO +127
P-scale factors must fall in the range -127 to +127.

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.

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

STATEMENT MUST BE UNLABELED
A DATA, SUBROUTINE, FUNCTION, BLOCK DATA, arithmetic statement function definition, or declarative statement must not be labeled.

STATEMENT TOO COMPLEX
An arithmetic statement function has more than 10 dummy arguments. Or the statement is too long to compile. Break it up into 2 or more smaller statements.

SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST
A SUBROUTINE, FUNCTION or BLOCK DATA Statement, if present, must be the first statement in a program unit.

SYNTAX ERROR
Check the general form of the statement with the general form outlined in the Language Reference Manual section that describes that type of statement.

TARGET MUST BE ARRAY
The third argument in an ENCODE or DECODE statement must be an array name.

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.

UNLABELED FORMAT STATEMENT
All FORMAT Statements must be labeled.

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.

VARIABLE **** INVALID IN ADJUSTABLE DIMENSION
A variable used as an adjustable dimension must be an integer dummy argument in the subprogram unit.

WRONG NUMBER OF SUBSCRIPTS FOR ARRAY ****
An array reference does not have the same number of subscripts as specified when the array was dimensioned.

C.1.3 Warning Diagnostics

Warning diagnostics report conditions which are not true error conditions, but which may be potentially dangerous at execution time, or which may present compatibility problems with FORTRAN Compilers running on other PDP-11 Operating Systems. The warning diagnostics are normally enabled, but may be suppressed by use of the /-WR

Compiler switch. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics (see section C.1.2) except that the line number reference is replaced with '%WARNING%'. A listing of the warning diagnostics follows:

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****
Adjustable arrays must be a dummy argument in a subprogram, and the adjustable dimensions must be integer dummy arguments in the subprogram. Any variation from this rule will cause a dimension of 1 to be used and this warning message to be issued.

NON-STANDARD STATEMENT ORDERING
Although the RSX-11M FORTRAN IV Compiler has less-restrictive statement ordering requirements than those outlined in chapter 7 of the PDP-11 FORTRAN Language Reference Manual, non-adherence to the stricter requirements may cause error conditions on other FORTRAN Compilers. See section 3.5 of this document.

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

VARIABLE **** NAME EXCEEDS SIX CHARACTERS
A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN Compilers may treat this as an error condition. See section 3.1 of this document.

C.1.4 Fatal Compiler Error Diagnostics

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

FATAL ERROR n

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

Code	Meaning
C	Constant subscript overflow. Too many constant subscripts have been employed in a statement. SOLUTION - simplify the statement
L	More than 80 characters in input record. SOLUTION - simplify statement or use continuation lines.

O Unrecoverable error occurred while the Compiler was writing the object file (.OBJ). Possibly, insufficient output file space.

SOLUTION - rectify hardware problem, or make more space available for output by deleting unnecessary files.

P Optimizer push down overflow - statement too complex, or too many common subexpressions occurred in one basic block of a program.

SOLUTION - simplify complex statements; report the error to your local software support representative.

R Unrecoverable hardware error occurred while the Compiler was reading source file.

SOLUTION - rectify hardware problem.

S Subexpression stack overflow - statement too complex.

SOLUTION - simplify complex statements.

T Memory Overflow

SOLUTION - break up program into subprograms or compile in a larger partition.

W Unrecoverable error occurred while the Compiler was writing listing file. Possibly, listing file space is not large enough.

SOLUTION - rectify hardware problem, or make more space available for listing file by deleting unnecessary files.

Y Code generation stack overflow - statement too complex.

SOLUTION - simplify complex statements.

Z Compiler error

SOLUTION - report this error to your local software support representative. Please include program listing.

C.2 OBJECT TIME SYSTEM ERROR DIAGNOSTICS

C.2.1 Error Processing Algorithm

The Object Time System detects many Input/Output, arithmetic, invalid argument and other kinds of errors and reports them on the user's terminal via logical device TI:. The action taken for each error is determined by an error control table within the OTS. (This table may be modified during program execution by means of the ERRSET subroutine, see section B.8.)

APPENDIX C. ERROR DIAGNOSTICS

Error processing for each error is controlled by a control byte. Significant bits are as follows:

Continuation Bit	If not set, this bit directs the task to exit as a result of this error. If set, the task will continue provided certain other conditions are met.
Count Bit	If set, this error is counted against the task error count limit. If that limit is exceeded, the task will exit.
Continuation Type Bit	Two types of continuation action are possible: 1) Return to routine that reported the error to take appropriate recovery action and proceed, or 2) Take an ERR= transfer in an I/O statement. If an ERR = transfer is specified for the error and none was included in the Input/Output statement, the task will exit.

The above three conditions must all be satisfied for the task to continue.

Log Bit	If the task continues, then the log bit is tested. If the bit is set, an error message is produced before continuing; otherwise, the task continues.

If any of the above conditions is not satisfied, the task will exit and an error message will always be produced. In this case, the additional text "EXITING DUE TO" is included in the error message so that it is clear why a task is abnormally terminating.

Two additional bits are of interest here since they control the acceptability of ERRSET arguments.

Return permitted bit	If set, then the continuation type bit may be set by ERRSET to specify return.
ERR= permitted bit	If set, then the contination type bit may be set by ERRSET to specify that an ERR= transfer is to occur.

These two bits are used by ERRSET to check the validity of ERRSET arguments. At least one of these must be set in order to set the continuation bit. Also the continuation type argument is checked against these bits for acceptability.

All four combinations of these two bits occur in the OTS, although most errors are in one of two groups.

1) I/O errors generally permit ERR= continuation type but not return continuation.
2) Most other errors permit return continuation but not ERR= transfer continuation (even if they occur during I/O statement processing.)

Notable exceptions are the synchronous system trap errors (numbers 3 through 10) and recursive Input/Output error (number 40) which will always result in task termination, and the Input and Output Formatted Conversion Errors (numbers 63 and 64) which allow both types of continuation.

The initial setting of the error control bits is shown together with error messages in section C.2.3.

C.2.2 Object Time System Error Message Format

An OTS error message consists of several lines of information formatted as follows:

```
tsknam -- [EXISTING DUE TO] ERROR number
text
[AT PC = address]
     [FCS:  f.err f.errl filename unit]
    IN   xxxxxx AT yyy
    FROM xxxxxx AT yyy
    ...
    FROM .MAIN. AT yyy
```

(In the above message prototype, fixed parts of the message are shown in capital letters and variable parts in lower case letters.)

The variable parts of the message are:

tsknam	-the name of the task in which the error occurred.
number	-the error number
text	-a one-line description of the error.

If the OTS error resulted from one of the synchronous system traps, then the program counter will be shown in the line "AT PC =". This line is only produced for errors numbered 5 through 12.

If the OTS error resulted from an error reported to it by File Control Services, the line beginning "FCS:" will be included. Consult the I/O Operations Reference Manual for a description of the FCS error codes.

f.err	the value of the F.ERR field of the File Descriptor Block (FDB).
f.errl	the value of the F.ERR+1 field of the FDB.
filename	the name of the file (not including type or version)
unit	the logical unit on which this error occurred.

Next follows a traceback of the subprogram calling nest at the time of the error. Each line represents one level of subprogram call and shows

xxxxxx	the name of the subprogram.

The name of the main program is shown as .MAIN. The name of a subprogram is the same as the name used in the SUBROUTINE or FUNCTION statement. Arithmetic statement functions, OTS system routines and routines written in assembly language will not be shown in the traceback.

yyy — the internal sequence number of the subprogram at which the error, call statement, or function reference occurred.

A question mark, "?", instead of a number indicates that the subprogram was compiled with the /-SN compiler switch (suppress sequence number accounting) in effect and hence the line number is not known for that program unit.

C.2.2.1 Short Message File - In order to save space, a short message file may be included which prints only the error number. This may be used by specifying the module $SHORT at task build time; e.g.,

```
>TKB
TKB>MAIN=[1,1]SYSLIB/LB:$SHORT,MAIN
```

C.2.3 Object Time System Error Codes

C.2.3.1 Initial Control Bit Settings - The following table shows the initial settings of the significant bits in the error control byte as described in section C.2.1.

Table C-1
Error Control Bit Settings

ERROR NUMBER	CONTINUE?	COUNT?	CONTINUE TYPE	LOG?	PERMITTED ERR=?	PERMITTED RETURN?
1	NO	NO	FATAL	YES	NO	NO
2	NO	NO	FATAL	YES	NO	NO
3	NO	NO	FATAL	YES	NO	NO
4	NO	NO	FATAL	YES	NO	NO
5	NO	NO	FATAL	YES	NO	NO
6	NO	NO	FATAL	YES	NO	NO
7	NO	NO	FATAL	YES	NO	NO
8	NO	NO	FATAL	YES	NO	NO
9	NO	NO	FATAL	YES	NO	NO
10	NO	NO	FATAL	YES	NO	NO
20	YES	YES	ERR=	YES	YES	NO
21	YES	YES	ERR=	YES	YES	NO
22	YES	YES	ERR=	YES	YES	NO
23	YES	YES	ERR=	YES	YES	NO
24	YES	YES	ERR=	YES	YES	NO
25	YES	YES	ERR=	YES	YES	NO
26	YES	YES	ERR=	YES	YES	NO
27	YES	YES	ERR=	YES	YES	NO
28	YES	YES	ERR=	YES	YES	NO
29	YES	YES	ERR=	YES	YES	NO
30	YES	YES	ERR=	YES	YES	NO
31	YES	YES	ERR=	YES	YES	NO
32	YES	YES	ERR=	YES	YES	NO
33	YES	NO	RETURN	YES	NO	YES
34	YES	YES	ERR=	YES	YES	NO
37	YES	YES	ERR=	YES	YES	NO
38	YES	YES	ERR=	YES	YES	NO
39	YES	YES	ERR=	YES	YES	NO
40	NO	NO	FATAL	YES	NO	NO
41	YES	YES	ERR=	YES	YES	NO
42	YES	YES	ERR=	YES	YES	NO
43	YES	YES	RETURN	YES	NO	YES
44	YES	YES	ERR=	YES	YES	NO

Table C-1 (Cont.)
Error Control Bit Settings

ERROR NUMBER	CONTINUE?	COUNT?	CONTINUE TYPE	LOG?	PERMITTED ERR=?	PERMITTED RETURN?
60	YES	YES	ERR=	YES	YES	NO
61	YES	YES	ERR=	YES	YES	YES
62	YES	YES	ERR=	YES	YES	NO
63	YES	NO	RETURN	NO	YES	YES
64	YES	YES	ERR=	YES	YES	YES
65	YES	YES	ERR=	YES	YES	NO
66	YES	YES	ERR=	YES	YES	NO
67	YES	YES	ERR=	YES	YES	NO
70	YES	YES	RETURN	YES	NO	YES
71	YES	YES	RETURN	YES	NO	YES
72	YES	YES	RETURN	YES	NO	YES
73	YES	YES	RETURN	YES	NO	YES
74	YES	NO	RETURN	NO	NO	YES
80	YES	YES	RETURN	YES	NO	YES
81	YES	YES	RETURN	YES	NO	YES
82	YES	YES	RETURN	YES	NO	YES
83	YES	YES	RETURN	YES	NO	YES
84	YES	YES	RETURN	YES	NO	YES
85	YES	YES	RETURN	YES	NO	YES
86	YES	YES	RETURN	YES	NO	YES
90	NO	NO	FATAL	YES	NO	NO
91	YES	NO	RETURN	NO	NO	YES
100	NO	NO	FATAL	YES	NO	NO
101	NO	NO	FATAL	YES	NO	NO

C.2.3.2 Error Messages

Group 0 - Severe Errors

These messages result from severe error conditions for which no error recovery is possible. Consult the RSX-11M Executive Reference Manual for details of what error conditions will cause traps to the System Synchronous Trap Table entries cited below.

1 INVALID ERROR CALL

A TRAP instruction has been executed whose low byte is within the range used by the OTS for error reporting (see Section C.2.4) but for which no error condition is defined.

2 TASK INITIALIZATION FAILURE

Task start up has failed for one of the following reasons:

1. The directive to initialize synchronous system trap handling (SVTK$S) has returned an error indication.

2. The executive directive to enable the FPP asynchronous trap (SFPA$S) has returned an error indication.

3. The File Control Services initialization call (FINIT$) has returned an error indication.

3 ODD ADDRESS TRAP (SST 0)

4 SEGMENT FAULT (SST 1)
This is most likely due to a subscript value out of range on an array reference.

5 T-BIT OR BPT TRAP (SST 2)

6 IOT TRAP (SST 3)

7 RESERVED INSTRUCTION (SST 4)
The program has attempted to execute an illegal instruction. This may be caused by task building with the wrong FORTRAN library for the given hardware configuration. Hardware may have been linked.

8 NON-RSX EMT (SST 5)
The program has executed an EMT instruction whose low byte is not in the range used by the RSX-11M executive.

9 TRAP INSTRUCTION TRAP (SST 6)
A trap instruction has been executed whose low byte is outside the range used for OTS error messages (see C.2.4 below).

10 PDP11/40 FIS TRAP (SST 7)
A module using FIS was linked with a non-FIS FORTRAN library.

11 FPP HARDWARE FAULT
The FPP Floating Exception Code (FEC) register contained the value 0 following an FPP interrupt. This is probably a hardware malfunction.

12 FPP ILLEGAL OPCODE TRAP
The FPP has detected an illegal floating point instruction.

13 FPP UNDEFINED VARIABLE TRAP
The FPP loaded an illegal value (-0.0). This trap should not occur since the OTS initialization routine does not enable this trap condition. A negative zero value should never be produced by any FORTRAN operation.

14 FPP MAINTENANCE MODE TRAP
The FPP has interrupted with a Floating Point Exception Code register value of 14 (octal). This is probably a hardware malfunction.

Group 1 - General Input/Output Errors

These messages result from errors related to the file system.

20 REWIND ERROR
An error condition was detected by FCS during the .POINT operation used to position to the beginning of a file.

21 DEFINEFILE ALREADY DONE
A DEFINEFILE statement was attempted on a unit for which one has already been done. The second DEFINEFILE is ignored. To change a DEFINEFILE specification a CLOSE operation may be performed.

22 RECORD TOO LONG
A record has been read which is too large to fit into the buffer specified by the MAXBUF TKB option. Rebuild the task using a larger MAXBUF specification.

23 BACKSPACE ERROR
One of the following errors has occurred:

a. BACKSPACE was attempted on a file opened for appending
b. FCS has detected an error condition during the .POINT operation used to rewind the file
c. FCS has detected an error condition while reading forward to the desired record.

24 END-OF-FILE DURING READ
Either an end-file record produced by the ENDFILE statement or the FCS end-of-file condition has been encountered during a READ statement and no END= transfer specification was provided.

25 INVALID RECORD NUMBER
A direct-access READ, WRITE, or FIND statement has specified a record number outside the range from one to the value specified in a DEFINEFILE statement.

26 DEFINEFILE NOT DONE
A direct access READ, WRITE, or FIND operation was attempted before a DEFINE FILE was performed.

27 MORE THAN ONE RECORD
An attempt was made to read or write more than a single record in an ENCODE or DECODE statement.

28 CLOSE ERROR
An error condition has been detected by FCS during a CLOSE operation when attempting to close a file.

29 NO SUCH FILE
A file with the specified name could not be found during an open operation.

30 OPEN FAILURE
FCS has detected an error condition during an open operation. (This message is used when the error condition is not one of the more common conditions for which specific error messages are provided.)

31 MIXED ACCESS MODES
An attempt was made to use both formatted and unformatted operations, or both sequential and direct access operations, on the same unit.

32 INVALID LOGICAL UNIT NUMBER
A logical unit number was used which is outside the range specified by the TKB UNITS= option.

33 ENDFILE TO DIRECT ACCESS FILE
An end-file record may not be written to a direct access file.

34 UNIT ALREADY OPEN
A DEFINEFILE statement, CALL ASSIGN, or CALL FDBSET was attempted which specified a logical unit already opened for input/output.

37 INCONSISTENT RECORD LENGTH
An existing direct access file has been opened whose record length attribute is not the same as specified in the DEFINEFILE or OPEN statement. The record length is not changed.

38 ERROR DURING WRITE
FCS has detected an error condition while writing.

39 ERROR DURING READ
FCS has detected an error condition while reading.

40 RECURSIVE I/O ATTEMPT
An expression in the I/O list of a READ or WRITE statement has caused initiation of another READ or WRITE operation. This can happen if a FUNCTION that performs I/O is referenced in an expression in a READ or WRITE statement I/O list.

41 NO FCS BUFFER ROOM
There is not enough free core left in the File Control Services buffer area to set up required I/O buffers. Rebuild the task with a larger ACTFIL declaration or reduce the level of multibuffering.

42 DEVICE HANDLER NOT RESIDENT
During open operation, the filename specifiction included a device for which no handler task is resident.

43 FILE NAME SPECIFICATION ERROR
The file name string used in a CALL ASSIGN is syntactically invalid, contains a switch specification, references an undefined device mnemonic, or is otherwise not acceptable to the RSX-11M operating system.

44 RECORDSIZE TOO BIG FOR 'MAXBUF'
A DEFINEFILE statement has specified a record size which exceeds the size available in the record buffer. Rebuild the task using a larger TKB MAXBUF specification.

Group 2 - Element Transmission Errors

These messages result from errors related to transmitting data between a FORTRAN program and an internal record.

60 INFINITE FORMAT LOOP
The format associated with an I/O statement that includes an I/O list has no field descriptors to use in transferring those variables.

61 FORMAT/VARIABLE - TYPE MISMATCH
An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor.

62 SYNTAX ERROR IN FORMAT
A syntax error was encountered while the OTS was scanning format specifications stored in an array.

63 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 *'s.

64 INPUT CONVERSION ERROR
During a formatted input operation an illegal character was detected in an input field or the input value overflowed the range representable in the input variable. The value of the variable is set to zero.

65 FORMAT TOO BIG FOR 'FMTBUF'
The OTS has run out of memory while scanning an array format that was generated at run time. The default internal format buffer length is 64 bytes. This may be increased by using the FMTBUF TKB option (see section 1.3.2.8).

66 RECORD TOO BIG FOR 'MAXBUF'
During an output operation a record was specified that was longer than the maximum record length. The default maximum record length is 132 (decimal)

bytes. This may be changed by use of the MAXBUF Task Builder option (see section 1.3.2.4).

67 RECORD TOO SMALL FOR I/O LIST
A READ statement has attempted to input more data than existed in the record being read. For example, the I/O list might have too many elements.

Group 3 - Arithmetic Errors

These messages result from arithmetic overflow and underflow conditions.

70 INTEGER OVERFLOW
During an arithmetic operation an integer's magnitude has exceeded 32767.

71 INTEGER ZERO DIVIDE
During an integer mode arithmetic operation an attempt was made to divide by zero.

72 FLOATING OVERFLOW
During an arithmetic operation a real value has exceeded the largest representable real number. The result of the operation is set to zero.

73 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.

74 FLOATING UNDERFLOW
During an arithmetic operation a real value has become less than the smallest representable real number, and has been replaced with a value of zero.

75 FPP FLOATING TO INTEGER CONVERSION OVERFLOW
During a type conversion, an FPP overflow trap occurred.

Group 4 - Argument Errors

These messages result from incorrect calls to FORTRAN-IV supplied functions or subprograms.

80 WRONG NUMBER OF ARGUMENTS
An improper number of arguments were used in a call to a FORTRAN library function or system subroutine.

81 INVALID ARGUMENT
One of the FORTRAN Library Functions or System Subroutines has detected an invalid argument value. See Appendix B.

82 UNDEFINED EXPONENTIATION
An exponentiation has been attempted which is mathematically undefined; e.g., 0.**0.

83 LOGARITHM OF NEGATIVE VALUE
An attempt was made to take the logarithm of a negative number. The result returned is zero.

84 SQUARE ROOT OF NEGATIVE VALUE
An attempt was made to evaluate the square root of a negative value. Zero is returned as the result.

85 INVALID ARGUMENT TO LIBRARY FUNCTION
An invalid argument was used in a call to a FORTRAN library function.

86 INVALID ERROR NUMBER
The error number argument to one of the subroutines ERRSET or ERRTST is not a valid error number.

Group 7 - Miscellaneous Errors

90 COMPILER DETECTED ERROR
If an attempt is made to link and run an object file, with errors reported during compilation, generated by the FORTRAN Compiler, this error will result when the illegal source statement is executed.

91 COMPUTED GO TO OUT OF RANGE
The integer variable or expression in a computed GO TO statement was less than 1 or greater than the number of statement label references in the list. Control is passed to the next executable statement (see the PDP-11 FORTRAN Language Reference Manual).

Group 8 - System Directive Subroutine Errors

These messages result from incorrect calls to RSX-11M system directive subroutines.

100 DIRECTIVE: MISSING ARGUMENTS
A call to a system directive subroutine was made in which one or more of the arguments required for directive execution was not given.

101 DIRECTIVE: INVALID EVENT FLAG NUMBER.
A call to a system directive subroutine was made in which the argument used for event flag specification was not in the valid range (1 to 64).

C.2.4 Notes on OTS Error Processing Implementation

OTS error reports are initiated by execution of a TRAP instruction in which the low byte of the instruction contains the internal error code. Internal error codes are 128 (decimal) greater than the externally printed error code.

Users who wish to do so may use the first 128 (0 to 127) trap codes as follows. TRAP instructions transfer control to the OTS error processor by means of a System Synchronous Trap Table located in the OTS impure work area. The first word of this table has the global symbol $SST. Coding similar to the following might be used to intercept control:

```
;
; INITIALIZATION
;
INIT:     MOV     $SST+14,SST6          ;SAVE OTS TRAP ADDR
          MOV     #INTCEP,$SST+14       ;PUT NEW ADDR IN SST
                                         TABLE
          ...


SST6:     .WORD   0
          ...
;
; TRAP HANDLER
;
INTCEP:   CMP     #128.*2,@SP           ;LOW BYTE *2 OF TRAP
                                         INSTRC.
          BHI     1$                    ;BRANCH IF USER CODE
          JMP     @SST6                 ;GOTO OTS
1$:       ...                           ;USER PROCESSING

          TST     (SP)+                 ;DISCARD EXTRA WORD
          RTI                           ;EXIT INTERRUPT
```

Similar techniques could be used to intercept the other synchronous traps.

OTS error messages are output to logical device TI: by means of a LUN automatically provided by the task builder in addition to those specified by the UNITS directive. If for any reason the QIO directive used to write the error lines receives an error condition return, then an immediate exit is taken to the RSX-11M Executive.

APPENDIX D

COMPATIBILITY WITH OTHER PDP-11 FORTRANS

D.1 COMPATIBILITY WITH PDP-11 FORTRAN V08.04 UNDER RSX-11D V4A

FORTRAN IV (FOR) is a new implementation of FORTRAN for the PDP-11. It has been designed to be compatible with the earlier FORTRAN (MOP, FTN) on RSX-11D. However, some differences exist as a result of

1. Correcting deficiencies in FTN FORTRAN.

2. Language specification decisions necessary to promote the goal of an upward compatible line of FORTRANs, including FORTRAN IV under RT-11 and RSX-11M, FORTRAN IV-PLUS under RSX-11D, and FORTRAN-10 on the DECsystem-10.

3. Providing significant extensions to FTN FORTRAN in a manner consistent with the remainder of the FORTRAN language.

This section summarizes the differences that may affect conversion of programs from FTN to FOR.

D.1.1 Language Differences

1. FTN permits transfer of control to FORMAT statements, which execute as CONTINUE statements. FOR does not, and will issue compile-time error messages.

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

3. FTN logical tests treat any non-zero bit pattern as .TRUE. and an all-zero bit pattern as .FALSE.. FOR tests only the lowest-order byte of a variable.

4. The FTN BYTE statement and data type do not exist in FOR. LOGICAL*1 is equivalent to BYTE in most respects; LOGICAL*1 does not force alignment on a word boundary.

5. The syntax of the IMPLICIT statement is not the same in FOR and FTN.

6. FTN provides expressions in FORMAT statements (called variable format expressions); FOR does not.

7. FTN permits Hollerith strings in certain expressions; FOR does not. DATA statements may be used to initialize variable with strings to achieve the same result.

D.1.2 Object Time System Differences

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

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

3. The TRCLIB library for statement level execution tracing is not available in FOR.

4. Some FTN service subroutines are not available, but their function is provided in a different manner:

FTN	FOR
SETERR	ERRSET subroutine
TSTERR	ERRTST subroutine
APPEND	FDBSET subroutine

5. Default format width and number of digits for A, D, E, F, G, I, L, and O formats are supported in FTN. There are no format defaults under FOR.

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

D.1.3 Implementation Differences

1. FTN performs the following actions when opening a unit for direct access I/O: first check if a file already exists; if so, use it. If not, create a new file for use.

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

   The action sequence described above for FTN may be obtained by using 'UNKNOWN' as the second argument in a call to FDBSET.

2. The implementation of error handling in FOR is significantly different from that in FTN. In particular, the use of error

classes for controlling error handling is replaced by control over each individual error condition. FOR provides much more complete error information to the executing program if desired.

3. The FTN ASSIGN service subroutine returns a success/error indication in its fourth argument. The FOR ASSIGN subroutine ignores the fourth argument. Errors may be detected by means of CALL ERRTST or CALL ERRSNS.

4. FTN does not compile format specifications for use at run-time, whereas FOR does. Two differences result:

   a. A special internal buffer is provided in FOR to contain the compiled form of format specifications stored in arrays which must be compiled at run-time. This may require the use of the FMTBUF TKB option to obtain a sufficiently large buffer area for this purpose.

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

   These considerations do not apply to format specifications defined in FORMAT statements.

5. FTN does not enforce the restriction against using a DEFINEFILE statement referring to an existing file to designate a record size different from that specified when the file was created. Most of the time the new record size appeared to work as expected, but it was subject to obscure errors if references were made to records nearly at the end-of-file. This restriction is detected and enforced by FOR.

6. FTN implements REWIND as a close, open sequence. FOR repositions the file at its beginning by means of an FCS .POINT operation.

7. FTN implements the ENDFILE statement as a close operation. FOR implements the ENDFILE statement by writing an end-of-file record on a file.

8. FOR checks that the labels used in an assigned GO TO 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 GO TO statement. FTN checks at run time.

D.1.4 Operational Differences

1. Under FTN a task which has suspended execution as a result of a PAUSE statement resumes execution by means of the CON (continue) MCR command. Under FOR, the RES (resume) MCR command must be used.

2. The default device assignments for logical units 5 and 6 are CL: and TI: respectively on RSX-11D V4A. The default assignments on RSX-11M are TI:5 and CL:6.

D.2 COMPATIBILITY WITH PDP-11 FORTRAN IV-PLUS

1. FORTRAN IV-PLUS (F4P) logical tests check only the highest-order bit of the value, and it treats a one as .TRUE. and a zero as .FALSE.

   FOR treats any non-zero bit pattern in the lowest-order byte of a variable as .TRUE. and an all-zero bit pattern as .FALSE..

2. The FOR implementation of DO loops does not enforce the restrictions, stated in the language manual, concerning changing DO loop parameters, including the control variable, within the range of the loop. FORTRAN IV-PLUS permits DO parameters other than the control variable to be changed. Such changes do not affect the number of times the loop is executed (the iteration count computed at the start for the loop). FORTRAN programs which violate the FOR rules but execute as desired may behave differently under F4P.

3. FOR defines named common blocks in terms of named .CSECTs. F4P uses .PSECTs with an explicit D attribute. Since .CSECTs are equivalent to .PSECTs with I attribute specification assembly language programs which communicate with FORTRAN programs by means of COMMON blocks must be changed in order to be successfully linked by TKB. (TKB treats such conflicting attribute specifications as non-fatal errors.)

4. In F4P, INTEGER*4 causes both 32 bit allocation and 32 bit computation. Only 16 bits are used for computation in FOR.

5. F4P provides the BYTE statement and data type as a synonym for LOGICAL*1; F4P also provides LOGICAL*2; FOR does not.

6. F4P checks the label list in an assigned GO TO at run time; FOR checks only for the validity of the labels.

APPENDIX E

BIT STRING MANIPULATIONS

Realtime Process Control, Process I/O, and most of the RSX-11M system directives may be performed by means of FORTRAN callable subroutines. These routines are described in the RSX-11M Executive Reference Manual, DEC-11-OMERA-A-D.

Two types of bit string manipulations are provided in order to support these calls: logical and shifting. In order to process words from arrays it is necessary to be able to manipulate information on a bit by bit basis.

E.1 LOGICAL OPERATIONS

These operations are implemented as function subprograms. In the following functions, m and n specify integer variables or array elements. Operations are performed on a full word, bit by bit.

These operations are supported for compatibility purposes only. The FORTRAN logical operators .AND., .OR., .XOR., XEQV., and ,NOT. are simpler to use and generate better code.

E.1.1 Inclusive OR

```
IOR (m, n)
```

Where m and n designate arguments which are logically added.

E.1.2 Logical Product

```
IAND (m, n)
```

Where m and n designate arguments which are logically multiplied.

E.1.3 Logical Complement

NOT (m)

Where m designates the argument which is logically complemented.

E.1.4 Exclusive OR

IEOR (m, n)

Where m and n designate arguments which are exclusively added.

E.2 SHIFT OPERATIONS

The logical shift is implemented as a function subprogram. A right or left shift can be specified. Zeros are propagated following the shifted value and the argument's sign is not extended.

ISHFT (m, n)

m designates the argument to be shifted

n n specifies the number of positions to be shifted and the direction of the shift:

n>0 shift right

n<0 shift left

n=0 no shift

The absolute value of n, n , should not exceed fifteen. If it does, the function result will be zero.

APPENDIX F

SOFTWARE PERFORMANCE REPORTS

From time to time problems and/or errors will be encountered in the use of the FORTRAN-IV Compiler or Object Time System. These should be communicated to Digital Equipment Corporation by means of a Software Performance Report (SPR). Software Performance Report Forms, such as the one shown at the end of this section, may be obtained from the local Digital office.

Software Performance Reports should be submitted directly to Software Communications, P. O. Box F, Maynard, Mass., 01754 for handling. Software Support Specialists in each Digital office receive regular reports on known software problems and will in many cases be able to provide software patches for correcting the problem and/or an alternate programming technique for temporarily avoiding the problem.

Reports which appear to be new will be dealt with by the appropriate group within the Software Engineering Department. Every SPR will be acknowledged upon receipt, and every SPR will be answered in writing. SPR's that are of general interest will be published so that other Software Specialists and customers may benefit.

In preparing a Software Performance Report, the following guidelines are recommended:

1. Give as complete a description as possible of the problem encountered. Often a detail that may seem irrelevant will give a clue to solving the problem.

2. If possible, isolate the problem to a small example. Large, unfamiliar programs are difficult to work with and may result in a misunderstanding of what the problem is or an inability to duplicate the problem.

3. Include console samples, listings, link maps, and so on with the SPR. Annotations showing where the error occurred are extremely helpful.

4. If a program reads input data, include sample input listings and, if possible, sample output.

5. If an error example cannot be isolated to a single program unit, include listings, etc., on all program units involved.

APPENDIX F. SOFTWARE PERFORMANCE REPORTS

Experience shows that as many as one-third of all SPR's do not contain sufficient information to duplicate the problem. Complete and concise information will help Digital give accurate and timely service to software problems.

The OTS version and edit number should be specified when communicating with DEC Software Support concerning the OTS. The version and edit numbers can be found in the Task Builder map. They will appear as a symbol and associated value in the <. ABS.> program section of the $OTI module map listing and will have the following form:

```
V00xc    nnnnnn
```

where x = a decimal digit
c = an alphabetic character
n = an octal digit

The Compiler version number appears in the <. ABS.> section of the module $FORT, and takes the form MF4xxx, where x is a decimal digit.

A system may crash if Task Builder switches such as /FP or /EA are used which do not match the hardware configuration.

digital

SPR # _____

Field # _____

Page _____ of _____

SOFTWARE PERFORMANCE REPORT

Name			Software Specialist
Company			Office / Cost Center
Address			Report type ☐ Logic/coding error ☐ Documentation error ☐ Suggestion ☐ Inquiry
Zip			
Phone		Date sent	
Computer	System device	Memory	Other hardware
System program & version		Monitor & version	Document / Page
Attachments	☐ terminal printout	☐ object tape	☐ source tape ☐ listing ☐ example

SAMPLE

FOR SOFTWARE SPECIALIST USE ONLY		FOR SOFTWARE COMMUNICATIONS USE ONLY	
DATE REC'D	TO S.C.	DATE REC'D	TO MAINTAINER
DATE ANS'D	TO CUSTOMER	DATE ANS'D	TO SPECIALIST

DEC 7-(360)-1044B-R373

UARCO FORM NO. 81-307-34

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

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

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England	Milan, Italy
Paris, France	Solna, Sweden
The Hague, Holland	Geneva, Switzerland
Tel Aviv, Israel	Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation	Digital Equipment Corporation
Software Distribution Center	Software Distribution Center
146 Main Street	1400 Terra Bella
Maynard, Massachusetts 01754	Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS	DECUS
Digital Equipment Corporation	Digital Equipment, S.A.
146 Main Street	81 Route de l'Aire
Maynard, Massachusetts 01754	1211 Geneva 26
	Switzerland

RSX-11M FORTRAN-IV

READER'S COMMENTS USER'S GUIDE

DEC-11-LMFUA-A-D

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

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

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

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

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

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

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

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