

Honeywell

**FORTRAN
SUBROUTINES LIBRARIES**

**LEVEL 66
SOFTWARE**



FORTRAN
SUBROUTINE LIBRARIES
ADDENDUM A

SERIES 60(LEVEL 66)/6000

SOFTWARE

SUBJECT:

Additions and Changes to Series 60 (Level 66)/6000 FORTRAN Subroutine Libraries

SPECIAL INSTRUCTIONS:

This update, Order Number DD20A, is the first addendum to DD20, Rev. 0, dated May 1975. The attached pages are to be inserted into the manual as indicated in the collating instructions on the back of this cover. Change bars in the page margins indicate technical additions and changes; asterisks indicate deleted material. These changes will be incorporated into the next revision of the manual.

NOTE: This cover should be placed following the manual cover to indicate that the document has been updated with Addendum A.

SOFTWARE SUPPORTED:

Series 60 Level 66 Software Release 3
Series 6000 Software Release I

DATE:

March 1977

ORDER NUMBER:

DD20A, Rev. 0

21363

1.5778

Printed in U.S.A.

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove

v, vi
vii, blank
2-1 through 2-4
2-7 through 2-12
2-15, 2-16
2-31 through 2-34
2-49, 2-50
3-1, 3-2
3-7, 3-8
3-23, 3-24

3-27, 3-28
3-31, blank

4-1 through 4-60

Insert

v, vi
vii, blank
2-1 through 2-4
2-7 through 2-12
2-15, 2-16
2-31 through 2-34
2-49, 2-50
3-1, 3-2
3-7, 3-8
3-23, 3-24
3-24.1, blank
3-27, 3-28
3-31, 3-32
3-33 through 3-42
3-43, blank
4-1 through 4-66
4-67, blank

**SERIES 60 (LEVEL 66)/6000
FORTRAN
SUBROUTINE LIBRARIES**

SUBJECT

**FORTRAN Input/Output, Error Monitoring, Mathematical, and
Nonmathematical Subroutines**

SPECIAL INSTRUCTIONS

For Series 6000 systems, this manual replaces the manual of the same name, Order Number BR95, dated June 1971. Order Number BR95 remains an active publication for Series 600 systems and for Series 6000 systems on prior software releases.

SOFTWARE SUPPORTED

Series 60 Level 66 Software Release 2
Series 6000 Software Release H

ORDER NUMBER

DD20, Rev. 0

May 1975

Honeywell

PREFACE

This manual describes the FORTRAN Subroutine Libraries. These libraries include subroutines for input/output, mathematical and nonmathematical functions, and execution error monitoring.

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 60 (LEVEL 66) and SERIES 6000 SYSTEMS

FUNCTION	APPLICABLE REFERENCE MANUAL	ORDER NO.
	TITLE	
	Series 60 (Level 66)/Series 6000:	
Hardware reference:		
Series 60 Level 66 System	Series 60 Level 66 Summary Description	DC64
Series 6000 System	Series 6000 Summary Description	DA48
DATANET 355 Processor	DATANET 355 Systems Manual	BS03
DATANET 6600 Processor	DATANET 6600 Systems Manual	DC88
Operating system:		
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	DD19
Job Control Language	Control Cards Reference Manual	DD31
Table Definitions	System Tables	DD14
I/O Via MME GEINOS	I/O Programming	DB82
System initialization:		
System Startup	System Startup	DD33
System Operation	System Operation Techniques	DD50
Communications System	GRTS/355 and GRTS/6600 Startup Procedures	DD05
Communications System	NPS Startup	DD51
DSS180 Subsystem Startup	DSS180 Startup	DD34
Data management:		
File System	File Management Supervisor	DD45
Integrated Data Store (I-D-S)	I-D-S/I Programmer's Guide	DC52
Integrated Data Store (I-D-S)	I-D-S/I User's Guide	DC53
File Processing	Indexed Sequential Processor	DD38
File Input/Output	File and Record Control	DD07
File Input/Output	Unified File Access System (UFAS) (Series 60 only)	DC89
I-D-S Data Query System	I-D-S Data Query System Installation	DD47
I-D-S Data Query System	I-D-S Data Query System User's Guide	DD46
Program maintenance:		
Object Program	Source and Object Library Editor	DD06
System Editing	System Library Editor	DD30
Test system:		
Online Test Program	Total Online Test System (TOLTS)	DD39
Test Descriptions	Total Online Test System (TOLTS) Test Pages	DD49
Error Analysis and Logging	Honeywell Error Analysis and Logging System (HEALS)	DD44
Language processors:		
Macro Assembly Language	Macro Assembler Program	DD08
COBOL-68 Language	COBOL	DD25
COBOL-68 Usage	COBOL User's Guide	DD26
JOVIAL Language	JOVIAL	DD23
FORTRAN Language	FORTRAN	DD02
Generators:		
Sorting	Sort/Merge Program	DD09
Merging	Sort/Merge Program	DD09

TITLESeries 60 (Level 66)/Series 6000:

Simulators:		
DATANET 355/6600 Simulation	DATANET 355/6600 Simulator	DD32
Service and utility routines:		
Loader	General Loader	DD10
Utility Programs	Utility	DD12
Utility Programs	UTL2 Utility Routine (Series 60 only)	DC91
Media Conversion	Bulk Media Conversion	DD11
System Accounting	Summary Edit Program	DD24
FORTRAN	FORTRAN Subroutine Libraries	DD20
FNP Loader	DATANET 355/6600 Relocatable Loader	DD35
Service Routines	Service Routines	DD42
Software Debugging	Debug and Trace Routines	DD43
Time Sharing systems:		
Operating System	TSS General Information	DD22
System Programming	TSS Terminal/Batch Interface	DD21
System Programming	TSS System Programmer's Reference Manual	DD17
BASIC Language	Time Sharing BASIC	DD16
FORTRAN Language	FORTRAN	DD02
Text Editing	Time Sharing Text Editor	DD18
Remote communications:		
DATANET 30/305/355/6600 FNP	Remote Terminal Supervisor (GRTS)	DD40
DATANET 355/6600 FNP	Network Processing Supervisor (NPS)	DD48
DATANET 700 RNP	RNP/FNP Interface	DB92
Transaction processing:		
User's Procedures	Transaction Processing System User's Guide	DD41
Handbooks:		
System-operator communication	System Console Messages	DD13
Pocket guides:		
Control Card Formats	Control Cards and Abort Codes	DD04
FORTRAN	FORTRAN Pocket Guide	DD82

CONTENTS

		Page
Section I	Introduction.	1-1
Section II	Input/Output Library.	2-1
	Library Calls.	2-1
	Input/Output Library Subroutines	2-3
	Subroutines Implicitly Called.	2-4
	Linked Binary Input/Output Interface.	2-4
	Short List Input/Output Processor	2-8
	Random Binary Input/Output.	2-10
	Format Controlled Sequential Input/Output	2-12
	Namelist Input.	2-17
	Namelist and Dump Output.	2-21
	Character String Assignment	2-23
	Character String Compare.	2-24
	Output Stop and Pause Information	2-25
	Object Time Debug Processor	2-27
	Pre-execution Initializer (Batch)	2-28
	Pre-execution Initializer (Time Sharing).	2-30
	Arithmetic Fault Processor.	2-31
	Backspace Record.	2-32
	Rewind and Endfile Processor.	2-34
	End-of-File (On Input) Processor.	2-36
	File Opening.	2-37
	Carriage Control Simulator.	2-39
	Subroutines That Are User Callable	2-41
	Console Communication	2-41
	Memory Dump	2-43
	File Control Block and Logical Unit	
	Table Routines	2-44
	Define File Control Block.	2-44
	Define Buffer(s) for a Specified File	
	Control Block	2-45
	Define Logical Unit Table.	2-46
	File Closing.	2-47
	Initialization of End-of-File Processing.	2-48
	Initialization of Data Error Processing	2-49
	Job Termination	2-50
	File Forwardspace and Backspace	2-51
	Call TSS Subsystem.	2-52
	Create TSS Temporary File	2-53
	Specify Record Size, Random Binary File	2-54
	Set or Reset Some I/O Parameters of	
	Run-Time Library	2-55
Section III	Miscellaneous Library Subroutines	3-1
	Subroutines Implicitly Called.	3-3
	Double Precision Powers of Ten Table.	3-3
	Restore Link - H*	3-4
	Terminal Input Recovery	3-6
	ASCII/BCD Indicators.	3-7
	Subroutines That Are User Callable	3-8
	Exponent Register Overflow and Divide	
	Check Tests.	3-8

CONTENTS (cont)

	Page
Sense Light Simulator	3-9
Sense Switch Test	3-10
Restore Links During Execution (Batch).	3-11
Restore Links During Execution (Time Sharing)	3-12
Execution Error Monitor	3-13
Switch Word Groups	3-13
GMAP Calling Sequence.	3-14
FORTRAN Calling Sequences.	3-15
File Transliteration.	3-25
Date and Time	3-27
Access a Permanent File	3-28
Close File, Detach Buffers, Remove from AFT	3-30
Attach a Temporary Mass Storage or Terminal File.	3-31
FORTRAN Debugging System (FDS) Subroutines	3-32
Core Allocator.	3-32
Special Entry Point	3-33
FDEBUG Bootstrap.	3-34
FDUMP Bootstrap	3-35
LINK/LLINK Interface.	3-36
Delete From Wrapup List	3-37
Release Unused Memory	3-38
Dummy Setup	3-39
Add to Wrapup List.	3-40
Timing Facility	3-41
Wrapup and Loader	3-42
Linking Subroutine.	3-43
 Section IV	
Mathematical Library Subroutines.	4-1
Mathematical Library Descriptions.	4-2
Definitions and Considerations	4-6
Exponentiation - Integer Base and Exponent	4-8
Exponentiation - Real Base, Integer Exponent	4-10
Exponentiation - Real Base and Exponent	4-11
Exponentiation - Complex Base, Any Exponent	4-13
Natural, Base 2, and Common Logarithms.	4-14
Real Hyperbolic Arcsine, Arccosine, and Arctangent	4-16
Real Arcsine and Arccosine.	4-18
Real Arctangent	4-21
Complex Absolute Value.	4-23
Complex Exponential	4-24
Complex Natural Logarithm	4-25
Complex Multiplication and Division	4-26
Real Cube Root.	4-28
Complex Sine and Cosine	4-30
Complex Square Root	4-32
Exponentiation - Complex Base, Integer Exponent	4-33
Double Precision Hyperbolic Arcsine, Arccosine, and Arctangent	4-34
Double Precision Arcsine and Arccosine.	4-35
Double Precision Arctangent	4-37
Double Precision Cube Root.	4-39
Double Precision Exponential, Base 2, and Base 10.	4-40

CONTENTS (cont)

	Page
Double Precision Natural, Base 2, and Common Logarithms.	4-42
Double Precision Remaindering	4-44
Double Precision Hyperbolic Sine, Cosine, and Tangent	4-45
Double Precision Sine and Cosine.	4-47
Double Precision Square Root.	4-49
Double Precision Tangent.	4-50
Double Precision Exponential Complement	4-52
Double Precision Exponentiation, Real Base or Exponent (or Both)	4-54
Real Exponential Complement	4-56
Real Exponential, Base 2, and Base 10	4-58
Real Sine and Cosine.	4-60
Real Square Root.	4-62
Real Tangent.	4-64
Real Hyperbolic Sine, Cosine, and Tangent	4-66
 Section V	
Nonmathematical Library Subroutines	5-1
Mode Determination by FORTRAN	5-2
Character String Manipulation	5-3
Sort Array of Data.	5-4
Memory Usage	5-5
User Time and Identification.	5-6
Set/Reset Switch Word	5-7
Shift/Rotate Word Contents.	5-8
Random Number Generator	5-9
File and Record Control I/O Error Recovery.	5-10

TABLES

Table 2-1	Input/Output Library Subroutines.	2-3
Table 2-2	Linked Binary Subroutines and Entry Points.	2-6
Table 2-3	Initialization of I/O	2-13
Table 2-4	List and End-of-File Processing	2-14
Table 3-1	Miscellaneous Library Subroutines	3-2
Table 3-2	Error Codes and Returns	3-19
Table 4-1	Mathematical Library Subroutines.	4-3
Table 5-1	Nonmathematical Library Subroutines	5-1



SECTION I

INTRODUCTION

FORTRAN is an automatic coding language especially suited to solving complex scientific and engineering problems. This capability is greatly enhanced by the use of subroutines from the Input/Output (I/O) and math libraries to perform the basic elements of the FORTRAN language - arithmetic, control, and input/output.

This manual describes the subroutines that make up the FORTRAN Library.



SECTION II

INPUT/OUTPUT LIBRARY

The FORTRAN Input/Output (I/O) subroutines perform the functions associated with the execution of and input/output requirements of the user's program. The descriptions in this section define the general mode of operation of the subroutines that constitute the FORTRAN I/O Library. In these descriptions, it is assumed that the reader is familiar with the manner in which the General Loader processes General Comprehensive Operating Supervisor (GCOS) and \$ FFILE control cards.

The I/O subprograms require that program execution be started with the FORTRAN initialization subprogram .FSETU, because one of its functions is the initialization of fault vector cell 25 (octal). This location contains the address where the "logical file/file control block" table begins. The user can also accomplish this initialization by calling SETLGT when he has created his own table. The library subroutines also depend on the limits of unused memory being expressed in fault vector cell 37 (octal). This is always done by the General Loader. When the \$ OPTION FORTRAN card is encountered in the batch mode, the General Loader ensures that the FORTRAN initialization subprogram is loaded. In the time sharing mode, the RUN subsystem ensures that the proper initialization subprogram is loaded.

LIBRARY CALLS

A call to any I/O Library subroutine from a FORTRAN language program contains, as one of the arguments, the logical file code expressed as an integer. This integer is placed in character position 5 (bits 30-35) of cell .FBAD. (defined in subroutine FOPEN) by the called I/O subprogram. The called subroutine then calls FOPEN which searches the "logical file - file control block" table defined as follows:

1. Fault vector cell 25 (octal) is of the form

```
ZERO TAB,0
```

2. The actual logical unit (LGU) table has the form

```
ZERO ENDTAB,0
```

```
TAB VFD 18/FCB1,6/LGU1,6/LGU2,6/LGU3
```

```
VFD 18/FCB2,6/LGU4,6/LGU5,6/LGU6
```

```
.
```

```
.
```

```
ZERO 0,0
```

```
.
```

```
.
```

```
ENDTAB ZERO 0,0
```

- where:
- a. TAB - 1 contains the address of the last available location in the table.
 - b. FCBl contains the address of cell LOCSYM of file control block 1.
 - c. LGU1, LGU2, LGU3 are the FORTRAN logical files that reference file control block 1. Missing files are filled in with zeros.

If more than three logical files reference the same file control block, the FCBl pointer and the additional files can occur at any other place in the table.

There are as many entries in the table as needed to express the various file control blocks and logical files referencing them. After the last entry in the table, zeros fill out the table.

FOPEN places the address of the file control block for the referenced file in bits 0-18 of cell .FBAD., but does not destroy the logical file code in bits 30-35 (character position 5) of that cell. FOPEN then proceeds to open the file and return.

The subroutine that called FOPEN now has the information necessary to perform calls to the proper File and Record Control subprograms. In the case of an output file, bits 30-35 of .FBAD. are used as the report code of the output record. Thus, if many logical files are connected to SYSOUT, they are separated automatically at printing time according to the calling code originally specified in the FORTRAN program calling sequence.

INPUT/OUTPUT LIBRARY SUBROUTINES

The input/output library subroutines are summarized in Table 2-1. Table 2-1 gives the input/output function and the library subroutine that performs the function in the different modes -- batch and time sharing and using the BCD and ASCII character sets. Subroutines can be categorized as those implicitly called and those that are user callable.

Table 2-1. Input/Output Library Subroutines

Function	Batch		Time Sharing	
	BCD	ASCII	BCD	ASCII
Linked Binary I/O Interface	FRDB	FRDB	FRDB	FRDB
Short List Binary I/O Interface	FBLO	FBLO	FBLO	FBLO
Short List I/O Processor	FSLI	FSLI	FSLI	FSLI
Random Binary I/O	FRRD	FRRD	FRRD	FRRD
Format Controlled Sequential I/O (Common Procedure)	FDIO	FDIO	FDIO	FDIO
Format Controlled Sequential I/O (BCD/ASCII Procedure and Data)	FRDD	FRDA	FRDD	FRDA
Format Controlled Short List I/O Processor	FSLO	FSLO	FSLO	FSLI
Namelist Input (Common Procedure)	FNLI	FNLI	FNLI	FNLI
Namelist Input (BCD/ASCII Procedure and Data)	FVFI	FIFA	FVFI	FIFA
Namelist and Dump Output (Common Procedure)	FNLO	FNLO	FNLO	FNLO
Namelist and Dump Output (BCD/ASCII Procedure and Data)	FVFO	FOFA	FVFO	FOFA
Character String Assignment	FCHA	FCHA	FCHA	FCHA
Character String Compare	FCOM	FCOM	FCOM	FCOM
Output Stop and Pause Information	FPAW	FPAW	FPAW	FPAW
● Console Communications	FCSL	FCSL	FCSL	FCSL
Object Time Debug Processor	FDBG	FDBG	FDBG	FDBG
Memory Dump (Common Procedure)	FDMP	FDMP	FDMP	FDMP
● Memory Dump (BCD/ASCII Procedure and Data)	FDPD	FDPA	FDPD	FDPA
Pre-execution Initializer	FSTU	FSTU	FSTU	FSTU
Arithmetic Fault Processor	FFLT	FFLT	FFLT	FFLT
● Define Buffer(s) for File Control Block	FSET	FSET	FSET	FSET
● Define File Control Block	FSET	FSET	FSET	FSET
● Define Logical File Table	FSET	FSET	FSET	FSET
Backspace Record	FBST	FBST	FBST	FBST
● File Closing	FCLO	FCLO	FCLO	FCLO
Rewind and Endfile Processor	FEFT	FEFT	FEFT	FEFT
End-of-File (On Input) Processor	FEOF	FEOF	FEOF	FEOF
● Initialization of End-of-File Processing	FFEE	FFEE	FFEE	FFEE
● Initialization of Data Error Processing	FFER	FFER	FFER	FFER
File Opening	FOPE	FOPE	FOPE	FOPE
● Job Termination	FXIT	FXIT	FXIT	FXIT
● File Forward and Backspace	FFFB	FFFB	FFFB	FFFB
Carriage Control Simulator	FSLW	--	FSLW	FTGF
● Call TSS Subsystem	--	--	FCAL	FCAL
● Create TSS Temporary File	--	--	FDEF	FDEF
● Specify Record Size, Random Binary File	FRRD	FRRD	FRRD	FRRD
● Set or Reset Some I/O Parameters of Run-Time Library	FSTU	FSTU	FTGF	FTGF
● User callable				

SUBROUTINES IMPLICITLY CALLED

Linked Binary Input/Output Interface

FUNCTION

Linked Binary I/O Interface consists of two subroutines (FRDB and FBLO) called for linked binary input/output interface; for processing linked binary elements; linked binary arrays; and for end-of-list processing for input/output statements of the following forms:

```
READ (fc, END=S1, ERR=S2) list
```

```
WRITE (fc, END=S1, ERR=S2) list
```

fc = file code
END = optional transfer location for end-of-file
ERR = optional transfer for program error
list = integer, real, complex, double precision, logical, and character elements and arrays

The FBLO subroutine is called to process short list items; that is, arrays referenced without a subscript.

The following sequence results for a READ statement where the list is an integer, real, logical, double precision, or complex element.

TSX1	.FRDB.	}	Initialization for	
TRA	* + offset		}	READ
ZERO	.E.L., E.I.			
ARG	pointer to file code			
TRA	ERR-clause	}	List processing	
TRA	END-clause			
TSX1	.FBxT.			
STy	element	}	Signals end of record	
TSX1	.FRLR.			

x = L for integer, real, complex, or logical elements
= D for double precision elements
y = A for integer, real, complex, or logical elements
= AQ for double precision elements

The following list processing sequence results for a READ statement where the list is a character element.

```
EAA list  
TSX1 .FBBC. (or .FBBCA in ASCII mode)  
NOP element size, DL
```

The following list processing sequence results for a READ statement where the list is an integer, real, complex, double precision, character, or logical array.

```

TSX1      .FBxI. (.FXDI. for double precision arrays)
ARG       array locator
ARG       array size, DL (total number of words in array)

x = L for integer, real, complex, and logical arrays
  = S for character arrays

```

The following sequence results for a WRITE statement where the list is an integer, real, complex, double precision, or logical element.

```

TSX1      .FWRB.
TRA       * + offset
ZERO     .E.L.,E.I.
ARG       pointer to file code
TRA       ERR-clause
TRA       END-clause
LDx      element
TSX1     .FByT.
TSX1     .FWLR.

```

} Initialization for
WRITE

} List processing

} Signals end of output

```

x = A for integer, real, logical, or complex elements
  = AQ for double precision elements
y = L for integer, real, complex, or logical elements
  = D for double precision elements

```

The following list processing sequence results for a WRITE statement where the list is a character element.

```

EAA      list
TSX1     .FBBC. (or .FBBCA in ASCII mode)
NOP      element size, DL

```

The following list processing sequence results for a WRITE statement where the list is an integer, real, complex, double precision, character, or logical arrays.

```

TSX1     .FBxO. (.FXDO. for double precision arrays)
TRA      * + offset
ZERO     .E.L.,E.I.
ARG      array locator
ARG      array size, DL (total number of words in array)

x = L for integer, real, complex, or logical arrays
  = S for character arrays

```

The following table, Table 2-2, contains a summary of the linked binary subroutines and entry points.

Table 2-2. Linked Binary Subroutines and Entry Points

Type	Initialization		Element		Short List Input		Short List Output		End of Input		End of Output	
	Routine	Entry	Routine	Entry	Routine	Entry	Routine	Entry	Routine	Entry	Routine	Entry
Integer	FRDB	FRDB	FRDB	FBLT	FBLO	FBLI	FBLO	FBLO	FRDB	FRLR	FRDB	FWLR
Real	FRDB	FRDB	FRDB	FBLT	FBLO	FBLI	FBLO	FBLO	FRDB	FRLR	FRDB	FWLR
Complex	FRDB	FRDB	FRDB	FBLT	FBLO	FBLI	FBLO	FBLO	FRDB	FRLR	FRDB	FWLR
Double-Precision	FRDB	FRDB	FRDB	FBDT	FBLO	FBDI	FBLO	FBDI	FRDB	FRLR	FRDB	FWLR
Logical	FRDB	FRDB	FRDB	FBLT	FBLO	FBLI	FBLO	FBLO	FRDB	FRLR	FRDB	FWLR
Character	FRDB	FRDB	FRDB	FBBC	FBLO	FBSI	FBLO	FBSO	FRDB	FRLR	FRDB	FWLR

Subroutine FBLO calls FSLI to set up indexing for the processing of the input/output of nonsubscripted arrays.

CALLING SEQUENCE

See Function.

METHOD

Binary I/O is standard system format I/O unless otherwise specified and reads or writes media code 1 records. The binary I/O routine uses File and Record Control buffers, eliminating the need for an internal work area. Records are assumed to be variable length records and can be up to 4095 words in length. Logical, variable length records too large to fit in one buffer are partitioned and follow the rules for partitioned records as defined below.

1. Only logical, variable length (VLR) records too large to fit in one buffer are partitioned.
2. A partitioned record begins in a new buffer and ends in a buffer with no other records in that final buffer.
3. Partition indicators:

Initial Block:

RCW bits 24-25 = 01

Intermediate Blocks:

RCW bits 24-25 = 10

Bits 26-35 = Partition counter starting with 2 and advancing by 1 for each new block in the record.

Final Block:

RCW bits 24-25 = 11

Bits 26-35 = Partition counter described above.

Valid batch mode loader \$ FFILE options include NSTDLB, NBUFFS, BUFSIZ, FIXLNG, and NOSRLS. Additional information on FRDB can be obtained from documentation at the front of the assembly listing.

RETURN

Normal return is to the next executable statement in the calling program. Error conditions are handled as follows:

1. Error code 39 when an end-of-file condition, other than 17 or 23 (octal numbers), is detected.
2. Error code 33 because Read after Write is illegal.
3. Error code 40 when list exceeds logical record length. All remaining list items are set to zero.
4. Error code 41 when SYSOUT or fixed length records are not smaller than block size.

Short List Input/Output Processor

FUNCTION

FSLI provides list processing for a nonsubscripted array in conjunction with subprograms FSLO, FRDB, and FRRD.

CALLING SEQUENCE

The subprograms FSLO, FRDB, and FRRD have calling sequences to the subprogram FSLI to accomplish the data transfer.

For Formatted I/O (FSLO)

<u>Output</u>		<u>Input</u>	
E	STX2 .FSLII	E	STX2 .FSLII
	TSX2 .FSLI(m)		TSX2 .FSLI(m)
E	LD(r) **,2	E	TSX1 c
	TSX1 c		ST(r) **,2
E	ZERO p,0	E	ZERO p,1

m = O,I
r = A for single precision
= AQ for double precision
c = .FCNV. for single and double precision
= .FCNVI for integer
= .FCNVR for real
= .FCNVD for double precision
= .FCNVL for logical
= .FCNVC for character
p = 1 for single precision
= 2 for double precision

For Binary I/O (FRDB, FRRD)

LDA n,DU
(ADLA 1,DL) double precision only
XED .FSLIB,AU*
ARG ENDTL
ARG CRTAL
ARG SVX1

METHOD

For formatted I/O, FSLI initializes the I/O loop and processes the entire array, starting with the first element. For binary I/O, an EIS move is used to process the entire array.

RETURN

For formatted I/O, FSLI restores $C(X2) = C(.FSLII)0-17$ and exits to FSLO. For binary I/O, return is directly to the object code.

Random Binary Input/Output

FUNCTION

FRRD is called for random binary source statements of the following form:

```
READ(f'n,opt1,opt2) list
```

```
WRITE(f'n,opt2) list
```

f = a file reference
n = sequence number of the logical record
opt1 = end of file transfer (optional)
opt2 = error transfer (optional)
list = input/output list that can consist of single precision,
double precision, character, or array items

This subroutine also includes the RANSIZ (Random Record Size) subroutine that is called by the following sequence:

```
CALL RANSIZ (arg1,arg2,arg3)
```

arg1 = a file code
arg2 = the logical record size
arg3 = standard system format

When arg3 is missing or zero, the file is assumed to be in standard system format.

RANSIZ needs to be called only once for each file.

CALLING SEQUENCES

FRRD contains different entry points for I/O list and end-of-list processing. The sequences generated are similar to those generated for linked binary.

TSX1	.FRRD.	For READ (n',f) list
TSX1	.FRWD.	For WRITE(n',f) list
TSX1	.FRLT.	For single precision I/O list entry
TSX1	.FRDT.	For double precision I/O list entry
TSX1	.FRST.	For binary character (BCD)
TSX1	.FRSTA.	For binary character (ASCII)
TSX1	.FRLI.	For single precision short list input
TSX1	.FRDI.	For double precision short list input
TSX1	.FRSI.	For binary character short list input
TSX1	.FRLO.	For single precision short list output
TSX1	.FRDO.	For double precision short list output
TSX1	.FRSO.	For binary character short list output
TSX1	.FRRR.	For binary input end-of-list
TSX1	.FWRR.	For binary output end-of-list
CALL	RANSIZ	Random Record Size Routine

METHOD

Random binary files are standard system format (unless specified otherwise), 320-word blocks, with fixed record size. The module calculates logical record location based on physical device block size (64 words), logical record size (given by CALL RANSIZ command), buffer block size (320), and record number. The record size is compared to the buffer block size and if greater the file is considered partitioned and follows the rules for partitioned files of standard system format (see the File and Record Control manual).

The user should express record size as the sum of items in the I/O list in computer words, remembering that storage allocation for character variables and arrays differs when compiling BCD or ASCII (six characters versus four characters per word). The record size specified via RANSIZ should not include the record control word for the logical record in the size argument. An abnormal termination with an associated error message occurs when records are read that fall outside the file limits.

RANSIZ needs to be called once prior to the first file reference. FRRD calls FSLI for the processing of the input/output of nonsubscripted arrays. *

RETURN

Normal return is to the next executable statement of the calling program, except as specified below:

1. Error code 72 if the list exceeds logical record length.
2. Error code 73 for any of the following conditions:
 - a. Zero block count on random read.
 - b. Block serial number error on random read.
 - c. Record count of zero on random read.
3. Error code 74 if the file is not present.
4. Error code 75 for a zero or negative record number on a random read or write.
5. Error code 76 where the record size is not given. Random files have fixed length records. Record size must be specified by a CALL RANSIZ for the referenced file. This CALL should precede the first I/O to a linked file.
6. Error code 77 for an attempted random I/O to a linked file.
7. Error code 78 when the record number in the random read or write statement is outside file limits. However, if "END=" is given (for READ only), that exit is taken if the record number is outside the file limits. A write outside file limits causes the file space to grow.
8. Error code 79 when the list is greater than specified record size.
9. Error code 80 when file space is exhausted (on output, the file can no longer be "grown").

Format Controlled Sequential Input/Output

FUNCTION

Formatted sequential input/output is contained in five subroutines:

FDIO contains the procedures that are common to BCD and ASCII formatted sequential input/output.
FRDD contains the BCD peculiar procedures and data.
FRDA contains the ASCII peculiar procedures and data.
FSLO contains the short list input/output interface.
FSLI contains the short list input/output processor.

Formatted sequential input/output is accomplished by calls to these subroutines to initialize the input/output; to process the input/output list; and for end-of-file processing.

Subroutines FRDD and FRDA also contain the processing for the DECODE and ENCODE statements.

CALLING SEQUENCE

The calling sequences to initialize the input/output are as follows:

TSX1	entry point in FRDD or FRDA
TRA	* + offset
ZERO	.E.L.,E.I.
ARG	file code for formatted statements or 41,42,43 for READ, PRINT/WRITE, or PUNCH respectively.
ARG	format statement reference or FORMAT(V) for list directed input/output.
TRA	ERR-clause
TRA	END-clause

The subroutines and entry points for initialization of input/output are as listed in Table 2-3.

Table 2-3. Initialization of I/O

Statement Type	BCD		ASCII	
	Module	Entry Point	Module	Entry Point
WRITE (fc,format) list	FRDD	FWRD	FRDA	FWRDA
WRITE format, list	FRDD	FWRD	FRDA	FWRDA
PRINT format, list	FRDD	FWRD	FRDA	FWRDA
PRINT, list	FRDD	FWRD	FRDA	FWRDA
PUNCH format, list	FRDD	FPUN	FRDA	FPUNA
READ (fc,format) list	FRDD	FRDD	FRDA	FRDDA
READ format, list	FRDD	FRDD	FRDA	FRDDA
READ, list	FRDD	FRDD	FRDA	FRDDA

The calling sequence for list processing where list is integer, real, double precision, logical, or complex element is as follows:

```
LDx      element      if output
TSX1     .FCNVy
STx      element      if input
```

```
x = A for single precision items
    = AQ for double precision items
y = I for integer, R for real and complex,
    D for double precision, and
    L for logical
```

The calling sequence for list processing of character elements is as follows:

```
EAA      element
TSX1     .FCNVC
NOP      element size, DL
```

The calling sequence for all short list routines (except character) is as follows:

```
TSX1     short list routine
ARG      array locator
ARG      array size
```

The calling sequence for character short list input/output is as follows:

```

TSX1      .FSCx.
TRA       * + offset
ZERO      .E.L.,E.I.
ARG       array locator
ARG       array size, DL
NOP       element size, DL
  
```

```

x = 0 for output
  = 1 for input
  
```

The following table, Table 2-4, gives the subroutines and entry points for list processing and end-of-file processing.

Table 2-4. List and End-of-File Processing

Type	Element		Short List Output		Short List Input		End-of-File Output		End-of-File Input	
	Module	Entry Point	Module	Entry Point	Module	Entry Point	Module	Entry Point	Module	Entry Point
BCD										
Integer	FRDD	FCNVI	FSLO	FSIO	FSLO	FSII	FRDD	FFIL	FRDD	FRTN
Real	FRDD	FCNVR	FSLO	FSRO	FSLO	FSRI	FRDD	FFIL	FRDD	FRTN
Complex	FRDD	FCNVR	FSLO	FSRO	FSLO	FSRI	FRDD	FFIL	FRDD	FRTN
Double-Precision	FRDD	FCNVD	FSLO	FS20	FSLO	FS2I	FRDD	FFIL	FRDD	FRTN
Logical	FRDD	FCNVL	FSLO	FSB0	FSLO	FSBI	FRDD	FFIL	FRDD	FRTN
Character	FRDD	FCNVC	FSLO	FSCO	FSLO	FSCI	FRDD	FFIL	FRDD	FRTN
ASCII										
Integer	FRDA	FCNVI	FSLO	FSIO	FSLO	FSII	FRDA	FFILA	FRDA	FRTNA
Real	FRDA	FCNVR	FSLO	FSRO	FSLO	FSRI	FRDA	FFILA	FRDA	FRTNA
Complex	FRDA	FCNVR	FSLO	FSRO	FSLO	FSRI	FRDA	FFILA	FRDA	FRTNA
Double-Precision	FRDA	FCNVD	FSLO	FS20	FSLO	FS2I	FRDA	FFILA	FRDA	FRTNA
Logical	FRDA	FCNVL	FSLO	FSB0	FSLO	FSBI	FRDA	FFILA	FRDA	FRTNA
Character	FRDA	FCNVC	FSLO	FSCO	FSLO	FSCI	FRDA	FFILA	FRDA	FRTNA

The calling sequences for the DECODE and ENCODE statements are:

```

TSX1      .FDEC      for DECODE
TSX1      .FENC      for ENCODE
TRA       * +4 or 6
ZERO      .E.L.,E.I.
ARG       buffer
ARG       format
TRA       ERR-clause
TRA       END-clause
NOP       record size in characters, DL
NOP       buffer size in words, DL
  
```

The following calling sequences are used for internal conversion routines with format:

```
CALL .BDCNV (buf,form,wda,lines) memory to buffer
CALL .DBCNV (buf,form,wda,lines) buffer to memory
```

```
lines = number of lines specified by FORMAT statement.
wda   = word count of each line.
form  = starting location of format.
buf   = starting location of buffer.
```

METHOD

On the entry for initialization, the file code is determined and the file is opened and read if input. An appropriate tally word for data transfer is built. For output, the address of a 41-word buffer is used in the tally word. (This provides for a maximum length record of 160 characters). For input, the address of the File and Record Control buffer is used directly so no intermediate storage is necessary; this allows records up to 4095 characters in length to be processed. After initialization, the format scan begins and when a field separator or a right parenthesis is encountered, return is made to the user's generated code. The generated code causes control to transfer to the appropriate entry point for list processing. Thus, each list item corresponds to a format specification. This continues until all list items are processed, rescanning the format if necessary. The end-of-list entry determines that processing of the record is complete.

The processing for ENCODE and DECODE is the same except that the address of the in-memory buffer provided by the user is used in the data transfer tally word.

The routine FDIO contains code that is common for ASCII/BCD character sets while FRDA and FRDD contain ASCII and BCD unique code and symbols.

FSLI is called by FSLO to set up indexing for the processing of short list input/output.

RETURN

Normal return is to the next executable statement of the calling program, except as specified for the following error conditions:

1. Error code 31 for an illegal format statement. Format scan proceeds as for end of format.
2. Error code 32 for an illegal character in data or bad format. Data scan treats illegal character as zero.
3. Error code 33 if user attempts to read an output file. Execution is terminated.
4. Error code 34 for illegal end-of-file mark. FRWD calls FEOF for error recovery.

5. Error code 57 for illegal character for L conversion. Data scan treats illegal character as space.
6. Error code 81 for excessive line length.
7. Error code 82 for illegal character as first nonblank character. Execution is terminated.
8. Error code 86 for a recursive entry to the I/O routine.

Namelist Input

FUNCTION

The NAMELIST input function is contained in the following subroutines:

FNLI contains the procedures that are common to BCD and ASCII NAMELIST input.

FVFI contains the BCD peculiar procedures and data.

FIFA contains the ASCII peculiar procedures and data.

NAMELIST input is accomplished using File and Record Control variable length records for source statements of the following forms:

```
NAMELIST /naml/v,w,x
```

```
READ (fc,naml,END=S1, ERR=S2)
```

fc = logical file reference.

naml = NAMELIST name.

S1 = (optional) is the statement label pointing to the statement to which control is transferred upon encountering an EOF for a READ.

S2 = (optional) is the statement label pointing to the statement to which control is transferred upon encountering an error.

v,w,x = NAMELIST variables.

CALLING SEQUENCE

TSX1 .FVFI. (BCD)

TSX1 .FVFIA. (ASCII)

TRA * + offset

ZERO .E.L.,E.I.

ARG pointer to location of DEC FC

ARG pointer to location of BCI 1,NAM1

ARG pointer to location containing address of the EOF exit

METHOD

The input file is scanned for the proper NAMELIST name. When the name is found, it is verified that the variables are included in the NAMELIST and the input values are stored according to the type specified in the NAMELIST table. The entries for the above NAMELIST example (NAM1) are:

```

BCI      1,NAM1
BCI      1,V
TALLYD   LV,C,K
ZERO     N1,N2           Present only when K = 6 (character data)
ZERO     PROD,d1        if C = 1
ZERO     0/-1/d1*d2*d3,d1*d2   if C = 2
ZERO     -1/d1*d2...d5,d1*d2...d4  if C = 3
ZERO     -1, d1*d2 ..... d6      if C = 4
BCI      1,W
.
.
.
BCI      1,X
    
```

NAM1 is the NAMELIST name

LV is the location of the variable

V is the variable

C = 07777 if V is not dimensioned
 = 1 if V has 1 or 2 dimensions
 = 2 if V has 3 or 4 dimensions
 = 3 if V has 5 or 6 dimensions
 = 4 if V has 7 dimensions

K is the data type as follows:
 K = 1 for integer
 = 2 for real
 = 3 for double precision
 = 4 for complex
 = 5 for logical
 = 6 for character

PROD is the product of all dimensions

N1,N2 are present when K=6 (character data)

N1 = number of words per element

N2 = number of characters per element (maximum of 120)

d₁,d₂, ...,d_n are the dimensions for the variable

Bits of index register 6 are set (=1) to indicate the following:

<u>Bit number</u>	<u>Meaning</u>
0	Logical variable
1	Integer variable
2-4	Free
5	Slash preceded current variable name
6	Complex mode not allowed
7	Complex mode
8	Neither star nor character field allowed
9	Decimal point present
10	Negative number
11	Negative decimal exponent
12	Decimal exponent allowed
13	D field allowed
14	E field allowed
15	No decimal point allowed
16	Two words per element
17	One word per element

EXAMPLE

The following example shows the expansion using variable V with 0,1,2,3,4 dimensions (these are designated as V0,V1,V2,V3,V4). The dimensions (d_i) are designated as L,M,N,O.

V0 = undimensioned
V1 (L)
V2 (M,N)
V3 (O,P,Q)
V4 (R,S,T,U)

L through U are integers

1. With V not dimensioned

```
BCI      1,V0  
TALLYD  V0,O7777,K
```

2. V with a dimension of 1

```
BCI      1,V1  
TALLYD  V1,1,K  
ZERO    L,L
```

3. V with a dimension of 2 (V2)

```
BCI      1,V2  
TALLYD  V2,1,K  
ZERO    M*N,L
```


4. V with a dimension of 3 (V3)

```
BCI      1,V3
TALLYD   V3,2,K
ZERO     O*P*Q,O
ZERO     0 or -1,O*P
```

5. V with a dimension of 4 (V4)

```
BCI      1,V4
TALLYD   V4,2,K
ZERO     R*S*T*U,R
ZERO     R*S*T,R*S
```

6. Where K = 6 (character data), there is a slight variation in the expansion for ASCII and BCD character data. For example,

CH*10 will cause the following expansion for BCD:

```
BCI      1,CH
TALLYD   CH,07777,6
ZERO     2,10
```

CH*10 will cause the following expansion for ASCII:

```
BCI      1,CH
TALLYD   CH,07777,6
ZERO     3,10
```

RETURN

Normal return is to the next executable statement of the calling program. Under the error conditions listed below, execution is continued only if the user initialized FLGERR, which causes a normal return with an indication that bad data was encountered, or if the ERR = option was used. FLGERR and ERR = can be used in any combination.

1. Error code 42 for illegal heading card. FNLI continues as for end-of-data.
2. Error code 43 for illegal variable name. FNLI continues as for end-of-data.
3. Error code 44 for illegal subscript or when the array size is exceeded. FNLI continues as for end-of-data.
4. Error code 45 for illegal character after right parenthesis. Data scan assumes a comma between right parenthesis and the next character.
5. Error code 46 for illegal character in the data. Data scan treats the illegal character as zero.
6. Error code 48 for illegal logical constant. Data scan treats an illegal constant as .FALSE.
7. Error code 52 for an illegal character field. FNLI continues as for end-of-data.
8. Error code 59 for an empty character field.

Namelist and Dump Output

FUNCTION

Three subroutines process NAMELIST, DEBUG, DUMP, and PDUMP output using standard File and Record Control variable length records for source statements of the following forms:

```
NAMELIST /naml/v,w,x
```

```
WRITE (fc,naml,ERR=S2)
```

or

```
CALL DUMP (arguments)      BCD
CALL DUMPA (arguments)     ASCII
CALL PDUMP (arguments)     BCD
CALL PDUMPA (arguments)    ASCII
```

DEBUG is invoked via General Loader control cards.

The subroutines that process NAMELIST and DUMP output are as follows:

FNLO contains the procedures that are common to BCD and ASCII NAMELIST output.

FVFO contains the BCD peculiar procedures and data.

FOFA contains the ASCII peculiar procedures and data.

fc = logical file reference
S2 = (optional) is the statement label pointing to the
 statement to which control is transferred upon
 encountering an error
arguments = see "Memory Dump" for the memory dump subroutines, in this
 section
v,w,x = NAMELIST variables

CALLING SEQUENCE

```
TSX1 .FVDO (For DEBUG, DUMP, or PDUMP output)
TSX1 .FVFO (For NAMELIST output)
TRA * + offset
ZERO .E.L., E.I.
ARG pointer to location of DEC FC
ARG pointer to location of BCI 1,NAM1
ARG pointer to location containing address of error exit
ARG pointer to location containing address of EOF exit
```

METHOD

These subroutines scan a NAMELIST table and print the current value of each NAMELIST variable in the format specified by its entry in the table. Refer to the description of NAMELIST input for a description of the entries for NAMELIST, DUMP (DUMPA), and PDUMP (PDUMPA).

For NAMELIST, DUMP, and PDUMP output or for NAMELIST input, the entry for a NAMELIST variable, V, is of the following form:

BCI	1,V	always present
TALLYD	LV,D,K	always present
ZERO	PROD,D1	present if D is 1 or 2
ZERO	0,D1*D2	present if D is 2

LV is the location of variable V (must not be zero)

D is octal 7777 if V has no dimensions
1 if V has 1 or 2 dimensions
2 if V has 3 dimensions

D1 and D2 are the first and second dimensions (if any)
PROD is the product of the dimensions

K = 0 for octal
= 1 for integer
= 2 for real
= 3 for double precision
= 4 for complex
= 5 for logical
= 6 for character

For DEBUG output, the entry for variable V is:

BCI	1,V	always present
TALLYD	LV,D,K+16	always present
ZERO	I2,I1	present if D is not octal 7777
I1 and I2		are the initial and final subscripts respectively, in increments of D (or 1 if D is zero)

If V is octal 777777777777, the output is a memory dump from I1 to I2 in increments of D elements (1 or 2 words). For memory dumps, LV is ignored and can be octal 7777.

K is the same as for NAMELIST

If V is not octal 777777777777, LV and K are the same as for NAMELIST and the octal value 7777 for D indicates that LV has no dimensions.

All arrays in the DEBUG mode are considered as one-dimensional arrays.

RETURN

Normal return is to the next executable statement in the calling program.

Character String Assignment

FUNCTION

FCHA moves character strings from one character variable to another when the character variable is used in an assignment statement.

CALLING SEQUENCE

The following calling sequence is for $A = B$ where A and B are character variables.

```
TSX1   .FCHM. (or .FCHMA for ASCII)
TRA    * + 6
ZERO   .E.L..., E.I.
ARG    B,I
ARG    J,I
ARG    A,I
ARG    I,I
```

B = location of sending field
J = location of variable containing size (in characters) of B
A = location of receiving field
I = location of variable containing size (in characters) of A

METHOD

The character string identified by character variable B is assigned to character variable A. If $\text{size } B > \text{size } A$, B is truncated at the size of A. If $\text{size } B < \text{size } A$, the remaining characters in A after the assignment are blank filled.

RETURN

Return is to the next executable statement in the calling program.

Character String Compare

FUNCTION

FCOM compares BCD or ASCII character strings for character variables appearing in logical or relational expressions of the form IF (A .EQ. B) where A and B are character variables.

CALLING SEQUENCE

```
TSX1  .FCOM (or .FCOMA for ASCII)
TRA   * + 6
ZERO  .E.L.,E.I.
ARG   A,I
ARG   I,I
ARG   B,I
ARG   J,I
```

A = address of character string A
I = address containing size of A in characters
B = address of character string B
J = address containing size of B in characters

METHOD

1. The length of both strings in words is determined. If the lengths are different, the difference is saved.
2. The two strings are compared on a word-for-word basis over equal length. Indicator register is set as follows:
 - a. If $A < B$, the carry indicator is set off
(At the first non-compare, the comparison stops; $>$ or $<$ is determined by the collating sequence of the character set.)
 - b. If $A = B$, the zero indicator is set
 - c. If $A > B$, the carry indicator is set
3. If the strings are equal over their equal length but one string is longer than the other, the difference is compared against blanks. If the difference is blanks, the strings are equal.

RETURN

Return is to the next executable statement in the calling program.

Output Stop and Pause Information

FUNCTION

FPAW outputs a message to the system console (user terminal for time sharing) when a PAUSE or STOP statement is encountered in the source program.

CALLING SEQUENCES

TSX1 .FPAW. (or .FPAWA for ASCII)
TRA *+2
ZERO .E.L.,E.I.

Generated for a PAUSE statement with no message.

TSX1 .FPAI. (or .FPAIA for ASCII)
TRA *+3
ZERO .E.L.,E.I.
ARG pointer to integer

Generated for a PAUSE statement with integer message.

TSX1 .FPAC. (or .FPACA for ASCII)
TRA *+4
ZERO .E.L.,E.I.
ARG pointer to character message
ARG number of characters, DL

Generated for a PAUSE statement with character string message.

TSX1 .FIXT. (or .FIXTA for ASCII)
TRA *+3
ZERO .E.L.,E.I.
ARG pointer to integer

Generated for a STOP statement with integer message.

TSX1 .FCXT. (or .FCXTA for ASCII)
TRA *+4
ZERO .E.L.,E.I.
ARG pointer to character message
ARG number of characters, DL

Generated for a STOP statement with a character string message.

METHOD

This routine returns the following information to the user terminal (time sharing user) or the system console (batch user) when the following statements are encountered:

<u>STATEMENT</u>	<u>MESSAGE</u>
PAUSE	PAUSE LINE # nnnn SNUMB xxxx-xx (batch only)
PAUSE integer	PAUSE nnnnnn SNUMB xxxxx-xx (batch only)
PAUSE character string	PAUSE user message
STOP integer	STOP AT LINE nnnn
STOP character string	STOP user message

RETURN

Normal return is to the next executable statement in the calling program.

Object Time Debug Processor

FUNCTION

FDBG decides whether or not to produce DEBUG output based on the contents of the IF and FOR statements.

CALLING SEQUENCE

FDBG is entered by a DRL (Derail) instruction.

METHOD

The location of the DRL instruction by which FDBG was entered is checked to see if it was inserted by the General Loader (a legal DEBUG request), or if it was originally present in the interrupted program. If it was a DEBUG request, the IF and FOR statements are examined in the order in which they were specified. If these statements are satisfied, FDBG writes information describing the particular input it is interrupting. It then calls FVFO or FOFA for the list output, using the NAMELIST table supplied in the DEBUG table by the General Loader. The instruction replaced by DRL is then executed and control is returned to the next instruction in the interrupted program. If DRL was not inserted by the General Loader, it is ignored and control is returned to the next instruction in the interrupted program. For additional details on DEBUG, see the General Loader manual.

FDBG assumes that the General Loader has placed a debug table in memory.

RETURN

Normal return is to the next statement in the interrupted program.

Pre-execution Initializer (Batch)

FUNCTION

FSTU performs certain initialization functions prior to the execution of the user program. The subroutine used depends on the system installation.

CALLING SEQUENCE

FSTU is entered by the General Loader only. The entry point is .SETU.; the General Loader assumes a six-position storage block (cells .SETU.-1 to .SETU.-6 inclusive).

<u>CELL</u>	<u>DEFINITION</u>
.SETU.-6	Batch/time sharing flag. Set to zero.
.SETU.-5	Upper half contains lowest address of memory used by the program and the LABELED COMMON region; lower half contains the highest memory address used in BLANK COMMON region.
.SETU.-4	Logical unit table pointer in address field.
.SETU.-3	Upper half = lowest cell used by the program. Lower half \neq 0 = address of pointer to DEBUG subroutine in link 0.
.SETU.-2	Memory reset constant.
.SETU.-1	Upper half = entry point address. Lower half \neq 0 indicates low-load job.

METHOD

FSTU clears unused memory and sets it to either the constant specified in the \$ OPTION control card or to zeros if the constant is not defined. It places the address of the "logical file - file control block" table in fault vector location 25 (octal). In a link job with DEBUG requested in link 0, it places a transfer to the DEBUG table in fault vector location 15 (octal). It places the entry point and a bit indicating a low-load job in fault vector location 24 (octal). It then calls the subroutine .FLTPR to initialize for fault processing. A secondary SYMDEF, .FLTPR, is imbedded in the Execution Error Monitor subroutine to satisfy this SYMREF in a FORTRAN execution. This routine places transfers to fault processing routines in fault vector locations 7 and 11 (both octal numbers). These are also imbedded in the Execution Error Monitor subroutine. There also exists in the library a separate subroutine with SYMDEF .FLTPR which zeros fault vector locations 7 and 11 (both octal numbers) (divide check and overflow). The default option, if either condition occurs, is to abort. This subroutine is used if the job uses no FORTRAN library subroutines. Either of the .FLTPR subroutines returns to FSTU, which zeros out all index registers and performs a TSX1 to the real entry point of the user's program. FSTU can be easily changed by an installation to perform the fault processing, accounting techniques, etc., for the particular installation.

RETURN

Normal return is to the next executable statement of the calling program.

RESTRICTIONS

If the user requires initialization of the cells specified in FSTU, he must either use this subroutine or supply one of his own to perform this initialization.

Pre-execution Initializer (Time Sharing)

FUNCTION

FTSU performs the same function for time sharing that FSTU performs for batch programs. In addition, FTSU sets up the file control block and logical unit table (performed by the General Loader in a batch environment) for all files specified in the time sharing RUN command. Cell .SETU.-6 is set to nonzero so that it can be used as a test for batch/time sharing. Refer to the description of FSTU (batch) for additional details on FTSU.

Arithmetic Fault Processor

FUNCTION

FFLT zeros fault vectors for overflow, divide check, and underflow faults; called by FTSU and FSTU if FXEM is not loaded. (FXEM has its own version of .FLTPR that initializes fault vectors.)

CALLING SEQUENCE

CALL .FLTPR

RETURN

After the execution of the subroutine, control is returned to the calling program.

Backspace Record

FUNCTION

FBST backspaces one logical record of a file on magnetic tape or sequential disk.

CALLING SEQUENCE

In FORTRAN (indirect):

BACKSPACE n

For GMAP:

CALL .FBST.(n)

n = logical file to be backspaced

METHOD

If the file has buffers and the device is magnetic tape or disk, the current record index is examined. If the current record index is for the first logical record in a block of logical records, a further check is made to determine if partitioned (segmented) records are present. When partitioned records are found, the control word of the current record is checked to determine if the segment now in memory is the last segment of a partitioned record. If so, the file is backspaced until it is positioned at the beginning of the first segment of the record. At exit, the "next record not there" bit is on, and the file is in the input mode.

If the current record index is not for the first logical record in the block, or if the record is not partitioned, the pointer in the file control block and buffer control word are altered. This indicates that the logical record preceding the desired logical record has just been read.

If positioned at end-of-file, the execution of one BACKSPACE positions the file so that a WRITE statement allows appending to the file, while a READ repeats the end-of-file exit if mass store, or an abnormal termination if magnetic tape.

The execution of two BACKSPACE commands positions the file so that a READ statement will obtain the last record prior to the end of file.

RETURN

Normal return is to the next executable statement of the calling program, except under the following error conditions:

1. Error code 47 if the logical file is assigned to SYSOUT or to a device other than magnetic tape or disk. The Execution Error Monitor terminates this execution.
2. Error code 49 if an erroneous end-of-file appears. The Execution Error Monitor terminates this execution.
3. Error code 50 if the backspace request is refused because the block count in the file control block is zero; i.e., inhibits backing out of the file. A message is written on SYSOUT to indicate the refusal, and the operation continues.

Rewind and Endfile Processor

FUNCTION

FEFT, for FORTRAN statement ENDFILE(i), writes an end-of-file record on the file. If the file is in input mode prior to executing ENDFILE, FEFT changes the mode to output (assuming there is no error condition). FEFT, for FORTRAN statement REWIND(i), rewinds magnetic tape or disk. If the file is in output mode prior to executing REWIND, FRWT writes an end-of-file record prior to the rewind operation. If the exit from FRWT is without an error condition, the file is in the input mode.

CALLING SEQUENCES

CALL .FEFT.(arg) compiled for FORTRAN statement ENDFILE(i)

CALL .FRWT.(arg) compiled for FORTRAN statement REWIND(i).

arg is the location of

DEC i

METHOD

1. ENDFILE - .FEFT. sets the file designator word options to provide the following responses:
 - a. File initially open and in input mode: FEFT calls GSTOT to change the mode to output, and then calls GCLSE to write an end-of-file record and close without rewind.
 - b. File initially open and in output mode - FEFT calls GCLSE to write an end-of-file record and close without rewind.
 - c. File initially closed - FEFT calls .GOPEN to open the file as output without rewind, and then calls .GCLSE to write an end-of-file record and close without rewind.
2. REWIND - FRWT sets the file designator word options to provide the following responses:
 - a. File initially open and in input mode - FRWT calls GCLSE to close the file with rewind.
 - b. File initially open and in output mode - FRWT calls GCLSE to write an end-of-file record and close the file with rewind. Following the rewind, the file is set to the input mode.
 - c. File initially closed - FRWT calls GOPEN to open the file as input with rewind, and then calls GCLSE to close the file.

RETURN

Normal return is to the next executable statement in the calling program, except for error condition 35. Error code 35 applies if there is an attempt to rewind or write an end-of-file record if the device assigned to the file is SYSOUT or is not magnetic tape or disk. Execution is continued; however the endfile/rewind request is ignored.

End-of-File (On Input) Processor

FUNCTION

FEOF writes an end-of-file message and either terminates execution or returns to the calling program when an end-of-file on input is encountered.

CALLING SEQUENCE

CALL .FEOF.

It is assumed that the address of the proper file control block is in the FOPE subprogram cell .FBAD.

METHOD

If a library subroutine detects an end-of-file on an input file, it calls FEOF. FEOF places the file number in the message and calls the Execution Error Monitor to print the message.

RETURN

FEOF selects one of the following:

1. Terminates execution with error code 34 writing end-of-file message.
2. If the user has provided for end-of-file condition by previously calling FLGEOF for the current file, returns to the calling program, indicating an end-of-file was encountered.

File Opening

FUNCTION

FOPE selects and assures that the physical file associated with the specified logical file is open.

CALLING SEQUENCE

1. CALL .FGTFB obtains the file control block address that is in the field of cell .FBAD. only.
2. CALL .FOPEN(s) obtains the information described in CALL .FGTFB and assures that the logical file is open. S indicates the mode in which the file is to be opened:

s = -1,DL, open the file in its previous mode.
s = Odd, open the file as output.
s = Even, open the file as input.
If s ≠ -1,DL, only bit 35 is examined.

In both these calling sequences, it is assumed that upon entry to FOPE the logical file references are contained in character position 5 of cell .FBAD.

NOTE: When a file is opened, a table is used to find the available buffers that can be assigned to the file being opened. The table of standard length reusable buffers (321 words) is defined as in .FBFTB.

An equivalent entry to .FOPEN is .FXOP., which is used by the Execution Error Monitor to prevent destroying calling sequences in case of recursive entry. This entry is used to open P* in order to write an error message.

3. CALL FLGFRC (lgu, ptr) allows the user to set his error routine address in a file control block in the case of a File and Record Control error.

lgu = numeric file code
ptr = address of recovery routine

See "File and Record Control I/O Error Recovery" in Section V.

METHOD

FOPE is performed in three phases: (1) locating the physical file, (2) assigning buffers to the file, and (3) assuring that the file is open. FOPE examines the logical file table for a logical file identical to the one in cell .FBAD. The file control block is examined to determine if buffers are required that have not previously been assigned. If buffer assignment is necessary, table .FBFTB is examined to see if any buffers have been released that were previously assigned to another file. These buffers are assigned first. If none of these buffers are available, buffers are assigned from available unused memory. The file control block is again examined to see if the file is open. If it is not open, the File and Record Control subprogram OPEN is called for the proper file control block. Control is returned to the calling program.

By the use of FLGFRC, the user can cause word -15 of the file control block to point to his error routine. If word -15 is zero when a file is first referenced, no change is made to the file control block. However, if word -15 is not zero, then a pointer to a translation routine within FOPE is stored in word -5, so that a File and Record Control error results in a trap to FOPE, an error code 85 message, and a transfer to the user error routine in word -15. When running in batch mode, a user error routine can be specified by use of a \$ FFILE card. This causes the address of the error routine to be placed in word-5 of the file control block, thus avoiding the trap to FOPE on a File and Record control error.

RETURN

Normal return from .FGTFB and .FOPEN is to the next executable instruction in the calling program, except for the following error conditions:

1. Error code 37 if the logical file requested is not in the logical unit table. FXER terminates execution.
2. Error code 38 if there is not enough memory available for I/O buffer assignment. FXER terminates execution.
3. Error code 56 if there is an attempt to read SYSOUT. FXER terminates execution.
4. Error code 54 if there is an attempt to write GIN. FXER terminates execution.
5. Abort code Q2 if no logical file table exists.
6. Abort code Q3 if logical file 06 does not exist in the logical unit table. A message from FXER cannot be written.

Carriage Control Simulator

FUNCTION

FSLW formats FORTRAN generated print lines for the printer.

CALLING SEQUENCE

CALL .FSLEW(pl)

pl = location of the print line. It is assumed that location .FBAD. in subroutine FOPE contains the address of the file control block for the output file, and that word +1 of the file control block contains the size of the print line.

METHOD

FSLW is called by the FORTRAN I/O routine FRWD (I/O Interface by Format Control). Control is passed to FSLW after each print line has been prepared according to the format specification. Recognized carriage control characters are 0, 1, +, and Ø, and FSLW looks to see if the first character of the prepared print line is one of these.

If the first character is not a recognized carriage control character, FSLW assumes the normal case: single space carriage positioning (Ø). FSLW therefore appends one word of single space slew information to the current print line.

If the first character of the prepared print line is a recognized carriage control character, FSLW proceeds as follows:

1. Ø - Single Space - This is the normal case, single space carriage positioning (Ø).
2. + - Space Suppress - The single space slew information appended to the previous print line is replaced by space suppress slew information.

NOTE: For space suppression control, two conditions are assumed: (1) that the last record written on the current file is the line on which overprinting is desired and (2) that its slew information can be changed. Both conditions must be satisfied for proper operation.

3. 1 - Eject Before Printing - A one-word print line (that is, a one-word record) consisting of slew-to-top-of-page information is generated. This causes a slew to top of page to follow immediately after the single space resulting from the information appended to the previous line.

4. 0 - Double Space - A one-word print line (that is, a one-word record) consisting of single space slew information is generated. This information and the single space information appended to the previous print line result in a double space operation.

Besides taking the actions described above for the recognized carriage control characters, FSLW sets any such control character to a blank (Ø), if it is not already a blank.

The NOSLEW option on the \$ FFILE card causes bit 23 of FCB word -6 to be set to 1.

FSLW recognizes this option and changes its normal operation as follows:

1. The addition of a slew word at the end of a data record is inhibited.
2. The generation of one-word print lines containing only slew information is inhibited.
3. The substitution of a blank character for the carriage control character (first character of data record) is inhibited.
4. A media code 0 is stored in the record control word in place of media code 3.
5. The "blank line" records generated by consecutive slashes in FORMAT statements are represented by one-word records consisting only of blanks. (In the absence of the NOSLEW option, consecutive slashes in FORMAT statements cause one-word records containing slew characters for single-line slews to be generated.

RETURN

Normal return is to the next executable statement of the calling program.

SUBROUTINES THAT ARE USER CALLABLE

Console Communication

FUNCTION

FCSL permits operator-program communication via the console; restricted to batch mode, not executable in time sharing.

CALLING SEQUENCE

CALL CNSLIO (console,message,nwords,nreply,nrepws)

console = Any of - BCI 1,0000T/ for master console,
BCI 1,0000T* for tape console,
BCI 1,0000*T for unit record console,
BCI 1,0000/T for special purposes.

If CONSOLE is none of these, BCI 1,0000/T is used.

message = an array containing one line of the message (in BCI) to be output. The message, as received, is prefixed by the SNUMB: (SLEW) SSSSS-AA; it will be suffixed by "07701".

nwords = number of words to be output. Any value greater than 11 is set to 11.

nreply = optional and used if a reply is desired (it need not be any particular type). The reply is limited to a maximum of six characters unless "nrepws" is supplied.

nrepws = contains the maximum length in words (up to 11) of the reply. Characters are leftjustified in the reply word.

METHOD

FCSL outputs the message to the console specified by the calling sequence. If the console specification is not one of the four valid specifications, the message is output to the master console. This subroutine takes nwords (11 maximum) starting at message, prefixes the message with SNUMB and suffixes the message with a one line slew, and sends the message to the console. If a reply is called for (nreply used), the reply is limited to six characters, and is left justified. (Nreply is optional.)

The program relinquishes control until the console I/O is completed. The registers and indicators are restored and a return is made to the next executable statement in the calling program.

RETURN

Return is to the next executable statement of the calling program and is done after the console input/output is completed. All registers and indicators are restored prior to the return.

RESTRICTIONS

Message may not exceed 11 words. Anything in excess of 11 words is ignored. Nwords = 0 results in only the SNUMB being output.

The reply buffer area (nreply) is not cleared by FCSL. If required, the user must clear this area prior to the call to CNSLIO.

Memory Dump

FUNCTION

FDMP dumps registers and all of memory or designated areas of memory (that has been allocated to variables) in a specified format.

CALLING SEQUENCE

CALL DUMP (a₁,b₁,f₁,....a_n,b_n,f_n) for BCD
CALL DUMPA (a₁,b₁,f₁,....a_n,b_n,f_n) for ASCII
CALL PDUMP (a₁,b₁,f₁,....a_n,b_n,f_n) for BCD
CALL PDUMPA (a₁,b₁,f₁,....a_n,b_n,f_n) for ASCII

a and b are variables at the beginning and end of the area to be dumped. a or b may represent the first and last variables in the program unit, in which case all memory allocated to the variables is dumped.

f is an integer specifying the dump format as follows:

f_i = 0 Octal
 = 1 Integer
 = 2 Real
 = 3 Double precision
 = 4 Complex
 = 5 Logical
 = 6 Character

If f_i is omitted, it is assumed to be zero. If no arguments are given, all of memory, including the program object code, is dumped in octal.

METHOD

An appropriate NAMELIST table is created, using the parameters specified in the calling sequence. FDMP calls FNLO for the actual NAMELIST output processing. The panel is dumped, followed by the blocks of memory requested.

RETURN

If DUMP (or DUMPA) is called, execution is terminated by a call to EXIT. If PDUMP (or PDUMPA) is called, the panel is restored and control is returned to the calling program.

File Control Block and Logical Unit Table Routines

FUNCTION

FSET contains three subroutines to allow the user to assign space in memory for use as an input/output buffer; to define a file control block for use by the input/output subroutines; and to define a logical unit table for use by the input/output library subroutines. Each of these three subroutines are described separately.

DEFINE FILE CONTROL BLOCK

FUNCTION

This entry point in FSET, SETFCB, allows the user to define a file control block for use by the I/O subroutines. SETLGT must be called first if the General Loader has not created any file control blocks.

CALLING SEQUENCE

CALL SETFCB(a,i,j...)

a = location of LOCSYM in the user created file control block
i,j... = logical files that refer to this file control block

METHOD

SETFCB searches the previously defined logical file table for an open space to insert the reference to the file control block. It accepts the file control block address and appends characters 3,4,5 to the various logical file codes referring to this file control block. SETFCB makes as many entries as necessary in the "logical file - file control block" table.

RETURN

Normal return is to the next executable statement of the calling program except when error conditions are encountered. Possible error conditions are:

1. Abort code Q2 if there is no logical file table.
2. Abort code Q1 if there is no space available in the logical file table for inserting a specified file control block.

DEFINE BUFFER(S) FOR A SPECIFIED FILE CONTROL BLOCK

FUNCTION

This entry point in FSET, SETBUF, allows the user to assign space in memory for use as an input/output buffer.

CALLING SEQUENCE

CALL SETBUF (i,a)

CALL SETBUF (i,a,b)

i = logical file designator

a = location of first buffer

b = location of a second buffer if necessary

METHOD

SETBUF searches the logical file table for the specified file and its associated file control block. It then attaches the buffers defined to the file control block. No check is made to verify that the buffers are of sufficient size, as this is the user's responsibility.

RETURN

Normal return is to the next executable statement in the calling program.

RESTRICTIONS

The size of the total buffer must be one location greater than the area to be used for actual record storage. Therefore standard buffer size is 321 words.

DEFINE LOGICAL UNIT TABLE

FUNCTION

This entry point in FSET, SETLGT, allows the user to define a logical unit table for use by the I/O library subroutine.

CALLING SEQUENCE

CALL SETLGT(a,i)

a = location of logical unit table to be used
i = number of cells in table a

METHOD

SETLGT accepts the array specified by the user as the logical unit table. It changes its first location to be a pointer to the last usable position of the array and places the address of the array + 1 in fault vector location 25 (octal).

RETURN

Normal return is to the next executable statement of the calling program.

RESTRICTIONS

SETLGT must be called before any input/output is requested. SETLGT is called only when the user wants to suppress the logical file table generated by the General Loader and to place the table in his own portion of memory. The user should use the NOFCB option on the \$ OPTION General Loader control card.

File Closing

FUNCTION

FCLO closes a file and releases the buffer assigned to that file. The buffer is released only if it is standard size (321 words).

CALLING SEQUENCE

CALL FCLOSE(u)

u = logical file number

METHOD

FCLO opens the file in its previous mode and calls the File and Record Control routine to close the file without rewind. FCLO examines and releases any standard size buffers assigned to the file. It places the memory address of these buffers in a table of available buffers (location .FBFTB in the FOPE subroutine) for possible reassignment to a newly opened file.

RETURN

Normal return is to the next executable statement in the calling program.

RESTRICTIONS

If more than one logical file refers to one physical file, the physical file must be closed, using FCLO only once.

Initialization of End-of-File Processing

FUNCTION

FFEE provides a signal to FEOF requesting a return to the calling subroutine if an end-of-file condition occurs.

CALLING SEQUENCE

CALL FLGEOF(u,v)

u = logical file number

v = address of the variable used to indicate an end-of-file condition (user must test v when an end-of-file condition could have occurred)

METHOD

The address of the variable to be used for end-of-file processing is placed in word -15 (upper) of the file control block. The value of the variable is set to 0 (set to nonzero if an end-of-file is encountered on logical file u).

RETURN

Normal return is to the next executable statement in the calling program.

RESTRICTIONS

The user must observe the following restrictions on FFEE operation:

1. If more than one logical file refers to one physical file, the same variable must be used for all logical files referring to that physical file.
2. This call must be made prior to any reference to file u.

Initialization of Data-Error Processing

FUNCTION

FFER provides a variable used in detecting the occurrence of erroneous data.

CALLING SEQUENCE

CALL FLGERR(u,v)

u = logical file number

v = variable used to indicate an input data error (user should test v before processing the data because this subroutine gives a normal return)

METHOD

FFER places the address of the variable in word -16 (upper) of the file control block. The value of the variable is initialized to zero (an error detected in the input data sets the variable to nonzero).

RETURN

Normal return is to the next executable statement of the calling program.

RESTRICTIONS

If more than one logical file refers to one physical file, the same variable must be used for all logical files referring to that physical file.

Job Termination

FUNCTION

FXIT contains an entry point .FTERM that is used to terminate the current activity.

CALLING SEQUENCE

CALL .FTERM

METHOD

FXIT transforms the logical file table created by the General Loader or by FTSU into a file designator word list for closing all files. FXIT calls CLOSE which purges all buffers, writes an end-of-file on an output file, and notes the closing of an input file. Execution is terminated by a MME GEFINI or DRL RETURN. Control is returned to the General Comprehensive Operating Supervisor.

This call will terminate the current activity without checking the wrapup list that may have been generated via a CALL ATCALL or CALL NOCALL statement. (Refer to Appendix F, FORTRAN Debugging System, in the FORTRAN manual.) A CALL EXIT or CALL .FEXIT statement may also be used to terminate the current activity and the wrapup list will be inspected.

File Forwardspace and Backspace

FUNCTION

FFFB allows users who generate multifile tapes to space from one file to another (valid only with tape files).

CALLING SEQUENCE

CALL FILBSP(xx,n) Backspace n files

CALL FILEFSP(xx,n) Forwardspace n files

xx = filecode, integer constant or variable

n = number of files to skip; integer variable or constant

METHOD

This subroutine is called directly by the user. To ensure proper positioning, the current file, if output, should be closed with an ENDFILE xx statement and counted as one of the files to be backspaced over.

There are some file restrictions based on the File and Record Control.

1. File must be declared multifile.
2. File must be unlabeled.

This can be accomplished by using a \$ FFILE card as follows:

```
$ FFILE XX,MLTFIL,NSTDLE
```

RETURN

Normal return is to the next executable statement of the calling program. However, there is one error exit from the subroutine, error code 36. The user has requested a backspace larger than the number of files currently processed on the tape. The error message is printed and the tape is positioned at the first file; execution continues.

Call TSS Subsystem

FUNCTION

FCAL is used to provide the user with a method to call or go to a time sharing system subsystem.

CALLING SEQUENCE

CALL CALLSS (string)

or

CALL CALLSS (string, name)

string = ASCII character constant or variable that is the time sharing command to invoke the subsystem. The string must contain a carriage return or backward slash (octal 134) as a terminating character.

name = 4-character ASCII name of subsystem to be called

Return is to the next executable statement in the FORTRAN program.

CALLGT (string)

or

CALLGT (string, name)

String and name are as for CALLSS. The program terminates rather than returning.

Both calls are ignored in batch.

METHOD

DRL CALLSS or DRL T.GOTO are used.

Create TSS Temporary File

FUNCTION

FDEF is used to create a named temporary file and to access the file in a user's available file table (AFT).

CALLING SEQUENCE

CALL DEFIL (name, links, mode, istat)

name = up to 8-character variable or constant containing the ASCII name of the temporary file to be created
links = size in links of file to be created
mode = 0, sequential file is created
 ≠ 0, random file is created
istat = status return word as follows:
 0, successful
 3, no room in AFT
 4, temporary file not available
 5, duplicate file name
 6, no room in PAT

This call is ignored in batch.

METHOD

DRL DEFIL is used.

Specify Record Size, Random Binary File

FUNCTION

FRRD permits the user to specify the record size for a random binary file. Normal return is to the next executable statement of the calling program. If the record size for a given random file is not provided at load time via the \$ FFILE card, a call to this subroutine before opening (first I/O to) the file is mandatory.

CALLING SEQUENCE

CALL RANSIZ (u,n,m)

u = logical file designator
n = record size
m = file format indicator

u, n and m must be of type integer. They can be any legal arithmetic expression.

Note that a call to RANSIZ can also be used to override a \$ FFILE size specification and that this is the preferred method of specification since its function works for both batch and time sharing.

The third argument (m) is optional. When not supplied, file u will be processed in standard system format (blocked, variable length records, etc.). When supplied, zero indicates standard system format; non-zero indicates that block and record control words are not to be processed. This latter format provides compatibility with random files generated in the time sharing mode. The total file space is available for data; records are not blocked, can begin anywhere in a sector and can span device boundaries.

Set or Reset Some I/O Parameters of Run-Time Library

FUNCTION

This subroutine (FSTU for batch, FTGF for time sharing) permits the user to set or reset some of the I/O parameters of the run-time library. Specifically, it may be used to:

1. Set the line length (modulo 4) for formatted output directed to a terminal. The default setting for this parameter is 72.
2. Set the media code for unformatted file output. The default setting of this parameter is 1.
3. Set the reflexive read characters that are sent to a terminal to request input. The default setting of this parameter is the ASCII CHARACTER constant 'carriage return', 'line feed', 'equal sign', X-ON.

CALLING SEQUENCE

CALL FPARAM (i,j)

i = integer, with a value of 1, 2, or 3 corresponding to one of the three functions above

j = integer, providing the line length or media code for i values of 1 and 2, or providing the octal value of four ASCII characters for an i value of 3

SECTION III

MISCELLANEOUS LIBRARY SUBROUTINES

This section contains descriptions of library subroutines other than input/output. These subroutines are listed in Table 3-1. These subroutines can be categorized as those implicitly called and those that are user callable.

Table 3-1. Miscellaneous Library Subroutines

Function	Batch		Time Sharing	
	BCD	ASCII	BCD	ASCII
Double Precision Powers of Ten Table	FDPT	FDPT	FDPT	FDPT
● Exponent Register Overflow and Divide Check Tests	FDCK	FDCK	FDCK	FDCK
● Sense Light Simulator	FLIT	FLIT	FLIT	FLIT
● Sense Switch Test	FSWI	FSWI	FSWI	FSWI
● Restore Links During Execution	FLNK	FLNK	FTLK	FTLK
● Execution Error Monitor (Common Procedure)	FXER	FXER	FXER	FXER
● Execution Error Monitor (Batch/Time Sharing Procedures and Data)	FXEM	FXEM	FXMA	FXMA
● Restore Link - H*	FLHS	FLHS		
● File Transliteration	FMED	FMED	FMED	FMED
● Terminal Input Recovery	FRBC	FRCV	FRBC	FRCV
● ASCII/BCD Indicator	FBCD	FASC	FBCD	FASC
● Date and Time	FDTM	FDTM	FDTM	FDTM
● Access a Permanent File	FTAC	FTAC	FTAC	FTAC
● Close File, Detach Buffers, From AFT	FDTH	FDTH	FDTH	FDTH
● Attach a Temporary Mass Storage or File Core Allocator (FDS)	FCRA	FCRA	FCRA	FCRA
● Special Entry Point (FDS)	FALC	FALC	FALC	FALC
● FDEBUG Bootstrap (FDS)	FDBD	FDBD	FDBD	FDBD
● FDUMP Bootstrap (FDS)	FDEB	FDEB	FDEB	FDEB
● LINK/LLINK Interface (FDS)	FDUM	FDUM	FDUM	FDUM
● Delete From Wrapup List (FDS)	FLKL	FLKL	FLKL	FLKL
● Release Unused Memory (FDS)	FNCL	FNCL	FNCL	FNCL
● Dummy Setup (FDS)	FREL	FREL	FREL	FREL
● Add to Wrapup List (FDS)	FSTP	FSTP	FSTP	FSTP
● Timing Facility (FDS)	FTCL	FTCL	FTCL	FTCL
● Wrapup and Loader (FDS)	FTMR	FTMR	FTMR	FTMR
● Linking Subroutine (FDS)	FWRP	FWRP	FWRP	FWRP
	FYLK	FYLK	FYLK	FYLK
● User callable				

SUBROUTINES IMPLICITLY CALLED

Double Precision Powers of Ten Table

FUNCTION

FDPT stores a table of double precision powers of ten for quick reference by all decimal radix conversion routines.

CALLING SEQUENCE

.F1DO. is the only SYMDEF symbol.

METHOD

.F1DO. +2*n is the location of DEC 1.0Dn = 10^{*n} , for n = -34, -37, ..., -1, 0, 1, ..., 37, 38. There are no executable instructions.

Restore Link - H*

FUNCTION

FLHS reloads a program from an H* file (tape) generated in a previous General Loader activity. The H* file was generated by a \$ TAPE H* control card at execution time.

CALLING SEQUENCE

This program is called directly from the subroutine library and requires no other subprograms. The entire job could be set up as follows:

```
$ SNUMB
$ IDENT
$ USE      .LHSF
$ ENTRY    .LHSF
$ EXECUTE
$ LIMITS
$ TAPE     H*,...
$ DATA    AB (Optional)
. .
. .
$ ENDJOB
***EOF
```

If the NOFCB option was in effect when the General Loader generated the H* file, an entry card of the form:

```
$ ENTRY    .LHSNF
```

must replace the card following the \$ USE card.

METHOD

The H* file generated by the General Loader contains a link identified as //, which is the main or common subprogram of the job. If the FCB option was in effect during loading (generation of H* file), a second link identified as //1 containing all file blocks generated by the General Loader will also be present on H*. LHSF searches the file for these identifiers (//1 is optional), restores them, and enters the main subprogram at the entry location specified during the General Loader activity.

Under time sharing execution, the YFORTRAN RUN command restores the main link.

RETURN

Normal return is to the next executable statement of the calling program.

RESTRICTIONS

The following restrictions apply to FLHS:

1. A \$ LOWLOAD card (see General Loader manual) must be included if H* file was generated under this option.
2. The memory limits in effect when H* file was generated must be requested.
3. One of the setup subroutines must have been used when H* file was generated. Entry to the main link is made through those subroutines to initialize fault vectors.

Terminal Input Recovery

FUNCTION

FRCV (ASCII) and FRBC (BCD) permit the FORTRAN user to correct a string of characters input from a terminal when a character is illegal for the current format conversion.

CALLING SEQUENCE

CALL .RCOV (curtal,initl)TRA-OK

curtal = current input tally

initl = initial input tally

TRA-OK = recovery transfer point

If no recovery is possible, the return is 0,1

METHOD

When a character is illegal for the current format conversion, the current record (current input line) is output with a pointer to the illegal character. The user can then input a correction or change (several characters) that will replace the corresponding characters previously input. The input/output routine will resume with the new string. If the user responds with a carriage return, the usual execution error monitor message will be output. This routine is not callable by the user. It is called by FDIO when appropriate.

ASCII/BCD Indicators

FUNCTION

FASC is a module that is selected when the ASCII option is used.

Secondary SYMDEF Entry .ASCB = nonzero
.ASCB+1 = ASCII blanks (040040040040)
Other ASCII constants are included.

FBCD is a module that is selected when the BCD option is used.

Secondary SYMDEF Entry .ASCB = 0
.ASCB+1 BCD blanks (202020202020)
Other BCD constants are included.

SUBROUTINES THAT ARE USER CALLABLE

Exponent Register Overflow and Divide Check Tests

FUNCTION

FDCK tests the Fault Status Word for a previous exponent register overflow or divide check.

CALLING SEQUENCE

CALL DVCHK(j) for divide check

CALL OVERFL(j) for exponent register overflow

j = location of integer variable

METHOD

This routine checks the fault vector area and sets an integer variable to 1 if a fault has occurred, or 2 if not. The Fault Status Word (31 octal) is not maintained in the time sharing environment.

RETURN

Normal return is to the next executable statement in the calling program. This routine assumes normal recovery from exponent register overflow and divide check.

Sense Light Simulator

FUNCTION

ELITE simulates the setting and testing of sense lights.

CALLING SEQUENCE

CALL SLITE(0) to turn off all sense lights.

CALL SLITE(i) to turn on sense light i.

CALL SLITEP(i,j) to test and turn off sense light i if on.

0 = integer 0
i = integer variable or constant
j = integer variable to be set to 1 if sense
light i was ON, or 2 if it was OFF.

METHOD

Bits 1-35 of .ELITE correspond to sense lights 1-35 respectively, with 0 denoting OFF and 1 denoting ON. For zero, all sense lights are cleared (turned OFF). When sense light i is i, the sense light specified by integer i (1,2,3,...,35) is turned ON. When sense light i is tested, the integer variable at j will be set to 1 if i was ON, or 2 if i was OFF. Note that .ELITE is a SYMBOL symbol.

RETURN

Normal return is to the next executable statement in the calling program. A possible error condition is error code 21 if i is not 1-35, or if i is 0, in which case error code 21 is error code 21. In case of error, sense light i is ignored if setting, or is declared OFF if testing. sense light i is ignored if setting, or is declared OFF if testing.

Sense Switch Test

FUNCTION

FSWI tests the GCOS switch word for the status of a sense switch.

CALLING SEQUENCE

CALL SSWTCH(i,j) to test sense switch i.

i = integer variable or constant, value between 1 and 6
j = integer variable to be set to 1 if sense
switch i is ON, or 2 if it is OFF

METHOD

Bits 6-11 of the GCOS switch word correspond to the sense switches 1-6, with 0 denoting OFF and 1 denoting ON. Sense switch specified by i is tested and the integer in j is set to 1 if i was ON or to a 2 if i was OFF.

RETURN

Normal return is to the next executable statement in the calling program. A possible error condition is error code 53. If i is not 1-6, sense switch i is declared OFF.

Restore Links During Execution (Batch)

FUNCTION

FLNK enables the programmer to call program overlays in batch mode.

CALLING SEQUENCES

CALL LINK(name)

CALL LLINK(name)

name = link identifier specified as a character variable or character constant

METHOD

FLNK assumes that the General Loader has created a file (file code H*) containing the user's program segmented into links as specified by \$ LINK control cards. (H* file is generated by GESAVE.) Both LINK and LLINK use the GERSTR function of GCOS. The procedure for restoring a link depends on the entry used as follows:

LINK - Restore the link and transfer to its entry point as specified at load time.

LLINK - Restore the link and return to the next statement or instruction in the calling subroutine.

If DEBUG is requested at load time in any or all of the links, these subroutines join the respective DEBUG tables, enabling the user to take snap dumps of any links in memory at the time of his request.

After restoring a link, the link is tested to determine if it contained a DEBUG table. If it contained a DEBUG table, the address of this table is chained to existing tables. If there was no DEBUG table in memory at this time, the address of this table is placed in the DRL cell (cell 13 of the fault vector). DEBUG tables, corresponding to links which are overlaid in the process, are deleted from the chain.

The link overlay names must be five to eight characters for ASCII mode executions.

RETURN

Returns are defined under Method.

Restore Links During Execution (Time Sharing)

FUNCTION

FTLK performs the same functions for time sharing programs as FLNK performs for batch programs. Refer to FORTRAN manual for specific details.

1. The YFORTRAN time sharing subsystem RUN command is used to load the main program link from an H* permanent file for execution.
2. The overlay link names on the H* must be five to eight characters when running in the time sharing ASCII mode.
3. The DEBUG option is not available under time sharing execution.

Execution Error Monitor

FUNCTION

The Execution Error Monitor performs the following functions:

1. Prints a trace of subroutine calls, if applicable.
2. Prints execution error messages.
3. Terminates execution with a Q6 abort or does one of the following:
 - a. Continues with execution of the program.
 - b. Transfers to an alternate error routine.
4. Allows the user to determine if an error has been processed by the Execution Error Monitor.

The Execution Error Monitor functions are performed by the following subroutines:

FXER contains the procedures that are common to both batch and time sharing operations.

FXEM contains the procedures and data peculiar to batch operations.

FXMA contains the procedures and data peculiar to time sharing operations.

Execution error monitor functions are optional and are determined by three groups of switch words (presently there are four words per group) in which each bit corresponds to an error code. The first group (.FXSW1) controls termination; the second (.FXSW2) controls message printing and trace; and the third (.FXSW3) controls the alternate error returns. Table 3-2 contains the error codes and default procedures.

SWITCH WORD GROUPS

1. .FXSW1 (termination). Table 3-2 shows the standard bit settings and the routines that use the corresponding error codes. The bit settings are defined as:

C (Continue execution)

A (Terminate with a Q6 abort)

Termination may be overridden by the corresponding bit of .FXSW3.

2. .FXSW2 (message printing and trace) - The meaning of the bit setting is

1 - Suppress printing

0 - Print

This group is initialized to zero. Settings may be changed by a program call to FXOPT.

3. .FXSW3 (alternate error return) - The meaning of the bit setting is

1 - Alternate error return (overrides termination set in .FXSW1).

0 - Use normal return.

This group is initialized to zero. Settings may be changed by a program call to FXOPT.

GMAP CALLING SEQUENCE

CALL .FXEM. (x,y)

x = address of error description controls

y = address of tally word used to indicate card column found in error; y is optional and is used only if a card image is to be printed

Instruction sequence at x

```
X ZERO  A,B
      ZERO C,D
      ZERO E,F
```

A = Address of card image to be printed, or zero when card image is not to be printed.

B = Error code expressed as an integer(n) in the range 1_n_143.

C = Address of message 1.

D = Word count of message 1.

E = Address of message 2.

F = Word count of message 2.

FORTRAN CALLING SEQUENCES

1. CALL FXOPT(ncode,i1,i2,i3)

FXOPT is an entry to .FXER and may be called to alter the standard switch word settings. In the statement FXOPT(ncode,i1,i2,i3), ncode is an error code, and i1, i2, and i3 provide the settings for the corresponding bits in the three switch word groups.

Examples:

1. CALL FXOPT(32,0,1,0)
2. CALL FXOPT(32,1,0,0)
3. CALL FXOPT(32,0,0,1)

NOTE: Error code 32 denotes an illegal character in input data.

Example 1 causes a Q6 abort when the error occurs, and no message or trace is printed.

Example 2 causes execution to continue after message and trace are printed.

Example 3 indicates that return is to an alternate error routine after trace and message are printed, since the alternate return option takes precedence over termination.

2. CALL FXALT(SR)

FXALT is an entry to .FXER and may be called to set the alternate error return location. The statement CALL FXALT(SR) communicates the name SR of the alternate error routine to the execution error monitor. An EXTERNAL SR must be included in the calling routine. If the alternate return option for an error code is indicated but no call to FXALT has been made, a Q5 abort follows when the error occurs. A RETURN statement in the alternate routine continues execution at the instruction immediately following the one where the error occurred.

The statement CALL FXALT(\$n) designates statement n in the calling program as the alternate error return.

NOTE: If the same error occurs in the alternate error routine, an interminable loop results.

3. Overflow and divide check fault test

The fault processor processes divide check, overflow, exponent overflow, and exponent underflow faults. A message is output on file 06 stating the type of fault and the location at which the fault occurred. Execution continues in the normal manner, although the EAQ registers may have been reset as depicted in the following table.

<u>FAULT</u>	<u>EAQ REGISTERS</u>
Exponent overflow	Large floating-point value ¹
Divide check (FP)	Large floating-point value ¹
Exponent underflow	Floating-point zero
Overflow (Integer)	No change
Divide check (Integer)	No change

To have another value returned in the EAQ registers after a divide check, CALL FXDVCK(r,m) should be executed prior to the occurrence of the fault. This statement causes the value of r to be returned in the EAQ registers after a real divide check and the value of m to be returned in the Q register after an integer divide check. The first argument must be double precision. The second argument may be omitted.

4. CALL FXEM(ncode,msg,n)

A FORTRAN-callable entry has been provided so that it may be called when the program detects an error condition.

This statement causes the printing of an error trace and the Hollerith message contained in the msg array. The number of words (n) to be printed must be within the limits $0 < n \leq 20$. If only the first argument is given, only the trace is printed.

5. CALL ANYERR (v)

The user may desire to detect some error that has occurred in an input/output routine or a mathematical FUNCTION routine.

v is a variable into which .FXER will initially place the value zero. If an error occurs, the error code is placed in v. The logical IF statement provides a suitable means of testing if v is typed LOGICAL.

¹Allows further computations without another immediate fault. This value is set to approximately 10^{**36} .

METHOD

1. Error linkage for tracing calls is generated by the Macro Assembler Program (GMAP) and by the FORTRAN compiler. Tracing stops when the address of the CALL instruction in the error linkage word is zero, or when the number of traces exceeds a constant.

The error trace prints in reverse order. It includes the name of each calling routine, identifying number of the CALL instruction, absolute location of the CALL instruction, and up to five calling arguments.

2. The functions of this routine are optional. The options are controlled by the following switch word groups:

- .FXSW1 - Termination
- .FXSW2 - Message printing
- .FXSW3 - Alternate error returns

Each of the bits (1-143) in a switch word group corresponds to an error code.

3. Special processing applies to error code 55. When this error is encountered, the following message is written:

ILLEGAL VALUE FOR COMPUTED GO TO AT ID NUMBER XXXXX

4. The error code is always stored in the location FXCODE in FXER. Since this is a SYMDEF, it may be accessed by a GMAP program.
5. The error code is also stored indirectly through a pointer defined in FXER. This pointer may be set by calls to ANYERR. If this pointer has been initialized to contain the address of a variable in the user's program via a call to ANYERR, the variable will contain the error code, expressed as an integer, upon return to the calling subprogram, after an error.
6. FXOPT is an entry to .FXER. which, for a given error code, sets the corresponding bits in .FXSW1, .FXSW2, and .FXSW3 to the low-order bit of the second, third, and fourth arguments. The first argument is the error code. When a call is made to .FXER., the error code is used to shift each switch word group and set the options accordingly.
7. FXALT stores the location of its argument in location FXALT1 in .FXER. If the alternate error return option is used, index register 1 and the indicator register are restored; and a transfer is made to FXALT1 indirect. Thus, if the alternate return is a subprogram, the RETURN statement transfers to the location following the call to .FXER. If no alternate has been supplied, a Q5 abort occurs.

8. A divide check, an overflow, or an underflow transfers to .FXER. via the program fault vector. (For a description of the fault vector, see the manual General Comprehensive Operating Supervisor.) .FXER. writes the error message and loads the proper values into the EAQ-registers. The normal return is RET 6 (divide check) or RET 8 (overflow and underflow). If an alternate return is requested, the indicators and index register 1 are loaded from the fault vector, so that a RETURN statement in the alternate routine will transfer to the location immediately following the one that generated the fault.
9. FXEM is the entry provided for error conditions detected by the user's program. Error codes 61-66 are reserved for users. The statement

CALL FXEM(ncode,msg,n)

prints an error trace and n words of the message in the array msg. Msg must be an array containing character information. If either msg or n is omitted or is zero, no message is printed. If n is greater than 20 words, only 20 words are printed.

RETURN

Error codes and returns are defined in Table 3-2.

Table 3-2. Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
0	A		Not used			
1	C	I**J	I=0, J=0	0 → QR	EXPONENTIATION ERROR 0**0	SET RESULT=0
2	C	I**J	I=0, J<0	2 ³⁵ -2 → QR	EXPONENTIATION ERROR 0**(-J)	SET RESULT=2**35-2
3	C	{ DA**J A**J }	DA=0, J=0 A=0, J=0	0 → EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
4	C	{ A**J DA**J }	A=0, J<0 DA=0, J<0	10 ³⁸ → EAQ	EXPONENTIATION ERROR 0**(-J)	SET RESULT=10**38
5	C	B**C	B<0, C=0	0 → EAQ	EXPONENTIATION ERROR (-B)**C	SET RESULT=0
6	C	A**B	A=0, B=0	0 → EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
7	C	A**C	A=0, C<0	10 ³⁸ → EAQ	EXPONENTIATION ERROR 0**(-C)	SET RESULT=10**38
8	C	e**B	B>88.028	10 ³⁸ → EAQ	EXP(B), B GRT THAN 88.028 NOT ALLOWED	SET RESULT=10**38
9	C	LOG (A)	A=0	10 ³⁸ → EAQ	LOG (0) NOT ALLOWED	SET RESULT=10**38
10	C	LOG (B)	B<0	0 → EAQ	LOG (-B) NOT ALLOWED	SET RESULT=0.0
11	C	ARCTAN (A/B)	A=0, B=0	0 → EAQ	ATAN2 (0,0) NOT ALLOWED	SET RESULT=0
12	C	{ SIN (A) COS (A) }	A >2 ²⁷	0 → EAQ	SIN OR COS ARG GRT TH 2**27 NOT ALLOWED	SET RESULT=0
13	C	√B	B<0	√B=√ B	SQRT (-B) NOT ALLOWED	EVALUATE FOR +B
14	C	CA**K	CA=0, K=0	0 → AQ	EXPONENTIATION ERROR 0**0	SET RESULT=0

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
15	C	CA**J	CA=0, J<0	$10^{38} \rightarrow$ AR 0 \rightarrow QR	EXPONENTIATION ERROR 0**(-J)	SET RESULT=(10**38,0.0)
16	C	DA**DB	DA < 0, DB \neq 0	0 \rightarrow EAQ	EXPONENTIATION ERROR (-DA)**DB	SET RESULT=0
17	C	DA**DB	DA=0, DB=0	0 \rightarrow EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
18	C	DA**DB	DA=0, DB<0	$10^{38} \rightarrow$ EAQ	EXPONENTIATION ERROR 0**(-B)	SET RESULT=10**38
19	C	e**DA	DA > 88.028	$10^{38} \rightarrow$ EAQ	EXP(B), B GRT 88.028, NOT ALLOWED	SET RESULT=10**38
20	C	LOG (DA)	DA=0	$-10^{38} \rightarrow$ EAQ	DLOG (0) NOT ALLOWED	SET RESULT=- (10**38)
21	C	LOG (DA)	DA<0	0 \rightarrow EAQ	DLOG (-B) NOT ALLOWED	SET RESULT=0
22	C	\sqrt{DA}	DA<0	$\sqrt{DA}=\sqrt{ DA }$	SQRT (-B) NOT ALLOWED	EVALUATE FOR +B
23	C	$\left\{ \begin{array}{l} \text{SIN DA} \\ \text{COS DA} \end{array} \right\}$	$ DA > 2^{27}$	0 \rightarrow EAQ	DSIN OR DCOS ARG GRT 2**54 NOT ALLOWED	SET RESULT=0
24	C	ARCTAN (DA/DB)	DA=0, DB=0	0 \rightarrow EAQ	DATAN2 (0,0) NOT ALLOWED	SET RESULT=0
25	C	CA/CB	CB=(0,0)	AR= 10^{38} QR= 10^{38}	COMPLEX Z/0 NOT ALLOWED	SET RESULT=(10**38, 10**38)
26	C	e**CA	REAL CA>88.028	$10^{38} \rightarrow$ AR $10^{38} \rightarrow$ QR	EXP(Z), REAL PART GRT 88.028 NOT ALLOWED	SET RESULT=(10**38, 10**38)
27	C	e**CA	$ \text{IMAG CA} > 2^{27}$	0 \rightarrow AR 0 \rightarrow QR	EXP(Z), IMAG PART GRT 2**27 NOT ALLOWED	SET RESULT=0,0
28	C	LOG (CA)	CA=(0,0)	$-10^{38} \rightarrow$ AR 0 \rightarrow QR	CLOG (0) NOT ALLOWED	SET RESULT (- (10**38), 0.0)
29	C	$\left\{ \begin{array}{l} \text{SIN (CA)} \\ \text{COS (CA)} \end{array} \right\}$	$ \text{REAL (CA)} > 2^{27}$	0 \rightarrow AQ	CSIN OR CCOS ARG WITH REAL PART GRT 2**27 NOT ALLOWED	SET RESULT=0
30	C	COS (CA)	IMAG (CA)>88.028	$10^{38} \rightarrow$ AR $10^{38} \rightarrow$ QR	CSIN OR CCOS ARG WITH IM PART GRT 88.018 NOT ALLOWED	SET RESULT=(10**38, 10**38)

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
31	C	BCD I/O	ILLEGAL FORMAT STATEMENT		FORMAT AT LLLLLL, HAS ILLEGAL CHAR	TREAT AS END OF FORMAT
32	C	BCD I/O	ILLEGAL CHARACTER IN DATA OR BAD FORMAT		ILLEGAL CHAR IN DATA OR BAD FORMAT	TREAT ILLEGAL CHAR AS ZERO
33	A	LINKED BINARY I/O	ATTEMPT TO READ OUTPUT FILE		READ AFTER WRITE IS ILLEGAL	FC XX
34	C	BCD I/O	END-OF-FILE		END OF FILE READING FILE CODE FC	OPTIONAL RETURN NOT REQUESTED
35	C	REWIND AND END FILE PROCESSOR	ILLEGAL REQUEST		REQUEST TO XXXXXX ON FC WAS IGNORED	
36	C	FFFB	BACKSPACE ERROR		TAPE POSITIONED AT FIRST FILE	BACKSPACE REQ. LARGER THAN FILE COUNT
37	A	FILE OPENING	FILE NOT DEFINED		FC XX DOES NOT EXIST	
38	A	FILE OPENING	NO SPACE FOR I/O BUFFERS		INSUFFICIENT CORE AVAILABLE FOR BUFFERS	
39	C	LINKED BINARY I/O	ILLEGAL END-OF-FILE		UNEXPECTED EOF OR BAD FORMAT	FC XX
40	C	LINKED BINARY I/O	LIST EXCEEDS LOGICAL RECORD LENGTH		LIST EXCEEDS LOGICAL RECORD LENGTH	STORE ZEROS IN REMAINING LIST
41	A	LINKED BINARY I/O	SYSOUT/FIXED LENGTH RECORDS		SYSOUT/FIXED LENGTH RECORDS	ITEMS MAY NOT BE PARTITIONED-FC XX
42	C	NAMelist INPUT	ILLEGAL HEADING CARD		ILLEGAL HEADING CARD BELOW	SCAN TERMINATED
43	C	NAMelist INPUT	ILLEGAL VARIABLE NAME		ILLEGAL VARIABLE NAME BELOW	SKIPPING TO NEXT VARIABLE NAME
44	C	NAMelist INPUT	ILLEGAL SUBSCRIPT OR ARRAY SIZE EXCEEDED		ILLEGAL SUBSCRIPT BELOW, OR DATA EXCEEDS VARIABLE	SKIPPING TO NEXT VARIABLE NAME

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
45	C	NAMELIST INPUT	ILLEGAL CHARACTER AFTER RIGHT PARENTHESIS		ILLEGAL CHAR IN DATA BELOW	ASSUME COMMA PRECEDES CHAR
46	C	NAMELIST INPUT	ILLEGAL CHAR IN DATA		ILLEGAL CHAR IN DATA BELOW	TREAT CHAR AS ZERO
47	A	BACKSPACE RECORD	FILE CANNOT BE BACKSPACED		FILE CODE XX, BACKSPACE REFUSED	FILE IS SYSOUT OR IS NOT MAG TAPE, D/D
48	C	NAMELIST INPUT	ILLEGAL LOGICAL CONSTANT		ILLEGAL LOGICAL CONSTANT APPEARS BELOW (OR AT END OF PRECEDING RECORD)	TREAT ILLEGAL LOGICAL CONSTANT AS F
49	A	BACKSPACE FILE	ERRONEOUS END-OF-FILE		END-OF-FILE ON READ BUT NOT ON PREVIOUS BACKSPACE OF SAME TAPE RECORD	
50	C	BACKSPACE FILE	BLOCK COUNT OF ZERO		BLOCK COUNT = 0	
51	C	SENSE LIGHT SIMULATOR	INDEX NOT $0 < n < 35$		REFERENCE TO NON-EXISTENT SENSE LIGHT	DECLARED OFF IF TESTING IGNORED IF SETTING
52	C	NAMELIST INPUT	ILLEGAL CHARACTER FIELD		ILLEGAL CHARACTER FIELD BELOW	SKIPPING TO NEXT VARIABLE NAME
53	C	SENSE SWITCH TEST	INDEX NOT $1 < n < 6$		NON-EXISTENT SENSE SWITCH TESTED	SWITCH DECLARED OFF
54	A	FILE OPENING	ATTEMPT TO WRITE I*		ILLEGAL WRITE REQUEST ON SYSIN1	NO OPTIONAL EXIT EXECUTION TERMINATED
55	A	FXEP	ILLEGAL VALUE		ILLEGAL VALUE FOR COMPUTED GO TO AT ID NUMBER XXXXX	XXXXX
56	A	FILE OPENING	ATTEMPT TO READ P*		IT IS ILLEGAL TO READ FROM SYSOUT FC XX	
57	C	BCD I/O	ILLEGAL CHAR FOR L CONVERSION		ILLEGAL CHAR FOR L CONVERSION IN DATA BELOW	TREAT ILLEGAL CHARACTER AS SPACE
58	C	BACKSPACE RECORD			FILE NN IS CLOSED	

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
59	C	NAMELIST INPUT	EMPTY CHARACTER FIELD		EMPTY CHARACTER FIELD	TREAT AS BLANKS
60	C	I**J	J>36,J IS EVEN J>36,J IS ODD,I>0 J>36,J IS ODD,I<0	2 ³⁵ -2→QR 2 ³⁵ -2→QR -(2 ³⁵ -2)→QR	EXPONENT > 35 OR EXPONENTIATION OVERFLOW	SET RESULT = +/- ((2**35)-2)
61 } 62 } 63 } 64 } 65 } 66 }	A	RESERVED FOR USERS				
67	C	FAULT	EXPONENT UNDERFLOW		EXPONENT UNDERFLOW	AT LOCATION XXXXXX
68	C	FAULT	INTEGER OVERFLOW		OVERFLOW	AT LOCATION XXXXXX
69	C	FAULT	EXPONENT OVERFLOW		EXPONENT OVERFLOW	AT LOCATION XXXXXX
70	C	FAULT	INTEGER DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX
71	C	FAULT	FLOATING POINT DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX
72	C	RANDOM BINARY I/O	LIST EXCEEDS LOGICAL RECORD LENGTH		LIST EXCEEDS LOGICAL RECORD LENGTH	STORE ZEROS IN REMAINING LIST ITEMS FC XX
73	A	RANDOM BINARY I/O	FILE NOT STANDARD SYSTEM FORMAT. ZERO BLOCK COUNT; BSN ERROR; ZERO RECORD COUNT		FILE NOT STANDARD SYSTEM FORMAT FC XX	
74	A	RANDOM BINARY I/O	NO FCB FOR FILE		LOGICAL FILE CODE XX DOES NOT EXIST	NO OPTIONAL EXIT EXECUTION TERMINATED
75	A	RANDOM BINARY I/O	BAD RANDOM RECORD REFERENCE		ZERO OR NEGATIVE RANDOM REC #	FC XX
76	A	RANDOM BINARY I/O	RECORD SIZE NOT SPECIFIED IN FCB. GIVE VIA \$ FFILE CARD OR CALL RANSIZ (FC,SIZE)		REC SIZE NOT GIVEN FOR RANDOM FILE	FC XX

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
77	A	RANDOM BINARY I/O	RANDOM WRITE TO LINKED FILE ILLEGAL. LINKED FILE MAY BE READ RANDOMLY BUT NOT WRITTEN TO.		RANDOM WRITE TO LINKED FILE ILLEGAL	FC XX
78	A	RANDOM BINARY I/O	THE RECORD NO. GIVEN IN THE RANDOM READ OR WRITE STATEMENT IS OUTSIDE THE FILE LIMITS.		REC # OUT-OF-BOUNDS-	FC XX
79	A	RANDOM BINARY I/O	LIST EXCEEDS DECLARED RECORD LENGTH.		LIST EXCEEDS DECLARED RECORD LENGTH	FC XX
80	A	RANDOM BINARY I/O	FILE IS NOT LARGE ENOUGH TO CONTAIN RECORD		FILE SPACE EXHAUSTED-	FC XX
81	C	FORMAT I/O ENCODE/DECODE	LINE EXCEEDS SIZE OF RECEIVING FIELD		LINE EXCEEDS SIZE OF RECEIVING FIELD	TREAT AS END OF FORMAT
82	A	FORMAT I/O ENCODE/DECODE	FIRST NON-BLANK CHARACTER IS NOT (FIRST NON-BLANK CHARACTER IS NOT (TREAT AS END OF FORMAT
83	C	ARCSINE	ARG > 1.0		ARG > 1.0	EVALUATE FOR ARG=1.0
84	C	FORMAT I/O ENCODE/DECODE	INTEGER > 2**35-1		INTEGER > 2**35-1	LIMIT TO 2**35-1
85	C	I/O	"GFRC" ERROR		"GFRC" ERROR	FC XX
86	A	FORMAT I/O ENCODE/DECODE	ENCODE/DECODE-I/O MAY NOT BE USED RECURSIVELY		ENCODE/DECODE-I/O MAY	NOT BE USED RECURSIVELY
87	C	I/O	SPACE/CORE OBTAINED		SPACE/CORE OBTAINED FOR	LOG. FILE CODE XX

Table 3-2 (cont). Error Codes And Returns

<u>ERROR CODE</u>	<u>DEFAULT PROCEDURE ABORT/ CONTINUE</u>	<u>FUNCTION</u>	<u>ERROR</u>	<u>EXCEPTION RETURN</u>	<u>MESSAGE LINE 1</u>	<u>MESSAGE LINE 2</u>
88	C	CALLSS	END OF STRING CHARACTER MISSING			
89	C	EXP DEXP	UNDERFLOW		EXP (TOO LARGE A NEGATIVE NUMBER)	SET RESULT =0.0
90	C	TAN DTAN	ARG TOO LARGE		LARGE ARG(71E4) TO TAN	MAY CAUSE LOSS OF PRECISION
91	C	ACOSH DACOSH	ILLEGAL ARG		ACOSH OF NUMBER .LT. 1.0 NOT ALLOWED	SET RESULT TO 0.0
92	C	ATANH	ILLEGAL ARG		X .GE. 1.0 TO ATANH(X)	SET RESULT TO + CR -10**38
93-99		NOT PRESENTLY USED				

NOTATION: I,J,K are integers
 A,B,C are real numbers
 DA,DB,DC are double-precision numbers
 CA,CB,CC where CA=X,Y are complex numbers

File Transliteration

FUNCTION

FMED is a file media transliteration module.

CALLING SEQUENCE

CALL FMEDIA (fc,media)

fc = FORTRAN logical file code

media = pointer to a code representing the form of the output as follows:

media = 0, BCD NSLEW
= 2, BCD cards
= 3, Printer
= 5, Time Sharing ASCII (obsolete)
= 6, Standard System Format ASCII (no slew)
All others are ignored

The legal combinations are as follows:

0 to 2	3 to 0
0 to 3	3 to 2
0 to 5	3 to 5
0 to 6	3 to 6
2 to 0	6 to 0
2 to 3	6 to 2
2 to 5	6 to 3
2 to 6	6 to 5

METHOD

FMED sets bits 18-21 of the file control block LOCSYM+5. These bits are used to inform the I/O edit function that each output record directed to the file (fc) must be transliterated to the format represented by the media in bits 18-21 of FCB+5. FMED calls FOPEN to locate the file control block. FOPEN returns the file control block location. FMED calls GMEDIA. If media = 0, the nslew bit in the file control block is also set.

Automatic file transliteration is provided and/or reformatting on a logical record basis permits the following:

1. Executing of a BCD program under time sharing.
 - a. I/O can be directed to the terminal.
 - b. Input files can be ASCII (media 5 or 6).
 - c. Output files can be media 0,2,3 BCD or 5,6 ASCII.

2. Execution of an ASCII program in batch.
 - a. I/O can be directed to reader, printer, punch or SYSOUT.
 - b. Input files can be media 0,2,3 BCD or 5 ASCII.
 - c. Output files can be media 0,2,3 BCD or 6 ASCII.
3. Execution of a BCD program in batch.
 - a. Input files can be ASCII (either media 5 or 6).
 - b. Output files can be media 0,2,3 BCD or 6 ASCII.
4. Execution of an ASCII program under time sharing.
 - a. Terminal I/O is provided.
 - b. Input files can be media 5 ASCII or 0,2,3 BCD.
 - c. Output files can be media 0,2,3 BCD or 5,6 ASCII.

Date and Time

FUNCTION

FDTM allows a user to obtain the current date and time.

CALLING SEQUENCE

```
CHARACTER A*8  
REAL B  
.  
.  
CALL DATIM (A,B)
```

Upon return, A will contain the date in the form mm/dd/yy (with trailing blanks if in BCD mode); B will contain the time-of-day in hours as a floating point binary number.

METHOD

FDTM uses the MME GETIME in the batch mode and the DRL TIME in the time sharing mode. FDTM uses the GCVT module for transliteration on the date, if required.

Cell .SETU.-6 is used to determine if the environment is batch or time sharing.

Access a Permanent File

FUNCTION

FTAC is used to access an existing permanent file.

CALLING SEQUENCE

CALL ATTACH (lgu,catfil,iprmis,mode,istat,buffer)

lgu = the FORTRAN file code (an integer expression, variable, or constant)
catfil = a character constant, or variable, containing the catalog/file string. It must be terminated by a semicolon; embedded blanks are ignored. The user master catalog password is used if it exists; the system master catalog password is never used; however, subsequent passwords are required if they are part of the file description.
iprmis = the permissions desired. These will be ORed with any permissions in the catfil.
If iprmis = 1, READ only; if iprmis = 2, WRITE only; if iprmis =3, READ and WRITE; otherwise, undefined and may change.

mode = an integer variable or constant
mode = 0; Gets file as defined
mode = 1; Gets file as random
mode = 2; Gets terminal
istat = the status return from the file system (see the TSS System Programmer's Reference Manual for TSS codes), or will contain a status as follows:

0 = OK (batch mode only)
1 = File is currently open
2 = Terminal requested in batch mode (illegal)
3 = Additional memory needed, request denied (time sharing user is aborted)
4 = catfil all blanks

buffer = null arg: get a file system buffer.
= not null arg: use this variable array as a buffer (at least 380 words).

Example of null arg:

```
CALL ATTACH (lgu,"catfil;",iprmis,mode,istat, )
```

METHOD

Upon successful return from FTAC (attach), a file control block will have been created and the file name (or alternate name) will be in FCB -10, -9 (in ASCII). If the file was in the available file table (AFT), it will be deaccessed and reaccessed with the new permissions, if necessary.

Close File, Detach Buffers, Remove from APT

FUNCTION

FDTH is used to close a file and release its buffers. In time sharing, the file is also removed from the APT. If more memory is needed (to deaccess the file) and the request is denied, the time sharing user is aborted.

CALLING SEQUENCE

CALL DETACH (lgu,istat,buffer)

lgu = the FORTRAN code (an integer expression, variably, or constant)

istat = the status return word

istat = 0, means OK

istat = 1, means could not get file system buffer

buffer = null arg: get a file system buffer

= not null: use this buffer (at least 380 words)

The following is an example of a null argument.

CALL DETACH (lgu,istat,)

METHOD

See Function.

Attach a Temporary Mass Storage or Terminal File

FUNCTION

FCRA is used to create and access a temporary mass storage or terminal file.

CALLING SEQUENCE:

CALL CREATE (lgu, isize, mode, istat)

lgu = the FORTRAN file code (an integer expression, variable, or constant)
isize = the size, in words, of the temporary file wanted
mode = 0 for a linked mass storage file
= 1 for a random mass storage file
= 2 for a terminal file
istat = the status return word (see the TSS System Programmer's Reference Manual for TSS codes). The following codes also apply:

- = 0, successful
- = 1, mode is invalid
- = 2, file is currently open
- = 3, no room in APT
- = 4, temporary file not available
- = 5, duplicate file name
- = 6, no room in PAT
- = 7, illegal device specified

METHOD

If the function is successful, a FCB is created and the file code, in ASCII, is placed in FCB -10, -9.

Core Allocator

FUNCTION

FALC is a general dynamic storage allocator capable of dispensing memory space from a region composed of a list of disjoint free blocks.

CALLING SEQUENCE

The following call will allocate a block of n consecutive words:

```
LDQ  n,DL  
TSX1 .FDSGT
```

If the request is denied, the Q register will be returned with the value zero. If the request is granted, bits 0-17 of the Q register will contain the address of the allocated block and bits 18-35 will contain the length of the block as designated in the call. (Currently, denial returns are given only in the batch mode of operation.)

METHOD

The free blocks are chained together with a linked list running through bits 18-35 of their initial words. The last block on the chain contains zero in this field. Bits 0-17 of the first word of a free block contain the address of the last word in the block plus one. Equality of these two fields indicates that the block is immediately followed by another free block and the two may be coalesced into one.

To locate memory space to satisfy a storage request, the chain of free blocks is first searched, and blocks are coalesced as the search proceeds. If a sufficiently large free block is found, the requested space is allocated and any excess memory is returned to the free block chain.

If no free block is found that satisfies the request, space is sought in the "core-hole" described by word 37 (octal) of the slave prefix. If enough memory space is available at this location, the requested space is allocated out of the low end and the core-hole limits are adjusted accordingly.

If no memory space is available in the core-hole, a request is made to the operating system for additional memory. The memory space obtained in this operation is transformed into a new core-hole and the old core-hole is merged into it, if possible.

To ensure that the core-hole does not become too small, any free blocks adjacent to the core-hole are merged with the core-hole and are removed from the free block chain.

Special Entry Point

FUNCTION

FDBD provides a special entry point to a dummy routine. If a user-generated version of the FDEBUG module has been provided, control will be passed to the dummy routine rather than to the FDEBUG bootstrap routine.

FDEBUG Bootstrap

FUNCTION

FDEB is the bootstrap subroutine for the FDEBUG module.

CALLING SEQUENCE

CALL FDEBUG

METHOD

The FDEBUG overlay is loaded into memory (if not already loaded) and control is transferred to the entry point.

FDUMP Bootstrap

FUNCTION

FDUM is the bootstrap subroutine for the symbolic dump facility FDUMP.

CALLING SEQUENCE

CALL FDUMP

METHOD

The FDUMP overlay is loaded into memory (if not already loaded) and control is transferred to the entry point.

LINK/LLINK Interface

FUNCTION

FLKL provides an interface between LINK/LLINK and the FDEBUG module.

CALLING SEQUENCE

TSX1 .FDLNK

It is assumed that the A register contains the link origin in bits 0-17 and the link size in bits 18-35.

The entry point .FDLLD has the calling sequence

CALL .FDLLD (name)

where: 'name' is the address of the link name in character form.

METHOD

Control is first passed to FDEBUG following the entry to .FDLNK so that any overlaid breakpoints can be removed from the breakpoint table. Control is then passed to .FDBDL, a secondary entry point in the FDEBUG bootstrap subroutine.

The subroutine .FDLLD is an elaborate NOP that is capable of being breakpointed. The user can therefore gain control after a link has been loaded to install new breakpoints.

Delete From Wrapup List

FUNCTION

FNCL allows the user to delete subroutines from the wrapup list.

CALLING SEQUENCE

CALL NOCALL(subr)

METHOD

When a CALL NOCALL(subr) statement is included, all occurrences of the subroutine are deleted from the wrapup list.

Release Unused Memory

FUNCTION

FREL provides the capability to return a block of memory space to the free block list.

CALLING SEQUENCE

To free a block of memory, the address of the block is placed in bits 0-17 of the Q register and the length of the block is placed in bits 18-35 (this is the data returned by .FDSGT). The following call is made:

```
TSX1 .FDSRL
```

The following call will release as much memory as possible from the high end of core:

```
TSX1 .FDSRM
```

METHOD

Returning a block of memory space to the free list is a simple matter of finding the appropriate location in the chain and then adjusting the pointers. Blocks are not merged with the core-hole during this search.

Dummy Setup

FUNCTION

FSTP is a dummy subroutine that prevents binding of FDS modules when the FDS option has not been invoked.

CALLING SEQUENCE

TSX0 .FDSET

METHOD

If called, this subroutine simply returns via index register zero. Secondary SYMDEFs are provided for FDEBUG and FDUMP that allow calls to FDEBUG and FDUMP to return to the user unless the FDS option has been invoked.

Add to Wrapup List

FUNCTION

FTCL allows the user to add subroutines to the wrapup list that is maintained dynamically by the FORTRAN debugging system.

CALLING SEQUENCES

CALL ATCALL(subr)
CALL NTCALL(subr)

METHOD

The wrapup list is inspected whenever a program terminates abnormally or terminates when a STOP statement is executed. However, when a program terminates upon the execution of a CALL FTERM statement, the wrapup list is not inspected, thus providing a mechanism to avoid calling procedures on the wrapup list.

The wrapup list is maintained in a first-in/first-out basis so that the subroutines entered first will be called first. If the entry is made using ATCALL, the subroutine will be called when the program terminates abnormally. If the entry is made using NTCALL, the subroutine will be called when the program terminates normally.

Timing Facility

FUNCTION

FTMR provides the subprogram timing measurement system of FDS, called FTIMER.

CALLING SEQUENCE

CALL FTIMER

NOTE: A \$ USE FTIMER control card may be included in the batch mode to force the timing measurement system to be loaded.

METHOD

FTMR contains replacements for the entry points .ENTY, .RETY, and .FRETY that are contained in the linking subroutine FYLK. These replacements dynamically allocate data blocks in which timing measurement statistics are accumulated as the program executes. These statistics are summarized and printed when the program terminates (even if the termination is abnormal). The measurements are given only for those subprograms compiled with the FDS option.

Wrapup and Loader

FUNCTION

FWRP provides the standard FORTRAN wrapup procedure as well as the overlay loader used by the FDEBUG and FDUMP bootstrap subroutines.

CALLING SEQUENCE

TSX1 .FEXIT (Termination)
TSX1 .FDSL (Overlay loader)

METHOD

FWRP contains the standard FORTRAN wrapup procedure .FEXIT. When this procedure is called, it first determines whether the termination is normal or abnormal and then invokes each user-supplied wrapup routine as required. When all of the subroutines in the wrapup list have been called, the subroutine .FTERM is then invoked to provide standard termination.

FWRP contains the overlay loader. When it is called, the A register contains zero (0) if an FDUMP overlay is to be loaded or two (2) if an FDEBUG overlay is to be loaded.

Linking Subroutine

FUNCTION

FYLK contains the subroutines .ENTY, .RETY, and .FRETY, as well as the setup procedure.

CALLING SEQUENCE

```
TSX0 .ENTY (or .RETY or .FRETY)
NOP  .E.L..
```

METHOD

Subroutine .ENTY is called whenever a FORTRAN subprogram is entered. Its function is to save the indicator register and index register 1 in the error linkage area and to insert the address of the error linkage area into bits 0-17 of .CE.L.

Subroutine .RETY is called whenever a RETURN statement is executed. Its function is to restore the indicator register and to set bits 0-17 of .CE.L. to the address of the caller's error linkage area.

Subroutine .FRETY is called whenever a RETURN n statement is executed. Its function is to restore the indicator register, to set bits 0-17 of .CE.L. to the address of the caller's error linkage area, and to execute the return.

The setup procedure is called by the fault processor (.FLTPR) via a TSX0 .FDSET. Its primary function is to call FDEBUG when required prior to entering the main program. In the batch mode of operation, tests are performed to verify whether file designator 44 is present; if it is, the FDEBUG module is called. In the time sharing mode, bit 4 of the program switch word is checked to determine whether or not the FDEBUG module should be called.

SECTION IV

MATHEMATICAL LIBRARY SUBROUTINES

The subroutines that make up the Mathematical Library offer the FORTRAN user a wide range of options to help solve complex mathematical problems. These subroutines provide increased flexibility and computational capability to the FORTRAN scientific programming language.

The purpose of the FORTRAN mathematical library subroutines is to provide a fast method of calculating the basic mathematical functions. The purpose of this update is to improve the accuracy and performance of the present subroutines and to add several new functions. The results obtained with this improved version of the library will in many cases differ from those obtained when using earlier versions; in these cases, answers obtained are more accurate. The following statements relate to the degree of accuracy for both single- and double-precision floating-point binary numbers.

The single-precision subroutines were written to provide accuracy to full single precision plus or minus one bit, whenever possible. This gives a relative error of less than 1×2^{-27} .

The double-precision subroutines use polynomials with theoretical accuracy beyond that of the floating-point hardware (greater than 18 decimal digits). Any error that exists is due to rounding or truncation. Programming techniques were used to minimize this type of error within the routine with the result that most routines are accurate to about 17 or more decimal digits.

It is assumed that the binary representation of the argument (in single or double precision) is the exact value, and the ensuing calculations and results are based on that assumption. Rounding or conversion error prior to entering the subroutine cannot be anticipated. However, since argument error can cause serious problems in calculations, the discussion of each new or revised subroutine contains a brief summary of the possible effects of argument error on the results. This information is presented in terms of a formula that usually gives only the first term in the series expansion for the error term.

The following notation is used in the error formulas:

ACT - Correct result
AEA - Absolute error of the argument
AER - Absolute error of the result
REA - Relative error of the argument
RER - Relative error of the result
RES - Result given by the subprogram

Overflow and underflow faults are not masked during the execution of the subroutines. In cases where the result cannot be represented as a floating-point number because it is too small or too large, a call is made to an error routine and an appropriate error message is generated.

In certain cases, underflow may occur for valid input arguments but this condition will not affect the accuracy of the result. Intermediate overflow should not occur during the execution of the subroutines.

MATHEMATICAL LIBRARY DESCRIPTIONS

The FORTRAN mathematical library subroutines are composed of the function subprograms and are summarized in Table 4-1. The definition, calling name, and edit name are given for each library subroutine. The page numbers are included since the indexed page numbers for Section IV are no longer valid for this addendum.

Table 4-1. Mathematical Library Subroutines

Subprogram Function	Definition	Calling Name	Edit Name	Page No.
• • Absolute Value, Complex	$ a $	CABS	FCAB	4-23
Arithmetic, Complex, Multiply and Divide	$a*b$ a/b	Implied	FCMP	4-26
• • Arccosine, Real	$\cos^{-1}(a)$	ARCOS	FASN	4-18
• Arccosine, Double		DARCOS	FDAS	4-35
• Arccosine, Hyperbolic, Real	$\cosh^{-1}(a)$	ACOSH	FASH	4-16
• Arccosine, Hyperbolic, Double		DACOSH	FDAH	4-34
• • Arcsine, Real	$\sin^{-1}(a)$	ARSIN	FASN	4-18
• Arcsine, Double		DARSIN	FDAS	4-35
• Arcsine, Hyperbolic, Real	$\sinh^{-1}(a)$	ASINH	FASH	4-16
• Arcsine, Hyperbolic, Double		DASINH	FDAH	4-34
• • Arctangent, Real	$\tan^{-1}(a)$	ATAN	FATN	4-21
• • Arctangent, Double		DATAN	FDAT	4-37
• • Arctangent 2, Real	$\tan^{-1}(a/b)$	ATAN2	FATN	4-21
• • Arctangent 2, Double		DATAN2	FDAT	4-37
• Arctangent, Hyperbolic, Real	$\tanh^{-1}(a)$	ATANH	FASH	4-16
• Arctangent, Hyperbolic, Double		DATANH	FDAH	4-34
• • Cosine, Real	$\cos(a)$	COS	FSIN	4-60
• • Cosine, Double		DCOS	FDSN	4-47
• Cosine, Complex		CCOS	FCSN	4-30
• Cosine, Hyperbolic, Real	$\cosh(a)$	COSH	FTNH	4-66
• Cosine, Hyperbolic, Double		DCOSH	FDPH	4-45
• New Subroutine				
• • Revised Subroutine				

Table 4-1 (cont). Mathematical Library Subroutines

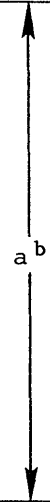
Subprogram Function	Definition	Calling Name	Edit Name	Page No.
<ul style="list-style-type: none"> • Cube Root, Real • Cube Root, Double 	$(a)^{1/3}$	CBRT DCBRT	FCRT FDCR	4-28 4-39
<ul style="list-style-type: none"> • • Exponential, Real • • Exponential, Double Exponential, Complex 	e^a	EXP DEXP CEXP	FEXP FDEX FCEX	4-58 4-40 4-24
<ul style="list-style-type: none"> • • Exponential 2, Real • • Exponential 2, Double 	2^a	EXP2 DEXP2	FEXP FDEX	4-58 4-40
<ul style="list-style-type: none"> • • Exponential 10, Real • • Exponential 10, Double 	10^a	EXP10 DEXP10	FEXP FDEX	4-58 4-40
<ul style="list-style-type: none"> • Exponential Complement, Real • Exponential Complement, Double 	$e^a - 1.0$	EXPC DEXPC	FEXC FDXC	4-56 4-52
<ul style="list-style-type: none"> • Exponential Complement 2, Real • Exponential Complement 2, Double 	$2^a - 1.0$	EXPC2 DEXPC2	FEXC FDXC	4-56 4-52
<ul style="list-style-type: none"> • Exponential Complement 10, Real • Exponential Complement 10, Double 	$10^a - 1.0$	EXPC10 DXPC10	FEXC FDXC	4-56 4-52
<ul style="list-style-type: none"> Exponentiation, Complex Base, Any Exponent Exponentiation, Complex Base, Integer Exponent • • Exponentiation, Double Precision, Base or Exponent (or Both) Exponentiation, Integer Base and Exponent • • Exponentiation, Real Base and Exponent Exponentiation, Real Base, Integer Exponent 		Implied Implied Implied Implied Implied Implied	F4XP FCXP FDXP FLXP F3XP F2XP	4-13 4-33 4-54 4-8 4-11 4-10
<ul style="list-style-type: none"> • New Subroutine • • Revised Subroutine 				

Table 4-1 (cont). Mathematical Library Subroutines

Subprogram Function	Definition	Calling Name	Edit Name	Page No.
<ul style="list-style-type: none"> • • Logarithm, Natural, Real • • Logarithm, Natural, Double • • Logarithm, Natural, Complex 	$\log_e (a)$	ALOG DLOG CLOG	FALG FDLG FCLG	4-14 4-42 4-25
<ul style="list-style-type: none"> • • Logarithm, Base 2, Real • • Logarithm, Base 2, Double 	$\log_2 (a)$	ALOG2 DLOG2	FALG FDLG	4-14 4-42
<ul style="list-style-type: none"> • • Logarithm, Common, Real • • Logarithm, Common, Double 	$\log_{10} (a)$	ALOG10 DLOG10	FALG FDLG	4-14 4-42
<ul style="list-style-type: none"> • • Power, Real (Exponentiation) • • Power, Double (Exponentiation) 	a^b	POW DPOW	F3XP FDXP	4-11 4-54
Remaindering, Double	$a_1 \pmod{a_2}$	DMOD	FDMD	4-44
<ul style="list-style-type: none"> • • Sine, Real • • Sine, Double • • Sine, Complex 	$\sin (a)$	SIN DSIN CSIN	FSIN FDSN FCSN	4-60 4-47 4-30
<ul style="list-style-type: none"> • Sine, Hyperbolic, Real • Sine, Hyperbolic, Double 	$\sinh (a)$	SINH DSINH	FTNH FDPH	4-66 4-45
<ul style="list-style-type: none"> • • Square Root, Real • • Square Root, Double • • Square Root, Complex 	$(a)^{1/2}$	SQRT DSQRT CSQRT	FSQR FDSQ FCSQ	4-62 4-49 4-32
<ul style="list-style-type: none"> • Tangent, Real • Tangent, Double 	$\tan (a)$	TAN DTAN	FTAN FDTN	4-64 4-50
<ul style="list-style-type: none"> • • Tangent, Hyperbolic, Real • Tangent, Hyperbolic, Double 	$\tanh (a)$	TANH DTANH	FTNH FDPH	4-66 4-45
<ul style="list-style-type: none"> • New Subroutine • • Revised Subroutine 				

DEFINITIONS AND CONSIDERATIONS

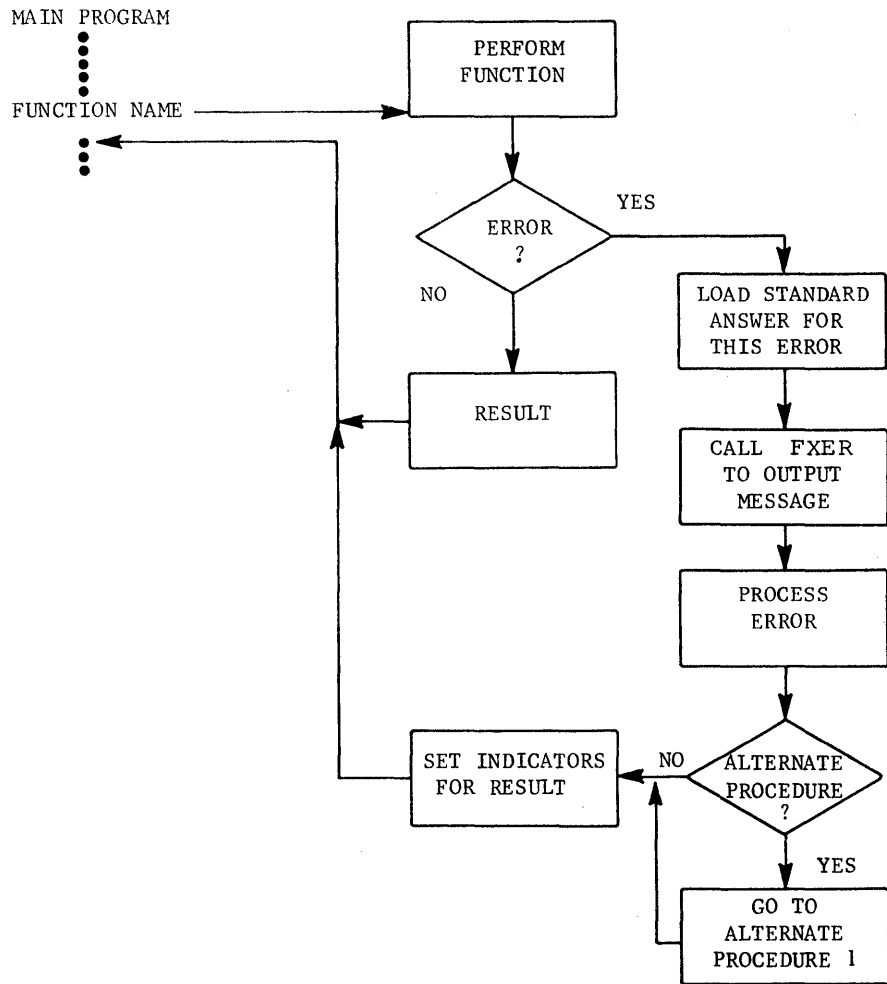
1. The following descriptions show both the GMAP calling sequence and an example of user coding. The subroutines that do not show user coding are generally called by other subroutines that make use of the called subroutines.
2. The subroutines in the math library are function subprograms and are not called by a CALL statement by the user. Some are called by the FUNCTION name (i.e., A=SQRT(X)) while others are called implicitly. For example, subprogram FLXP is invoked via an arithmetic statement of the following type: K=I**J.
3. The answer is always returned in processor registers. The FORTRAN compiler generates the appropriate store instructions (if a replacement operation is involved). The registers used are determined by the type of the function and are as follows:

<u>Answer</u>	<u>Register</u>	<u>Processor Instruction</u>
Real	EAQ	FSTR
Double Precision	EAQ	DFSTR
Integer, Logical	Q	STQ
Complex	AQ	STAQ

If the answer is to be stored into a variable of a different type, the compiler will supply a suitable conversion algorithm. See the FORTRAN reference manual for the various combinations and normal type of results.

4. When the letter Z appears in a function description, it indicates a complex variable, unless otherwise noted.

5. Errors are detected by many of the subprograms. The functional flow when an error is detected is as follows:



6. Where the argument types are listed for a subprogram, the subprogram assumes that the arguments are of the type listed. No explicit error messages are issued if incorrect arguments are supplied.

1

It is assumed here that the alternate procedure is a FUNCTION subprogram which ends with a RETURN statement. This is not necessary; the alternate procedure could be a section of the main program (or subprogram) in which case the flow would not include the "SET INDICATORS---".

Exponentiation - Integer Base and Exponent

FUNCTION

FLXP computes $I^{**}J$ in an expression.

USAGE

$K=I^{**}J$ I, J are integers.

GMAP CALLING SEQUENCE

CALL .FXP1(I,J)

METHOD

1. For $I=0, J=0$, then $K=0$, error code 1
2. For $I=0, J<0$, then $K=(2^{**}35)-2$, error code 2
3. For $I=0, J>0$, then $K=0$
4. For $I\neq 0, J=0$, then $K=1$
5. For $I=1, J\neq 0$, then $K=1$
6. For $I=-1, J$ is even, $K=1$
7. For $I=-1, J$ is odd, $K=-1$
8. For $|I|\geq 2, J<0, K=0$
9. For $|I|\geq 2, J<36, K=I^{**}J$
10. For $|I|\geq 2, J\geq 36$
11. If $|K|>(2^{**}35)-1$, error code 60
If J is even, $K=(2^{**}35)-2$
If J is odd, I is negative, $K=-(2^{**}35)-2$
If J is odd, I is positive, $K=(2^{**}35)-2$

RETURNS

Normal return is to the next executable statement in the calling program, when AQ(0-71) contains K. Normally only the last 36 bits of Q (36-71) are used.

1. Error code 1 if $I=0$, $J=0$, then $K=0$. Execution continues.
2. Error code 2 if $I=0$, $J<0$, then $K=(2^{35})-2$. Execution continues.
3. Error code 60 if $K=I**J$ and $|K| > (2^{35})-1$. Execution continues.

Exponentiation - Real Base, Integer Exponent

FUNCTION

F2XP computes $A^{**}K$ in an expression.

USAGE

$B = A^{**}K$ A is real or double precision, K is an integer.

GMAP CALLING SEQUENCE

CALL .FXP2(A,K) for real A

CALL .FDXP1(A,K) for double precision A

METHOD

1. For $A=0$, $K=0$, then $B=0$, error code 3
2. For $A=0$, $K>0$, then $B=0$
3. For $A=0$, $K<0$, then $B=10^{**}38$, error code 4
4. For $A\neq 0$, $K=0$, then $B=1.0$
5. For negative values of K , change sign of K and take the reciprocal of the result.
6. K is an integer with values from $-2^{**}35$ to $(2^{**}35)-1$.
 A and $A^{**}K$ are floating-point numbers with values from $-2^{**}127$ to $(2^{**}127)-2^{**}64$.
7. $A^{**}K$ is accurate to eight decimal positions for FXP2 or 16 decimal positions for FDXP1.

RETURNS

Normal return is to the next executable statement of the calling program.

1. Error code 3 if $A=0$ and $K=0$. Then $A^{**}K=0.0$. Execution continues.
2. Error code 4 if $A=0$ and $K<0$. Then $A^{**}K=10^{**}38$. Execution continues.

Exponentiation - Real Base and Exponent

FUNCTION

F3XP computes $X^{**}Y$ in an expression.

USAGE

$C=X^{**}Y$ X and Y are real.

GMAP CALLING SEQUENCE

CALL .FXP3(X,Y)

CALL POW(X,Y)

METHOD

1. The subroutine is based on the formula
$$X^{**}Y = 2^{**}(Y*ALOG2(X)).$$
2. The subroutine uses the entry point .ALOG2 into the ALOG2 subroutine and the entry point .EXP2 into the EXP2 subroutine. (EXP2(Z) finds the value of $2^{**}Z$.)
3. The largest known relative error is less than $1*2^{**-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

The error propagation can be calculated from the guidelines provided in the ALOG2 and EXP2 subroutines.

RETURNS

Normal return is to the next executable statement in the calling program, except as specified below:

1. If (X.LT.0.0), the result is set to 0.0 and FXEM is called with error code 5.
2. If (X.EQ.0.0 .AND. Y.EQ.0.0), the result is set to the largest possible number and FXEM is called with error code 7.

3. Appropriate overflow and underflow errors may occur in EXP2 (edit name FEXP) if the results cannot be expressed as a single-precision number.

Exponentiation - Complex Base, Any Exponent

FUNCTION

F4XP computes $A^{**}B$ in an expression.

USAGE

$C=A^{**}B$ A and B are complex.
 $C=A^{**}B$ A is complex; B is real or double precision.

GMAP CALLING SEQUENCE

CALL .FCXP2(A,B) for complex base and exponent
CALL .FCXP3(A,B) for complex base and real or double-precision exponent

METHOD

(CA, CB are the arguments)

CALL	CLOG(CA)	Complex logarithm of complex A
STAQ	AA	CLOG of argument 1
CALL	.FCFMP(AA,CB)	Product
STAQ	AA	
CALL	CEXP(AA)	Complex exponential

For complex base and real or double-precision exponent, a complex number is formed with the real part equal to the argument and the imaginary part equal to zero. Otherwise, it uses the same method as complex base and exponent.

Accuracy: 7 places $\pm 1.0 \cdot 10^{*-7}$

RETURNS

Normal return is to the next executable statement in the calling program.

Natural, Base 2, and Common Logarithms

FUNCTION

FALG computes $\log_e(a)$ for ALOG(X), where $X.GT. 0.0$; $\log_2(a)$ for ALOG2(X), the logarithm base 2; and $\log_{10}(a)$ for ALOG10(X), the common logarithm or logarithm base 10.

USAGE

A=ALOG(X)
 A=ALOG2(X) X is real.
 A=ALOG10(X)

GMAP CALLING SEQUENCE

CALL ALOG(X) for $\log_e(a)$
 CALL ALOG2(X) for $\log_2(a)$
 CALL ALOG10(X) for $\log_{10}(a)$

METHOD

1. The basic function calculated by the subroutine is ALOG2(X). The results for ALOG and ALOG10 are obtained by using the expression $ALOG(X) = ALOG2(X)*ALOG(2.0)$ or $ALOG10(X) = ALOG2(X)*ALOG10(2.0)$.
2. The constants ALOG(2.0) and ALOG10(2.0) are stored in the subroutine; the necessary multiplication is performed just prior to the return. No multiplication is performed for ALOG2.
3. The polynomial approximation for ALOG2 is accurate over the range (.707, 1.414) and is of the form

$$P00+Z**2*(P01+Z**2*(P02+Z**2*P03))$$

where: $Z = (RRX-1.)/(RRX+1.)$.

The range reduction of the argument utilizes the fact that:

$$ALOG2(RRX*2**IEXP) = ALOG2(RRX) + IEXP.$$

4. The following steps are performed:
 - a. If X is already within the range (.707, 1.414), omit the remaining steps and use the polynomial directly. This prevents loss of precision for arguments near 1.0.
 - b. EXP = exponent of X.
 - c. RRX = (X with exponent set to 0), so RRX is within the range (.5, 1.0).
 - d. Since the polynomial is accurate over the range (.707, 1.414), multiply RRX by the SQRT(2) and compensate for the multiplication by subtracting 0.5 (which is ALOG2(SQRT(2))) from the final answer. The actual multiplication is not performed; instead, the quotient $(X-1/\text{SQRT}(2))/(X+1/\text{SQRT}(2))$ is formed in finding Z for use in the polynomial. Note that $(\text{SQRT}(2)*X-1)/(\text{SQRT}(2)*X+1)$ is equivalent to the previous quotient.
 - e. In summary: $\text{ALOG2}(X) = \text{exponent of } X - 0.5 + \text{ALOG}(\text{SQRT}(2)*\text{RRX})$

where: RRX is X with the exponent set to 0.0.
5. The largest known relative error is less than $1*2^{*-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

AER = REA, which means that the RER for arguments near 1.0 may be large.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.EQ. 0.0), the result is set to the most negative number and FXEM is called with error code 9.
2. If (X.LT.0.0), the result is set to 0.0 and FXEM is called with error code 10.

Real Hyperbolic Arcsine, Arccosine, and Arctangent

FUNCTION

FASH computes $\sinh^{-1}(a)$ for ASINH(X), the value of the hyperbolic arcsine of X for all inputs; $\cosh^{-1}(a)$ for ACOSH(X), the value of the hyperbolic arccosine; and $\tanh^{-1}(a)$, the value of the hyperbolic arctangent.

USAGE

A=ASINH(X)
A=ACOSH(X)
A=ATANH(X) } X is real.

GMAP CALLING SEQUENCE

CALL ASINH(X) for $\sinh^{-1}(a)$
CALL ACOSH(X) for $\cosh^{-1}(a)$
CALL ATANH(X) for $\tanh^{-1}(a)$

METHOD

1. ASINH is calculated as $\text{ALOG}(X+\text{SQRT}(X**2+1.0))$.
2. ACOSH is calculated as $\text{ALOG}(X+\text{SQRT}(X**2-1.0))$.
For the above two expressions, $\text{SQRT}(X**2-1.)$ is set equal to X for $X.\text{GE}.1.0\text{E}5$.
3. ATANH is calculated as $.5*\text{ALOG}((1.+X)/(1.-X))$.
4. These subroutines call other routines that are accurate to single precision. An attempt was made to minimize the accumulated error in calculating the mathematically correct expressions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.LT.1.0) for ACOSH, the result is set to 0.0 and FXEM is called with error code 91.
2. If (X.LE.-1.0) for ATANH, the result is set to the most negative number and FXEM is called with error code 92.
3. If (X.GE.1.0) for ATANH, the result is set to the most positive number and FXEM is called with error code 92.

Real Arcsine and Arccosine

FUNCTION

FASN computes $\sin^{-1}(a)$ for ARSIN(X), the principal value of the trigonometric arcsine of X (in radians); and $\cos^{-1}(a)$ for ARCOS(X), the principal value of the trigonometric arccosine of X (in radians).

USAGE

A=ARSIN(X)
A=ARCOS(X)

X represents any number within the range (-1.0, 1.0), meaning $-1.0 \leq X \leq 1.0$. The arcsine and arccosine are undefined for arguments outside of this range.

GMAP CALLING SEQUENCE

CALL ARSIN(X) for $\sin^{-1}(a)$
CALL ARCOS(X) for $\cos^{-1}(a)$

METHOD

1. Initial range reduction is performed by setting $AX=ABS(X)$, which reduces the range to (0, ARSIN(PI/2)).
2. The range between (0, ARSIN(PI/2)) is divided into four parts, with the partition points of R expressed as follows:

R(0) = 0. = SIN(0)
R(1) = .5 = SIN(PI/6)
R(2) = .866 = SIN(PI/3)
R(3) = .965 = SIN(5*PI/12)
R(4) = 1.0 = SIN(PI/2)

3. The basic formulas used to derive the actual formulas used in the subroutine for the above four ranges are:

- a. $TN(X) = \cos(N \cdot \text{ARCOS}(X))$, which means:
 - T1(X) = X
 - T2(X) = $2 \cdot X^2 - 1$
 - T4(X) = $8 \cdot X^4 - 8 \cdot X^2 + 1$

This is the standard Chebyshev polynomial of the first kind.

- b. $\text{ARSIN}(X) = (N-1) \cdot \text{PI} / (2 \cdot N) + (1/N) \cdot \text{ARSIN}(TN(X))$

where: $\text{SIN}((N-2) \cdot \text{PI} / (2 \cdot N)) \leq X \leq 1.0$.

This form is most efficient near:

$$\text{SIN}(\text{PI}/2 - \text{PI}/(2*N)), \text{ where } \text{TN}(X) \text{ approaches } 0.$$

c. $\text{ARSIN}(X) = \text{PI}/2 - 2*\text{ARSIN}(\text{SQRT}((1-X)/2)).$

d. $\text{ARCOS}(X) = \text{PI}/2 - \text{ARSIN}(X) \text{ for } 0 \leq \text{ARCOS}(X) \leq \text{PI}.$

4. The following chart presents the applicable formulas and the values of N for the different ranges.

Interval $\text{SIN}(A) \leq X \leq \text{SIN}(B)$ A	Transformation Used Formula	N	Transform Range $\text{RRX} \leq \text{SIN}(Z)$ Z
0	PI/6	None	PI/6
PI/6	PI/3	(3a) 2	PI/6
PI/3	5*PI/12	(3b) 4	PI/6
5*PI/12	PI/2	(3c)	PI/24

Two different minimax polynomials are used to approximate ARSIN(RRX) over the ranges (0, PI/6) and (0, PI/24). For the range (0, PI/6), a polynomial of the form

$$Y*P(Y**2)/Q(Y**2)$$

is used, where P is of the second order and Q is of the third order. For the range (0, PI/24), P and Q are both of the first order.

5. Using all of the above formulas, the following formulas actually used in the subroutine can be derived.

Range 1:

for X (0.0, 0.5); equivalently
X (SIN(0.0), SIN(PI/6)), or
X (COS(PI/2), COS(5*PI/6));

$$\begin{aligned} \text{PQARG} &= X \\ \text{ARSIN} &= 0.0 + \text{PQARG}*1.0*\text{P}(\text{PQARG**2})/\text{Q}(\text{PQARG**2}) \\ \text{ARCOS} &= \text{PI}/2. - \text{PQARG}*1.0*\text{P}(\text{PQARG**2})/\text{Q}(\text{PQARG**2}) \end{aligned}$$

Range 2:

for X (0.5, .866025404); equivalently
X (SIN(PI/6), SIN(PI/3)), or
X (COS(5*PI/6), COS(2*PI/3));

$$\begin{aligned} \text{PQARG} &= 2*X**2-1 \\ \text{ARSIN} &= \text{PI}/4. + \text{PQARG}*0.5*\text{P}(\text{PQARG**2})/\text{Q}(\text{PQARG**2}) \\ \text{ARCOS} &= \text{PI}/4. - \text{PQARG}*0.5*\text{P}(\text{PQARG**2})/\text{Q}(\text{PQARG**2}) \end{aligned}$$

Range 3:

for X (.866025404, .965925826); equivalently
 X (SIN(PI/3), SIN(5*PI/12)), or
 X (COS(2*PI/3), COS(PI/12));

PQARG = $8*X^{**4} - 8*X^{**2} + 1.$
 ARSIN = $3*PI/8 + PQARG*0.25*P(PQARG^{**2})/Q(PQARG^{**2})$
 ARCOS = $PI/8 - PQARG*0.25*P(PQARG^{**2})/Q(PQARG^{**2})$

Range 4 (uses a different polynomial approximation):

for X (.965925826, 1.0); equivalently
 X (SIN(5*PI/12), SIN(PI/2)), or
 X (COS(PI/12), COS(0.0));

PQARG = $(1-X)/2.$
 ARSIN = $PI/2 - SQRT(PQARG)*2*P(PQARG^{**2})/Q(PQARG^{**2})$
 ARCOS = $0.0 + SQRT(PQARG)*2*P(PQARG^{**2})/Q(PQARG^{**2})$

Now, for all ranges

If the original argument was negative:

ARSIN = -ARSIN
 ARCOS = PI - ARCOS

Then, return.

For ranges 1, 2, and 3, the same formula for the polynomial is used. In range 4, a different formula is used and a call to SQRT must be made.

6. The largest known relative error is less than $1*2^{*-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

$AER = AEA/(1-X^{**2})^{*.5}$

If the value of X is small, RER = AEA for the arcsine.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.LT.-1.0), FXEM is called with error code 83. Then the answer is found for X = -1.0.
2. If (X.GT.1.0), FXEM is called with error code 83. Then the answer is found for X = +1.0.

Real Arctangent

FUNCTION

FATN computes the principal value (in radians) of the arctangent of Z for ATAN(Z) or the arctangent of (Y,X) for ATAN2 (Y,X), where $Z = Y/X$, for any valid input argument(s).

USAGE

A=ATAN(Z)
A=ATAN2(Y,X)

(Y,X) or Z are any numbers, except that the expression (Y,X)=(0.,0.) is invalid.

CALLING SEQUENCE

CALL ATAN(Z) for $\tan^{-1}(a)$
CALL ATAN2(Y,X) for $\tan^{-1}(a/b)$

METHOD

1. The algorithm is the same for both ATAN and ATAN2, except that for ATAN2(Z) the Z is calculated as $Z = Y/X$. For ATAN(Z), the answer may be located in either quadrant 1 or quadrant 4 ($-\pi/2 \leq \text{ATAN} \leq \pi/2$). For ATAN2(Y,X), the answer will be in the correct quadrant with ($-\pi \leq \text{ATAN2} \leq \pi$).
2. The ABS(Z) is found, thus reducing the range to (0.0, arctangent ($\pi/2$)).
3. The range between (0, arctangent ($\pi/2$)) is then divided into nine parts with partition points R and evaluation nodes S. The partition points are chosen as follows:

$$\begin{aligned} R(0) &= 0 \\ R(I) &= \text{tangent}((2*I-1)*\pi/32.) \quad I=1,2,\dots,8 \\ R(9) &= \text{infinity} \\ S(I) &= \text{tangent}((2*I-2)*\pi/32.) \quad I=2,3,\dots,9 \end{aligned}$$
4. The range of Z, and thus the value of the index I, is then found within the range (R(I-1), R(I)). That is, $R(I-1) < Z < R(I)$. I is found using a binary search to determine the range of Z. After I is found, the formula

$$\text{ATAN}(Z) = \text{ATAN}(S(I)) + \text{ATAN}(T)$$

is used, where: $T = S(I)^{-1} - (S(I)^{-2} + 1) / (S(I)^{-1} + Z)$.

5. ATAN(S(I)) and the values for (S(I)**-1) and (S(I)**-2+1) are stored in tables.
6. T is within the range (-PI/32., PI/32.) and ATAN(T) is calculated using a polynomial approximation of the third order which has the form
$$P00+T*(P01+T*(P02+T*P03))$$
and provides accuracy beyond single precision.
7. After the value of ATAN is found for positive Z (quadrant 1), the answer is placed in the proper quadrant by subtracting it from PI and/or negating it if necessary.
8. The largest known relative error is less than $1*2**{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

AER=AEA/(1-Z**2) for a lesser value of Z.
RER=REA for the larger values of Z.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (Y.EQ.0.0.AND. X.EQ.0.0), the result is set to 0.0 and FXEM is called with error code 11.

Complex Absolute Value

FUNCTION

FCAB computes the absolute value of the complex number X in the expression CABS(X).

USAGE

A=CABS(X)

X represents any complex number of the form (A,B).

GMAP CALLING SEQUENCE

CALL CABS(X)

METHOD

1. The complex argument X is treated as a number pair (A,B):
SML = MIN(A,B)
BIG = MAX(A,B)
2. CABS = ABS(BIG*SQRT(1.+(SML/BIG)**2)). Note that the answer is a real number.
3. The largest known relative error is less than $1*2**{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

RER = (1./2.)*REA

RETURNS

Normal return is to the next executable statement of the calling program.

Complex Exponential

FUNCTION

FCEX computes e^{**Z} for CEXP(Z) in an expression.

USAGE

A=CEXP(Z) Z is complex.

GMAP CALLING SEQUENCE

CALL CEXP(Z)

METHOD

1. $Z = (X, Y)$
 $A = Y + \frac{\pi}{2}$
 $B = e^{**X}$
 $X = B * \text{SIN } A$
 $Y = B * \text{SIN } Y$
ANSWER = (X, Y)
2. Z and e^{**X} are complex numbers (X, Y), with
 $X \leq 88.028$, $|Y| < 2^{**27}$, and $\left| Y + \frac{\pi}{2} \right| < 2^{**27}$.
3. Each part (X, Y) of e^{**Z} is accurate to seven decimal positions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. Error code 26 if $X > 88.028$. Then $e^{**Z} = (10^{**38}, 10^{**38})$. Execution continues.
2. Error code 27 if $|Y| \geq 2^{**27}$ or if $|Y + \frac{\pi}{2}| \geq 2^{**27}$. Then $e^{**Z} = (0.0)$. Execution continues.

Complex Natural Logarithm

FUNCTION

FCLG computes $\log_e Z$ for CLOG(Z) in an expression.

USAGE

A=CLOG(Z) Z is complex.

GMAP CALLING SEQUENCE

CALL CLOG(Z)

METHOD

1. $\log_e Z = \log_e(X, Y)$ (where $Z = (X, Y)$)
 $\quad = (\log_e |Z|, \arctan Y/X)$
2. Z and $\log_e Z$ are complex numbers; values of X and Y range from $-2^{**}127$ to $2^{**}127 - 2^{**}100$ inclusive.
3. $\log_e Z$ is accurate to seven decimal positions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. Error code 28 if $Z = (0, 0)$. Then $\log_e Z = (-10^{**}38, 0.0)$.

Complex Multiplication and Division

FUNCTION

FCMP computes $(A,B) * (C,D)$ or $(A,B) / (C,D)$ in an expression.

USAGE

$A=B*C$ or $A=B/C$ B and C are complex.

GMAP CALLING SEQUENCE

CALL .FCFMP(R,S) for R*S

CALL .FCFDP(R,S) for R/S

R = (A,B)

S = (C,D)

METHOD

1. $(A,B) * (C,D) = (A*C - B*D, A*D + B*C)$
2. $(A,B) / (C,D) = ((A,B) * (C,-D)) / (C**2 + D**2) = (A*C + B*D, B*C - A*D) / (C**2 + D**2)$
3. If $(A,B) = (0,0)$, then the quotient = $(0,0)$. Otherwise,
 - U = D/C
 - Y = (A/C) / (1+U**2)
 - T = B/A
 - X = Y*(T*U+1)
 - Y = Y*(T-U)
4. Before computing $(A,B) / (C,D)$, replace the numerator by $(-B,A)$ if $|A| < |B|$, and the denominator by $(-D,C)$ if $|C| \leq |D|$. Adjust the quotient (X,Y) accordingly:
 - a. If $|A| > |B|$ and $|C| > |D|$, then the result = (X,Y) .
 - b. If $|A| \leq |B|$ and $|C| \leq |D|$, then the result = (X,Y) .
 - c. If $|B| \geq |A|$ and $|C| > |D|$, then the result = $(Y,-X)$.
 - d. If $|A| > |B|$ and $|D| \geq |C|$, then the result = $(-Y,X)$.

5. A, B, C, D, X, and Y are real numbers, with values from -2^{127} to $(2^{127})-2^{100}$ inclusive.
6. The answer is accurate to eight decimal positions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. In a division operation, error code 25 if $(C,D)=(0,0)$. Then $(A,B)/(C,D)=(10^{38},10^{38})$. Execution continues.

Real Cube Root

FUNCTION

FCRT computes the cube root of X in the expression CBRT(X).

USAGE

A=CBRT(X)

X represents any number.

GMAP CALLING SEQUENCE

CALL CBRT(X)

METHOD

1. X is expressed as $RM*2^{**}IE$. RM is real. IE is integer. Range reduction is performed using the formula

$$CBRT(RM*2^{**}IE) = RM/3*CBRT(2^{**}IE).$$

If $I=IE/3$ and $J=IE-3*I$, then

$$CBRT(RM*2^{**}IE) = CBRT(RM*2^{**}J)*2^{**}I.$$

2. RRX is $RM*2^{**}J$ and is within the range (.5,4.). The $CBRT(RRX)$ is found using an initial polynomial approximation of the form

$$Y0 = P00+RRX*(P01+RRX*P02)$$

which makes Y0 accurate to approximately 1.8 decimal digits.

3. Three Newton's iterations are then used with the formula

$$Y1 = Y0 - (Y0 - X/Y0^{**}2) / 3$$

to extend the accuracy to full single precision.

4. The largest known relative error is less than $1*2^{**-27}$ (one bit in single precision).

FCRT

FCRT

EFFECT OF ARGUMENT ERROR

RER = (1./3.)*REA-(1./9.)*REA**2...

RETURNS

Normal return is to the next executable statement of the calling program.

Complex Sine and Cosine

FUNCTION

FCSN computes $\sin Z$ or $\cos Z$ for CSIN(Z) or CCOS(Z) in an expression, where Z is in radians.

USAGE

A=CSIN(Z) or A=CCOS(Z) Z is complex.

GMAP CALLING SEQUENCE

CALL CSIN(Z) for $\sin Z$

CALL CCOS(Z) for $\cos Z$

METHOD

1. $\sin Z = \sin(X, Y)$ (where $Z = (X, Y)$)
 $= \sin X \cos(0, Y) + \cos X \sin(0, Y)$
 $= (\sin X \cosh Y, 0) + (0, \cos X \sinh Y)$
 $= (\sin X \cosh Y, \cos X \sinh Y)$
2. $\cos Z = \sin \left(Z + \frac{\pi}{2} \right)$
3. Z, $\sin Z$, and $\cos Z$ are complex numbers, with $|X| < 2^{**27}$,
 $|X + \frac{\pi}{2}| < 2^{**27}$, and $|Y| < 88.028$.
4. The answer is accurate to seven decimal positions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. Error code 29 if: $|X| \geq 2^{**27}$

$$|X + \pi/2| \geq 2^{**27}$$

$$e^{**Y} = 0$$

Then the answer is (0,0). Execution continues.

2. Error code 30 if $|Y| > 88.028$. Then the answer is $(10^{**38}, 10^{**38})$. Execution continues.

Complex Square Root

FUNCTION

FCSQ computes the square root of Z in the expression CSQRT(Z).

USAGE

A=CSQRT(Z)

Z is any complex number of the form (A,B).

GMAP CALLING SEQUENCE

CALL CSQRT(Z)

METHOD

1. The argument is expressed as $A+B*i$.
2. $ANSA = \text{SQRT}(/A/+CABS((A,B))/2.)$.
3. If $(A.LT.0.AND.B.LT.0)$, $ANSA = -ANSA$.
4. $ANSB = B/(2.*ANSA)$.
5. If $(A.LT.0)$, return answer $(ANSB, ANSA)$; else return $(ANSA, ANSB)$.
6. These formulas can be derived from the definitions for complex number multiplication.
7. The largest known relative error is less than $1*2**{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

1. Using $i = (-1.)**.5$, express $A+B*i$ as $R1*e**(A1*i)$.
2. Express the answer as $R2*e**(A2*i)$.
3. $RER(R2) = .5*REA(R1)$
 $RER(A2) = REA(A1)$

RETURNS

Normal return is to the next executable statement of the calling program.

Exponentiation - Complex Base, Integer Exponent

FUNCTION

FCXP computes $A^{**}K$ in an expression.

USAGE

$C=A^{**}K$ A is complex. K is an integer.

GMAP CALLING SEQUENCE

CALL .FCXP1(A,K)

METHOD

1. If $A=(0,0)$, $K=0$ then $A^{**}K=(0,0)$ error code 14
2. If $A=(0,0)$, $K>0$ then $A^{**}K=(0,0)$
3. If $A=(0,0)$, $K<0$ then $A^{**}K=(10^{**}38,0)$ error code 15
4. A and C are complex numbers (X,Y) with values X and Y from $-2^{**}127$ to $(2^{**}127)-2^{**}100$.
K is an integer with values from $-2^{**}35$ to $(2^{**}35)-1$.
5. Each $C(X,Y)$ is accurate to eight decimal positions.

RETURNS

Normal return is to the next executable statement in the calling program.

1. Error code 14 if $A=(0,0)$ and $K=0$. Then $A^{**}K=(0,0)$.
2. Error code 15 if $A=(0,0)$ and $K<0$. Then $A^{**}K=(10^{**}38,0.0)$.

Double Precision Hyperbolic Arcsine, Arccosine, and Arctangent

FUNCTION

FDAH computes $\sinh^{-1}(a)$ for DASINH(X), the value of the hyperbolic arcsine of X for all inputs; $\cosh^{-1}(a)$ for DACOSH(X), the value of the hyperbolic arccosine; and $\tanh^{-1}(a)$ for DATANH(X), the value of the hyperbolic arctangent.

USAGE

A=DASINH(X)
 A=DACOSH(X)
 A=DATANH(X) } X is double precision.

GMAP CALLING SEQUENCE

CALL DASINH(X) for $\sinh^{-1}(a)$
 CALL DACOSH(X) for $\cosh^{-1}(a)$
 CALL DATANH(X) for $\tanh^{-1}(a)$

METHOD

1. DASINH is calculated as $\text{DALOG}(X + \text{DSQRT}(X^2 + 1.0))$.
2. DACOSH is calculated as $\text{DALOG}(X + \text{DSQRT}(X^2 - 1.0))$.

For the above two expressions, $\text{DSQRT}(X^2 - 1.0)$ is set equal to X for $X \geq 1.0E10$.

3. DATANH is calculated as $.5 * \text{DALOG}((1+X)/(1-X))$.
4. These subroutines call other routines that are accurate to double precision. An attempt was made to minimize the accumulated error in calculating the mathematically correct expressions.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If $(X < 1.0)$ for DACOSH, the result is set to 0.0 and FXEM is called with error code 91.
2. If $(X \leq -1.0)$ for DATANH, the result is set to the most negative number and FXEM is called with error code 92.
3. If $(X \geq 1.0)$ for DATANH, the result is set to the most positive number and FXEM is called with error code 92.

Double Precision Arcsine and Arccosine

FUNCTION

FDAS computes $\sin^{-1}(a)$ for DARSIN(X), the value of the trigonometric arcsine of X (in radians); and $\cos^{-1}(a)$ for DARCOS(X), the value of the trigonometric arccosine of X (in radians).

USAGE

A=DARSIN(X)
A=DARCOS(X)

X represents any number within the range $(-1.0, 1.0)$, meaning $-1.0 \leq X \leq 1.0$. The arcsine and arccosine are undefined for arguments outside of this range.

GMAP CALLING SEQUENCE

CALL DARSIN(X) for $\sin^{-1}(a)$
CALL DARCOS(X) for $\cos^{-1}(a)$

METHOD

1. The method used for DARSIN is the same as that used for the single-precision ARSIN subroutine (edit name FASN), except that more accurate minimax polynomials are used for the approximation to DARSIN.
2. The polynomial used in ranges 1, 2, and 3 is of the order 6 and the polynomial used in range 4 has P of the order 3 and Q of the order 4.
3. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

$AER = AEA / (1 - X^2)^{.5}$

If the value of X is small, $RER = AEA$ for the arcsine.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.LT.-1.0), FXEM is called with error code 83. Then the answer is found for X=-1.0.
2. If (X.GT.1.0), FXEM is called with error code 83. Then the answer is found for X=+1.0.

Double Precision Arctangent

FUNCTION

FDAT computes the principal value (in radians) of the arctangent of Z for DATAN(Z) or the arctangent of (Y,X) for DATAN2(Y,X), where $Z=Y/X$ for any valid input argument(s).

USAGE

A=DATAN(Z)
A=DATAN2(Y,X)

(Y,X) or Z are any numbers, except that the expression (Y,X) = (0.,0.) is invalid.

GMAP CALLING SEQUENCE

CALL DATAN(Z) for $\tan^{-1}(a)$
CALL DATAN2(Y,X) for $\tan^{-1}(a/b)$

METHOD

1. The algorithm is the same for both DATAN and DATAN2, except that for DATAN2(Z) the Z is calculated as $Z=Y/X$. For DATAN(Z), the answer may be located in either quadrant 1 or quadrant 4 ($-\pi/2 \leq \text{DATAN} \leq \pi/2$). For DATAN2(Y,X), the answer will be in the correct quadrant with ($-\pi \leq \text{DATAN2} \leq \pi$).
2. The method used for DATAN and DATAN2 is the same as that used for the single-precision subroutines ATAN and ATAN2 (edit name FATN), except that a minimax polynomial of the order seven (instead of the order three) is used for approximating the DATAN function. Using the same notation described for the ATAN function, the polynomial is

$$P00+1*(P01+T*(P02+T*(P03+T*(P04+T*(P05+T*(P06+T*P07))))))$$

which provides accuracy suitable for double precision.

3. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

FDAT

FDAT

EFFECT OF ARGUMENT ERROR

AER=AEA/(1-Z**2) for a lesser value of Z.
RER=REA for the larger values of Z.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (Y.EQ.0.0.AND.X.EQ.0.0), the result is set to 0.0 and FXEM is called with error code 24.

Double Precision Cube Root

FUNCTION

FDCR computes the cube root of X in the expression DCBRT(X).

USAGE

A=DCBRT(X)

X represents any double-precision number.

GMAP CALLING SEQUENCE

CALL DCBRT(X)

METHOD

1. The method used for DCBRT is the same as that used for the single-precision CBRT subroutine (edit name FCRT), except that one more Newton's iteration is performed (for a total of four) to provide accuracy to full double precision.

EFFECT OF ARGUMENT ERROR

$RER = (1./3.)*REA - (1./9.)*REA**2...$

RETURNS

Normal return is to the next executable statement of the calling program.

Double Precision Exponential, Base 2, and Base 10

FUNCTION

FDEX computes e^a for DEXP(X), the value of the exponential of X for all valid input argument(s); 2^a for DEXP2(X), the value of $2.**X$ and 10^a for DEXP10(X), the value of $10.**X$.

USAGE

A=DEXP(X)
 A=DEXP2(X)
 A=DEXP10(X)

X represents any double-precision value so that the result will be presented as a double-precision number.

GMAP CALLING SEQUENCE

CALL DEXP(X)
 CALL DEXP2(X)
 CALL DEXP10(X)

METHOD

1. The basic function calculated by the subroutine is DEXP2(X). The results for DEXP and DEXP10 are calculated considering that

$$\begin{aligned} \text{DEXP}(X) &= \text{DEXP2}(X \cdot \log_2(e)) \quad \text{and} \\ \text{DEXP10}(X) &= \text{DEXP2}(X \cdot \log_2(10)). \end{aligned}$$

The constants $\log_2(e)$ and $\log_2(10)$ are stored within the subroutine.

2. Range reduction is performed considering that

$$\text{DEXP2}(\text{RRX} + \text{INTX}) = \text{DEXP2}(\text{RRX}) * \text{DEXP2}(\text{INTX}).$$

INTX is the integer value nearest to X and RRX is X minus INTX. Therefore, RRX is within the range $(-.5, .5)$.

3. DEXP2(RRX) is found using a minimax polynomial approximation of the form

$$(Q(X^{**2}) + X * P(X^{**2})) / (Q(X^{**2}) - X * P(X^{**2}))$$

where the polynomial P is of the second order and Q is of the third order. This provides results that are suitable for double precision.

4. $DEXP2(RRX) * DEXP2(INTX)$ is found by adding $INTX$ to the exponent of the result for $DEXP2(RRX)$.
5. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

RER = AEA
AER = REA*X

NOTE: Serious error amplification can occur if small errors are encountered in a large value of X .

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If $(result.GT.2.**127)$, the result is set to the largest number and $FXEM$ is called with error code 19.
2. If $(result.LT.2.**-128)$, the result is set to 0.0 and $FXEM$ is called with error code 89.

Double Precision Natural, Base 2, and Common Logarithms

FUNCTION

FDLG computes $\log_e(a)$ for DLOG(X), the natural logarithm of any input argument X where $X.GT.0.0$; $\log_2(a)$ for DLOG2(X), the logarithm base 2; and $\log_{10}(a)$ for DLOG10(X), the common logarithm or logarithm base 10.

USAGE

A=DLOG(X)
 A=DLOG2(X)
 A=DLOG10(X)

X represents any double-precision number .GT.0.0.

GMAP CALLING SEQUENCE

CALL DLOG(X) for $\log_e(a)$
 CALL DLOG2(X) for $\log_2(a)$
 CALL DLOG10(X) for $\log_{10}(a)$

METHOD

1. The method used in this subroutine is exactly the same as that used for the single-precision subroutine ALOG, ALOG2, and ALOG10 (edit name FALG), except that a more accurate polynomial approximation is used. The polynomial used for approximating DLOG2 is of the form

$$Z^*P(Z^{**2})/Q(Z^{**2})$$

where Z is the same as that found in the single-precision subroutine.

2. P is a third-order polynomial of the form

$$P00+Z^{**2}*(P01+Z^{**2}*(P02+Z^{**2}*P03))$$

and Q is a fourth-order polynomial of the form

$$Q00+Z^{**2}*(Q01+Z^{**2}*(Q02+Z^{**2}*(Q03+Z^{**2}*Q04)))$$

which provides results suitable for double precision.

3. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

AER = REA, which means that the RER for arguments near 1.0 may be large.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.EQ.0.0), the result is set to the most negative number and FXEM is called with error code 20.
2. If (X.LT.0.0), the result is set to 0.0 and FXEM is called with error code 21.

Double Precision Remaindering

FUNCTION

FDMD computes $A=X(\text{mod } Y)$ for $\text{DMOD}(X,Y)$ in an expression.

USAGE

$A=\text{DMOD}(X,Y)$ X and Y are double precision.

GMAP CALLING SEQUENCE

CALL $\text{DMOD}(X,Y)$

METHOD

1. If $Y=0$, then $A=X$. Otherwise, compute $Z =$ the greatest integer $\leq X/Y$ and give Z the same sign as that of X/Y . Then $A=X- Y*Z$.
2. A , X , and Y are double-precision numbers, with values from -2^{127} to $(2^{127})-2^{64}$ inclusive.
3. A is accurate to 63 binary positions.

NOTE: If $X>Y*10^{10}$, the results may be incorrect; A may be greater than Y .

RETURNS

Normal return is to the next executable statement in the calling program.

Double Precision Hyperbolic Sine, Cosine, and Tangent

FUNCTION

FDPH computes $\sinh(a)$ for DSINH(X), the value of the hyperbolic sine of X for all valid input arguments; $\cosh(a)$ for DCOSH(X), the value of the hyperbolic cosine; and $\tanh(a)$ for DTANH(X), the value of the hyperbolic tangent.

USAGE

A=DSINH(X)
A=DCOSH(X)
A=DTANH(X)

X represents any double-precision value so that the result will be presented as a double-precision number.

GMAP CALLING SEQUENCE

CALL DSINH(X) for $\sinh(a)$
CALL DCOSH(X) for $\cosh(a)$
CALL DTANH(X) for $\tanh(a)$

METHOD

1. DCOSH(X) is calculated as $(\text{DEXP}(X)+1./\text{DEXP}(X))/2.0$.
2. DSINH(X) is calculated as follows:
If $(/X/.GE.1.44269)$, $\text{DSINH} = (\text{DEXP}(X)-1./\text{DEXP}(X))/2.0$.
If $(/X/.LT.1.44269)$, $\text{DSINH} = (\text{DEXPC}(X)/(\text{DEXPC}(X)+1.0)+\text{DEXPC}(X))/2.0$.
3. DTANH(X) is calculated as follows:
If $(X.GE.22.0)$, $\text{DTANH} = 1.0$.
If $(X.LE.-22.0)$, $\text{DTANH} = -1.0$.
If $(/X/.LT.1.44269)$, $\text{DTANH} = \text{DEXPC}(2*X)/(2.+ \text{DEXPC}(2*X))$.
If $(/X/.GT.1.44269.AND./X/.LT.22.0)$,
 $\text{DTANH} = (\text{DEXP}(X)-1./\text{DEXP}(X))/(\text{DEXP}(X)+1./\text{DEXP}(X))$.
4. These subroutines call other routines that are accurate to double precision. The complementary forms of the exponential functions were used to help minimize accumulated errors in calculating the mathematically correct expressions.

EFFECT OF ARGUMENT ERROR

1. For DSINH: $AER = AEA * \cosh(X)$
 $RER = AEA * \cosh(X) = REA$
2. For DCOSH: $AER = AEA * \operatorname{dsinh}(X)$
 $RER = REA * \tanh(X) = AEA$
3. For DTANH: $AER = (1. - \tanh(X)^2) * AEA$
 $RER = 2. * AEA / \sinh(2 * X)$

For a small value of X, RER = REA; for a large value of X, RER is affected less and less by REA.

RETURNS

No error handling is performed by this subroutine, but potential underflow/overflow conditions will be displayed by the DEXP or DEXPC routines that are called by this subroutine.

Double Precision Sine and Cosine

FUNCTION

FDSN computes $\sin(a)$ for DSIN(X), the value of the trigonometric sine of X for any valid input argument (in radians); and $\cos(a)$ for DCOS(X), the value of the trigonometric cosine of X for any valid input argument (in radians).

USAGE

A=DSIN(X)
A=DCOS(X)

X represents any double-precision number, but accuracy may be affected in finding the sine or cosine of large input arguments due to the periodicity of the trigonometric functions.

GMAP CALLING SEQUENCE

CALL DSIN(X) for $\sin(a)$
CALL DCOS(X) for $\cos(a)$

METHOD

1. The method used for this subroutine is equivalent to that used for the single-precision sine and cosine subroutines (edit name FSIN), except that a more accurate minimax polynomial approximation is used. The minimax polynomial is of the ninth order and has the form

$$P(Y) = P00 + Y * (P01 + Y * (P02 + Y * (P03 + Y * (P04 + Y * (P05 + Y * (P06 + Y * (P07 + Y * (P08 + Y * P09)))))))).$$

2. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

AER = AEA

NOTE: The relative error may increase for larger arguments outside of the principal range $(-\pi, +\pi)$.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If $(/X.GT.2^{*}27)$, FXEM is called with error code 23 to issue a warning. Then the sine or cosine is found with as much accuracy as possible.

Double Precision Square Root

FUNCTION

FDSQ computes the square root of X in the expression DSQRT(X), where X.GE.0.0.

USAGE

A=DSQRT(X)

X represents any double-precision number .GE.0.0.

GMAP CALLING SEQUENCE

CALL DSQRT(X)

METHOD

1. The method used for DSQRT is exactly the same as that used for the single-precision SQRT subroutine (edit name FSQR), except that one more Newton's iteration is performed (for a total of three).

EFFECT OF ARGUMENT ERROR

RER = .5*REA

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.LT.0.0), FXEM is called with error code 22. Then find DSQRT(/X/).

Double Precision Tangent

FUNCTION

FDTN computes $\tan(a)$ for DTAN(X), the value of the trigonometric tangent of X for any valid input argument (in radians).

USAGE

A=DTAN(X)

X represents any double-precision number, but accuracy may be affected in finding the tangent of large input arguments due to the periodicity of the trigonometric functions.

GMAP CALLING SEQUENCE

CALL DTAN(X) for $\tan(a)$

METHOD

1. The algorithm used to find DTAN(X) is identical to that used for the single-precision tangent subroutine (edit name FTAN), except that a more accurate minimax polynomial is used. In this case, the polynomials P and Q are of the fourth order; P has the form

$$P00+Y*(P01+Y*(P02+Y*(P03+Y*P04)))$$

and Q has the form

$$Q00+Y*(Q01+Y*(Q02+Y*(Q03+Y*Q04))).$$

2. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

$$\begin{aligned} \text{AER} &= \text{AEA}/(\text{COS}(X)**2) \\ \text{RER} &= 2./\text{SIN}(2*X) \end{aligned}$$

NOTE: Near singularities of the expression $X=(K+.5)*\text{PI}$, large errors in values may occur.

FDTN

FDTN

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (/X/.GT.2**27), FXEM is called with error code 90 to issue a warning. Then the tangent is found with as much accuracy as possible.

Double Precision Exponential Complement

FUNCTION

FDXC computes $e^a - 1.0$ for DEXPC(X), the value of the exponential complement of X ($e^{**X} - 1.0$) for all valid input arguments; $2^a - 1.0$ for DEXPC2(X), the value of $2.**X - 1.0$; and $10^a - 1.0$ for DXPC10(X), the value of $10.**X - 1.0$.

These functions are useful for calculating the above expressions when the arguments are near the value zero. However, if the standard exponential subroutines were used for calculating an expression such as $2.**X - 1.0$, accuracy would be lost for arguments that are near 0.0 because of the subtraction of two almost identical numbers (1.0 and -1.0). This loss of accuracy is avoided by using these subroutines.

USAGE

A=DEXPC(X)
A=DEXPC2(X)
A=DXPC10(X)

X represents any double-precision value so that the result will be presented as a double-precision number.

GMAP CALLING SEQUENCE

CALL DEXPC(X)
CALL DEXPC2(X)
CALL DXPC10(X)

METHOD

1. The basic function calculated by the subroutine is DEXPC2(X). The results for DEXPC and DXPC10 are calculated considering that

$$\begin{aligned} \text{DEXPC}(X) &= \text{DEXPC2}(X \cdot \log_2(e)) \quad \text{and} \\ \text{DXPC10}(X) &= \text{DEXPC2}(X \cdot \log_2(10)). \end{aligned}$$

The constants $\log_2(e)$ and $\log_2(10)$ are stored within the subroutine.

2. For $/X/.LT.0.5$, the function is calculated using a minimax polynomial approximation of the form

$$\text{EXPC2} = 2 \cdot X \cdot P(X^{**2}) / (Q(X^{**2}) - X \cdot P(X^{**2}))$$

$$\begin{aligned} \text{where: } P(Y) &= P00 + Y \cdot (P01 + Y \cdot P02) \quad \text{and} \\ Q(Y) &= Q00 + Y \cdot (Q01 + Y \cdot (Q02 + Y \cdot Q03)). \end{aligned}$$

3. For $/X/.GE.0.5$, the function DEXPC2 is simply calculated as $DEXP2(X)-1.0$. That is, a call to DEXP2 is made followed by a subtraction of 1.0.
4. The minimax polynomial is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

RER = AEA
AER = REA*X

NOTE: Serious error amplification can occur if small errors are encountered in a large value of X.

RETURNS

No error handling is performed by this subroutine. However, for large positive or negative arguments the subroutine DEXP2 is called and it will generate any appropriate error messages.

Double Precision Exponentiation, Real Base or Exponent (or Both)

FUNCTION

FDXP has five entry points that compute the value of $D1^{**}D2$ for any valid double-precision numbers $D1$ and $D2$. For entry points other than `.FDXP2`, the arguments are converted to double precision. The entry points and computation are as follows:

`DPOW(D1,D2)` or `.FDXP2(D1,D2)` for $D1^{**}D2$
`.FDXP3(D1,R2)` for $D1^{**}R2$
`.FXP4(R1,D2)` for $R1^{**}D2$
`.FXP5(I1,R2)` for $I1^{**}R2$
`.FXP6(I1,D2)` for $I1^{**}D2$

where: $D1$ = any double-precision number `.GE.0.0`
 $D2$ = any double-precision number
 $I1$ = any integer `.GE.0`
 $I2$ = any integer
 $R1$ = any single-precision number `.GE.0.0`
 $R2$ = any single-precision number

USAGE

$X = D1^{**}D2$

GMAP CALLING SEQUENCE

CALL <code>DPOW(D1,D2)</code>	}	Notation is described above
CALL <code>.FDXP2(D1,D2)</code>		
CALL <code>.FDXP3(D1,R2)</code>		
CALL <code>.FXP4(R1,D2)</code>		
CALL <code>.FXP5(I1,R2)</code>		
CALL <code>.FXP6(I1,D2)</code>		

METHOD

- The subroutine is based on the formula

$$X^{**}Y = 2^{**(Y*DLOG2(X))}.$$
- The subroutine uses the entry point `.DLOG2` into the `DLOG2` subroutine and the entry point `.DEXP2` into the `DEXP2` subroutine. (`DEXP2(Z)` finds the value of $2^{**}Z$.)
- The minimax polynomial used is accurate to more than 19 places. Any error in the result is caused only by rounding and truncation; the result should be accurate to approximately 17 places.

EFFECT OF ARGUMENT ERROR

The error propagation can be calculated from the guidelines provided in the DLOG2 and DEXP2 subroutines.

RETURNS

Normal return is to the next executable statement in the calling program, except as specified below:

1. If (D1.LT.0.0), the result is set to 0.0 and FXEM is called with error code 16.
2. If (D1.EQ.0.0.AND.D2.EQ.0.0), the result is set to 0.0 and FXEM is called with error code 17.
3. If (D1.EQ.0.0.AND.D2.LT.0.0), the result is set to the largest positive number and FXEM is called with error code 18.
4. Appropriate overflow and underflow errors may occur in DEXP2 (edit name FDEX) if the results cannot be expressed as a double-precision number.

Real Exponential Complement

FUNCTION

FEXC computes $e^a - 1.0$ for EXPC(X), the value of the exponential complement of X ($e^{**X} - 1.0$) for all valid input arguments; $2^a - 1.0$ for EXPC2(X), the value of $2.**X - 1.0$; and $10^a - 1.0$ for EXPC10(X), the value of $10.**X - 1.0$.

These functions are useful for calculating the above expressions when the arguments are near the value zero. However, if the standard exponential subroutines were used for calculating an expression such as $2.**X - 1.0$, accuracy would be lost for arguments that are near 0.0 because of the subtraction of two almost identical numbers (1.0 and -1.0). This loss of accuracy is avoided by using these subroutines.

USAGE

```
A=EXPC(X)
A=EXPC2(X)
A=EXPC10(X)
```

X represents any value so that the result will be presented as a single-precision number.

GMAP CALLING SEQUENCE

```
CALL EXPC(X)
CALL EXPC2(X)
CALL EXPC10(X)
```

METHOD

1. The basic function calculated by the subroutine is EXPC2(X). The results for EXPC and EXPC10 are calculated considering that

$$\begin{aligned} \text{EXPC}(X) &= \text{EXPC2}(X \cdot \log_2(e)) && \text{and} \\ \text{EXPC10}(X) &= \text{EXPC2}(X \cdot \log_2(10)). \end{aligned}$$

The constants $\log_2(e)$ and $\log_2(10)$ are stored within the subroutine.

2. For $/X/.LT.1.0$, the function is calculated using a minimax polynomial approximation of the form

$$P00 + X * (P01 + X * (P02 + X * (P03 + X * (P04 + X * (P05 + X * (P06 + X * P07)))))).$$

3. For $/X/.GE.1.0$, the function EXPC2 is simply calculated as $\text{EXP2}(X) - 1.0$. That is, a call to EXP2 is made followed by a subtraction of 1.0.

FEXC

FEXC

4. The largest known relative error is less than $1 \cdot 2^{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

RER = AEA
AER = REA*X

NOTE: Serious error amplification can occur if small errors are encountered in a large value of X.

RETURNS

No error handling is performed by this subroutine. However, for large positive or negative arguments, the subroutine EXP2 is called and it will generate any appropriate error messages.

Real Exponential, Base 2, and Base 10

FUNCTION

FEXP computes e^a for EXP(X), the value of the exponential of X for all valid input argument(s); 2^a for EXP2(X), the value of $2.0^{**}X$; and 10^a for EXP10(X), the value of $10^{**}X$.

USAGE

A=EXP(X)
A=EXP2(X)
A=EXP10(X)

X represents any value so that the result will be presented as a single-precision number.

GMAP CALLING SEQUENCE

CALL EXP(X)
CALL EXP2(X)
CALL EXP10(X)

METHOD

1. The basic function calculated by the subroutine is EXP2(X). The results for EXP and EXP10 are calculated considering that

$$\begin{aligned} \text{EXP}(X) &= \text{EXP2}(X \cdot \log_2(e)) \quad \text{and} \\ \text{EXP10}(X) &= \text{EXP2}(X \cdot \log_2(10)). \end{aligned}$$

The constants $\log_2(e)$ and $\log_2(10)$ are stored within the subroutine.

2. Range reduction is performed considering that

$$\text{EXP2}(\text{RRX} + \text{INTX}) = \text{EXP2}(\text{RRX}) * \text{EXP2}(\text{INTX}).$$

INTX is the largest integer less than X and RRX is X minus INTX. Therefore, RRX is within the range (0.,1.).

3. EXP2(RRX) is found using a minimax polynomial approximation of the form

$$P00+RRX*(P01+RRX*(P02+RRX*(P03+RRX*(P04+RRX*(P05+RRX*(P06+RRX*P07))))))$$

which provides results suitable for single precision.

4. EXP2(RRX)*EXP2(INTX) is found by adding INTX to the exponent of the result for EXP2(RRX).
5. The largest known relative error is less than $1*2**{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

RER = AEA
AER = REA*X

NOTE: Serious error amplification can occur if small errors are encountered in a large value of X.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (result .GT.2.**127), the result is set to the largest number and FXEM is called with error code 8.
2. If (result .LT.2.**{-128}), the result is set to 0.0 and FXEM is called with error code 89.

Real Sine and Cosine

FUNCTION

FSIN computes $\sin(a)$ for SIN(X), the value of the trigonometric sine of X for any valid input argument (in radians); and $\cos(a)$ for COS(X), the value of the trigonometric cosine of X for any valid input argument (in radians).

USAGE

A=SIN(X)
A=COS(X)

X represents any single-precision number, but accuracy may be affected in finding the sine or cosine of large input arguments due to the periodicity of the trigonometric functions.

GMAP CALLING SEQUENCE

CALL SIN(X) for $\sin(a)$
CALL COS(X) for $\cos(a)$

METHOD

1. The first procedure to be followed in calculating this function is to reduce the range of the argument to $(-.5,+.5)$ by employing the periodicity of the sine or cosine function. The steps are:
 - a. $AX = X*(1./PI)$
 - b. If the entry was for COS(X), then $AX=AX+.5$, since $COS(X) = SIN(PI/2+X)$.
 - c. INTX = the nearest integer to AX.
 - d. If INTX is odd, $RRX = INTX$ minus AX. If INTX is even, $RRX = AX$ minus INTX.
2. Since RRX is now within the range $(-.5,+.5)$, the polynomial must be accurate within the range $(-PI/2,+PI/2)$.
3. A polynomial approximation is then used to find SIN(RRX) and is of the form

$$RRX*2*P(4*RRX**2)$$

where: P(Y) has the form

$$P00+Y*(P01+Y*(P02+Y*(P03+Y*(P04+Y*P05))))).$$

FSIN

FSIN

4. The range reduction described in step 1d above is where the error for large values of the argument may occur.
5. The largest known relative error is less than $1 \cdot 2^{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

AER = AEA

NOTE: The relative error may increase for large arguments outside of the principal range $(-\pi, +\pi)$.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If $(/X/.GT.2^{27})$, FXEM is called with error code 12 to issue a warning. Then the sine or cosine is found with as much accuracy as possible.

Real Square Root

FUNCTION

FSQR computes the square root of X in the expression $\text{SQRT}(X)$, where $X \geq 0.0$.

USAGE

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

X represents any single-precision number ≥ 0.0 .

GMAP CALLING SEQUENCE

CALL SQRT(X)

METHOD

1. X is expressed as $M \cdot 2^E$, where
 - M = the mantissa of X (.5, 1.0) and
 - E = the exponent of X.
2. Range reduction is performed using formulas based on
 - E even: $\text{SQRT}(X) = (2 \cdot \text{SQRT}(.25 \cdot M)) \cdot 2^{E/2}$
 - E odd: $\text{SQRT}(X) = (2 \cdot \text{SQRT}(.5 \cdot M)) \cdot 2^{(E-1)/2}$
3. The $\text{SQRT}(.25 \cdot M)$ or $\text{SQRT}(.5 \cdot M)$ is expressed as $\text{SQRT}(\text{RRX})$ and $E/2$ or $(E-1)/2$ is expressed as EXPNT.
4. $\text{SQRT}(\text{RRX})$, where RRX is (.125, .5), is found using an initial polynomial approximation accurate to more than two places. Two Newton's iterations are then used to find $2 \cdot \text{SQRT}(\text{RRX})$:

$$\begin{aligned} Y_0 &= P_0 + \text{RRX} \cdot (P_1 + P_2 \cdot \text{RRX}) \\ Y_1 &= .5 \cdot (Y_0 + \text{RRX} / Y_0) \\ Y_2 &= 2 \cdot \text{SQRT}(\text{RRX}) = 2 \cdot .5 (Y_1 + \text{RRX} / Y_1) \end{aligned}$$

The result is then expressed as

$$\text{SQRT}(X) = Y_2 \cdot 2^{\text{EXPNT}}$$

5. The largest known relative error is less than $1 \cdot 2^{-27}$ (one bit in single precision).

FSQR

FSQR

EFFECT OF ARGUMENT ERROR

RER = .5*REA

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If (X.LT.0.0), FXEM is called with error code 13. Then SQRT(/X/) is found with as much accuracy as possible.

Real Tangent

FUNCTION

FTAN computes $\tan(a)$ for $\text{TAN}(X)$, the value of the trigonometric tangent of X for any valid input argument (in radians).

USAGE

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

X represents any single-precision number, but accuracy may be affected in finding the tangent of large input arguments due to the periodicity of the trigonometric functions.

GMAP CALLING SEQUENCE

CALL $\text{TAN}(X)$ for $\tan(a)$

METHOD

1. The first procedure to be followed in calculating this function is to reduce the range of the argument to $(-.5, +.5)$ by employing the periodicity of the tangent function. The steps are:
 - a. $AX = X * (2/PI)$. The value $2/PI$ is stored as a constant.
 - b. $\text{INTAX} =$ the nearest integer to AX .
 - c. $\text{RRX} = \text{INTAX}$ minus AX .
2. Since RRX is now within the range $(-.5, +.5)$, the polynomial must be accurate within the range $(-PI/4, +PI/4)$.
3. A minimax polynomial is used of the form
$$\text{RRX} * P(\text{RRX}) / Q(\text{RRX})$$
where the polynomials P and Q are of the second order; $P(Y)$ has the form
$$P00 + Y * (P01 + Y * P02)$$
and Q has the form
$$Q00 + Y * (Q01 + Y * Q02).$$
4. If the original argument was in octant 1, 2, 5, or 6, the result is $Q(\text{RRX}) / (P(\text{RRX}) * \text{RRX})$.

If the original argument was in octant 0, 3, 4, or 7, the result is $-RRX * P(RRX) / Q(RRX)$.

These final adjustments are required because of the symmetry of the tangent function.

5. The largest known relative error is less than $1 * 2^{-27}$ (one bit in single precision).

EFFECT OF ARGUMENT ERROR

$$\begin{aligned} \text{AER} &= \text{AEA} / (\text{COS}(X) ** 2) \\ \text{RER} &= 2. / \text{SIN}(2 * X) \end{aligned}$$

NOTE: Near singularities of the expression $X = (K + .5) * \text{PI}$, large errors in values may occur.

RETURNS

Normal return is to the next executable statement of the calling program, except as specified below:

1. If $(/X / .GT. 2^{-27})$, FXEM is called with error code 90 to issue a warning. Then the tangent is found with as much accuracy as possible.

Real Hyperbolic Sine, Cosine, and Tangent

FUNCTION

FTNH computes $\sinh(a)$ for $\text{SINH}(X)$, the value of the hyperbolic sine of X for all valid input arguments; $\cosh(a)$ for $\text{COSH}(X)$, the value of the hyperbolic cosine; and $\tanh(a)$ for $\text{TANH}(X)$, the value of the hyperbolic tangent.

USAGE

A= $\text{SINH}(X)$
A= $\text{COSH}(X)$
A= $\text{TANH}(X)$

X represents any value so that the result will be presented as a single-precision number.

GMAP CALLING SEQUENCE

CALL $\text{SINH}(X)$ for $\sinh(a)$
CALL $\text{COSH}(X)$ for $\cosh(a)$
CALL $\text{TANH}(X)$ for $\tanh(a)$

METHOD

1. $\text{COSH}(X)$ is calculated as $(\text{EXP}(X)+1./\text{EXP}(X))/2.0$.
2. $\text{SINH}(X)$ is calculated as follows:
If $(/X/.GE.1.44269)$, $\text{SINH} = (\text{EXP}(X)-1./\text{EXP}(X))/2.0$.
If $(/X/.LT.1.44269)$, $\text{SINH} = (\text{EXPC}(X)/(\text{EXPC}(X)+1.0) + \text{EXPC}(X))/2.0$.
3. $\text{TANH}(X)$ is calculated as follows:
If $(X.GE.22.0)$, $\text{TANH} = 1.0$.
If $(X.LE.-22.0)$, $\text{TANH} = -1.0$.
If $(/X/.LT.1.44269)$, $\text{TANH} = \text{EXPC}(2*X)/(2.+ \text{EXPC}(2*X))$.
If $(/X/.GT.1.44269.AND./X/.LT.22.0)$, $\text{TANH} = (\text{EXP}(X)-1./\text{EXP}(X))/(\text{EXP}(X)+1./\text{EXP}(X))$.
4. These subroutines call other routines that are accurate to single precision. The complementary forms of the exponential functions were used to help minimize accumulated errors in calculating the mathematically correct expressions.

EFFECT OF ARGUMENT ERROR

1. For SINH: $AER = AEA * COSH(X)$
 $RER = AEA * COSH(X) = REA$
2. For COSH: $AER = AEA * SINH(X)$
 $RER = REA * TANH(X) = AEA$
3. For TANH: $AER = (1. - TANH(X)**2) * AEA$
 $RER = 2. * AEA / SINH(2 * X)$

For a small value of X, RER = REA; for a large value of X, RER is affected less and less by REA.

RETURNS

No error handling is performed by this subroutine, but potential underflow/overflow conditions will be displayed by the EXP or EXPC routines that are called by this subroutine.

SECTION V

NONMATHEMATICAL LIBRARY SUBROUTINES

This section describes library subroutines that are utilized in the same manner as those described in Section IV. The subroutines described in this section are nonmathematical in nature.

Table 5-1 lists the nonmathematical library subroutines.

Table 5-1. Nonmathematical Library Subroutine

Edit Name	Function	Number
FMDE	Mode determination by FORTRAN	CD600E1.075
FCAT	Character string manipulation	CD600E1.069
FSRT	Sort character array of data	CD600E1.070
FTSF	Memory usage	CD600E1.072
FTSG	User time and identification	CD600E1.071
FRSW	Set/reset switch word	CD600E1.074
FSHF	Shift/rotate word contents	CD600E1.073
FRND	Random number generator	CD600D3.009
FFER	File and Record Control I/O error recovery	CD600E1.011

Mode Determination by FORTRAN

FUNCTION

FMDE is used to provide the FORTRAN user with means to determine whether the environment is batch or time sharing and whether it is BCD or ASCII.

CALLING SEQUENCE:

$j = \text{MODE}(i)$

for $i = 1$, $j = 0$ if batch, $j = 1$ if time sharing

for $i = 2$, $j = 0$ if BCD, $j = 1$ if ASCII

If i is neither 1 nor 2, j is set to -1.

METHOD

The FORTRAN compiler and library provide flags for determination of mode and character set. These are interrogated to give information to the user.

Character String Manipulation

FUNCTION

FCAT is used to provide the FORTRAN user with ability to move or compare a character substring of arbitrary length and position within a string.

CALLING SEQUENCE:

CALL CONCAT (a,n,b,m,f)

where:

a = string to be replaced

n = initial character of a (n=1 implies first character)

b = replacement string

m = initial character of b (m=1 implies first character)

f = number of characters to be replaced; if f is not given, 1 is assumed

Causes characters n through (n + f -1) of a to be replaced with characters m through (m + f -1) of b.

Example:

```
0010 CHARACTER A*20/"FIFTEEN WERE THERE "/
0020 CHARACTER B*20/"SIXTEEN WERE ABSENT "/
0030 PRINT A,B
0040 CALL CONCAT (A,1,B,1,3)
0050 PRINT,A,B
0060 STOP;END
```

READY

*RUN

```
FIFTEEN WERE THERE      SIXTEEN WERE ABSENT
SIXTEEN WERE THERE      SIXTEEN WERE ABSENT
```

i = KOMPCH (a,n,b,m,f) is function where substring of b is compared character by character to substring of a.

if b = a, I = 0

if b > a, I = 1

if b < a, I = -1

METHOD

The move is done by the way of the SC modification after tally words are built using parameters in the calling sequence.

Sort Array of Data

FUNCTION

FSRT is used to sort an array of data in ascending or descending order. SORT is for ascending sort; SORTD is for descending sort.

CALLING SEQUENCE:

CALL SORT/SORTD (array, nrec, lrs, key₁, ..., key_n)

array = the name of the array to be sorted.

nrec = the number of items, or logical records in the array.

lrs = the logical record size, or the size in words of each item in the array.

key = the relative word number of the *i*th sort key in each logical record and must be such that $0 \leq \text{key}_i < \text{lrs}$. Record comparisons are made starting with key₁ and either progress through to key_n, or until a non-equal comparison is made. Any number of sort keys may be specified; however, at least one must always be specified.

METHOD

Implementation is by use of the Shell method as described in ACM Computer Survey, December 1971.

Memory Usage

FUNCTION

FTSF is used to obtain memory usage of a program, the terminal station code, and to move data from/to a 10-word memory file.

CALLING SEQUENCE:

CALL MEMSIZ (j)

j is returned as the number of 1024-word blocks of memory currently allocated this job. The return is a 3-character variable. MEMSIZ can also be used as a function; e.g., IF (MEMSIZ (j) .GE.n) ...

CALL TERMNO (a)

where a is a type character variable capable of receiving the 2-character station code.

For batch, the call returns blanks.

CALL CORFL (loc,i,j,k)

loc = first word address of area to or from which data is to be moved

i = number of words to be moved such that $1 \leq i \leq 10$

j = relative location in the 10-word area at which transfer is to begin

k = 0, data to be transferred into the 10-word area

k = 1, data to be transferred out of the 10-word area

This call is ignored in batch.

METHOD

Memory size is determined from the BAR. TERMNO and CORFL use services provided by the time sharing system.

User Time And Identification

FUNCTION

FTSG is used to obtain user time and user identification.

CALLING SEQUENCE

CALL CORSEC (a)

a, a real variable, is returned as the product of 1024-word blocks currently allocated and processor time in seconds. CORSEC can also be used as a function. For example,

```
IF (CORSEC (A) .GE.B)
```

CALL TERMTM (a)

a, a real variable, is returned as hours since log-on time; ignored in batch.

CALL USRCOD (s)

s is character type variable of at least 12 characters into which the user identification will be returned. (Time sharing only; ignored in batch.)

CALL PTIME (a)

a, a real variable, is returned as elapsed processor time in hours. PTIME can also be used as a function.

METHOD

Service routines provided by the time sharing system are used to obtain data.

Set/Reset Switch Word

FUNCTION

FRSW sets or resets bits in the program switch word.

CALLING SEQUENCE

j = ISETSW(i) for setting switch word.

j = IRETSW(i) for resetting switch word.

i contains an octal value that specifies one or more bits that are to be set or reset.

1 = set bit on

0 = leave bit as is

j is resultant switch word setting in the respective positions of the program switch word.

1 = bit is on

0 = bit is off

In time sharing, bits 0-17 cannot be changed.

METHOD

MME GESETS/GERETS or DRL SETSWH/RSTSWH is used.

Shift/Rotate Word Contents

FUNCTION

FSHF shifts/rotates word contents.

CALLING SEQUENCE:

j = ILS(k,p) k left shift by p bit positions
j = IRS(k,p) k right shift by p bit positions (fill vacated positions
 with contents of bit position zero)
j = ILR(k,p) k left rotate by p bit positions
j = ILR(k,p) k right logical shift by p bit positions (fill vacated
 positions with zeros)
 j = new word
 k = word to shift
 p = number of bits

METHOD

k is shifted as indicated by p bit positions and stored into j. k and p are assumed to be integers; if j is not an integer, the compiler provides a conversion prior to storage.

Random Number Generator

FUNCTION

FRND provides four different calls to generate random numbers from a uniform (rectangular) distribution.

USAGE

A = RAND (range) (seed is always 1)

Use of this version of the function results in the same sequence of random numbers each time the program is executed.

A = RANDT (range) (seed = current time register value)

Use of this version results in a different sequence of random numbers each time the program is executed.

For the above two uses, the algorithm used is the same. A is a positive real number such that $0 < A < \text{range}$.

A = FLAT (seed)

This version allows the seed to be varied, but the range is constant (0 A 1)

A = UNIFM2 (seed, mean, width)

This version allows the seed and the range to be varied. The range will be such that $(\text{mean} - \text{width}/2 < A < \text{mean} + \text{width}/2)$.

For example:

A = UNIFM2 (6.5, 0.5, 1.0)

will generate a set of random numbers between 0.0 and 1.0.

For the above two uses, the algorithm used is the same.

NOTE: For all calls, the value of the initial argument (seed) initializes the algorithm for generation of the sequence of random numbers. For all subsequent calls to the function, during execution of the same program unit, the value of the argument is ignored.

File and Record Control I/O Error Recovery

FUNCTION

FFER provides the user with some control on File and Record Control errors only (not FORTRAN). The subroutine allows the user to set his error routine address into the file control block; a default address is provided if the user does not provide the address.

File control block (FCB) word -5 contains an address for user error recovery. When a file is first used, this word (FCB -5) is set with a pointer to FOPE and the user-specified address, if one is supplied, is placed in the lower half of FCB -15. At any time, the user can place his recovery address in FCB -15 by the following call.

CALLING SEQUENCE

CALL FLGRC (lgu,ptr)

lgu = numeric file code

ptr = address of the recovery routine

METHOD

See Function.

INDEX

.FBD1. .FBDT.	2-8
.FBLT. .FBLT.	2-8
.FBST. CALL .FBST.	2-32
.FCFDP CALL .FCFDP	4-7
.FCFMP CALL .FCFMP	4-7
.FCNV. .FCNV.	2-8
.FCNVC. .FCNVC.	2-8
.FCNVD. .FCNVD.	2-8
.FCNVI. .FCNVI.	2-8
.FCNVL. .FCNVL.	2-8
.FCNVR. .FCNVR.	2-8
.FCXP1 CALL .FCXP1	4-19
.FCXP2 CALL .FCXP2	4-18
.FCXP3 CALL .FCXP3	4-18
.FDXP1 CALL .FDXP1	4-14
.FDXP2 CALL .FDXP2	4-21
.FDXP3 CALL .FDXP3	4-21
.FEFT. CALL .FEFT.	2-34

.FEOF. CALL .FEOF.	2-36
.FEXIT CALL .FEXIT	2-50
.FGTFB CALL .FGTFB	2-37
.FLTPR CALL .FLTPR	2-31
.FOPEN CALL .FOPEN	2-37
.FRWT. CALL .FRWT.	2-34
.FSLEW CALL .FSLEW	2-39
.FXEM. CALL .FXEM.	3-14
.FXP1 CALL .FXP1	4-12
.FXP2 CALL .FXP2	4-14
.FXP3 CALL .FXP3	4-16
.FXP4 CALL .FXP4	4-21
.FXP5 CALL .FXP5	4-21
.FXP6 CALL .FXP6	4-21
.FXSW1 .FXSW1	3-13
.FXSW2 .FXSW2	3-14
.FXSW3 .FXSW3	3-14
.RCOV CALL .RCOV	3-6
.SETU. .SETU.	2-28
ABSOLUTE Complex Absolute Value	4-10
ACCESS Access a Permanent File	3-28
AFT Close File, Detach Buffers, Remove from AFT	3-30

ALOG		
CALL ALOG		4-39
ALOG10		
CALL ALOG10		4-39
ARCOS		
CALL ARCOS		4-60
ARCTANGENT		
Double Precision Arctangent		4-31
Real Arctangent		4-42
ARITHMETIC		
Arithmetic Fault Processor		2-31
ARRAY		
Sort Array of Data		5-4
ARSIN		
CALL ARSIN		4-60
ASCII/BCD		
ASCII/BCD Indicators		3-7
ASSIGNMENT		
Character String Assignment		2-23
ATAN		
CALL ATAN		4-42
ATAN2		
CALL ATAN2		4-42
ATTACH		
Attach a Temporary Mass Storage or Terminal File		3-31
CALL ATTACH		3-28
BACKSPACE		
Backspace Record		2-32
File Forwardspace and Backspace		2-51
BINARY		
Linked Binary Input/Output Interface		2-4
Random Binary Input/Output		2-10
Specify Record Size, Random Binary File		2-54
BLOCK		
File Control Block		2-44
BUFFERS		
Close File, Detach Buffers, Remove from AFT		3-30
CABS		
CALL CABS		4-10
CALL		
CALL .FBST.		2-32
CALL .FCFDP		4-7
CALL .FCFMP		4-7
CALL .FCXP1		4-19
CALL .FCXP2		4-18
CALL .FCXP3		4-18
CALL .FDXP1		4-14
CALL .FDXP2		4-21
CALL .FDXP3		4-21

CALL .FEFT.	2-34
CALL .FEOF.	2-36
CALL .FEXIT	2-50
CALL .FGTFB	2-37
CALL .FLTPR	2-31
CALL .FOPEN	2-37
CALL .FRWT.	2-34
CALL .FSLEW	2-39
CALL .FXEM.	3-14
CALL .FXP1	4-12
CALL .FXP2	4-14
CALL .FXP3	4-16
CALL .FXP4	4-21
CALL .FXP5	4-21
CALL .FXP6	4-21
CALL .RCOV	3-6
CALL ALOG	4-39
CALL ALOG10	4-39
CALL ARCOS	4-60
CALL ARSIN	4-60
CALL ATAN	4-42
CALL ATAN2	4-42
CALL ATTACH	3-28
CALL CABS	4-10
CALL CALLSS	2-52
CALL CCOS	4-58
CALL CEXP	4-52
CALL CLOG	4-54
CALL CONCAT	5-3
CALL CORFL	5-5
CALL CORSEC	5-6
CALL COS	4-45
CALL CREATE	3-31
CALL CSIN	4-58
CALL CSQRT	4-56
CALL DATAN	4-31
CALL DATAN2	4-31
CALL DATIM	3-27
CALL DCOS	4-28
CALL DEFIL	2-53
CALL DETACH	3-30
CALL DEXP	4-24
CALL DLOG	4-34
CALL DLOG10	4-34
CALL DMOD	4-5
CALL DSIN	4-28
CALL DSQRT	4-26
CALL DUMP	2-43
CALL DUMP	2-21
CALL DUMPA	2-21
CALL DUMPA	2-43
CALL DVCHK	3-8
CALL EXIT	2-50
CALL EXP	4-37
CALL FCLOSE	2-47
CALL FILBSP	2-51
CALL FILFSP	2-51
CALL FLGEOF	2-48
CALL FLGERR	2-49
CALL FLGFRC	2-37
CALL FLGRC	5-10
CALL FMEDIA	3-25
CALL FPARAM	2-55
CALL FXALT	3-15
CALL FXEM	3-16
CALL FXOPT	3-15

CALL LINK	3-11
CALL LLINK	3-11
CALL MEMSIZ	5-5
CALL OVERFL	3-8
CALL PDUMP	2-21
CALL PDUMP	2-43
CALL PDUMPA	2-43
CALL PDUMPA	2-21
CALL PTIME	5-6
CALL RANSIZ	2-10
CALL RANSIZ	2-54
CALL SETBUF	2-45
CALL SETFCB	2-44
CALL SETLGT	2-46
CALL SIN	4-45
CALL SLITE	3-9
CALL SLITET	3-9
CALL SORT/SORTD	5-4
CALL SQRT	4-50
CALL SSWTCH	3-10
CALL TANH	4-48
CALL TERMNO	5-5
CALL TERMTM	5-6
CALL USRCOD	5-6
Call TSS Subsystem	2-52
CALLGT	
CALLGT	2-52
CALLSS	
CALL CALLSS	2-52
CARRIAGE	
Carriage Control Simulator	2-39
CCOS	
CALL CCOS	4-58
CEXP	
CALL CEXP	4-52
CHARACTER	
Character String Assignment	2-23
Character String Compare	2-24
Character String Manipulation	5-3
CLOG	
CALL CLOG	4-54
CLOSE	
Close File, Detach Buffers, Remove from AFT	3-30
COMMON	
Double Precision Natural and Common Logarithms	4-34
Natural and Common Logarithms	4-39
COMPARE	
Character String Compare	2-24

COMPLEX	
Complex Absolute Value	4-10
Complex Base and Double Precision Exponent	4-18
Complex Base and Exponent	4-18
Complex Base and Real Exponent	4-18
Complex Exponential	4-52
Complex Multiplication and Division	4-7
Complex Natural Logarithm	4-54
Complex Sine and Cosine	4-58
Complex Square Root	4-56
Exponentiation - Complex Base, Integer Exponent	4-19
CONCAT	
CALL CONCAT	5-3
CONSOLE	
Console Communication	2-41
CORFL	
CALL CORFL	5-5
CORSEC	
CALL CORSEC	5-6
COS	
CALL COS	4-45
COSINE	
Complex Sine and Cosine	4-58
Double Precision Sine and Cosine	4-28
Real Sine and Cosine	4-45
CREATE	
CALL CREATE	3-31
Create TSS Temporary File	2-53
CSIN	
CALL CSIN	4-58
CSQRT	
CALL CSQRT	4-56
DATA-ERROR	
Initialization of Data-Error Processing	2-49
DATAN	
CALL DATAN	4-31
DATAN2	
CALL DATAN2	4-31
DATE	
Date and Time	3-27
DATIM	
CALL DATIM	-27
DCOS	
CALL DCOS	4-28
DEBUG	
Object Time Debug Processor	2-27
DEFIL	
CALL DEFIL	2-53

DETACH		
CALL DETACH		3-30
Close File, Detach Buffers, Remove from AFT		3-30
DEXP		
CALL DEXP		4-24
DIVIDE		
Exponent Register Overflow and Divide Check Tests		3-8
DIVISION		
Complex Multiplication and Division		4-7
DLOG		
CALL DLOG		4-34
DLOG10		
CALL DLOG10		4-34
DMOD		
CALL DMOD		4-5
DOUBLE		
Complex Base and Double Precision Exponent		4-18
Double Precision Arctangent		4-31
Double Precision Base and Exponent		4-21
Double Precision Base and Real Exponent		4-21
Double Precision Exponential		4-24
Double Precision Modulus		4-5
Double Precision Natural and Common Logarithms		4-34
Double Precision Powers of Ten Table		3-3
Double Precision Sine and Cosine		4-28
Double Precision Square Root		4-26
Integer Base and Real or Double Precision Exponent		4-21
Real Base and Double Precision Exponent		4-21
DSIN		
CALL DSIN		4-28
DSQRT		
CALL DSQRT		4-26
DUMP		
CALL DUMP		2-43
CALL DUMP		2-21
Memory Dump		2-43
Namelist and Dump Output		2-21
DUMPA		
CALL DUMPA		2-21
CALL DUMPA		2-43
DVCHK		
CALL DVCHK		3-8
END-OF-FILE		
End-of-File (On Input) Processor		2-36
Initialization of End-of-File Processing		2-48
ENDFILE		
Rewind and Endfile Processor		2-34
ERROR		
Execution Error Monitor		3-13
I/O Error Recovery		5-10

EXIT		
CALL EXIT		2-50
EXP		
CALL EXP		4-37
EXPONENT		
Complex Base and Double Precision Exponent		4-18
Complex Base and Exponent		4-18
Complex Base and Real Exponent		4-18
Double Precision Base and Exponent		4-21
Double Precision Base and Real Exponent		4-21
Exponent Register Overflow and Divide Check Tests		3-8
Exponentiation - Complex Base, Integer Exponent		4-19
Exponentiation - Floating-Point Base Integer Exponent		4-14
Exponentiation - Integer Base and Exponent		4-12
Exponentiation - Real Base and Exponent		4-16
Integer Base and Real or Double Precision Exponent		4-21
Real Base and Double Precision Exponent		4-21
EXPONENTIAL		
Complex Exponential		4-52
Double Precision Exponential		4-24
Natural Exponential		4-37
EXPONENTIATION		
Exponentiation - Complex Base, Integer Exponent		4-19
Exponentiation - Floating-Point Base Integer Exponent		4-14
Exponentiation - Integer Base and Exponent		4-12
Exponentiation - Real Base and Exponent		4-16
F1XP		
F1XP		4-12
F2XP		
F2XP		4-14
F3XP		
F3XP		4-16
F4XP		
F4XP		4-18
FALG		
FALG		4-39
FASC		
FASC		3-7
FASN		
FASN		4-60
FATN		
FATN		4-42
FAULT		
Arithmetic Fault Processor		2-31
FBCD		
FBCD		3-7
FBLO		
FBLO		2-8
FBLO		2-4

FBST		
FBST		2-32
FCAB		
FCAB		4-10
FCAL		
FCAL		2-52
FCAT		
FCAT		5-3
FCEX		
FCEX		4-52
FCHA		
FCHA		2-23
FCLG		
FCLG		4-54
FCLO		
FCLO		2-47
FCLOSE		
CALL FCLOSE		2-47
FCMP		
FCMP		4-7
FCOM		
FCOM		2-24
FCRA		
FCRA		3-31
FCSL		
FCSL		2-41
FCSN		
FCSN		4-58
FCSQ		
FCSQ		4-56
FCXP		
FCXP		4-19
FDAT		
FDAT		4-31
FDBG		
FDBG		2-27
FDEF		
FDEF		2-53
FDEX		
FDEX		4-24
FDIO		
FDIO		2-12
FDLG		
FDLG		4-34
FDMD		
FDMD		4-5

FDMP		
FDMP		2-43
FDPT		
FDPT		3-3
FDSN		
FDSN		4-28
FDSQ		
FDSQ		4-26
FDTH		
FDTH		3-30
FDTM		
FDTM		3-27
FDXP		
FDXP		4-21
FEFT		
FEFT		2-34
FEOF		
FEOF		2-36
FEXP		
FEXP		4-37
FFEE		
FFEE		2-48
FFER		
FFER		2-49
FFER		5-10
FFFB		
FFFB		2-51
FFLT		
FFLT		2-31
FIFA		
FIFA		2-17
FILBSP		
CALL FILBSP		2-51
FILE		
Access a Permanent File		3-28
Attach a Temporary Mass Storage or Terminal File		3-31
Close File, Detach Buffers, Remove from AFT		3-30
Create TSS Temporary File		2-53
File Closing		2-47
File Control Block		2-44
File Forwardspace and Backspace		2-51
File Opening		2-37
File Transliteration		3-25
Specify Record Size, Random Binary File		2-54
FILFSP		
CALL FILFSP		2-51
FLAT		
FLAT		5-9

FLGEOF CALL FLGEOF	2-48
FLGERR CALL FLGERR	2-49
FLGFRC CALL FLGFRC	2-37
FLGRC CALL FLGRC	5-10
FLHS FLHS	3-4
FLIT FLIT	3-9
FLNK FLNK	3-11
FLOATING-POINT Exponentiation - Floating-Point Base Integer Exponent	4-14
FMDE FMDE	5-2
FMED FMED	3-25
FMEDIA CALL FMEDIA	3-25
FNLI FNLI	2-17
FNLO FNLO	2-21
FOFA FOFA	2-21
FOPE FOPE	2-37
FORMAT Format Controlled Sequential Input/Output	2-12
FORTRAN Mode Determination by FORTRAN	5-2
FORWARSPACE File Forwardspace and Backspace	2-51
FPARAM CALL FPARAM	2-55
FPAW FPAW	2-25
FRBC FRBC	3-6
FRCV FRCV	3-6
FRDA FRDA	2-12

FRDB		
FRDB		2-4
FRDD		
FRDD		2-12
FRND		
FRND		5-9
FRRD		
FRRD		2-8
FRRD		2-10
FRRD		2-54
FRSW		
FRSW		5-7
FSET		
FSET		2-44
FSHF		
FSHF		5-8
FSIN		
FSIN		4-45
FSLI		
FSLI		2-8
FSLI		2-12
FSLO		
FSLO		2-12
FSLO		2-8
FSLW		
FSLW		2-39
FSQR		
FSQR		4-50
FSRT		
FSRT		5-4
FSTU		
FSTU		2-55
FSTU		2-28
FSWI		
FSWI		3-10
FTAC		
FTAC		3-28
FTGF		
FTGF		2-55
FTLK		
FTLK		3-12
FTNH		
FTNH		4-48
FTSF		
FTSF		5-5
FTSG		
FTSG		5-6

FTSU		2-30
FTSU		
FVFI		2-17
FVFI		
FVFO		2-21
FVFO		
FXALT		3-15
CALL FXALT		
FXEM		3-16
CALL FXEM		3-13
FXEM		
FXER		3-13
FXER		
FXIT		2-50
FXIT		
FXMA		3-13
FXMA		
FXOPT		3-15
CALL FXOPT		
GENERATOR		
Random Number Generator		5-9
H*		
Restore Link - H*		3-4
HYPERBOLIC		
Real Hyperbolic Tangent		4-48
I/O		
I/O Error Recovery		5-10
I/O Parameters of Run-Time Library		2-55
IDENTIFICATION		
User Time And Identification		5-6
ILR		
ILR		5-8
ILS		
ILS		5-8
INDICATORS		
ASCII/BCD Indicators		3-7
INITIALIZER		
Pre-execution Initializer		2-30
Pre-execution Initializer		2-28
INPUT		
End-of-File (On Input) Processor		2-36
Namelist Input		2-17
Terminal Input Recovery		3-6

INPUT/OUTPUT	
Format Controlled Sequential Input/Output	2-12
INPUT/OUTPUT LIBRARY	2-1
INPUT/OUTPUT LIBRARY SUBROUTINES	2-3
Linked Binary Input/Output Interface	2-4
Random Binary Input/Output	2-10
Short List Input/Output Processor	2-8
INTEGER	
Exponentiation - Complex Base, Integer Exponent	4-19
Exponentiation - Floating-Point Base Integer Exponent	4-14
Exponentiation - Integer Base and Exponent	4-12
Integer Base and Real or Double Precision Exponent	4-21
INVERSE	
Inverse Sine/Cosine	4-60
IRETSW	
IRETSW	5-7
IRS	
IRS	5-8
ISETSW	
ISETSW	5-7
LIBRARY	
I/O Parameters of Run-Time Library	2-55
INPUT/OUTPUT LIBRARY	2-1
INPUT/OUTPUT LIBRARY SUBROUTINES	2-3
LIBRARY CALLS	2-1
MATHEMATICAL LIBRARY	4-1
LINK	
CALL LINK	3-11
Restore Link - H*	3-4
Linked Binary Input/Output Interface	2-4
Restore Links During Execution (Batch)	3-11
Restore Links During Execution (Time Sharing)	3-12
LIST	
Short List Input/Output Processor	2-8
LLINK	
CALL LLINK	3-11
LOGARITHM	
Complex Natural Logarithm	4-54
Double Precision Natural and Common Logarithms	4-34
Natural and Common Logarithms	4-39
LOGICAL	
DEFINE LOGICAL UNIT TABLE	2-46
Logical Unit Table Routines	2-44
MANIPULATION	
Character String Manipulation	5-3
MASS	
Attach a Temporary Mass Storage or Terminal File	3-31
MATHEMATICAL	
MATHEMATICAL LIBRARY	4-1

MEMORY		
Memory Dump		2-43
Memory Usage		5-5
MEMSIZ		
CALL MEMSIZ		5-5
MODE		
MODE		5-2
Mode Determination by FORTRAN		5-2
MODULUS		
Double Precision Modulus		4-5
MONITOR		
Execution Error Monitor		3-13
MULTIPLICATION		
Complex Multiplication and Division		4-7
NAMELIST		
Namelist and Dump Output		2-21
Namelist Input		2-17
NATURAL		
Complex Natural Logarithm		4-54
Double Precision Natural and Common Logarithms		4-34
Natural and Common Logarithms		4-39
Natural Exponential		4-37
NUMBER		
Random Number Generator		5-9
OBJECT		
Object Time Debug Processor		2-27
OUTPUT		
Namelist and Dump Output		2-21
Output Stop and Pause Information		2-25
OVERFL		
CALL OVERFL		3-8
OVERFLOW		
Exponent Register Overflow and Divide Check Tests		3-8
PARAMETERS		
I/O Parameters of Run-Time Library		2-55
PAUSE		
Output Stop and Pause Information		2-25
PDUMP		
CALL PDUMP		2-43
CALL PDUMP		2-21
PDUMPA		
CALL PDUMPA		2-21
CALL PDUMPA		2-43
PERMANENT		
Access a Permanent File		3-28

PRE-EXECUTION		
Pre-execution Initializer		2-28
Pre-execution Initializer		2-30
PRECISION		
Complex Base and Double Precision Exponent		4-18
Double Precision Arctangent		4-31
Double Precision Base and Exponent		4-21
Double Precision Base and Real Exponent		4-21
Double Precision Exponential		4-24
Double Precision Modulus		4-5
Double Precision Natural and Common Logarithms		4-34
Double Precision Powers of Ten Table		3-3
Double Precision Sine and Cosine		4-28
Double Precision Square Root		4-26
Integer Base and Real or Double Precision Exponent		4-21
Real Base and Double Precision Exponent		4-21
PROCESSING		
Initialization of Data-Error Processing		2-49
Initialization of End-of-File Processing		2-48
PTIME		
CALL PTIME		5-6
RAND		
RAND		5-9
RANDOM		
Random Binary Input/Output		2-10
Random Number Generator		5-9
Specify Record Size, Random Binary File		2-54
RANDOM RECORD SIZE		
Random Record Size		2-10
RANDT		
RANDT		5-9
RANSIZ		
CALL RANSIZ		2-10
CALL RANSIZ		2-54
REAL		
Complex Base and Real Exponent		4-18
Double Precision Base and Real Exponent		4-21
Exponentiation - Real Base and Exponent		4-16
Integer Base and Real or Double Precision Exponent		4-21
Real Arctangent		4-42
Real Base and Double Precision Exponent		4-21
Real Hyperbolic Tangent		4-48
Real Sine and Cosine		4-45
Real Square Root		4-50
RECORD		
Backspace Record		2-32
Specify Record Size, Random Binary File		2-54
RECOVERY		
I/O Error Recovery		5-10
Terminal Input Recovery		3-6
REGISTER		
Exponent Register Overflow and Divide Check Tests		3-8
REMOVE		
Close File, Detach Buffers, Remove from AFT		3-30

RESTORE		
Restore Link - H*		3-4
Restore Links During Execution (Batch)		3-11
Restore Links During Execution (Time Sharing)		3-12
REWIND		
Rewind and Endfile Processor		2-34
RUN-TIME		
I/O Parameters of Run-Time Library		2-55
SENSE		
Sense Light Simulator		3-9
Sense Switch Test		3-10
SEQUENTIAL		
Format Controlled Sequential Input/Output		2-12
SET/RESET		
Set/Reset Switch Word		5-7
SETBUF		
CALL SETBUF		2-45
SETFCB		
CALL SETFCB		2-44
SETLGT		
CALL SETLGT		2-46
SHIFT/ROTATE		
Shift/Rotate Word Contents		5-8
SIN		
CALL SIN		4-45
SINE		
Complex Sine and Cosine		4-58
Double Precision Sine and Cosine		4-28
Real Sine and Cosine		4-45
SINE/COSINE		
Inverse Sine/Cosine		4-60
SIZE		
Specify Record Size, Random Binary File		2-54
SLITE		
CALL SLITE		3-9
SLITET		
CALL SLITET		3-9
SORT		
Sort Array of Data		5-4
SORT/SORTD		
CALL SORT/SORTD		5-4
SPECIFY		
Specify Record Size, Random Binary File		2-54
SQRT		
CALL SQRT		4-50

SQUARE ROOT	
Complex Square Root	4-56
Double Precision Square Root	4-26
Real Square Root	4-50
SSWTCH	
CALL SSWTCH	3-10
STOP	
Output Stop and Pause Information	2-25
STORAGE	
Attach a Temporary Mass Storage or Terminal File	3-31
STRING	
Character String Assignment	2-23
Character String Compare	2-24
Character String Manipulation	5-3
SUBSYSTEM	
Call TSS Subsystem	2-52
TANGENT	
Real Hyperbolic Tangent	4-48
TANH	
CALL TANH	4-48
TEMPORARY	
Attach a Temporary Mass Storage or Terminal File	3-31
Create TSS Temporary File	2-53
TERMINAL	
Attach a Temporary Mass Storage or Terminal File	3-31
Terminal Input Recovery	3-6
TERMINATION	
Job Termination	2-50
TERMNO	
CALL TERMNO	5-5
TERMTM	
CALL TERMTM	5-6
TRANSLITERATION	
File Transliteration	3-25
TSS	
Call TSS Subsystem	2-52
Create TSS Temporary File	2-53
UNIFM2	
UNIFM2	5-9
USER	
User Time And Identification	5-6
USRCOD	
CALL USRCOD	5-6
VALUE	
Complex Absolute Value	4-10

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE

SERIES 60 (LEVEL 66) / 6000 FORTRAN SUBROUTINES
LIBRARIES, ADDENDUM A

ORDER NO.

DD20A, REV. 0

DATED

MARCH 1977

ERRORS IN PUBLICATION

[Empty box for reporting errors in publication]

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

[Empty box for providing suggestions for improvement to publication]



Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below.

FROM: NAME _____

DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

PLEASE FOLD AND TAPE —

NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 395
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell