

**FORTRAN IV XVM
OPERATING ENVIRONMENT
MANUAL**

DEC-XV-LE4EA-A-D



XVM
Systems
digital

**FORTRAN IV XVM
OPERATING ENVIRONMENT
MANUAL
DEC-XV-LF4EA-A-D**

First Printing, December 1975

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

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

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

Copyright © 1975 by Digital Equipment Corporation

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

The following are trademarks of Digital Equipment Corporation:

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

CONTENTS

			Page
		PREFACE	ix
CHAPTER	1	INTRODUCTION	1-1
	1.1	OPERATING PROCEDURES	1-1
	1.2	SOFTWARE ENVIRONMENTS	1-6
	1.2.1	XVM/DOS	1-6
	1.2.2	BOSS XVM	1-7
	1.2.3	XVM/RSX	1-7
	1.3	HARDWARE ENVIRONMENT	1-7
CHAPTER	2	INPUT-OUTPUT PROCESSING	2-1
	2.1	GENERAL INFORMATION	2-1
	2.1.1	Device Assignment	2-2
	2.1.2	Data Structures	2-2
	2.1.3	Data Transmission	2-3
	2.2	DATA TRANSMISSION (FIOPS)	2-3
	2.3	SEQUENTIAL INPUT-OUTPUT	2-4
	2.3.1	OTS Formatted Input/Output	2-5
	2.3.2	OTS Binary Input/Output (BINIO)	2-6
	2.3.3	OTS Auxiliary Input/Output (AUXIO)	2-7
	2.4	DIRECT ACCESS I/O	2-9
	2.4.1	The DEFINE Routine	2-9
	2.4.2	Formatted Input/Output (RBCDIO)	2-11
	2.4.3	Unformatted Input/Output (RBINIO)	2-12
	2.4.4	Initialization and Actual Data Transfer (RANCOM)	2-13
	2.5	DATA-DIRECTED INPUT-OUTPUT (DDIO)	2-13
	2.6	ENCODE/DECODE (EDCODE)	2-15
	2.7	USER SUBROUTINES	2-15
	2.7.1	DOS Directoried Subroutines	2-15
	2.7.2	RSX Directoried Subroutines	2-17
	2.7.3	BOSS Routines	2-18
CHAPTER	3	THE SCIENCE LIBRARY	3-1
	3.1	INTRINSIC FUNCTIONS	3-2
	3.2	EXTERNAL FUNCTIONS	3-6
	3.2.1	Square Root (SQRT, DSQRT)	3-6
	3.2.2	Exponential (EXP, DEXP)	3-6
	3.2.3	Natural and Common Logarithms (ALOG, ALOG10, DLOG, DLOG10)	3-8
	3.2.4	Sine and Cosine (SIN, COS, DSIN, DCOS)	3-9

CONTENTS (Cont.)

			Page
	3.2.5	Arctangent (ATAN, DATAN, ATAN2, DATAN2)	3-10
	3.2.6	Hyperbolic Tangent	3-10
	3.3	SUB-FUNCTIONS	3-11
	3.3.1	Logarithm, Base 2 (.EE, .DE)	3-11
	3.3.2	Polynomial Evaluator (.EC, .DC)	3-14
	3.4	THE ARITHMETIC PACKAGE	3-14
CHAPTER	4	UTILITY ROUTINES	4-1
	4.1	OTS ROUTINES	4-1
	4.2	FLOATING POINT PROCESSOR ROUTINES	4-6
	4.3	FORTRAN-CALLABLE UTILITY ROUTINES	4-6
	4.4	RSX LIBRARY (.LIBRX BIN OR .LIBFX BIN) ROUTINES	4-6
CHAPTER	5	FORTRAN-IV AND MACRO	5-1
	5.1	INVOKING MACRO SUBPROGRAMS FROM FORTRAN	5-1
	5.2	INVOKING FORTRAN SUBPROGRAMS FROM MACRO	5-2
	5.3	COMMON BLOCKS	5-3
APPENDIX	A	FORTRAN LANGUAGE SUMMARY	A-1
	A.1	EXPRESSION OPERATORS	A-1
	A.2	STATEMENTS	A-2
APPENDIX	B	ERROR MESSAGES	B-1
	B.1	COMPILER ERROR MESSAGES	B-1
	B.2	OTS ERROR MESSAGES	B-7
	B.3	OTS ERROR MESSAGES IN FPP SYSTEMS	B-9
APPENDIX	C	PROGRAMMING EXAMPLES	C-1
	C.1	A FUNCTION TO READ THE AC SWITCHES	C-1
	C.2	IFLOW AND IDZERO EXAMPLES	C-1
	C.3	INPUT-OUTPUT EXAMPLES	C-2
APPENDIX	D	SYSTEM LIBRARIES	D-1
	D.1	DOS-15 PAGE MODE NON-FPP LIBRARY	D-1
	D.2	DOS-15 BANK MODE NON-FPP LIBRARY	D-4
	D.3	DOS-15 PAGE MODE FPP LIBRARY	D-7
	D.4	DOS-15 BANK MODE FPP LIBRARY	D-10
	D.5	RSX PLUS III NON-FPP LIBRARY	D-13
	D.6	RSX PLUS III FPP LIBRARY	D-17

CONTENTS (Cont.)

FIGURES

		Page
Figure 1-1	Sample XVM/DOS Session	1-5

TABLES

		Page
Table 3-1	Intrinsic Functions	3-3
3-2	External Functions	3-7
3-3	Sub-Functions	3-12
3-4	Arithmetic Package	3-17
4-1	FORTTRAN-Callable Utility Routines	4-7
A-1	FORTTRAN Library Functions	A-12

INDEX		Index-1
-------	--	---------

LIST OF ALL XVM MANUALS

The following is a list of all XVM manuals and their DEC numbers, including the latest version available. Within this manual, other XVM manuals are referenced by title only. Refer to this list for the DEC numbers of these referenced manuals.

BOSS XVM USER'S MANUAL	DEC-XV-OBUAA-A-D
CHAIN XVM/EXECUTE XVM UTILITY MANUAL	DEC-XV-UCHNA-A-D
DDT XVM UTILITY MANUAL	DEC-XV-UDDTA-A-D
EDIT/EDITVP/EDITVT XVM UTILITY MANUAL	DEC-XV-UETUA-A-D
8TRAN XVM UTILITY MANUAL	DEC-XV-UTRNA-A-D
FOCAL XVM LANGUAGE MANUAL	DEC-XV-LFLGA-A-D
FORTTRAN IV XVM LANGUAGE MANUAL	DEC-XV-LF4MA-A-D
FORTTRAN IV XVM OPERATING ENVIRONMENT MANUAL	DEC-XV-LF4EA-A-D
LINKING LOADER XVM UTILITY MANUAL	DEC-XV-ULLUA-A-D
MAC11 XVM ASSEMBLER LANGUAGE MANUAL	DEC-XV-LMLAA-A-D
MACRO XVM ASSEMBLER LANGUAGE MANUAL	DEC-XV-LMALA-A-D
MTDUMP XVM UTILITY MANUAL	DEC-XV-UMTUA-A-D
PATCH XVM UTILITY MANUAL	DEC-XV-UPUMA-A-D
PIP XVM UTILITY MANUAL	DEC-XV-UPPUA-A-D
SGEN XVM UTILITY MANUAL	DEC-XV-USUTA-A-D
SRCCOM XVM UTILITY MANUAL	DEC-XV-USRCA-A-D
UPDATE XVM UTILITY MANUAL	DEC-XV-UUPDA-A-D
VP15A XVM GRAPHICS SOFTWARE MANUAL	DEC-XV-GVPAA-A-D
VT15 XVM GRAPHICS SOFTWARE MANUAL	DEC-XV-GVTAA-A-D
XVM/DOS KEYBOARD COMMAND GUIDE	DEC-XV-ODKBA-A-D
XVM/DOS READERS GUIDE AND MASTER INDEX	DEC-XV-ODGIA-A-D
XVM/DOS SYSTEM MANUAL	DEC-XV-ODSAA-A-D
XVM/DOS USERS MANUAL	DEC-XV-ODMAA-A-D
XVM/DOS V1A SYSTEM INSTALLATION GUIDE	DEC-XV-ODSIA-A-D
XVM/RSX SYSTEM MANUAL	DEC-XV-IRSMA-A-D
XVM UNICHANNEL SOFTWARE MANUAL	DEC-XV-XUSMA-A-D

C
.
G
O
E
C

PREFACE

This manual describes the system software facilities which support the Digital XVM 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 System¹ (OTS), and the Science Library². All descriptions presented apply to the XVM versions of the FORTRAN compiler. Appendix E presents overall outlines and descriptions and detailed data specifying the difference among the various compilers for the XVM/DOS and XVM/RSX software systems.

A companion manual FORTRAN IV XVM LANGUAGE MANUAL describes the elements, syntax and use of the FORTRAN IV language as implemented for the XVM computer.

The following is a list of XVM documents which either support directly or contain information useful in understanding FORTRAN IV XVM and its function:

XVM/DOS User's Manual

XVM/RSX System Manual

Linking Loader XVM Utility Manual

CHAIN XVM/EXECUTE XVM Utility Manual

¹The 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.

²The 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 or a floating point instruction, depending on the hardware configuration on which the program is produced.

The programmer 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 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, the basic procedures are described 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 XVM/DOS monitor, and differences for the XVM/RSX environment are noted. The term DOS/RSX implies XVM/DOS and XVM/RSX.

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 XVM assembled routines and with required OTS library routines. Program linkage may be

Introduction

accomplished via the linking loader, LOAD or GLOAD (DOS), which loads the resulting program directly into core for immediate execution. The user may, alternatively, use one of the overlay linkage editors - CHAIN (DOS) or TKB (RSX). These construct core images onto auxiliary storage for later execution.

In DOS, 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:

```
F4M XVM Vnxnnn
```

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. Filename is the name of the source file to compile, and its extension must be SRC. A carriage return specifies that FORTRAN-IV should be restarted after the current program has been compiled. ALT MODE returns control to the monitor.

In RSX, the compiler is invoked from TDV by typing the compiler task name, followed by a command string. As provided with the XVM/RSX monitor, the compiler task name is FOR..., and it is invoked by typing FOR or F4F following the monitor's TDV response. The compiler call plus the command string has the form:

```
{ FOR } optionlist ← filename1, filename2, ... filenameN  
{ F4F }
```

The options are delimited by a left arrow, and may optionally be separated by commas. Any number of filenames, whose extensions are SRC, may be specified, separated by commas. The entire line is terminated by a carriage return or an ALT MODE, and no continuation of the command string is allowed.

Introduction

For either DOS or RSX, the option list may be empty or contain any of the following options:

<u>Option</u>	<u>Meaning</u>
O	Object Listing
S	Symbol Map
L	Source Listing
B	Binary Output
H	Use subroutine for calculation of array element addresses ¹
R	RSX only - print compiler version and "End Pass 1" on output terminal.

The output listing file always has the extension LST. All file names must be legal FORTRAN symbols. At the end of pass 1, the compiler types

END PASS1

and allows the repositioning of a source tape if using the paper tape reader. When compiling from paper tape in DOS only, to initiate pass 2, the user types ↑P (control P).

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

	<u>Message</u>	<u>Meaning</u>
	?	Bad command string - retype
DOS	{ IOPS4 IOPSn	I/O device not ready - type CTRL R when ready See XVM/DOS User's Manual for IOPS error codes
RSX	{ FORTRAN-I/O ERROR LUN xx yyyy	An I/O error occurred during compilation; xx represents the logical unit number (decimal) on which the error occurred; yyyy is the octal event variable indicating the cause of the error. (See XVM/RSX Reference Manual for details.)

¹The subroutine .SS is used to calculate the addresses of 2 and 3 dimensional array elements; the default uses in line code in most cases.

Introduction

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).

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

```
LOADER XVM Vnxnnn  
>
```

and awaits a command string specifying programs to be loaded and output options. See the LINKING LOADER XVM UTILITY Manual for detailed instructions. Figure 1-1 shows the printout from a typical XVM/DOS session from source-program preparation to loading.

With CHAIN, the DOS 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 overlaid. A link's core image is not recorded or "swapped out" when it is overlaid. The same image is brought into core each time a link is loaded. See the CHAIN XVM/EXECUTE XVM Utility Manual for detailed instructions.

In RSX, linking is accomplished by using the TDV function Task Builder (TKB). TKB is similar in operation to CHAIN. Its function is to record core images in a file in the format expected by the RSX INSTALL Function. The task name is used as the file name, and TSK is used as the extension. TKB accepts the same overlay descriptions as CHAIN. In RSX it is called by typing "TKB" following the Monitor's TDV request. When loaded, TKB types its name and version number and makes the following requests:

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

For further information, see the XVM/RSX System Manual.

Introduction

```
XVM/DOS V1A000
  ENTER DATE (MM/DD/YY) - 11/6/75

PAGE MODE   32K  API ON  UC15 ON  POLLER ON  SCR
$LOGIN DEM

XVM/DOS V1A000

PAGE MODE   32K  API ON  UC15 ON  POLLER ON  DEM
$PIF
PTP XVM V1A000
>N SY
>C
XVM/DOS V1A000
$EDIT
EDITOR XVM V1A000
>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 = R1 ** R2
      WRITE (6, ) R3
      STOP
      END

EDIT
>EXIT

XVM/DOS V1A000
$F4
FFF4M XVM V1A001
>B_IOTST
END PASS1
PROGRAM SIZE = 00105, NO ERRORS

XVM/DOS V1A000
$A TT 6
$LOAD
```

Figure 1-1
Sample XVM/DOS Session

Introduction

```
LOADER XVM V1A000
>P_IOTST
P IOTST      077532
P DDIO      F18 075514
P .BE       F06 075464
P .EE       F02 075372
P .EF       F08 075232
P .EC       F01 075172
P BCDT0     F51 071256
P .SS       009 071146
P STOP      008 071065
P SPMSG     012 070746
P .FLTB     004 070460
P FIOPS     D40 066557
P .FFP      F18 070020
P OTSER     F14 066344
P .CB       004 066322
^SCS
IN:
11.2,3.0

OUT:
'R3' = 1404.9279

STOP 000000
```

```
XVM/DOS V1A000
```

```
*
```

Figure 1-1 (Cont.)
Sample XVM/DOS Session

1.2 SOFTWARE ENVIRONMENTS

Each version of FORTRAN-IV has its own version of the object timesystem 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 XVM/DOS

XVM/DOS is a single-user, interactive, disk-resident operating system. It includes the DOS 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 disk file structure supports both direct and sequential access to disk files, dynamic disk storage allocation, and file protection. The DOS 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 XVM/DOS User's Manual, for more detailed information.

Introduction

1.2.2 BOSS XVM

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

Some highlights of BOSS XVM are:

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

1.2.3 XVM/RSX

XVM/RSX is a monitor system designed to handle real-time information in a multiprogramming environment. RSX 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. System software includes the FORTRAN IV compiler, the MACRO Assembler, the TASK BUILDER, and numerous utility programs required to edit, debug, and run user programs. Details are available in the XVM/RSX System Manual.

1.3 HARDWARE ENVIRONMENT

Systems with an FP15 Floating-Point Processor (FPP) have a special version of the FORTRAN-IV compiler and OTS which utilizes hardware instructions rather than subroutine calls to handle certain arithmetic functions. For example, RELEAE, the REAL arithmetic package, is not included in FPP systems since REAL arithmetic expressions may be compiled directly into FPP instructions.

The FPP FORTRAN System consists of the standard FORTRAN-IV compiler and Object-Time System (OTS) interfaced (via conditional assembly, and additional routines) to the hardware FPP. The interface applies to Single and Double Precision Floating-Point Arithmetic and Extended Integer Arithmetic (double integers). Single integer arithmetic is still handled by EAE (extended arithmetic element, KE15) instructions and, in part, by software.

C
.
g
(
O
B
n
C

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, process formatted and unformatted direct-access READ and WRITE statements;
- (5) DDIO, manages data-directed input-output;
- (6) DCODE, 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.

Input-Output Processing

2.1.1 Device Assignment

In DOS, device assignment is managed through the monitor Device Assignment Table (.DAT) which associates legal device units to physical ones. .DAT has "slot" numbers which correspond to the FORTRAN logical device numbers. Each slot, at run time, contains the physical device unit number and a pointer to the appropriate device handler in memory. Sixteen¹ 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. An analogous structure is maintained in the RSX system. It is called the Logical Unit Table (LUT), and for each Logical Unit Number (LUN) assigned, it maintains an address to information about the associated physical device.

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 disks which are also directoried devices may be used.

Serial devices used for FORTRAN Input-Output include paper tape, magnetic tape and (in certain modes) 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.

Magnetic tape and 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 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_g words per block) which comprise a file are chained via a forward link word (377_g) and backward link word (376_g). 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). In DOS only, if this is not done, default names are used. A default name has the form .TM0mn OTS where mn is the logical device number in decimal.

¹ This number is the standard size for DOS but may be changed by system generation and assembly parameters.

Input-Output Processing

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_g words is used. In DOS, 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. In RSX, the size used is always that of the FIOPS buffer, 400_g.

2.2 DATA TRANSMISSION (FIOPS)

The FIOPS package provides the necessary communication between the OTS and the I/O Device Handlers. 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 device that is in use. 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. (See also the .ZR call, described in paragraph 4.1.)

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. (In RSX, the first call to .FC sets the I/O flag and a fixed buffer size. Subsequent calls to .FC only reset the I/O flag.)

(continued next page)

Input-Output Processing

FIOPS Package (Cont)

Routine	Function
<p>.FQ Call: LAC (address of .DAT slot number (bits 9-17) IOPS mode (bits 6-8) JMS* .FQ</p>	<p>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. In RSX, the READ, WRITE, and WAITFOR I/O functions are used.</p>
<p>.FP Call: JMS* .FP</p>	<p>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.</p>

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

<u>Value</u>	<u>Error</u>
-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 if DOS on 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.

Input-Output Processing

2.3.1 OTS Formatted Input/Output

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. On output to a printing device, 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, 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) ¹	Initialize BCDIO for Input (output)
.FE Call: (ones complement of mode in AC) JMS* .FE .DSA (address of data item (first word)) (return with original AC)	Inputs or outputs a data item using format decoder (.FD). Contents of AC prior to call: 777777: INTEGER ØØ LOGICAL 777776: REAL 777775: DOUBLE PRECISION 777774: DOUBLE INTEGER
.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)

¹This word is 0 for data-directed (implied format) I/O.

BCDIO Package (Cont)

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 377777₈ 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.

Input-Output Processing

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 (IOPS 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: (one's complement of mode in AC) JMS* .FJ .DSA (address of item (first) word) (returns with original AC)	Transfers a data item to or from the line buffer (all modes). Mode of item is: 77777 = INTEGER or LOGICAL 77776 = REAL 77775 = DOUBLE PRECISION 77774 = 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. In DOS, AUXIO routines issue .MTAPE monitor calls giving .DAT slot and a code specifying the magnetic tape function desired:

Input-Output Processing

<u>Code</u>	<u>Magnetic Tape</u>	<u>Disk</u>
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.

In RSX, AUXIO issues the magnetic tape I/O functions BSPREC, REWIND, and WREOF. Thus, in RSX, the BACKSPACE, REWIND, and ENDFILE commands should be issued only to an actual magnetic tape device, not to a disk.

AUXIO External Calls: FIOPS, .FLTb Errors: OTS 10 - illegal I/O device OTS 11 - illegal input data (IOPS mode incorrect)		
Routine	Magnetic Tape	Disk (DOS only)
.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 (address of .DAT slot)	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 is described in Chapter 6 of the FORTRAN IV XVM Language Manual.

The following discusses the DOS implementation of DEFINE; for information on the RSX version, see the above manual.

In DOS, the DEFINE call is:

```
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 ($\leq 131071_{10}$)
- F - array reference to filename and extension; if 0, this is a temporary file using a default name
- V - associated variable - set to number of the last accessed record plus one
- M - mode 0 = IOPS binary (unformatted)
 non-0 = IOPS ASCII (formatted)
- A - file size adjustment indicator
 0 = no adjustment
 non-0 = adjust
- L - deletion indicator
 0 = no deletion
 non-0 = delete upon closing, if this is a temporary file

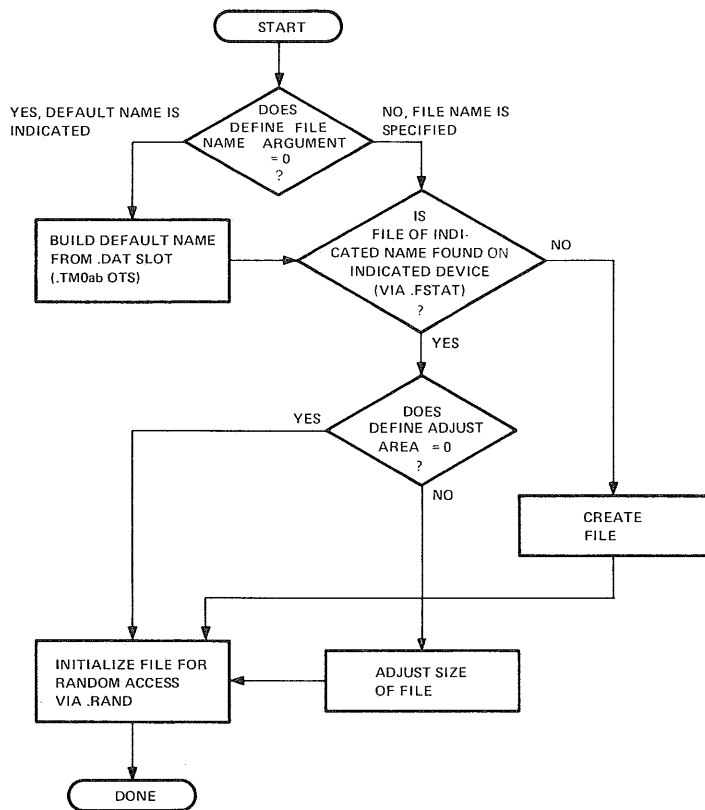
The DEFINE routine initializes a file for direct-access in one of four ways, depending on the combination of parameters supplied.

- 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 (040g). If the mode is binary, all data words are set to 0 and the ID word for each record to 400000_g.

Input-Output Processing

- 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 .TM0abL OTS (unless a file of that name already exists on .DAT slot D) where ab specifies .DAT slot in decimal. 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.



Input-Output Processing

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	File active bit (1 if active - always set for ASCII files)
	2-11	Number of blocks per record
	12-17	.DAT slot number
2	0	mode - 0 if binary; 1 if ASCII
	5-11	Word pairs per record
	12-17	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 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 made 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 in DOS, and the DSKGET and DSKPUT I/O functions in RSX.

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.

Input-Output Processing

RBCDIO	
External Calls: FIOPS, BCDIO (.FE, .FA), OTSER, RANCOM	
Errors: None	
Routine	Purpose
.RW (.RR) Call: JMS* .RW(.RP) .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 format 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. If RSX, the last record is packed, and a DSKPUT is performed if the buffer is full.

Entry points to BCDIO are:

RBCDIO Entry

.RE
.RA

BCDIO Routine

.FE
.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 on next page)

Input-Output Processing

RBINIO (Cont)

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

Entry points to BINIO are:

<u>RBINIO Entry</u>	<u>BINIO Routine</u>
.RJ	.FJ
.RA	.FA

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 accommodate. On output, DDIO has a set of default format parameters for each type of variable.

Input-Output Processing

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. 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 ¹	
Routine	Function
.GA Call: (one's complement of mode in AC) JMS* .GA name 1 first 3 characters ,radix 50 name 2 last 3 characters ,radix 50 .DSA address item (returns with original AC)	Outputs a data item in the 'NAME'=value form. If the mode is 0 (integer-logical), bit 0 of the name word indicates which (0 for integer, 1 for logical).
.GC Call: (one's complement of mode in AC) JMS* .GC name 1 name 2 .DSA item (returns with original AC)	Outputs an array element in 'NAME(I)'=value form. .GC should only be used when .SS has been used to calculate the subscript address.
.GB Call: JMS* .GB name 1 name 2 .DSA array descriptor block (word # 5 address)	Outputs an entire array in 'NAME(I)'=value form.
.GD Call: (one's complement of mode in AC) JMS* .GD .DSA item (returns with original AC)	Inputs an item.
.GE Call: JMS* .GE .DSA addr. of array descriptor block word 5	Inputs an array.

¹For Terminal input - 'BAD INPUT DATA - RETYPE FROM INPUT WITH ERROR' is typed.

Input-Output Processing

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 DOS Directoried Subroutines

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

Input-Output Processing

<p>FILE</p> <p>External Calls: FIOPS, .DA</p> <p>Errors: OTS 10 - illegal device number OTS 13 - file not found (SEEK) OTS 14 - directory full (ENTER)</p>		
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.
RENAM	CALL RENAM (n,A,B,I) Where: A is an array containing exist- ing name B is an array containing a new file name I = 0 if file not found; -1 if found and action complete	Searches for named file and renames it.
DELETE	CALL DELETE (n,A,I) Where: A is an array containing exist- ing file name I = 0 if file not found; -1 if found and action complete	Searches for named file and deletes it.

Input-Output Processing

2.7.2 RSX Directoried Subroutines:

Routine	Calling Sequence	Purpose
SEEK	<p>CALL SEEK (n,nHname,nHext[,ev])</p> <p>Where:</p> <p>n = LUN number</p> <p>nHname = is a Hollerith constant or DOUBLE INTEGER or REAL variable which specifies the 1 to 5 character file name.</p> <p>nHext = is a Hollerith constant or DOUBLE INTEGER or REAL variable which specifies the 1 to 3 character file extension.</p> <p>ev = is an optional integer variable which will if specified, contain upon return, the setting of the returned event variable from the SEEK CPB.</p>	Finds and opens a named input file.

Routine	Calling Sequence	Purpose
ENTER	<p>CALL ENTER (n,nHname,nHext[,ev])</p> <p>Where:</p> <p>All parameters are as for SEEK</p>	Creates and opens a named output file
Routine	Calling Sequence	Purpose
CLOSE	<p>CALL CLOSE (n[,nHname,nHext[,ev]])</p> <p>Where:</p> <p>Parameters are as with SEEK, except file name and extension are optional, used only with CALL RENAME.</p>	Terminates an input or output file and closes it. Also is used to specify the new name of a RENAME'd file (see RENAME)

(continued next page)

Input-Output Processing

Routine	Calling Sequence	Purpose
RENAME	CALL RENAME (n,nHname,nHext[,ev]) Where: Parameters are as with SEEK.	Open the specified file for the purpose of renaming it to the name specified in the following CLOSE command.

Routine	Calling Sequence	Purpose
DELETE	CALL DELETE (n,nHname,nHext[,ev]) Where: Parameters are as with SEEK.	Searches for named file and deletes it.

2.7.3 BOSS Routines

These FORTRAN-callable routines affect BOSS I/O when called by a program running under BOSS.

BOSTT

External Calls: None

Errors: None

Routine	Call	Purpose
TON	CALL TON	Allows output intended for the teletype to be printed on the teletype.
TTOF	CALL TTOF	Restores normal BOSS function by directing all teletype output to the line printer.

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. Subprograms called by MACRO programs must be declared with a .GLOBL pseudo-operation as in:

Examples:

	Standard System	Floating Point (FPP) System
<pre> .TITLE .GLOBL SIN, .AH . . . JMS* SIN JMP .+2 .DSA A JMS* .AH .DSA X . . . X .DSA 0 .DSA 0 </pre>	<pre> /JUMP beyond argument /+400000 if indirect /store in real format at /X </pre>	<pre> .TITLE .GLOBL SIN FST = 713640 . . . JMS* SIN JMP .+2 .DSA A FST .DSA X . . . X .DSA 0 .DSA 0 </pre>

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

3.1 INTRINSIC FUNCTIONS

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

Intrinsic functions may be explicitly named, as when referenced via an arithmetic statement.

For example

$$X = \text{ABS}(A)$$

They must have proper mode specification, and the correct number of arguments.

References to intrinsic functions are also generated implicitly. For instance, when

$$X = A^{**}B$$

is coded, and A and B are type REAL, a call to .BE is generated.

(Table 3-1 begins 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.	INTEGE
		.BC .BC .BL	R**I (or DI) R=R**I R=R**DI	ARG1 IN FLT. ACC JMS* SUBR .DSA ADDR of ARG2	None	N.A.	REAL
		.BD .BD .BM	DP**I (or DI) DP=DP**I DP=DP**DI	ARG1 IN FLT. ACC JMS* SUBR .DSA ADDR of ARG2	None	N.A.	REAL
		.BE .BF .BG .BH	R=R**R DP=R**DP DP=DP**R DP=DP**DP	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**DI, DI**DI (or I) I=I**DI DI=DI**DI DI=DI**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 SINGL DBLE JFIX ISINGL JDBLE JDFIX FLOATJ DBLEJ	R=FLOAT(I) I=IFIX(R) R=SINGL(D) D=DBLE(R) DI=JFIX(R) or JFIX(DP) I=ISINGL(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. * N.A.	.DA, REAL .DA, INTEGER .DA, DOUBLE .DA, DBLINT

*ARG1/ARG2 < 131072

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)		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)	JMS*SUBR JMP .m+1 .DSA ADDR of ARG1 : : .DSA ADDR of ARGn			INTEGER, REAL
		Double- precision (DMNMX) DMAX1 DMIN1	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 sub-program 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:

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

If EXP is even:

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

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

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

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 3-8)

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
Exponential	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	$\text{Log}_e ARG$	ALOG DLOG	R=ALOG(R) DP=DLOG(DP)	Same	Same	26 32	.DA,.EE,.ER,REAL .DA,.DE,.ER,DOUBLE
Common logarithm	$\text{Log}_{10} ARG$	ALOG10 DLOG10	R=ALOG10(R) DP=DLOG10(DP)	Same	Same	Same	Same
Sine	$\text{Sin}(ARG)$	SIN DSIN	R=SIN(R) DP=DSIN(DP)	Same	None	26 34	.DA,.EB,REAL .DA,.DB,DOUBLE
Cosine	$\text{cos}(ARG)$	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)	$\tan^{-1} \left(\frac{ARG1}{ARG2} \right)$	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
Hyperbolic tangent	$\text{tanh}(ARG)$	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.

<u>Value of i</u>	<u>Value of C_i</u>
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:

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

Where:

$$n = 2 \text{ (ALOG)}$$

$$n = 3 \text{ (DLOG)}$$

The values of C are given below:

ALOG and ALOG 10

$$C_1 = 2.8853913$$

$$C_3 = 0.96147063$$

$$C_5 = 0.59897865$$

DLOG and DLOG 10

$$C_1 = 2.8853900$$

$$C_3 = 0.96180076$$

$$C_5 = 0.57658434$$

$$C_7 = 0.43425975$$

(continued next page)

The final computation is:

$$\text{ALOG and DLOG:} \qquad \log_e X = (\log_2 X) (\log_e 2)$$

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

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.

<u>Low-Order Bits</u>	<u>Quadrant</u>	<u>Modified Value (Z)</u>
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:

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

(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, $Z = \text{arg}$ and:

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

$n = 3$ for ATAN and ATAN2

$n = 7$ for DATAN and DATAN2

For arguments greater than 1, $Z = 1/\text{arg}$ and:

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

$n = 3$ for ATAN and ATAN2

$n = 8$ for DATAN and DATAN2

The values of C are given below.

ATAN and ATAN2

$$C_1 = 0.9992150$$

$$C_3 = -0.3211819$$

$$C_5 = 0.1462766$$

$$C_7 = -0.0389929$$

DATAN and DATAN2

$$C_1 = 0.9999993329$$

$$C_3 = -0.3332985605$$

$$C_5 = 0.1994653599$$

$$C_7 = -0.1390853351$$

$$C_9 = 0.0964200441$$

$$C_{11} = -0.0559098861$$

$$C_{13} = 0.0218612288$$

$$C_{15} = -0.0040540580$$

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)).

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:

<u>Value of i</u>	<u>Value of C_i</u>
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 subsection. 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	$\sin(\text{ARG})$.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	$\tan^{-1}(\text{ARG})$.ED .DD	R=.ED(R) DP=.DD(DP)	Same	None	26 34	Same
Logarithm (base 2) Computation	$\log_2 \text{ARG}$.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	$\text{VAR} = \sum_{i=0}^n C_i Z^{2i+1}$ $\text{VAR} = \sum_{i=0}^n C_i Z^{2i+1}$.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 ₀ / 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	<p>Calling Routine</p> <p>SUBR CAL 0 JMS* .DA JMP .+n+1 (address of ARG1) (address of ARG2) . . . (address of ARGn)</p> <p>Is Called By</p> <p>JMS* SUBR JMP .+n+1 .DSA ARG1 .DSA ARG2</p>	None	N.A	None

Then:

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

$$n = 2 \text{ (.EE)}$$

$$n = 3 \text{ (.DE)}$$

The values of C are:

<u>.EE</u>
$C_1 = 2.8853913$
$C_3 = 0.96147063$
$C_5 = 0.59897865$

<u>.DE</u>
$C_1 = 2.8853900$
$C_3 = 0.96180076$
$C_5 = 0.57658434$
$C_7 = 0.43425975$

3.3.2 Polynomial Evaluator (.EC, .DC)

A polynomial is evaluated as:

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

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.

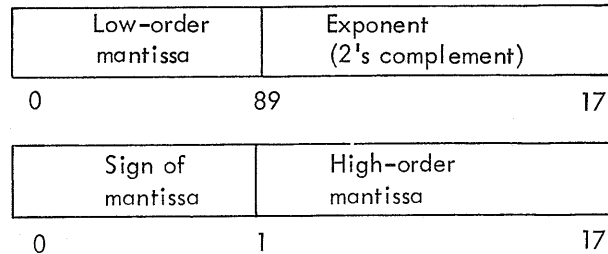
The three major routines of the arithmetic package are INTEAE, RELEAE, and DOUBLE. INTEAE contains integer arithmetic routines; RELEAE, real and floating arithmetic; and DOUBLE, double-precision arithmetic.

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 XVM 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 XVM words and have the format shown below. (Negative mantissae are indicated by a change of sign.)

The Science Library

<u>Held AC Labels</u>	<u>Floating AC Labels</u>					
CE01	.AA	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2" style="text-align: center;">Exponent (2's complement)</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">17</td> </tr> </table>	Exponent (2's complement)		0	17
Exponent (2's complement)						
0	17					
CE02	.AB	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; text-align: center;">Sign of mantissa</td> <td style="width: 50%; text-align: center;">High-order mantissa</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1 17</td> </tr> </table>	Sign of mantissa	High-order mantissa	0	1 17
Sign of mantissa	High-order mantissa					
0	1 17					
CE03	.AC	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2" style="text-align: center;">Low order mantissa</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">17</td> </tr> </table>	Low order mantissa		0	17
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:



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

<u>Flag</u>	<u>Meaning</u>	<u>Action</u>
non-0 positive value	overflow - an attempt to store a REAL constant whose binary exponent is greater than 377_8	\pm largest representable real value stored (DOS);
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_8) in the general floating divide routine (.CI). The result of the division is \pm 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 Science Library

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:

<u>I</u>	<u>Action</u>	<u>Value</u>
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) and flag unchanged.

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 \neq 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 1

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

INTEAE

DOUBLE

¹FPP versions require only Integer Arithmetic (INTEAE).

(continued next page)

Table 3-4 (Cont)
Arithmetic Package

Function	Definition	Symbolic Name	Mode	Calling Sequence	External Calls
Real Arithmetic (includes floating)	N.A	.AG	R=.AG(R)	<p>ARG1 FL.AC</p> <p>ARG2 address address addend subtrahend minuend</p> <p>JMS* SUBR .DSA ARG2</p> <p>multiplier divisor dividend</p>	
Load	N.A	.AH	R=.AH(R)		
Store	N.A	.AI	R=R+R		
Add	ARG1+ARG2	.AJ	R=R-R		
Subtract	ARG1-ARG2	.AM	R=R-R		
Reverse	ARG2-ARG1				
subtract					
Multiply	ARG1*ARG2	.AK	R=R*R		
Divide	ARG1/ARG2	.AL	R=R/R		
Reverse divide	ARG2/ARG1	.AN	R=R/R		
Floating Arithmetic					
Float	R ← IARG	.AW	R=.AW(I)	<p>A-Register integer</p> <p>FL.AC F.P num F.P num</p> <p>JMS* SUBR</p>	
Fix	I ← RARG	.AX	I=.AX(R)		
Negate	R ← RARG	.BA	R=.BA(R)		
Multiply	ARG1*ARG2	.CA	R=R*R	<p>FL.AC HELD AC</p> <p>multiplier addend value</p> <p>JMS* SUBR</p>	
Add	ARG1+ARG2	.CC	R=R+R		
Normalize	N.A	.CD	R=.CD(R)		
Hold	N.A	.CF	R=.CF(R)		
Sign Control (Footnote 2)	N.A	.CG	R=.CG(R)		
Short get argument	N.A	.CB	R=.CB(R)	<p>CAL 0 SUBR ENTRY-EXIT</p> <p>JMS* .CB</p> <p>CAL 0 STORAGE FOR ARG ADDR</p>	

RELEASE

(continued next page)

Table 3-4 (Cont)
Arithmetic Package

Function	Definition	Symbolic Name	Mode	Calling Sequence	External Calls
Floating Arithmetic (Cont) Divide Round and sign ¹	ARG1/ARG2 N.A	.CI .CH	R=R/R R=.CHR	<p>FL.AC HELD .AC</p> <p>divisor value } dividend } JMS*SUBR² CONST1 CONST2</p>	
Load Store Add Subtract Reverse subtract Multiply Divide Reverse divide	N.A N.A ARG1+ARG2 ARG1-ARG2 ARG2-ARG1 ARG1*ARG2 ARG1/ARG2 ARG2/ARG1	.JG .JH .JI .JJ .JM .JK .JL .JN	DI=.JF (DI) DI=.JH (DI) DI=DI+DI DI=DI-DI DI=DI-DI DI=DI*DI DI=DI/DI DI=DI/DI	<p>ARG1 AC, MQ value augend minuend subtrahend multiplicand dividend divisor</p> <p>ARG2 address address addend subtrahend minuend multiplier divisor dividend</p> <p>JMS* SUBR .DSA ARG2</p>	
Float Fix Negate	R←JARG J←RARG J←JARG	.JW .JX .JA	R=.JW (DI) DI=.JX (DI) DI≠.JA (DI)	<p>AC, MQ Doub. Int. FL.AC F.P. Number F.P. Number</p> <p>JMS* SUBR</p>	.CD, REAL REAL

INT

¹ 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.

² CONST1 and CONST2 are required for both EAE and NON-EAE operations, however, they are not 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).

C
.
.
C
(S
O
B
C
.C

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.

Routine	.SS
Purpose	Calculates the address of an array element
Calling Sequence	.GLOBL .SS JMS* .SS .DSA address of fifth word of array descriptor block LAC or LAC* first subscript value LAC or LAC* second subscript value, omitted if only one subscript LAC or LAC* third subscript value, omitted if only one or two subscripts DAC into location at which subscript is used (This instruction is XCT'd by .SS) (returns at location following DAC, with one's complement of mode of array 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 five-word table with the contents described below.

Utility Routines

Word 1	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 5%; text-align: center;">0</td> <td style="width: 25%; text-align: center;">NDIM -1</td> <td style="width: 50%; text-align: center;">0</td> <td style="width: 20%; text-align: center;">Data Mode</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1 2</td> <td></td> <td style="text-align: center;">16 17</td> </tr> </table>	0	NDIM -1	0	Data Mode	0	1 2		16 17
0	NDIM -1	0	Data Mode						
0	1 2		16 17						
Word 2	Size (in words)								
Word 3	0 - for one-dimensional array Size of first dimension								
Word 4	0 - for one- and two-dimensional arrays Size of the first two dimensions								
Word 5	Address of first word of array								

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 2, the size 2 in word 3, and the size 4 in word 4. A REAL array of the same dimensions will have 16, 4, and 8 in these locations.

NDIM is the number of dimensions in the array.

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

<u>Array Mode</u>	<u>M</u>	<u>N</u>
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:

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

FORTTRAN subprograms maintain an array descriptor block that is local for every array declared, including dummy arrays. Further, the dimensioning information of a dummy array may include integer dummy parameters, such as in

```
SUBROUTINE SUB (X, ARRAY, Y, I, J)
DIMENSION ARRAY (5, I, J)
```

Utility Routines

Such arrays are called adjustable arrays. In systems preceding the XVM, it was necessary to use subroutines ADJ, ADJ1, ADJ2, or ADJ3 to perform this type of operation; it is now handled automatically, by implicitly generated calls to the OTS subroutine .DJ. The MACRO program, if it uses .DA to fetch arguments, will not have access to the ADB in a calling program. It will probably need a local ADB, and to make use of .DJ. Further information about this appears in Section 5.1.

.DJ

Routine	.DJ														
Purpose	Completes dimensioning information in array descriptor block of an adjustable array														
Calling Sequence	<table style="border: none;"> <tr> <td style="padding-right: 20px;">JMS*</td> <td>.DJ</td> </tr> <tr> <td style="padding-right: 20px;">.DSA</td> <td>address of word 5 of array descriptor block</td> </tr> <tr> <td style="padding-right: 20px;">{ .DSA</td> <td>integer value of constant first maximum dimension</td> </tr> <tr> <td style="padding-right: 20px;">.DSA</td> <td>or</td> </tr> <tr> <td style="padding-right: 20px;">.DSA</td> <td>400000 + address of dummy integer used as first maximum dimension</td> </tr> <tr> <td style="padding-right: 20px;">.DSA</td> <td>same for second subscript, if any</td> </tr> <tr> <td style="padding-right: 20px;">.DSA</td> <td>same for third subscript, if any</td> </tr> </table>	JMS*	.DJ	.DSA	address of word 5 of array descriptor block	{ .DSA	integer value of constant first maximum dimension	.DSA	or	.DSA	400000 + address of dummy integer used as first maximum dimension	.DSA	same for second subscript, if any	.DSA	same for third subscript, if any
JMS*	.DJ														
.DSA	address of word 5 of array descriptor block														
{ .DSA	integer value of constant first maximum dimension														
.DSA	or														
.DSA	400000 + address of dummy integer used as first maximum dimension														
.DSA	same for second subscript, if any														
.DSA	same for third subscript, if any														
External Calls	None														
Errors	None														

A .DJ call is generated for each adjustable array in a subprogram. It uses constant and adjustable dimensions to complete the dimensioning information in the array descriptor block, to the format described above.

GO TO

Routine	.GO												
Purpose	Computes index of computed GO TO												
Calling Sequence	<table style="border: none;"> <tr> <td style="padding-right: 20px;">LAC V</td> <td>/ index value in A-register</td> </tr> <tr> <td style="padding-right: 20px;">JMS* .GO</td> <td></td> </tr> <tr> <td style="padding-right: 20px;">-N</td> <td>/ number of statement address</td> </tr> <tr> <td style="padding-right: 20px;">STMT(1)</td> <td></td> </tr> <tr> <td style="padding-right: 20px;">STMT(2)</td> <td></td> </tr> <tr> <td style="padding-right: 20px;">STMT(N)</td> <td></td> </tr> </table>	LAC V	/ index value in A-register	JMS* .GO		-N	/ number of statement address	STMT(1)		STMT(2)		STMT(N)	
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)												

Utility Routines

STOP	Routine	.ST
	Purpose	Processes STOP statement (returns to monitor)
	Calling Sequence	LAC (number) /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 (DOS), waits for operator to RESUME task being executed (RSX).
	Calling Sequence	LAC (number) /octal number JMS* .PA
	External Calls	.SP
	Errors	None
	Routine	.SP
	Purpose	Prints octal number for PAUSE and STOP (DOS). Zero assumed if none supplied. See 4.4 for RSX behavior.
	Calling Sequence	LAC (number) /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

Utility Routines

OTSER
(continued)

Routine	.ZR
Purpose	Initializes END and ERR exits in READ and WRITE statements
Calling Sequence	JMS* .ZR .DSA END address .DSA ERR address (The AC is saved and restored to accommodate Direct Access I/O. Both addresses appear if either END or ERR is coded; if one of the two exits is not specified a zero appears instead of an address. The call sets a gate in OTSER to allow branching to the specified address when an OTS error would have occurred in the absence of the special exit.)
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

Utility Routines

PARTWD (continued)

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 Inter- face 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	Entry point defined by JEA address
	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
	Routine	.ZE
Purpose	Return, in AC, \emptyset if $DI = \emptyset$, 1 if $DI > \emptyset$, -1 if $DI < \emptyset$.	

4.3 FORTRAN-CALLABLE UTILITY ROUTINES

These routines are described in Table 4-1.

4.4 RSX LIBRARY (.LIBRX BIN OR .LIBFX BIN) ROUTINES

A special set of routines is provided for use with XVM/RSX. These libraries include, in addition to the subprograms described previously, numerous FORTRAN-callable external subroutines for operations peculiar to RSX. For further details refer to the XVM/RSX System Manual.

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 ¹	Records elapsed time in minutes and seconds on 60hz or 50hz machines	CALL TIME(IMIN,ISEC,IOFF) Where: IMIN = minutes ISEC = seconds IOFF = zero to start clock (Later setting IOFF to non-zero stops clock)	CALL TIME(IM,IS,IOF) A . IOF = 1 WRITE(4,100)IM,IS [outputs time to execute A]	.DA .TIMER	None
	TIME10 ¹	Records elapsed time in minutes, seconds, and tenths of seconds on 60hz or 50hz machines	CALL TIME10(IMIN,ISEC,IOTSE) Where: IMIN = minutes ISEC = seconds IOTSE = tenths of seconds IOFF = zero to start 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 = 75. If N ≤ 0, no messages output.			

¹ Not supported with RSX. Other RSX supplied routines can be used for this purpose.

(continued next page)

Utility Routines

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 - 000001 - TSKNAM

The PAUSE message is always output, but the STOP message is output only if its argument is non-zero.

STOP - uses RSX EXIT 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:

Routine	.FTSB
Purpose	To convert two words from .ASCII to .SIXBT
.ASCII to .SIXBT Conversion	Calling Sequence:
	SUBA 0 JMS* .DAA / get call args JMP ARGEND
	FROM 0 / Pointer to ASCII word-pair
	ARGEND JMS* .FTSB .DSA FROM .DSA TO . . .
	TO .BLOCK 2 / two .SIXBT 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		
C		.TITLE	MIN
C	TEST MACRO SUBR	.GLOBL	MIN, .DA
C	READ A NUMBER (A)	MIN	0 /ENTRY & EXIT
C		JMS*	.DA /GENERAL GET
1	READ (1,100) A		/ARGUMENT.
100	FORMAT (E12.4)	JMP	+.2+1 /JUMP AROUND
C			/THE ARGUMENTS.
C	NEGATE THE NUMBER	MIN1	.DSA 0 /ARG1
C	AND PUT IT IN B	MIN2	.DSA 0 /ARG2
C		LAC*	MIN1 /FIRST WORD OF A
C	CALL MIN (A,B)	DAC*	MIN2 /STORE AT B
C		ISZ	MIN1 /POINT TO SECOND WORD
C	WRITE OUT NUMBER (B)	ISZ	MIN2 /OF A AND B.
C		LAC*	MIN1 /SECOND WORD OF A.
C	WRITE (2,100) B	RAL	/SIGN BIT = 1.
C	STOP	CML	/
C	END	RAR	/
		DAC*	MIN2 /STORE IN SECOND
			/WORD OF B.
		JMP*	MIN /EXIT
		.END	

FORTRAN-IV and MACRO

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 1-17 contain the 17-bit address of the first word. 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 first word in the array is given. If the MACRO program needs to know the array dimensions, either they must be passed as parameters or .DA must not be used. The parameter list entry in a calling FORTRAN program for an array reads .DSA 400000 +address of last word of the Array Descriptor Block (ADB). From this, if .DA is not used at the called entry point, the address of the ADB can be obtained.

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  MACPRG
.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.

FORTRAN-IV and MACRO

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.

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.

DOS and RSX MACRO programs may also use the .CBD pseudo-op. For instance

```
BASE1 .CBD NAME, 1
```

will provide the base address of the common block NAME in the word that is created and labeled BASE1; the size of the common block is 1. For blank common (which is given the special system name of .XX), use for example:

```
BASE2 .CBD .XX, 2
```

C

.

.

0000

0

000

.

.

C

APPENDIX A
FORTRAN LANGUAGE SUMMARY

A.1 EXPRESSION OPERATORS

Operators in each type are shown in order of descending precedence.

Type	Operator	Operates Upon	
Arithmetic	** - *,/ +,-	exponentiation unary minus multiplication, division, addition and sub- traction (but not unary minus)	arithmetic or logical constants, variables, array elements, function references and expressions
Relational	.GT. .GE. .LT. .LE. .EQ. .NE.	greater than greater than or equal to less than less than or equal to equal to not equal to	arithmetic or logical constants, variables, array elements, function references and expres- sions (all relational operators have equal priority)
Logical	.NOT. .AND. .OR. .XOR.	.NOT.A is true if and only if A is false A.AND.B is true if and only if A and B are both true A.OR.B is true if and only if either A or B or both are true A.XOR.B is true if and only if A is true and B is false or B is true and A is false	logical or integer constants, variables, array elements, function references and expres- sions

FORTRAN Language Summary

A.2 STATEMENTS

The following summary of statements available in the FORTRAN IV XVM language defines the general format for the statement. If more detailed information is needed, the reader is referred to the Section(s) in the FORTRAN IV XVM Language Manual dealing with that particular statement.

<u>Statement Formats</u>	<u>Effect</u>	<u>FORTRAN IV XVM Language Manual Section</u>
Arithmetic/Logical Assignment		
$v_1=v_2=\dots=v_n=e$		3.1
v_i	is a variable name or an array element name	
e	is an expression	
	The value of the arithmetic or logical expression is assigned to each variable, from right to left.	
Arithmetic Statement Function		
$f(p[,p]...)=e$		8.1.1
f	is a symbolic name	
p	is a symbolic name	
e	is an arithmetic expression	
	Creates a user-defined function having the variables p as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call.	
ASSIGN s TO v		
		3.3
s	is an executable statement label	
v	is an integer variable name	
	Associate the statement number s with the integer variable v for later use in an assigned GO TO statement.	

FORTRAN Language Summary

FORTRAN IV XVM
Language Manual
Section

<u>Statement Formats</u>	<u>Effect</u>	
BACKSPACE u		6.8.2
u	is an integer variable or constant	
	The currently open file on logical unit number u is backspaced one record.	
BLOCK DATA		8.1.5
	Specifies the subprogram which follows as a BLOCK DATA subprogram.	
CALL s[(a[,a]...)]		4.5
s	is a subprogram name	
a	is an expression, a procedure name, or an array name	
	Calls the SUBROUTINE subprogram with the name specified by s, passing the actual arguments a to replace the dummy arguments in the SUBROUTINE definition.	
COMMON [/[cb]/] nlist [/[cb]/ nlist] ...		5.4
cb	is a common block name	
nlist	is a list of one or more variable names, array names, or array declarators separated by commas.	
	Reserves one or more blocks of storage space under the name specified to contain the variables associated with that block name.	
CONTINUE		4.4
	Causes no processing, and is most often used to terminate DO loops.	
DATA nlist/clist[,nlist/clist/] ...		5.7
nlist	is a list of one or more variable names, array names, or array element names separated by commas. Subscript expressions must be constant.	
clist	is a list of one or more constants separated by commas, each optionally preceded by j*, where j is a nonzero, unsigned integer constant.	
	Causes elements in the list of values to be initially stored in the corresponding elements of the list of variable names.	

<u>Statement Formats</u>	<u>Effect</u>	<u>Section</u>
<p>DECODE (c,a[f[,ERR=s])[list]</p> <p style="margin-left: 2em;">c is an integer expression</p> <p style="margin-left: 2em;">a is an array name</p> <p style="margin-left: 2em;">f is a FORMAT statement label or array name</p> <p style="margin-left: 2em;">s is a statement label</p> <p style="margin-left: 2em;">list is an I/O list</p> <p style="margin-left: 4em;">Changes the elements in the I/O list from ASCII into the desired internal format; c specifies the number of characters, f specifies the format, and a is the name of an array containing the ASCII characters to be converted.</p>		6.9
<p>DIMENSION a(d)[,a(d)]...</p> <p style="margin-left: 2em;">a(d) is an array declarator</p> <p style="margin-left: 4em;">Specifies storage space requirements for arrays.</p>		5.3
<p>DO s i = v1 ,v2[,[-]v3]</p> <p style="margin-left: 2em;">s is the label of an executable statement</p> <p style="margin-left: 2em;">i is an integer variable name</p> <p style="margin-left: 2em;">vn are integer expressions</p> <p style="margin-left: 4em;">1. Set i = v1</p> <p style="margin-left: 4em;">2. Execute statements through statement number s</p> <p style="margin-left: 4em;">3. Evaluate i = i+v3</p> <p style="margin-left: 4em;">4. Repeat 2 through 3 for INT((v2-v1)/v3) iterations</p>		4.3
<p>ENCODE (c,a,[f[,ERR=s])[list]</p> <p style="margin-left: 2em;">c is an integer expression</p> <p style="margin-left: 2em;">a is an array name</p> <p style="margin-left: 2em;">f is a FORMAT statement label or an array name</p> <p style="margin-left: 2em;">s is a statement label</p> <p style="margin-left: 2em;">list is an I/O list</p>		6.9

FORTRAN Language Summary

FORTRAN IV XVM
Language Manual
Section

<u>Statement Formats</u>	<u>Effect</u>	
ENCODE (cont.)	Changes the elements in the list of variables into ASCII format; c specifies the number of characters in the buffer, f specifies the format statement number, and a is the name of the array to be used as a buffer.	
END	Specifies the physical end of a program unit.	4.9
ENDFILE u		6.8.3
u	is an integer variable or constant	
	An end-file record is written on logical unit u, following output statements to that unit.	
ENTRY nam(p[,p]...)		8.1.4
nam	is a symbolic name	
p	is a symbolic name	
	Defines an alternate entry point within a SUBROUTINE or FUNCTION subprogram.	
EQUIVALENCE (nlist)[, (nlist)]...		5.5
nlist	is a list of two or more variable names, array names, or array element names separated by commas. Subscript expressions must be constant.	
	Each of the names (nlist) within a set of parentheses is assigned beginning at the same storage location.	
EXTERNAL v[,v]...		5.6
v	is a procedure name	
	Informs the system that the names specified are those of FUNCTION or SUBROUTINE programs.	
EXTERNAL v[,v]...		5.6
v	is a procedure name	
	Informs the system that the names specified are user-defined.	

<u>Statement Formats</u>	<u>Effect</u>	<u>Section</u>
<p>FORMAT (field specification, ...)</p>	<p>Describes the format in which one or more records are to be transmitted; a statement label must be present.</p>	<p>7.1 - 7.8</p>
<p>[typ] FUNCTION nam(p[, p]...)</p> <p style="margin-left: 2em;">typ is a type specifier</p> <p style="margin-left: 2em;">nam is a symbolic name</p> <p style="margin-left: 2em;">p is a symbolic name</p>	<p>Begins a FUNCTION subprogram, indicating the program name and any dummy argument names, p. An optional type specification can be included.</p>	<p>8.1.2</p>
<p>GO TO s</p> <p style="margin-left: 2em;">s is an executable statement label</p>	<p>(Unconditional GO TO) Transfers control to statement number s.</p>	<p>4.1.1</p>
<p>GO TO (slist), v</p> <p style="margin-left: 2em;">slist is a list of one or more executable statement labels separated by commas.</p> <p style="margin-left: 2em;">v is an integer variable</p>	<p>(Computed GO TO) Transfers control to the statement label specified by the value v. (If v=1 control transfers to the first statement label. If v=2 it transfers to the second statement label, etc.) If v is less than 1 or greater than the number of statement labels present, no transfer takes place.</p>	<p>4.1.2</p>
<p>GO TO v[, (slist)]</p> <p style="margin-left: 2em;">v is an integer variable name</p> <p style="margin-left: 2em;">slist is a list of one or more executable statement labels separated by commas</p>	<p>(Assigned GO TO) Transfers control to the statement most recently associated with v by an ASSIGN statement.</p>	<p>4.1.3</p>

FORTRAN Language Summary

FORTRAN IV XVM
Language Manual
Section

<u>Statement Formats</u>	<u>Effect</u>	
IF (e) v1, v2, v3		4.2.1
e	is an expression	
v1	are executable statement labels or variables to which statement labels have been ASSIGNED	
	(Arithmetic IF) Transfers control to statement number v1 depending upon the value of the expression. If the value of the expression is less than zero, transfer to v1; if the value of the expression is equal to zero, transfer to v2; if the value of the expression is greater than zero, transfer to v3.	
IF (e) st		4.2.2
e	is an expression	
st	is any executable statement except a DO or a logical IF statement	
	(Logical IF) Executes the statement if the logical expression is true.	
IMPLICIT typ (a[,a]...)[,typ(a[,a]...)]...		5.1
typ	is a data type specifier	
a	is either a single letter, or two letters in alphabetical order separated by a dash (i.e., x-y)	
	The elements a represent single (or a range of) letter(s) whose presence as the initial letter of a variable specifies the variable to be of that type.	
PAUSE [disp]		4.7
disp	is an octal integer constant	
	Suspends program execution and prints the display, if one is specified.	
PRINT	See WRITE for which PRINT is a synonym.	6.4.5
READ (u, [f][, END=s][, ERR=s])[list]		6.4.1
u	is an integer variable or constant	
f	is a FORMAT statement label or an array name	
s	is an executable statement label	

Statement Formats

Effect

READ (u, [f][, END=s][, ERR=s])[list] (cont.)

list is an I/O list

(Formatted Sequential) Reads at least one logical record from device u according to format specifications f and assigns values to the variables in the optional list.

READ (u'r, [f][, ERR=s])(list)

6.6.1

u is an integer variable or constant

r is an integer expression

f is a FORTRAN statement label or an array name

s is an executable statement label

list is an I/O list

(Formatted Direct Access READ) Reads record number r from unit u and assigns values to the elements of the list according to format f.

READ(u[, END=s][, ERR=s])[list]

6.3.1

u is an integer variable or constant

s is an executable statement label

list is an I/O list

(Unformatted Sequential READ) Reads one unformatted record from device u, assigning values to the variables in the optional list.

READ(u'r[, ERR=s])[list]

6.5.1

u is an integer variable or constant

r is an integer expression

s is an executable statement label

list is an I/O list

(Unformatted Direct Access READ) Reads record r from logical unit u, assigning values to the variables in the optional list.

FORTRAN Language Summary

FORTRAN IV XVM
Language Manual
Section

<u>Statement Formats</u>	<u>Effect</u>	
RETURN [v]		4.6
v	is an integer variable	
	Returns control to the calling program from the current subprogram. If v is specified, control is returned to the statement label associated with v in the subprogram call.	
REWIND u		6.8.1
u	is an integer variable or constant	
	Repositions logical unit number u to the beginning of the physical medium or to the currently opened file.	
STOP [disp]		4.8
disp	is an octal integer constant	
	Terminate program execution and print the display, if one is specified.	
SUBROUTINE nam[(p[,p]...)]		8.1.3
nam	is a symbolic name	
p	is a symbolic name	
	Begins a SUBROUTINE subprogram, indicating the program name and any dummy argument names, p.	
TYPE	See WRITE, for which TYPE is a synonym.	
Type Declaration		
typ v[,v]...		5.2
typ	is a data type specifier, one of:	
	DOUBLE PRECISION REAL DOUBLE INTEGER INTEGER LOGICAL	
v	is a variable name, an array name, a function or function entry name, or an array declarator.	

<u>Statement Formats</u>	<u>Effect</u>	
typ v[,v]... (cont.)	(Type Declarations) The symbolic names, v, are assigned the specified data type in the program unit.	
WRITE (u,[f[,ERR=s)](list)		6.4.2
u	is an integer variable or constant	
f	is a FORMAT statement label or an array name	
s	is an executable statement label	
list	is an I/O list	
	(Formatted sequential WRITE) Causes one or more logical records containing the values of the variables in the optional list to be written onto device u, according to the format specification f.	
WRITE (u'r,[f[,ERR=s)](list)		6.6.2
u	is an integer variable or constant	
r	is an integer expression	
f	is a FORMAT statement label or an array name	
s	is an executable statement label	
list	is an I/O list	
	(Formatted Direct Access WRITE) Causes a record formed from the list and format f to be written onto record r of unit u.	
WRITE (u[,ERR=s)](list)		6.3.2
u	is an integer variable or constant	
s	is an executable statement label	
list	is an I/O list	
	(Unformatted Sequential WRITE) Causes one unformatted record containing the values of the variables in the optional list to be written onto device u.	

<u>Statement Formats</u>	<u>Effect</u>	<u>Section</u>
WRITE (u'r[,ERR=s]) [list]		6.5.2

u is an integer variable or constant

r is an integer expression

s is an executable statement label

list is an I/O list

(Unformatted Direct Access WRITE) Causes a record containing the values of the variables in the list to be written onto record r of logical unit u.

END=s,ERR=s		6.7
-------------	--	-----

(Transfer of control on end-of-file or error condition) Is an optional element in each type of I/O statement allowing the program to transfer to statement number s on an end-of-file (END=) or error (ERR=) condition.

FORTRAN Language Summary

Table A-1
FORTRAN Library Functions

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
ABS(X) IABS(I) DABS(X) JABS(I)	Real absolute value Integer absolute value Double precision absolute value Double Integer absolute value	Real Integer Double P Double I	Real Integer Double P Double I
FLOAT(I) IFIX(X) SNGL(X) DBLE(X) JFIX(X) JFIX(X) ISNGL(I) JDBLE JDFIX(X) FLOATJ(I) DBLEJ(I)	Integer to Real conversion Real to Integer conversion IFIX(X) is equivalent to INT(X) Double precision to Real conversion Real to Double precision conversion Real to Double integer conversion Double precision to Double integer conversion Double integer to integer conversion Integer to Double integer Double precision to Double integer conversion Double integer to Real conversion Double integer to Double precision conversion	Integer Real Double P Real Real Double P Double I Integer Double P Double I Double I Double P Double I Double I	Real Integer Real Double P Double I Double I Integer Double I Double I Real Double P
AIN(X) INT(X) IDINT(X) JINT(X) JDINT(X)	Truncation functions return the sign of the argument * largest integer $\leq arg $ Real to Real truncation Real to Integer truncation Double precision to Integer truncation Real to Double integer truncation Double precision to Double integer truncation	 Real Real Double P Real Double P	 Real Integer Integer Double I Double I
AMOD(X,Y) MOD(I,J) DMOD(X,Y) JMOD(I,J)	Remainder functions return the remainder when the first argument is divided by the second. Real remainder ($x/y < 131072$) Integer remainder Double precision remainder ($x/y < 131072$) Double integer remainder ($I/J < 131072$)	 Real Integer Double P Double I	 Real Integer Double P Double I
AMAX(I,J,...) AMAX1(X,Y,...) MAX(I,J,...) MAX1(X,Y,...)	Maximum value functions return the largest value from among the argument list; ≥ 2 arguments. Real maximum from Integer list Real maximum from Real list Integer maximum from Integer list Integer maximum from Real list	 Integer Real Integer Real	 Real Real Integer Integer

FORTRAN Language Summary

Table A-1 (Cont.)
FORTRAN Library Functions

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
DMAXI(X,Y,...) JMAXØ(I,J,...)	Double precision maximum from Double precision list Double integer maximum from Double integer list	Double P Double I	Double P Double I
AMINØ(I,J,...) AMINI(X,Y,...) MINØ(I,J,...) MINI(X,Y,...) DMINI(X,Y,...) JMINØ(I,J,...)	Minimum value functions return the smallest value from among the argument list; ≥ 2 arguments. Real minimum of Integer list Real minimum of Real list Integer minimum of Integer list Integer minimum of Real list Double minimum of Double list Double integer minimum of Double integer list	Integer Real Integer Real Double P Double I	Real Real Integer Integer Double P Double I
SIGN(X,Y) ISIGN(I,J) DSIGN(X,Y) JSIGN(I,J)	The transfer of sign functions return (sign of the second argument) * (absolute value of the first argument). Real transfer of sign Integer transfer of sign Double precision transfer of sign Double integer transfer of sign	Real Integer Double P Double I	Real Integer Double P Double I
DIM(X,Y) IDIM(I,J) JDIM(I,J)	Positive difference functions return the first argument minus the minimum of the two arguments. Real positive difference Integer positive difference Double integer positive difference	Real Integer Double I	Real Integer Double I
EXP(X) DEXP(X)	Exponential functions return the value of e raised to the argument power. e^x ($x \geq \emptyset$) e^x ($x \geq \emptyset$)	Real Double P	Real Double P
ALOG(X) ALOGØ(X) DLOG(X) DLOGØ(X)	Returns $\log_e(X)$ Returns $\log_{10}(X)$ Returns $\log_e(X)$ Returns $\log_{10}(X)$ } $x \geq \emptyset$	Real Real Double P Double P	Real Real Double P Double P
SQRT(X) DSQRT(X)	Square root of Real argument ($x \geq \emptyset$) Square root of Double precision argument ($x \geq \emptyset$)	Real Double P	Real Double P
SIN(X) DSIN(X)	Real sine Double precision sine	Real Double P	Real Double P

FORTRAN Language Summary

Table A-1 (Cont.)
FORTRAN Library Functions

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
COS(X) DCOS(X)	Real cosine Double precision cosine	Real Double P	Real Double P
TANH(X)	Hyperbolic tangent	Real	Real
ATAN(X) DATAN(X) ATAN2(X, Y) DATAN2(X, Y)	Real arc tangent Double precision arc tangent Real arc tangent of (X/Y) Double precision arc tangent of (X/Y)	Real Double P Real Double P	Real Double P Real Double P

APPENDIX B
ERROR MESSAGES

B.1 COMPILER ERROR MESSAGES

Compiler error messages are printed in the form:

>mnA<

where:

mn is the error number

A is the alphabetic mnemonic characterizing the error class.

All error messages 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)

Error Messages

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 or simple dummy 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	Missing left parenthesis

(continued on next page)

Error Messages

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:		
02	H	More than two characters in Integer or logical Hollerith constant
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)

Error Messages

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	Local array length larger than 8K
04	M	Element position in local array larger than 8K or in common array larger than 32K (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
09	N	Invalid statement label or continuation

(continued on next page)

Error Messages

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	Adjustable dimension not in dummy array
13	S	Adjustable dimension not a dummy integer
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
04	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)

Error Messages

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 exits 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)

Error Messages

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
31	X	Illegal operator sequence
32	X	Illegal use of =.

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 Messages

	12 (T)	Bad FORMAT	.FA, .FE, .FF
	13 (T)	Negative or zero REAL logarithmic argument	.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
	16 (R)	ATAN2 ($\emptyset, \emptyset, \emptyset, \emptyset$) attempted; PI/2 returned	ATAN2
	17 (R)	DATAN2 ($\emptyset, \emptyset, D\emptyset, \emptyset, \emptyset D\emptyset$) attempted; PI/2 returned	DATAN2
	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)	Too many records per file or illegal record number	DEFINE, RANCOM
	25 (T)	Mode discrepancy	RANCOM
	26 (T)	Too many open files	DEFINE
	30 (R)	Single integer overflow ¹	RELEASE, .FPP
	² 31 (R)	Extended (double) integer overflow ⁴	DBLINT, JFIX, JDFIX, ISNGL
	² 32 (R)	Single floating point overflow	RELEASE
	² 33 (R)	Double floating point overflow [†]	
	² 34 (R)	Single floating point underflow	RELEASE
	² 35 (R)	Double floating point underflow [†]	
	² 36 (R)	Floating point divide check	RELEASE
	³ 37 (R)	Integer divide check	INTEAE
	40 (T)	Illegal number of characters specified [legal: $0 < c < 625$]	ENCODE
	41 (R)	Array exceeded	ENCODE
	42 (T)	Bad input data	DD10
	² 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
	52 (T)	Attempt to initialize JEA register on a machine without floating point hardware	

¹Only detected when fixing a floating point number.

²Also prints out PC with FPP system

³If extended integer divide check, prints out PC with FPP system.

⁴With non-floating Point Processor system, only detected when fixing a floating point number.

[†]Not detected by software floating point routines (only by FPP system).

Error Messages

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, a value is assumed as the final result of the computation. Where a "none" value is indicated, the results are meaningless. Results differ depending upon whether or not floating point hardware is used.

Error	ASSUMED VALUE	
	FPP Hardware	FPP Software
Single Floating Overflow (.OTS 32)	± largest single floating value	±largest single floating value
Double Floating Overflow (.OTS 33)	± largest single floating value	not detected
Single Floating Underflow (.OTS 34)	zero	zero
Double Floating Underflow (.OTS 35)	zero	not detected
Floating Divide Check (.OTS 36)	± largest single floating value	±largest single floating value
Integer Overflow (.OTS 30)	limited detection ¹	limited detection
Double Integer Overflow (.OTS 31)	none ²	limited detection ¹
Integer Divide Check (.OTS 37)	none	none

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.

¹When 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 assumed as the result.

²With the FPP hardware all extended double integer overflow conditions are detected, but the results are meaningless.

C
.
c
C
O
B
c
C

APPENDIX C
PROGRAMMING EXAMPLES

C.1 A FUNCTION TO READ THE AC SWITCHES

It is frequently desirable to use the AC switches of the XVM CPU 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 CALLING SEQUENCE
/      .GLORL  ITOG
/      JMS*   ITOG
/      JMP    .+2      /JUMP OVER ARGUMENT
/      .DSA   (MASK)  /ADDRESS OF MASK
/                      /RETURN WITH MASKED ACS IN AC.
ITOG  .GLORL  ITOG..DA
      0      /INTEGER FUNCTION
      JMS*   .DA      /GET ARGUMENTS
      JMP    .+1+1    /ONE ARGUMENT
MASK  0      /MASK ADDRESS
      LAS    /LOAD AC FROM SWITCHES
      AND*   MASK     /MASK AC
      JMP*   ITOG     /RETURN WITH MASKED AC SWITCHES
      .END

```

C.2 IFLOW AND IDZERO EXAMPLES

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

```

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

```

(continued on next page)

Programming Examples

```
3      PAUSE 2
      C = A/O.
      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 CHECKING.
C
      SUBROUTINE CHECK (N)
      LOGICAL IFLOW, IDZERO
      IF (IFLOW(N)) WRITE (1,10)
      IF (IFLOW(-N)) WRITE (1,11)
      IF (IDZERO(N)) WRITE (1,12)
10     FORMAT (/9H OVERFLOW)
11     FORMAT (/10H UNDERFLOW)
12     FORMAT (/13H DIV. BY ZERO)
      RETURN
      END
```

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

NOTE

The "OTS nn" and "PC=nnnnnn" lines are not typed on systems which do not have hardware floating point.

```
.OTS 32
PC=077522
```

```
OVERFLOW
PAUSE 000001
CP .OTS 34
PC=077553
```

```
UNDERFLOW
PAUSE 000002
CP .OTS 36
PC=077564
```

```
DIV. BY ZERO
PAUSE 000003
CP
STOP 000000
```

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).

Programming Examples

```
0001      C
0002      C      PROGRAM EXAMPLE TO SHOW OBJECT CODE OUTPUT FOR
0003      C      VARIOUS TYPES OF I/O STATEMENTS.
0004      C
0005      C      IMPLICIT REAL (N)
0006      C      DIMENSION RL1(2),RL2(3),ARR(20),NM1(2),NM2(2)
0007      C      DATA NM1/'NAME1','ASRC'/,NM2/'NAME2','ASRC'/
00046 472031 542542
00050 406472 241500
00057 472031 542544
00051 406472 241500
0008      C
0009      C      100      FORMAT (15,G10.3,2(E12.2))
00000      JMP      S00000
00001      .DSA 242226
00002      .DSA 526216
00003      .DSA 305405
00004      .DSA 631530
00005      .DSA 311210
00006      .DSA 530544
00007      .DSA 271445
00010      .DSA 124500
S00000 = 00011
0010      C      200      FORMAT (1X,15,G10.3,2(E12.2))
00011      JMP      S00011
00012      .DSA 241433
00013      .DSA 026222
00014      .DSA 325310
00015      .DSA 730540
00016      .DSA 271465
00017      .DSA 431120
00020      .DSA 425425
00021      .DSA 227144
00022      .DSA 245224
00023      .DSA 020100
S00011 = 00024
0011      C      CALL DEFINE (2,100,5,5,JVB,0,0,0)
00024      JMS* DEFINE
00025      JMP      00036
00026      .DSA (000002
00027      .DSA (000144
00030      .DSA (000005
00031      .DSA (000005
00032      .DSA JVB
00033      .DSA (000000
00034      .DSA (000000
00035      .DSA (000000
0012      C      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
```

Programming Examples

```
00046 .DSA (000000
00047 .DSA (000000
0013      CALL SEEK (5,MM1)
00050 JMS* SEEK
00051 JMP 00054
00052 .DSA (000005
00053 .DSA 400000 +MM1
0014      CALL ENTER (6,MM2)
0015      C
0016      C      1.  BINARY
0017      C      A.  DIRECT ACCESS
0018      C
00054 JMS* ENTER
00055 JMP 00060
00056 .DSA (000006
00057 .DSA 400000 +MM2
0019      READ (2'JVB) INT,RL2(3),RL1
00060 LAC JVB
00061 JMS* .RS
00062 .DSA (000002
00063 .DSA 777777
00064 JMS* .RJ
00065 .DSA INT
00066 CMA!CLA
00067 TAD (000003
00070 RCL
00071 TAD RL2
00072 DAC $00072
00073 .DSA 77777b
00074 JMS* .RJ
$00072 = 00075
00075 .DSA $00075
00076 JMS* .RB
00077 .DSA RL1
00100 JMS* .RG
0020      WRITE (2'3) INT,RL2(3),RL1
00101 LAC (000003
00102 JMS* .RX
00103 .DSA (000002
00104 .DSA 777777
00105 JMS* .RJ
00106 .DSA INT
00107 CMA!CLA
00110 TAD (000003
00111 RCL
00112 TAD RL2
00113 DAC $00113
00114 .DSA 77777b
00115 JMS* .RJ
$00113 = 00116
00116 .DSA $00116
0021      C
0022      C      B.  SEQUENTIAL ACCESS
0023      C
00117 JMS* .RB
00120 .DSA RL1
```

Programming Examples

```
00121 JMS* .RG
0024      READ (1) INT,RL2(3),RL1
00122 JMS* .FS
00123 .DSA (000001
00124 .DSA 777777
00125 JMS* .FJ
00126 .DSA INT
00127 CHA!CLA
00130 TAD (000003
00131 RCL
00132 TAD RL2
00133 DAC S00133
00134 .DSA 777776
00135 JMS* .FJ
S00133 = 00136
00136 .DSA S00136
00137 JMS* .FB
00140 .DSA RL1
00141 JMS* .FG
0025      WRITE (3) INT,RL2(3),RL1
00142 JMS* .FX
00143 .DSA (000003
00144 .DSA 777777
00145 JMS* .FJ
00146 .DSA INT
00147 CHA!CLA
00150 TAD (000003
00151 RCL
00152 TAD RL2
00153 DAC S00153
00154 .DSA 777776
00155 JMS* .FJ
S00153 = 00156
00156 .DSA S00156
0026 C
0027 C      11. ASCII
0028 C      A. DIRECT ACCESS
0029 C      1. FORMATTED
0030 C
00157 JMS* .FB
00160 .DSA RL1
00161 JMS* .FG
0031      READ (4'JVA,100) INT,RL2(3),RL1
00162 LAC JVA
00163 JMS* .RR
00164 .DSA (000004
00165 .DSA .100
00166 .DSA 777777
00167 JMS* .RE
00170 .DSA INT
00171 CHA!CLA
00172 TAD (000003
00173 RCL
00174 TAD RL2
00175 DAC S00175
00176 .DSA 777776
```

Programming Examples

```
00177 JMS* .RE
S00175 = 00200
00200 .DSA S00200
00201 JMS* .RA
00202 .DSA RL1
00203 JMS* .RF
0032 WRITE (4'5,200) INT,RL2(3),RL1
00204 LAC (000005
00205 JMS* .RW
00206 .DSA (000004
00207 .DSA .200
00210 .DSA 777777
00211 JMS* .RE
00212 .DSA INT
00213 CMAICLA
00214 TAD (000003
00215 RCL
00216 TAD RL2
00217 DAC S00217
00220 .DSA 777776
00221 JMS* .RE
S00217 = 00222
00222 .DSA S00222
0033 C
0034 C 2. DATA DIRECTED
0035 C
00223 JMS* .RA
00224 .DSA RL1
00225 JMS* .RF
0036 READ (4'7,) INT,RL2(3),RL1
00226 LAC (000007
00227 JMS* .RR
00230 .DSA (000004
00231 .DSA 000000
00232 .DSA 777777
00233 JMS* .GD
00234 .DSA INT
00235 CMAICLA
00236 TAD (000003
00237 RCL
00240 TAD RL2
00241 DAC S00241
00242 .DSA 777776
00243 JMS* .GD
S00241 = 00244
00244 .DSA S00244
00245 JMS* .GE
00246 .DSA RL1
00247 JMS* .RF
0037 WRITE (4'8,) INT,RL2(3),RL1
00250 LAC (000010
00251 JMS* .RW
00252 .DSA (000004
00253 .DSA 000000
00254 .DSA 777777
00255 JMS* .GA
```

Programming Examples

```
00256 .DSA 035204
00257 .DSA 000000
00260 .DSA INT
00261 JMS* .SS
00262 .DSA RL2
00263 LAC (000003
00264 DAC $00264
00265 JMS* .GC
00266 .DSA 071177
00267 .DSA 000000
$00264 = 00270
00270 .DSA $00270
0038 C
0039 C
0040 C
0041 C
00271 JMS* .GB
00272 .DSA 071176
00273 .DSA 000000
00274 .DSA RL1
00275 JMS* .RF
0042 READ (5,100) INT,RL2(3),RL1
00276 JMS* .FR
00277 .DSA (000005
00300 .DSA .100
00301 .DSA 777777
00302 JMS* .FE
00303 .DSA INT
00304 CMA!CLA
00305 TAD (000003
00306 RCL
00307 TAD RL2
00310 DAC $00310
00311 .DSA 777776
00312 JMS* .FE
$00310 = 00313
00313 .DSA $00313
00314 JMS* .FA
00315 .DSA RL1
00316 JMS* .FF
0043 WRITE (6,200) INT,RL2(3),RL1
00317 JMS* .FW
00320 .DSA (000006
00321 .DSA .200
00322 .DSA 777777
00323 JMS* .FE
00324 .DSA INT
00325 CMA!CLA
00326 TAD (000003
00327 RCL
00330 TAD RL2
00331 DAC $00331
00332 .DSA 777776
00333 JMS* .FE
$00331 = 00334
00334 .DSA $00334
```

B. SEQUENTIAL ACCESS

1. FORMATTED

Programming Examples

```
00335 JMS* .FA
00336 .DSA RL1
00337 JMS* .FF
0044 ENCODE (10,ARR,100) INT,RL2(3),RL1
00340 JMS* .GF
00341 .DSA (000012
00342 .DSA ARR
00343 .DSA .100
00344 .DSA 777777
00345 JMS* .FE
00346 .DSA INT
00347 CMA!CLA
00350 TAD (000003
00351 RCL
00352 TAD RL2
00353 DAC S00353
00354 .DSA 777776
00355 JMS* .FE
S00353 = 00356
00356 .DSA S00356
00357 JMS* .FA
00360 .DSA RL1
00361 JMS* .FF
0045 DECODE (10,ARR,100) INT,RL2(3),RL1
00362 JMS* .GG
00363 .DSA (000012
00364 .DSA ARR
00365 .DSA .100
00366 .DSA 777777
00367 JMS* .FE
00370 .DSA INT
00371 CMA!CLA
00372 TAD (000003
00373 RCL
00374 TAD RL2
00375 DAC S00375
00376 .DSA 777776
00377 JMS* .FE
S00375 = 00400
00400 .DSA S00400
0046 C
0047 C
0048 C
2. DATA DIRECTED
00401 JMS* .FA
00402 .DSA RL1
00403 JMS* .FF
0049 READ (5,) INT,RL2(3),RL1
00404 JMS* .FR
00405 .DSA (000005
00406 .DSA 000000
00407 .DSA 777777
00410 JMS* .GD
00411 .DSA INT
00412 CMA!CLA
00413 TAD (000003
00414 RCL
```

Programming Examples

```
00415 TAD R02
00416 DAC S00416
00417 .DSA 777776
00420 JMS* .GD
S00416 = 00421
00421 .DSA S00421
00422 JMS* .GE
00423 .DSA R01
00424 JMS* .FF
0050 WRITE (6,) INT,R02(3),R01
00425 JMS* .FW
00426 .DSA (000000
00427 .DSA 000000
00430 .DSA 777777
00431 JMS* .GA
00432 .DSA 035204
00433 .DSA 000000
00434 .DSA INT
00435 JMS* .SS
00436 .DSA R02
00437 LAC (000003
00440 DAC S00440
00441 JMS* .GC
00442 .DSA 071177
00443 .DSA 000000
S00440 = 00444
00444 .DSA S00444
00445 JMS* .GB
00446 .DSA 071176
00447 .DSA 000000
00450 .DSA R01
00451 JMS* .FF
0051 DECODE (15,ARR,) INT,R02(3),R01
00452 JMS* .GG
00453 .DSA (000017
00454 .DSA ARR
00455 .DSA 000000
00456 .DSA 777777
00457 JMS* .GD
00460 .DSA INT
00461 CMA!CLA
00462 TAD (000003
00463 RCL
00464 TAD R02
00465 DAC S00465
00466 .DSA 777776
00467 JMS* .GD
S00465 = 00470
00470 .DSA S00470
00471 JMS* .GE
00472 .DSA R01
00473 JMS* .FF
0052 ENCODE (25,ARR,) INT,R02(3),R01
00474 JMS* .GF
00475 .DSA (000031
00476 .DSA ARR
```

Programming Examples

```
00477 .DSA 000000
00500 .DSA 777777
00501 JMS* .GA
00502 .DSA 035204
00503 .DSA 000000
00504 .DSA LNT
00505 JMS* .SS
00506 .DSA R02
00507 LAC (000003
00510 DAC $00510
00511 JMS* .GC
00512 .DSA 071177
00513 .DSA 000000
$00510 = 00514
00514 .DSA $00514
00515 JMS* .GB
00516 .DSA 071176
00517 .DSA 000000
00520 .DSA RLI
00521 JMS* .FF
0053      ENDFILE 1
00522 JMS* .FV
00523 .DSA (000001
0054      ENDFILE 2
00524 JMS* .FV
00525 .DSA (000002
0055      ENDFILE 3
00526 JMS* .FV
00527 .DSA (000003
0056      ENDFILE 4
00530 JMS* .FV
00531 .DSA (000004
0057      ENDFILE 5
00532 JMS* .FV
00533 .DSA (000005
0058      ENDFILE 6
00534 JMS* .FV
00535 .DSA (000006
0059      STOP
00536 LAC (000000
00537 JMP* .SI
0060      END
00540 CLA
00541 JMP* .SI
00542 JMS* .ZB
00543 JMS* .FP
00544 JMP 00000
00545 .BLK 000004
00551 .DSA 000001
00552 .DSA 000004
00553 .DSA 000000
00554 .DSA 000000
00555 .DSA RLI
00556 .BLK 000006
00564 .DSA 000001
00565 .DSA 000006
```

Programming Examples

```
00566 .DSA 000000
00567 .DSA 000000
00570 .DSA RL2
00571 .BLK 000050
00641 .DSA 000001
00642 .DSA 000050
00643 .DSA 000000
00644 .DSA 000000
00645 .DSA ARR
00646 .BLK 000004
00652 .DSA 000001
00653 .DSA 000004
00654 .DSA 000000
00655 .DSA 000000
00656 .DSA MM1
00657 .BLK 000004
00658 .DSA 000001
00659 .DSA 000004
00665 .DSA 000000
00666 .DSA 000000
00667 .DSA MM2
00670 .DSA DEFINE
00671 .BLK 000001
00672 .BLK 000001
00673 .DSA SEEK
00674 .DSA ENTER
00675 .DSA .RS
00676 .BLK 000001
00677 .DSA .RJ
00700 .DSA .RH
00701 .DSA .RG
00702 .DSA .RX
00703 .DSA .FS
00704 .DSA .FJ
00705 .DSA .FB
00706 .DSA .FG
00707 .DSA .FX
00710 .DSA .RK
00711 .DSA .RE
00712 .DSA .RA
00713 .DSA .RF
00714 .DSA .RW
00715 .DSA .GD
00716 .DSA .GE
00717 .DSA .GA
00720 .DSA .SS
00721 .DSA .GC
00722 .DSA .GB
00723 .DSA .FR
00724 .DSA .FE
00725 .DSA .FA
00726 .DSA .FF
00727 .DSA .FN
00730 .DSA .GF
00731 .DSA .GG
00732 .DSA .FV
```

Programming Examples

```
00733 .DSA .SF
00734 .DSA .ZB
00735 .DSA .FP
00736 .DSA 000002
00737 .DSA 000144
00740 .DSA 000005
00741 .DSA 000009
00742 .DSA 000004
00743 .DSA 001130
00744 .DSA 000012
00745 .DSA 000006
00746 .DSA 000003
00747 .DSA 000001
00750 .DSA 000007
00751 .DSA 000010
00752 .DSA 000017
00753 .DSA 000031

RL1 00545
RL2 00556
ARR 00571
NM1 00646
NM2 00657
.100 00000
.200 00011
* DEFINE 00670
JVB 00671
JVA 00672
* SEEK 00673
* ENTER 00674
* .RS 00675
INT 00676
* .RJ 00677
* .RB 00700
* .RG 00701
* .RX 00702
* .FS 00703
* .FJ 00704
* .FB 00705
* .FG 00706
* .FX 00707
* .RR 00710
* .RE 00711
* .RA 00712
* .RF 00713
* .RW 00714
* .GD 00715
* .GE 00716
* .GA 00717
* .SS 00720
* .GC 00721
* .GB 00722
* .FR 00723
* .FE 00724
* .FA 00725
* .FF 00726
* .FW 00727
* .GF 00730
* .GG 00731
* .FV 00732
* .ST 00733
* .ZB 00734
* .FP 00735
PROGRAM SIZE = 00754, NO ERRORS
```

APPENDIX D
SYSTEM LIBRARIES

D.1 DOS-15 PAGE MODE NON-FPP LIBRARY

LIBRARY FILE LISTING FOR .LIBRP

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
BOSTT	001	16	
RBCDIO	007	136	
RBINIO	006	113	
RANCOM	014	504	
DEFINE	017	1126	
DDIO	017	2045	
EDCODE	003	253	
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	004	23	
JDIM	001	21	
JMOD	003	23	
JMNMX	03P	103	
ERRSET	000	25	
IOERR	002	40	
FILE	010	366	
TIME	011	70	
TIME10	010	117	
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	002	22	
IDIM	000	15	
SNGL	004	27	
DBLE	001	11	
IMNMX	07P	107	
RMNMX	11P	120	
DMNMX	09P	106	
.BB	004	60	

LIBRARY FILE LISTING FOR .LIBRP

PAGE 2

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.BC	010	133	
.BD	010	133	
.BE	006	33	
.BF	005	34	
.BG	008	35	
.BH	005	34	
.BI	004	121	
SQRT	008	73	
SIN	003	13	
COS	003	20	
ATAN	002	13	
ATAN2	008	70	
EXP	002	13	
ALOG	002	20	
ALOG10	002	20	
TANH	004	47	
.EB	004	102	
.ED	006	67	
.EE	002	71	
.EF	008	143	
.EC	001	44	
DSQRT	007	71	
DSIN	001	13	
DCOS	002	21	
DATAN	001	13	
DATAN2	008	73	
DEXP	001	13	
DLOG	003	21	
DLOG10	001	21	
IDZERO	001	10	
ISENSW	001	30	
IFLOW	001	22	
.DD	006	148	
.DB	004	120	
.DE	003	101	
.DF	001	137	
.DC	001	47	
.DA	11P	56	
.DJ	000	51	
BCCIO	048	4023	
BINIO	020	357	
AUXIO	016	133	
.SS	009	110	
GOTO	003	26	
STOP	008	61	
PAUSE	006	14	

System Libraries

LIBRARY FILE LISTING FOR .LIBRP

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
SPMSG	012	117	
.FLT8	004	266	
FIOPS	035	751	
PARTWD	03P	140	
DBLINT	07P	377	
INTEAE	07P	131	
DOUBLE	004	203	
RELEASE	10P	1077	
OTSER	013	210	
.CB	004	22	C

System Libraries

D.2 DOS-15 BANK MODE NON-FPP LIBRARY

LIBRARY FILE LISTING FOR .LIBRB

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
BOSTT	001	16	
RBCDIO	007	136	
RBINIO	006	113	
RANCOM	014	504	
DEFINE	017	1126	
DDIO	017	2045	
EDCODE	003	253	
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	004	23	
JDIM	001	21	
JMOD	003	23	
JMNMX	03B	103	
ERRSET	000	25	
IOERR	002	40	
FILE	010	366	
TIME	011	70	
TIME10	010	117	
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	002	22	
IDIM	000	15	
SNGL	004	27	
DBLE	001	11	
IMNMX	07B	107	
RMNMX	11B	120	
DMNMX	09B	106	
.BB	004	60	

System Libraries

LIBRARY FILE LISTING FOR .LIBRB

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.BC	010	133	
.BD	010	133	
.BE	006	33	
.BF	005	34	
.BG	008	35	
.BH	005	34	
.BI	004	121	
SQRT	008	73	
SIN	003	13	
COS	003	20	
ATAN	002	13	
ATAN2	008	70	
EXP	002	13	
ALOG	002	20	
ALOG10	002	20	
TANH	004	47	
.EB	004	102	
.ED	006	67	
.EE	002	71	
.EF	008	143	
.EC	001	44	
DSQRT	007	71	
DSIN	001	13	
DCOS	002	21	
DATAN	001	13	
DATAN2	008	73	
DEXP	001	13	
DLOG	003	21	
DLOG10	001	21	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	006	146	
.DB	004	120	
.DE	003	101	
.DF	001	137	
.DC	001	47	
.DA	118	56	
.DJ	000	51	
BCDIO	048	4023	
BINIO	020	357	
AUXIO	016	133	
.88	009	110	
GOTO	003	26	
STOP	008	61	
PAUSE	006	14	

System Libraries

LIBRARY FILE LISTING FOR .LIBRB

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
SPMSG	012	117	
.FLTB	004	266	
FIOPS	035	751	
PARTWD	038	141	
DBLINT	078	404	
INTEAE	008	134	
DOUBLE	004	203	
RELEASE	016	1111	
OTSER	013	210	
.CB	004	22	C

D.3 DOS-15 PAGE MODE FPP LIBRARY

LIBRARY FILE LISTING FOR .FPAG

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
BOSTT	001	16	
RBCDIO	007	136	
RBINIO	006	113	
RANCOM	014	504	
DEFINE	017	1126	
DDIO	F17	2016	
EDCODE	003	253	
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	F04	16	
JDIM	F01	17	
JMOD	F03	17	
JMNMX	03U	102	
ERRSET	000	25	
IDERR	002	40	
FILE	010	366	
TIME	011	70	
TIME10	010	117	
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	000	20	
DIM	F02	17	
IDIM	000	15	
SNGL	F04	16	
DBLE	F01	10	
IMNMX	07P	107	
RMNMX	11U	116	
DMNMX	09U	104	
.BB	004	60	

System Libraries

LIBRARY FILE LISTING FOR .PPAG

PAGE 2

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.BC	F10	127	
.BD	F10	127	
.BE	F06	30	
.BF	F05	31	
.BG	F06	31	
.BH	F05	31	
.BI	F04	114	
SQRT	F08	73	
SIN	F03	12	
COS	F03	16	
ATAN	F02	12	
ATAN2	F08	61	
EXP	F02	12	
ALOG	F02	16	
ALOG10	F02	16	
TANH	F04	46	
.EB	F04	77	
.ED	F06	70	
.EE	F02	72	
.EF	F08	140	
.EC	F01	40	
DSQRT	F07	70	
DSIN	F01	12	
DCOS	F02	17	
DATAN	F01	12	
DATAN2	F08	64	
DEXP	F01	12	
DLOG	F03	17	
DLOG10	F01	17	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	F06	137	
.DB	F04	115	
.DE	F03	104	
.DF	F01	130	
.DC	F01	43	
.DA	11P	56	
.DJ	000	51	
BCDIO	F48	3731	
BINIO	020	357	
AUXIO	016	133	
.SS	009	110	
GOTO	003	26	
STOP	008	61	
PAUSE	006	14	

System Libraries

LIBRARY FILE LISTING FOR .FPAG

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
SPMSG	012	117	
.FLTB	004	266	
FIOPS	035	751	
PARTWD	03U	146	
INTEAE	07P	131	
.FPP	F18	440	
OTSER	F13	210	
.CB	004	22	C

System Libraries

D.4 DOS-15 BANK MODE FPP LIBRARY

LIBRARY FILE LISTING FOR .FBNK

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
BOSTT	001	16	
RBCDIO	007	136	
RBINIO	006	113	
RANCOM	014	504	
DEFINE	017	1126	
DDIO	F17	2016	
EDCODE	003	253	
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	F04	16	
JDIM	F01	17	
JMOD	F03	17	
JMNMx	03V	102	
ERRSET	000	25	
IOERR	002	40	
FILE	010	366	
TIME	011	70	
TIME10	010	117	
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	000	20	
DIM	F02	17	
IDIM	000	15	
SNGL	F04	16	
DBLE	F01	10	
IMNMx	07B	107	
RMNMx	11V	116	
DMNMx	09V	104	
.BB	004	60	

System Libraries

LIBRARY FILE LISTING FOR .FBNK

PAGE 2

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.BC	F10	127	
.BD	F10	127	
.BE	F06	30	
.BF	F05	31	
.BG	F08	31	
.BH	F05	31	
.BI	F04	114	
SQRT	F08	73	
SIN	F03	12	
COS	F03	16	
ATAN	F02	12	
ATAN2	F08	61	
EXP	F02	12	
ALOG	F02	16	
ALOG10	F02	16	
TANH	F04	46	
.EB	F04	77	
.ED	F06	70	
.EE	F02	72	
.EF	F08	140	
.EC	F01	40	
DSQRT	F07	70	
DSIN	F01	12	
DCOS	F02	17	
DATAN	F01	12	
DATAN2	F08	64	
DEXP	F01	12	
DLOG	F03	17	
DLOG10	F01	17	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	F06	137	
.DB	F04	115	
.DE	F03	104	
.DF	F01	130	
.DC	F01	43	
.DA	118	56	
.DJ	000	51	
BCDIO	F48	3731	
BINIO	020	357	
AUXIO	016	133	
.SS	009	110	
GOTO	003	26	
STOP	008	61	
PAUSE	006	14	

System Libraries

LIBRARY FILE LISTING FOR .FBNK

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
SPMSG	012	117	
.FLTB	004	266	
FIOPS	035	751	
PARTWD	03V	147	
INTEAE	008	134	
.FPP	F18	440	
OTSER	F13	210	
.CB	004	22	C

System Libraries

D.5 RSX PLUS III NON-FPP LIBRARY

LIBRARY FILE LISTING FOR .LIBRX

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
MTGP.2	SRC	53	
FMF.1	SRC	24	
LAB.3	SRC	34	
MNT.1	SRC	24	
BSF.1	SRC	24	
BSP.1	SRC	24	
SPE.2	SRC	20	
SPR.1	SRC	24	
SPF.1	SRC	24	
RDC.2	SRC	33	
ADSMAP	001	114	
ADRMAP	002	126	
ADSSET	003	74	
ADRSET	002	66	
ADSTRT	001	20	
ADSTOP	001	20	
ADCON	001	20	
ADDIS	001	20	
ROP.1	SRC	53	
DOUT.0	SRC	105	
RDDI.6	SRC	205	
RBIN.0	SRC	31	
RBCD.2	SRC	73	
AI.3	SRC	43	
WAIF.1	SRC	4	
EXU.11	SRC	161	
DATF.5	SRC	40	
REDF.2	SRC	27	
SCHF.6	SRC	50	
RUNF.5	SRC	47	
SYNF.4	SRC	55	
CANF.3	SRC	23	
RESF.3	SRC	24	
HINF.1	SRC	17	
FIXF.3	SRC	23	
UNFF.4	SRC	23	
DISF.3	SRC	23	
ENAF.4	SRC	23	
ATTF.3	SRC	20	
DETF.2	SRC	20	
RENF.1	SRC	35	
DELF.1	SRC	35	
SEEF.2	SRC	35	
ENTF.2	SRC	35	
CLOF.5	SRC	57	
DSAF.6	SRC	40	

System Libraries

LIBRARY FILE LISTING FOR .LIBRX

PAGE 2

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
OSDF.4	SRC	41	
OSGP.1	SRC	102	
MARF.3	SRC	23	
WAFF.2	SRC	13	
SUSF.1	SRC	4	
EXIF.1	SRC	3	
UPKF.0	SRC	21	
SPYF.1	SRC	24	
SPYR.0	SRC	40	
SPYS.1	SRC	24	
QJOB.1	SRC	43	
EXQT.1	SRC	41	
COMCOM	001	65	
FYS.3	SRC	70	
RBCDIO	007	123	
RBINIO	006	100	
RANCOM	014	525	
DEFINE	018	747	
DDIO	017	2026	
EDCODE	003	253	
JABS	001	15	
JOFIX	001	13	
JFIX	001	13	
FLOATJ	001	13	
JDRLE	001	10	
ISNGL	002	30	
JSIGN	004	23	
JDIM	001	21	
JMOD	003	23	
JMMX	003	105	
ERRSET	000	25	
IOERR	002	40	
ABS	002	16	
IABS	000	14	
OABS	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	
YSIGN	000	20	

System Libraries

LIBRARY FILE LISTING FOR .LIBRX

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
DIM	002	22	
IDIM	000	15	
SNGL	004	27	
DBLE	001	11	
IMNMX	007	111	
RMNMX	011	122	
DMNMX	009	110	
.BB	004	60	
.BC	010	133	
.BD	010	133	
.BE	006	33	
.BF	005	34	
.BG	008	35	
.BH	005	34	
.BI	004	121	
SQRT	008	73	
SIN	003	13	
COS	003	20	
ATAN	002	13	
ATAN2	008	70	
EXP	002	13	
ALOG	002	20	
ALOG10	002	20	
TANH	004	47	
.EB	004	102	
.ED	006	67	
.EE	002	71	
.EF	008	143	
.EC	001	44	
DSQRT	007	71	
DSIN	001	13	
DCOS	002	21	
DATAN	001	13	
DATAN2	008	73	
DEXP	001	13	
DLOG	003	21	
DLOG10	001	21	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	006	146	
.DB	004	120	
.DE	003	101	
.DF	001	137	
.DC	001	47	
.DA	011	101	

System Libraries

LIBRARY FILE LISTING FOR .LIBRX

PAGE 4

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.DJ	000	51	
BCDIO	008	3733	
BINIO	000	267	
AUXIO	016	133	
.SS	000	110	
GOTO	003	26	
STOP	007	14	
PAUSE	006	14	
FIOPS	035	676	
PARTWD	03P	140	
OBLINT	07P	377	
INTEAE	07P	131	
DOUBLE	004	203	
RELEASE	10P	1077	
OTSER	013	242	
SPMSG	012	124	
.CB	004	22	
.BP	000	10	
.FP	000	2	C

System Libraries

D.6 RSX PLUS III FPP LIBRARY

LIBRARY FILE LISTING FOR .LIBFX

PAGE 1

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
MTGP.2	SRC	53	
FMF.1	SRC	24	
LAB.3	SRC	34	
MNT.1	SRC	24	
BSF.1	SRC	24	
BSP.1	SRC	24	
SPE.2	SRC	20	
SPR.1	SRC	24	
SPF.1	SRC	24	
RDC.2	SRC	33	
ADSMAP	001	114	
ADRMAP	002	126	
ADSSET	003	74	
ADRSET	002	66	
ADSTRT	001	20	
ADSTOP	001	20	
ADCON	001	20	
ADDIS	001	20	
RDP.1	SRC	53	
DOUT.0	SRC	105	
RDDI.6	SRC	205	
RBIN.0	SRC	31	
RBCD.2	SRC	73	
AI.3	SRC	43	
WAIF.1	SRC	4	
EXU.11	SRC	161	
DATF.5	SRC	40	
REOF.2	SRC	27	
SCHF.6	SRC	50	
RUNF.5	SRC	47	
SYNF.4	SRC	55	
CANF.3	SRC	23	
RESF.3	SRC	24	
HINF.1	SRC	17	
FIXF.3	SRC	23	
UNFF.4	SRC	23	
DISF.3	SRC	23	
ENAF.4	SRC	23	
ATTF.3	SRC	20	
DETF.2	SRC	20	
RENF.1	SRC	35	
DELF.1	SRC	35	
SEEF.2	SRC	35	
ENTF.2	SRC	35	
CLOF.5	SRC	57	
DSAF.6	SRC	40	

System Libraries

LIBRARY FILE LISTING FOR .LIBFX

PAGE 2

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
DSDF.4	SRC	41	
DSGP.1	SRC	102	
MARF.3	SRC	23	
WAF.2	SRC	13	
SUSF.1	SRC	4	
EXIF.1	SRC	3	
UPKF.0	SRC	21	
SPYF.1	SRC	24	
SPYR.0	SRC	40	
SPYS.1	SRC	24	
QJOB.1	SRC	43	
EXGT.1	SRC	41	
COMCOM	001	65	
FTS.3	SRC	70	
RBCDIO	007	123	
RBINIO	006	100	
RANCOM	014	525	
DEFINE	018	747	
DDIO	F17	1777	
EDCODE	003	253	
JABS	F01	14	
JDFIX	F01	12	
JFIX	F01	12	
FLOATJ	F01	10	
JDBLE	F01	10	
ISNGL	F02	13	
JSIGN	F04	16	
JDIM	F01	17	
JMOD	F03	17	
JMNMX	F03	104	
ERRSET	000	25	
IDERR	002	40	
ABS	F02	13	
IABS	000	14	
DABS	F01	13	
AINTE	F02	14	
INT	F02	12	
IDYNT	F05	12	
AMOD	F03	23	
MOD	000	24	
DMOD	F04	23	
FLOAT	002	11	
IFIX	F02	12	
SIGN	F04	24	
DSIGN	F04	24	
ISIGN	000	20	

System Libraries

LIBRARY FILE LYSTING FOR .LIBFX

PAGE 3

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
DIM	F02	17	
IDIM	000	15	
SNGL	F04	16	
DBLE	F01	10	
IMNMX	007	111	
RMNMX	F11	120	
DMNMX	F09	106	
.BB	004	60	
.BC	F10	127	
.BD	F10	127	
.BE	F06	30	
.BF	F05	31	
.BG	F08	31	
.BH	F05	31	
.BI	F04	114	
SQRT	F08	73	
SIN	F03	12	
COS	F03	16	
ATAN	F02	12	
ATAN2	F08	61	
EXP	F02	12	
ALOG	F02	16	
ALOG10	F02	16	
TANH	F04	46	
.EB	F04	77	
.ED	F06	70	
.EE	F02	72	
.EF	F08	140	
.EC	F01	40	
DSORT	F07	70	
DSIN	F01	12	
DCOS	F02	17	
DATAN	F01	12	
DATAN2	F08	64	
DEXP	F01	12	
DLOG	F03	17	
DLOG10	F01	17	
IDZERO	001	16	
ISENSW	001	30	
IFLOW	001	22	
.DD	F06	137	
.DB	F04	115	
.DE	F03	104	
.DF	F01	130	
.DC	F01	43	
.DA	011	101	

System Libraries

LIBRARY FILE LISTING FOR .LIBFX

PAGE 4

PROGRAM NAME	SOURCE EXTENSION	PROGRAM SIZE	ACTION
.DJ	000	51	
BCDIO	F4R	3641	
BINIO	000	267	
AUXIO	016	133	
.SS	009	110	
GOTO	003	26	
STOP	007	14	
PAUSE	006	14	
FIOPS	035	676	
PARTWD	F3P	146	
INTEAF	07P	131	
.FPP	F18	461	
OTSER	F13	251	
SPMSG	012	124	
.CB	004	22	
.BP	000	10	
.FP	000	2	C

INDEX

- Absolute program, 1-4
- Absolute value, 3-3
- Accumulators, 3-14
- Adjustable arrays, 4-3
- Arctangent function, 3-7, 3-10, 3-12
- Arguments, MACRO, 5-2
- Arithmetic operator, A-1
- Arithmetic package, 2-14, 3-17
- Array descriptor block, 4-1
- Arrays, 4-2
- ASCII data, 2-13
- .ASCII to .SIXBT conversion, 4-8
- AUXIO routines, 2-7

- BACKSPACE command, 2-7, 2-8
- Batch-processing monitor, 1-7
- BCDIO package, 2-5
- Binary record, 2-12
- BINIO routines, 2-6
- Blank common, 5-3
- BOSS operating system, 1-7
- BOSS routines, 2-18
- BSPREC magnetic tape function, 2-8

- Calling FORTRAN IV, 1-2
- Calling FORTRAN subprograms from
MACRO, 5-2
- Calling library subprograms, 3-1
- Calling MACRO subprograms, 5-1
- Carriage return, 1-2
- CHAIN program, 1-4
- Clock handling routine, 4-7
- CLOSE routine, 2-16, 2-17
- Code, relocatable object, 1-1
- Command string format, 1-2
- COMMON blocks, 5-3
- Common logarithm, 3-7
- Compiler error messages, B-1
- Compiler task name, 1-2
- Control P (†P), 1-3
- Conversion, 3-4
- Cosine function, 3-7, 3-9

- .DAA routine, 4-8
- Data, ASCII, 2-13
- Data-directed input-output (DDIO), 2-13
- Data modes, 4-2
- Data structure, 2-2
- Data transmission, 2-3
 - statements, 2-1
- .DAT (Device Assignment Table), 2-2
- .DAT slot, 2-8
- DDIO (Data-Directed Input-Output), 2-13
- DEctape, 2-2
- DEFINE function (flowchart), 2-10
- DEFINE routine, 2-9
- DELETE routine, 2-18
- Device Assignment Table (.DAT), 2-2
- Device handlers, 2-3
- Devices, serial, 2-2
- Diagnostics, 1-4
- Difference, positive, 3-4
- Direct access I/O, 2-9
- Directoried I/O, 2-2
- Disk operating system, 1-6
- Division by zero, 3-15
- .DJ routine, 4-3
- DLETE routine, 2-16
- DOS directoried subroutines, 2-15
- DOS-15 Bank Mode FPP library, D-10
- DOS-15 Bank Mode non-FPP library, D-4
- DOS-15 Page Mode FPP library, D-7
- DOS-15 Page Mode non-FPP library, D-1
- DOS monitor system, 1-6
- Double (extended) integer arithmetic, 3-14
- DOUBLE INTEGER mode, 4-2
- DOUBLE PRECISION mode, 4-2
- Double-precision numbers, 3-15
- Dummy array, 4-2

- EAE (Extended Arithmetic Element), 1-7
- EDCODE (encode/decode), 2-15
- ENDFILE command, 2-7, 2-8
- ENTER routine, 2-16, 2-17
- Error handling routine, 4-7
- Error, I/O, 2-5
- Error messages, 1-3, B-1

INDEX (Cont.)

- Errors, OTS, 1-4
- ERRSET, 4-7
- Examples of programming, C-1
- Exponential function, 3-6, 3-7, 3-12
- Expression operators, A-1
- Extended integer (double integer)
 - interface routines, 4-6
- Extension,
 - LST., 1-3
 - SRC, 1-2
- External functions, 3-6, 3-7

- File directories, 2-2
- File initialization, 2-9
- Filename, 1-2
- FLOPS package, 2-1, 2-3, 4-8
- Floating accumulator, 3-14
- Floating point processor (FPP), 3-14
 - routines, 4-6
- Flowchart, DEFINE function, 2-10
- Format, command string, 1-2
- FORMAT statements, 2-6
- Formatted input/output,
 - OTS, 2-5
 - RBCDIO, 2-11
- Forms-control character, 2-5
- FPI5 floating-point processor (FPP), 1-7
 - routines, 4-6
- FSTAT routine, 2-16
- Functions, intrinsic, 3-2
 - table, 3-3

- General Get argument, 3-13
- .GLOBL pseudo-operation, 3-1
- .GO routine, 4-3

- Handlers, device, 2-3
- Hardware, 1-7
- Hardware accumulator, 3-14
- Header pair, 2-5
- Header words, 2-6
- Hyperbolic tangent, 3-7, 3-10

- IDZERO routine, 3-16
 - example, C-1

- IFLOW routines, 3-16
 - example, C-1
- Initialization and actual data transfer (RANCOM), 2-13
- Initialization, file, 2-9
- Input-output examples, C-2
- Input-output processing, 2-1
- INTEGER array, 4-2
- INTEGER, LOGICAL mode, 4-2
- Intrinsic functions, 3-2, 3-3
- I/O error, 2-4

- Language summary, A-1
- Left arrow (+) usage, 1-2
- Libraries, system, D-1
- Library functions, summary, A-12
- Library routines, RSX, 4-6
- Library subprograms, 3-1
- Linking loader, 1-4
- Links, 1-4
- Logarithm, base 2, 3-11, 3-12
- Logarithms, 3-8
- Logical operators, A-1
- Logical record, 2-6
- Logical Unit Number (LUN), 2-2
- Logical Unit Table (LUT), 2-2
- LST. extension, 1-3
- LUN (Logical Unit Number), 2-2
- LUT (Logical Unit Table), 2-2

- MACRO programming, 2-6
- MACRO subprograms, 5-1
- Magnetic tape, 2-2
- Mantissae, negative, 3-14
- Master File Directory (MFD), 2-2
- Maximum/minimum value, 3-5
- Memory-to-memory transfers, 2-15
- MFD (Master File Directory), 2-2

- Natural logarithms, 3-7
- Negative mantissae, 3-14

- Object Time System - see OTS
- Operating procedures, 1-1
- Operators, A-1

INDEX (Cont.)

- Option list, 1-3
- OTS auxiliary input/output (AUXIO), 2-7
- OTS binary input/output (BINIO), 2-6
- OTS errors, 1-4
 - messages, B-7, B-9
- OTSER routine, 4-4, 4-8
- OTS formatted input/output, 2-5
- OTS utility routines, 1-1, 4-1
- Output listing file, 1-3
- Output to a printing device, 2-5
- Overflow, 3-15
- Overlays, 1-4

- † P (control P), 1-3
- Paper tape, 2-2
- Paper tape reader, 1-3
- Parameter table (.PRMTB), 2-11
- PARTWD routine, 4-5
- Pass 1, 1-3
- Pass 2, 1-3
- PAUSE routine, 4-4, 4-8
- Physical records, 2-6
- Polynomial evaluation, 3-12, 3-14
- Positive difference, 3-4
- .PRMTB (parameter table), 2-11
- Processing, input-output, 2-1

- RANCOM (initialization and actual data transfer), 2-13
- RBCDIO (formatted input/output), 2-11
- RBINIO (unformatted input/output), 2-12
- READ statement, 2-6
- REAL array, 4-2
- REAL mode, 4-2
- Real-time multiprogramming, 1-7
- Record identification number, 2-6
- Record length, 2-5
- Relational operators, A-1
- RELEAE, real arithmetic package, 1-7, 3-15
- Relocatable object code, 1-1
- Remaindering, 3-4
- RENAME routine, 2-18
- RENAM routine, 2-16
- Retrieval Information Block (RIB), 2-2
- REWIND command, 2-7, 2-8
- REWIND magnetic tape function, 2-8

- Routines,
 - floating point processor, 4-6
 - OTS 4-8
 - RSX library, 4-6
 - utility, 4-1, 4-7
- RSX directoried subroutines, 2-17
- RSX library routines, 4-6
- RSX monitor system, 1-7
- RSX PLUS III FPP library, D-17
- RSX PLUS III non-FPP library, D-13

- Sample XVM/DOS session (figure), 1-5
- Science library, 3-1
- SEEK routine, 2-16, 2-17
- Sequential files, 2-2
- Sequential input-output, 2-4
- Serial devices, 2-2
- Sine function, 3-7, 3-9, 3-12
- Single integer arithmetic routines, 3-14
- Single-precision numbers, 3-15
- Software, 1-6
- Software accumulators, 3-14
- SPMSG routine, 4-8
- Square root function, 3-6, 3-7
- SRC extensions, 1-2
- .SS routine, 4-1
- Statements, data-transmission, 2-1
- Statements, summary, A-2
- STOP routine, 4-4, 4-8
- Subfunctions, 3-11
 - table, 3-12
- Subroutines, user, 2-15
- System libraries, D-1

- TDV function task builder (TKB), 1-4
- TIME routine, 4-7
- Transfer of sign, 3-4
- Truncation, 3-3
- TTOF routine, 2-18
- TTON routine, 2-18

- UFDs (User File Directories), 2-2
- Underflow, 3-15
- Unformatted input/output (RBINIO), 2-12
- Unformatted statements, 2-6
- User File Directories (UFDs), 2-2
- User subroutines, 2-15

INDEX (Cont.)

User subroutines, 2-15
Utility routines, 4-1, 4-7

Word-pairs, 2-5
WREOF magnetic tape function, 2-8
Write statement, 2-6

READER'S COMMENTS

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

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

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

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

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____

or
Country

If you require a written reply, please check here.

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

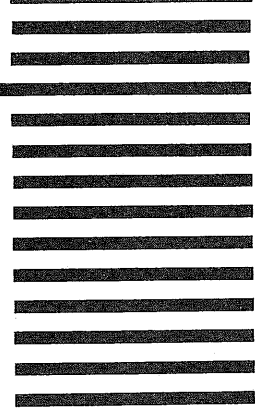
FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

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



C
A
O
C
C
C
C

digital

digital equipment corporation

digital

digital equipment corporation