H. Bergkvist

digital

fortran IV
operating environment

pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15
pdp15

digital equipment corporation

DEC-15-GFZA-D

PDP-15 FORTRAN IV OPERATING ENVIRONMENT

For additional copies of this manual, order DEC-15-GFZA-D from Digital Equipment Corporation, Program Library, Maynard Mass. 01754 Price $6.00

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Edition September 1971

CONTENTS

CONTENTS (Cont)

APPENDICES

ILLUSTRATIONS

TABLES

PREFACE

This manual describes the system software facilities which support the PDP-15 FORTRAN IV compilers together with hardware features which affect the FORTRAN programmer. Included are discussions of monitor features which are of interest to the FORTRAN programmer, the FORTRAN IV Object Time System[1] (OTS), and the Science Library[2]. All descriptions presented are based on the most comprehensive version of the FORTRAN compiler. Appendix E presents overall outlines and descriptions and detailed data specifying the differences between the various compilers for all of the FORTRAN IV versions offered.

A companion manual "PDP-15 FORTRAN IV LANGUAGE MANUAL", order code DEC-15-GFWA-D, describes the elements, syntax and use of the FORTRAN IV language as implemented for the PDP-15 computer.

---

[1]The Object Time System is a set of subroutines which are automatically invoked by certain FORTRAN language elements. A FORTRAN input-output statement, for example is not compiled directly into executable object code but becomes a call to the appropriate OTS input-output routine.

[2]The Science Library is a set of intrinsic functions, external functions, subfunctions, and subroutines which the user may invoke explicitly in a FORTRAN statement.

CHAPTER 1
INTRODUCTION

A FORTRAN-IV program may be compiled and run in several different environments. The FORTRAN programmer need not be concerned with the details of his environment since the FORTRAN Object-Time System (OTS) will ensure that his statements invoke the appropriate computer instructions. For example, an arithmetic statement such as A = A*B will appear the same in any FORTRAN-IV program. In the object program it may be transformed to a subroutine call, an EAE instruction, or a floating point instruction, depending on the hardware configuration on which the program is produced.

He will need to know procedures for compiling and loading his program and for using the peripheral devices available to him. In addition, a number of software facilities may be of interest to a FORTRAN programmer who requires maximum program efficiency or functions not performed by FORTRAN statements. In this case, he may invoke FORTRAN-callable functions and subroutines from the FORTRAN library or augment his program by linking to MACRO assembler programs and invoking the OTS utility routines.*

In this chapter, we describe the basic procedures for using FORTRAN and the major facilities available to a FORTRAN program. These facilities are described in greater detail in subsequent chapters, and Appendix C contains a collection of illustrative programming examples. The main discussion is based on the DOS-15 monitor, and differences for other environments are noted.

1.1 OPERATING PROCEDURES

The FORTRAN-IV compiler is a two-pass system program which produces relocatable object code. This code is then linked with user-specified FORTRAN-compiled or MACRO-15 assembled routines and with required OTS library routines. Program linkage may be accomplished via the linking loader, LOAD, which loads the resulting program directly into core in absolute format. The user may, alternatively, use one of the overlay linkage editors - CHAIN (DOS-15, ADSS, B/F, Basic I/O Monitor) or TKB (RSX). These construct core images onto auxiliary storage.

---

*In all MACRO calling sequences given - when an address is required as an argument, it may be expressed as +400000 to indicate indirection.

The FORTRAN-IV compiler is called by typing F4 after the monitor has issued a $. When FORTRAN has been loaded, the version name is typed at the left margin as in:

F4X Vnn

A carriage return is issued and the character > at the left margin indicates that a command string is expected with the FORTRAN source program on the appropriate input.

The command string has the form:

optionlist ← filename

where the options are delimited by a left arrow and may optionally be separated by commas, and the string is terminated by a carriage return or ALT MODE. A carriage return specifies that FORTRAN-IV should be restarted after the current program has been compiled. ALT MODE returns control to the monitor.

The option list may be blank or contain any of the following options:

Option*	Meaning
O	object listing
S	symbol map
L	source listing
B	binary output
D	output listing on DECtape unit 2
U	write output on DECtape unit 1

Filename must be a legal FORTRAN symbol. The output listing always has the extension LST.

At the end of pass 1, the compiler types

END PASS1

to accomodate the repositioning of a paper-tape source file in the reader. When compiling from paper tape, to initiate pass 2, the user types ↑P (control P). Otherwise, pass 2 is initiated automatically.

*Refer to Appendix E for list of options applicable to each version of FORTRAN

The following error messages indicate that the command procedures cannot be carried out:

Message	Meaning
?	Bad command string - retype
IOPS 4	I/O device not ready - type CTRL R when ready
IOPS	See PDP-15/20 User's Guide for IOPS error codes

Other diagnostics which may be printed at compile time are FORTRAN error messages (see Appendix B, Section B.1). OTS errors are given at run time for those routines whose calls are generated by the compiler (see Appendix B, Section B.2).

When the user program has been successfully compiled, it may be relocated and made absolute (executable) via LOAD, CHAIN, or TKB (the RSX Task Builder).

The Linking Loader is called by typing LOAD or GLOAD (load-and-go) after a monitor-issued $. The Linking Loader types

```
LOADER Vnn
>
```

and awaits a command string specifying programs to be loaded and output options. See the PDP-15/20 User's Guide[1] for detailed instructions. Figure 1-1 shows the printout from a typical DOS-15 session from source-program preparation to loading.

With CHAIN, the user generates a system of overlays - a resident main program which may include resident subprograms, a resident blank COMMON storage area, and a set of subroutines which overlay each other at the user's request. Subroutines are organized into units called LINKS which may overlay each other. Several LINKS may overlay a larger LINK without overlaying each other. A LINK is loaded into core when a subroutine within the LINK is called and it remains resident until overlayed. A LINK's core image is not recorded or "swapped out" when it is overlayed. The same image is brought into core each time a LINK is loaded. See the PDP-15 CHAIN and EXECUTE manual for detailed instructions (DEC-15-YWZA-DN2).

[1] Order code DEC-15-MG2C-D

```
DOS-15 V02
 ENTER DATE (MM/DD/YY) - 6/8/71

$LOGIN DEM

$PIP

DOSPIP VIA

>N DK


>↑C

DOS-15 V02
$EDIT



EDITOR V10A
>OPEN IOTST
FILE IOTST SRC NOT FOUND.
INPUT
C
C  TTY:    .DAT 6
C
            WRITE (6,100)
100         FORMAT (1X,$IN:$)
            READ (6,) R1,R2
            WRITE (6,200)
200         FORMAT (1X, 'OUT:')
            R3=P1**R2
            WRITE (6,) R3
            STOP
            END

EDIT
>CLOSE

 EDITOR V10A
>↑C

DOS-15 V02
$F4

F4X V15A
 >B←IOTST
END PASS1


DOS-15 V02
$A TT 6

$LOAD
```

Figure 1-1   Sample DOS-15 Session

(continued next page)

```
BLOADER  V11A
>P←IOTST
P   IOTST           77535
P   DDIO     007    75463
P   .BE      006    75430
P   .EE      002    75337
P   .EF      004    75221
P   .EC      001    75155
P   BCDIO    028    71230
P   .SS      005    71150
P   STOP     003    71135
P   SPMSG    004    71042
P   .FLTB    004    70554
P   FIOPS    016    67652
P   DBLINT   05B    67246
P   INTEAE   008    67112
P   DOUBLE   004    66707
P   RELEAE   016    65576
P   OTSER    009    65366
P   .CB      003    65346
↑ S↑S
IN:
11.2,3.0


OUT:
'R3'=      1404.9282

STOP    000000


DOS-15  V02
$
```

Figure 1-1 Sample DOS-15 Session (Cont)

TKB is similar to CHAIN. Its function is to record core images in a file in the format expected by the RSX INSTALL MCR Function. The task name is used as the file name, and TSK is used as the extension. TKB uses the same .DAT slots and accepts the same overlay descriptions as CHAIN. It is called by typing "TKB" following the Monitor's $ request. When loaded, TKB types its name and version number and makes the following requests:

```
LIST OPTIONS
NAME TASK
SPECIFY DEFAULT PRIORITY
DESCRIBE PARTITION
DESCRIBE SYSTEM COMMON BLOCKS
DEFINE RESIDENT CODE
DESCRIBE LINKS AND STRUCTURE
```

For further information, see RSX-15 Reference Manual (DEC-15-GRQA-D).

1.2 SOFTWARE ENVIRONMENTS

Each version of FORTRAN-IV has its own version of OTS and the Science Library so that routines may utilize both hardware and software features. Each of the monitor systems under which FORTRAN operates is summarized below.

1.2.1 DOS-15

DOS-15 is a single-user, interactive, disk-resident Operating System. It includes the DOS-15 Monitor, I/O device handlers, and an integrated set of system programs including FORTRAN-IV. Program editing, loading, and debugging facilities are provided as well as powerful file manipulation capabilities. The DOS-15 disk file structure supports both direct and sequential access to disk files, dynamic disk storage allocation, and file protection. The DOS-15 Monitor itself provides the interface between the user and peripheral devices via Monitor calls and allows the user to load system or user programs, for example, FORTRAN programs, via simple commands from the user terminal. The reader is directed to the DOS-15 Software System User's Manual, DEC-15-MRDA-D, for more detailed information.

1.2.2 ADVANCED Monitor Software System (ADSS)

The ADVANCED Monitor Software System is an integrated system of programs which includes the ADVANCED Monitor, an Input-Output Processor (IOPS), and a set of system programs which prepare, compile, assemble, debug, and operate user programs. The monitor itself serves as the interface between FORTRAN and peripheral devices and between the user console and the system. Detailed information on the components of ADSS may be obtained in the ADVANCED Monitor Software System Manual, DEC-15-MR2B-D.

1.2.3 PDP-15/30 Background/Foreground Monitor System

The Background/Foreground Monitor (B/F) is an extension of the ADVANCED Monitor which permits concurrent, time-shared use of the PDP-15/30. This is done through protected, foreground user programs with a background of batch processing, through program development, or through low-priority user programs. Details are available in the PDP-15/30/40 Background/Foreground Monitor Software System manual (DEC-15-MR3A-D).

1.2.4 RSX-15 Real-Time Execution

RSX-15 is a monitor system designed to handle real-time information in a multiprogramming environment. RSX-15 controls and supervises all operations within the system including any number of core- and disk-resident programs (called tasks). The user can dynamically schedule tasks via simple time-directed commands issued from the terminal or from within a task. RSX uses the ADVANCED Software Monitor (1.2.2) and a Real-Time Monitor. System software includes the FORTRAN-IV compiler, the MACRO Assembler, the TASK BUILDER, and numerous utility programs required to edit, compile, debug, and run user programs. Details are available in the RSX-15 Real/Time Executive Reference Manual (DEC-15-GRQA-D).

1.2.5 BOSS-15

BOSS-15 is a batch-processing monitor which is part of DOS-15; it, therefore, utilizes the DOS-15 system program and file structures. DOS-15 itself has a facility to batch commands from cards or paper tape; BOSS-15, however, is a separate entity from DOS-15 batch. BOSS-15's command language is batch-oriented, noniterative, easy to use, and highly flexible.

Some highlights of BOSS-15 are:

- Procedure driven command language
- Job timing for accounting purpose
- Line editor
- Facility for user-defined commands

BOSS-15 provides the user with the ability to use any system program (with exception of some programs that work only in an interactive environment) and the disk-file structure of DOS-15.

1.3 HARDWARE ENVIRONMENT

Systems with a Floating-Point Processor (FPP) have a special version of the FORTRAN-IV compiler and OTS which utilizes hardware instructions rather than software calls. For example, RELEAE, the REAL arithmetic package, is not included in FPP systems since REAL arithmetic expressions may be compiled into computer instructions.

The FPP F4X System consists of the standard DOS-15 FORTRAN-IV compiler and Object-Time System (OTS) interfaced (via conditional assembly, and additional routines) to the hardware PDP-15 FPP (Floating-Point Processor). The interface applies to Single and Double Precision Floating-Point Arithmetic and Extended Integer Arithmetic (double integers). Single integer arithmetic is still handled by software.

Floating-Point (FPP) FORTRAN-IV is available in different forms for use in PDP-15 software systems other than the DOS-15 system. See Appendix E for descriptions of the available types of FORTRAN-IV.

The following points should be noted with respect to the software modifications which accompany the FPP software systems:

(1) The calling sequence for integer power involution (raising numbers to integer powers) has been changed. The associated OTS routines will have to be updated throughout any systems using F4X.

(2) All systems that support a bank mode will require a bank mode version of the F4X compiler to go along with their respective OTS libraries in order to suppress generation of PDP-15 instructions (see Appendix D). Note that a bank mode version of the FPP F4X is not needed because the FPP cannot be added to a PDP-9.

The FPP libraries (given in Appendix D) include the program .FPP which contains a special FPP error-handling routine, and routines which handle communication between the hardware CPU AC used by FORTRAN and the FPP accumulator.

All routines described in the science library and OTS utility programs are available in FPP versions with the exception of RELEAE, DOUBLE and DBLINT which are no longer required.

CHAPTER 2
INPUT-OUTPUT PROCESSING

FORTRAN data-transmission statements automatically invoke a number of OTS subroutines which serve as an interface between the user program and the Monitor. These routines may also be explicitly referred to in a MACRO program.

The actual transmission of data between memory and a peripheral device is, in general, performed by the FIOPS package, a set of routines which communicate directly with the Monitor. Other packages, each associated with a particular type of data-transmission statement, perform three major functions:

a. Initialization,

b. Transmission of data to and from the FORTRAN line-buffer in the appropriate structure, and

c. Termination;

The packages are:

(1) BCDIO, processes formatted sequential READ or WRITE statements;

(2) BINIO, processes unformatted sequential READ or WRITE statements;

(3) AUXIO, processes auxiliary input-output statements;

(4) RBCDIO and RBINIO, processes formatted and unformatted direct-access READ and WRITE statements;

(5) DDIO, manages data-directed input-output;

(6) ENCODE, processes ENCODE and DECODE statements.

Also described in this chapter is a set of FORTRAN-callable subprograms which support OTS input-output functions.

2.1 GENERAL INFORMATION

The three major I/O functions:

a. To associate logical devices with physical devices,

b. To associate user data structures with device data structures, and

c. To perform actual transfer of data

are described in the following paragraphs.

2.1.1 Device Assignment

In all systems except RSX, device assignment is managed through the monitor Device Assignment Table (.DAT) which associates logical device units to physical ones. .DAT has "slot" numbers which correspond to the logical device numbers. Each slot, at run time, contains the physical device number and a pointer to the appropriate device handler. Sixteen* entries in .DAT may be used for user-program device assignment performed via monitor ASSIGN commands at run time. Default assignments are defined during system generation.

2.1.2 Data Structures

Each peripheral device has an associated data structure which governs the manner in which data are stored. There are basically two modes in which data may be stored externally - serially or directoried. For a sequential file, either structure may be used. If it is serial, the physical sequence of records is identical to the logical sequence. If it is directoried, the logical sequence is established by pointers which link one record to another although their physical locations need not be in sequence. For a direct-access file, only directoried devices may be used.

Serial devices used for FORTRAN Input-Output include magnetic tape and DECtape. Records are transmitted directly from the user buffer to the device and an end-of-file is written after the last record by a CALL CLOSE or ENDFILE n. A file is accessed simply by virtue of device assignment.

DECtape may also be used in a directoried mode. In this case, a directory containing file information is maintained. Each entry contains a filename and extension and a pointer to the first block of the file. Files stored in this way may be referenced in the OTS directoried subroutine calls.

Directoried FORTRAN input-output to a disk, using DOS-15 file structure, is a special case. This structure is based on a hierarchy of directories with a Master File Directory (MFD) pointing to user file directories (UFDs). User files are created sequentially but may be accessed either sequentially or directly. Data blocks (400_8 words per block) which comprise a file are chained via a forward link word (377_8) and backward link word (376_8). Forward links are also stored in a retrieval information block (RIB) for direct access. Files stored in this mode are accessed by name. This name may be assigned by the user via directoried subroutines (e.g., SEEK and ENTER). If this is not done, default names are used. A default name has the form .TM0mn OTS where mn is the logical device number.

---

*This number is the standard size for DOS-15 but may be changed by system generation and assembly parameters.

2.1.3 Data Transmission

Data is transmitted to and from the FORTRAN-IV I/O buffer via the OTS FIOPS package. A single I/O buffer of 400_8 words is used. The size of the buffer which is to be transmitted for a particular device is set in accordance with information provided in an .INIT to the device used.

2.2 OTS IOPS COMMUNICATION (FIOPS)

The FIOPS package provides the necessary communication between OTS and Input-Output Processor. Its two main functions are device assignment and the transfer of data to and from the FORTRAN internal I/O buffer.

FIOPS maintains a status table with one-word entries for each file that is opened. A table entry is as shown below.

I/O Flag 0=READ 1=WRITE	0=SEQU. 1 = DIR. ACC.	For dir. acc. only 1=DELETE 0=NO	not used	Buffer size (from .INIT)
0	1	2	3 8	9 17

The routines of the FIOPS package and their functions are given below.

FIOPS Package External Calls: OTSER Errors: OTS ERROR 10 - illegal device number	
Routine	**Function**
.FC (initialize I/O Device) Call: LAC DEVICE (address of slot number) JMS* .FC To set I/O flag: DZM* .FH (input) LAC (1) (output) DAC* .FH	.DAT slot numbers are initialized by .FC. The first call to .FC for any device generates a monitor .INIT call which opens the file for I/O and enters the buffer size and I/O flag in the device status table. Subsequent calls to .FC call .INIT only if the I/O flag has been changed or the file has been closed.

(continued next page)

Routine	Function
.FQ Call: LAC (address of .DAT slot number (bits 9-17) IOPS mode (bits 6-8) JMS* .FQ	Data are transferred between the I/O buffer and an I/O device. .FQ checks the monitor I/O flag. If it is zero, a .READ call is made; if it is one, a .WRITE call is made. A call to .WAIT is made in either case.
.FP Call: JMS* .FP	Sets all words in the device status table to zero. Called at the beginning of all FORTRAN main programs to indicate that all devices are initialized.
.ZR Call: JMS* .ZR .DSA END addr .DSA ERR addr . . . . . JMS* .FF (.FG)(.RF)(.RG)	Initializes END or ERR exits. The AC is saved and restored to accomodate direct access. If one of the two exit addresses is not to be specified, an address of 0 should be passed. Direct and sequential access BCD and BINIO terminate routines reinitialize OTSER.

An integer function - IOERR (N) is available to the user and may be invoked at an ERR exit to determine the I/O error which has occured. The value of IOERR will be one of the following:

Value	Error
-1	Parity error
-2	Checksum
-3	Shortline
-5	End-of-file
-6	End-of-medium
OTS error number	Other errors (up to 77)

2.3 SEQUENTIAL INPUT-OUTPUT

Sequential input-output operations access consecutive records of a file, beginning with the first record and then record-by-record until the end of the file. A file which is accessed sequentially may be stored serially (on magnetic tape or DECtape) or in directoried mode (on disk and DECtape). That is, the physical sequence of records may or may not conform to the logical sequence.

2.3.1 OTS Binary Coded Input/Output (BCDIO)

The formatted READ and WRITE statements generate calls to routines in the BCDIO package. Input and output operations are performed on a character-to-character basis under the control of a FORMAT statement. All BCDIO routines use FIOPS to perform transfer of data. BCDIO routines may also be called directly by MACRO programs.

Each formatted record is an IOPS ASCII line with a two-word header pair. The first character after the header is always a forms-control character. Record length, given in the header, is always in terms of word-pairs. The last character in the last word-pair is always a carriage return.

BCDIO routines are described below.

BCDIO Package

External Calls:	FIOPS, OTSER, REAL, RELNON or RELEAE
Errors:	OTS 10 - illegal I/O device number OTS 11 - bad input data (IOPS mode incorrect) OTS 12 - illegal format

Routine	Function
.FR (.FW) Call: JMS* .FR (.FW) .DSA (address of .DAT slot number) .DSA (address of first word of FORMAT statement or array)*	Inputs (outputs) a data item.
.FE Call: JMS* .FE .DSA (address of data item (first word))	Inputs or outputs a data item using format decoder (.FD).
.FA Call: JMS* .FA .DSA (address of last word in array descriptor block)	Inputs or outputs an entire array using format decoder (.FD).

(continued next page)

*This word is 0 for data-directed I/O

Routine	Function
.FD Call: JMS* .FD	Decodes format into four parameters: .D - decimal places .W - field width .SF - scale factor .S - mode
.FF Call: JMS* .FF	Terminates the current logical record.

As described in the language manual*, FORMAT statements may be entered or changed at run time, at which point they are interpreted by BCDIO. In addition to providing the FORTRAN programmer with greater flexibility, this feature permits the MACRO programmer to use the formatted I/O capabilities of BCDIO. (See Appendix C for examples.)

2.3.2 OTS Binary Input/Output (BINIO)

The BINIO package processes unformatted READ and WRITE statements. Data transfer is on a word-to-word basis. A logical record, the amount of data associated with a single READ or WRITE statement, may consist of several physical records whose size (except for the last) is always the standard IOPS I/O buffer size. Thus, when a WRITE statement is processed, each physical record generated contains an ID word (word 3) in addition to the two required header words. This word contains a record identification number. For the first record, this is zero. The last record is indicated by setting bit 0 of the ID word to 1. Up to 377777_8 physical records may be generated for a single logical record.

For example, if four physical records are generated, the four ID words would be:

```
000000
000001
000002
400003
```

If only one record is generated, its ID word will be 400000 signifying the first and last of a set.

An unformatted READ statement accepts logical records of the form described above until its I/O list has been satisfied. If this occurs in the middle of a logical record, the remainder of the record is ignored. That is, the next READ will access the beginning of the next logical record.

*DEC-15-GFWA-D

The routines of BINIO are described below.

BINIO	
External Calls: FIOPS, OTSER	
Errors: OTS 10 - illegal I/O device number OTS 11 - illegal input data (IOP mode)	
Routine	**Function**
.FS Call: JMS* .FS .DSA (address of .DAT slot)	Initializes a device for binary input and reads first record.
.FX Call: JMS* .FX .DSA DEVICE	Initializes a device for binary output; initializes line buffer.
.FJ Call: JMS* .FJ .DSA (address of item (first) word)	Transfers a data item to or from the line buffer (all modes). Mode of item indicated by bits 1 - 2 of argument are: 00 = INTEGER 01 = REAL 10= DOUBLE PRECISION 11 = DOUBLE INTEGER
.FB Call: JMS* .FB .DSA (address of last word in array descriptor block)	Transfers an array.
.FG Call: JMS* .FG	Terminates current logical record. For WRITE, packs the line buffer with zeroes as required and sets bit 0 of the ID word.

2.3.3 OTS Auxiliary Input/Output (AUXIO)

The AUXIO package processes the commands BACKSPACE, REWIND, and ENDFILE which have different meanings for magnetic tape and disk. AUXIO routines issue .MTAPE monitor calls giving .DAT slot and a code specifying the magnetic tape function desired:

Code	Magnetic Tape	Disk
00	Rewind to load point	Close file associated with .DAT slot.
02	Backspace record	Pointers resumed for previous ASCII or binary line.
04	Write end-of-file	N.A.

For magnetic tape, these operations require only calls to system macros. In order to simulate magnetic tape functions on disk, a file active table (.FLTB) must be referenced. This contains four-word entries for every positive .DAT slot indicating whether the file is active (open for input or output) or inactive. The routines of AUXIO and their serial and file-oriented functions are given below.

AUXIO		
External Calls:	FIOPS, .FLTB	
Errors:	OTS 10 - illegal I/O device OTS 11 - illegal input data (IOPS mode incorrect)	
Routine	**Magnetic Tape**	**Disk**
.FT (BACKSPACE) Call: JMS* .FT .DSA (address of .DAT slot)	Repositions device at a point just prior to the first physical record associated with the current logical record.	Resumes pointer to previous ASCII or binary line.
.FU (REWIND) Call: JMS* .FU .DSA (address of .DAT slot)	Repositions device at load point.	Closes file. If no file is open, nothing is done.
.FV (ENDFILE) Call: JMS* .FV .DSA DEVICE	Closes file. Writes an end-of-file mark on tape.	Closes file, zeroes words 0-3 of the associated .FLTB entry.

On a REWIND to disk, the filename is saved; thus, subsequent sequential input-output operations will open that file. On an ENDFILE, the filename is lost and subsequent operations will open a default file.

2.4 DIRECT ACCESS I/O

Direct access input-output files are referenced by name; records are retrieved or accessed by number. The OTS routines which perform direct-access transmission of data are similar to their sequential counterparts. Before they are invoked, however, the user must provide a detailed description of his file.

2.4.1 The DEFINE Routine

The FORTRAN user establishes a direct-access file by calling the DEFINE routine which was described in Part I, Chapter 6. The meanings of its arguments are iterated below for the call:

CALL DEFINE (D, S, N, F, V, M, A, L)

The parameters provided to OTS for performing direct-access functions are:

D - .DAT slot

S - record size
number of ASCII characters
or
number of binary words

N - number of records ($\leq 377777_8$)

F - array reference to file name and extension - if 0, default name

V - associated variable - set to number of the last accessed record plus one

M - mode -0 = IOPS binary
non-0 = IOPS ASCII

A - file size adjustment indicator
0 = no adjustment
non-0 = adjust

L - deletion indicator
0 = no deletion
non-0 = delete temporary file

The DEFINE routine initializes a file for direct-access in one of four ways, depending on the combination of parameters supplied.

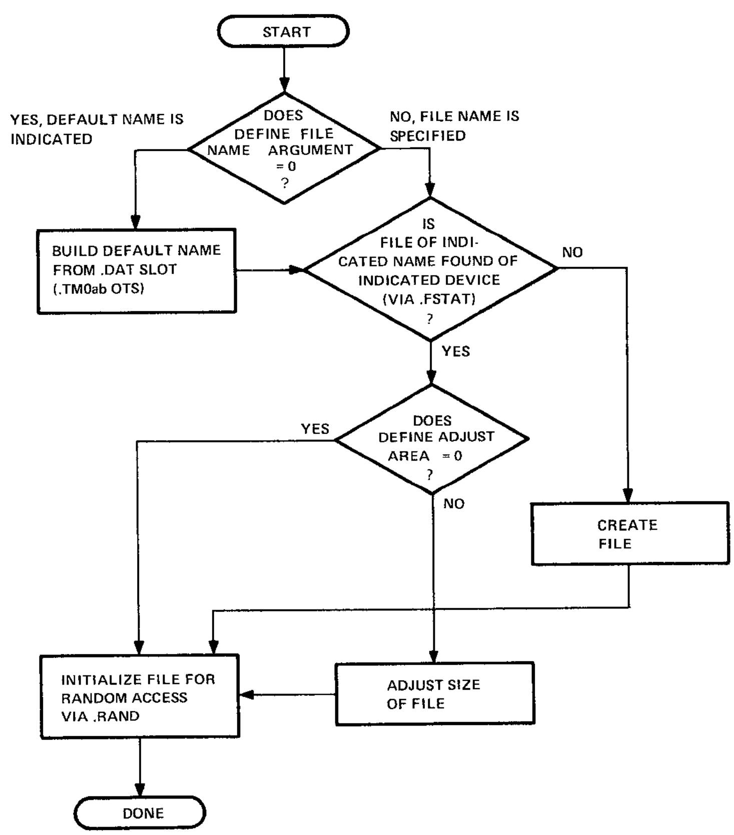
a. Simple Initialization - If F specifies a file which already exists and no adjustment has been indicated, DEFINE opens the file for direct access. The mode and record length parameters must conform to the file's characteristics. The associated variable is set to 1. The number of records N must be less than or equal to the actual number of records.

b. Named File Creation - If F specifies a file which does not exist on .DAT slot D, a file is created according to the characteristics given in the calling arguments. If the mode is ASCII, the data portion is filled with spaces (040_8). If the mode is binary, all data words are set to 0 and the ID word for each record to 400000_8.

c. Default-Named File Creation - If F=0 in the DEFINE call, a file is created as above but given a default name of the form .TM0ab OTS (unless a file of that name already exists on .DAT slot D) where ab specifies .DAT slot. If L=1, a bit is set in the FIOPS status table signifying that the file is to be deleted after an ENDFILE or CALL CLOSE to the .DAT slot.

d. File Size Adjustment - If a file F exists and A is not zero, N is used to adjust the number of records in the file. This is done by creating a temporary file (..TEMP OTS) on .DAT slot D via .DAT slot -1 which is temporarily loaded with the .DAT slot D handler address and UIC. The file is copied into it one record at a time up to the number N. If the file is to be lengthened, null records are added. The adjusted file is then assigned a name according to F. V is set to 1 if the file is reduced. If it is lengthened, it is set to the old length plus one.

The algorithm used for determining the function of DEFINE from its arguments is illustrated in the following flowchart.



From user-supplied arguments, the DEFINE routine establishes a parameter table (PRMTB) which is available to direct-access input-output routines.

Each device which has a file open for direct-access will have an active four-word entry composed as follows:

Word	Bits	Information
1	0 2-11 12-17	File active bit (1 if active - always set for ASCII files) Number of blocks per record .DAT slot number
2	0 5-11 12-17	mode - 0 if binary; 1 if ASCII Word pairs per record Records per block (0 for binary records larger than one physical block)
3	1-17	Records/file
4	3-17	Address of associated variable

.PRMTB will generally have four such entries but this number may be varied with an assembly parameter.

DEFINE also initializes the file in FIOPS, setting the appropriate bits in the FIOPS status table.

2.4.2 Formatted Input/Output (RBCDIO)

Direct-access operations may be performed on any formatted data file conforming to DOS-15 file structure and with a fixed record length. A direct-access WRITE will output formatted records which have the same form as with sequential operations. The distinction is that the direct-access records are transmitted into a series of records which already exist on the selected file. A single READ or WRITE will access records on the I/O device only as specified in the associated FORMAT statement. This means that a long I/O list will not cause a new record to be accessed, regardless of the length of the list, unless this access is indicated by the FORMAT statement. A carriage return is, as with sequential I/O, appended to each ASCII line. Any information from a previous WRITE mode to a record which remains after the carriage return, is inaccessible. The FIOPS buffer and tables are used as with sequential I/O. Data transfer, however, is performed using the .RTRAN system MACRO.

The RBCDIO routines described below correspond to the sequential I/O routines of BCDIO. Control is transferred to BCDIO for data transmission via the global entry points given.

RBCDIO External Calls: FIOPS, BCDIO (.FE, .FA), OTSER, RANCOM Errors: None	
Routine	**Purpose**
.RW (.RR) Call: JMS* .RW (RF) .DSA (address of .DAT slot) .DSA (address FORMAT) (AC holds integer record number)	BCD direct-access WRITE (READ) sets the direct-access flag; sets mode switch to ASCII; initializes direct-access READ/WRITE (.INRRW in RANCOM); checks mode of existing record; initializes - .STEOR and BFLOC in BCDIO for direct-access, line buffer, and form at decoder; sets .HILIM in BCDIO. .RW loads record number into .RCDNM and sets I/O flag in FIOPS to write. .RR loads record number into .RCDNM, sets I/O flag to read.
.RF Call: JMS* .RF	Terminates current logical record. Sets last record flag, reinitializes .ER in OTSER and, for WRITE, .RTRAN out last record.

Entry points to BCDIO are:

RBCDIO Entry	BCDIO Routines
.RE	.FE
.RA	.FA

2.4.3 Unformatted Input/Output (RBINIO)

Unformatted direct-access I/O differs from formatted in two respects. If a binary record does not totally fill the record into which it is written, the previous contents are still accessible. If a direct-access WRITE requires more words than exist in each record, successive records are accessed and written until the I/O list is exhausted. Records are linked by ID words as for sequential files.

The routines of RBINIO are described below. Direct-access entry points to BINIO follow.

RBINIO External Calls: FIOPS, RANCOM, BINIO Errors: None	
Routine	**Function**
.RS (.RX) Call: JMS* .RS (.RX) .DSA (address of .DAT slot) (AC holds integer record number)	Binary direct-access WRITE (READ) sets direct-access flag; sets mode switch to binary; initializes direct READ/WRITE (.INRRW in RANCOM); checks mode of existing record; initializes .BUFLC, .RDTV, and .WRTV in BINIO for direct access; initializes I/O buffer; loads record number into .RCDNM. .RX sets I/O flag to WRITE; .RS sets it to READ.

(continued next page)

Routine	Function
.RG Call: JMS* .FG	Terminates current logical record. Increments associated variable, reinitializes .ER in OTSER; if WRITE, sets last record flag and outputs final records.

2.4.4 Initialization and Actual Data Transfer (RANCOM)

RANCOM contains two major routines which are used by both RBCDIO and RBINIO. These routines perform initialization and data transfer functions which are identical to those performed for ASCII and Binary I/O.

RANCOM	
External Calls:	FIOPS, OTSER, DEFINE
Errors:	OTS 10 - illegal I/O device OTS 24 - illegal record number OTS 25 - mode discrepancy OTS 11 - illegal input data (IOPS mode incorrect) OTS 21 - undefined file OTS 23 - size discrepancy
Routine	**Function**
.INRRW Call: JMS* .INRRW (AC holds address of slot number.)	Initializes a direct access READ or WRITE
.RIO Call: JMS* .RIO	For I/O cleanup: Set up header pair and .RTRAN out block of data. For end-of-record routines: Output (if WRITE) and set pointers to new record.

2.5 Data-Directed Input-Output (DDIO)

The Data-Directed Input-Output package permits input or output of ASCII data without reference to a FORMAT statement. On input, DDIO extracts individual data fields by scanning the line buffer for terminators. It then determines the mode of the variable to which the item is to be transferred and converts the item to that mode if necessary. Unlike the format decoder, DDIO does not reject an item which is too large but simply assigns the maximum value which the variable can accomodate. On output, DDIO has a set of default format parameters for each type of variable.

The same buffer is used for both data-directed and formatted I/O, and the I/O action for both takes place between device and I/O list variables or vice versa in both cases. Thus, DDIO uses the same I/O initialization and termination routines as regular formatted I/O (found within BCDIO for sequential access and within RBCDIO for direct access). DDIO control routines are, however, unique due to the special features described above.

The routines of DDIO are given below.

DDIO

External Calls: BCDIO, .SS, OTSER, FIOPS, REAL, DBLINT

Errors: OTS 42 - bad input data*

Routine	Function
.GA Call: JMS* .GA / radix 50 name 1 } first 3 characters name 2 } last 3 characters .DSA address item	Outputs a data item in the 'NAME' = value form. Mode is obtained from bits 1-2 of the pointer word; if the mode is 0 (integer-logical), bit 0 of the name word indicates which (0 for integer, 1 for logical).
.GC Call: JMS* .GC / radix 50 name 1 name 2 .DSA item	Outputs an array element in 'NAME (I)' = value form. Also uses bits 1-2 for mode. .GC should only be used when .SS has been used to calculate the subscript address.
.GB Call: JMS* .GB / radix 50 name 1 name 2 .DSA array description block (word #4 address)	Outputs an entire array in 'NAME(I)' = value form.
.GD Call: JMS* .GD .DSA item	Inputs an item. Mode is in bits 1-2 of argument.
.GE Call: JMS* .GE .DSA addr. of array discriptor block word 4	Inputs an array. Mode is in bits 1-2 of argument.

*For Teletype input - 'BAD INPUT DATA - RETYPE FROM INPUT WITH ERROR' is typed.

2.6 ENCODE/DECODE (EDCODE)

Encode and Decode perform memory-to-memory transfers and conversions using the apparatus established for formatted input-output. That is, data is transferred from memory to the I/O buffer to memory. Since no peripheral device is involved, the initialization and termination mechanisms of EDCODE are unique while the data transfer is the same as for BCDIO.

The routines of EDCODE are given below.

EDCODE

External Calls: OTSER, BCDIO

Errors: OTS 40 - illegal number of characters
OTS 41 - array exceeded

Routine	Function
.GF Call: JMS* .GF .DSA number of characters .DSA array .DSA format	Encode.
.GG Call: JMS* .GG .DSA number of characters .DSA array .DSA format	Decode.

2.7 USER SUBROUTINES

The subroutines given below are FORTRAN-callable subroutines which support input-output operations.

2.7.1 Magnetic Tape Input-Output Routines*

Routine	Call	Function
EOF	CALL EOF(d,@n_1, @n_2) Where: d = .DAT slot (must be assigned to tape) n_1,n_2 = statement numbers	Control is passed to n_1 if EOF was encountered on last input operation; otherwise to n_2

*Not supported with RSX. END, ERR exits can be used in place of EOF. (continued next page)

Routine	Call	Function
IOCHECK	CALL IOCHECK (d,@n_1,@n_2)	Same
UNIT	CALL UNIT (d,@n_1,@n_2,@n_3, @n_4)	Control is passed to: n_1 - device not ready n_2 - device ready, no previous error n_3 - EOF sensed n_4 - parity or lost data error

2.7.2 Directoried Subroutines

The directoried subroutines described below comprise a package named FILE. These routines interact with the DOS-15 file-oriented data structure and with DECtape file structure.

FILE

External Calls: FIOPS, .DA

Errors:
OTS 10 - illegal device number
OTS 13 - file not found (SEEK)
OTS 14 - directory full (ENTER)

Routine	Call	Purpose
SEEK	CALL SEEK (n,A) Where: n = device number A = name of array containing the 9-character 5/7 ASCII file name and extension	Finds and opens a named input file.
ENTER	CALL ENTER (n,A)	Creates and opens a named output file.
CLOSE	CALL CLOSE (n)	Terminates an input or output file (required when SEEK or ENTER are used).
FSTAT	CALL FSTAT (n,A,I) Where: I = 0 if the file not found; = 1 if found and action complete	Searches for named file.

(continued next page)

Routine	Call	Purpose
RENAM	CALL RENAM (n,A,B,I) Where: A is an array containing existing name B is an array containg a new file name I = 0 if file not found; 1 if found and action complete	Searches for named file and renames it.
DLETE	CALL DLETE (n,A,I) Where: A is an array containing existing file name I = 0 if file not found; 1 if found and action complete	Searches for named file and deletes it.

CHAPTER 3
THE SCIENCE LIBRARY

The FORTRAN Science Library is a set of pre-defined subprograms which may be invoked by a FORTRAN-IV subprogram reference. These include intrinsic functions, external functions, the arithmetic-package functions, and external subroutines. Each of these may also be referenced by a MACRO program as may the sub-functions and OTS routines which are also part of the FORTRAN library.

Descriptions of each type of subprogram are given in the following subsections. Information given for these include errors, accuracy, size, and external calls (to other library subprograms). Each function description also includes the MACRO calling sequence. Where there are two arguments, it is assumed that the appropriate accumulator has been loaded (accumulators are described in Section 3.4). For calling sequences which use the .DSA pseudo-operation to define the symbolic address of arguments, 400000 must be added to the address field for indirect addressing.

FORTRAN library subprograms are called by FORTRAN programs in the manner described in the Language Manual (DEC-15-GFWA-D). Subprograms called by MACRO programs must be declared with a .GLOBL pseudo-operation as in:

Examples:

Standard System

```
    .TITLE
    .GLOBL SIN, .AH
    .
    .
    .
    JMS*  SIN
    JMP  .+2          /JUMP beyond argument
    .DSA  A           /+400000 if indirect
    JMS*  .AH         /store in real format at
    .DSA X            /X
    .
    .
    .
  X .DSA 0
    .DSA 0
```

Floating Point (FPP) System

```
         .TITLE
         .GLOBL SIN
FST = 713640
         .
         .
         .
         JMS  SIN
         JMP .+2
         .DSA  A
         FST
         .DSA X
X        .DSA 0
         .DSA 0
```

The number and type of arguments in the MACRO program must agree with those defined for the subprogram.

3.1 INTRINSIC FUNCTIONS

Table 3-1 contains a description of each of the intrinsic functions in the FORTRAN library.

An intrinsic function's type and arguments cannot be changed. It is referenced via an Arithmetic statement, as in:

```
X = ABS (A)
```

(Table 3-1 appears on the following page.)

Table 3-1
Intrinsic Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External Calls
		.BB	I=I**I	ARG1 IN FLT.ACC JMS*.BB .DSA ADDR of ARG2	15 if base = 0 and exp. ≤ 0	N.A.	INTEGER
		.BC .BC .BL	R**I(or J) R=R**I R=R**J	ARG1 IN FLT. ACC {JMS* SUBR .DSA ADDR of ARG2	None	N.A.	REAL
		.BD .BD .BM	D**I(or J) D=D**I D=D**J	ARG1 IN FLT. ACC {JMS* SUBR .DSA ADDR of ARG2	None	N.A.	REAL
		.BE .BF .BG .BH	R=R**R D=R**D D=D**R D=D**D	ARG1 IN FLT. ACC {JMS* SUBR .DSA ADDR of ARG2	13 if base ≤ 0 13 if base ≤ 0 14 if base ≤ 0 14 if base ≤ 0	26 26 32 32	.EE,.DF, REAL .EE,.DF, DOUBLE .DE,.DF, DOUBLE .DE,.DF, DOUBLE
		.BI .BI .BJ .BK	I**J, J**J(or I) I=I**J J=J**J J=J**I	ARG1 IN AC (and MQ) {JMS* SUBR .DSA ADDR of ARG2	None	N.A.	DBLINT
Absolute Value	|ARG|	ABS IABS JABS DABS	R=ABS(R) I=IABS(I) DI=JABS(DI) DP=DABS(DP)	JMS* SUBR JMP .+2 .DSA ADDR of ARG	None	N.A.	.DA,REAL .DA .DA, DBLINT .DA, DOUBLE
Truncation	Sign of ARG times largest integer ≤ |ARG|	AINT INT IDINT JINT JDINT	R=AINT(R) I=INT(R) I=IDINT(DP) DI=JINT(R) DI=JDINT(DP)	{JMS* SUBR JMP .+2 .DSA ADDR of ARG	None	N.A.	.DA, REAL .DA, REAL .DA, REAL, DOUBLE .DA, DOUBLE, DBLINT .DA, DOUBLE, DBLINT

*15 if base = 0 and exp ≤ 0.

Table 3-1 (Cont)
Intrinsic Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External Calls
Transfer of Sign	Sign of ARG2 ↓ Sign of ARG1	SIGN ISIGN DSIGN JSIGN	R=SIGN(R,R) I=ISIGN(I,I) DP=DSIGN(DP,DP) DI=JSIGN(DI,DI)	JMS* SUBR JMP .+3 .DSA ADDR of ARG1 .DSA ADDR of ARG2	None	N.A.	.DA, REAL .DA .DA, DOUBLE .DA, DBLINT
Positive Difference	ARG1-MIN(ARG1,ARG2)	DIM IDIM JDIM	R=DIM(R,R) I=IDIM(I,I) DI=JDIM(DI,DI)	JMS*SUBR JMP .+3 .DSA ADDR of ARG1 .DSA ADDR of ARG2	None	N.A.	.DA, REAL .DA, INTEGER .DA,DBLINT
Conversion	VMODE → ARG	FLOAT IFIX SNGL DBLE JFIX ISNGL IDBLE JDFIX FLOATJ DBLEJ	R=FLOAT(I) I=IFIX(R) R=SNGL(D) D=DBLE(R) DI=JFIX(R) or JFIX(DP) I=ISNGL(DI) DI=JDBLE(I) DI=JDFIX(DP) R=FLOATJ(DI) DP=DBLEJ(DI)	JMS* SUBR JMP .+2 .DSA ADDR of ARG	None	N.A.	.DA, REAL .DA, REAL .DA, DOUBLE .DA, REAL .DA, DOUBLE, DBLINT .DA, .DA, DBLINT .DA .DA, DOUBLE, DBLINT .DA, DBLINT .DA,DBLINT
Remaindering	ARG1-[ARG1/ARG2]ARG2 Where: [A1/A2] is an integer whose magnitude does not exceed the magnitude of A1/A2 and whose sign is the same	AMOD MOD DMOD JMOD	R=AMOD(R,R) I=MOD(I,I) DP=DMOD(DP,DP) DI=JMOD(DI,DI)	JMS* SUBR JMP .+3 .DSA ADDR of ARG1 .DSA ADDR of ARG2	None	N.A.	.DA, REAL .DA, INTEGER .DA, DOUBLE .DA, DBLINT

Table 3-1 (Cont)
Intrinsic Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External Calls
Maximum/ minimum value	VAR = max or min value of arglist	Integer min/max (IMNMX) MAX0 MIN0 AMAX0 AMIN0	 $I=MAX0(I_1,...I_n)$ $I=MIN0(I_1,...I_n)$ $R=AMAX0(I_1,...I_n)$ $R=AMIN0(I_1,...I_n)$	JMS*SUBR JMP .+n+1 .DSA ADDR of ARG1 ⋮ .DSA ADDR of ARGn (applies to all rows)	None	N.A.	INTEGER, REAL
		Real min/max (RMNMX) AMAX1 AMIN1 MAX1 MIN1	 $R=AMAX1(R_1,...R_n)$ $R=AMIN1(R_1,...R_n)$ $I=MAX1(R_1,...R_n)$ $I=MIN1(R_1,...R_n)$				INTEGER, REAL
		Double-precision (DMNMX) DMAX1	 $DP=DMAX1(DP_1,...DP_n)$ $DP=DMIN1(DP_1,...DP_n)$				DOUBLE
		Double integer (JMNMX) JMAX0 JMIN0	 $DI=JMAX0(DI_1,...DI_n)$ $DI=JMIN0(DI_1,...DI_n)$				DBLINT

3.2 EXTERNAL FUNCTIONS

Table 3-2 describes the external functions of the FORTRAN library. An external function is a subprogram which is executed whenever a reference to it appears within a FORTRAN expression and which returns a single value.

A description of the algorithm applied in implementing each of these functions is given below.

3.2.1 Square Root (SQRT, DSQRT)

A first-guess approximation of the square root of the argument is obtained as follows:

If the exponent (EXP) of the argument is odd:

$$P_0 = .5^{\left(\frac{EXP-1}{2}\right)} + ARG^{\left(\frac{EXP-1}{2}\right)}$$

If EXP is even:

$$P_0 = .5^{\left(\frac{EXP}{2}\right)} + ARG^{\left(\frac{EXP}{2}-1\right)}$$

Newton's iterative approximation, below, is then applied four times.

$$P_{i+1} = \frac{1}{2}\left(P_i + \frac{ARG}{P_i}\right)$$

3.2.2 Exponential (EXP, DEXP)

The following description also applies to the sub-functions .EF and .DF.

The function e^x is calculated as $2^{x\log_2 E}$ ($x\log_2 E$ will have an integer portion (I) and fractional portion (F)).

Then:

$$e^x = (2^I)(2^F)$$

Where:

$$2^F = \left(\sum_{i=0}^{n} C_i F^i\right)^2$$

n = 6 for EXP and .EF

n = 8 for DEXP and .DF

(continued page 3-7)

Table 3-2
External Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External Calls
Square root	$ARG^{1/2}$	SQRT DSQRT	R=SQRT(R) DP=DSQRT(DP)	JMS*SUBR JMP .+2 .DSA ADDR of ARG	5 if ARG < 0 6 if ARG < 0	26	.DA,.ER,REAL .DA,.ER,DOUBLE
Exponen-tial	e^{ARG}	EXP DEXP	R=EXP(R) DP=DEXP(DP)	Same	13 if ARG < 0 14 if ARG < 0	26 34	.DA,.EF,.ER,REAL .DA,.DF,.ER,DOUBLE
Natural logarithm	$Log_e ARG$	ALOG DLOG	R=ALOG(R) DP=DLOG(DP)	Same	Same	26 32	.DA,.EE,.ER,REAL .DA,.DE,.ER,DOUBLE
Common logarithm	$Log_{10} ARG$	ALOG10 DLOG10	R=ALOG10(R) DP=DLOG10(DP)	Same	Same	Same	Same
Sine	Sin(ARG)	SIN DSIN	R=SIN(R) DP=DSIN(DP)	Same	None	26 34	.DA,.EB,REAL .DA,.DB,DOUBLE
Cosine	cos(ARG)	COS DCOS	R=COS(R) DP=DCOS(DP)	Same	None	26 34	.DA,.EB,REAL .DA,.DB,DOUBLE
Arc tangent	tan^{-1}(ARG)	ATAN DATAN	R=ATAN(R) DP=DATAN(DP)	Same	None	26 34	.DA,.ED,REAL .DA,.DD,DOUBLE
Arc tangent (X/Y)	tan^{-1} (ARG1/ ARG2)	ATAN2 DATAN2	R=ATAN2(R,R) DP=DATAN2 (DP,DP)	JMS*SUBR JMP .+3 .DSA ADDR of ARG1 .DSA ADDR of ARG2	None	26 34	Same
Hyper-bolic tangent	tanh(ARG)	TANH	R=TANH(R)	JMS*TANH JMP .+2 .DSA ADDR of ARG	None	26	.DA,.EF,REAL

The values of C_i are given below.

Value of i	Value of C_i
0	1.0
1	0.34657359
2	0.06005663
3	0.00693801
4	0.00060113
5	0.00004167
6	0.00000241
7	0.00000119
8	0.00000518

3.2.3 Natural and Common Logarithms (ALOG, ALOG10, DLOG, DLOG10)

The exponent of the argument is saved as the integral portion of the result plus one. The fractional portion of the argument is considered to be a number between 1 and 2. Z is computed as follows:

$$Z = \frac{X-\sqrt{2}}{X+\sqrt{2}}$$

Then:

$$\log_2 X = \frac{1}{2} + (\sum_{i=0}^{n} C_{2i+1} Z^{2i+1})$$

Where:

n = 2 (ALOG)

n = 3 (DLOG)

The values of C are given below:

ALOG and ALOG10	DLOG and DLOG10
C_1 = 2.8853913	C_1 = 2.8853900
C_3 = 0.96147063	C_3 = 0.96180076
C_5 = 0.59897865	C_5 = 0.57658434
	C_7 = 0.43425975

(continued next page)

The final computation is:

ALOG and DLOG: $\log_e X = (\log_2 X)(\log_e 2)$

ALOG10 and DLOG10: $\log_{10} X = (\log_2 X)(\log_{10} 2)$

3.2.4 Sine and Cosine (SIN, COS, DSIN, DCOS)

This description also applies to the sub-functions .EB and .DB.

The argument is multiplied by $2/\pi$ for conversion to quarter-circles. The two low-order bits of the integral portion determine the quadrant of the argument and produce a modified value of the fractional portion (Z) as follows.

Low-Order Bits	Quadrant	Modified Value (Z)
00	I	F
01	II	1-F
10	III	-F
11	IV	-(1-F)

The value of Z is then applied to the polynomial expression:

$$\sin X = \left(\sum_{i=0}^{n} C_{2i+1} Z^{2i+1}\right)$$

n = 4 for SIN, COS, .EB

n = 6 for DSIN, DCOS, .DB

The values of C are as follows:

SIN, COS, .EB	DSIN, DCOS, .DB
$C_1 = 1.570796318$	$C_1 = 1.5707932680$
$C_3 = -0.645963711$	$C_3 = -0.6459640975$
$C_5 = 0.079689677928$	$C_5 = 0.06969262601$
$C_7 = -0.00467376557$	$C_7 = -0.004681752998$
$C_9 = 0.00015148419$	$C_9 = 0.00016043839964$
	$C_{11} = -0.000003595184353$
	$C_{13} = 0.000000054465285$

(continued next page)

The argument for COS and DCOS is adjusted by adding $\pi/2$. The sin subfunction is then used to compute the cosine according to the following relationship:

$$\text{COS } X = \sin\left(\frac{\pi}{2} + X\right)$$

3.2.5 Arctangent (ATAN, DATAN, ATAN2, DATAN2)

The following description also applies to the sub-functions .ED and .DD.

For arguments less than or equal to 1, Z = arg and:

$$\text{arctangent arg} = \left(\sum_{i=0}^{n} C_{2i+1} Z^{2i+1}\right)$$

n = 7 for ATAN and ATAN2

n = 3 for DATAN and DATAN2

For arguments greater than 1, Z = 1/arg and:

$$\text{arctangent arg} = \frac{\pi}{2} - \left(\sum_{i=0}^{n} C_{2i+1} Z^{2i+1}\right)$$

n = 8 for ATAN and ATAN2

n = 3 for DATAN and DATAN2

The values of C are given below.

ATAN and ATAN2	DATAN and DATAN2
$C_1 = 0.9992150$	$C_1 = 0.9999993329$
$C_3 = -0.3211819$	$C_3 = -0.3332985605$
$C_5 = 0.1462766$	$C_5 = 0.1994653599$
$C_7 = -0.0389929$	$C_7 = -0.1390853351$
	$C_9 = 0.0964200441$
	$C_{11} = -0.0559098861$
	$C_{13} = 0.0218612288$
	$C_{15} = -0.0040540580$

3.2.6 Hyperbolic Tangent

The hyperbolic tangent function is defined as:

$$\tanh |X| = \left(1 - \frac{2}{1+e^{2|X|}}\right)$$

e^x is calculated as $2^{x\log_2 e}$ ($x\log_2 e$ will have an integral portion (I) and a fractional portion (F)).

(continued next page)

Then:

$$e^x = (2^I)(2^F)$$

Where:

$$2^F = \left(\sum_{i=0}^{n} C_i F^i\right)^2$$

$$n = 6$$

The values of C_i are:

Value of i	Value of C_i
0	1.0
1	0.34657359
2	0.06005663
3	0.00693801
4	0.00060113
5	0.00004167
6	0.00000241

3.3 SUB-FUNCTIONS

Table 3-3 describes the sub-functions which are included in the FORTRAN library. These functions are referenced by intrinsic and external functions but are not directly accessible to the user via FORTRAN. The sub-function .EB, for example, performs the computation of sine and is invoked by the external function SIN. MACRO programs may reference sub-functions directly. Algorithms for all sub-functions which have counterparts among external functions were given in the previous sub-section. This leaves the two general sub-functions Logarithm, base 2 and polynomial evaluator. Their algorithms are given below.

3.3.1 Logarithm, Base 2 (.EE, .DE)

The exponent of the argument is saved as the integer portion of the result plus one. The fractional portion of the argument is considered to be a number between 1 and 2. Z is computed as follows:

$$Z = \frac{X - \sqrt{2}}{X + \sqrt{2}}$$

(continued page 3-14)

Table 3-3
Sub-Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External Calls
Sine Computation	Sin (ARG)	.EB .DB	R=.EB(R) DP=.DB(DP)	JMS*SUBR At entry floating accumulator contains ARG; at return contains result	None	19 28	.EC,REAL .DC,DOUBLE
Arc tangent Computation	$\tan^{-1}$(ARG)	.ED .DD	R=.ED(R) DP=.DB(DP)	Same	None	26 34	Same
Logarithm (base 2) Computation	$\log_2$ARG	.EE .DE	R=.EE(R) DP=.DE(DP)	Same	13, ARG < 0 14, ARG < 0	26 32	.ER,REAL .ER,DOUBLE
Exponential Computation	e^{ARG}	.EF .DF	R=.EF(R) DP=DF(DP)	Same	None	26 34	REAL DOUBLE
Polynomial Evaluation	VAR = $\sum_{1=0}^{n} C_{2i+1} Z^{2i+1}$ VAR = $\sum_{i=0}^{n} C_{2i+1} Z^{2i+1}$	.EC .DC	R=.EC(R_2,R_1,...R_n) DP=.DC(DP_2,DP_1, ...DP_n)	JMS*SUBR CAL PLIST . . . PLIST-N/ - number of terms +1 C_n / last term C_n-1/next to last . . . C_1 /2nd term C_g /1st term	None	N.A.	REAL DOUBLE

(continued next page)

Table 3-3 (Cont)

Sub-Functions

Function	Definition	Symbolic Name	Mode	Calling Sequence	Errors	Accuracy (Bits)	External
General Get Argument	N.A	.DA	N.A	<u>Calling Routine</u> SUBR CAL 0 JMS*.DA JMP .+n+1 (address of ARG1) (address of ARG2) . . . (address of ARGn) <u>Is Called By</u> JMS*SUBR JMP .+n+1 .DSA ARG1 .DSA ARG2	None	N.A	None

Then:

$$\log_2 X = \frac{1}{2} + (\sum_{i=0}^{n} C_{2i+1} Z^{2i+1})$$

n = 2 (.EE)

n = 3 (.DE)

The values of C are:

.EE	.DE
$C_1 = 2.8853913$	$C_1 = 2.8853900$
$C_3 = 0.96147063$	$C_3 = 0.96180076$
$C_5 = 0.59897865$	$C_5 = 0.57658434$
	$C_7 = 0.43425975$

3.3.2 Polynominal Evaluator (.EC, .DC)

A polynomial is evaluated as:

$$X = Z(C_0 + Z^2 (C_1 \ldots + Z^2 (C_n Z^2 + C_{n-1})))$$

3.4 THE ARITHMETIC PACKAGE

The arithmetic package contains the OTS arithmetic routines which are invoked by FORTRAN arithmetic expressions. These routines may also be called directly by MACRO programs. Versions of FORTRAN-IV designed for use with the Floating Point Processor (FPP) require only single integer arithmetic routines. Double (extended) integer arithmetic will be handled by the hardware.

The three major routines of the arithmetic package are INTEAE, RELEAE, and DOUBLE. INTEAE contains integer arithmetic routines; RELEAE, real and floating arithmetic; and DOUBLE, double-precision arithmetic.

A description of these routines is given in Table 3-4. In the "calling sequence" column, reference is made to three accumulators - the A-register, the floating accumulator, and the held accumulator. The A-register is the standard PDP-15 hardware accumulator. The floating and held accumulators are software accumulators which are part of the RELEAE package. The held accumulator is used as temporary storage by some routines. Both consist of three consecutive PDP-15 words and have the format shown below. (Negative mantissae are indicated by a change of sign.)

Held AC Labels	Floating AC Labels	
CE01	.AA	Exponent (2's complement) 0 … 17
CE02	.AB	Sign of mantissa (0) | High-order mantissa (1 … 17)
CE03	.AC	Low order mantissa 0 … 17

The format shown above is that used for double-precision numbers. Single-precision numbers must be converted before and after use in the floating accumulator to the single-precision format:

Low-order mantissa	Exponent (2's complement)
0 … 8	9 … 17

Sign of mantissa	High-order mantissa
0	1 … 17

RELEAE routines check for underflow and overflow and set a flag (.OVUDF) in the REAL store routine .AH as follows:

Flag	Meaning	Action
non-0 positive value	overflow - an attempt to store a REAL constant whose binary exponent is greater than 377_8	± largest representable real value stored (DOS-15);
negative value	underflow - an attempt to store a REAL constant whose binary exponent is less than -400_8	zero is stored
zero	default value	value is stored

The user may test this flag under program control using the logical function IFLOW. Recoverable OTS messages are also given (see Appendix B, Section B.2).

Division by zero is also checked and a flag .DZERO set to zero (default value is 777777) in the general floating divide routine (.CI). The result of the division is ± the largest representable value. An OTS error message is also given for this condition. The user may test .DZERO under program control using the logical function IDZERO.

The flags .OVUDF and .DZERO can only be initialized by reloading the program, by a separate user program, or by IFLOW or IDZERO. These functions are described below.

Routine	IFLOW
Purpose	Checks underflow and overflow
Call	IORLV = IFLOW(I)
External Calls	.DA
Errors	None

The argument I indicates the check to be performed and values are returned as follows:

I	Action	Value
0	no check	0(.FALSE) flag unchanged
<0	underflow check	-1(.TRUE) if underflow - flag set to 0; else 0 (.FALSE) and flag unchanged
>0	overflow check	-1(.TRUE) if overflow - flag set to zero; else 0 (.FALSE)

Routine	IDZERO
Purpose	Checks for division by zero
Call	IORLV = IDZERO (I)
External Calls	.DA
Errors	None

If I=0, no check is made, IORLV = 0(.FALSE) and the flag is unchanged. If I≠0, a check is made. If an attempt at division by zero was made, IORLV = -1 (.TRUE) and the flag is reinitialized. Otherwise the flag is unchanged and IORLV = 0(.FALSE).

Table 3-4
Arithmetic Package*

Routine	Function	Definition	Symbolic Name	Mode	Calling Sequence ARG1	Calling Sequence ARG2	Calling Sequence	External Calls
INTEAE	Integer Arithmetic				A-Register	ARG2		None
	*Multiplication	ARG1*ARG2	.AD	I=I*I	multiplicand	multiplier	JMS*SUBR LAC ARG2	
	*Division	ARG1/ARG2	.AE	I=I/I	dividend	divisor		
	*Reverse division	ARG2/ARG1	.AF	I=I/I	divisor	dividend		
	*Subtraction	ARG1-ARG2	.AY	I=I-I	minuend	subtrahend		
	*Reverse subtraction	ARG2-ARG1	.AZ	I=I-I	subtrahend	minuend		
DOUBLE	Double-Precision Arithmetic				FL.AC	ARG2		REAL
	Load	N.A	.AO	DP=.AO(DP)		address	JMS*SUBR .DSA ARG2	
	Store	N.A	.AP	DP=.AP(DP)	value	address		
	Add	ARG1+ARG2	.AQ	DP=DP+DP	augend	addend		
	Subtract	ARG1-ARG2	.AR	DP=DP-DP	minuend	subtrahend		
	Reverse subtract	ARG2-ARG1	.AU	DP=DP-DP	subtrahend	minuend		
	Multiply	ARG1*ARG2	.AS	DP=DP*DP	multiplicand	multiplier		
	Divide	ARG1/ARG2	.AT	DP=DP/DP	dividend	divisor		
	Reverse divide	ARG2/ARG1	.AV	DP=DP/DP	divisor	dividend		

*FPP versions require only Integer Arithmetic (INTEGE).

(continued next page)

Table 3-4 (Cont)
Arithmetic Package

RELEAE

Function	Definition	Symbolic Name	Mode	Calling Sequence			External Calls
Real Arithmetic (includes floating)				ARG1 FL.AC	ARG2		
Load	N.A	.AG	R=.AG(R)		address	JMS*SUBR .DSA ARG2	
Store	N.A	.AH	R=.AH(R)	value	address		
Add	ARG1+ARG2	.AI	R=R+R	augend	addend		
Subtract	ARG1-ARG2	.AJ	R=R-R	minuend	subtrahend		
Reverse subtract	ARG2-ARG1	.AM	R=R-R	subtrahend	minuend		
Multiply	ARG1*ARG2	.AK	R=R*R	multiplicand	multiplier		
Divide	ARG1/ARG2	.AL	R=R/R	dividend	divisor		
Reverse divide	ARG2/ARG1	.AN	R=R/R	divisor	dividend		
Floating Arithmetic							
				A-Register	FL.AC		
Float	R IARG	.AW	R=.AW(I)	integer	F.P num	JMS*SUBR	
Fix	I RARG	.AX	I=.AX(R)		F.P num		
Negate	R RARG	.BA	R=.BA(R)				
				FL.AC	HELD AC		
Multiply	ARG1*ARG2	.CA	R=R*R	multiplicand	multiplier	JMS*SUBR	
Add	ARG1+ARG2	.CC	R=R+R	augend	addend		
Normalize	N.A	.CD	R=.CD(R)	value			
Hold	N.A	.CF	R=.CF(R)	value			
Sign Control	(Note 1)	.CG	R=.CG(R)	value	value		
Short get argument	N.A	.CB	R=.CB(R)	CAL0 JMS*.CB CAL0	SUBR ENTRY-EXIT STORAGE FOR ARG ADDR		

(continued next page)

Table 3-4 (Cont)
Arithmetic Package

Function	Definition	Symbolic Name	Mode	Calling Sequence			External Calls
Floating Arithmetic (Cont)				FL.AC	HELD .AC		
Divide	ARG1/ARG2	.CI	R=R/R	divisor	dividend	JMS*SUBR **	
*Round and sign	N.A	.CH	R=.CHR	value		CONST1	
						CONST2	
INT				ARG1 AC,MQ	ARG2		
Load	N.A	.JG	J=.JG(J)		address	JMS*SUBR	
Store	N.A	.JH	J=.JH(J)	value	address	.DSA ARG2	
Add	ARG1+ARG2	.JI	J=J+J	augend	addend		
Subtract	ARG1-ARG2	.JJ	J=J-J	minuend	subtrahend		
Reverse subtract	ARG2-ARG1	.JM	J=J-J	subtrahend	minuend		
Multiply	ARG1*ARG2	.JK	J=J*J	multiplicand	multiplier		
Divide	ARG1/ARG2	.JL	J=J/J	dividend	divisor		
Reverse divide	ARG2/ARG1	.JN	J=J/J	divisor	dividend		
				AC,MQ	FL.AC		
Float	R←JARG	.JW	R=.JW(J)	Doub. Int.	F.P. Number	JMS*SUBR	.CD,REAL
Fix	J←RARG	.JX	J=.JX(R)		F.P. Number		REAL
Negate	J←JARG	.JA	J=.JA(J)				

*The sign of the result (exclusive OR of the sign bits of .AB and CE02) is stored in .CE. The sign of .AB is saved in CE05.

**CONST1 and CONST2 are required for both EAE and NON-EAE operations, however, they are used only by the NON-EAE version of .CI. CONST1 indicates the number of bits to be generated (-34 for single precision, -44 for double precision). CONST2 is the least significant quotient bit (400 for single precision, 1 for double precision).

CHAPTER 4
UTILITY ROUTINES

Two types of subprogram are described in this chapter - OTS routines, automatically invoked by FORTRAN statements; and external subprograms which may be invoked via a FORTRAN CALL statement. Both types are accessible to MACRO programs.

4.1 OTS ROUTINES

OTS utility routines perform a number of functions specified by FORTRAN statements. These functions of FORTRAN, like the input-output functions discussed previously, use OTS as an interface between the user program and the monitor environment in which it will operate.

Each of these routines is described below.

.SS

Routine	.SS
Purpose	Calculates the address of an array element
Calling Sequence	.GLOBL .SS JMS* .SS .DSA ARRAY / addr wd. 4 - array descriptor block LAC (K_i) / subscript i . . . LAC (K_k) / subscript k DAC ALOC / return with element address in AC
External Calls	None
Errors	None

.SS references the array-descriptor block associated with the array whose element is to be located.

An array descriptor block is a four-word table with the contents depicted below.

Word 1	0 (bits 0-2)	Data mode (bits 3-4)	Size (in words) (… 17)
Word 2	0 - for one-dimensional array Size of first dimension		
Word 3	0 - for one- and two-dimensional arrays Size of the first two dimensions		
Word 4	Address of first word of array with mode in bits 1-2.		

Size is determined by multiplying the dimensions of the array by the number of words (N) used for a data item of the specified mode (M). Thus, an INTEGER array defined by DIMENSION (2,2,2) has the size 8 in word 1, the size 2 in word 2, and the size 4 in word 3. A REAL array of the same dimensions will have 16, 4, and 8 in these locations.

The values of M and N for the various data modes are:

Array Mode	M	N
INTEGER, LOGICAL	00	1
DOUBLE INTEGER	11	2
REAL	01	2
DOUBLE PRECISION	10	3

The address of an array element $A(K_1, K_2, K_3)$ is calculated by .SS using the following formula:

$$addr = WD4 + (K_1 - 1) * N + (K_2 - 1) * WD2 + (K_3 - 1) * WD3$$

GO TO

Routine	.GO
Purpose	Computes index of computed GO TO
Calling Sequence	LAC V / index value in A-register JMS* .GO -N / number of statement address STMT(1) STMT(2) STMT(N)
External Calls	OTSER
Errors	OTS 7 - illegal index (< 0)

STOP

Routine	.ST
Purpose	Processes STOP statement (returns to monitor)
Calling Sequence	LAC /octal number to be printed JMS* .ST
External Calls	.SP
Errors	None

PAUSE

Routine	.PA
Purpose	Processes PAUSE. Waits for ↑P and returns control to user program
Calling Sequence	LAC /octal number JMS* .PA
External Calls	.SP
Errors	None

SPMSG

Routine	.SP
Purpose	Prints octal number for PAUSE and STOP. Zero assumed if none supplied.
Calling Sequence	LAC /octal integer JMS* .SP .DSA (control return for PAUSE) LAC (first character) . . . LAC (sixth character)
External Calls	None
Errors	None

OTSER

Routine	.ER
Purpose	To print error messages on Teletype and take action according to class of error
Calling Sequence	JMS* .ER .DSA (error number)
External Calls	None
Errors	None

Recoverable errors are indicated when bit 0 of the error number is a 1. In this case, the AC and link are restored to their original contents and control is returned to the calling program at the first location following the error.

Unrecoverable errors are indicated when bit 0 of the error number is 0. Control is returned to the monitor by means of an .EXIT function. In the case of an unrecoverable error in a FORMAT statement, the current 5/7 ASCII word pair of the erroneous FORMAT is also printed. The calling sequence for .ER for a FORMAT statement differs from other calls and is:

```
        JMS*  .ER
        .DSA  12                    / error number
        LAC chars                   / current 5 characters
        LAC chars
```

PARTWD

Routine	.PB
Purpose	Part word fetch result in AC or ACMQ
Calling Sequence	JMS* .PB .DSA address
External Calls	None
Errors	None

PARTWD

Routine	.PC
Purpose	Stores contents of AC or ACMQ
Calling Sequence	JMS* .PC .DSA address
External Calls	None
Errors	None

4.2 FLOATING POINT PROCESSOR ROUTINES

General Interface Routine .FPP

Routine	.AX
Purpose	FPP version of software .AX
Routine	.AW
Purpose	FPP version of software .AW
Routine	.ZA
Purpose	Loads high order mantissa of FPP AC into the regular AC
Routine	.ZB
Purpose	Initializes FPP error handling
Routine	
Purpose	Error handling

Extended Integer (Double Integer) Interface Routines		
	Routine	.ZC
	Purpose	Converts integer in CPU AC to extended integer in FPP AC
	Routine	.ZD
	Purpose	Converts extended integer in FPP AC to single integer in CPU AC

4.3 FORTRAN - CALLABLE UTILITY ROUTINES

These routines are described in Table 4-1.

4.4 RSX LIBRARY (.LIBRX BIN) ROUTINES

A special set of routines is provided for use with the RSX-15 real-time monitor system. This library includes, in addition to the subprograms described previously, the FORTRAN-callable external subroutines given in Table 4-2. The even variable values have the following meaning:

a. Positive values signal successful completion.

b. Zero indicates a request is still pending.

c. Negative values indicate rejection or unsuccessful completion.

- -5 Illegal header word from device (data mode incorrect or data validity bits improperly set) (DVH)
- -6 Unimplemented or illegal function (DVH)
- -7 Illegal data mode (DVH)
- -10 File still open (DVH)
- -11 File not open (DVH)
- -12 DECtape error (DVH)
- -13 File not found (DVH)
- -14 Directory full (DVH)
- -15 Medium full (DVH)
- -16 Output word-pair-count or input-buffer-size error (DVH)
- -23 Input word-pair-count error (DVH)
- -24 LUN has been REASSIGNed while an ATTACH or DETACH request was in an I/O request queue (DVH)
- -101 Out of range Logical Unit Number (IO.)
- -102 Unassigned Logical Unit Number (IO.)
- -103 Non-resident Device Handler (IO.)
- -104 Control Table argument error (DVH)
- -201 Task not in system (RQ., SC,. RN., SY., DA., EA., FX,. UF., CN.)

(continued page 4-15)

Table 4-1
FORTRAN-Callable Utility Routines

Routine	ENTRY Name	Purpose	Calling Sequence	Examples	External Calls	Errors
Clock Handling - only one call may be active at any point in a user's program	TIME*	Records elapsed time in minutes and seconds on 60-cycle machine	CALL TIME(IMIN,ISEC,IOFF) Where: IMIN = minutes ISEC = seconds IOFF = non-zero to stop clock	CALL TIME(IM,IS,IOF) A . . . IOF = 1 WRITE(4,100)IM,IS [outputs time to execute A]	.DA .TIMER	None
	TIME10*	Records elapsed time in minutes, seconds, and tenths of seconds	CALL TIME10(IMIN,ISEC, ISEC10,IOFF) Where: IMIN = minutes ISEC = seconds ISEC10 = tenths of seconds IOFF = non-zero stops clock	See TIME	.DA .TIMER	None
Error Handling	ERRSET	Controls the number of run-time arithmetic errors output by OTSER	CALL ERRSET(N) Where: N = integer giving number of times message to be output before suppression. If ERRSET is not given, OTSER assumes N = 2. If N ≤ 0, no messages output.			

*Not supported with RSX. Other RSX supplied routines can be used for this purpose.

(continued next page)

Table 4-1 (Cont)
FORTRAN-Callable Utility Routines

Routine	ENTRY Name	Purpose	Calling Sequence	Examples	External Calls	Errors
Adjustable Dimensioning	ADJ1	To adjust one-dimensional array	DIMENSION B(1) CALL ADJ1(B,A) Where: B = array name A = beginning storage location of B array element (e.g., C(200) which is the beginning storage location of B) Note: The dimensions of A must be sufficient to hold all entries of array B. A may be a dummy argument in a subroutine	DIMENSION A(300),B(1),C(1) . . . CALL ADJ1 (B,A(101)) CALL ADJ1 (C,A(201)) . . . B and C may be referenced as if they had been dimensioned as (100) each	.DA	None
Adjustable Dimensioning (Cont)	ADJ2	To adjust a two-dimensional array	DIMENSION B(1,1) CALL ADJ2(B,A,NR) Where: A and B are as for ADJ1 NR = the number of rows to appear in B	DIMENSION A(300),B(1,1), C(1,1) . . . CALL ADJ2(B,A(1),10) CALL ADJ2(C,A(101),20) . . . B and C may be referenced as if they had been dimensioned (10,10) and (20,10), respectively	.DA .AD	None

(continued next page)

Table 4-1 (Cont)
FORTRAN-Callable Utility Routines

Routine	ENTRY Name	Purpose	Calling Sequence	Examples	External Calls	Errors
Adjustable Dimensioning (Cont)	ADJ3	To adjust a three-dimensional array	DIMENSION B(1,1,1) CALL ADJ3(B,A,NR,NC) Where: A,B, and NR are as for ADJ2 NC = number of columns to appear in array B	DIMENSION A(300),B(1,1), C(1,1) CALL ADJ3(B,A(1),10,5) CALL ADJ3(C,A(101),10,10) B and C may be referenced as if they had been dimensioned (2,10,5) and (2.10,10), respectively	.DA .AD	None

Table 4-2
FORTRAN-Callable RSX Routines*

Routine	Purpose	Calling Sequence	Event Variables Returned
REQUEST	Requests task execution	CALL REQST(nHTSKNAM,IP[,IEV]) Where: n = no. of characters in task name TSKNAM = name of task (1 to 5 characters) IP = task priority (1-512) may be variable or constant IEV = event variable	+1, -201, -202, -204, -777
SCHEDULE	Schedules task execu- tion	CALL SCHED(nHTSKNAM,IT,IP[,IEV]) Where: IT = name of 5-word integer array describing schedule IT(1) = schedule of hour (0-23) IT(2) = schedule of minute (0-59) IT(3) = schedule of second (0-59) IT(4) = reschedule interval (up to one day) IT(5) = reschedule units (1 = ticks, 2 = seconds, 3 = minutes, 4 = hours)	+1, -201, -203, and -777
RUN	Run task in delta time	CALL RUN(nHTSKNAM,IT,IP[,IEV]) Where: IT = name of 4-word integer array IT(1) = schedule delta time from now (up to one day) IT(2) = delta schedule units (1 = ticks, 2 = seconds, 3 = minutes, 4 = hours) IT(3) = reschedule interval (up to one day) IT(4) = reschedule units	+1, -201, -203, and -777

*Square brackets indicate that the event variable is an optional argument.

(continued next page)

Table 4-2 (Cont)
FORTRAN-Callable RSX Routines*

Routine	Purpose	Calling Sequence	Event Variables Returned
SYNC	Execute task at a specified interval	CALL SYNC(nHTSKNAM,IT,IP[,IEV]) Where: IT = name of 5-word integer array IT(1) = synchronization units (1 = ticks, ...) IT(2) = schedule interval from synchronization time (up to one day) IT(4) = reschedule interval (up to one day) IT(5) = reschedule units (1 = ticks, ...)	+1, -201, -203, and -777
CANCEL	Cancel task execution (no effect for an active task)	CALL CANCEL(nHTSKNAM[,IEV])	+1, -201, and -777
SUSPEND	Suspend execution of task issuing this call. Execution not permitted until a RESUME call	CALL SUSPEND	
RESUME	Resume task execution	CALL RESUME(nHTSKNAM[,IEV)	+1, -202, and -205
MARK	Set an event variable in delta time	CALL MARK(IT,IEV) Where: IT = name of 2-word integer array IT(1) = delta interval (up to one day) IT(2) = delta units (1 = ticks, ...)	+1, -203, and -777
WAIT FOR	Suspend task if event variable = 0; resume when non-zero	CALL WAITER(IEV)	

*Square brackets indicate that the event variable is an optional argument.

(continued next page)

Table 4-2 (Cont)
FORTRAN-Callable RSX Routines*

Routine	Purpose	Calling Sequence	Event Variables Returned
WAIT	Suspend execution of task until occurrence of next significant event	CALL WAIT	
EXIT	Terminate task execution	CALL EXIT	
DSKAL	Allocate disk storage	CALL DSKAL(ICTB,NW[,IEV]) Where: ICTB = control table (integer array returned at end of operation) ICTB(1) = amount actually allocated ICTB(2) = physical disk unit number ICTB(3) = absolute starting address of the space allocation relative to physical disk unit number NW = desired storage (in words)	+1, -6, -15, -101, -104, and -777
DSKDAL	Deallocate disk storage	CALL DSKDAL(ICTB[,IEV]) Where: ICTB = control table (same address as used in the corresponding DSKAL)	+1, -6, -15, -101, -104, and -777
DSKPUT	Put data on disk	CALL DSKPUT(ICTA,IOA,NW,ARRAY[,IEV]) Where: ICTA = device control table (same as for corresponding DSKAL) IOA = disk offset address NW = number of words (decimal) to transfer ARRAY = name of array containing data to be transferred	+1 and -N Where: N = the contents of the disk status register on error

*Square brackets indicate that the event variable is an optional argument.

(continued next page)

Table 4-2 (Cont)
FORTRAN-Callable RSX Routines*

Routine	Purpose	Calling Sequence	Event Variables Returned
DSKGET	Get data from disk	CALL DSKGET(ICTA,IOA,NW,ARRAY[,IEV])	+1 and -N
ATTACH	Attach I/O Handler task	CALL ATTACH(LUN[,IEV]) Where: LUN = logical unit number	+1, -6, -24, -101, -103, and -777
DETACH	Detach I/O Handler task	CALL DETACH(LUN[,IEV]	+1, -6, -101, -103, and -777
SEEK	Seek open file for input	CALL SEEK(LUN,nHFLNAM,nHEXT[,IEV]) Where: LUN = logical unit number n = number of characters in file name or extension FLNAM = 1-5 character file name EXT = 1-3 character extension	+1, -6, -10, -12, -13, -101, -102, -103, and -777
ENTER	Open file for output	CALL ENTER(LUN,nHFLNAM,nHEXT[,IEV])	+1. -6, -11, -12, -14, -101, -102, -103, and -777
CLOSE	Closes file	CALL CLOSE(LUN,nHFLNAM,nHEXT[,IEV])	+1, -6, -11, -12, -13, -14, -101, -102, -103
HINF	Provides information about the physical device and the I/O Handler associated with a particular Logical Unit Number (LUN)	CALL HINF(LUN,IEV)	Single word containing the following Handler information: Bit 0 - unused Bit 2 - input - set to 1 if data can be input Bit 2 - output - set to 1 if data can be output Bit 3 - file-oriented - set to 1 if file-oriented (SEEK and ENTER have been used)

*Square brackets indicate that the event variable is an optional argument.

(continued next page)

Table 4-2 (Cont)
FORTRAN-Callable RSX Routines*

Routine	Purpose	Calling Sequence	Event Variables Returned
HINF(Cont)			Bits 4-11 - unit number Bits 12-17 - device code (1 to 63 decimal devices). Codes below are fixed for standard devices 1 - TTY (console, LT15, LT19) 2 - DK - RF15 fixed-head DECdisk 3 - DP - RP02 disk pack 4 - DT - TC02D DECtape 5 - MT - TC59 MAGtape 6 - PR - PC15 paper-tape reader 7 - CD - CR03B card reader 10 - PP - PC15 paper-tape punch 11 - LP - LP15 line printer 12 - VP - VP15 storage scope 13 - VT - VT15 display Users should assign codes to their own devices starting at 63 and working back
DISABLE	Disable task	CALL DISABL(nHTSKNAM[,IEV])	+1, -201, -210
ENABLE	Enable task	CALL ENABLE(nHTSKNAM[,IEV])	+1, -201, -210
FIX	Fix task in core	CALL FIX(nHTSKNAM[,IEV])	+1, -201, -207
UNFIX	Unfix task in core	CALL UNFIX(nHTSKNAM[,IEV])	+1, -201, -207
DECLAR	Declares a significant event	CALL DECLAR	
TIME	Obtain time from Executive	CALL TIME(ITIME) Where: ITIME = 3-word integer array ITIME(1) = hours (0-23) ITIME(2) = minutes (0-59) ITIME(3) = seconds (0-59)	

*Square brackets indicate that the event variable is an optical argument.

(continued next page)

Table 4-2 (Cont)
FORTRAN-Callable RSX Routine*

Routine	Purpose	Calling Sequence	Event Variables Returned
DATE	Obtain time and date from Executive	CALL DATE(IDATE) Where: IDATE = 6-word integer array IDATE(1) = month (1-12) IDATE(2) = day (1-31) IDATE(3) = year (0-99) IDATE(4) = hours (0-23) IDATE(5) = minutes (0-59) IDATE(6) = seconds (0-59)	

*Square brackets indicate that the event variable is an optical argument.

-202 Task is active (RQ., FX.) or not active (RS.)

-203 CAL not Task issued (SC., RN., SY., MT.)

-204 Task is DISABLED (RQ., SC., RN., SY., FX.)

-205 Task not suspended (RS.)

-207 Task already FIXed (FX.) or not FIXed (UP.)

-210 Partition occupied (FX.)

-301 Line number rejected (CI., DI.)

-302 Line is CONNECTed (CI.) or DI CONNECTed (DI.)

-777 Pool is empty

DVH - Device Handler

IO. - 'QUEUE I/O' Directive

RQ. - 'REQUEST' Directive

SC. - 'SCHEDULE' Directive

RN. - 'RUN' Directive

SY. - 'SYNC' Directive

CN. - 'CANCEL' Directive

RS. - 'RESUME' Directive

CI. - 'CONNECT' Directive

DI. - 'DISCONNECT' Directive

FX. - 'FIX IN CORE' Directive

UF. - 'UNFIX' Directive

DA. - 'DISABLE' Directive

EA. - 'ENABLE' Directive

MT. - 'MARK' Directive

OTS routines which have been modified for RSX are:

FIOPS - modified to use the RSX I/O CAL'S..FP, which initializes the I/O status table has been converted to a dummy subroutine.

If a Negative Event Variable occurs as a result of a FIOPS issued I/O request, an error message (OTS 20) is issued and the task is EXITed.

SPMSG - rewritten to include the task name. The message is output to LUN 4 in the following format:

STOP - 000000 - TSKNAM

STOP - uses RSXEXIT CAL

PAUSE - SUSPENDs the issuing task. To continue, the RESUME MCR function is used.

OTSER - passes its name and an octal OTS error message number to SPMSG.

Additional routine used by RSX for bank/page mode determination is .BP.

Two additional OTS routines are given below:

.ASCII to .SIXBT Conversion

Routine	.FTSB	
Purpose	To convert two words from .ASCII to .SIXBT	
Calling Sequence:		

```
SUBA     0
         JMS*  .DAA    / get call args
         JMP ARGEND
FROM     0             / PTR to ASCII word-pair
ARGEND   JMS*  .FTSB
         .DSA FROM
         .DSA TO
           .
           .
           .
TO       BLOCK 2       / two 6-bit words
```

.DAA is a routine which performs the argument list transfer function formerly performed by .DA. The calling sequence has not been changed, but the transfer stops with the end of the shortest argument.

CHAPTER 5
FORTRAN-IV AND MACRO

In previous chapters, MACRO calling sequences have been given for OTS and Science Library Subprograms. This general form is used in a MACRO program to call any FORTRAN external subroutine or function. A FORTRAN program may also invoke MACRO subprograms. The method for each type of linkage is given below.

5.1 INVOKING MACRO SUBPROGRAMS FROM FORTRAN

A FORTRAN program may invoke any MACRO program whose name is declared in a MACRO .GLOBL statement. The MACRO subprogram must also include the same number of open registers as there are arguments. These will serve as transfer vectors for arguments supplied in the FORTRAN CALL statement or function reference. A FORTRAN-IV program and the MACRO subprogram it invokes are shown below. More extensive examples are given in Appendix C.

FORTRAN	MACRO
	.TITLE MIN
C TEST MACRO SUBR	.GLOBL MIN, .DA
	MIN 0 / entry/exit
C READ A NUMBER(A)	JMS* .DA / general get
	/ argument
1 READ(1,100)A	/ (OTS)
	JMP .+2+1 / jump around
100 FORMAT(E12.4)	argument
	registers
C NEGATE THE NUMBER	
C AND PUT IT IN B	MIN1 .DSA 0 / ARG1
	MIN2 .DSA 0 / ARG2
CALL MIN(A,B)	LAC* MIN1 / first word of A
	DAC* MIN2 / store at B
C WRITE OUT NUMBER(B)	ISZ MIN1 / point to second word
	ISZ MIN2 / of A and B
WRITE(2,100)B	LAC* MIN1 / second word of A
	TAD (400000) / sign bit = 1
STOP	DAC* MIN2 / store in second
	/ word of B
END	JMP* MIN / exit
	.END

The FORTRAN statement CALL MIN(A,B) is expanded by the compiler to:

```
        00013   JMS* MIN          / to MACRO subprog
        00014   JMP$ 00014
        00015   .DSA A
        00016   .DSA B
      $00014 =  00017
```

When the FORTRAN-IV program is loaded, the addresses (plus relocation factor) of A and B are stored in registers 15 and 16, respectively. When the MACRO program invokes .DA, these addresses are stored in MIN1 and MIN2 and the values themselves are accessed by indirect reference.

Arguments are, as described above, transmitted by .DA using a single word. Bits 3-17 contain the 15-bit address of the first word. Bits 0-2 serve as flag. FORTRAN uses bit 0 to indicate that the word specifying the argument contains the address of a word containing the address of the first word of the argument. The MACRO argument word always contains the address of the first word of the argument. For array name arguments (unsubscripted), the address of the fourth word of the array descriptor block is given. .SS must be invoked to locate the element.

For external functions, the MACRO subprogram must return with a value in the AC (LOGICAL, INTEGER), AC-MQ (DOUBLE INTEGER) or in the floating accumulator (REAL or DOUBLE PRECISION).

5.2 INVOKING FORTRAN SUBPROGRAMS FROM MACRO

The MACRO calling conventions for FORTRAN subprograms are: the name of the subprogram must be declared as global; there must be a jump around the argument address; and the number and mode of arguments in the call must agree with those of the subprogram. This form is shown below.

```
TITLE
.GLOBL  SUBR
JMS*    SUBR
JMP     .+N+1          / jump around arguments ignored by .DA
.DSA    ARG1           / address of first argument - bit 0 set to 1
.DSA    ARG2           / indicates indirect reference
   .
   .
   .
.DSA    ARGN
   .
   .
   .
```

When the subprogram is compiled, a call is generated to .DA which performs the transmission of arguments from MACRO. The beginning of a subroutine might be expanded as follows.

```
        C                        TITLE SUBR
                                 SUBROUTINE SUBR(A,B)
      000000                     CAL 0
      000001                     JMS* .DA
      000002                     JMP $000002
      000003                     .DSA A
      000004                     .DSA B
    $ 000002 = 000005
```

If a value is to be returned by the subroutine, it is most convenient to have this be one of the calling arguments. An external function is called in the same manner as a subroutine but returns a value in the AC (single integers), AC-MQ (double integers), or floating accumulator (real and double-precision). To store the AC, the MACRO program uses a DAC instruction. Values from the floating accumulator may be stored via the OTS routines .AH (real) and .AP (double-precision). For FPP systems, values are returned in a hardware accumulator and stored with an FST instruction.

A number of examples of MACRO-FORTRAN linkage are given in Appendix C.

5.3 COMMON BLOCKS

FORTRAN COMMON blocks (and block-data subprograms) may be linked to MACRO programs. When the MACRO program is loaded, global symbols are first sought in the user and system libraries. Any remaining are matched, where possible, to COMMON block names. For example:

FORTRAN	MACRO	
INTEGER A,B,C COMMON/NAME/C COMMON A,B . . .	.GLOBL NAME, .XX DZM* .XX ISZ .XX DZM* .XX DZM* NAME	/ .XX is name given to blank COMMON / by the F4 Compiler / CLEAR A - NOTE INDIRECT REFERENCE / BUMP COUNTER / CLEAR B / CLEAR C

Note that if the values are REAL (two words) or DOUBLE PRECISION (three words), the MACRO program must account for the number of words when accessing specific variables. This cannot be done if programs are loaded via CHAIN and EXECUTE.

APPENDIX A
LANGUAGE SUMMARY

Statement	Model	Effect	Text Reference
Arithmetic	var = value array (i) = value	value is assigned to var or array (i)	2.1
ASSIGN	ASSIGN n TO label	Statement n is assigned the symbol name label	2.2
BLOCK DATA	BLOCK DATA	Identifies subprogram which enters data into COMMON block at run time	4.4
CALL	CALL subr(a_1,a_2,...a_n) CALL subr	Control is transferred to the subroutine; a_1,a_2,...a_n are substituted for dummy variables	5.2.2
COMMON	COMMON/ b_1 /$vlist_1$ /b_2/ $vlist_2$/...	vlist items are allocated to b blocks where they are shared by other programs	4.2.2
CONTINUE	CONTINUE	Dummy statement used to prevent illegal termination of DO loops	3.2.3
DATA	DATA $vlist_1$ /$clist_1$ /,$vlist_2$ / $clist_2$ /,...$vlist_n$ /$clist_n$	clist is assigned to its corresponding vlist	4.3
DECODE	DECODE(c,v,f,ERR=n) list	Converts character data stored in the array (v) into binary and assigns them to variables in list	6.3.4
DIMENSION	DIMENSION a_1(l_1),a_2(l_2),... a_n(l_n)	Storage is allocated for array (a) to the dimensions specified by the subscript list (l)	4.2.1
DO	DO n i=m_1,m_2,m_3 DO n i=m_1,m_2 DO n i=m_1,m_2,-m_3	Statements following the DO are executed repeatedly for values m_1 through m_2 in increments or decrements of m_3	3.2

Statement	Model	Effect	Text Reference
ENCODE	ENCODE(c,v,f,ERR=n)list	Converts binary data represented by variables in list into characters according to FORMAT specification (f) or data-directed I/O rules and stores them in the array (v)	6.3.4
EQUIVALENCE	EQUIVALENCE(l_1),(l_2),... (l_n)	Elements of each list (l) are assigned to the same storage location	4.2.3
EXTERNAL	EXTERNAL $a_1,a_2,...a_n$	Defines subprograms named a for use as arguments of other subprograms	4.1.3
FORMAT	n FORMAT$(s_1,s_2,...s_n)$	FORMAT statement n established as field-specification reference	6.1
FUNCTION	m FUNCTION $f(a_1,a_2,...a_n)$	Defines FUNCTION named f with dummy arguments a and optional mode specification m	5.1.2
GO TO	GO TO n	Control is unconditionally transferred to statement n	3.1.1
	GO TO$(n_1,n_2,...n_k)$,i	Control is transferred to the i^{th} statement in the list of n's	3.1.2
	GO TO label GO TO label,$(n_1,n_2,...n_k)$	Control is transferred to the location specified by label; the list of n's may specify legally ASSIGNable statement numbers	3.1.3

IF	IF(expr)n_1,n_2,n_3
	
	IF(expr)s
		Control is transferred to statement number or ASSIGNed label n_1, n_2, or n_3 if evaluated expr is <0, $=0$, or >0 respectively
		Statement s is executed if expr is .TRUE. (non-zero), ignored if .FALSE. (zero)
			3.3.1
			
			3.3.2

IMPLICIT	IMPLICIT $m_1(l_1),m_2(l_2),...$ $m_n(l_n)$	Declares mode (m) for variables beginning with alphabetic characters in list (l)	4.1.2
PAUSE	PAUSE PAUSE n	Interrupts program execution; if present, integer n is printed on the console to distinguish one PAUSE from another	3.4.1

Statement	Model	Effect	Text Reference
PRINT	PRINT(d,f)list	The values of variables in list are converted to ASCII according to FORMAT reference (f) and transferred to external device (d)	6.3.2
	PRINT(d)list	The values of variables in list are written in binary on external device (d)	6.3.2
	PRINT(d,)list	The variable names in list are written on external device (d), each followed by its value in the form 'A' = value	6.3.2
	PRINT(d,f)	FORMAT reference (f) is written on external device (d)	6.3.2
READ	READ(d,f)list	The values represented by variables in list are read from external device (d) and converted according to FORMAT reference (f)	6.3.2
	READ(d)list	The binary values represented by variables in list are read from external device (d)	6.3.2
	READ(d,)list	The values represented by variables in list are read from external device (d)	6.3.2
	READ(d,f)	Values are read into FORMAT reference (f)	6.3.2
	READ(d)	A binary record is read from external device (d) and ignored	6.3.2
STOP	STOP STOP n	Signifies the logical end of a program and returns control to the MONITOR after n is printed; if present, n distinguishes one STOP from another	3.4.2
SUBROUTINE	SUBROUTINE name $(a_1,a_2,\ldots a_n)$ SUBROUTINE name	Defines an external subroutine named name; a's are dummy arguments representing values supplied by the calling program or returned by the subroutine	5.2.1

Statement	Model	Effect	Text Reference
TYPE	TYPE(d,f)list	The values of variables in list are converted to ASCII according to FORMAT reference (f) and transferred to external device (d)	6.3.2
	TYPE(d)list	The values of variables in list are written in binary on external device (d)	6.3.2
	TYPE(d,)list	The variable names in list are written on external device (d), each followed by its value in the form 'A' = value	6.3.2
	TYPE(d,f)	FORMAT reference (f) is written on external device (d)	6.3.2
WRITE	WRITE(d,f)list	The values of variables in list are converted to ASCII according to FORMAT reference (f) and transferred to external device (d)	6.3.2
	WRITE(d)list	The values of variables in list are written in binary on external device (d)	6.3.2
	WRITE(d,)list	The variable names in list are written on external device (d), each followed by its value in the form 'A' = value	6.3.2
	WRITE(d,f)	FORMAT reference (f) is written on external device (d)	6.3.2

APPENDIX B
ERROR MESSAGES

B.1 COMPILER ERROR MESSAGES

In the F4X version of FORTRAN, compiler error messages are printed in the form:

>mnA<

where:

mn is the error number
A is the alphabetic mnemonic

characterizing the error class.

In F4I and F4A versions, only the alphabetic character is printed, in the form:

>A<

All error messages and the version(s) of FORTRAN to which they are applicable are given below.

Number	Letter	Meaning
		Common, equivalence, data errors:
01	C	No open parenthesis after variable name in DIMENSION statement
02	C	No slash after common block name
03	C	Common block name previously defined
04	C	Variable appears twice in COMMON
05	C	EQUIVALENCE list does not begin with open parenthesis
06	C	Only one variable in EQUIVALENCE class
07	C	EQUIVALENCE distorts COMMON
08	C	EQUIVALENCE extends COMMON down
09	C	Inconsistent EQUIVALENCing
10	C	EQUIVALENCE extends COMMON down
11	C	Illegal delimiter in EQUIVALENCE list

(continued on next page)

Number	Letter	Meaning
		Common, equivalence, data errors: (cont)
12	C	Non-COMMON variables in BLOCK DATA
15	C	Illegal repeat factor in DATA statement
16	C	DATA statement stores in COMMON in non-BLOCK DATA statement or in non-COMMON in BLOCK DATA statement
		DO errors:
01	D	Statement with unparenthesized = sign and comma not a DO statement
04	D	DO variable not followed by = sign
05	D	DO variable not integer
06	D	Initial value of DO variable not followed by comma
07	D	Improper delimiter in DO statement
09	D	Illegal terminating statement for DO loop
		External symbol and entry-point errors:
01	E	Variable in EXTERNAL statement not simple non-COMMON variable
02	E	ENTRY name non-unique
03	E	ENTRY statement in main program
04	E	No = sign following argument list in arithmetic statement function
05	E	No argument list in FUNCTION subprogram
06	E	Subroutine list in CALL statement already defined as variable
08	E	Function or array name used in expression without open parenthesis
09	E	Function or array name used in expression without open parenthesis
		Format errors:
01	F	Bad delimiter after FORMAT number in I/O statement
02	F	Missing field width, illegal character or unwanted repeat factor
03	F	Field width is 0
04	F	Period expected, not found
05	F	Period found, not expected
06	F	Decimal length missing (no "d" in "Fw.d")
07	F	Unparenthesized comma

(continued on next page)

Number	Letter	Meaning
		Format errors: (cont)
08	F	Minus without number
09	F	No P after negative number
10	F	No number before P
12	F	No number or 0 before H
13	F	No number or 0 before X
15	F	Too many left parentheses
		Hollerith errors:
03	H	Number preceding H not between 1 and 5
04	H	Carriage return inside Hollerith field
05	H	Number preceding H not an integer
06	H	More than five characters inside quotes
07	H	Carriage return inside quotes
		Various illegal errors:
01	I	Unidentifiable statement
02	I	Misspelled statement
03	I	Statement out of order
04	I	Executable statement in BLOCK DATA subroutine
05	I	Illegal character in I/O statement, following unit number
06	I	Illegal delimiter in ASSIGN statement
07	I	Illegal delimiter in ASSIGN statement
08	I	Illegal type in IMPLICIT statement
09	I	Logical IF as target of logical IF
10	I	RETURN statement in main program
11	I	Semicolon in COMMON statement outside of BLOCK DATA
12	I	Illegal delimiter in IMPLICIT statement
13	I	Misspelled REAL or READ statement
14	I	Misspelled END or ENDFILE statement
15	I	Misspelled ENDFILE statement
16	I	Statement function out of order or undimensioned array
17	I	Typed FUNCTION statement out of order
18	I	Illegal character in context
19	I	Illegal logical or relational operator

(continued on next page)

Number	Letter	Meaning
		Various illegal errors: (cont)
20	I	Illegal letter in IMPLICIT statement
21	I	Illegal letter range in IMPLICIT statement
22	I	Illegal delimiter in letter section of IMPLICIT statement
23	I	Illegal character in context
24	I	Illegal comma in GOTO statement
26	I	Illegal variable used in multiple RETURN statement
		Pushdown list errors:
01	L	DO nesting too deep
02	L	Illegal DO nesting
03	L	Subscript/function nesting too deep
04	L	Backwards DO loop (also caused by some illegal I/O lists). Appears after END statement.
		Overflow errors:
01	M	EQUIVALENCE class list full
02	M	Program size exceeds 8K
03	M	Array length larger than 8K
04	M	Element position in array larger than 8K (EQUIVALENCE, DATA)
06	M	Integer negative or larger than 131071
07	M	Exponent of floating point number larger than 76
08	M	Overflow accumulating constant - too many digits
09	M	Overflow accumulating constant - too many digits
10	M	Overflow accumulating constant - too many digits
		Statement number errors:
01	N	Multiply defined statement number or compiler error
02	N	Statement erroneously labeled
03	N	Undefined statement number
04	N	FORMAT statement without statement number
05	N	Statement number expected, not found
07	N	Statement number more than five digits
08	N	Illegal statement number

(continued on next page)

Number	Letter	Meaning
		Partword errors:
01	P	Expected colon, found none
02	P	Expected close bracket, found none
03	P	Last bit number larger than 35
04	P	First bit number larger than last bit number
05	P	First and last bit numbers not simple integer constants
		Subscripting errors:
01	S	Illegal subscript delimiter in specification statements
02	S	More than three subscripts specified
03	S	Illegal delimiter in subroutine argument list
04	S	Non-integer subscript
05	S	Non-scalar subscript
06	S	Integer scalar expected, not found
10	S	Two operators in a row
11	S	Close parenthesis following an operator
12	S	Non-integer subscript
13	S	Non-scalar subscript
14	S	Two arguments in a row
15	S	Digit or letter encountered after argument conversion
16	S	Number of subscripts stated not equal to number declared
		Table overflow errors:
01	T	Arithmetic statement, computed GOTO list, or DATA statement list too large
02	T	Too many dummy variables in arithmetic statement function
03	T	Symbol and constant tables overlap
		Variable errors:
01	V	Two modes specified for same variable name
02	V	Variable expected, not found
03	V	Constant expected, not found
03	V	Array defined twice
05	V	Error: variable is EXTERNAL or argument (EQUIVALENCE, DATA)
07	V	More than one dimension indicated for scalar variable

(continued on next page)

Number	Letter	Meaning
		Variable errors: (cont)
08	V	First character after READ or WRITE not open parenthesis in I/O statement
09	V	Illegal constant in DATA statement
11	V	Variables outnumber constants in DATA statement
12	V	Constants outnumber variables in DATA statement
14	V	Illegal dummy variable (previously used as non-dummy variable)
16	V	Logical operator has non-integer, non-logical arguments
17	V	Illegal mixed mode expression
19	V	Logical operator has non-integer, non-logical arguments
21	V	Signed variable left of equal sign
22	V	Illegal combination for exponentiation
25	V	.NOT. operator has non-integer, non-logical argument
27	V	Function in specification statement
28	V	Two exponents in one constant
29	V	Illegal redefinition of a scalar as a function
30	V	No number after E or D in a constant
32	V	Non-integer record number in random access I/O
35	V	Illegal delimiter in I/O statement
36	V	Illegal syntax in READ, WRITE, ENCODE, or DECODE statement
37	V	END and ERR exists out of order in I/O statement
38	V	Constant and variable modes don't match in DATA statement
39	V	ENCODE or DECODE not followed by open parenthesis
40	V	Illegal delimiter in ENCODE/DECODE statement
41	V	Array expected as first argument of ENCODE/DECODE statement
42	V	Illegal delimiter in ENCODE/DECODE statement
		Expression errors:
01	X	Carriage return expected, not found
02	X	Binary WRITE statement with no I/O list
03	X	Illegal element in I/O list
04	X	Illegal statement number list in computed or assigned GOTO
05	X	Illegal delimiter in computed GOTO
07	X	Illegal computed GOTO statement

(continued on next page)

Number	Letter	Meaning
		Expression errors: (cont)
10	X	Illegal delimiter in DATA statement
11	X	No close parenthesis in IF statement
12	X	Illegal delimiter in arithmetic IF statement
13	X	Illegal delimiter in arithmetic IF statement
14	X	Expression on left of equals sign in arithmetic statement
15	X	Too many right parentheses
16	X	Illegal open parenthesis (in specification statements)
17	X	Illegal open parenthesis
19	X	Too many right parentheses
20	X	Illegal alphabetic in numeric constant
21	X	Symbol contains more than six characters
22	X	.TRUE., .FALSE., or .NOT. preceded by an argument
23	X	Unparenthesized comma in arithmetic expression
24	X	Unary minus in I/O list
26	X	Illegal delimiter in I/O list
27	X	Unterminated implied - DO loop in I/O list
28	X	Illegal equals sign in I/O list
29	X	Illegal partword operator
30	X	Illegal arithmetic expression

B.2 OTS ERROR MESSAGES

Following is a list of OTS error messages. (R) indicates a recoverable error; (T) a terminal error.

Error Number	Error Description	Possible Source
05 (R)	Negative REAL square root argument	SQRT
06 (R)	Negative DOUBLE PRECISION square root argument	DSQRT
07 (R)	Illegal index in computed GO TO	.GO
10 (T)	Illegal I/O device number	.FR, .FW, .FS, .FX, DEFINE, RANCOM
11 (T)	Bad input data - IOPS mode incorrect	.FR, .FA, .FE, .FF, .FS, RANCOM, RBINIO, RBCDIO

(continued on next page)

Error Number	Error Description	Possible Source
12 (T)	Bad FORMAT	.FA, .FE, .FF
13 (T)	Negative or zero REAL logarithmic argument (terminal)	.BC, .BE, ALOG
14 (R)	Negative or zero DOUBLE PRECISION logarithmic argument	.BD, .BF, .BG, .BH, DLOG, DLOG10
15 (R)	Zero raised to a zero or negative power (zero result is passed)	.BB, .BC, .BD, .BE, .BF, .BG, .BH
20 (T)	Fatal I/O error (RSX only)	FIOPS
direct access errors: 21 (T)	Undefined file	RANCOM
22 (T)	Illegal record size	DEFINE
23 (T)	Size discrepancy	RANCOM
24 (T)	Illegal record number	DEFINE, RANCOM
25 (T)	Mode discrepancy	RANCOM
26 (T)	Too many open files	DEFINE
30 (R)	Single integer overflow*	RELEAE, .FPP
**31 (R)	Extended (double) integer overflow****	DBLINT, JFIX, JDFIX, ISNGL
**32 (R)	Single flt. overflow	RELEAE
**33 (R)	Double flt. overflow†	
**34 (R)	Single flt. underflow	RELEAE
**35 (R)	Double flt. underflow†	
**36 (R)	Flt. divide check	RELEAE
***37 (R)	Integer divide check	INTEAE
40 (T)	Illegal number of characters specified [legal: o<c≤625]	ENCODE
41 (R)	Array exceeded	ENCODE
42 (T)	Bad input data	DD10
**50 (T)	FPP memory protect/non-existent memory	
51 (T)	(READ to WRITE Illegal I/O Direction Change to Disk) without intervening CLOSE or REWIND	BCDIO, BINIO

*Only detected when fixing a floating point number.
**Also prints out PC with FPP system
***If extended integer divide check, prints out PC with FPP system.
****With software F4 system only detected when fixing a floating point number.
†Not detected by software system (only by FPP system).

B.3 OTS ERROR MESSAGES IN FPP SYSTEMS

In software systems, arithmetic errors resulting in the OTS error messages summarized above are detected in the arithmetic package (RELEAE and INTEAE). In the hardware FPP systems, these errors are detected by the hardware (with the exception of single integer divide check) and serviced by a trap routine in the FPP routine .FPP.

Where applicable, on such error conditions, the result is patched for both software and hardware systems as summarized in the following table.

Error	PATCHED VALUE***	
	FPP Hardware System	Software System
Single Floating Overflow (.OTS 32)	± largest single floating value	same
Double Floating Overflow (.OTS 33)	± largest single floating value	not detected
Single Floating Underflow (.OTS 34)	zero	same
Double Floating Underflow (.OTS 35)	zero	not detected
Floating Divide Check (.OTS 36)	± largest single floating value	same
Integer Overflow (.OTS 30)	limited detection*	same
Double Integer Overflow (.OTS 31)	none**	limited detection*
Integer Divide Check (.OTS 37)	none	same

*When fixing a floating point number, integer and extended integer overflow is detected. In these instances, plus or minus the largest integer for the data mode is patched as result.

**With the FPP system all extended integer overflow conditions are detected, but the results are meaningless.

***Where "none" is specified, the result is meaningless unless otherwise indicated.

Further, when converting an extended integer, the magnitude of which is $>2^{17}-1$, to a single integer, no error is indicated and the high order digits are lost.

APPENDIX C
PROGRAMMING EXAMPLES

C.1 MACRO-FORTRAN Linkages

Example 1. A New Dimension Adjustment Routine

The present versions of the OTS routines ADJ1, ADJ2, and ADJ3 do not alter the size of the array being adjusted. If only the array name of an adjusted array is given in a READ or WRITE argument list, FORTRAN uses this size information; therefore, undesired results can occur. A new routine (ADJ) can be loaded with a user program which completely handles all cases of dimension adjustment, although it occupies 72 octal locations. (ADJ3 occupies 41 octal locations.) Consider the following programs:

```
C     PROGRAM 1
       DIMENSION A(4,3,2)
        .
        .
        .
C     MAKE ARRAY A ACT LIKE IT
C     WAS DIMENSIONED A (2,3,4)
       CALL ADJ(A,A(1,1,1),2,3,4)

C     PROGRAM 2
       DIMENSION A(3,2)
        .
        .
        .
C     ADJUST ARRAY A TO BE A (2,3)
       CALL ADJ (A,A(1,1),2,3,0)
C     THE LAST ARGUMENT MUST BE 0
        .
        .
        .

C     PROGRAM 3
       DIMENSION A(2)
        .
        .
        .
C     ADJUST ARRAY A TO BE A(1)
       CALL ADJ(A,A(1),1,0,0)
C     THE LAST 2 ARGUMENTS MUST BE ZERO
C     THE NO. OF SUBSCRIPTS IS NOT ADJUSTABLE
```

(continued on next page)

```
     .TITLE ADJ
/
/SUBROUTINE TO PERFORM DIMENSION ADJUSTMENT
/
/MACRO-15 CALLING SEQUENCE
          .GLOBL ADJ
/         JMS* ADJ
/         JMP .+6
/         .DSA ARRAY           /ADDRESS OF WD4
/         .DSA B     /NEW WD4
/         .DSA K1    /ADDRESS OF NEW MAXIMUM 1ST SUBSCRIPT
/         .DSA K2    /ADDRESS OF NEW MAXIMUM 2ND SUBSCRIPT
/         .DSA K3    /ADDRESS OF NEW MAXIMUM 3RD SUBSCRIPT
/
          .GLOBL ADJ,.DA,.AD
ADJ       0
          JMS* .DA   /GET ARGUMENTS
          JMP .+5+1  /# OF ARGUMENTS = 5
ARRAY     0
B         0
K1        0
K2        0
K3        0
          LAC (LAC* B          /INITIALIZE SUBSCRIPT POINTER
          DAC C
          LAC B      /SET NEW STARTING ADDRESS
          DAC* ARRAY
          LAW -3
          DAC CTR#   /MAXIMUM OF 3 SUBSCRIPTS
          TAD ARRAY
          DAC ARRAY /POINT TO FIRST WORD
          DAC ARRAYP#          /OF ARRAY DESCRIPTOR BLOCK
          LAC* ARRAY           /ARRAY TYPE IN BITS 3-4
          AND (60000           /ZERO OUT ARRAY SIZE
          DAC* ARRAY           /SAVE CLEAN ARRAY TYPE
          RTL
          RTL
          RTL
          TAD (1     /ADD 1 FOR # OF WORDS
          AND (3     /AND TREAT DOUBLE INTEGER
          SNA        /AS 2 WORD PER ARRAY ELEMENT
          LAC (2
LOOP      ISZ C      /POINT TO NEXT SUBSCRIPT
          JMS* .AD   /MULTIPLY INTEGERS
C         LAC* K1    /PROGRAM MODIFIED
          SNA        /IS SUBSCRIPT PRESENT
          JMP D      /RAN OUT OF SUBSCRIPTS
          DAC SIZE#  /UPDATE SIZE
          ISZ CTR    /ARE WE FINISHED?
          SKP
          JMP E      /YES

          ISZ ARRAYP           /STORE INTO ARRAY
          DAC* ARRAYP          /DESCRIPTOR BLOCK
          JMP LOOP  /OFFSET WORDS (2,3)
D         DZM* ARRAYP          /ZERO THE REST
          ISZ ARRAYP           /OF THE OFFSET WORDS
```

(continued on next page)

```
                ISZ CTR    /ARE WE FINISHED
                JMP LOOP   /NO
E               LAC SIZE   /FINISHED
                AND (17777           /PACK SIZE
                XOR* ARRAY           /ARRAY DESCRIPTOR BLOCK
                DAC* ARRAY
                JMP* ADJ   /RETURN
                .END
```

Example 2. A Function to Read the AC Switches

It is very often desirable to use the AC switches to alter the sequence of instructions executed in a FORTRAN program. The following program can be used as a function in an arithmetic IF statement to conditionally branch.

```
        .TITLE ITOG
/
/SUBROUTINE TO READ AC SWITCHES
/
/MACRO-15 CALLING SEQUENCE
/               .GLOBL ITOG
/               JMS* ITOG
/               JMP .+2    /JUMP OVER ARGUMENT
/               .DSA (MASK                  /ADDRESS OF MASK
/                                           /RETURN WITH MASKED ACS IN AC
                .GLOBL ITOG,.DA
ITOG            Ø          /INTEGER FUNCTION
                JMS* .DA   /GET ARGUMENTS
                JMP .+1+1  /1 ARGUMENT
MASK            Ø          /MASK ADDRESS
                LAS        /LOAD AC FROM SWITCHES
                AND* MASK  /MASK AC
                JMP* ITOG  /RETURN WITH MASKED AC SWITCHES
                .END
```

Example 3. A Routine to Read an Array in Octal

A MACRO subroutine which reads octal information (REDAR) is as follows:

```
        .TITLE REDAR
/
/SUBROUTINE TO READ ARRAY IN OCTAL
/
/MACRO-15 CALLING SEQUENCE
/               .GLOBL REDAR
/               JMS* REDAR
/               JMP .+5
/               .DSA SLOT /ADDRESS OF SLOT #
/               .DSA FORMAT          /ADDRESS OF FORMAT STATEMENT ADDRESS
/               .DSA DIGITS          /ADDRESS # OF DIGITS
/               .DSA ARRAY           /ADDRESS OF ARRAY DESCRIPTOR
/                                    /BLOCK WORD 4
/
```

(continued on next page)

```
        .GLOBL REDAR,.DA,.FR,.FE,.FF
REDAR   0
        JMS* .DA   /GET ARGUMENTS
        JMP .+4+1  /#ARGUMENTS = 4
SLOT    0
FORMAT  0
DIGITS  0
ARRAY   0
        LAC SLOT
        DAC A
        LAC* FORMAT
        DAC B
        JMS* .FR   /FORMATTED WRITE
A       XX         /ADDRESS DAT SLOT #
B       XX         /ADDRESS OF FORMAT STATEMENT
        LAW -3
        TAD ARRAY
        DAC SLOT   /ADDRESS OF ARRAY DESCRIPTOR BLOCK WORD 1
        LAC* SLOT  /PICK UP PACKED SIZE OF ARRAY
        AND (17777            /CLEAN OFF MODE #
        SNA
        JMP E      /NO ELEMENTS IN ARRAY
        CMA
        DAC SLOT
        ISZ SLOT   /COUNTER FOR # WORDS IN ARRAY
        LAC* DIGITS           /#DIGITS IN EACH WORD
        AND (7     /CLEAN ARGUMENT
        SZA
        SAD (7
        JMP E      /0 OR 7 DIGITS ILLEGAL
        CMA
        TAD (1
        DAC C      /INITIALIZE LAW INSTRUCTION
        LAC* ARRAY
        DAC ARRAY /POINTER TO FIRST WORD OF ARRAY
        XX         /LAW -DIGITS
        DAC DIGITS
        CLA        /INITIALIZE DIGIT PACK
        DAC TEMP# /STORE DIGIT PACK
        JMS* .FE  /READ DIGIT
        .DSA FORMAT           /DIGIT READ INTO FORMAT
        LAC TEMP  /LOAD DIGIT PACK
        CLL
        CTL         /MULTIPLY BY 8
        RAL
        TAD FORMAT            /ADD DIGIT
        ISZ DIGITS            /COUNT DIGITS
        JMP D       /GO BACK FOR MORE
        DAC* ARRAY            /STORE VALUE IN ARRAY ELEMENT
        ISZ ARRAY   /POINT TO NEXT ARRAY WORD
        ISZ SLOT    /COUNT ARRAY WORDS
        JMP C       /READ ANOTHER WORD
        JMS* .FF    /END OF READ
        JMP* REDAR            /EXIT
        .END
```

Example 4. A FORTRAN Program Using the Foregoing Programs

This FORTRAN program uses the preceding three MACRO programs to read in an array from the Teletype in octal and type it in decimal. The Teletype should be assigned to .DAT slot 4. Note how the arguments are specified. Notice that EQUIVALENCE performs the array element calculation at compile time.

```
C FORTRAN PROGRAM TO READ AN ARBITRARY INTEGER ARRAY IN OCTAL
C AND WRITE IT IN DECIMAL
        DIMENSION J(2000)
C USE EQUIVALENCE TO GET J(1) WITHOUT USING .SS
        EQUIVALENCE (J(1),K)
C I CONTAINS ADDRESS OF FORMAT
C STATEMENT + 1 TO MOVE OVER JMP INSTRUCTION
        ASSIGN 1 TO I
        I=I+1
1       FORMAT(6I1,1X,6I1,1X,6I1,1X,6I1,1X,6I1,1X,6I1,1X,6I1,1X,
        16I1)
C TO SIMULATE FORMAT(06,1X,06,1X,06,1X,06,1X,06,1X,06,1X,
C 06,1X,06)
C WRITE SOMETHING TO SHOW INFORMATION NEEDED
2       WRITE(4,3)
3       FORMAT(/19H READ K1 K2 K3(3I4))
C READ IN DIMENSION INFORMATION
        READ(4,4) K1,K2,K3
4       FORMAT(3I4)
C ADJUST ARRAY J TO THE PROPER SIZE
        CALL ADJ(J,K,K1,K2,K3)
C READ IN ARRAY IN OCTAL
5       CALL REDAR(4,I,6,J)
C WRITE OUT ARRAY
        WRITE(4,6) J
6       FORMAT(8I7)
C WAIT FOR ↑P
        PAUSE
C IF A0S17-0 READ IN IDENTICAL ARRAY TYPE
        IF (ITOG(1)) 2,5,2
        END
```

C.2 IFLOW AND IDZERO EXAMPLES

The following is a programming example of both the IFLOW and IDZERO functions.

```
C      MAIN PROGRAM TO SHOW USE OF IFLOW AND IDZERO
       A=10.**70
       B=10.**10
1      C=A*B
C      CALL SUBROUTINE TO CHECK FOR UNDERFLOW, OVERFLOW
C      AND DIVISION BY ZERO.
       CALL CHECK (1)
       PAUSE 1
2      C=(10.**(-70))*10,**(-20)
       CALL CHECK (1)
```

(continued on next page)

```
      PAUSE 2
3     C=A/Ø.
      CALL CHECK (1)
      PAUSE 3
      STOP
      END

C     SUBROUTINE TO CHECK FOR UNDERFLOW, OVERFLOW OR
C     DIVISION BY ZERO IN FLOATING POINT ARITHMETIC.
C     PASSING A NON-ZERO POSITIVE ARGUMENT WILL CHECK
C     FOR ALL. A ZERO ARGUMENT RESULTS IN NO
C     CHECKING.
      SUBROUTINE CHECK (N)
      LOGICAL IFLOW,IDZERO
      IF (IFLOW(N)) WRITE (1,1Ø)
      IF (IFLOW(-N)) WRITE (1,11)
      IF (IDZERO(N)) WRITE (1,12)
1Ø    FORMAT (/9H OVERFLOW)
11    FORMAT (/1ØH UNDERFLOW)
12    FORMAT (/13H DIV. BY ZERO)
      RETURN
      END
```

The result of running those programs is (with .DAT slot 1 assigned to the TTY):

```
OVERFLOW

PAUSE ØØØØØ1
↑P
UNDERFLOW

PAUSE ØØØØØ2
↑P
DIV. BY ZERO

PAUSE ØØØØØ3
↑P
STOP ØØØØØØ
```

C.3 INPUT-OUTPUT EXAMPLES

The following is a program composed mainly of I/O statements with no connected purpose. The program is presented to illustrate the possible combinations of the different types of I/O (sequential access, direct access, data-directed, ENCODE/DECODE).

```
001      C
002      C
003      C  PROGRAM EXAMPLE TO SHOW OBJECT CODE OUTPUT FOR
004      C     VARIOUS TYPES OF I/O STATEMENTS
005      C
006               IMPLICIT REAL (N)
007               DIMENSION RL1(2), RL2(3), ARR(20), NM1(2), NM2(2)
008               DATA NM1/5HNAME1, 4HASRC/,NM2/5HNAME2, 4HASRC/
 00603 472031 542542
 00605 406472 241500
 00613 472031 542544
 00615 406472 241500
009      C
010      100      FORMAT (I5,G10.3,2(E12.2))
 00000   JMP  $00000
 00001   .DSA 242226
 00002   .DSA 526216
 00003   .DSA 305405
 00004   .DSA 631530
 00005   .DSA 311210
 00006   .DSA 530544
 00007   .DSA 271445
 00010   .DSA 124500
 $00000 = 00011
011      200      FORMAT (1X,I5,G10.3,2(E12.2))
 00011   JMP  $00011
 00012   .DSA 241433
 00013   .DSA 026222
 00014   .DSA 325310
 00015   .DSA 730540
 00016   .DSA 271465
 00017   .DSA 431120
 00020   .DSA 425426
 00021   .DSA 227144
 00022   .DSA 245224
 00023   .DSA 020100
 $00011 = 00024
012               CALL DEFINE (2,100,5,0,JVB,0,0,0)
 00024   JMS* DEFINE
 00025   JMP  00036
 00026   .DSA (000002
 00027   .DSA (000144
 00030   .DSA (000005
 00031   .DSA (000000
 00032   .DSA JVB
 00033   .DSA (000000
 00034   .DSA (000000
 00035   .DSA (000000
013               CALL DEFINE (4,600,10,0,JVA,5,0,0)
 00036   JMS* DEFINE
 00037   JMP  00050
 00040   .DSA (000004
 00041   .DSA (001130
 00042   .DSA (000012
 00043   .DSA (000000
 00044   .DSA JVA
 00045   .DSA (000005
 00046   .DSA (000000
 00047   .DSA (000000
014               CALL SEEK (5,NM1)
 00050   JMS* SEEK
```

```
 00051  JMP   00054
 00052  .DSA (000005
 00053  .DSA 100000 +NM1
015              CALL ENTER (6,NM2)
016     C
017     C   I) BINARY
018     C        A) DIRECT ACCESS
019     C
 00054  JMS* ENTER
 00055  JMP   00060
 00056  .DSA (000006
 00057  .DSA 100000 +NM2
020              READ (2#JVB) INT, RL2(3), RL1
 00060  LAC   JVB
 00061  JMS* .RS
 00062  .DSA (000002
 00063  JMS* .RJ
 00064  .DSA INT
 00065  .DSA 777776
 00066  TAD   (000003
 00067  TAD   (000003
 00070  TAD   RL2
 00071  DAC   $00071
 00072  JMS* .FJ
 $00071 = 00073
 00073  .DSA $00073
 00074  JMS* .RB
 00075  .DSA 100000 +RL1
 00076  JMS* .RG
021              WRITE (2'3) INT, RL2(3), RL1
 00077  LAC   (000003
 00100  JMS* .RX
 00101  .DSA (000002
 00102  JMS* .RJ
 00103  .DSA INT
 00104  .DSA 777776
 00105  TAD   (000003
 00106  TAD   (000003
 00107  TAD   RL2
 00110  DAC   $00110
 00111  JMS* .RJ
 $00110 = 00112
 00112  .DSA $00112
022     C
023     C        B) SEQUENTIAL ACCESS
024     C
 00113  JMS* .RB
 00114  .DSA 100000 +RL1
 00115  JMS* .RG
025              READ (1) INT, RL2(3), RL1
 00116  JMS* .FS
 00117  .DSA (000001
 00120  JMS* .FJ
 00121  .DSA INT
 00122  .DSA 777776
 00123  TAD   (000003
 00124  TAD   (000003
 00125  TAD   RL2
 00126  DAC   $00126
 00127  JMS* .FJ
 $00126 = 00130
 00130  .DSA $00130
```

```
 00131  JMS* .FB
 00132  .DSA 100000 +RL1
 00133  JMS* .FG
026            WRITE (3) INT, RL2(3), RL1
 00134  JMS* .FX
 00135  .DSA (000003
 00136  JMS* .FJ
 00137  .DSA INT
 00140  .DSA 777776
 00141  TAD  (000003
 00142  TAD  (000003
 00143  TAD  RL2
 00144  DAC  $00144
 00145  JMS* .FJ
 $00144 = 00146
 00146  .DSA $00146
027     C
028     C II) ASCII
029     C      A) DIRECT ACCESS
030     C         1) FORMATTED
031     C
 00147  JMS* .FB
 00150  .DSA 100000 +RL1
 00151  JMS* .FG
032            READ (4#JVA,100) INT, RL2(3), RL1
 00152  LAC  JVA
 00153  JMS* .RR
 00154  .DSA (000004
 00155  .DSA .100
 00156  JMS* .RE
 00157  .DSA INT
 00160  .DSA 777776
 00161  TAD  (000003
 00162  TAD  (000003
 00163  TAD  RL2
 00164  DAC  $00164
 00165  JMS* .RE
 $00164 = 00166
 00166  .DSA $00166
 00167  JMS* .RA
 00170  .DSA 100000 +RL1
 00171  JMS* .RF
033            WRITE (4'5,200) INT, RL2(3), RL1
 00172  LAC  (000005
 00173  JMS* .RW
 00174  .DSA (000004
 00175  .DSA .200
 00176  JMS* .RE
 00177  .DSA INT
 00200  .DSA 777776
 00201  TAD  (000003
 00202  TAD  (000003
 00203  TAD  RL2
 00204  DAC  $00204
 00205  JMS* .RE
 $00204 = 00206
 00206  .DSA $00206
034     C
035     C         2) DATA-DIRECTED
036     C
 00207  JMS* .RA
 00210  .DSA 100000 +RL1
```

```
 00211  JMS* .RF
037              READ (4'7,) INT, RL2(3), RL1
 00212  LAC  (000007
 00213  JMS* .RR
 00214  .DSA (000004
 00215  .DSA 000000
 00216  JMS* .GD
 00217  .DSA INT
 00220  .DSA 777776
 00221  TAD  (000003
 00222  TAD  (000003
 00223  TAD  RL2
 00224  DAC  $00224
 00225  JMS* .GD
 $00224 = 00226
 00226  .DSA $00226
 00227  JMS* .GE
 00230  .DSA 100000 +RL1
 00231  JMS* .RF
038              WRITE (4#8,) INT, RL2(3), RL1
 00232  LAC  (000010
 00233  JMS* .RW
 00234  .DSA (000004
 00235  .DSA 000000
 00236  JMS* .GA
 00237  .DSA 035204
 00240  .DSA 000000
 00241  .DSA INT
 00242  JMS* .SS
 00243  .DSA RL2
 00244  LAC  (000003
 00245  DAC  $00245
 00246  JMS* .GC
 00247  .DSA 071177
 00250  .DSA 000000
 $00245 = 00251
 00251  .DSA $00251
039     C
040     C      B) SEQUENTIAL ACCESS
041     C         1) FORMATTED
042     C
 00252  JMS* .GB
 00253  .DSA 071176
 00254  .DSA 000000
 00255  .DSA 100000 +RL1
 00256  JMS* .RF
043              READ (5,100) INT, RL2(3), RL1
 00257  JMS* .FR
 00260  .DSA (000005
 00261  .DSA .100
 00262  JMS* .FE
 00263  .DSA INT
 00264  .DSA 777776
 00265  TAD  (000003
 00266  TAD  (000003
 00267  TAD  RL2
 00270  DAC  $00270
 00271  JMS* .FE
 $00270 = 00272
 00272  .DSA $00272
 00273  JMS* .FA
 00274  .DSA 100000 +RL1
```

```
 00275   JMS* .FF
044            WRITE (6,200) INT, RL2(3), RL1
 00276   JMS* .FW
 00277   .DSA (000006
 00300   .DSA .200
 00301   JMS* .FE
 00302   .DSA INT
 00303   .DSA 777776
 00304   TAD  (000003
 00305   TAD  (000003
 00306   TAD  RL2
 00307   DAC  $00307
 00310   JMS* .FE
 $00307 = 00311
 00311   .DSA $00311
 00312   JMS* .FA
 00313   .DSA 100000 +RL1
 00314   JMS* .FF
045            ENCODE (10,ARR,100) INT, RL2(3), RL1
 00315   JMS* .GF
 00316   .DSA (000012
 00317   .DSA 100000 +ARR
 00320   .DSA .100
 00321   JMS* .FE
 00322   .DSA INT
 00323   .DSA 777776
 00324   TAD  (000003
 00325   TAD  (000003
 00326   TAD  RL2
 00327   DAC  $00327
 00330   JMS* .FE
 $00327 = 00331
 00331   .DSA $00331
 00332   JMS* .FA
 00333   .DSA 100000 +RL1
 00334   JMS* .FF
046            DECODE (10,ARR,100) INT, RL2(3), RL1
 00335   JMS* .GG
 00336   .DSA (000012
 00337   .DSA 100000 +ARR
 00340   .DSA .100
 00341   JMS* .FE
 00342   .DSA INT
 00343   .DSA 777776
 00344   TAD  (000003
 00345   TAD  (000003
 00346   TAD  RL2
 00347   DAC  $00347
 00350   JMS* .FE
 $00347 = 00351
 00351   .DSA $00351
047      C
048      C       2) DATA-DIRECTED
049      C
 00352   JMS* .FA
 00353   .DSA 100000 +RL1
 00354   JMS* .FF
050            READ (5,) INT,RL2(3), RL1
 00355   JMS* .FR
 00356   .DSA (000005
 00357   .DSA 000000
 00360   JMS* .GD
```

```
 00361   .DSA INT
 00362   .DSA 777776
 00363   TAD  (000003
 00364   TAD  (000003
 00365   TAD  RL2
 00366   DAC  $00366
 00367   JMS* .GD
 $00366 = 00370
 00370   .DSA $00370
 00371   JMS* .GE
 00372   .DSA 100000 +RL1
 00373   JMS* .FF
051               WRITE (6,) INT, RL2(3), RL1
 00374   JMS* .FW
 00375   .DSA (000006
 00376   .DSA 000000
 00377   JMS* .GA
 00400   .DSA 035204
 00401   .DSA 000000
 00402   .DSA INT
 00403   JMS* .SS
 00404   .DSA RL2
 00405   LAC  (000003
 00406   DAC  $00406
 00407   JMS* .GC
 00410   .DSA 071177
 00411   .DSA 000000
 $00406 = 00412
 00412   .DSA $00412
 00413   JMS* .GB
 00414   .DSA 071176
 00415   .DSA 000000
 00416   .DSA 100000 +RL1
 00417   JMS* .FF
052               DECODE (15,ARR,) INT, RL2(3), RL1
 00420   JMS* .GG
 00421   .DSA (000017
 00422   .DSA 100000 +ARR
 00423   .DSA 000000
 00424   JMS* .GA
 00425   .DSA 035204
 00426   .DSA 000000
 00427   .DSA INT
 00430   JMS* .SS
 00431   .DSA RL2
 00432   LAC  (000003
 00433   DAC  $00433
 00434   JMS* .GC
 00435   .DSA 071177
 00436   .DSA 000000
 $00433 = 00437
 00437   .DSA $00437
 00440   JMS* .GB
 00441   .DSA 071176
 00442   .DSA 000000
 00443   .DSA 100000 +RL1
 00444   JMS* .FF
053               ENCODE (25,ARR,) INT, RL2(3), RL1
 00445   JMS* .GF
 00446   .DSA (000031
 00447   .DSA 100000 +ARR
 00450   .DSA 000000
```

```
 00451   JMS*  .GD
 00452   .DSA  INT
 00453   .DSA  777776
 00454   TAD   (000003
 00455   TAD   (000003
 00456   TAD   RL2
 00457   DAC   $00457
 00460   JMS*  .GD
$00457 = 00461
 00461   .DSA  $00461
054      C
 00462   JMS*  .GE
 00463   .DSA  100000 +RL1
 00464   JMS*  .FF
055               ENDFILE 1
 00465   JMS*  .FV
 00466   .DSA  (000001
056               ENDFILE 2
 00467   JMS*  .FV
 00470   .DSA  (000002
057               ENDFILE 3
 00471   JMS*  .FV
 00472   .DSA  (000003
058               ENDFILE 4
 00473   JMS*  .FV
 00474   .DSA  (000004
059               ENDFILE 5
 00475   JMS*  .FV
 00476   .DSA  (000005
060               ENDFILE 6
 00477   JMS*  .FV
 00500   .DSA  (000006
061               END
 00501   CLA
 00502   JMP*  .ST
 00503   JMS*  .FP
 00504   JMP   00000
 00505   .BLK  000004
 00511   .DSA  020004
 00512   .DSA  000000
 00513   .DSA  000000
 00514   .DSA  100000  +RL1
 00515   .BLK  000006
 00523   .DSA  020006
 00524   .DSA  000000
 00525   .DSA  000000
 00526   .DSA  100000  +RL2
 00527   .BLK  000050
 00577   .DSA  020050
 00600   .DSA  000000
 00601   .DSA  000000
 00602   .DSA  100000  +ARR
 00603   .BLK  000004
 00607   .DSA  020004
 00610   .DSA  000000
 00611   .DSA  000000
 00612   .DSA  100000  +NM1
 00613   .BLK  000004
 00617   .DSA  020004
 00620   .DSA  000000
 00621   .DSA  000000
 00622   .DSA  100000  +NM2
```

```
00623  .DSA DEFINE
00624  .BLK 000001
00625  .BLK 000001
00626  .DSA SEEK
00627  .DSA ENTER
00630  .DSA .RS
00631  .BLK 000001
00632  .DSA .RJ
00633  .DSA .RB
00634  .DSA .RG
00635  .DSA .RX
00636  .DSA .FS
00637  .DSA .FJ
00640  .DSA .FB
00641  .DSA .FG
00642  .DSA .FX
00643  .DSA .RR
00644  .DSA .RE
00645  .DSA .RA
00646  .DSA .RF
00647  .DSA .RW
00650  .DSA .GD
00651  .DSA .GE
00652  .DSA .GA
00653  .DSA .SS
00654  .DSA .GC
00655  .DSA .GB
00656  .DSA .FR
00657  .DSA .FE
00660  .DSA .FA
00661  .DSA .FF
00662  .DSA .FW
00663  .DSA .GF
00664  .DSA .GG
00665  .DSA .FV
00666  .DSA .ST
00667  .DSA .FP
00670  .DSA 000002
00671  .DSA 000144
00672  .DSA 000005
00673  .DSA 000000
00674  .DSA 000004
00675  .DSA 001130
00676  .DSA 000012
00677  .DSA 000006
00700  .DSA 000003
00701  .DSA 000001
00702  .DSA 000007
00703  .DSA 000010
00704  .DSA 000017
00705  .DSA 000031
   RL1     00505
   RL2     00515
   ARR     00527
   NM1     00603
   NM2     00613
   .100    00000
   .200    00011
 * DEFINE  00623
   JVB     00624
   JVA     00625
 * SEEK    00626
```

```
* ENTER   00627
* .RS     00630
  INT     00631
* .RJ     00632
* .RB     00633
* .RG     00634
* .RX     00635
* .FS     00636
* .FJ     00637
* .FB     00640
* .FG     00641
* .FX     00642
* .RR     00643
* .RE     00644
* .RA     00645
* .RF     00646
* .RW     00647
* .GD     00650
* .GE     00651
* .GA     00652
* .SS     00653
* .GC     00654
* .GB     00655
* .FR     00656
* .FE     00657
* .FA     00660
* .FF     00661
* .FW     00662
* .GF     00663
* .GG     00664
* .FV     00665
* .ST     00666
* .FP     00667
```

APPENDIX D
SYSTEM LIBRARIES

D.1 .LIBR - Page Mode Non-FPP

LIBRARY FILE LISTING FOR .LIBR PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
RBCDIO	006	136	
RBINIO	005	113	
RANCOM	009	504	
DEFINE	011	1130	
DDIO	012	2037	
EDCODE	002	255	
EOF	000	30	
UNIT	001	66	
JABS	001	15	
JDFIX	001	13	
JFIX	001	13	
FLOATJ	001	13	
JDBLE	001	10	
ISNGL	002	30	
JSIGN	003	23	
JDIM	001	21	
JMOD	001	23	
JMNMX	01P	103	
ERRSET	000	25	
IOERR	002	40	
FILE	008	376	
TIME	009	45	
TIME10	008	72	
ADJ1	000	17	
ADJ2	000	36	
ADJ3	000	41	
ABS	002	16	
IABS	000	14	
DABS	001	16	
AINT	002	15	
INT	002	13	
IDINT	005	13	
AMOD	003	27	
MOD	000	24	
DMOD	004	30	
FLOAT	002	11	
IFIX	002	13	
SIGN	004	31	
DSIGN	004	31	
ISIGN	000	20	
DIM	001	22	
IDIM	000	15	
SNGL	004	27	
DBLE	001	11	
IMNMX	05P	107	
RMNMX	08P	120	

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
DMNMX	08P	106	
.BB	004	60	
.BC	009	132	
.BD	009	132	
.BE	006	33	
.BF	005	34	
.BG	008	35	
.BH	005	34	
.BI	003	120	
SQRT	008	73	
SIN	003	13	
COS	003	20	
ATAN	002	13	
ATAN2	007	44	
EXP	002	13	
ALOG	002	20	
ALOG10	002	20	
TANH	004	47	
.EB	004	102	
.ED	005	67	
.EE	002	71	
.EF	004	116	
.EC	001	44	
DSQRT	007	71	
DSIN	001	13	
DCOS	001	21	
DATAN	001	13	
DATAN2	007	46	
DEXP	001	13	
DLOG	003	21	
DLOG10	001	21	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	005	146	
.DB	004	120	
.DE	003	101	
.DF	001	137	
.DC	001	47	
.DA	P06	56	
BCDIO	033	3724	
BINIO	015	363	
AUXIO	010	133	
.SS	005	60	
GOTO	003	26	
STOP	003	13	

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
PAUSE	005	14	
SPMSG	004	73	
.FLTB	004	266	
FIOPS	017	735	
PARTWD	03P	140	
DBLINT	07P	377	
INTEAE	07P	131	
DOUBLE	004	203	
RELEAE	10P	1077	
OTSER	009	210	
.CB	004	22	CLOSE

D.2 .LIBRF - Page Mode FPP

LIBRARY FILE LISTING FOR .LIBRF PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
RBCDIO	005	136	
RBINIO	005	113	
RANCOM	009	504	
DEFINE	011	1130	
DDIO	F12	2012	
EDCODE	002	255	
EOF	000	30	
UNIT	001	66	
JABS	F01	14	
JDFIX	F01	12	
JFIX	F01	12	
FLOATJ	F01	10	
JDBLE	F01	10	
ISNGL	F02	13	
JSIGN	F03	16	
JDIM	F01	17	
JMOD	F01	17	
JMNMX	F1P	100	
ERRSET	000	25	
IOERR	002	40	
FILE	008	376	
TIME	009	45	
TIME10	008	72	
ADJ1	000	17	
ADJ2	000	36	
ADJ3	000	41	
ABS	F02	13	
IABS	000	14	
DABS	F01	13	
AINT	F02	14	
INT	F02	12	
IDINT	F05	12	
AMOD	F03	23	
MOD	000	24	
DMOD	F04	23	
FLOAT	002	11	
IFIX	F02	12	
SIGN	F04	24	
DSIGN	F04	24	
ISIGN	006	20	
DIM	F01	17	
IDIM	000	15	
SNGL	F04	16	
DBLE	F01	10	
IMNMX	05P	107	
RMNMX	F8P	115	

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
DMNMX	F8P	104	
.BB	004	60	
.BC	F09	126	
.BD	F09	126	
.BE	F06	30	
.BF	F05	31	
.BG	F08	31	
.BH	F05	31	
.BI	F03	113	
SQRT	F08	73	
SIN	F03	12	
COS	F03	16	
ATAN	F02	12	
ATAN2	F07	36	
EXP	F02	12	
ALOG	F02	16	
ALOG10	F02	16	
TANH	F04	46	
.EB	F04	77	
.ED	F05	66	
.EE	F02	72	
.EF	F04	111	
.EC	F01	40	
DSQRT	F07	70	
DSIN	F01	12	
DCOS	F01	17	
DATAN	F01	12	
DATAN2	F07	42	
DEXP	F01	12	
DLOG	F03	17	
DLOG10	F01	17	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	F05	137	
.DB	F04	115	
.DE	F03	104	
.DF	F01	130	
.DC	F01	43	
.DA	P06	56	
BCDIO	F33	3634	
BINIO	015	363	
AUXIO	010	133	
.SS	005	60	
GOTO	003	26	
STOP	003	13	

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
PAUSE	005	14	
SPMSG	004	73	
.FLTB	004	266	
FIOPS	017	735	
PARTWD	F3P	146	
INTFAE	07P	131	
.FPP	F12	407	
OTSER	009	210	
.CB	004	22	CLOSE

APPENDIX E
PDP-15 FORTRAN FACILITIES

The extended FORTRAN language described in this manual and in the companion manual (Operating Environmental Manual DEC-15-GFZA-D) is available only on the systems described below. The FORTRAN existing on other PDP-15 systems is described in a manual entitled "PDP-15 FORTRAN IV Programmer's Reference Manual" (DEC-15-KFZB-D).

The following tables describe the existing versions of the extended compiler, the extended Object Time System Libraries, and the compiler-library pairs available for different systems. All versions of the compiler are written in PDP-9 code, however, 'PDP-9 mode' versions produce only PDP-9 code as output while 'PDP-15 mode' versions may produce PDP-15 instructions where suitable. Page and Bank Mode libraries differ not only in the use of the PDP-15 versus PDP-9 code, but also in the values of address masking constants used in a few of the routines. Note that the Floating Point Processor (FPP) is supported only on the PDP-15, thus there is no PDP-9 mode version.

The library names used in the following tables are given for designational purposes within this appendix only and do not necessarily reflect the names under which the libraries are distributed.

Table E-1
Versions of the Extended Compiler

Main Version	Features	Version	System	Approx. Size (8)
F4X	All	F4X F4X9 FPF4X	Non-FPP, PDP-15 mode DOS-15 Non-FPP, PDP-9 mode DOS-15 FPP, PDP-15 mode DOS-15	15406 15363 15661
F4B	All except direct-access I/O	F4B F4B9 FPF4B	Non-FPP, PDP-15 mode, ADSS (V5B) Non-FPP, PDP-9 mode ADSS (V5B) FPP, PDP-15 mode ADSS (V5B)	15251 15226 15522
F4RX	All except direct-access I/O	F4RX FPF4RX	Non-FPP, PDP-15 mode RSX FPP, PDP-15 mode RSX	

Table E-2
Versions of the OTS Libraries for the Extended Compiler

System	Contents	Libraries	Subsystem
DOS-15 (BOSS-15)	Contains all routines, assembled for DOS-15 operation.	.LBXP .LBXB .LBXPF .LBXBF	Non-FPP, Page Non-FPP, Bank FPP, Page FPP, Bank
ADSS	Contains all routines except direct-access (DEFINE, RANCOM, RBINIO, RBCDIO) assembled for ADSS operation.	.LBRP .LBRB .LBRPF .LBRBF	Non-FPP, Page Non-FPP, Bank FPP, Page FPP, Bank
RSX	Contains all routines except direct-access (DEFINE, RANCOM, RBINIO, RBCDIO) and magtape subroutines (UNIT, EOF), assembled for RSX operation and includes added routines applicable to RSX only.	.LIBRX .LIBFX	Non-FPP, Page/Bank FPP, Page/Bank

Table E-3
Compilers and Libraries for Extended FORTRAN
Distributed with PDP-9/15 Systems

System		Non-FPP		FPP	
		Page	Bank	Page	Bank
DOS-15	Compiler	F4X	F4X or F4X9	FPF4X	FPF4X
(BOSS-15	Library	.LBXP	.LBXB	.LBXPF	.LBXBF
ADSS V5B	Compiler	F4B	F4B or F4B9	FPF4B	FPF4B
	Library	.LBRP	.LBRB	.LBRPF	.LBRBF
RSX	Compiler	F4RX	F4RX	FPF4RX	FPF4RX
	Library	.LIBRX	.LIBRX	.LIBFX	.LIBFX

INDEX

U

W

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 & PDP-12
Digital Software News for the PDP-11
Digital Software News for the PDP-9/15 Family

These newsletters contain information applicable to software available from Digital's Program Library, Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's Software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problem to:

Software Information Service
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

These forms which are provided in the software kit should be fully filled out and accompanied by teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software and manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. U.S.A. customers may order directly from the Program Library in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

--- 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

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability and readability.

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

How can this manual be improved?

Other comments?

Please state your position. ______________________ Date: ____________

Name: ______________________ Organization: ______________________

Street: ______________________ Department: ______________________

City: ______________ State: ______________ Zip or Country ____________

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

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754