DOS/RTE Relocatable Library

Reference Manual

HEWLETT
PACKARD

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 24998-90001
Printed in U.S.A. October 1981

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Fifth Edition	Dec 1978	
Update 1	Jul 1979	
Update 2	Apr 1980	
Update 3	Jul 1980	
Reprint	Jul 1980	Updates 1 thru 3 incorporated
Update 4	Jul 1981	Manual enhancement
Update 5	Oct 1981	
Reprint	Oct 1981	Updates 4 and 5 incorporated

PREFACE

This manual is a programmer's guide to subroutines contained in HP 1000 Operating Systems. For the RTE-IVB Real-Time Executive Operating System (Product Number 92068A), this manual covers the following libraries:

Library Mnemonic	Library Name	Part Number
MLIB1	Math/Formatter Library, Part 1	24998-12001
MLIB2	Math/Formatter Library, Part 2	24998-12002
$YSLB	RTE-IVB System Library (Also see RTE-IVB Programmer's Reference Manual part number 92068-90004)	92067-16268

For the following operating systems:

Product Number	Operating System Name
24307B/C	DOS-III Disc Operating System
92001B	RTE-II Real-Time Executive Operating System
92060B	RTE-III Real-Time Executive Operating System
92067A	RTE-IV Real-Time Executive Operating System
92064A	RTE-M Real-Time Executive Operating System

this manual covers the following libraries:

Library Mnemonic	Library Name	Product or Part Number
RLIB.N	DOS/RTE Relocatable Library	24998-16001
FF4.N	FORTRAN IV Formatter	24998-16002
FF.N	FORTRAN Formatter	24153

Section I of this manual introduces the libraries, describes the order in which they should be generated into your operating system, and explains the format this manual uses to describe the individual subroutines. Sections II and III are, respectively, alphabetical groupings of the DOS/RTE Relocatable Library mathematical and utility subroutines. Section IV is a discussion of the FORTRAN IV Formatter and the FORTRAN Formatter.

Appendix A is a list of error messages for all subroutines that generate error messages.

Appendix B is a description of how to use the RTE DEBUG Library Subroutine.

Three indexes are included, to help you find the subroutines you need: Index 1 is a list of all entry points to the DOS/RTE Relocatable Subroutines; and Index 2 is a list of subroutines by function.

There are several other relocatable libraries currently distributed with your DOS or RTE operating system. The following table identifies them, and directs you to the documentation which describes their use:

Library Mnemonic	Library Name	Library Part or Product Number	Related Manual (and Part Number)
FLIB.N	Floating Point Library (DOS III only)	24998-16001	DOS III Disc Operating System (24307-90006)
FFP.N	2100 FFP Subroutine Library	12907-16001	Implementing 2100 FFP (12907-90010)
$SETP	2100 FFP $SETP System Subroutine (DOS-III only)	12907-16002	Implementing 2100 FFP (12907-90010)
FPM.N	21MX.FFP Subroutine Library	24998-16008	12977A FFP Installation and Programming Manual (12977-90001)
$SETP	21MX FFP $SETP System Subroutine (DOS-III only)	12977-16002	12977A FFP Installation and Programming Manual (12977-90001)
na	7210A Plotter Library (RTE only)	92409-60001	Utility Subroutines for 7210A X-Y Plotter (92409-93001)
na	CalComp Plotter Library	20810	12360A Digital Plotter Interface Kit (12560-9001)

CONTENTS

SECTION I

INTRODUCING THE LIBRARIES

INTRODUCING THE LIBRARIES

The libraries of relocatable subroutines distributed with your operating system have two functions:

1. The libraries provide you with tested and supported subroutines that save you programming time. These subroutines can be called from your Assembly language, FORTRAN, or ALGOL application programs.
2. The libraries contain subroutines used by the operating system to perform its functions. Therefore the libraries are required to generate the operating system.

USING THE LIBRARIES IN A DISC-BASED OPERATING SYSTEM

When you generate your disc-based operating system, you must include the proper relocatable libraries in your system, and they must be included in a definite order. Follow the flowchart in Figure 1-1 (for DOS-III) or in Figure 1-2 (for RTE) for the correct entry order. The libraries are included during the program input phase of system generation.

When an operating system module or one of your application programs executes a call to one of the library subroutines, the Relocating Loader ensures that the correct linkages are made between the calling routine and the proper subroutine.

For a complete discussion of program input order, refer to the system generation instructions in the following manuals:

Product No.	Manual Title	Manual Part No.
92068A	RTE-IVB System Manager's Manual	92068-90006
92067A	RTE-IV Programming and Operating Manual	92067-90001
92060A/B	RTE-III Programming and Operating Manual	92060-90004
92001A/B	RTE-II Programming and Operating Manual	92001-93001
24307B/C	DOS-III Disc Operating System Reference Manual	24307-90006

USING THE LIBRARIES IN A MEMORY-BASED OPERATING SYSTEM

When one of your application programs executes a call to one of the library subroutines, the generator or relocating loader ensures that the correct linkages are made between the calling routine and the proper subroutine. For each program, you must direct the generator or relocating loader to search the file (or device) containing the required subroutines as described in the following manuals:

Product No.	Manual Title	Manual Part No.
92064A	RTE-M Programmer's Reference Manual	92064-90002
	RTE-M System Generation Reference Manual	92064-90003

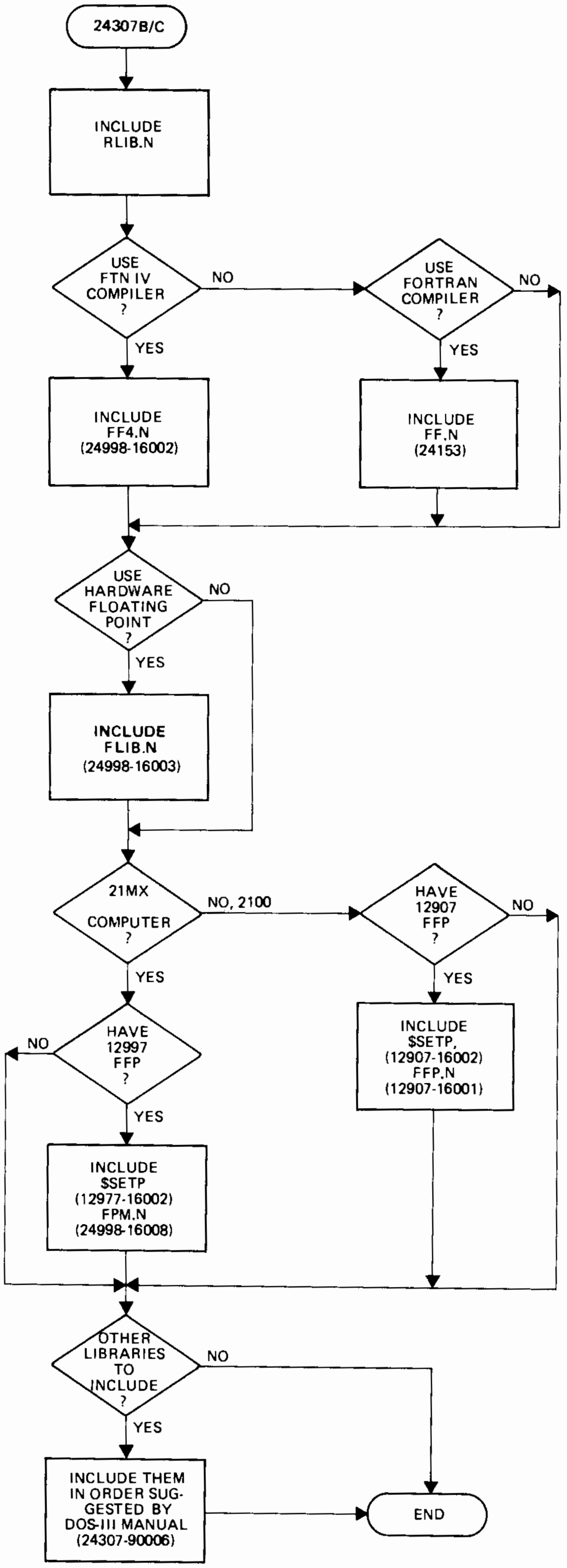


Figure 1-1 DOS-III Library Selection

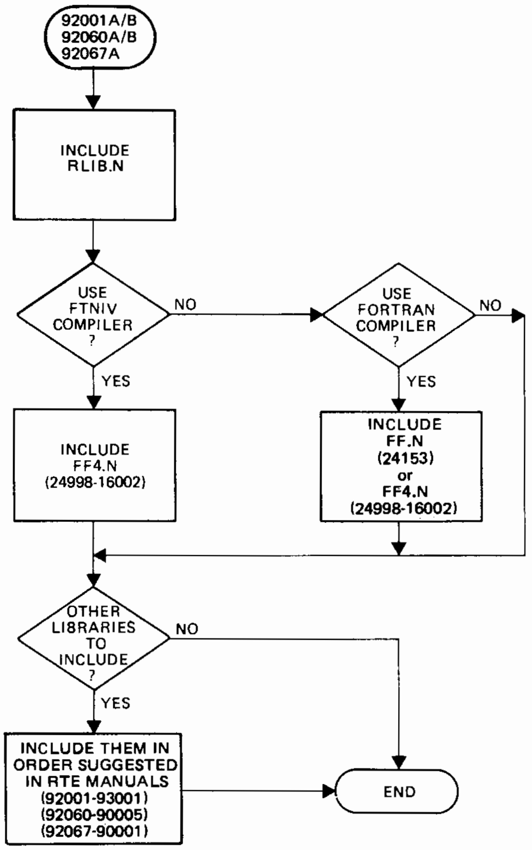
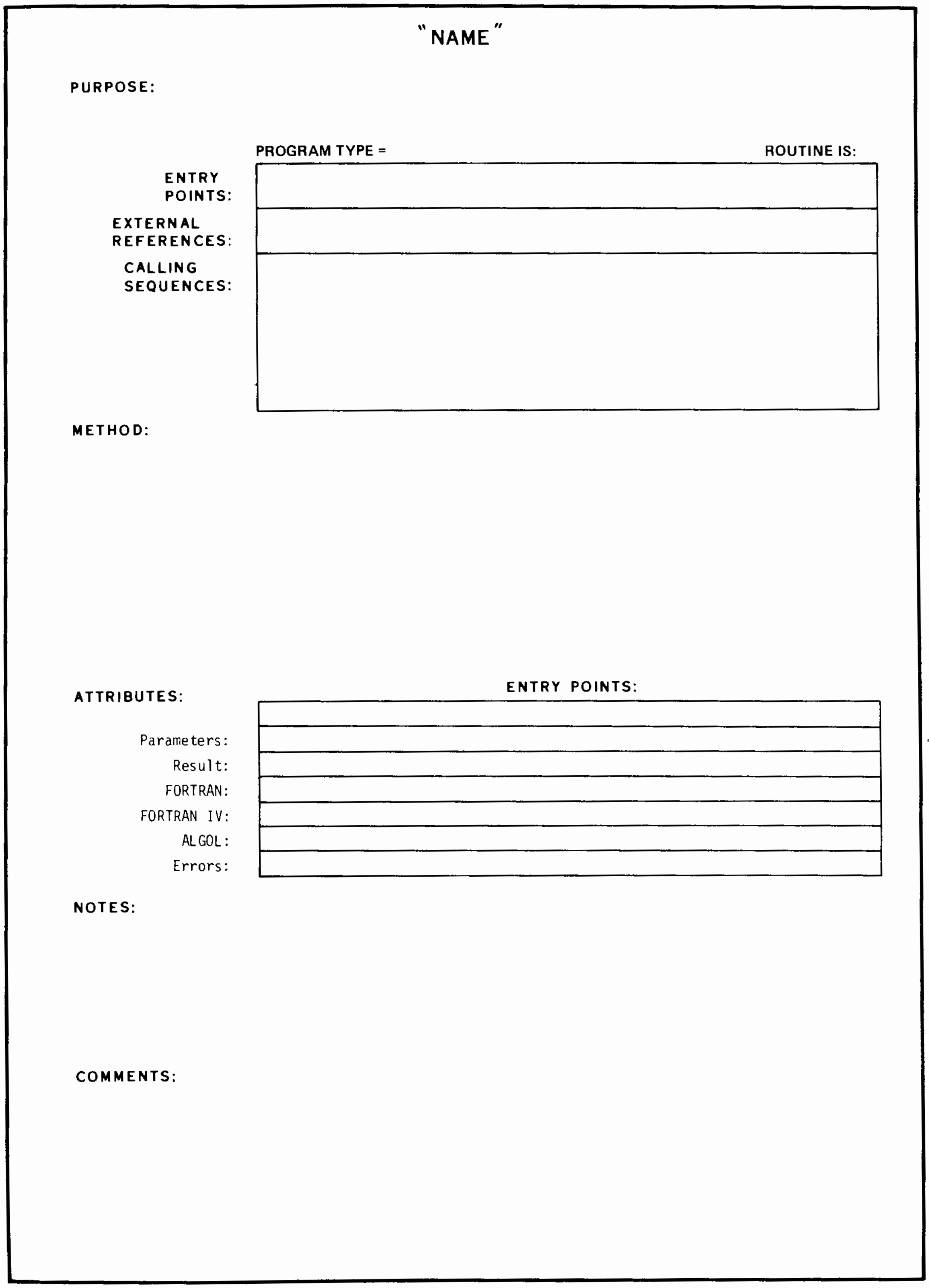


Figure 1-2. RTE Disc-Resident Library Selection

HOW SUBROUTINES ARE PRESENTED

In this manual, the subroutines in each section are presented one to a page, in alphabetic order. Study the sample page format, Figure 1-3, and the following information to optimize your use of this manual.



"NAME"

PURPOSE:

PROGRAM TYPE =

ROUTINE IS:

ENTRY POINTS:

EXTERNAL REFERENCES:

CALLING SEQUENCES:

METHOD:

ATTRIBUTES:

ENTRY POINTS:

Parameters:

Result:

FORTRAN:

FORTRAN IV:

ALGOL:

Errors:

NOTES:

COMMENTS:

Figure 1-3. Sample Page Format

"NAME"	The name of the routine record in the NAM record.
Purpose	The use of the routine.
Program Type =	Will be either 6 or 7
Routine is:	Will be P for Privileged, R for Reentrant, or U for Utility.
Entry Points	The entry points to the routine.
External References	These are other subroutines that are called by the subroutine. All external references except EXEC, $OPSY, REIO, IFBRK, .ZPRV, and .ZRNT are entry points in RLIB. EXEC and $OPSY are system entry points. IFBRK & REIO are system library entry points. These symbols receive special handling by the DOS and RTE generators and loaders. In DOS, both JSB .ZPRV and JSB .ZRNT are always changed to RSS. In RTE, both JSB .ZPRV and JSB .ZRNT are changed to RSS unless the routine is generated into the resident library. If the routine is in the resident library, the generator modifies its code as follows:

```
ENTRY  NOP          → ENTRY NOP
       JSB .ZPRV    →       JSB $LIBR
       DEF EXIT     →       NOP
        :                    :
EXIT   JMP ENTRY,I  → EXIT  JSB $LIBX
       DEF ENTRY    →       DEF ENTRY
ENTRY  NOP          → ENTRY NOP
       JSB .ZRNT    →       JSB $LIBR
       DEF EXIT     →       DEF TDB
        :                    :
EXIT   JMP ENTRY,I  → EXIT  JSB $LIBX
       DEF TDB      →       DEF TDB
       DEC 0        →       DEC 0
```

	$LIBR and $LIBX are system entry points that allow multiple RTE programs to share code.
Calling Sequences	This is the assembly language calling sequence for each entry point. The arrow (→) indicates a return point. "A" and "B" indicate the A- and B- registers.
Method	This gives the algorithm for producing the result and/or the accuracy of the routine.
Attribute Chart	For each entry point, this chart gives the following information: a. Parameters: their type (real, integer, double real or complex) and whether they are loaded into the A- and B- registers. b. Result: the type of the result and the registers (if any) where it is returned.

c. Fortran: whether the routine is callable as a function (e.g., ABS(*x*)), callable as a subroutine (e.g., CALL RMPAR (TBUF)) or uncallable in HP FORTRAN.

d. FORTRAN IV: whether the routine is callable as a function (e.g., ABS(x)), callable as a subroutine (e.g., CALL RMPAR (IBUF)), or uncallable in HP FORTRAN IV.

e. ALGOL: whether the routine is an intrinsic, callable or uncallable as a procedure in HP ALGOL.

f. ERRORS: This gives a summary of the error conditions reported by the subroutine. Errors generated by external references are not described. See Appendix A for a fuller discussion of error messages.

MICROCODED SUBROUTINES

Fast Fortran Processor

The HP 2100 and 21MX computers have, as an option, a Fast FORTRAN Processor (FFP). The HP 12907 FFP is optional for the HP 2100 computers and the HP 12977 FFP is optional for the HP 21MX computers. The FFP firmware feature provides for faster execution of the following routines:

.GOTO ..MAP .ENTR .ENTP DBLE
SNGL .XMPY .XDIV .DFER .XFER
.XADD .XSUB $SETP

The following additional Relocatable Subroutine entry points are available only in HP 12977 FFP:

.PWR2 .XPAK .FLUN .XCOM .PACK ..DCM DDINT

No change from the calling sequence defined in this manual is required to use these routines in FFP, if installed. The user should be aware that after the first execution of a subroutine call, "JSB .GOTO" for example, the main memory location containing the JSB is modified to hold a branch to the ROM address where the .GOTO microcode begins.

Floating Point Library

A second microcode option on the HP 2100 computer, HP 12901 Floating Point, provides firmware for faster execution of the following routines:

.FAD .FSB .FMP .FDV FLOAT IFIX

These routines are implemented in the same fashion as the FFP routines. (The HP 21MX computers include the floating point firmware as part of the basic instruction set.)

SECTION II

MATHEMATICAL SUBROUTINES

ABS

PURPOSE: Calculate the absolute value of a real *x*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	ABS
EXTERNAL REFERENCES:	..FCM,.ZPRV
CALLING SEQUENCES:	DLD *x* JSB ABS → result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	ABS
Parameters:	Real: A & B
Result:	Real: A & B
FORTRAN:	Function: ABS (*x*)
FORTRAN IV:	Function: ABS (*x*)
ALGOL:	Intrinsic: ABS (*x*)
Errors:	None

AIMAG

PURPOSE: Extract the imaginary part of a complex *x*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: AIMAG

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB AIMAG
DEF *+2
DEF x
→ result in A & B
```

ATTRIBUTES:

	ENTRY POINTS:
	AIMAG
Parameters:	Complex
Result:	Real: A & B
FORTRAN:	Callable as function
FORTRAN IV:	Function: AIMAG (*x*)
ALGOL:	Callable as real procedure
Errors:	None

AINT

PURPOSE: Truncate a real x.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	AINT
EXTERNAL REFERENCES:	.FAD .ZPRV
CALLING SEQUENCES:	DLD x JSB AINT → y in A & B

METHOD: y = largest integer $\leq|x|$

ATTRIBUTES:

ENTRY POINTS:

	AINT
Parameters:	Real: A&B
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Function: AINT (x)
ALGOL:	Not callable
Errors:	None

ALOG

PURPOSE: Calculate the natural logarithm of a real x:

$$y = \ln (x)$$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:

LN
ALOG

EXTERNAL REFERENCES:

.FLUN, FLOAT, .FAD, .FSB, .FDV, .FMP .ZPRV

CALLING SEQUENCES:

```
DLD x
JSB ALOG (or LN)
JSB ERR0 (error return)
→ return (y in A & B)
```

METHOD: The range is reduced to (.707, 1.414) using the identity:

$$\text{ALOG } (x) = \text{Ln } (2) * (\text{N} + \text{Log}_2 \ (\frac{x}{2^\text{N}}))$$

Then the Following Formula is used:

$$\log_2(y) = z * (\text{A} + \frac{\text{B}}{\text{C} + z^2})$$

where

$$y = \frac{x}{2^\text{N}}$$

$$z = \frac{y-1}{y+1}$$

A = 1.29061344
B = 2.6444261
C = -1.6581795

ATTRIBUTES:

ENTRY POINTS:

	ALOG	LN
Parameters:	Real: A & B	Real: A & B
Result:	Real: A & B	Real: A & B
FORTRAN:	Function: ALOG(x)	Not callable
FORTRAN IV:	Function: ALOG (x)	Not callable
ALGOL:	Not callable	Intrinsic Procedure
Errors:	$x < 0$ → (02 UN)	Same

NOTES: ALOG is the FORTRAN entry point; LN is the ALGOL entry point.

ALOGT

PURPOSE: Calculate the common logarithm (base 10) of real x:

$$y = \log_{10} x$$

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	ALOGT ALOG0
EXTERNAL REFERENCES:	ALOG, .FMP
CALLING SEQUENCES:	DLD x JSB ALOGT (or ALOG0) JSB ERRO (error return) → return (y in A&B)

METHOD:

$y = \log_{10} x = \log_{10}e * \log_e x$

Accuracy depends on the accuracy of ALOG.

ATTRIBUTES:

	ENTRY POINTS:
	ALOGT (ALOG0)
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not Callable
FORTRAN IV:	Function: ALOGT (x)
ALGOL:	Not callable
Errors:	If $x \leq 0$ → (02 UN)

AMOD

PURPOSE: Calculate the real remainder of X/Y for real X and Y:

$$Z = X \text{ modulo } Y$$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:

```
AMOD
```

EXTERNAL REFERENCES:

```
.ENTP, .ZPRV
AINT, .FDV, .FMP, .FSB
```

CALLING SEQUENCES:

```
JSB AMOD
DEF * + 3
DEF X
DEF Y
→ Z in A & B
```

METHOD:

$z = x - [\text{AINT}(x/y)]_{*}y$

ATTRIBUTES:

ENTRY POINTS:

	AMOD
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Callable as Function
FORTRAN IV:	Function: AMOD (x,y)
ALGOL:	Callable as Real Procedure
Errors:	If $Y = 0$, then $Z = X$

ATAN

PURPOSE: Calculate the arctangent of a real x:

$$Y = \tan^{-1}(x)$$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:

ARCTA
ATAN

EXTERNAL REFERENCES:

.ZPRV, ..FCM, .FAD,
.FSB, .FDV, .FMP

CALLING SEQUENCES:

```
DLD x
JSB ATAN (or ARCTA)
→ return (Y in A&B)
```

METHOD: x is reduced to the range [-.5, -5] using the identities:

$$\text{ATAN }(x) = -\text{ATAN}(-x) \text{ For } x<0$$

$$\text{ATAN }(x) = \pi/4-\text{ATAN }\left(\frac{1-x}{1+x}\right) \text{ For } .5\leq x<2$$

$$\text{ATAN }(x) = \pi/2-\text{ATAN }\left(\frac{1}{x}\right) \text{ For } x\geq 2$$

Then the Following Formula is used:

$$\text{ATAN}(x) = x/(\text{A}+\text{B}*(x^2 + \text{C}/(\text{D}+x^2)))$$

where A = 1.3504734 C = -4.4369869
B = .15700588 D = 1.9876921

ATTRIBUTES:

ENTRY POINTS:

	ATAN	ARCTA
Parameters:	Real: A & B	Real: A & B
Result:	Real: A & B (radians)	Real: A & B (radians)
FORTRAN:	Function: ATAN (x)	Not callable
FORTRAN IV:	Function: ATAN (x)	Not callable
ALGOL:	Not callable	Intrinsic Function: ARCTAN(x)
Errors:	None	None

NOTES:

1. ATAN is the FORTRAN entry point and ARCTA is the ALGOL entry point.
2. Result ranges from $-\pi/2$ to $\pi/2$.

ATAN2

PURPOSE: Calculate the real arctangent of the quotient of two reals: z = arctan (Y/X)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	ATAN2
EXTERNAL REFERENCES:	.ENTP, SIGN, ATAN, .ZRNT, .FDV, .FAD

CALLING SEQUENCES:

```
JSB ATAN2
DEF * + 3
DEF Y
DEF X
→ z in A & B
```

METHOD:

If $X = 0$, $z = \text{sign}(Y)\ \pi/2$

If $X > 0$, $z = \arctan(Y/X)$

If $X < 0$, $z = \arctan(Y/X) + \text{sign}(Y)\ .\pi$

Accuracy depends on accuracy of ATAN.

ATTRIBUTES:	**ENTRY POINTS:**
	ATAN2
Parameters:	Real
Result:	Real: A & B
FORTRAN:	Callable as Function
FORTRAN IV:	Function: ATAN2 (y,x)
ALGOL:	Callable as Real Procedure
Errors:	None

CABS

PURPOSE: Calculate the real absolute value (modulus) of complex X: $Y = |X|$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: CABS

EXTERNAL REFERENCES: ABS, .FSB, .FAD, .FDV, .FMP, .ENTP, SQRT, .ZRNT,

CALLING SEQUENCES:

```
JSB CABS
DEF *+2
DEF X
→ Y in A & B
```

METHOD:

$$Y = |X| = |X_1 + i * X_2| = \sqrt{X_1^2 + X_2^2} = |X_1| \sqrt{1 + \left(\frac{X_2}{X_1}\right)^2} \text{ for } |X_1| \geq |X_2|, \text{ or}$$

$$= |X_2| \sqrt{\left(\frac{X_1}{X_2}\right)^2 + 1} \text{ for } |X_2| > |X_1|$$

Accuracy depends on the accuracy of SQRT.

ATTRIBUTES:

	ENTRY POINTS:
	CABS
Parameters:	Complex
Result:	Real: A&B
FORTRAN:	Callable as Function
FORTRAN IV:	Function: CABS (x)
ALGOL:	Callable as Real Procedure
Errors:	None

CADD

PURPOSE: Interface routine to allow FORTRAN II program to utilize the FORTRAN IV complex add routine, .CADD.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: CADD

EXTERNAL REFERENCES: .RCNG, .CADD

CALLING SEQUENCES:

```
JSB CADD
DEF * + 4
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:	**ENTRY POINTS:**
	CADD
Parameters:	Complex
Result:	Complex
FORTRAN:	Callable CADD (*z*,*x*,*y*)
FORTRAN IV:	NOT APPLICABLE
ALGOL:	NOT APPLICABLE
Errors:	Overflow bit set if result out of range

Note: See OVF function for testing results

CDIV

PURPOSE: Interface routine which allows FORTRAN II programs to utilize the FORTRAN IV complex divide routine .CDIV.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	CDIV
EXTERNAL REFERENCES:	.RCNG, .CDIV

CALLING SEQUENCES:

```
JSB CDIV
DEF * + 4
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:	**ENTRY POINTS:**
	CDIV
Parameters:	Complex
Result:	Complex
FORTRAN:	Callable CDIV (*z*,*x*,*y*)
FORTRAN IV:	NOT APPLICABLE
ALGOL:	NOT APPLICABLE
Errors:	Overflow bit set if result out of range

Note: See OVF function for testing results

CEXP

PURPOSE: Calculate the complex exponential of a complex *x*.

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: CEXP

EXTERNAL REFERENCES: .ENTP, EXP, .ZRNT
SIN, COS, .FMP

CALLING SEQUENCES:

```
JSB CEXP
DEF *+3
DEF Y (result)
DEF X
→ Error return
→ Normal return
```

METHOD: $Y = Y_1 + i \cdot Y_2 = e^X = e^{(X_1 + i X_2)} = e^{X_1} (\cos X_2 + i \cdot \sin X_2)$

Accuracy: depends on the accuracy of EXP and SIN.

ATTRIBUTES:	**ENTRY POINTS:**
	CEXP
Parameters:	Complex
Result:	Complex
FORTRAN:	Not Callable
FORTRAN IV:	Function: CEXP(*x*)
ALGOL:	Not callable
Errors:	If $x_1 \cdot \log_2 e \geq 124$, →(07 OF). (EXP) If $\frac{1}{2}\left|\frac{X_2}{\pi} + \frac{1}{2}\right| > 2^{14}$→(05 OR). (SIN)

CLOG

PURPOSE: Calculate the complex natural logarithm of a complex *x*.

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	CLOG
EXTERNAL REFERENCES:	.ENTP, ALOG, .ZRNT CABS, ATAN2
CALLING SEQUENCES:	JSB CLOG DEF *+3 DEF *y* (result) DEF *x* → Error return → Normal return

METHOD:

$$y = y_1 + i \cdot y_2 = \log_e x = \log_e (x_1 + i.x_2) = \log_e(r) + i.\Theta$$

where

$$r = \sqrt{x_1^2 + x_2^2}$$

$$\Theta = \arctan\left(\frac{x_2}{x_1}\right)$$

Accuracy depends on the accuracy of ALOG and SQRT.

ATTRIBUTES:

	ENTRY POINTS:
	CLOG
Parameters:	Complex
Result:	Complex
FORTRAN:	Not Callable
FORTRAN IV:	Function: CLOG(*x*)
ALGOL:	Not Callable
Errors:	If *x* = 0 → (02 UN)

CMPLX

PURPOSE: Combine a real *x* and an imaginary *y* into a complex *z*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	CMPLX
EXTERNAL REFERENCES:	.ENTP, .ZPRV

CALLING SEQUENCES:

```
JSB CMPLX
DEF *+4
DEF z
DEF x
DEF y
→
```

ATTRIBUTES:

ENTRY POINTS:

	CMPLX
Parameters:	Real & Real (imaginary part)
Result:	Complex
FORTRAN:	Callable
FORTRAN IV:	Function: CMPLX (*x*,*y*)
ALGOL:	Callable as real procedure
Errors:	None

CMPY

PURPOSE: Interface routine to allow FORTRAN II programs to utilize the FORTRAN IV complex multiply routine, .CMPY.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	CMPY
EXTERNAL REFERENCES:	.RCNG, .CMPY
CALLING SEQUENCES:	JSB CMPY DEF * + 4 DEF *z* (result) DEF *x* DEF *y* →

ATTRIBUTES:

	ENTRY POINTS:
	CMPY
Parameters:	Complex
Result:	Complex
FORTRAN:	CALL CMPY (*z*,*x*,*y*)
FORTRAN IV:	NOT APPLICABLE
ALGOL:	NOT APPLICABLE
Errors:	Overflow bit set if result out of range

Note: See OVF function for testing results

CONJG

PURPOSE: Form the conjugate Y of a complex X.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: CONJG

EXTERNAL REFERENCES: .ENTP ..DLC, .ZPRV

CALLING SEQUENCES:

```
JSB CONJG
DEF * + 3
DEF Y (result)
DEF X
→
```

METHOD: If $X = X_1 + i \cdot X_2$, then $Y = X_1 - i \cdot X_2$

ATTRIBUTES:

	ENTRY POINTS:
	CONJG
Parameters:	Complex
Result:	Complex
FORTRAN:	Callable
FORTRAN IV:	Function: CONJG (x)
ALGOL:	Callable as real procedure
Errors:	None

COS

PURPOSE: See .SNCS

CSNCS

PURPOSE: Calculate the complex sine or cosine of complex x: y = sine (x)
y = cosine (x)

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:
CSIN
CCOS

EXTERNAL REFERENCES:
.ENTR, SIN, COS
EXP, ..FCM,

CALLING SEQUENCES:

```
JSB CSIN (or CCOS)
DEF * + 3
DEF Y
DEF X
JSB error routine
→ Normal return
```

METHOD: Sine: $y = Y_1 + i \cdot Y_2 = \sin(x) = \sin(x_1 + i \cdot x_2) =$

$$\frac{\sin(x_1)}{2}(e^{x_2} + e^{-x_2}) + i\left(\frac{\cos(x_1)}{2}\right)(e^{x_2} - e^{-x_2})$$

Cosine: $y = Y_1 + Y_2 . i = \cos(x) = \cos(x_1 + i.x_2) =$

$$\left(\frac{\cos(x_1)}{2}\right)(e^{x_2} + e^{-x_2}) + \left(\frac{i \cdot \sin(x_1)}{2}\right)(e^{-x_2} - e^{x_2})$$

Accuracy depends on the accuracy of EXP and SIN.

ATTRIBUTES:

ENTRY POINTS:

	CSIN	CCOS
Parameters:	Complex	Complex
Result:	Complex	Complex
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Function: CSIN (x)	Function: CCOS (x)
ALGOL:	Not callable	Not callable
Errors:	$\frac{1}{2} \mid \frac{x}{\pi} + \frac{1}{2} \mid > 2^{14}$ → (05 OR) (SIN) $x_2 \cdot \log_2 e \geq 124$ → (07 OF) (EXP)	

CSQRT

PURPOSE: Calculate the complex square root of complex x: $Y = y_1 + i\cdot y_2 = \sqrt{x_1 + i.x_2}$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	CSQRT
EXTERNAL REFERENCES:	.ENTP, ..DLC, .CFER SQRT, CABS, .ZRNT,

CALLING SEQUENCES:

```
JSB CSQRT
DEF * + 3
DEF Y (result)
DEF X
→
```

METHOD:

If $X = 0$, $Y = 0$

If $X_1 \geq 0$; $Y_1 = \sqrt{\frac{X_1 + |X|}{2}}$, $Y_2 = \frac{X_2}{2Y_1}$

If $X_1 < 0$; $Y_2 = \text{sign}(X_2)\sqrt{\frac{-X_1 + |X|}{2}}$, $Y_1 = \frac{X_2}{2Y_2}$

Accuracy depends on the accuracy of SQRT.

ATTRIBUTES: **ENTRY POINTS:**

	CSQRT
Parameters:	Complex
Result:	Complex
FORTRAN:	Callable: CALL CSQRT (y, x)
FORTRAN IV:	Function: CSQRT (x)
ALGOL:	Callable as a real procedure
Errors:	Overflow bit set if result out of range.

Note: See OVF function for testing results.

CSUB

PURPOSE: Interface routine which allows FORTRAN II programs to use the FORTRAN IV complex subtract routine, .CSUB.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: CSUB

EXTERNAL REFERENCES: .RCNG, .CSUB

CALLING SEQUENCES:

```
JSB CSUB
DEF *+4
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:

ENTRY POINTS:

	CSUB
Parameters:	Complex
Result:	Complex
FORTRAN:	Callable: Call CSUB (*z*,*x*,*y*)
FORTRAN IV:	Not Applicable
ALGOL:	Not Applicable
Errors:	Overflow bit set if result out of range.

Note: See OVF function for testing results.

DABS

PURPOSE: Calculate the absolute value of an extended real x: $y = |x|$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	DABS
EXTERNAL REFERENCES:	..DCM, .DFER, .ENTP, .ZRNT

CALLING SEQUENCES:

```
JSB DABS
DEF *+3
DEF y
DEF x
→
```

ATTRIBUTES:

ENTRY POINTS:

	DABS
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Function: DABS (x)
ALGOL:	Callable as real procedure
Errors:	If x = smallest negative number (-2^{127}), then y = largest positive number [$(1-2^{-39})\cdot 2^{127}$] and the overflow bit is set.

DATAN

PURPOSE: Calculate the extended real arctangent of extended real x: y = arctan (x)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	DATAN
EXTERNAL REFERENCES:	.ZRNT, .XADD, .XSUB, .XMPY, .XDIV, .ENTP, ..DCM, .FLUN, .DFER

CALLING SEQUENCES:

```
JSB DATAN
DEF *+3
DEF Y (result)
DEF X
→
```

METHOD:

If $x < 0$, $y = -\arctan(-x)$

If $|x|>1$, let $z = \frac{1}{|x|}$, then $y = \frac{\pi}{2} - \arctan(z)$

If $|x|<1$, let $z = |x|$

If $z \leq \sqrt{2} - 1$, set $v = \tan\frac{\pi}{16}$, $w=\frac{\pi}{16}$

If $z < \sqrt{2} - 1$, set $v = \tan\frac{3\pi}{16}$, $w = \frac{3\pi}{16}$

Then $T = \frac{z-v}{1+z \cdot v}$

$\text{Arctan}(z) = w + \arctan(T)$

$$\text{Arctan}(T) = T\left[C_0 + \frac{C_1[(T^2+B_2)(T^2+B_3)+C_3]}{(T^2+B_1)[(T^2+B_2)(T^2+B_3)+C_3]+C_2(T^2+B_3)}\right]$$

C_0 = .208979591837

C_1 = 2.97061224490 B_1 = 5.10299532839

C_2 = -3.35025248131 B_2 = 2.58417875505

C_3 = -.128720995297 B_3 = 1.21282591656

Accuracy: The relative error in $y = \arctan(x+\Delta x)$ is $R = \frac{\Delta x}{(x^2+1)\arctan(x)}$ where Δx represents the round-off error in x. Hence, at $x = \pm.001$, the accuracy will be 9 significant digits due to the round-off error in the 39th bit of x. As x diverges from 0, the accuracy becomes 11 significant digits.

ATTRIBUTES:

ENTRY POINTS:

	DATAN
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable: CALL DATAN (y,x)
FORTRAN IV:	Function: DATAN (x)
ALGOL:	Callable as real procedure
Errors:	None

DATN2

PURPOSE: Calculate the extended real arctangent of the quotient of two extended reals:

$$Z = \arctan (Y/X)$$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: DATN2 DATA2

EXTERNAL REFERENCES: .ENTP, DSIGN, DATAN, .ZRNT
.XADD, .XDIV, .DFER

CALLING SEQUENCES:

```
JSB DATN2 (or DATA2)
DEF *+4
DEF Z (result)
DEF Y
DEF X
→
```

METHOD:

If $X = 0$, $Z = \text{sign}(Y) \cdot \frac{\pi}{2}$

If $X > 0$, $Z = \arctan (Y/X)$

If $X < 0$, $Z = \arctan (Y/X) + \text{sign}(Y) \cdot \pi$

Accuracy depends on accuracy of DATAN.

ATTRIBUTES:

	ENTRY POINTS:
	DATN2 DATA2
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable: DATAN2 (I*z*,I*y*,I*x*)
FORTRAN IV:	Function: DATN2 (y,x)
ALGOL:	Callable as real procedure
Errors:	None

DBLE

PURPOSE: Convert a real *x* to an extended real *Y*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: DBLE

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB DBLE
DEF *+3
DEF y (result)
DEF x
→
```

ATTRIBUTES:	**ENTRY POINTS:**
	DBLE
Parameters:	Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Function: DBLE (*x*)
ALGOL:	Callable as real procedure
Errors:	None

Note: This routine is available in firmware. See description of FFP on page 1-6.

DCOS

PURPOSE: Calculate the extended real cosine of extended real *x* (angle in radians): $y = \cos(x)$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: DCOS

EXTERNAL REFERENCES: .ENTP, DSIN, .ZRNT, .XADD

CALLING SEQUENCES:

```
JSB DCOS
DEF *+3
DEF Y (result)
DEF X
→
```

METHOD:

$Y = \cos(x) = \sin(x + \pi/2)$

Accuracy depends on the accuracy of DSIN.

ATTRIBUTES:

	ENTRY POINTS:
	DCOS
Parameters:	Extended Real (radians)
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Function: DCOS (*x*)
ALGOL:	Callable as real procedure
Errors:	None

DDINT

PURPOSE: Truncate an extended real *x* to an extended real *y*:

PROGRAM TYPE = 6

ROUTINE IS: R

ENTRY POINTS: DDINT

EXTERNAL REFERENCES:
.XADD, .ENTP, .ZRNT
ENTIX

CALLING SEQUENCES:

```
JSB DDINT
DEF *+3
DEF Y
DEF X
→
```

METHOD: y = Largest integer $\leq x$

ATTRIBUTES:	ENTRY POINTS:
	DDINT
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Function: DDINT (x)
ALGOL:	Callable as real procedure
Errors:	None

Note: This routine is available in 21MX FFP firmware.
See summary in section I.

DEXP

PURPOSE: Calculate the extended real exponential of a extended real x: $y = e^x$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: DEXP

EXTERNAL REFERENCES: .ENTP,.XADD, .XSUB, .XMPY, .XDIV, .DFER, .ZRNT, DDINT, SNGL, IFIX, .FLUN, .XPAK

CALLING SEQUENCES:

```
JSB DEXP
DEF *+3
DEF y (result)
DEF x
→ error return
→ normal return
```

METHOD:

$$e^x = 2^N e^z \qquad \text{where: } z = \ln 2\ (x\log_2 e - N)$$

$$N = [x\log_2 e \pm 1/2] \text{ (see DDINT)}$$

$$e^z = Co + \frac{C_1(z(z^2+C_4)+C_3z)}{(z+B_1)(z(z^2+C_4)+C_3z)+C_2(z^2+C_4)}$$

$Co = 1.0$	$C_2 = 138.0$	$C_4 = 12.17391304348$
$C_1 = 40.0$	$C_3 = 29.8260869565$	$B_1 = -20.0$

Accuracy: The relative error in $y = e^{x + \Delta x}$ is $R = \Delta x$ where Δx represents the error in the argument. Thus for $|x| < 1$, the accuracy will be 11 significant digits, but for $|x|$ near 100, the accuracy will be 8 significant digits.

ATTRIBUTES:

	ENTRY POINTS:
	DEXP
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Function: DEXP (x)
ALGOL:	Not callable
Errors:	If $e^x > (1-2^{-39})\ 2^{127}$ → (10 OF)

DIM

PURPOSE: Calculate the positive difference between real *x* and *y*: $z = x - \min(x,y)$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	DIM
EXTERNAL REFERENCES:	.FSB, .ZPRV
CALLING SEQUENCES:	JSB DIM DEF *+3 DEF *x* DEF *y* → *z* in A & B

ATTRIBUTES:

	ENTRY POINTS:
	DIM
Parameters:	Real
Result:	Real
FORTRAN:	Callable: Z = DIM (x,y)
FORTRAN IV:	Function: DIM (x,y)
ALGOL:	Callable as Real Procedure
Errors:	None

DLOG

PURPOSE: Calculate the extended real natural logarithm of a extended real x:

$$y = \log_e x$$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: DLOG

EXTERNAL REFERENCES: .ENTP, .XADD, .XSUB, .XMPY, .XDIV, .FSB, .FLUN, FLOAT, DBLE, .DFER, .ZRNT

CALLING SEQUENCES:

```
JSB DLOG
DEF *+3
DEF y (result)
DEF x
→ error return
→ normal return
```

METHOD:

$$\ln(x) = (n-1/2)\ln 2 + \ln\left(\frac{1+z}{1-z}\right)$$

where: n = Exponent of x

m = Mantissa of x

$$z = \frac{m - \sqrt{2}/2}{m + \sqrt{2}/2}$$

$$\ln \frac{1+z}{1-z} = z\left[\frac{c_1[(z^2+B_2)(z^2+B_3)+C_3]}{(z^2+B_1)[(z^2+B_2)(z^2+B_3)+C_3]+C_2(z^2+B_3)}\right]$$

C_1 = -18.4800000000 B_1 = -15.8484848485

C_2 = -23.643709825 B_2 = -3.75400078147

C_3 = -.246270037272 B_3 = -1.39751437005

Accuracy: See Note.

ATTRIBUTES:

	ENTRY POINTS:
	DLOG
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Function: DLOG (x)
ALGOL:	Not callable
Errors:	If $x \leq 0$ → (11 UN)

NOTE: The relative error in $y = \ln(x+\Delta x)$ is $R = \frac{\Delta x}{x \ln x}$. Hence, the relative error increases as x approaches 1. At $x = 1.000 \pm .001$ the accuracy will be 9 significant digits due to an error in the 39th bit in the representation of x. As x diverges from 1 the accuracy becomes 11 significant digits.

DLOGT

PURPOSE: Calculate the extended real common logarithm of extended real x:

$Y = \log_{10} x$

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: DLOGT (DLOG0)

EXTERNAL REFERENCES: .ENTP, DLOG, .XMPY

CALLING SEQUENCES:

```
JSB DLOGT (DLOG0)
DEF *+3
DEF Y (result)
DEF X
→ error return
→ normal return
```

METHOD:

$Y = \log_{10} X = \log_e x / \log_e 10$

Accuracy depends on the accuracy of DLOG.

ATTRIBUTES:

	ENTRY POINTS:
	DLOGT (or DLOG0)
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Function: DLOGT (x)
ALGOL:	Not callable
Errors:	If $x < 0$ → (11 UN)

DMOD

PURPOSE: Calculate the extended real remainder of two extended real values:

$$z = X \text{ mod } Y$$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	DMOD
EXTERNAL REFERENCES:	.ENTP, .XSUB, .XMPY, .XDIV, DDINT, .ZRNT
CALLING SEQUENCES:	JSB DMOD DEF *+4 DEF z (result) DEF X DEF Y →

METHOD: $z = x - [\text{DDINT } (x/y)]y$

ATTRIBUTES:

	ENTRY POINTS:
	DMOD
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable: CALL DMOD (Iz,Ix,Iy)
FORTRAN IV:	Function: DMOD (x,y)
ALGOL:	Callable as real procedure
Errors:	If $Y = 0$, then $z = X$

DSIGN

PURPOSE: Transfer the sign of a extended real y to a extended real x:

$$z = \text{sign}\ (Y)\ .\ |x|$$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS: DSIGN

EXTERNAL REFERENCES: .DFER, .ENTP, ..DCM, .ZRNT

CALLING SEQUENCES:

```
JSB DSIGN
DEF *+4
DEF z (result)
DEF x
DEF Y
→
```

ATTRIBUTES:

	ENTRY POINTS:
	DSIGN
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Function: DSIGN (x,y)
ALGOL:	Callable as real procedure
Errors:	If $Y = 0$, $z = 0$.

DSIN

PURPOSE: Calculate the extended real sine of extended real x (angle in radians):

$$y = \sin(x)$$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:		DSIN
EXTERNAL REFERENCES:	.ENTP, .XSUB.	..DCM, XPOLY, .DFER ENTIX, .XADD, .XMPY, .XDIV, .ZRNT
CALLING SEQUENCES:		JSB DSIN DEF *+3 DEF *y* DEF *x* →

METHOD: x is reduced to the range $-\frac{\pi}{2} \leq x < \frac{\pi}{2}$

If $x < 10^{-6}$, $\sin(x) = x$.

Otherwise $\sin(x) = \left(\sum_{i=1}^{6} C_i x^{2i} + 1 \right) x$

C_1 = -.166666666667 E+0 C_3 = -.198412663895 E-3 C_5 = -.250294478915 E-7

C_2 .833333331872 E-2 C_4 = .275569300800 E-5 C_6 = .154001500048 E-9

When x is near a non-zero multiple of π, the accuracy of the result is limited by the accuracy of the subtraction $n\pi - x$.

ATTRIBUTES:

	ENTRY POINTS:
	DSIN
Parameters:	Extended Real (radians)
Result:	Extended Real
FORTRAN:	Callable: CALL DSIN (I*y*,I*x*)
FORTRAN IV:	Function: DSIN (*x*)
ALGOL:	Callable as real procedure
Errors:	None

DSQRT

PURPOSE: Calculate the extended real square root of extended real x: Y = sqrt (x)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: DSQRT

EXTERNAL REFERENCES: .ENTP, DBLE, SNGL, SQRT, .XDIV, .XADD, .ZRNT, .XMPY

CALLING SEQUENCES:

```
JSB DSQRT
DEF *+3
DEF Y (result)
DEF X
→ error return
→ normal return
```

METHOD:

A first approximation is found using the single precision SQRT: z = SQRT (x)

Then $Y = \frac{z+x/z}{2}$

Accuracy is 11 significant digits.

ATTRIBUTES:

ENTRY POINTS:

	DSQRT
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Function: DSQRT (x)
ALGOL:	Not callable
Errors:	If $x < 0$ → (03 UN)

DTAN

PURPOSE: Calculate tangent of extended real X.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: DTAN

EXTERNAL REFERENCES: .ENTR, .DFER, .TMPY, .TSUB, .TINT, .ITBL, .XADD, .XMPY, .XDIV, XPOLY

CALLING SEQUENCES:

```
JSB DTAN
DEF *+3
DEF <result>
DEF x
 <error return>
→
```

METHOD: The range is reduced to (-1,1) using the identities:

TAN(X) = TAN(X-N*π)
TAN(X*4/π) = TAN(X*4/π-4*N)
TAN(X) = -1.0 / TAN(X-π/2)
TAN(X*4/π) = -1.0 / TAN(X*4/π-2)

The following approximation is used on the reduced range:

$$TAN(X) = Z * \left(\frac{C1}{C2+ZSQ} + C3+ZSQ*(C4+ZSQ*(C5+ZSQ*C6))\right)$$

C1 = -.254660667110D+01
C2 = -.400002835440D+01
C3 = .148751008558D+00
C4 = .233036398271D-02
C5 = .564290881573D-04
C6 = .133098254545D-05

Z = X*4/π
ZSQ = Z*Z

ATTRIBUTES:

ENTRY POINTS:

	DTAN
Parameters:	Extended real (radians)
Result:	Extended real
FORTRAN:	Callable: Call DTAN(Y,X)
FORTRAN IV:	Function: DTAN(X)
ALGOL:	Callable as real procedure
Errors:	X outside [-8192*π,+8191.75*π] → 09 OR

NOTES:

DTANH

PURPOSE: Calculate hyperbolc tangent of extended real X

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: DTANH

EXTERNAL REFERENCES: .ENTR, .DFER, .XFER, .FLUN, .PWRZ, DEXP, .XADD, .XMPY, .XDIV

CALLING SEQUENCES:

```
JSB DTANH
DEF *+3
DEF <result>
DEF x
→
```

METHOD: Outside the range [-32,+32) the result is 1.0 times the sign of the argument. Within the above range but outside the range [-0.25,+0.25) the definition is used:

$$\text{TANH(X)} = \frac{\text{EXP(2*X)} - 1}{\text{EXP(2*X)} + 1}$$

Within [-0.25,+0.25) the following approximation is used:

$$\text{TANH(X)} = \text{X} * \left(\frac{\text{C1}}{\text{C2+XSQ}} + \text{C3} + \text{C4*XSQ} \right)$$

WHERE:
C1 = .201101929221D+01
C2 = .247073386009D+01
C3 = .186063976899D+00
C4 = .390245451777D-02
XSQ = X*X

ATTRIBUTES:

	ENTRY POINTS:
	DTANH
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Callable: Call DTANH(Y,X)
FORTRAN IV:	Function: DTANH(X)
ALGOL:	Callable as real procedure
Errors:	None

NOTES:

ENTIE

PURPOSE: 1) Calculate the greatest integer not algebraically exceeding a real *x* (ENTIE);

2) Round a real *x* to the nearest integer; if half way between two integers, select the algebraically larger integer (.RND).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	ENTIE .RND
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	DLD *x* JSB .RND (or ENTIE) → result in A

ATTRIBUTES:

ENTRY POINTS:

	ENTIE	.RND
Parameters:	Real	Real
Result:	Two integers: sign in A; integer in B	Integer in A
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Intrinsic Function: ENTIER (*x*)	Not callable
Errors:	See Note 1	See Note 1

NOTE 1: If exponent >15, then overflow is indicated as follows:

If $x \geq 0$ then A = 32767
else A = -32768
Result: Integer in A

ENTIX

PURPOSE: Calculate ENTIER of extended real x:

y = ENTIER (x) = greatest integer not algebraically exceeding x.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:

```
.XENT
ENTIX
```

EXTERNAL REFERENCES:

```
.ENTP, .ZPRV
```

CALLING SEQUENCES:

```
JSB .XENT(or ENTIX)
DEF * + 3
DEF y
DEF x
→
```

ATTRIBUTES:

ENTRY POINTS:

	ENTIX	.XENT
Parameters:	Extended Real	Extended Real
Result:	Extended Real	Extended Real
FORTRAN:	Callable	Not Callable
FORTRAN IV:	Callable	Not Callable
ALGOL:	Callable as real procedure	Not Callable
Errors:	None	None

EXP

PURPOSE: Calculate e^x, where x is real.

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	EXP
EXTERNAL REFERENCES:	.ZPRV, .CMRS, .PWRZ, .FMP, .FSB, .FAD, .FDV
CALLING SEQUENCES:	DLD x JSB EXP JSB ERR0 (error) → (y in A & B)

METHOD:

$EXP(x) = 2^N * 2^Z$ where z $x/\ln(2)-N$.
N is chosen as the closest integer to $x/\ln(2)$.

The Following Formula is used:

$$EXP\ (x) = 2^{N+1} * \left(0.5 + \frac{y}{A-y+B*y^2}\right)$$

where

$A = 5.7708162$
$B = .05761803$

$y = 2*(x/\ln(2)-N)$
y is in the range $[-1,+1]$

ATTRIBUTES:

	ENTRY POINTS:
	EXP
Parameters:	Real: A & B
Result:	Real: A & B
FORTRAN:	Function: EXP (x)
FORTRAN IV:	Function: EXP (x)
ALGOL:	Intrinsic Procedure: EXP (x)
Errors:	$x*\log_2 e \geq 127$ → (07 0F)

NOTE: If $x < -129*\log_e (2)$, underflow occurs. A zero will be returned with no error indication.

FADSB

PURPOSE:

.FAD: Add real x to y

$z = x + y$

.FSB: Subtract real y from x

$z = x - y$

PROGRAM TYPE = 6

ROUTINE IS: P

ENTRY POINTS: .FAD, .FSB

EXTERNAL REFERENCES: .PACK, .ZPRV

CALLING SEQUENCES:

DLD x
FAD (FSB) y
→ result in A&B

ATTRIBUTES:

ENTRY POINTS:

	.FAD	.FSB
Parameters:	Real	Real
Result:	Real	Real
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	See Note 1	See Note 1

NOTES:

1. If the result is outside the range of representable floating point numbers $[-2^{127}, 2^{127}(1-2^{-23})]$ the overflow flag is set and the result $2^{127}(1-2^{-23})$ is returned. If an underflow occurs, (result within the range $(-2^{-129}(1+2^{-22}), 2^{-129})$ excluding 0), the overflow flag is set and the result 0 is returned.

FLOAT

PURPOSE: Convert integer I to real x

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	FLOAT
EXTERNAL REFERENCES:	.PACK, .ZPRV
CALLING SEQUENCES:	LDA I JSB FLOAT → (x in A & B)

ATTRIBUTES:

	ENTRY POINTS:
	FLOAT
Parameters:	Integer: A
Result:	Real: A & B
FORTRAN:	Function: FLOAT (I)
FORTRAN IV:	Function: FLOAT (I)
ALGOL:	Not callable
Errors:	None

IABS

PURPOSE: Calculate absolute value of integer *I*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	IABS
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	LDA *I* JSB IABS → result in A

ATTRIBUTES:

	ENTRY POINTS:
	IABS
Parameters:	Integer: A
Result:	Integer: A
FORTRAN:	Function: IABS (*I*)
FORTRAN IV:	Function: IABS (*I*)
ALGOL:	Not Callable
Errors:	NOTE 1

NOTE: 1. Note that if IABS is (-32768), the result is 32767 and the overflow bit is set.

IAND

PURPOSE: Take the logical product and integers *I* and *J*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: IAND

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
JSB IAND
DEF I
DEF J
→ result in A
```

ATTRIBUTES:

	ENTRY POINTS:
	IAND
Parameters:	Integer
Result:	Integer in A
FORTRAN:	Function: IAND (*I*,*J*)
FORTRAN IV:	Function: IAND (*I*,*J*)
ALGOL:	Not callable
Errors:	None

IDIM

PURPOSE: Calculate the positive difference between integers I & J

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	IDIM
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB IDIM DEF *+3 DEF I DEF J → result in A

METHOD: A = I-min(I,J)

	ENTRY POINTS:
ATTRIBUTES:	IDIM
Parameters:	Integer in A
Result:	Integer
FORTRAN:	Callable
FORTRAN IV:	Function: IDIM (I,J)
ALGOL:	Callable as integer procedure
Errors:	NOTE 1

NOTE: 1. If IDIM(I,J) is out of range, the overflow bit is set and a value of 32767 returned.

IDINT

PURPOSE: Truncate an extended real X

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	IDINT
EXTERNAL REFERENCES:	IFIX, .ZPRV, SNGM
CALLING SEQUENCES:	JSB IDINT DEF *+2 DEF *x* → result in A

METHOD: A = largest integer $\leq x$

ATTRIBUTES:

	ENTRY POINTS:
	IDINT
Parameters:	Extended Real
Result:	Integer in A
FORTRAN:	Callable as function
FORTRAN IV:	Function: IDINT (x)
ALGOL:	Callable as integer procedure
Errors:	If IDINT (x) is out of range, then result = 32767 and the overflow bit is set.

IFIX

PURPOSE: Convert a real x to an integer
I = SIGN(x), (largest integer $\leq|x|$), or $J = |x|$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: IFIX (P)

EXTERNAL REFERENCES:
Non-floating point libraries: .FLUN
Floating point library: .ZPRV
(See note 2)

CALLING SEQUENCES:

```
DLD x
JSB IFIX
→ result in A
```

ATTRIBUTES:

	ENTRY POINTS:
	IFIX
Parameters:	Real: A & B
Result:	Integer: A (See Note 1)
FORTRAN:	Function: IFIX (x)
FORTRAN IV:	Function: IFIX (x)
ALGOL:	Not callable
Errors:	None

NOTES:

1. Any fractional portion of the result is truncated. If the integer portion is greater than or equal to 2^{15}, the result is set to 32767.
2. The *routine* IFIX exists only in non-floating point libraries.

INT

PURPOSE: Truncate a real x to an integer

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	INT
EXTERNAL REFERENCES:	IFIX
CALLING SEQUENCES:	DLD x JSB INT → result in A

METHOD: result = (sign of x)*(largest integer $\leq |x|$)

ATTRIBUTES:

	ENTRY POINTS:
	INT
Parameters:	Real
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Function: INT (x)
ALGOL:	Not callable
Errors:	If INT (x) is out of range, the overflow bit is set. The result is set to 32767.

IOR

PURPOSE: Take logical inclusive - or of integers *I* and *J*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	IOR
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB IOR DEF *I* DEF *J* → result in A

ATTRIBUTES:

	ENTRY POINTS:
	IOR
Parameters:	Integer
Result:	Integer
FORTRAN:	Function: IOR (*I*,*J*)
FORTRAN IV:	Function: IOR (*I*,*J*)
ALGOL:	Not callable
Errors:	None

ISIGN

PURPOSE: Calculate the sign of z times the absolute value of I, where z is real or integer and I is integer: $Y=\text{sign}(z)*|I|$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	ISIGN
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB ISIGN DEF I DEF z → result in A

METHOD: Same as SIGN

ATTRIBUTES:

	ENTRY POINTS:
	ISIGN
Parameters:	Real (or int) & integer
Result:	Integer: A
FORTRAN:	Function: ISIGN (I,z)
FORTRAN IV:	Function: ISIGN (I,z)
ALGOL:	Not callable
Errors:	None

IXOR

PURPOSE: Perform integer exclusive OR

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	IXOR
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB IXOR DEF *+3 DEF INTA DEF INTB → result in A

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	IXOR
Parameters:	INTEGER
Result:	INTEGER
FORTRAN:	Callable as a function: IXOR (INTA, INTB)
FORTRAN IV:	Callable as a function: IXOR (INTA, INTB)
ALGOL:	Callable as a function: IXOR (INTA, INTB)
Errors:	None

NOTES:

COMMENTS:

MOD

PURPOSE: Calculate the integer remainder of *I*/*J* for integer *I* & *J*;
result = *I* modulo *J*

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: MOD

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB MOD
DEF *+3
DEF I
DEF J
→ result in A & B
```

METHOD: result = *I* - [The truncated value of *I*/*J*]**J*

ATTRIBUTES:

ENTRY POINTS:

	MOD
Parameters:	Integer
Result:	Integer
FORTRAN:	Callable as function
FORTRAN IV:	Function: MOD (*I*,*J*)
ALGOL:	Part of language: I MOD J
Errors:	If *J* = 0, then result = *I*

MXMND

PURPOSE: Calculate the maximum or minimum of a series of extended real values:
$Y = \max(A,B,C,....)$ $Y = \min(A,B,C,....)$

PROGRAM TYPE = 7 **ROUTINE IS: R**

ENTRY POINTS:	DMAX1 DMIN1
EXTERNAL REFERENCES:	.XSUB .DFER

CALLING SEQUENCES:

```
JSB DMAX1(or DMIN1)
DEF *+N+2
DEF Y (result)
DEF A (1)
DEF B (2)
 :  .  :
DEF X (N)
→
```

ATTRIBUTES:

ENTRY POINTS:

	DMAX1	DMIN1
Parameters:	Extended Real	Extended Real
Result:	Extended Real	Extended Real
FORTRAN:	Callable as Subroutine	Callable as Subroutine
FORTRAN IV:	Note 1	Note 1
ALGOL:	Note 2	Note 2
Errors:	If $N < 2$, then $Y = 0$	If $N < 2$, then $Y = 0$

NOTES:

1. Intrinsic functions: DMAX1 $(A,B,C,\ ....)$
   DMIN1 $(A,B,C,\ ....)$
2. Callable as a real procedure, but only with a fixed number of parameters.

COMMENTS: Requires at least two parameters.

MXMNI

PURPOSE: Calculate the maximum or minimum of a series of integer values:

Y = MAX ($A,B,C,$ ......) Y = MIN ($A,B,C,$ ......)

PROGRAM TYPE = 7   **ROUTINE IS:** U

ENTRY POINTS: AMAX0, MAX0, AMIN0, MIN0

EXTERNAL REFERENCES: FLOAT

CALLING SEQUENCES:

```
JSB Entry Point
DEF *+N+1
DEF A (1)
DEF B (2)
 :
DEF X (N)
→ Result in A or A & B
```

ATTRIBUTES:

	ENTRY POINTS:			
	AMAX0	MAX0	AMIN0	MIN0
Parameters:	Integer	Integer	Integer	Integer
Result:	Real	Integer	Real	Integer
FORTRAN:	Note 1	Note 1	Note 1	Note 1
FORTRAN IV:	Note 1	Note 1	Note 1	Note 1
ALGOL:	Note 2	Note 2	Note 2	Note 2
Errors:	Note 3	Note 3	Note 3	Note 3

NOTES:

1. Functions: AMAX0 (A,B,C ....), MAX0 (A,B,C ....) AMN0 (A,B,C ....), MIN0 (A,B,C....)
2. Callable as integer or real procedure, but only with a fixed number of parameters.
3. If the number of parameters is less than 2, Y = 0.

COMMENTS: Requires at least two parameters.
AMAX0 provides a real maximum.
MAX0 provides an integer maximum.
AMIN0 provides a real minimum.
MIN0 provides an integer minimum.

MXMNR

PURPOSE: Calculate the maximum or minimum of a series of real values:

Y = Max (*A*,*B*,*C* ....) *Y* = Min (*A*,*B*,*C* ....)

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS: AMAX1, MAX1, AMIN1, MIN1

EXTERNAL REFERENCES: IFIX, .FSB

CALLING SEQUENCES:

```
JSB Entry Point
DEF *+ N + 1
DEF A (1)
DEF B (2)
 :     :
DEF X (N)
→ Y in A or A & B
```

ATTRIBUTES:

ENTRY POINTS:

	AMAX1	MAX1	AMIN1	MIN1
Parameters:	Real	Real	Real	Real
Result:	Real	Integer	Real	Integer
FORTRAN:	Note 1	Note 1	Note 1	Note 1
FORTRAN IV:	Note 1	Note 1	Note 1	Note 1
ALGOL:	Note 2	Note 2	Note 2	Note 2
Errors:	Note 3	Note 3	Note 3	Note 3

NOTES:

1. Functions: AMAX1 (*A*,*B*,*C*, ...), MAX1 (*A*,*B*,*C*, ...),
   AMIN1 (*A*,*B*,*C*, ...), MIN1 (*A*,*B*,*C*, ...).
2. Callable as integer or real procedure, but only with a fixed number of parameters.
3. If the number of parameters is less than 2, *Y* = 0.

COMMENTS:

Requires at least two parameters.
AMAX1 provides a real maximum.
MAX1 provides an integer maximum.
AMIN1 provides a real minimum.
MIN1 provides an integer minimum.

REAL

PURPOSE: Extract the real part of a complex *x*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	REAL
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB REAL DEF *+2 DEF *x* → result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	REAL
Parameters:	Complex
Result:	Real
FORTRAN:	Callable as Function
FORTRAN IV:	Function: REAL (*x*)
ALGOL:	Callable as real procedure
Errors:	None

SIGN

PURPOSE: Calculate the sign of z times the absolute value of x, where z is real or integer and x is real; if z = 0, then the result equals abs (x) unless used under the RTE-II, III or IVA operating system in which case the result equals 0.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	SIGN
EXTERNAL REFERENCES:	..FCM, .ZPRV
CALLING SEQUENCES:	JSB SIGN DEF x DEF z → (result in A & B)

ATTRIBUTES:

	ENTRY POINTS:
	SIGN
Parameters:	Real or Integer and Real
Result:	Real
FORTRAN:	Function: SIGN (x, z)
FORTRAN IV:	Function: SIGN (x, z)
ALGOL:	Not callable
Errors:	None

SIN

PURPOSE: See .SINCS

SNGL

PURPOSE: Convert an extended real *x* to a real *Y*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: SNGL

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB SNGL
DEF *+2
DEF x
→ Y in A & B
```

ATTRIBUTES:

	ENTRY POINTS:
	SNGL (See note 1)
Parameters:	Extended Real
Result:	Real
FORTRAN:	Callable
FORTRAN IV:	Function: SNGL (*x*)
ALGOL:	Callable as Real Procedure
Errors:	If $x > (1-2^{-23})*2^{127}$ (the maximum real number), then $Y = (1-2^{-23})*2^{127}$, and the overflow bit is set

Note: The routine is available in firmware.
See description of FFP on page 1-6.

SNGM

PURPOSE: Convert a extended real *x* to a real *y* without rounding.
$|y| \leq |x|$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	SNGM
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB SNGM DEF *+2 DEF *x* (extended precision 3-word parameter) → *y* in A & B

ATTRIBUTES:

	ENTRY POINTS:
	SNGM
Parameters:	Extended Real
Result:	Real
FORTRAN:	Callable
FORTRAN IV:	Function: SNGM (*x*)
ALGOL:	Callable as Real Procedure
Errors:	If y < ABS $((-1+2^{-23}) *2^{-128})$, zero is returned

NOTE: 1. Maximum error will be less than the least significant bit.

SQRT

PURPOSE: Calculate the square root of a real X: $Y = \sqrt{X}$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	SQRT
EXTERNAL REFERENCES:	.ZPRV, .FLUN, .PWR2, .FMP, .FAD, .FDV
CALLING SEQUENCES:	DLD *x* JSB SQRT JSB ERR0 (error) → (*Y* in A and B)

METHOD:

The range is reduced to [.5, 2) using the identity:

$$SQRT\ (x) = 2^N * SQRT\ (x/2^{2N})$$

The initial approximation is

$$X0 = A * y + B \text{ with } y = x/2^{2N}$$

where A = .5901621 B = .4173076 For y in [.5,1)	A = .4173076 B = .5901621 For y in [1,2)

Heron's rule is then applied twice:

$$2*X1 = X0 + \frac{y}{X0} \text{ and } 4*X2 = 2*X1 + \frac{4*y}{2*X1}$$

$$SQRT\ (x) = (4*X_2)*2^{(N-2)}$$

ATTRIBUTES:

	ENTRY POINTS:
	SQRT
Parameters:	Real: A & B
Result:	Real: A & B
FORTRAN:	Function: SQRT (*x*)
FORTRAN IV:	Function: SQRT (*x*)
ALGOL:	Intrinsic Procedure: SQRT (*x*)
Errors:	$x < 0$ → (03 UN)

TAN

PURPOSE: Calculate the tangent of a real x (radians): Y = tangent (x)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	TAN
EXTERNAL REFERENCES:	.ZPRV, .CMRS, .FMP, . FAD, .FDV
CALLING SEQUENCES:	DLD x JSB TAN JSB ERR0 (error) → (Y in A & B)

METHOD: x is reduced to the range ($\pi/4$, $\pi/4$) using the identities:

$$\mathrm{TAN}(x) = \mathrm{TAN}(x + K*\pi)$$
$$\mathrm{TAN}(x) = -1 / \mathrm{TAN}\ (x + \pi/2)$$

Then the Following Formula is used:

$$\mathrm{TAN}(x) = Y*(A + B + (Y^2 + \frac{C}{D+Y^2}))$$

$$Y=x*(4/\pi)$$

where: A = .14692695 C = -1279.5424

B = .0019974806 D = -4.0030956

ATTRIBUTES:

	ENTRY POINTS:
	TAN
Parameters:	Real Radians: A and B (Radians)
Result:	Real: A and B
FORTRAN:	Function: TAN (x)
FORTRAN IV:	Function: TAN (x)
ALGOL:	Intrinsic Procedure: TAN (x)
Errors:	X outside [-8192*π,+8191.75*π] → 09 OR

TANH

PURPOSE: Calculate the hyperbolic tangent of a real x: y=TANH (x)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	TANH
EXTERNAL REFERENCES:	.ZPRV, EXP, .FAD .FSB, .FDV, .FMP
CALLING SEQUENCES:	DLD x JSB TANH → (y in A and B)

METHOD:

1. $|x| \geq 8$: TANH (x) = SIGN $(1.0,x)$
2. $.5 \leq |x| \leq 8$: TANH (x) = $(\mathrm{EXP}(2*x)-1)/(\mathrm{EXP}(2*x) + 1)$
3. $|x| \leq .5$: TANH (x) = $x * (\mathrm{A} + \dfrac{\mathrm{B}}{x^2 + \mathrm{C}})$

where:

A = .16520923
B = 2.0907609
C = 2.5046337

ATTRIBUTES: **ENTRY POINTS:**

	TANH
Parameters:	Real: A and B
Result:	Real: A and B
FORTRAN:	Function: TANH (x)
FORTRAN IV:	Function: TANH (x)
ALGOL:	Intrinsic Procedure: TANH (x)
Errors:	None

DPOLY

PURPOSE: Evaluate the quotient of two polynomials in double precision

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	DPOLY, TRNL
EXTERNAL REFERENCES:	.ENTR, .CFER, .TADD, .TSUB, .TMPY, .TDIV, .4ZRO

CALLING SEQUENCES:

```
JSB DPOLY                      OR  JSB DPOLY
DEF *+6                            OCT <Flags>
DEF <result>                       DEF <result>
DEF <argument>                     DEF <argument>
DEF <coefficient list>             DEF <coefficient list>
DEF <order of numerator>           DEF <order of numerator>
DEF <order of denominator>         DEF <order of denominator>
```

METHOD: Horner's rule is used. If the order of the denominator is zero, the denominator's value is one; the divide is not done. In this case, TRNL acts as a polynominal evaluator. If bit 15 of the flag word of the call is set, the polynomials are evaluated in X^2 instead of X and:

Bit 14=1: The numerator is subtracted from the denominator before the divide (N>0 only).

Bit 0=0: The quotient is multiplied by X.

ATTRIBUTES:

	ENTRY POINTS:
	DPOLY
Parameters:	Double real (last two parameters are integer)
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: Z=DPOLY(X,C,M,N)
ALGOL:	Not callable
Errors:	None

NOTES:

1) The coefficients must be presented in the order used, i.e., for:

$$\frac{P_M X^M + P_{M-1} X^{M-1} + \ldots + P_1 X + P_0}{X^N + Q_{N-1} X^{N-1} + \ldots + Q_1 X + Q_0}$$

The coefficient array must be:

$$P_M, P_{M-1}, \ldots, P_1, P_0, Q_{N-1}, \ldots Q_1, Q_0$$

If m = 0, Q_0 = 1.0 and need not be supplied.

2) This routine may alter the X and Y registers.

3) Since bit 15 of the flag word must be set to enable any options, these options are not FORTRAN callable.

XADD

PURPOSE: Interface routine to allow FORTRAN II to use the FORTRAN IV Extended Real addition, .XADD.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	XADD
EXTERNAL REFERENCES:	.RCNG, .XADD
CALLING SEQUENCES:	JSB XADD DEF *+4 DEF 2 (result) DEF *x* DEF *y*

ATTRIBUTES:

	ENTRY POINTS:
	XADD
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Not Applicable
ALGOL:	Not Applicable
Errors:	See XADSB

XADSB

PURPOSE: Extended real addition and subtraction: $Z = X + Y$ $Z = X - Y$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:

.XADD
.XSUB

EXTERNAL REFERENCES:

.XPAK, ADRES .ZPRV

CALLING SEQUENCES:

```
JSB(.XADD or .XSUB)
DEF Z (result)
DEF X
DEF Y
→
```

ATTRIBUTES:

ENTRY POINTS:

	.XADD	.XSUB
Parameters:	Extended Real	Extended Real
Result:	Extended Real	Extended Real
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	Note 1	Note 1

NOTES:

1. If $z > 2^{127}(1-2^{-39})$, overflow is set and $z = 2^{127}(1-2^{-39})$.
   If $z < -2^{127}$, overflow is set and $z = -2^{127}$.
   If $0 < z < 2^{129}$, overflow is set and $z = 0$.
   If $-2^{-129}(1 + 2^{-38}) < z < 0$, overflow is set and $z = 0$.

2. These routines are available in firmware. See description of FFP on page 1-6.

XDIV

PURPOSE: Interface routine which allows FORTRAN II programs to use Extended Real Divide routine .XDIV.

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	XDIV
EXTERNAL REFERENCES:	.RCNG, .XDIV
CALLING SEQUENCES:	JSB XDIV DEF *+4 DEF *z* (result) DEF *x* DEF *y*

ATTRIBUTES:

	ENTRY POINTS:
	XDIV
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Not Applicable
ALGOL:	Not Applicable
Errors:	See XADSB

NOTES: See notes for XADSB

XMPY

PURPOSE: Interface routine which allows FORTRAN II programs to use FORTRAN IV Extended Real multiply routine .XMPY.

PROGRAM TYPE = 7

ROUTINE IS: U

ENTRY POINTS: XMPY

EXTERNAL REFERENCES: .RCNG, .XMPY

CALLING SEQUENCES:

```
JSB XMPY
DEF * + 4
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:

	ENTRY POINTS:
	XMPY
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Not Applicable
ALGOL:	Not Applicable
Errors:	See XADSB

NOTE: This routine is available in firmware. See description on page 1-6.

XPOLY

PURPOSE: Evaluate extended real polynomial: $Y = c_1 X^{n-1} + c_2 X^{n-2} + \ldots + c_{n-1} X + c_n$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:

.XPLY
XPOLY

EXTERNAL REFERENCES:

.ZRNT, .ENTP, .XADD, .XMPY, .DFER,

CALLING SEQUENCES:

```
JSB .XPLY or XPOLY
DEF * + 5
DEF Y (result)
DEF n (degree + 1)
DEF X
DEF c1 (first element of coefficient array)
```

ATTRIBUTES:

ENTRY POINTS:

	.XPLY	XPOLY
Parameters:	Extended Real, Integer	Extended Real, Integer
Result:	Extended Real	Extended Real
FORTRAN:	Not callable	Callable
FORTRAN IV:	Not callable	Callable
ALGOL:	Not callable	Callable
Errors:	If $n \leq 0$, $Y = 0$	If $n \leq 0$, $Y = 0$

NOTE: See notes for XADSB.

XSUB

PURPOSE: Interface routine which allows FORTRAN II programs to use the FORTRAN IV routine XADSB to do Extended Real subtraction.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: XSUB

EXTERNAL REFERENCES: .RCNG, .XSUB

CALLING SEQUENCES:

```
JSB XSUB
DEF *+4
DEF z (result)
DEF x
DEF y
```

ATTRIBUTES:

ENTRY POINTS:

	XSUB
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Callable
FORTRAN IV:	Not Applicable
ALGOL:	Not Applicable
Errors:	

.ABS

PURPOSE: Finds the absolute value of a double real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.ABS
EXTERNAL REFERENCES:	.CFER, .TSUB, 4ZRO, .ENTR
CALLING SEQUENCES:	JSB .ABS DEF *+3 DEF <result> DEF *x* →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.ABS
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DABS (with *y* option)
ALGOL:	Not callable
Errors:	None

NOTES: See ..TCM

.ATAN

PURPOSE: Calculate the inverse tangent of double real *x*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.ATAN
EXTERNAL REFERENCES:	TRNC, .TDIV, ..TCM, .ENTR, CRER, .FLUN, .TSUB, /ATCG
CALLING SEQUENCES:	JSB .ATAN DEF *+3 DEF <result> DEF *x* →

METHOD: The following identities are used to reduce the range of X to [-.414213,+.414213] :

```
          IDENTITY                          RANGE USED    WHERE:
ATAN(X) = -ATAN(-X)                         [-INF,-.414]  C1 = +.445452376106737266D2
ATAN(X) = PI/4 - ATAN((1-X)/(1+X))          [.414,2.414]  C2 = +.774832800120330864D2
ATAN(X) = PI/2 - ATAN(1/X)                  [2.414,+INF]  C3 = +.409713682601679458D2
                                                          C4 = +.666072298720980281D1
on this range, the following approximation is used:       C5 = +.158970310916497573D0
                                                          C6 = +.445452376106737267D2
              C1+XSQ*(C2+XSQ*(C3+XSQ*(C4+XSQ*C5)))        C7 = +.923316925489242028D2
ATAN(X) = X * ------------------------------------        C8 = +.628395515876957856D2
              C6+XSQ*(C7+XSQ*(C8+XSQ*(C9+XSQ)))           C9 = +.155045070449078784D2
                                                          XSQ = X*X
```

ATTRIBUTES:

	ENTRY POINTS:
	.ATAN
Parameters:	Double real radians
Result:	Not callable
FORTRAN:	Function: DATAN (with Y option)
FORTRAN IV:	Not callable
ALGOL:	None
Errors:	None

NOTES:

.ATN2

PURPOSE: Calculate the arctangent of the quotient *x*/*y* of two double real variables *x* and *y*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.ATN2, .ATA2
EXTERNAL REFERENCES:	.ATAN, .TADD, .TSUB, .TDIV, .ENTR, .4ZERO, .CFER

CALLING SEQUENCES:

```
JSB .ATN2
DEF *+4
DEF <result>
DEF x
DEF y
<error return>
→normal return
```

METHOD: The signs of *x* and *y* are used to place the result in the proper quandrant.

ATTRIBUTES:

	ENTRY POINTS:
	.ATN2, .ATA2
Parameters:	Double real
Result:	Double real (radians)
FORTRAN:	Not callable
FORTRAN IV:	Function: DATN2 or DATAN2 (with Y option)
ALGOL:	Not callable
Errors:	$x = y = 0$ gives error code 15 UN

NOTES:

.BLE

PURPOSE: Convert real to double real.

PROGRAM TYPE = 7

ROUTINE IS: U

ENTRY POINTS: .BLE

EXTERNAL REFERENCES: .ENTR

CALLING SEQUENCES:

```
JSB  .BLE
DEF  *+3
DEF  <result>
DEF  x
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.BLE
Parameters:	Real
Result:	Double Real
FORTRAN:	Not callable
FORTRAN IV:	Function: DBLE (with Y option)
ALGOL:	Not callable
Errors:	None

NOTES:

.CADD

PURPOSE: Add complex x to complex y: $z = x + y$ (z is complex)

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	.CADD
EXTERNAL REFERENCES:	.ENTC, .ZRNT, .FAD
CALLING SEQUENCES:	JSB .CADD DEF z (result) DEF x DEF y →

ATTRIBUTES:

	ENTRY POINTS:
	.CADD
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Overflow bit set if result out of range.

Note: See OVF function for testing results.

.CDBL

PURPOSE: Extracts the real part of a complex *x* and returns it as an extended precision real *y*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .CDBL

EXTERNAL REFERENCES: DBLE

CALLING SEQUENCES:

```
JSB .CDBL
DEF y (DP result)
DEF x (complex)
→
```

ATTRIBUTES:

ENTRY POINTS:

	.CDBL
Parameters:	Complex
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.CDIV

PURPOSE: Divide complex *X* by complex *Y*: *Z* = *X*/*Y*

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	.CDIV
EXTERNAL REFERENCES:	.ZRNT, .ENTC,
CALLING SEQUENCES:	JSB .CDIV DEF *Z* (result) DEF *X* DEF *Y* →

ATTRIBUTES:

	ENTRY POINTS:
	.CDIV
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Overflow bit set if result out of range.

.CFER

PURPOSE: Moves four words from address *x* to address *Y*. Used to transfer a complex *x* to complex *Y*.

PROGRAM TYPE = 6 **ROUTINE IS: U**

ENTRY POINTS: .CFER

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB .CFER
DEF Y
DEF X
→
A = direct address of (X +4)
B = direct address of (Y +4)
```

ATTRIBUTES:

ENTRY POINTS:

	.CFER
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.CHEB

PURPOSE: Evaluate the Chebyshev series at a real *x* for a particular table of coefficients *c*.

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.CHEB
EXTERNAL REFERENCES:	.ZRNT, .FAD, .FMP, .FSB
CALLING SEQUENCES:	DLD *x* JSB .CHEB DEF *c* (table, note 1) → result in A & B

METHOD: $T_i = 2 \cdot T_{i-1} - T_{i-2} + c_{n-i}$ (i = 0,1,....,n-1)

where

$T_{-2} = T_{-1} = 0$

n = number of coefficients

Answer = $\frac{T_{n-1} - T_{n-3}}{2}$

ATTRIBUTES:

ENTRY POINTS:

	.CHEB
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTE: Table *c* consists of a series of real coefficients terminated by an integer zero.

.CINT

PURPOSE: Convert the real part of a complex *x* to an integer.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.CINT
EXTERNAL REFERENCES:	IFIX
CALLING SEQUENCES:	JSB .CINT DEF *x* →result in A

ATTRIBUTES:

	ENTRY POINTS:
	.CINT
Parameters:	Complex
Result:	Integer in A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.CMPY

PURPOSE: Multiply complex x by complex y: $z = x . y$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS: .CMPY

EXTERNAL REFERENCES: .ZRNT, .ENTC,

CALLING SEQUENCES:

```
JSB .CMPY
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:

ENTRY POINTS:

	.CMPY
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Overflow bit set if result out of range.

.CMRS

PURPOSE: Reduce argument for SIN, COS, TAN, EXP

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.CMRS
EXTERNAL REFERENCES:	.ZPRV, .XMPY, .XSUB, SNGL, IFIX, FLOAT

CALLING SEQUENCES:

```
DLD x                  (real)
JSB .CMRS
DEF CONST              (extended precision)
DEF N                  (integer, also result)
→ error return
→ normal return        (real result in A and B)
```

METHOD: The argument is converted to extended precision and multiplied by the constant. The nearest even integer, N, to this value is found. N is then converted to extended precision and subtracted from the above product. The result is rounded to single precision.

ATTRIBUTES:

	ENTRY POINTS:
	.CMRS
Parameters:	Real
Result:	Real and extended precision
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	N outside $[-2^{15},2^{15})$ gives error return

NOTES:

.CSUB

PURPOSE: Subtract complex *y* from complex *x*: $z = x - y$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	.CSUB
EXTERNAL REFERENCES:	.ENTC, .ZRNT,
CALLING SEQUENCES:	JSB .CSUB DEF *z* (result) DEF *x* DEF *y* →

ATTRIBUTES:

	ENTRY POINTS:
	.CSUB
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Overflow bit set if result out of range.

.CTBL

PURPOSE: Convert a complex real to a double real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.CTBL
EXTERNAL REFERENCES:	.BLE
CALLING SEQUENCES:	JSB .CTBL DEF <result> DEF <argument>

METHOD: The real part of the argument is converted to double real using .BLE.

ATTRIBUTES:

	ENTRY POINTS:
	.CTBL
Parameters:	Complex real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

.CTOI

PURPOSE: Raise a complex x to an integer power I: $z = x^I$ (z is complex).

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: .CTOI

EXTERNAL REFERENCES: .CMPY, .CDIV, .CFER, .ENTC, .ZRNT

CALLING SEQUENCES:

```
JSB .CTOI
DEF z (result)
DEF x
DEF I
→ Error Return
→ Normal Return
```

METHOD:

See .RTOI

ATTRIBUTES:

	ENTRY POINTS:
	.CTOI
Parameters:	Complex & integer
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	$x = 0$, $I \leq 0$ → (14 UN)

.DCPX

PURPOSE: Converts an extended real *x* to a complex *y*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .DCPX

EXTERNAL REFERENCES:

SNGL
CMPLX

CALLING SEQUENCES:

```
JSB .DCPX
DEF Y
DEF X
→
```

ATTRIBUTES:

ENTRY POINTS:

	.DCPX
Parameters:	Extended real
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.DFER

PURPOSE: Extended real transfer: *Y* = *X*; three word move.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.DFER
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB .DFER DEF *Y* DEF *X* → A = direct address of *X*+3 B = direct address of *Y*+3

ATTRIBUTES:

	ENTRY POINTS:
	.DFER
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Note: This routine is available in firmware (see note on pg. 1-6)
.DFER (2100 MICROCODE) returns *X*+4, *Y*+4 in A,B registers
.DFER (21MX MICROCODE) returns *X*+3, *Y*+3 in A,B registers.

.DINT

PURPOSE: Converts a double real *x* to an integer. $|\text{result}| \leqslant |x|$

PROGRAM TYPE = 6 **ROUTINE IS: U**

ENTRY POINTS:	.DINT, .XFTS
EXTERNAL REFERENCES:	SNGM, IFIX, .ZPRV
CALLING SEQUENCES:	JSB .DINT DEF *x* → result in A

ATTRIBUTES:	**ENTRY POINTS:**
	.DINT
Parameters:	Double Real
Result:	Integer in A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.DIV

PURPOSE: DOS-III routine to replace the subroutine call with the hardware instruction to divide a two-word integer *I* by the one-word integer *J*: $K = I/J$

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DIV
EXTERNAL REFERENCES:	.MAC.
CALLING SEQUENCES:	DLD *I* JSB DIV DEF *J* → result in A, remainder in B

ATTRIBUTES:

	ENTRY POINTS:
	.DIV
Parameters:	Two-word integer (Note 1), integer
Result:	Integer quotient in A and remainder in B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	-32768 > quotient > 32767 → overflow, quotient ← 32767

NOTES:

1. The DLD loads the two-word value I into the A and B registers with the sign and 15 most significant bits in B and the least significant bits in A.
2. Since the subroutine call is replaced by the hardware instructions, the routine is entered only once for each subroutine call.

.DLD

PURPOSE: DOS-III routine to replace the subroutine call with the hardware instruction to load the contents of memory locations *x* and *x*+1 into the A and B registers, respectively.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DLD
EXTERNAL REFERENCES:	.MAC.
CALLING SEQUENCES:	JSB .DLD or DLD *x* DEF *x*

ATTRIBUTES:

	ENTRY POINTS:
	.DLD
Parameters:	Two-word quantity
Result:	Two-word quantity: A & B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: Since the subroutine call is replaced by the hardware instruction, the routine is entered only once for each subroutine call.

.DST

PURPOSE: DOS-III routine to replace the subroutine call with the hardware instruction to store the contents of the A and B registers in memory locations *x* and *x+1*, respectively.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DST
EXTERNAL REFERENCES:	.MAC.
CALLING SEQUENCES:	JSB .DST DEF *x* →

ATTRIBUTES:

	ENTRY POINTS:
	.DST
Parameters:	Two-word quantity: A & B
Result:	Two-word quantity
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: Since the subroutine call is replaced by the hardware instruction, the routine is entered only once for each subroutine call.

.DTOD

PURPOSE: Raise a double real x to a double real power y:

$z = x^y$ (z is double real)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.DTOD
EXTERNAL REFERENCES:	DEXP, DLOG .XMPY, .DFER, .ENTC, .ZRNT
CALLING SEQUENCES:	JSB .DTOD DEF z (result) DEF x DEF y → error return → normal return

METHOD:

If $x = 0$ and $y>0$, $z = 0$.
If $x \neq 0$ and $y = 0$, $z = 1$.
If $x>0$ and $y \neq 0$, $z = \text{EXP}(y*\log(x))$

Accuracy depends on the accuracy of DLOG and DEXP.

ATTRIBUTES:	**ENTRY POINTS:**
	.DTOD
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Note

NOTE:

$x = 0,\ y \leq 0$ → (13 UN)
$x<0,\ y \neq 0$ → (13 UN)
$x>(1-2^{-39})2^{127}$ → (10 OF)

.DTOI

PURPOSE: Calculate an extended real X raised to an integer power I:

$Y = X^I$ (Y is extended real)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.DTOI
EXTERNAL REFERENCES:	.XMPY, .XDIV, .DFER, .ZRNT
CALLING SEQUENCES:	JSB .DTOI DEF Y (result) DEF X DEF I → Error return → Normal return

METHOD:

See .RTOI

ATTRIBUTES:

	ENTRY POINTS:
	.DTOI
Parameters:	Extended real & integer
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	If $X = 0$, $I \leq 0$ → (12 UN)

.DTOR

PURPOSE: Raise a double real x to a real power y:

$z = x^y$ (z is double real)

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	.DTOR
EXTERNAL REFERENCES:	.DTOD DBLE
CALLING SEQUENCES:	JSB .DTOR DEF z (result) DEF x DEF y → error return → normal return

METHOD:

Convert y to double precision and call .DTOD.

ATTRIBUTES:

	ENTRY POINTS:
	.DTOR
Parameters:	Real & double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .DTOD

.EXP

PURPOSE: Calculate e^x where x is double real

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.EXP
EXTERNAL REFERENCES:	.ENTR, .CFER, .4ZRO, /CMRT, /EXTH

CALLING SEQUENCES:

```
JSB .EXP
DEF *+3
DEF <result>
DEF x
<error return>
→
```

METHOD:

The range is reduced to [-.5, .5] using the identity EXP(X) = 2^N x 2^Z where $Z = \frac{X}{LN(2)} - N$ and N is chosen to minimize $|Z|$. Then /EXTH is called to compute $2^N x 2^Z$.

ATTRIBUTES:

	ENTRY POINTS:
	.EXP
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DEXP (with Y option)
ALGOL:	Not callable
Errors:	x > 127*LN(2) gives error code 07 OF

NOTES: For x < -129*LN(2), a zero will be returned with no error indication.

.FDV

PURPOSE: Divide real *X* by *Y*: *Z* = *X*/*Y*

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: .FDV

EXTERNAL REFERENCES: .PACK, .ZPRV

CALLING SEQUENCES:

```
DLD X
JSB .FDV
DEF Y
→ quotient in A & B  O - set if under/overflow
```

ATTRIBUTES:

	ENTRY POINTS:
Call:	.FDV
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See FADSB

.FLUN

PURPOSE: "Unpack" a real *x*; place exponent in A, lower part of mantissa in B.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.FLUN
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	DLD *x* JSB .FLUN → exponent in A Lower mantissa in B

ATTRIBUTES:

	ENTRY POINTS:
	.FLUN
Parameters:	Real
Result:	A & B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTE: This routine is available in 21MX FFP. See note on page 1-6.

.FMP

PURPOSE: Multiply real *x* by *y*: $z = x*y$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.FMP
EXTERNAL REFERENCES:	.PACK, .ZPRV
CALLING SEQUENCES:	DLD *y* JSB .FMP DEF *x* → product in A & B

ATTRIBUTES:

	ENTRY POINTS:
Call:	.FMP
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See FADSB

.FPWR

PURPOSE: Calculates X^I for real X and unsigned integer I.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.FPWR
EXTERNAL REFERENCES:	.FMP , FLOAT , .FLUN
CALLING SEQUENCES:	LDA *I* JSB .FPWR DEF *x* → (result in A & B)

METHOD:

The left-to-right binary method is used. The result is first set to *x*. Then for each bit after the highest bit set in I:

a) square the result.
b) if the current bit is set, multiply the result by the argument.

ATTRIBUTES:

	ENTRY POINTS:
	.FPWR
Parameters:	Real , integer
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1) "I" must be in the range [2,32768].
2) If overflow occurs, the maximum positive number is returned with overflow set. Overflow is set if underflow occurs.
3) The X and Y registers may be altered.

.ICPX

PURPOSE: Converts an integer *I* to a complex *Y*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.ICPX
EXTERNAL REFERENCES:	FLOAT CMPLX

CALLING SEQUENCES:

```
LDA I
JSB .ICPX
DEF Y
→
```

ATTRIBUTES:	**ENTRY POINTS:**
	.ICPX
Parameters:	Integer in A
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.IDBL

PURPOSE: Converts an integer *I* to extended real *Y*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.IDBL, .XFTS
EXTERNAL REFERENCES:	FLOAT, DBLE
CALLING SEQUENCES:	LDA *I* JSB .IDBL DEF *Y* →

ATTRIBUTES:

	ENTRY POINTS:
	.IDBL
Parameters:	Integer in A
Result:	Extended
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.IENT

PURPOSE: Calculate the greatest integer not algebraically exceeding a real *x*: *I* = ENTIER (*x*).

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.IENT
EXTERNAL REFERENCES:	IFIX, .FLUN, FLOAT, .ZPRV
CALLING SEQUENCES:	DLD *x* JSB .IENT JSB *error routine* → *I* in A

ATTRIBUTES:

	ENTRY POINTS:
	.IENT
Parameters:	Real
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	EXPO (*x*) > 14, user must supply error routine

.ITBL

PURPOSE: Convert integer to double real

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:	.ITBL, .TFTS
EXTERNAL REFERENCES:	.BLE, FLOAT
CALLING SEQUENCES:	LDA *x* JSB .ITBL DEF <result> →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.ITBL
Parameters:	Integer
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

.ITOI

PURPOSE: Calculate I^J for integer I and J: $K = I^J$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.ITOI
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB .ITOI DEF I DEF J JSB ERR0 (error return) → K in A

ATTRIBUTES:

	ENTRY POINTS:
	.ITOI
Parameters:	Integer
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable

Errors:

Condition	Error Code
$I = 0,\ J \leq 0$	08 UN
$I^J \geq 2^{15}$ or $I^J < -2^{15}$	08 OF

.LBT

PURPOSE: Replaces 21MX microcoded instruction LBT.

PROGRAM TYPE = 6

ROUTINE IS: P

ENTRY POINTS: .LBT

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
LDB X
JSB .LBT
```

METHOD: Bits 0 to 7 of the location specified by X are loaded into bits 0 to 7 of the A-reg. X must be a byte address. Bits 8 - 15 of A are cleared. The B register is incremented by 1.

ATTRIBUTES:

	ENTRY POINTS:
	.LBT
Parameters:	Integer
Result:	A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Note: A byte address is defined as two times the word address of the memory location containing the byte of data. If the byte is in bits 0 to 7, bit 0 of the byte address is set; if the byte is in bits 8 - 15, bit 0 of the byte address is clear.

.LOG

PURPOSE: Calculate the natural logarithm of double real X

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.LOG
EXTERNAL REFERENCES:	.ENTR, .CFER, .FLUN, .TADD, .TMPY, TRNL, /ATLG, FLOAT

CALLING SEQUENCES:

```
JSB .LOG
DEF *+3
DEF <result>
DEF x
<error return>
→
```

METHOD: The identity:

LN(X) = N*LN(2) + LN(X/LN(2)- N)

Is used to reduce the range to [.707,1.414] · on this range, the following approximation is used:

$$LN(Y) = Z * \frac{C1+ZSQ*(C2+ZSQ*C3)}{C4+ZSQ*(C5+ZSQ*(C5+ZSQ))}$$

WHERE:
Y = reduced X
Z = (1-Y) / (1+Y)
C1 = +.903435497728419518D2
C2 = -.935961251529860988D2
C3 = +.183395455436327320D2
C4 = -.451717748864209816D2
C5 = +.618553208719806812D2
C6 = -.207538580906546412D2
ZSQ = Z*Z

ATTRIBUTES:

	ENTRY POINTS:
	.LOG
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DLOG (with Y option)
ALGOL:	Not callable
Errors:	X < 0 → 02 UN

NOTES:

.LOG0

PURPOSE: Calculate the common (base 10) logarithm of double real *x*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .LOG0 (.LOGT)

EXTERNAL REFERENCES: .LOG, .TMPY, .ENTR

CALLING SEQUENCES:

```
JSB .LOG0 (.LOGT)
DEF *+3
DEF <result>
DEF x
<error return>
→
```

METHOD: $Y = LOG_{10}(x) = LOG_e(x) * LOG_{10}(e)$

ATTRIBUTES:

	ENTRY POINTS:
	.LOG0 (or .LOGT)
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function DLOGT (or DLOG10) (with Y option)
ALGOL:	Not callable
Errors:	$x \leq 0$ gives error code 02 UN

NOTES:

.MAC.

PURPOSE: Replaces a JSB *.subr* with a machine language Macro jump 105*nnn* that initiates firmware.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.MAC.
EXTERNAL REFERENCES:	
CALLING SEQUENCES:	*.subr* NOP JSB .MAC. OCT 105*nnn* END Where *nnn* is between 000 and 377

METHOD: Before execution of the subroutine jump to *.subr*, the program holds the standard calling sequence for the software subroutine *.subr*:

```
JSB .subr
DEF x
etc.
```

If *.subr* contains the .MAC. call as shown in the calling sequence above, the subroutine jump to the software subroutine *.subr* is replaced with the macro instruction that executes the *.subr* function in firmware:

```
OCT 105nnn
DEF x
etc.
```

ATTRIBUTES:

	ENTRY POINTS:
	.MAC.
Parameters:	Address
Result:	In-line code change
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: The same result is achieved in RTE during system generation using a replace command.

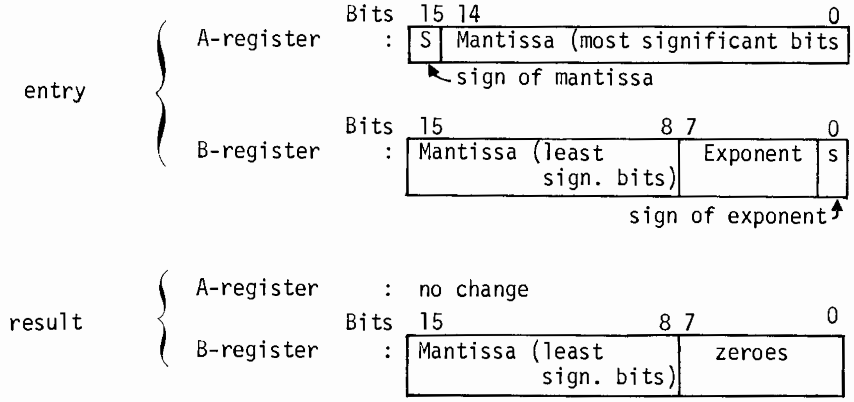
.MANT

PURPOSE: Extract mantissa of a real *x*

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.MANT
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	DLD *x* JSB .MANT → Real Mantissa in A & B

METHOD:



ATTRIBUTES:

ENTRY POINTS:

	.MANT
Parameters:	Real
Result:	Real
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	None

.MOD

PURPOSE: Calculate the remainder of X/Y, where X,Y and result are double reals.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.MOD
EXTERNAL REFERENCES:	.CFER, .TSUB, .TMPY, .TDIV .YINT, .ENTR, .4ZRO

CALLING SEQUENCES:

```
JSB .MOD
DEF * +4
DEF <result>
DEF X
DEF Y
→
```

METHOD: RESULT ← X -[.YINT(X/Y)]*Y

ATTRIBUTES:

	ENTRY POINTS:
	.MOD
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DMOD (with Y option)
ALGOL:	Not callable
Errors:	If Y=0 then the result is zero

NOTES:

1) The function .MOD will return X if Y=0, or X/Y overflows or underflows.
2) If an overflow or underflow occurs elsewhere in the calculation, the result will be incorrect.
3) No attempt is made to recover precision lost in the subtract.

.MPY

PURPOSE: DOS-III routine to replace the subroutine call with the hardware instruction to multiply integer *I* and *J*: *K* = *I***J*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.MPY
EXTERNAL REFERENCES:	.MAC.
CALLING SEQUENCES:	LDA *J* JSB .MPY DEF *I* → *K* in A&B (Note 1)

ATTRIBUTES:

	ENTRY POINTS:
	.MPY
Parameters:	Integer
Result:	Two-word integer (Note 1)
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1. B contains most significant bits of product; A contains least significant bits.

2. Since the subroutine call is replaced by the hardware instruction, the routine is called only once for each subroutine call.

.MXMN

PURPOSE: Find the maximum (or minimum) of a list of double reals.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.MAX1	MIN1
EXTERNAL REFERENCES:	.CFER, . TSUB , .4ZRO	

CALLING SEQUENCES:

```
JSB .MAX1          JSB .MIN1
DEF * + N+2        DEF * + N+2
DEF <result>       DEF <result>
DEF A(1)           DEF A(1)
DEF A(2)           DEF A(2)
 :                  :
DEF A(N)           DEF A(N)
→                  →
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:	
	.MAX1	.MIN1
Parameters:	Double reals	Double Reals
Result:	Double real	Double real
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Function: DMAX1 (with Y option)	Function: DMIN1(with Y option)
ALGOL:	Not callable	Not callable
Errors:	None	

NOTES:

If there is only one argument in the list, it is considered to be both the maximum and minimum of the list.

If the list is null, zero will be returned.

.NGL

PURPOSE: Convert double real to real.

PROGRAM TYPE =7 **ROUTINE IS:U**

ENTRY POINTS:	.NGL
EXTERNAL REFERENCES:	SNGL, .CFER
CALLING SEQUENCES:	JSB .NGL DEF *+2 DEF x →result in A & B

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.NGL
Parameters:	Double real
Result:	Real in A & B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: The result is rounded unless this would cause overflow. If so, overflow is set and the result is truncated to the greatest positive number.

.PACK

PURPOSE: Convert signed mantissa of real *x* into normalized real format.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.PACK
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	DLD *x* JSB .PACK BSS 1 (exponent) → result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	.PACK
Parameters:	Mantissa in A & B
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTE: This routine is available in 21MX FFP. See note on page 1-6.

.PWR2

PURPOSE: Calculate for real x and integer n: $Y = x \cdot 2^n$

PROGRAM TYPE = 6	**ROUTINE IS: P**
ENTRY POINTS:	.PWR2
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	DLD x JSB .PWR2 DEF n → Y in A & B

METHOD:

Exponent of x is increased by n.
Accuracy is 23 bits.

ATTRIBUTES:

	ENTRY POINTS:
	.PWR2
Parameters:	Real & Integer
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Notes: 1. This routine is available in 21MX FFP firmware. See note on page 1-6.

.RTOD

PURPOSE: Raise a real *x* to a double real power *y*: $z=x^{y}$ (*z* is double real)

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:

.RTOD

EXTERNAL REFERENCES:

.DTOD
DBLE

CALLING SEQUENCES:

```
JSB .RTOD
DEF z (result)
DEF x
DEF y
→ Error Return
→ Normal Return
```

METHOD: Convert *x* to double real and call .DTOD.

ATTRIBUTES:

ENTRY POINTS:

	.RTOD
Parameters:	Real and Double Real
Result:	Double Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .DTOD

.RTOI

PURPOSE: Calculate X^I for real X and integer I: $Y=X^I$.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .RTOI

EXTERNAL REFERENCES: .FPWR, .FDV

CALLING SEQUENCES:

```
JSB .RTOI
DEF X
DEF I
JSB ERR0
→ Y in A & B
```

METHOD: The only possibility of inaccuracy is that introduced by roundoff in the floating multiplies (and divide if $I < 0$).

The left-to-right binary method is used (see .FPWR). If $I < 0$ the result is $1.0/(X^{-I})$ In general, the result is slightly different (due to roundoff error) from
$X*X*X \; . \; . \; . \; *X$
I-1 times

ATTRIBUTES:

ENTRY POINTS:

	.RTOI	
Parameters:	Real & Integer	
Result:	Real	
FORTRAN:	Not Callable	
FORTRAN IV:	Not Callable	
ALGOL:	Not Callable	
Errors:	Condition	Error Code
	$x = 0, I \le 0$	06 UN
	$x^{\lvert I \rvert} > 2^{127}$	(floating point overflow)

.RTOR

PURPOSE: Calculate x^Y for real x and Y: $z = x^Y$

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS: .RTOR

EXTERNAL REFERENCES: ALOG, EXP, .ZRNT, .FMP

CALLING SEQUENCES:

```
JSB .RTOR
DEF x
DEF Y
JSB ERR0
→ z in A & B
```

ATTRIBUTES:

	ENTRY POINTS:
	.RTOR
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	

Condition	Error Code
$x = 0,\ Y \le 0$ $< 0,\ \ne 0$	04 UN
$\lvert x * \text{ALOG}(x) \rvert \ge 124$	07 OF

On error return, the overflow bit is set.

.RTOT

PURPOSE: Calculate X^Y, where X is a real and Y is a double real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .RTOT

EXTERNAL REFERENCES: .TTOT

CALLING SEQUENCES:

```
JSB .RTOT
DEF <result>
DEF x
DEF Y
<error return>
→
```

METHOD: *x* is converted to double real, then .TTOT is called.

ATTRIBUTES:

	ENTRY POINTS:
	.RTOT
Parameters:	Real *x* , double real Y
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See note

NOTES:

Underflow will give a zero result, with no error. Overflow returns the greatest positive number, sets overflow (cleared otherwise), and gives an error code of 07 OF.

If $(x<0)$ or $(x=0$ and $y\leq0)$ there will be an error code of 13 UN.

.SBT

PURPOSE: Replaces 21MX microcoded instruction SBT.

PROGRAM TYPE = 6 **ROUTINE IS:** P

ENTRY POINTS:	.SBT
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	LDB X JSB .SBT

METHOD:

Bits 0 - 7 of the A-reg are copied into the location specified by X. X must be a byte address. The B register is incremented by 1. A-reg bits 8 - 15 are ignored. The A-reg is unchanged by this routine.

ATTRIBUTES:

	ENTRY POINTS:
	.SBT
Parameters:	Integer
Result:	A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Note: A byte address is defined as two times the word address of the memory location containing the byte of data. If the byte is in bits 0 to 7, bit 0 of the byte address is set; if the byte is in bits 8 - 15, bit 0 of the byte address is clear.

.SIGN

PURPOSE: Transfer the sign of a double real y to a double real x

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.SIGN
EXTERNAL REFERENCES:	.CFER, .TSUB, 4ZRO, .ENTR

CALLING SEQUENCES:

```
JSB .SIGN
DEF *+4
DEF  result
DEF x
DEF y
→
```

METHOD: $|x|$. sign (y)

ATTRIBUTES:

	ENTRY POINTS:
	.SIGN
Parameters:	Double reals
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DSIGN (with Y option)
ALGOL:	Not callable
Errors:	None

NOTES:

1) Overflow will be set or cleared depending on occurrence. (Overflow only occurs if $y \geq 0$ and x is the maximum negative number)

2) .SIGN$(x,0)=|x|$

.SNCS

PURPOSE: Calculate the sine or cosine of real X (radians)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	SIN	COS
EXTERNAL REFERENCES:	.ZPRV, .CMRS, ..FCM, .FMP, .FAD	
CALLING SEQUENCES:	DLD *x* JSB SIN <error return> → *y* (in A & B)	DLD *x* JSB COS <error return> → *y* (in A & B)

METHOD: The argument is reduced to the range [-π/4,π/4] using the identities:

```
SIN(X)=SIN(X-2*K*π)
COS(X)=COS(X-2*K*π)
SIN(X)=-SIN(X-π)
COS(X)=-COS(X-π)
SIN(X)=COS(X-π/2)
COS(X)=SIN(X+π/2)
```

THEN
the following approximations are used

```
SINE(W) = X*(S1+XSQ*(S2+XSQ*(S3+XSQ*S4)))
COSINE(Z) = C1+YSQ*(C2+YSQ*(C3+YSQ*C4))
```

WHERE:

```
X=W*(4/π)
Y=Z*(4/π)

XSQ = X**2                  YSQ = Y**2
S1 = .78539816              C1 = 1.0
S2 = -.0807454325           C2 = -.30842483
S3 = .002490001             C3 = .015851077
S4 = -.000035950439         C4 = -.00031957
```

ATTRIBUTES:

	ENTRY POINTS:	
	SIN	COS
Parameters:	Real (radians)	Real (radians)
Result:	Real	Real
FORTRAN:	Function: SIN(X)	Function: COS(X)
FORTRAN IV:	Function: SIN(X)	Function: COS(X)
ALGOL:	Intrinsic proc. SIN(X)	Intrinsic proc. COS(X)
Errors:	X outside [-8192*π ,+8191.75*π] →	050R

NOTES:

.SQRT

PURPOSE: Calculate the square root of double real *x*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .SQRT

EXTERNAL REFERENCES:

.ENTR	.CFER	.PWRZ	.TDJV
.XADD	.XDIV	.TADD	.SQRT

CALLING SEQUENCES:

```
JSB .SQRT
DEF *+3
DEF <result>
DEF x
<error return>
→
```

METHOD: The initial approximation is (single prec.) X_0 = SQRT(*y*)

The Heron's rule is applied twice:

$$X_1 = .5*(X_0+Y/X_0)$$

$$X_2 = .5*(X_1+Y/X_1)$$

ATTRIBUTES:

ENTRY POINTS:

	.SQRT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DSQRT (with Y option)
ALGOL:	Not callable
Errors:	X < 0 gives error code 03 UN

NOTES:

.TAN

PURPOSE: Calculate the tangent of double real X (radians):

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TAN
EXTERNAL REFERENCES:	.ENTR, /CMRT, TRNL, .TDIV
CALLING SEQUENCES:	JSB .TAN DEF *+3 DEF <result> DEF x <error return> →

METHOD: X is reduced to the range (-π/4, +π/4) (see TAN). Then the following formula is used:

$$\text{TANGENT}(X) = Z * \frac{C1+Z^2*(C2+Z^2*(C3+Z^2*C4))}{C5+Z^2*(C6+Z^2*(C7+Z^2))}$$

WHERE:
C1 = -.160528895723771175D+5
C2 = +.127029221227298238D+4
C3 = -.171390807007805963D+2
C4 = +.281970876313544687D-1
C5 = -.204391738108172811D+5
C6 = +.582002294049071829D+4
C7 = -.181557373390721805D+3
Z = X*4/π

ATTRIBUTES:

	ENTRY POINTS:
	.TAN
Parameters:	Double real (radians)
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DTAN (with Y option)
ALGOL:	Not callable
Errors:	X outside $[-2^{23},2^{23})$ → 09 OR

NOTES:

.TANH

PURPOSE: Calculate the hyperbolic tangent of double real *x*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .TANH

EXTERNAL REFERENCES: .ENTR,.CFER,.TADD,.TDIV,/CMRT,/EXTH,.4ZRO

CALLING SEQUENCES:

```
JSB .TANH
DEF  *+3
DEF <result>
DEF  x
→
```

METHOD: The identities

TANH(X) = (EXP(2*X)-1)/(EXP(2*X+1)

EXP(X) = (2**N)*(2**(x/LN(2)-N))

are used to reduce the problem with /CMRT so that Y=X/LN(2)-N is minimized. Then /EXTH is called to calculate $2^N \cdot 2^Y = e^X$ and TANH is computed. If N=0, /EXTH computes TANH instead of 2^Y. If N is outside [-32,32) TANH returns SIGN(1,X).

ATTRIBUTES:

	ENTRY POINTS:
	.TANH
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function DTANH (with Y option)
ALGOL:	Not callable
Errors:	None

NOTES:

.TCPX

PURPOSE: Convert double real to complex real. The second value is set to zero.

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	.TCPX
EXTERNAL REFERENCES:	.NGL

CALLING SEQUENCES:

```
JSB .TCPX
DEF <result>
DEF x
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.TCPX
Parameters:	Double real
Result:	Complex real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: The result is rounded unless this would cause overflow. If so, overflow is set and the result is trunctuated to the greatest positive number.

.TENT

PURPOSE: Finds the greatest integer less than or equal to a double real (floor *x*).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .TENT

EXTERNAL REFERENCES: .FLUN, .ENTR, .CFER

CALLING SEQUENCES:

```
JSB .TENT
DEF *+3
DEF <result>
DEF x
→
```

METHOD: All mantissa bits after the binary point are set to zero.

ATTRIBUTES:

ENTRY POINTS:

	.TENT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: Result is a double real value with no bits set after the binary point.

"THIS PAGE WAS INTENTIONALLY LEFT BLANK"

.TINT

PURPOSE: Convert double real to integer

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TINT, .TFXS
EXTERNAL REFERENCES:	IFIX
CALLING SEQUENCES:	JSB .TINT DEF *x* → (*y* in A)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.TINT, .TFXS
Parameters:	Double real
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: 1) If the argument is outside the range $[-2^{15},2^{15})$ the result is $2^{15}-1$ and overflow is set. Overflow is cleared otherwise.

.TMTH

PURPOSE: Double real arithmetic

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .TADD .TSUB .TMPY .TDIV

EXTERNAL REFERENCES: .FLUN, .XFER, .CFER, FLOAT

CALLING SEQUENCES:

```
JSB .TADD   or   .TSUB   or   .TMPY   or    .TDIV
DEF z
DEF x
DEF y
```

FUNCTION: z=x+y z=x-y z=x*y z=x/y

ATTRIBUTES:	ENTRY POINTS:
	.TADD, .TSUB, .TMPY, .TDIV
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	

NOTES:

If underflow occurs, zero is returned with overflow set. If overflow or divide by zero occurs, the largest positive number is returned with overflow set. Otherwise, overflow is cleared.

.TPWR

PURPOSE: Calculates X^I, where x is a double real and I is an unsigned integer.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TPWR
EXTERNAL REFERENCES:	.TMPY,FLOAT,.FLUN,.CFER

CALLING SEQUENCES:

```
LDA  < I >
JSB  .TPWR
DEF  <result>
DEF  x
```

METHOD:

See .FPWR

ATTRIBUTES:

	ENTRY POINTS:
	.TPWR
Parameters:	Double real and integer
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: See .FPWR

.TSCS

PURPOSE: Calculate the sine or cosine of double precision Y (radians)

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.SIN	.COS
EXTERNAL REFERENCES:	.ENTR, /CMRT,	TRNL, .TDIV

CALLING SEQUENCES:

```
JSB .SIN                                JSB .COS
DEF *+3                                 DEF *+3
DEF <result>                            DEF <result>
DEF y                                   DEF y
<error return>                          <error return>
→                                       →
```

METHOD: The range is reduced to (-1,1) using /CMRT with the same identities used in .SNGS.

The approximations used for sine and cosine on [-1,+1] are:

$$\mathrm{SINE}(Y) = X * \frac{S1+XSQ*(S2+XSQ*(S3+XSQ*S4))}{S5+XSQ*(S6+XSQ*(S7+XSQ)}$$

$$\mathrm{COSINE}(Y) = \frac{C1+XSQ*(C2+XSQ*(C3+XSQ*C4))}{C5+XSQ*(C6+XSQ*(C7+XSQ))}$$

WHERE:
S1 = +.206643399057353636D7
S2 = -.181603957072347052D6
S3 = +.359993003561793397D4
S4 = -.201074790195269777D2
S5 = +.263106547338311489D7
S6 = +.392702372048540481D5
S7 = +.278119167978678163D3
C1 = +.129054063552079782D7
C2 = -.374567381232715042D6
C3 = +.134323138925688837D5
C4 = -.112314630290509841D3
C5 = +.129054063552079782D7

- -

C6 = +.234677917710655242D5
C7 = +.209695300876930826D3
X = Y*4/π
XSQ = X*X

ATTRIBUTES:

	ENTRY POINTS:	
	.SIN	.COS
Parameters:	Double real (radians)	Double real (radians)
Result:	Double real	Double real
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Function: DSIN (with Y option)	Function: DCOS (with Y option)
ALGOL:	Not callable	Not callable
Errors:	X outside $[-2^{23},2^{23})$ → 05 OR	05OR

NOTES:

.TTOI

PURPOSE: Calculates X^I, where x is a double real and I is an integer.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TTOI
EXTERNAL REFERENCES:	.TPWR,.TDIV,.CFER,.4ZRO

CALLING SEQUENCES:

```
JSB    .TTOI
DEF    <result>
DEF    x
DEF    I
<error return>
```

METHOD:

See .RTOI and .TPWR.

ATTRIBUTES:

	ENTRY POINTS:	
	.TTOI	
Parameters:	Double real x, integer I	
Result:	Double real	
FORTRAN:	Not callable	
FORTRAN IV:	Not callable	
ALGOL:	Not callable	
Errors:	Condition	Error Code
	x=0, $I \le 0$	12 UN
	$x^{\lvert I \rvert} \ge 2^{127}$	(Floating point overflow)

.TTOR

PURPOSE: Raise a double real *x* to a real power *y*:

$z = x^{y}$ (z is double real)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.TTOR
EXTERNAL REFERENCES:	.TTOT

CALLING SEQUENCES:

```
JSB .TTOR
DEF z (result)
DEF x
DEF y
→ error return
→ normal return
```

METHOD: Converts *y* to double real and calls .TTOT.

ATTRIBUTES:

	ENTRY POINTS:
	.TTOR
Parameters:	Double real, Real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .TTOT

.TTOT

PURPOSE: Calculate X^y, where x and y are both double reals.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TTOT
EXTERNAL REFERENCES:	.LOG,.EXP,.CFER,.TMPY,.4ZRO

CALLING SEQUENCES:

```
JSB     .TTOT
DEF     <result>
DEF     x
DEF     y
<error return>
→
```

METHOD:

X^y=.EXP (Y*.LOG(X))

ATTRIBUTES:

	ENTRY POINTS:
	.TTOT
Parameters:	Double Real
Result:	Double Real
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	See Note

NOTES: Underflow will give a zero result, with no error. Overflow returns no result and gives an error code of 07 OF.

If (x<0) or (x=0 and $y \leq 0$) there will be an error code of 13 UN.

.XCOM

PURPOSE: Complements a double real unpacked mantissa in place. Upon return, A-register = 1 if exponent should be adjusted; otherwise A = 0.

PROGRAM TYPE = 6

ROUTINE IS: P

ENTRY POINTS: .XCOM

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
JSB .XCOM
DEF x
ADA (exponent)
STA (exponent)
```

ATTRIBUTES:

ENTRY POINTS:

	.XCOM
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Note: This routine is available in 21MX FFP. See note on page 1-6.

.XDIV

PURPOSE: Divide an extended real x by extended real y: $z = x / y$

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS: .XDIV

EXTERNAL REFERENCES: .ZRNT, .XPAK

CALLING SEQUENCES:

```
JSB .XDIV
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:

	ENTRY POINTS:
	.XDIV
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See XADSB

Note: This routine is available in firmware. See note on FFP of FFP on page 1-6.

.XFER

PURPOSE: Moves three words from address x to address y. Used for extended real transfers.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: .XFER

EXTERNAL REFERENCES: .DFER, .ZPRV

CALLING SEQUENCES:

```
LDA (address of x)
LDB (address of y)
JSB .XFER
→
on return:
A = direct address of x+3
B = direct address of y+3
```

ATTRIBUTES:

	ENTRY POINTS:
	.XFER
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTE: This routine is available in firmware. See description on page 1-6.

.XMPY

PURPOSE: Multiply extended real *x* by extended real *y*: $z = x*y$

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: .XMPY

EXTERNAL REFERENCES: .XPAK .ZPRV

CALLING SEQUENCES:

```
JSB .XMPY
DEF z (result)
DEF x
DEF y
→
```

ATTRIBUTES:

	ENTRY POINTS:
	.XMPY
Parameters:	Extended Real
Result:	Extended Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See XADSB

NOTES: This routine is available in firmware. See description on page 1-6.

.XPAK

PURPOSE: Double real mantissa is normalized, rounded, and packed with exponent; result is double real.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS: .XPAK

EXTERNAL REFERENCES: .ZPRV

CALLING SEQUENCES:

```
LDA exponent
JSB .XPAK
DEF x (3-word mantissa)
→ result in x
```

ATTRIBUTES:	ENTRY POINTS:
	.XPAK
Parameters:	Double real, exponent
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See XADSB

NOTE: This routine is available in 21MX FFP firmware. See note on page 1-6.

If z is outside the range: $[-2^{128}, 2^{127} (1-2^{-39})]$, then the overflow bit is set and $z = 2^{127} (1-2^{-39})$.
If the result is within the range: $[-2^{-129}(1+2^{-22}), 2^{-129}]$, then the overflow bit is set and $z = 0$.

.YINT

PURPOSE: Truncate fractional part of double real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .YINT

EXTERNAL REFERENCES: .TENT, .TADD, .ENTR

CALLING SEQUENCES:

```
JSB .YINT
DEF *+3
DEF <result>
DEF x
→
```

METHOD: Result is double real representation of the integer with the same sign as x. The maximum absolute value of the result is $\le |x|$

ATTRIBUTES:

ENTRY POINTS:

	.YINT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Function: DDINT (with *Y* option)
ALGOL:	Not callable
Errors:	None

NOTES: Result is a double real value with no bits set after the binary point.

.4ZRO

PURPOSE: Common double real zero

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.4ZRO
EXTERNAL REFERENCES:	NONE
CALLING SEQUENCES:	NONE

METHOD:

The entry point .4ZRO is the first word of a block of 4 words of value zero. This constant is used by numerous relocatable library routines.

ATTRIBUTES:

	ENTRY POINTS:
	.4ZRO - data references only
Parameters:	None
Result:	None
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	Not applicable

NOTES:

..CCM

PURPOSE: Complements a complex variable *x* in place.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: ..CCM

EXTERNAL REFERENCES: ..DLC

CALLING SEQUENCES:

```
JSB ..CCM
DEF x
→
```

ATTRIBUTES:

	ENTRY POINTS: ..CCM
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

..DCM

PURPOSE: Extended real complement in place.

PROGRAM TYPE = 6 **ROUTINE IS:** R

ENTRY POINTS:	..DCM
EXTERNAL REFERENCES:	.ZRNT, .XSUB

CALLING SEQUENCES:

```
JSB ..DCM
DEF x
→
```

ATTRIBUTES:

ENTRY POINTS:

	..DCM
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See Note 2

NOTES:

1. This routine is available in 21MX FFP. See note on page 1-6.

2. If *x* is the smallest negative number (-2^{127}), then result is the largest positive number $[(1-2^{-39})\cdot 2^{127}]$ and the overflow bit is set.

..DLC

PURPOSE: Load and complement a real *x*.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	..DLC
EXTERNAL REFERENCES:	.ZPRV, .FSB
CALLING SEQUENCES:	JSB ..DLC DEF *x* → complement in A & B

ATTRIBUTES:

	ENTRY POINTS:
	..DLC
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

..FCM

PURPOSE: Complement real *x*

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	..FCM
EXTERNAL REFERENCES:	.ZPRV, .FSB
CALLING SEQUENCES:	DLD *x* JSB ..FCM → result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	..FCM
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

..TCM

PURPOSE: Negate a double real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	..TCM
EXTERNAL REFERENCES:	.TSUB, .4ZRO
CALLING SEQUENCES:	JSB ..TCM DEF *x* →

METHOD: $x \leftarrow 0 - x$

ATTRIBUTES:

	ENTRY POINTS:
	..TCM
Parameters:	Double real
Result:	Double real - same location
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: This routine modifies the memory locations designated by x. Overflow is cleared unless x is -2^{127} in which case overflow is set and x becomes $2^{127}-2^{82}$.

DOUBLE INTEGER SUBROUTINES

FIXDR

PURPOSE: Convert real to double length record number.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	FIXDR
EXTERNAL REFERENCES:	.FIXD, .ENTR

CALLING SEQUENCES:

```
REAL X,Y,FIXDR
   .
   .
   .
Y = FIXDR(X)
```

METHOD: Calls .FIXD

ATTRIBUTES:

	ENTRY POINTS:
	FIXDR
Parameters:	Real
Result:	Double length record number (may be in real variable)
FORTRAN:	Function
FORTRAN IV:	Function
ALGOL:	Callable as real procedure
Errors:	See .FIXD

NOTES: Result is incorrect if real value is greater than $2^{31}-1$ since this is the maximum record number. Record numbers greater than 2^{23} may not be represented exactly as real numbers.

FLTDR

PURPOSE: Convert double length record number to real.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	FLTDR
EXTERNAL REFERENCES:	.FLTD, .ENTR
CALLING SEQUENCES:	REAL X,Y,FLTDR . . Y=FLTDR(X)

METHOD: Calls .FLTD

ATTRIBUTES:

	ENTRY POINTS:
	FLTDR
Parameters:	Double length record number (may be in real variable or integer array)
Result:	Real
FORTRAN:	Function
FORTRAN IV:	Function
ALGOL:	Callable as real procedure
Errors:	None

NOTES: Should not be used for record numbers exceeding 2^{23}, as the conversion may not be exact for such numbers.

.DADS

PURPOSE: Double integer add and subtract.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DAD	.DSB	.DSBR
EXTERNAL REFERENCES:			
CALLING SEQUENCES:	DLD *x* JSB .DAD DEF *y* → result in A & B	DLD *x* JSB .DSB DEF *y* → result in A & B	DLD *x* JSB .DSBR DEF *y* → result in A & B
METHOD:	$x + y$	$x - y$	$y - x$

ATTRIBUTES:

	ENTRY POINTS:
	.DAD, .DSB, .DSBR
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If overflow occurs, the least significant 32 bits are returned with overflow set. Overflow is cleared otherwise. "E" is never cleared, but is set if carry occurs (.DAD) or borrow (.DSB & DSBR).

.DSBR is used to replace the sequence:

```
DST temp                          JSB .DSBR
DLD x             WITH            DEF x
JSB .DSB
DEF temp
```

.DCO

PURPOSE: Compare two double integers.

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:	.DCO
EXTERNAL REFERENCES:	
CALLING SEQUENCES:	DLD *x* JSB .DCO DEF *y* → (if $x=y$) → (if $x<y$) → (if $x>y$)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.DCO
Parameters:	Double integers
Result:	None
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

A, B, E & O are left unchanged. The compare is correct even if X-Y is not representable in 32 bits.

.DDE

PURPOSE: Decrement the double integer in the A & B registers.

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS: .DDE

EXTERNAL REFERENCES:

CALLING SEQUENCES:

```
DLD x
JSB .DDE
→ (result in A & B)
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.DDE
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If the largest negative number is decremented, the largest positive number is the result, with overflow set. Overflow is cleared otherwise.

"E" is preserved unless X = 0, in which case it is set.

.DDI

PURPOSE: Double integer divide. Z = Z/Y.

	PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	.DDI	.DDIR
EXTERNAL REFERENCES:	FLOAT	
CALLING SEQUENCES:	DLD *x* JSB .DDI DEF *y* → (result in A & B)	DLD *y* JSB .DOIR DEF *x* → (result in A & B)
METHOD:	X/Y	Y/X

ATTRIBUTES:

	ENTRY POINTS:
	.DDI, .DDIR
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If overflow or divide by zero occur, the largest positive integer is returned with overflow set. Overflow is cleared otherwise. "E" is preserved.

.DDIR is used to replace the sequence:

```
DST temp            JSB .DDIR
DLD x       with    DEF x
JSB .DDI
DEF temp
```

.DDS

PURPOSE: Double integer decrement and skip if zero.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DDS
EXTERNAL REFERENCES:	
CALLING SEQUENCES:	JSB .DDS DEF *x* → (if x-1 ≠ 0) → (if x-1 = 0)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.DDS
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: This routine decrements the double integer *x*.
A, B, E & O are left unchanged except that A & B will be changed if the effective address is zero.

.DIN

PURPOSE: Increment the double integer in the A & B registers.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DIN
EXTERNAL REFERENCES:	
CALLING SEQUENCES:	see below

```
DLD x
JSB .DIN
→ (result in A & B)
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.DIN
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If the largest positive number is incremented, the largest negative number is the result, with overflow set. Overflow is cleared otherwise.

"E" is preserved unless X = -1, in which case "E" is set.

.DIS

PURPOSE: Double integer increment and skip if zero.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .DIS

EXTERNAL REFERENCES:

CALLING SEQUENCES:

```
JSB .DIS
DEF x
→ (if x+1 ≠ 0)
→ (if x+1 = 0)
```

METHOD:

ATTRIBUTES:

ENTRY POINTS:

	.DIS
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: This routine increments the double integer *x* by 1. A, B, E & O are left unchanged except that A & B will be changed if the effective address is zero.

.DMP

PURPOSE: Double integer multiply. Z = X * Y.

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:	.DMP
EXTERNAL REFERENCES:	None

CALLING SEQUENCES:

```
DLD x
JSB .DMP
DEF y
→ (result in A & B)
```

METHOD: X*Y

ATTRIBUTES:

	ENTRY POINTS:
	.DMP
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If overflow occurs, the largest positive integer is returned with overflow set. Overflow is cleared otherwise.

"E" is preserved.

.DNG

PURPOSE: Negate Double Integer *x*. Z = -*x*

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.DNG
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	DLD *x* JSB .DNG → (result in A & B)

METHOD: -X

ATTRIBUTES:

	ENTRY POINTS:
	.DNG
Parameters:	Double integer
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If overflow occurs the arguement is returned unchanged and overflow is set. Overflow is cleared otherwise.

"E" is preserved unless X=0, in which case E=1.

.FIXD

PURPOSE: Convert real to double integer

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.FIXD
EXTERNAL REFERENCES:	.FLUN
CALLING SEQUENCES:	DLD *x* JSB .FIXD → (*y* in A and B)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.FIXD
Parameters:	Real
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1) If the argument is outside the range $[-2^{31}, 2^{31})$ the result is 2^{31} -1 and overflow is set. Overflow is cleared otherwise.

2) .FXDE is not a usable entry point. It is referenced by .XFXD and .TFXD.

.FLTD

PURPOSE: Convert double integer to real

PROGRAM TYPE = 7 **ROUTINE IS: R**

ENTRY POINTS:	.FLTD
EXTERNAL REFERENCES:	.PACK
CALLING SEQUENCES:	DLD *x* JSB .FLTD → result in A & B

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.FLTD
Parameters:	Double integer
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: 1) If the argument is outside the range $[-2^{23}, 2^{23})$ the excess low-order bits are truncated. Positive numbers may become smaller, negative numbers may become smaller in value (larger in absolute value).

.TFTD

PURPOSE: Convert double integer to double real

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TFTD
EXTERNAL REFERENCES:	.XPAK
CALLING SEQUENCES:	DLD *x* JSB .TFTD DEF <result>

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.TFTD
Parameters:	Double integer
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

.TFXD

PURPOSE: Convert double real to double integer

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TFXD
EXTERNAL REFERENCES:	.FLUN, .CFER, .FIXD, .FXDE
CALLING SEQUENCES:	JSB .TFXD DEF *x* → (*y* in A and B)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.TFXD
Parameters:	Double real
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If the argument is outside the range $[-2^{31}, 2^{31})$ the result is $2^{31}-1$ and overflow is set. Overflow is cleared otherwise.

.XFTD

PURPOSE: Convert double integer to extended real

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	.XFTD
EXTERNAL REFERENCES:	.XPAK

CALLING SEQUENCES:

```
DLD x
JSB .XFTD
DEF <result>
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.XFTD
Parameters:	Double integer
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

.XFXD

PURPOSE: Convert extended real to double integer

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.XFXD
EXTERNAL REFERENCES:	.FLUN, .FIXD, .FXDE, .XFER
CALLING SEQUENCES:	JSB .XFXD DEF *x* → (*y* in A and B)

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	.XFXD
Parameters:	Extended real
Result:	Double integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: If the argument is outside the range $[-2^{31}, 2^{31})$ the result is 2^{31} -1 and overflow is set. Otherwise, overflow is cleared.

SECTION III

UTILITY SUBROUTINES

ABREG

PURPOSE: FORTRAN A and B register get routine.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	ABREG
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB ABREG DEF *+3 DEF IA DEF IB

METHOD: Contents of A-register before the call returned in IA; contents of B-register returned in IB.

IA, IB must not be array elements in FORTRAN or Algol because the registers will be modified in the array calculations.

ATTRIBUTES:

	ENTRY POINTS:
	ABREG
Parameters:	Integer
Result:	See method
FORTRAN:	Callable Call ABREG (IA, IB)
FORTRAN IV:	Callable Call ABREG (IA, IB)
ALGOL:	Callable Call ABREG (IA, IB)
Errors:	

BINRY

PURPOSE: Reads or writes data at a specified location (logical unit number, track, sector, and offset) of a disc.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: BREAD, BWRIT

EXTERNAL REFERENCES: EXEC, $OPSY

CALLING SEQUENCES:

```
JSB BREAD (of BWRIT) (Note 1)
DEF *+7
DEF buffer
DEF buffer length (words)
DEF logical unit
DEF track
DEF sector
DEF offset (Note 2)
→
```

ATTRIBUTES:

	ENTRY POINTS:	
	BREAD	BWRIT
Parameters:	Mixed	Mixed
Result:	Mixed	Mixed
FORTRAN:	Callable	Callable
FORTRAN IV:	Callable	Callable
ALGOL:	Callable	Callable
Errors:	None	None

NOTES:

1. BREAD is the read entry point and BWRIT is the write entry point.
2. Offset: If the offset equals 0, the transfer begins on the sector boundary; if the offset equals n, the transfer skips n words into the sector before starting.

CLRIO

PURPOSE: CLRIO is a dummy compatibility routine for use by the FORTRAN compilers, (was used by BCS system).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	CLRIO
EXTERNAL REFERENCES:	None

CALLING SEQUENCES:

```
JSB CLRIO
DEF *+1
→ All registers remain intact.
```

ATTRIBUTES:	**ENTRY POINTS:**
	CLRIO
Parameters:	None
Result:	None
FORTRAN:	Call CLRIO
FORTRAN IV:	Call CLRIO
ALGOL:	Callable as CODE procedure
Errors:	None

DBGLU

PURPOSE: Establishes the console lu through which DEBUG interacts with the user. Not used in DOS or RTE-IV.

PROGRAM TYPE = 7 **ROUTINE IS: U**

Entry points:	DBGLU, $DBP3
External references:	None
Calling sequence:	JSB DBGLU → (Only called by DEBUG module. Not called upon entry to a segment.)

METHOD:

Stores first RMPAR parameter in $DBP3.

COMMENTS:

Some main programs require the first RMPAR parameter to be something other than the console lu. In these cases, the user should assemble one of the following routines to replace the library version of DBGLU:

```
       RTE-II, RTE-III
       NAM DBGLU,7
       ENT DBGLU,$DBP3
DBGLU  NOP
       JMP DBGLU,I
$DBP3  DEC lu
       END

       RTE-M
       NAM DBGLU,7
       ENT DBGLU,$DBP3
       EXT $CON
DBGLU  NOP
       LDA $CON,I
       AND =B77
       STA $DBP3
       JMP DBGLU,I
$DBP3  NOP
       END
```

DBKPT

PURPOSE: Utility routine used by DEBUG. Never called by user programs. See DEBUG. Not used in DOS or RTE-IV.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	$DBP2, $MEMR
EXTERNAL REFERENCES:	None

DEBUG

PURPOSE: Aids in debugging user relocatable programs. Not used in DOS or RTE-IV.

PROGRAM TYPE = 7 **ROUTINE IS: U**

Entry points:	$DBP1, DEBUG
External references:	REIO, EXEC, $LIBR, $LIBX, $DBP3, DBGLU, IFBRK

METHOD:

The operator links DEBUG to a program at load-time with the Relocating Loader.

COMMENTS:

DEBUG places jump subroutine instructions in each breakpoint location and allows the program to execute normally until it reaches a breakpoint. The operator can set a relocation base, set instruction breakpoints, dump memory, and set values in memory or registers.

For more information on DEBUG, refer to Appendix B.

ER0.E

PURPOSE: To specify the LU for printing out library error messages. ER0.E is defaulted to 6.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
EXT ER0.E
    .
    .
    .
LDA LU
STA ER0.E
```

METHOD: Note that a zero value for ER0.E will inhibit error messages.

ATTRIBUTES:

	ENTRY POINTS:
	ER0.E
Parameters:	Logical Unit Number
Result:	None
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	

ERR0

PURPOSE: Prints a 4-character error code and a memory address on the logical unit ER0.E.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	ERR0
EXTERNAL REFERENCES:	REIO, ER0.E, ,PNAME
CALLING SEQUENCES:	see below

```
LDA NN }
LDB XX } see below
JSB ERR0
→
```

METHOD: *NN* is the routine identifier }
XX is the error type } pairs of ASCII characters.

Prints this on the logical unit ER0.E: *name NN XX* @ Address B

where *name* is the name of the user program.

where ADDRESS is P-1 or the call to ERR0

See Appendix A for a list of error messages which may be produced by the relocatable library subroutines.

ATTRIBUTES:

	ENTRY POINTS:
	ERR0
Parameters:	ASCII Characters
Result:	Printed
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

GETAD

PURPOSE: Determines the true address of a parameter passed to a subroutine and places the address in ADRES.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	GETAD, ADRES
EXTERNAL REFERENCES:	NONE
CALLING SEQUENCES:	JSB GETAD DEF SUB,I LDA ADRES see below

METHOD:

```
    JSB SUB
    DEF X[,I]
     .
     .
     .
SUB NOP
    JSB GETAD
    DEF SUB,I
    LDA ADRES
    →
```

ATTRIBUTES:

	ENTRY POINTS:	
	GETAD	ADRES
Parameters:	Integer Address	NA
Result:	Address	Integer
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	None	None

NOTE: May not be called by privileged or re-entrant routines; refer to .PCAD.

IGET

PURPOSE: Provides FORTRAN and ALGOL programs with the ability to read the contents of a memory address.

PROGRAM TYPE = 7

ROUTINE IS: U

ENTRY POINTS:	IGET
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB IGET DEF *+2 DEF IADRS → results in A

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	IGET
Parameters:	Address
Result:	Contents of memory address
FORTRAN:	Callable as a function
FORTRAN IV:	Callable as a function
ALGOL:	Callable as a function
Errors:	None

NOTES: This routine is for FORTRAN and ALGOL users only.

IND.E

PURPOSE: Used by .INDR and .INDA routines to select output LU for error messages, Default is 6; a 0 inhibits messages

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	IND.E
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	EXT IND.E LDA LU STA IND.E

METHOD:

ATTRIBUTES:

ENTRY POINTS:

	IND.E
Parameters:	Logical Unit Number
Result:	None
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	

INDEX

PURPOSE: Returns the address (.INDA) or value (.INDR) of an ALGOL array element.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:

.INDA
.INDR

EXTERNAL REFERENCES:

REIO, IND.E

CALLING SEQUENCES:

```
 JSB .INDA (or .INDR)
 DEF array table (see below)
 DEF - number of indices
 DEF subscript number  1
  .
  .
  .
 DEF subscript N
→result in A or A & B
```

METHOD: Array Table:

```
TABLE DEC number of indices (+ = real, - = integer)
      DEC size of 1st dimension
      DEC -lower bound of 1st dimension
       .
       .
       .
      DEC size of last dimension
      DEC -lower bound of last dimension
      DEF array address
```

ATTRIBUTES:

ENTRY POINTS:

	.INDA	.INDR
Parameters:	Integer	Integer
Result:	Address: A	Value: A or A & B
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	See Note 1	See Note 1

NOTES:

1. Prints INDEX? *address*

   where *address* is the address of call. Routine returns with result = 0.

ISSR

PURPOSE: Sets the S-register to the value *N*.

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	ISSR
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB ISSR DEF *+2 DEF *N*

ATTRIBUTES:

	ENTRY POINTS:
	ISSR
Parameters:	Integer
Result:	None
FORTRAN:	Callable: CALL ISSR(*N*)
FORTRAN IV:	Callable: CALL ISSR(*N*)
ALGOL:	Callable as CODE Procedure
Errors:	None

ISSW

PURPOSE: Sets the sign bit (15) of A-Register equal to bit *N* of the switch register.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	ISSW
EXTERNAL REFERENCES:	NONE
CALLING SEQUENCES:	LDA *N* JSB ISSW → result in A

ATTRIBUTES:	**ENTRY POINTS:**
	ISSW
Parameters:	Integer
Result:	Integer
FORTRAN:	Function: ISSW (*N*)
FORTRAN IV:	Function: ISSW (*N*)
ALGOL:	Not callable directly
Errors:	None

MAGTP

PURPOSE: Performs utility functions on magnetic tape and other devices: checks status, performs rewind/standby, writes a gap, and issues a clear request.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	IEOF, IERR, IEOT, ISOT, LOCAL, IWRDS(N/A in RTE), RWSTB
EXTERNAL REFERENCES:	.ENTR, EXEC

ATTRIBUTES:

	ENTRY POINTS:
	IEOF, IERR, IEOT, ISOT, LOCAL, IWRDS, RWSTB
Parameters:	Integer
Result:	N/A
FORTRAN:	Callable as subroutine
FORTRAN IV:	Callable as subroutine
ALGOL:	Callable as CODE procedure
Errors:	Returns on illegal call

CALLING SEQUENCES:

The calling sequence and purpose of each entry point is:

```
JSB IEOF
DEF *+2
DEF unit
→
```

Returns a negative value in A if an end-of-file was encountered during last tape operation on the logical unit specified.

```
JSB IERR
DEF*+2
DEF unit
→
```

Returns a negative value in A if a parity or timing error was not cleared after three read attempts during the last operation on the specified unit (cannot occur if EOF occurs).

```
JSB IEOT
DEF *+2
DEF unit
→
```

Returns a negative value in A if an end-of-tape was encountered during the last forward movement of the specified unit.

```
JSB ISOT
DEF *+2
DEF unit
→
```

Returns a negative value in A if the start-of-tape marker is under the tape head of the specified unit.

```
JSB LOCAL
DEF *+2
DEF unit
→
```

Returns a negative value in A if the specified unit is in local mode.

```
JSB IWRDS
DEF *+2
DEF unit
→
```

(Not available in RTE.) Returns the value of the transmission log of the last read/write operation on the specified unit. (In the formatter environment, this value is always a positive number of characters.)

```
JSB RWSTB
DEF *+2
DEF unit
→
```

Rewinds the specified logical unit and sets it to LOCAL.

NAMR

PURPOSE: FORTRAN routine to read an input buffer of any length and produces a parameter buffer of 10 words.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: NAMR

EXTERNAL REFERENCES: .ENTR

CALLING SEQUENCES:

```
JSB NAMR
DEF *+5
DEF IPBUF
DEF INBUF
DEF LENGTH
DEF ISTRC
```

NAMR equals -1 if no characters are in INBUF.
NAMR equals 0 if the character string has been parsed.

```
WHERE:  IPBUF = 10 word destination parameter buffer.
        The ten words are described as follows:

        Word 1 = 0 if type = 0 (See below)
        Word 1 = 16 bit number if type = 1. If number is negative,
                 number is in two's complement.
        Word 1 = Chars 1 & 2 if type = 3
        Word 2 = 0 if type = 0 or 1, chars 2 & 3 or trailing space(s)
                 if 3.
        Word 3 = Same as word 2. (Type 3 param. is left justified)
        Word 4 = Parameter type of all 7 parameters in 2 bit pairs.
                 Note the difference between NAMR parameter types,
                 and those for the system library routine PARSE.
                 0 = Null parameter
                 1 = Integer numberic parameter
                 2 = Not implemented yet. (FMGR?)
                 3 = Left justified 6 ASCII character parameter.
                 Bits for ,FNAME : P1  : P2  : P3  : P4  : P5  : P6,
                             0,1   2,3   4,5   6,7   8,9  10,11 12,13
        Word 5 = 1st sub-parameter and has characteristics of word 1.
        Word 6 = 2nd sub-parameter delimeted by colons as in word 5.
        Word 7 = 3rd sub-param. as 5 & 6. (May be 0, number or 2 chars)
        Word 8 = 4th    "
        Word 9 = 5th    "
        Word 10 = 6th sub-param.  (For possible futures I.E. system #)
        INBUF = Starting addr of input buffer containing "NAMR".
        LENGTH = Character length of INBUF (must be positive value).
        ISTRC = Starting character number in INBUF. This parameter will
                be updated for possible next call to NAMR and the start
                character in INBUF. Caution: ISTRC is modified by this
                routine, therefore, it must be passed as a variable
                (not a constant) from caller (FTN).
```

ATTRIBUTES: **ENTRY POINTS:**

	NAMR
Parameters:	
Result:	
FORTRAN:	Callable: IF (NAMR (IPBUF,INBUF,LENTH,ISTRC)) 10,20
FORTRAN IV:	Callable: IF (NAMR (IPBUF,INBUF,LENTH,ISTRC)) 10,20
ALGOL:	Callable as integer function
Errors:	

EXAMPLES THAT CAN BE PARSED:

+12345, DOUG:DB:-12B:,,GEORGE: A, &PARSE:JB::4:-1:1775:123456B

WHERE:

NAMR #	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
1	12345	0	0	00001B	0	0	0	0	0	
2	DO	UG		00037B	DB	-10	0	0	0	0
3	0	0	0	00000B	0	0	0	0	0	0
4	GE	OR	GE	00017B	A	0	0	0	0	0
5	&P	AR	SE	12517B	JB	0	4	-1	1775	-22738

TEST PROGRAM

```
FTN,L
      PROGRAM TESTN
      DIMENSION IB(36),IDMY(2),IPBUF(10)
      EQUIVALENCE (IDMY,DMY),(LEN,IDMY(2))
   1  WRITE (1,100)
 100  FORMAT ("INPUT ASCII NAMR'S TO PARSE ?")
      DMY = EXEC (1,401B,IB,-72)
      ISCR = 1
      DO 200 I=1,10
      IF ( NAMR(IPBUF,IB,LEN,ISCR)) 1,210
 210  WRITE (1,220) ISCR,IPBUF,IPBUF
 220  FORMAT (" "/,I3,10(X,I6)/" "3A2,7(X,06))
 200  CONTINUE
      STOP
      END
      END$
```

OVF

PURPOSE: Returns value of overflow bit in bit 15 of the A-Register and clears the overflow bit.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	OVF
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB OVF DEF RTN → result in A

METHOD: If overflow bit is set (on), the A-Register is set negative; if the overflow bit is off, the A-Register is set to zero.

ATTRIBUTES:

	ENTRY POINTS:
	OVF
Parameters:	None
Result:	Integer: A
FORTRAN:	Callable: See notes
FORTRAN IV:	Callable: See notes
ALGOL:	Not callable
Errors:	None

```
NOTES:  IF (OVF(IDMY)) 10,20
     10 start of user's overflow set routine
     20 start of user's overflow clear routine
```

PAU.E

PURPOSE: Used by .PAUS and .STOP routines to select LU on which to output Pause message. Default is 1; a 0 inhibits messages.

PROGRAM TYPE = 7

ROUTINE IS: U

ENTRY POINTS:	PAU.E
EXTERNAL REFERENCES:	None

CALLING SEQUENCES:

```
EXT PAU.E
LDA LU
STA PAU.E
```

ATTRIBUTES:

ENTRY POINTS:

	PAU.E
Parameters:	Logical Unit Number
Result:	None
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	

PAUSE

PURPOSE: Prints the following message on the console device: *name:* PAUSE *xxxx* where *name* is the calling program name and *xxxx* is the specified integer *I*. Halts program execution and returns to operating system.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .PAUS, .STOP

EXTERNAL REFERENCES: EXEC, PAU.E, REIO, PNAME

CALLING SEQUENCES:

```
LDA I
JSB .PAUS (or .STOP)
→
See Note
```

ATTRIBUTES:

ENTRY POINTS:

	.PAUS	.STOP
Parameters:	Integer	Integer
Result:	None	None
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	None	None

NOTE: When .PAUS is used, the program may be continued using GO (RTE) or :GO (DOS).

PNAME

PURPOSE: Moves the name of the currently executing program from the program's ID segment to a three word array.

ENTRY POINTS: PNAME

EXTERNAL REFERENCES: .ENTR, $OPSY

CALLING SEQUENCES:

```
      JSB PNAME
      DEF *+2
      DEF IARAY
    →
  IARAY BSS 3
```

ATTRIBUTES:

ENTRY POINTS:

	PNAME
Parameters:	Integer
Result:	ASCII characters
FORTRAN:	Callable (CALL PNAME (IARAY))
FORTRAN IV:	Callable (CALL PNAME (IARAY))
Algol:	Callable as CODE procedure
Errors:	None

Note: The sixth character is returned as an ASCII space.

Sample Program:

```
    PROGRAM PRNAM
    DIMENSION IARAY(3)
    CALL PNAME (IARAY)
    WRITE (1,100) IARAY
100 FORMAT ("∧∧PROGRAM∧",3A2,"EXECUTING:/)
    STOP
```

PTAPE

PURPOSE: Positions a magnetic tape unit by spacing forward or backward a number of files and/or records.

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS: PTAPE

EXTERNAL REFERENCES: EXEC, .ENTR

CALLING SEQUENCES:

```
JSB PTAPE
DEF *+4
DEF logical unit
DEF file count       {  see below
DEF record count     {
→
```

For example:

0 means make no file movements.
-1 means backspace to the beginning of the current file.
1 means forward space to beginning of the next file.
-2 means backspace to the beginning of the previous file.

Record count: positive for forward, negative for backward.

The file count is executed first, then the record count.

EOF marks count as a record.

For example:

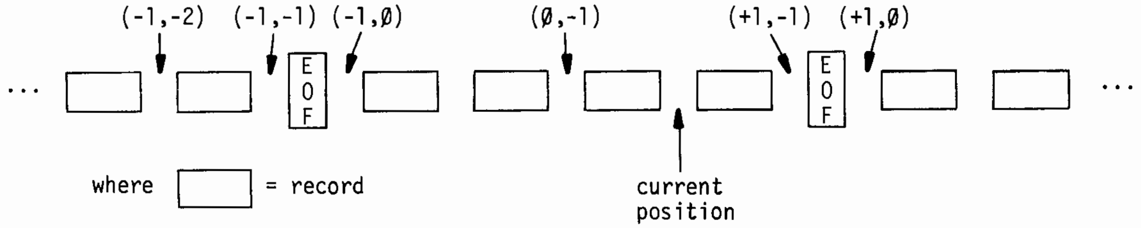
0,-1 means move back one record.
-1,0 means backspace to the first record of the current file.

See Note 1.

ATTRIBUTES:

	ENTRY POINTS:
	PTAPE
Parameters:	Integers
Result:	None
FORTRAN:	Callable: CALL PTAPE(logical unit,file cnt,record cnt)
FORTRAN IV:	Callable: CALL PTAPE(logical unit,file cnt,record cnt)
ALGOL:	Callable as CODE procedure
Errors:	None

NOTES: 1. The diagram below shows how the position of the magnetic tape would change with several file/record counts.



2. After using PTAPE, always check status with MAGTP.

RMPAR

PURPOSE: Move five parameters from the programs ID segment into a buffer within the programs memory space. If the program resides in a partition, the parameters are cross loaded from the system maps. Used to retrieve up to five parameters passed to a program by the operating system (See note 1).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: RMPAR

EXTERNAL REFERENCES: $OPSY

CALLING SEQUENCES:

```
          Suspend call or program
             entry point
          JSB RMPAR
          DEF *+2
          DEF ARRAY
          →
    ARRAY BSS 5
```

ATTRIBUTES:

ENTRY POINTS:

	RMPAR
Parameters:	Integer
Result:	Integer
FORTRAN:	Callable
FORTRAN IV:	Callable
ALGOL:	Callable
Errors:	None

Notes:

1. The operating system will insert parameters into a program's ID segment as a result of:
   - a. ON, GO, and other functions in RTE (refer to RTE manual for other functions of this call).
   - b. :PR or :GO in DOS (refer to a disc operating system manual).
   - c. Program execution of an EXEC call.
2. The RMPAR call must occur as the first executable instruction in the program or as the first executable instruction following the program suspend call.

Examples:

```
FTN,L                        ALGOL
     PROGRAM TEST            INTEGER P1, P2, P3, P4, P5
     DIMENSION IBUF (5)      .
     CALL RMPAR (IBUF)       .
           or                .
     PAUSE                   CALL RMPAR(P1)
     CALL RMPAR (IBUF)       Parameter cannot be an
                             array in ALGOL program.
```

RSFLG

PURPOSE: To set the save resource flag to RTE-BASIC. Certain subroutines used by RTE Real-Time Multi-User BASIC modify or store intermediate results within the device subroutine and expect those results to be intact for subsequent calls to those routines. The subroutine 'RSFLG' sets a flag which BASIC interogates to determine whether to save a copy of the device subroutine on the disc or allow the device subroutine to be overlayed.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:

```
RSFLG,#RSFG
```

EXTERNAL REFERENCES:

```
.ENTR
```

CALLING SEQUENCES:

```
JSB RSFLG
DEF *+1
→
```

ATTRIBUTES:

	ENTRY POINTS:
	RSFLG
Parameters:	None
Result:	A and B unchanged. #RSFG set to 1.
FORTRAN:	Callable
FORTRAN IV:	Callable
ALGOL:	Callable
Errors:	None

SREAD

PURPOSE: Reads a source record or sector from a device specified by a logical unit number. (Used only by system programs).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: %READ, %JFIL, %RDSC,

EXTERNAL REFERENCES: $OPSY, EXEC

CALLING SEQUENCES:

```
JSB %READ
DEF *+5
DEF input logical unit
DEF input buffer
DEF negative number of characters
EOP return
→ B = number of characters
LDA Code
LDB sector #
JSB %RDSC
→ A = last word in sector
JSB %JFIL
→ A = last word in sector
```

ENTRY POINTS: %READ reads a source record from disc or other device specified by logical unit number.
%RDSC reads a specified sector, returning the (RTE) code word.
%JFIL rewinds source; reads sector pointed to by the base page source-file code word.

#COS

PURPOSE: Entry to CCOS with no error return.

PROGRAM TYPE = 7

ROUTINE IS: U

ENTRY POINTS: #COS

EXTERNAL REFERENCES: ERR0, .ENTR, CCOS

CALLING SEQUENCES:

```
JSB #COS
DEF *+3
DEF y
DEF x
→
```

ATTRIBUTES:

	ENTRY POINTS:
	#COS
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

#EXP

PURPOSE: Entry to CEXP with no error return.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: #EXP

EXTERNAL REFERENCES: ERR0,.ENTR, CEXP

CALLING SEQUENCES:

```
JSB #EXP
DEF *+3
DEF y
DEF x
→
```

ATTRIBUTES:

ENTRY POINTS:

	#EXP
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

#LOG

PURPOSE: Entry to CLOG with no error return.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	#LOG
EXTERNAL REFERENCES:	ERR0, .ENTR, CLOG
CALLING SEQUENCES:	JSB #LOG DEF *+3 DEF *y* DEF *x* →

ATTRIBUTES:

	ENTRY POINTS:
	#LOG
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

#SIN

PURPOSE: Entry to CSIN with no error routine.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: #SIN

EXTERNAL REFERENCES: ERR0, .ENTR, CSIN

CALLING SEQUENCES:

```
JSB #SIN
DEF *+3
DEF y
DEF x
→
```

ATTRIBUTES:

	ENTRY POINTS:
	#SIN
Parameters:	Complex
Result:	Complex
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

$EXP

PURPOSE: Entry to DEXP with no alternate error routine.

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	$EXP
EXTERNAL REFERENCES:	ERR0, .ENTR, DEXP
CALLING SEQUENCES:	JEB $EXP DEF *+3 DEF *y* DEF *x* →

ATTRIBUTES:

	ENTRY POINTS:
	$EXP
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

$LOG

PURPOSE: Entry to DLOG with no error return.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	$LOG
EXTERNAL REFERENCES:	ERR0, .ENTR, DLOG

CALLING SEQUENCES:

```
JSB %EXP
DEF *+3
DEF Y
DEF X
→
```

ATTRIBUTES:

ENTRY POINTS:

	$LOG
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

$LOGT

PURPOSE: Entry to DLOGT with no error return.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: $LOGT $LOG0

EXTERNAL REFERENCES: DLOGT, .ENTR, ERR0

CALLING SEQUENCES:

```
JSB $LOGT (or $LOG0)
DEF *+3
DEF Y
DEF X
→
```

ATTRIBUTES:

	ENTRY POINTS:
	$LOGT ($LOG0)
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

$SETP

PURPOSE: Set up a list of pointers.

PROGRAM TYPE = 6	**ROUTINE IS: R**
ENTRY POINTS:	$SETP
EXTERNAL REFERENCES:	.ZPRV

CALLING SEQUENCES:

```
LDA <starting pointer>
LDB <starting address to be set>
JSB $SETP
DEF <count>
→
```

METHOD: The contents of A are stored in the address in B. A and B are then incremented. The process is performed "count" times, affecting "count" memory locations. Upon return:

A = 0
B = B + count

ATTRIBUTES:

	ENTRY POINTS:
	$SETP
Parameters:	Integer
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1) This routine is available in microcode.

2) The sign bit of B is ignored.

$SQRT

PURPOSE: Entry to DSQRT with no error return.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: $SQRT

EXTERNAL REFERENCES: DSQRT, ERR0, .ENTR

CALLING SEQUENCES:

```
JSB $SQRT
DEF *+3
DEF y
DEF x
→
```

ATTRIBUTES:

	ENTRY POINTS:
	$SQRT
Parameters:	Extended real
Result:	Extended real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%ABS

PURPOSE: Call-by-name entry to IABS(*I*)

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: %ABS

EXTERNAL REFERENCES: IABS

CALLING SEQUENCES:

```
JSB %ABS
DEF *+2
DEF I
→ result in A
```

ENTRY POINTS:

ATTRIBUTES:	%ABS
Parameters:	Integer: A
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

$TAN

PURPOSE: DTAN with no error return

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	$TAN
EXTERNAL REFERENCES:	DTAN , .ENTR
CALLING SEQUENCES:	JSB DTAN DEF * +3 DEF <result> DEF x →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	$TAN
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See DTAN

NOTES:

%AN

PURPOSE: Call-by-name entry to TAN(x).

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	%AN
EXTERNAL REFERENCES:	TAN, ERR0
CALLING SEQUENCES:	JSB %AN DEF *+2 DEF x → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%AN
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%AND

PURPOSE: Call-by-name entry to calculate the logical "and" (product) of two integers *I* and *J*.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%AND
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB %AND DEF *+3 DEF *I* DEF *J* → result in A

ATTRIBUTES:

	ENTRY POINTS:
	%AND
Parameters:	Integer
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%ANH

PURPOSE: Call-by-name entry to TANH(*x*).

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	%ANH
EXTERNAL REFERENCES:	TANH
CALLING SEQUENCES:	JSB %ANH DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%ANH
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%BS

PURPOSE: Call-by-name entry to ABS(*x*).

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	%BS
EXTERNAL REFERENCES:	ABS
CALLING SEQUENCES:	JSB %BS DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%BS
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%FIX

PURPOSE: Call-by-name entry to IFIX(*x*).

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	%FIX
EXTERNAL REFERENCES:	IFIX
CALLING SEQUENCES:	JSB %FIX DEF *+2 DEF *x* → result in A

ATTRIBUTES:

	ENTRY POINTS:
	%FIX
Parameters:	Real
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%IGN

PURPOSE: Call-by-name entry to SIGN (x, z)

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	%IGN
EXTERNAL REFERENCES:	SIGN
CALLING SEQUENCES:	JSB %IGN DEF *+3 DEF x DEF z →result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	%IGN
Parameters:	Real or integer and real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%IN

PURPOSE: Call-by-name entry to SIN (*x*).

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	%IN
EXTERNAL REFERENCES:	SIN, ERR0
CALLING SEQUENCES:	JSB %IN DEF *+2 DEF *x* →result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	%IN
Parameters:	Real
Result:	Real: A & B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See SIN

%INT

PURPOSE: Call-by-name entry to AINT (*x*).

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	%INT
EXTERNAL REFERENCES:	AINT
CALLING SEQUENCES:	JSB %INT DEF *+2 DEF *x* →result in A & B

ATTRIBUTES:

	ENTRY POINTS:
	%INT
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%LOAT

PURPOSE: Call-by-name entry to FLOAT (*I*)

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%LOAT
EXTERNAL REFERENCES:	FLOAT
CALLING SEQUENCES:	JSB %LOAT DEF *+2 DEF *I* → result in A&B

ATTRIBUTES:

ENTRY POINTS:

	%LOAT
Parameters:	Integer
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%LOG

PURPOSE: Call-by-name entry to ALOG (*x*).

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	%LOG
EXTERNAL REFERENCES:	ALOG, ERR0
CALLING SEQUENCES:	JSB %LOG DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%LOG
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See ALOG

% LOGT

PURPOSE: Call-by-name entry to ALOGT (*x*).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%LOGT %LOG0
EXTERNAL REFERENCES:	ALOGT, ERR0
CALLING SEQUENCES:	JSB %LOGT (%LOG0) DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%LOGT (%LOG0)
Parameters:	Real
Result:	Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%NT

PURPOSE: Call-by-name entry to INT (x).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%NT
EXTERNAL REFERENCES:	INT
CALLING SEQUENCES:	JSB %NT DEF *+2 DEF x (real) → result in A

ATTRIBUTES:

	ENTRY POINTS:
	%NT
Parameters:	Real
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%OR

PURPOSE: Call-by-name entry to calculate the inclusive "or" of two integers *I* and *J*.

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	%OR
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB %OR DEF *+3 DEF *I* DEF *J* → result in A

ATTRIBUTES:

	ENTRY POINTS:
	%OR
Parameters:	Integer
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%OS

PURPOSE: Call-by-name entry to COS (x).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%OS
EXTERNAL REFERENCES:	COS, ERR0
CALLING SEQUENCES:	JSB %OS DEF *+2 DEF x → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%OS
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See COS

%OT

PURPOSE: Standard call-by-name subroutine for NOT function.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%0T
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB %0T DEF *+2 DEF *I* → result in A

METHOD:

Executes ones complement of *I*.

ATTRIBUTES:

	ENTRY POINTS:
	%0T
Parameters:	Integer
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%QRT

PURPOSE: Call-by-name entry to SQRT (x).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%QRT
EXTERNAL REFERENCES:	SQRT, ERR0
CALLING SEQUENCES:	JSB %QRT DEF *+2 DEF x → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%QRT
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See SQRT

% SIGN

PURPOSE: Call-by-name entry to ISIGN (*I*, *Z*).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: %SIGN

EXTERNAL REFERENCES: ISIGN

CALLING SEQUENCES:

```
JSB %SIGN
DEF *+3
DEF I
DEF Z
→ result in A
```

ATTRIBUTES:

	ENTRY POINTS:
	%SIGN
Parameters:	Real (or integer) & integer
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%SSW

PURPOSE: Call-by-name entry to ISSW (*N*).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%SSW
EXTERNAL REFERENCES:	ISSW
CALLING SEQUENCES:	JSB %SSW DEF *+2 DEF *N* (integer) → result in A

ATTRIBUTES:

	ENTRY POINTS:
	%SSW
Parameters:	Integer
Result:	Integer: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

%TAN

PURPOSE: Call-by-name entry to ATAN (*x*).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%TAN
EXTERNAL REFERENCES:	ATAN, ERR0
CALLING SEQUENCES:	JSB %TAN DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%TAN
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See ATAN

%WRIS

PURPOSE: Writes a disc source file (used only by system programs).

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	%WRIS, %WRIN, %WEOF,
EXTERNAL REFERENCES:	EXEC

Note: This routine can only be called in the RTE System.

%WRIT

PURPOSE: Writes a load-and-go file on disc (used only by system programs).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	%WRIT, %WRIF, %WBUF,
EXTERNAL REFERENCES:	$OPSY, EXEC

%XP

PURPOSE: Call-by-name entry to EXP (*x*).

PROGRAM TYPE = 7 ROUTINE IS: U

ENTRY POINTS:	%XP
EXTERNAL REFERENCES:	EXP, ERR0
CALLING SEQUENCES:	JSB %XP DEF *+2 DEF *x* → result in A&B

ATTRIBUTES:

	ENTRY POINTS:
	%XP
Parameters:	Real
Result:	Real: A&B
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See EXP

.ENTC

PURPOSE: Transfers the true addresses of parameters from a calling sequence into a subroutine and adjusts return addresses to the true return point.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.ENTC
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	Same as .ENTP .ENTR

ATTRIBUTES:

	ENTRY POINTS:
	.ENTC
Parameters:	Address
Result:	Address
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES: This routine assumes the subroutine call is of the form:

```
JSB  SUB
DEF  P1       (first parameter)
.
.
.
DEF  Pm
```

COMMENTS: The number of parameter addresses actually passed by the calling routine must agree with the number requested by the receiving routine.

.ENTR

PURPOSE: Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.ENTR, .ENTP
EXTERNAL REFERENCES:	.ZPRV

CALLING SEQUENCES:

For all Utility routines:

```
PARAM BSS N      (N = maximum number of parameters)  see note 3.
  SUB NOP        (entry point to subroutine)
      JSB .ENTR
      DEF PARAM
```

For all privileged routines:

```
PARAM BSS N      (N = maximum number of parameters)
  SUB NOP        Subroutine entry point
      JSB .ZPRV
      DEF LIBX
      JSB .ENTP
      DEF PARAM
      .
      .
      .
 LIBX JMP SUB,I
      DEF LIBX
```

For all re-entrant routines:

```
  TDB NOP        (re-entrant processing table)
      DEC Q+N+3  (size of table)
      NOP
      BSS Q      (subroutine variables)
PARAM BSS N      (N = maximum number of parameters)
      SUB NOP    (Subroutine entry point)
      JSB .ZRNT
      DEF LIBX
      JSB .ENTP
      DEF PARAM
      STA TBD+2  (return address)
      .
      .
      .
 LIBX JMP TDB+2,I
      DEF TDB
      DEC 0
```

.ENTR

ATTRIBUTES:

ENTRY POINTS:

	.ENTR	.ENTP
Parameters:	Address	Address
Result:	Address	Address
FORTRAN:	Not callable	Not callable
FORTRAN IV:	Not callable	Not callable
ALGOL:	Not callable	Not callable
Errors:	None	None

NOTES:

1. The true parameter address is determined by eliminating all indirect references.

2. .ENTR and .ENTP assume the subroutine call is of the form:

```
JSB SUB
DEF *+M+1            (M = number of parameters)
DEF P1
.
.
.
DEF PM
```

   If M > N, then N parameters will be passed. If N > M, then M parameters will be passed, and any parameter addresses not passed remain as they were from the previous call.

3. "PARAM BSS N" must appear immediately before the subroutine entry point "SUB NOP". The entry point is set to the return address (DEF *+M+1). "JSB .ENTR" must be the first instruction after the subroutine entry point. "JSB .ENTP" must be the third instruction after the subroutine entry point.

4. This routine is available in FFP firmware. See note on page 1-6.

.FMUI

PURPOSE: .FMUI contains three entry points corresponding to three conversion procedures:

.FMUI — Convert an ASCII digit string to internal numeric form.

.FMUO — Convert A numeric value to ASCII.

.FMUP — Convert an unpacked internal format number (from .FMUI) to a normal format.

PROGRAM TYPE = 7

ENTRY POINTS: .FMUI, .FMUO, .FMUP

EXTERNAL REFERENCES: .PACK, .ENTR, .MVW, IFIX

CALLING SEQUENCES:

```
JSB  .FMUI
DEF  *+8
DEF  <buffer>     ASCII, one digit/word, FORTRAN R1 format
DEF  <bufsiz>     # of digits in <buffer> between 0 and 20, inclusive
DEF  <sign>       0 = positive, 1 = negative
DEF  <exp>        scale factor; power of ten
DEF  <result>     returned value
DEF  <type>       type of <result> (see below)
DEF  <ovfl>       returned from .FMUI, 1 if overflow or underflow else 0.


JSB  .FMUO
DEF  *+7
DEF  <buffer>     returned from .FMUO
DEF  <bufsize>    returned from .FMUO
DEF  <sign>       returned from .FMUO
DEF  <exp>        returned from .FMUO
DEF  <value>      input value
DEF  <type>       type of value (see below)


JSB  .FMUP
DEF  *+5
DEF  <result>
DEF  <type>
DEF  <unpkd>      input, <result> from .FMUI
DEF  <ovfl>       returned from .FMUP, 1 if overflow or underflow else 0.
```

.FMUI

METHOD: .FMUI — The value in <buffer> is converted to binary with the digit in buffer (i) having weight 10** (<exp>-i). As if it were of the form 0. <buffer> E <exp>. The result is negated if <sign> = 1, and rounded to the specified type:

<TYPE> =	TYPE
0	16-bit Integer (1 word)
1	32-bit Integer (2 words)
2	32-bit Real (2 words)
3	48-bit Real (3 words)
4	64-bit Real (4 words)
5	unpacked internal format (5 words)

.FMUO — Reverse of .FMUI, i.e., generates <buffer>, <exp>, and <sign> from <value> as described in .FMUI. The result should be rounded by calling .FMUR since there may be some round-off error by .FMUO such as 2.0 may convert to 1.99999.

.FMUP — A type 5 buffer <unpkd> created by .FMUI is converted to a normal type buffer <result>. The type of <result> is specified by <type> and must be 0 to 4.

ATTRIBUTES:

ENTRY POINTS:

	.FMUI, .FMUO, .FMUP
FORTRAN:	Callable (FORTRAN 77 only)
Pascal:	Callable
Errors:	None

.FMUR

PURPOSE: Rounding of digit string produced by .FMUO.

ENTRY POINTS: .FMUR

EXTERNAL REFERENCES: .ENTR

CALLING SEQUENCES:

```
JSB  .FMUR
DEF  *+5
DEF  <buffer>    ASCII, one digit/word, FORTRAN R1 format (input and returned
                 value)
DEF  <bufsiz>    # of digits in <buffer> between 0 and 20, inclusive
DEF  <rndsiz>    # digits to round to
DEF  <ovfl>      returned from .FMUR, 1 if carry overflow occurs else 0.
```

METHOD: Add 5 to the (<rndsiz>+1)th digit of <buffer>. If the (<rndsiz>+1)th digit of <buffer> is 5 and all other significant digits are 9 then the first digit in <buffer> is set to 1, all other digits are set to 0, and ovfl is set to 1 (i.e., if carry overflow occurs, rightshift carry into <buffer> and set ovfl to 1). If .FMUO was used to create <buffer> then the new scale factor should be <exp>+<ovfl>.

ATTRIBUTES:

ENTRY POINTS:

	.FMUR
FORTRAN:	Callable (FORTRAN 77)
Pascal:	Callable
Errors:	None

EXAMPLE: A conversion to 10 digits would be as follows:

```
.FMUO (buffer,11,sign,exp,value,type)
.FMUR (buffer,11,10,ovfl)
exp=exp+ovfl
```

.GOTO

PURPOSE: Transfers control to the location indicated by a FORTRAN computed GO TO statement: GO TO (K_1, K_2, ... K_N) J

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .GOTO

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
JSB .GOTO
DEF *+n+2
DEF J
DEF K1
 :
 :
DEF KN
→
```

ATTRIBUTES:

	ENTRY POINTS:
	.GOTO
Parameters:	Addresses
Result:	Branch to address K_J
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	If J <1 then K_1; if $J>N$ then K_N

Note: This routine is available in FFP firmware. See note on page 1-6.

.MAP.

PURPOSE: Returns actual address of a particular element of a two-dimensional FORTRAN array.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.MAP.
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	JSB .MAP. DEF array DEF first subscript DEF second subscript OCT first dimension, as below → result in A

METHOD:

Length of first dimension is actual for a real array, two's complement for an integer array.

ATTRIBUTES:

	ENTRY POINTS:
	.MAP.
Parameters:	Integer
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.OPSY

PURPOSE: Determines which operating system is in control. Included for compatibility with previous libraries.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.OPSY
EXTERNAL REFERENCES:	$OPSY,
CALLING SEQUENCES:	JSB .OPSY → result in A A = -7 (RTE-MI) A = -15 (RTE-MII) A = -5 (RTE-MIII) A = -3 (RTE-II) A = -1 (RTE-III) A = -9 (RTE-IV) A = 1 (DOS)

NOTE: This routine is equivalent to:
```
EXT $OPSY
LDA $OPSY
```

ATTRIBUTES:

	ENTRY POINTS:
	.OPSY
Parameters:	None
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

.PCAD

PURPOSE: Return the true address of a parameter passed to a subroutine.

PROGRAM TYPE = 6 **ROUTINE IS: P**

ENTRY POINTS:	.PCAD
EXTERNAL REFERENCES:	.ZPRV
CALLING SEQUENCES:	JSB .PCAD DEF SUB, I → result in A (See below for context)

METHOD:

```
     JSB SUB         (call to subroutine; indirect bit is optional
     DEF X[,I]        on parameter)
      :
 SUB NOP             (entry point to subroutine)
      :
     JSB .PCAD
     DEF SUB, I
     → address of X in A
```

ATTRIBUTES:

ENTRY POINTS:

	.PCAD
Parameters:	Indirect Address
Result:	Direct Address: A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1. .PCAD has the same purpose as GETAD.
2. .PCAD is used by re-entrant or privileged subroutines because they cannot use GETAD.

.PRAM

Processes parameter values and/or addresses passed to Assembly language subroutines by ALGOL programs.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: .PRAM

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
JSB .PRAM
1st code word
2nd code word
:
Last code word
1st parameter address or value (2 words for real)
2nd parameter address or value (2 words for real)
:
Last parameter address or value
```

Format of 1st code word:

15 10 8 6 4 2 0

N	P_1	P_2	P_3	P_4	P_5

Where N is number of parameters (maximum of 52)

P_i is two bit code for ith parameter

Two bit code: upper bit = 1 means ith parameter is a value
upper bit = 0 means ith parameter is address
lower bit = 1 means parameter is real value (2 words)
lower bit = 0 means parameter is integer value

Format of other code words (maximum of 7):

14 12 10 8 6 4 2 0

P_K	P_{K+1}	P_{K+2}	P_{K+3}	P_{K+4}	P_{K+5}	P_{K+6}	P_{K+7}

ATTRIBUTES:

ENTRY POINTS:

	.PRAM
Parameters:	Integer
Result:	Integer & Real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTE: Used in Assembly language subroutines to retrieve parameters from calling sequence inside the ALGOL calling program.

.RCNG

PURPOSE: Convert calls using .ENTR to .ENTC convention.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.RCNG
EXTERNAL REFERENCES:	None
CALLING SEQUENCES:	See Method

METHOD:

```
BEFORE CALL:              AFTER CALL:              HOW THIS ROUTINE IS USED

JSB XADD                   NOP                          XADD   NOP
DEF *+4                    JSB .XADD                           JSB .RCNG
DEF Z                      DEF Z                               DEF @XADD+0
DEF X                      DEF X                               ORB
DEF Y                      DEF Y                       @XADD   DEF .XADD+0
<RETURN>                  <RETURN>                             ORR
```

ATTRIBUTES:

ENTRY POINTS:

	.RCNG
Parameters:	None
Result:	See method
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	None

NOTES: The subroutine *subr* is one of the eight non-intrinsic entry points: XADD, XSUB, XDIV, CADD, CSUB, CDIV, CMPY.

.SWCH

PURPOSE: Switches execution control to the *I*th entry of a sequence of *N* labels (implements ALGOL SWITCH statement).

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.SWCH
EXTERNAL REFERENCES:	None

CALLING SEQUENCES:

```
  LDA I
  JSB S
  → return if I is out of range
  :
S NOP
  JSB .SWCH
  ABS N (see below)
  DEF Label 1
  DEF Label 2
  :
  DEF Label N
```

N is the number of labels.
If *I* is out of range, .SWCH returns.

ATTRIBUTES:

	ENTRY POINTS:
	.SWCH
Parameters:	Addresses
Result:	N/A
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	If *I* is out of range, returns.

.TAPE

PURPOSE: Performs magnetic tape rewind, backspace or end-of-file operations on a specified logical unit.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	.TAPE
EXTERNAL REFERENCES:	EXEC
CALLING SEQUENCES:	LDA *.constant* JSB .TAPE →

METHOD:

```
    Constant = ZZXXYY
where:
    XX = 1   to write end of file
       = 2   to backspace one record
       = 3   to forward space one record
       = 4   to rewind magnetic tape
       = 5   to rewind/standby
       = 12  to write a gap
       = 13  to forward space one file
       = 14  to backspace one file
    YY = logical unit number of the magnetic tape
    ZZ = don't care
```

ATTRIBUTES:

	ENTRY POINTS:
	.TAPE
Parameters:	Integer
Result:	None
FORTRAN:	Not callable (Note 1)
FORTRAN IV:	Not callable (Note 1)
ALGOL:	Not callable
Errors:	None

NOTES: In FORTRAN use utility statements or PTAPE and MGTAP.

..MAP

PURPOSE: Computes the address of a specified element of a 1 or 2 or 3 dimension array; returns the address in the A-Register.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: ..MAP

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
For 1 dimension:   For 2 dimensions:              For 3 dimensions:
CCA, <CLE>         CLA, <CLE>                     CLA, INA, <CLE>
LDB N (see below)  LDB N (see below)              LDB N (see below)
JSB ..MAP          JSB ..MAP                      JSB ..MAP
DEF base address   DEF base address               DEF base address
DEF 1st subscript  DEF 1st subscript              DEF 1st subscript
→ address in A     DEF 2nd subscript              DEF 2nd subscript
                   DEF length of 1st dimension    DEF 3rd subscript
                   → address in A                 DEF length of 1st dimension
                                                  DEF length of 2nd dimension
                                                  → address in A
```

N = number of words per element in the array (1, 2, 3 or 4)

E reg = 1 if store to this element
0 if read from this element

ATTRIBUTES:

ENTRY POINTS:

	..MAP
Parameters:	Integer
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

Note: This routine is available in FFP firmware. See note on page 1-6.

/ATLG

PURPOSE: Compute (1-X)/(1+X) in double precision

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	/ATLG
EXTERNAL REFERENCES:	.TADD .TSUB .TDIV
CALLING SEQUENCES:	JSB /ATLG DEF *x*

METHOD:

X ← (1-X)/(1+X)

ATTRIBUTES:

	ENTRY POINTS:
	/ATLG
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1) No error checking is performed.
2) The X and Y registers may be changed.

/COS

PURPOSE: .COS with no error return

PROGRAM TYPE = 7	ROUTINE IS: U
ENTRY POINTS:	/COS
EXTERNAL REFERENCES:	.COS , .ENTR
CALLING SEQUENCES:	JSB /COS DEF * +3 DEF <result> DEF x →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/COS
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .TSCS

NOTES:

/CMRT

PURPOSE: Range reduction for .SIN, .COS, .TAN, .EXP and .TANH

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: /CMRT

EXTERNAL REFERENCES: .CFER, .TADD, .TSUB, .TMPY, .TFXD, .TFTD, .FLUN, IFIX, FLOAT

CALLING SEQUENCES:

```
LDA <flag>
JSB /CMRT
DEF <result>
DEF <constant>
DEF <argument>
→ error return
→ normal return  (B-register contains least significant bits of N)
```

METHOD: /CMRT multiplies the argument by the constant, then subtracts from this product the nearest even integer, N. If too much cancellation occurs in the above subtraction, or the argument is too large, the computation (depending on the flag) may be repeated in higher precision. If this can occur, a second constant must immediately follow the first. The second constant must have the value obtained by truncating the exact constant after 28 bits (including sign), and subtracting this value from the exact constant.

ATTRIBUTES:

ENTRY POINTS:

	/CMRT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See below for argument too large.

NOTES: 1) The accepted range of arguments depends on the setting of the flag. The table below shows the outcome of the different flag settings.

Flag	Example	Range	Criteria for using higher precision
-2	.EXP,c=2/ln(2)	[-128,128)	number outside [-8,+8)
-1	.TANH,c=4/ln(2)	[-8192*ln(2), 8191.75*ln(2))	
0	.TAN,c=4/pi	$[-2^{23},+2^{23})$	outside [-8,+8) or excessive cancellation
2,6	.COS,c=4/pi	$[-2^{23},+2^{23})$	outside [-8,+8) or N/2 is odd and excessive cancellation
4,8	.SIN,c=4/pi	$[-2^{23},+2^{23})$	outside [-8,+8) or N/2 is even and excessive cancellation

2) This routine may alter the X and Y registers.

3) This routine should be used by system programs only.

/EXP

PURPOSE: .EXP with no error return

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:	/EXP
EXTERNAL REFERENCES:	.EXP
CALLING SEQUENCES:	JSB /EXP DEF * +3 DEF <result> DEF x →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/EXP
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .EXP

NOTES:

/EXTH

PURPOSE: Compute 2^N X 2^Z or TANH(Z) for small double real Z

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: /EXTH

EXTERNAL REFERENCES: .PWR2, .TADD, TRNL

CALLING SEQUENCES:

```
LDA <N>
JSB  EXTH
DEF <result>
DEF <Y>
```

METHOD: If N equals -32768, TANH is computed, otherwise EXP is. The argument Y is the result of range reduction by /CMRT, so it has been scaled down by 2/1n(2) for EXP, and 4/1n(2) for TANH. The following approximations are used:

0.5*(EXP(Y)-1)=P(Z)/(Q(Z)-P(Z)) Z=Y*(2/1n(2))

TANH(Y)=P(W)/Q(W) W=Y*(4/1n(2))

$P(X) = X\cdot(P_0+X^2\cdot(P_1 + X^2\cdot P_2))$

$Q(X) = Q_0 + X^2\cdot(Q_1 + X^2))$

P0 = 1513.86417304653562
P1 = 20.2017000069531260
P2 = .023094321272953857
Q0 = 4368.08867006741699
Q1 = 233.178232051431036

ATTRIBUTES:

	ENTRY POINTS:
	/EXTH
Parameters:	Double real, Integer
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	None

NOTES:

1) No error checking is performed. The final exponent will be in error by a multiple of 128 if overflow or underflow occurs.

/LOG

PURPOSE: .LOG with no error return

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: /LOG

EXTERNAL REFERENCES: .LOG , .ENTR

CALLING SEQUENCES:

```
JSB /LOG
DEF * +3
DEF <result>
DEF x
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/LOG
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .LOG

NOTES:

/LOGO

PURPOSE: .LOGO with no error return.

PROGRAM TYPE = 7 **ROUTINE IS:**

ENTRY POINTS:	/LOGO or /LOGT
EXTERNAL REFERENCES:	.LOGO , .ENTR
CALLING SEQUENCES:	JSB /LOGO or /LOGT DEF * +3 DEF <result> DEF *x* →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/LOGO or /LOGT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .LOGO

NOTES:

/SIN

PURPOSE: .SIN with no error return

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: /SIN

EXTERNAL REFERENCES: .SIN , .ENTR

CALLING SEQUENCES:

```
JSB /SIN
DEF * +3
DEF <result>
DEF x
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/SIN
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .TSCS

NOTES:

/SQRT

PURPOSE: .SQRT with no error return

PROGRAM TYPE = 7	**ROUTINE IS: U**
ENTRY POINTS:	/SQRT
EXTERNAL REFERENCES:	.SQRT , .ENTR

CALLING SEQUENCES:

```
JSB /SQRT
DEF *+3
DEF < result >
DEF x
→
```

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/SQRT
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .SQRT

NOTES:

/TAN

PURPOSE: .TAN with no error return

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	/TAN
EXTERNAL REFERENCES:	.TAN , .ENTR
CALLING SEQUENCES:	JSB /TAN DEF * +3 DEF <result> DEF x →

METHOD:

ATTRIBUTES:

	ENTRY POINTS:
	/TAN
Parameters:	Double real
Result:	Double real
FORTRAN:	Not callable
FORTRAN IV:	Not callable
ALGOL:	Not callable
Errors:	See .TAN

NOTES:

/TINT

PURPOSE: Conversion of double precision to integer.

PROGRAM TYPE = 7 **ROUTINE IS:** U

ENTRY POINTS:	/TINT
EXTERNAL REFERENCES:	.TINT
CALLING SEQUENCES:	JSB /TINT DEF *+2 DEF <arguments> → (result in A)

METHOD: Calls .TINT

ATTRIBUTES:

	ENTRY POINTS:
	/TINT
Parameters:	Double precision
Result:	Integer
FORTRAN:	Not callable
FORTRAN IV:	Callable as IDINT with *y* option
ALGOL:	Not callable
Errors:	Overflow set if argument outside $[-2^{15},2^{15})$

NOTES:

SECTION IV

THE FORMATTER

THE FORMATTER

The Formatter is a subroutine that is called by relocatable programs to perform formatted data transfers, to interpret formats, to provide unformatted input and output of binary data, to provide free field input, and to provide buffer-to-buffer conversion. The Formatter is first given a string of ASCII characters that constitutes a format code. This "format" tells the Formatter the variables to transfer, the order, and the conversion (on input, ASCII characters are converted to binary values and on output, binary values are converted to ASCII). Then the calling program gives the Formatter a string of variables to be output or filled by input.

In FORTRAN and ALGOL programming, the programmer first defines a FORMAT string through FORMAT statements.

Example:

```
FORTRAN:  10 FORMAT (I5,A2,5F12.3)
          identifier    actual format

ALGOL:    FORMAT F23 (I5,A2,5F12.3);
          identifier    actual format
```

Then the programmer uses a READ or WRITE statement giving the logical unit number of the device to be used, the format identifier, and a list of variables.

Example:

```
FORTRAN:  20 WRITE (2,10) INT,LETR,ARRAY
            logical  format      variable
              unit   identifier    list

ALGOL:    WRITE (2,F23, INT, LETR, VARI);
            logical  format      variable
              unit   identifier    list
```

The FORTRAN and ALGOL Compilers automatically generate the correct calls to the Formatter. In Assembly Language, the programmer is responsible for all calls to the Formatter.

Two different formatters are available in DOS and RTE software systems:

1. FORTRAN Formatter (product no. 24153)
2. FORTRAN IV Formatter (part no. 24998-16002)

The FORTRAN Formatter requires less memory than the FORTRAN IV Formatter. The FORTRAN IV Formatter may be used with HP FORTRAN programs, but the FORTRAN Formatter may not by used with FORTRAN IV programs.

The FORTRAN IV Formatter includes all the features of the FORTRAN Formatter and double precision and complex number conversion.

INPUT AND OUTPUT

When the programmer uses a READ or WRITE statement in FORTRAN and ALGOL, the compiler generates all the necessary calls to the Formatter.

FORTRAN and ALGOL use of the formatter is documented in the following manuals:

HP FORTRAN (02116-9015)
RTE FORTRAN IV Reference Manual (92060-90023)
HP ALGOL (02116-9014)

The following description of the formatter is provided for the Assembly language programmer.

In Assembly Language the programmer is responsible for all calls to the Formatter. For each I/O operation, the program must first make an "Initialization" call (entry points .DIO and .BIO). This call establishes the format to be used (if any), and the logical unit and a way to say whether the operation is input or output. Then, for each data item, the program must make a separate call which depends on the type of data. Finally, for output only, the program must make a termination call that tells the Formatter to output the last record.

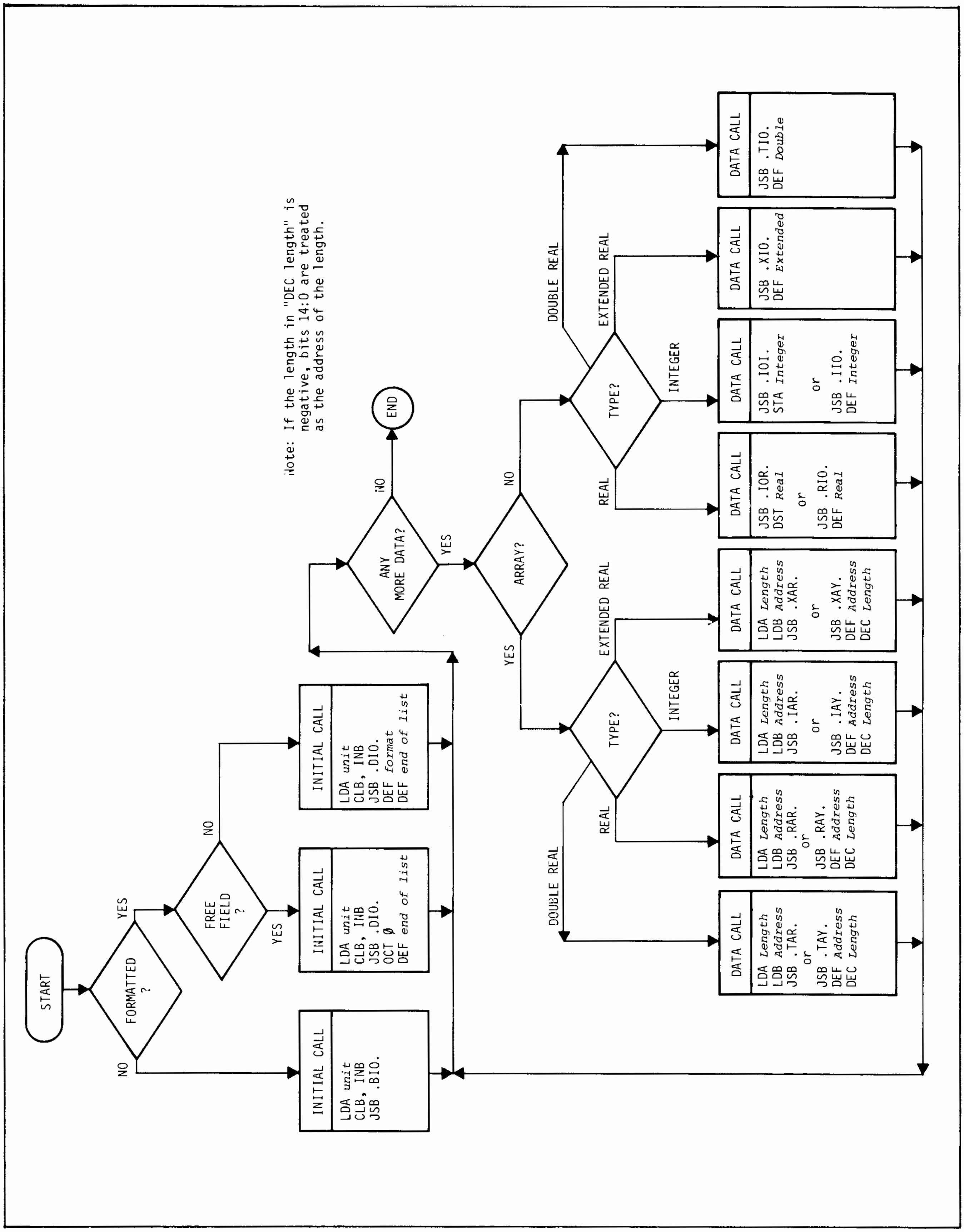
Figure 4-1 flowcharts the process of selecting an input calling sequence. Figure 4-2 flowcharts the output calling sequence.

Variable items in the calling sequences include:

unit	is the logical unit number of the desired I/O device.
format	is the label of an Assembly Language ASC pseudo-instruction that defines the format specification.
end of list	is the location following the last data call to the formatter. When an error occurs in the format specification or the input data, the formatter returns to this location.
real	is the address of the real variable.
integer	is the address of the integer variable.
double	is the address of the double precision variable
length	is the number of elements (not the number of memory locations) in the array. Maximum length of an external physical record may be specified by calling LGBUF. Otherwise the maximum external length is 67 words for formatted data and 60 words for binary data. Formatted data blocks can be of any length if the format breaks the data in multiple records using "/" and unlimited groups. If binary data exceeds 60 words, the record is read in or out and the formatter skips to the next record. (Note: For this reason, binary data should be read in with the same variable list as that used to write it out.)
address	is the first location of the array.

RECORDS

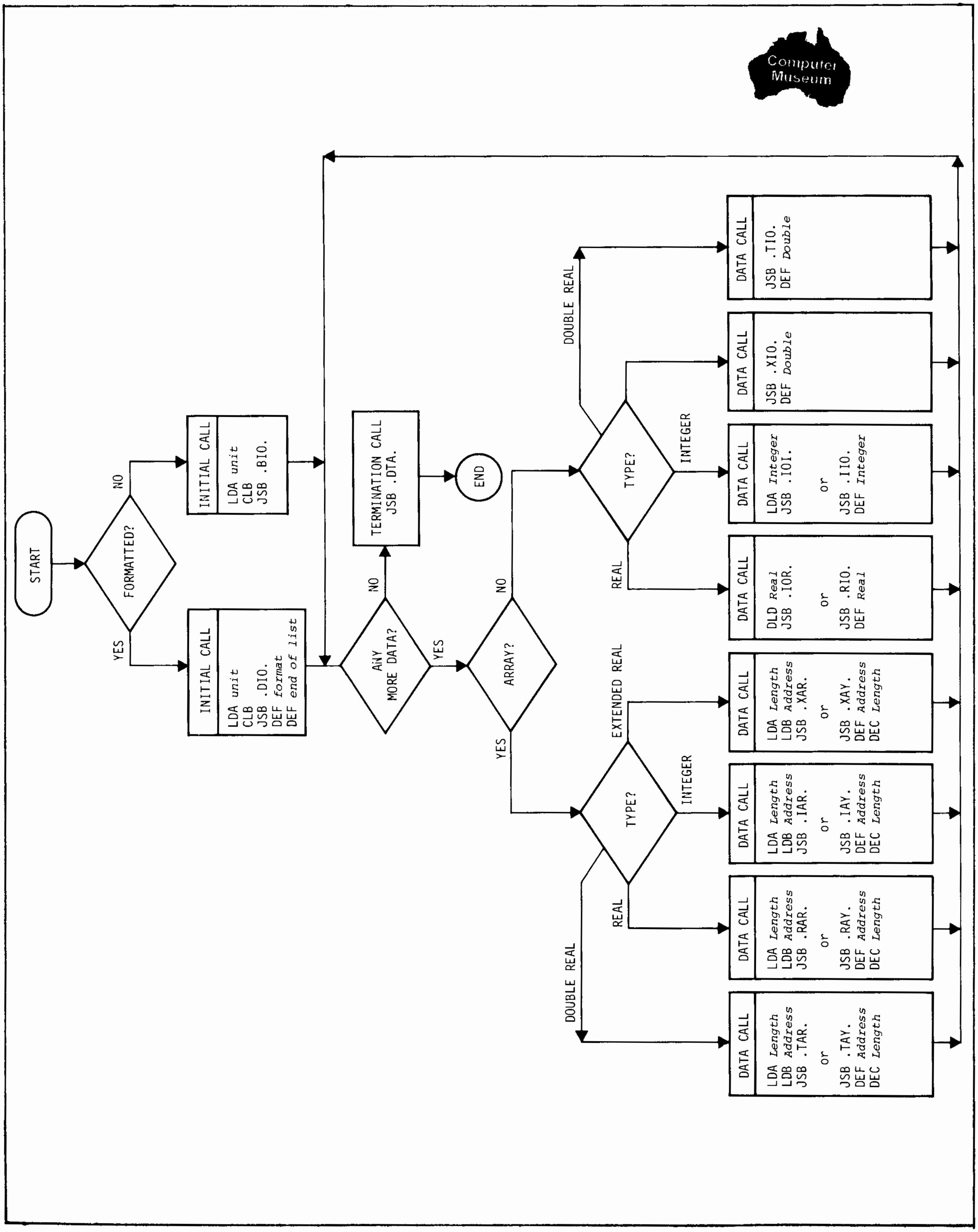
The formatter cannot be used for disc I/O. FMTIO does all input and output through calls to REIO. The subroutine LGBUF (see p. 4-34) can be used to specify the address and length of the I/O buffer. If the address and length of the I/O buffer are not given by calling LGBUF, a buffer within FMTIO will be used, and the maximum length will be 60 words for unformatted data or 67 words for formatted data.



Note: If the length in "DEC length" is negative, bits 14:0 are treated as the address of the length.

7700-422

Figure 4-1. Input Calling Sequence Selection



7700-423

4-2. Output Calling Sequence Selection

FORMATTED INPUT/OUTPUT

Formatted input/output is distinguished from unformatted input/output by the presence of an ASCII string format specification. (Refer to *format* definition in calling sequence items previously defined.) The ASCII characters consist of a series of format specifications or codes. Each code specifies either a conversion or an editing operation. Conversion specifications tell the formatter how to handle each variable in the data list.

Format specifications may be nested (enclosed in parenthesis) to a depth of one level. In the FORTRAN IV formatter they may be nested to a depth of four levels. Conversion specifications tell the formatter how to convert variables into ASCII output and how to convert ASCII input into binary variable data. Editing specifications tell the formatter what literal strings to output, when to begin new records and when to insert blanks.

FORMAT SPECIFICATIONS

A format has the following form: (*spec*,...,r(*spec*,...),*spec*,...)

where:

spec is a format specification and r is an optional repeat factor which must be an integer.

Conversion Specifications

rEw.d	Real number with exponent (E specification)	
rFw.d	Real number without exponent (F specification)	
rIw	Decimal Integer (I specification)	
r@w	Octal Integer	O, K, and @ specification
rKw,rOw	Octal Integer	
rAw,rRw	ASCII character (A and R specifications)	
srDw.d	Double precision number with exponent (D specification)	FORTRAN IV formatters only
srGw.d	Real number with digits (G specification)	
rLw	Logical variable (L specification)	

Editing Specifications

nX Blank field

nH *character string*

r" *character string*"

r' *character string*'

r/ begin new record

Tn tab to space n

TLn tab left n spaces

TRn tab right n spaces

where:

r is an integer repetition factor

w and n are non-zero integer constants representing the width of a field in the external character string

d is an integer constant representing the digital fraction in the part of the string

s is an optional scale factor

E SPECIFICATION

The E specification defines a field for a real number with exponent.

Output

On output, the E specification converts numbers (integers, real, or double precision) in memory into character form. The E field is defined in a format by the presence of the E specification (Ew.d). The field is w positions in the output record. The variable is printed out in floating-point form, right justified in the field as

$$\overbrace{-.\underbrace{x_1 \ldots x_d}_{d}\ E\pm ee}^{w}$$

where

$x_1 \ldots x_d$ are the most significant digits of the value, the e's are the digits of the exponent w is the width of the field, d is the number of significant digits, and the minus sign is present if the number is negative.

The w must be large enough to contain the significant digits (d), the sign, the decimal point, E, and the exponent. In general, w should be greater than or equal to d + 6.

If w is greater than the number of positions required for the output value, the quantity is right justified in the field with spaces to the left. If w is not large enough (e.g., less than d + 6), then the value of d is truncated to fit in the field. If this is not possible, the entire field is filled with dollar signs ($).

EXAMPLES:

FORMAT	DATA ITEM	RESULT
E10.3	+12.34	^^.123E+02
E10.3	-12.34	^-.123E+02
E12.4	+12.34	^^^.1234E+02
E12.4	-12.34	^^-.1234E+02
E7.3	+12.34	.12E+02
E5.1	+12.34	$$$$$

Input

The E specification on input tells the formatter to interpret the next w positions in the record as a real number with exponent. The formatter then converts the field into a number and stores it into the variable specified in the variable list.

The input field may consist of integer, fraction, and exponent subfields

$$\overbrace{\pm n....n}^{\text{integer field}}\overbrace{.n....n}^{\text{fraction field}}\overbrace{E\pm\ ee}^{\text{exponent field}}$$

where the format equals Ew.d.

Rules for E Field Input:

1. The width of the input item must not be greater than w characters.
2. Initial + and E are optional.
   Example: 123. = +123., 12.+6 = 12.E6
3. If E is present, the initial + of the exponent is optional.
   Example: 123.4E06
4. If the decimal point is left out, the formatter inserts it by multiplying the integer field by 10^{-d}.
   Example: If format = E9.4, 123456E+6 = 12.3456E+6
5. Spaces are ignored in the FORTRAN formatter and 4K Formatter, but in the FORTRAN IV Formatter blanks are evaluated as zeroes (0).
6. Any combination of integer field, fraction field, and exponent field is legal:

```
123.456E6
   .456E6
   .456
123.E6
123.
    E6
(all blanks = 0)
```

NOTE: Input to F, G, D and I fields is interpreted in the same way as the E field.

F SPECIFICATION

The F Specification defines a field for a fixed point real number (no exponent).

Output

On output, the F specification converts numbers (integer, real, or double precision) in a format by the presence of the F specification (Fw.d). The field is w positions in the output record. The variable is printed out right-justified in fixed-point form with d digits to the right of the decimal point:

$$\underbrace{\overbrace{-x\ldots.x}^{\text{integer field}}.\overbrace{xx..x}^{\text{fraction field (d)}}}_{w}$$

Where w is the total width of the field, the negative sign (-) is optional (positive numbers are unsigned), d is the length of the fraction field (empty if d=0).

If w is greater than the number of positions required for the output value, the quantity is right justified in the field with spaces to the left. If w is not large enough to hold the data item, then the value of d is reduced to fit. If this is not possible, the entire field is filled with dollar signs ($).

Examples:	FORMAT	DATA ITEM	RESULT
	F10.3	+12.34	^^^^12.340
	F10.3	-12.34	^^^-12.340
	F12.3	+12.34	^^^^^^12.340
	F12.3	-12.34	^^^^^-12.340
	F4.3	+12.34	12.3
	F4.3	+12345.12	$$$$

Input

Input to an F field is identical to an E field. All the rules under the E specification apply equally to the F specification.

D SPECIFICATION

The D specification is available only on the FORTRAN IV formatter. The effect is exactly the same as using an E specification with exception that on output "D" begins the exponent field instead of "E".

Examples: D10.3
D12.4
D7.3

G SPECIFICATION

The G specification is available only with the FORTRAN IV formatter and defines an external field for a real number. The magnitude of the number determines whether or not there is an exponent field.

Output

On output, the G specification converts numbers (integer, real, or double precision) in memory into character form. The G field is defined in a format by the presence of the G specification (Gw.d). The field is w spaces wide, with d significant digits. The format of the output depends on the magnitude of the number (N):

Magnitude	Output Conversion
$0.1 \leq N<1$	F(w-4).d,4X
$1 \leq N<10$	F(w-4).(d-1),4X
⋮	⋮
$10^{d-2} \leq N<10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N<10^{d}$	F(w-4).0,4X
Otherwise	sEw.d (s is scale factor)

NOTE: The scale factor is applied only when the G conversion is done as E.

Sample Output:

The following real numbers are converted under a G10.3 specification:

Number	Output Format
.05234	‸‸.523E-01
.5234	‸‸.523‸‸‸‸
52.34	‸‸52.3‸‸‸‸
523.4	‸‸523.‸‸‸‸
5234.	‸‸.523E+04

Input

Input processing of a Gw.d specification is identical to that of an Ew.d specification.

OPTIONAL SCALE FACTOR (FORTRAN IV FORMATTER ONLY)

The optional scale factor for F,E,G, and D conversions is of the form:

nP

The scale factor, n, is an integer constant or a minus followed by an integer constant. Upon initialization of the formatter, the scale factor equals zero. Once a scale factor is encountered, it remains in effect for all subsequent F,E,G and D fields until another scale is encountered.

The scale factor effects are as follows:

1. F,E,G,D input (provided no exponent exists in the external field): internally represented number equals externally represented number times ten raised to the -nth power. That is, $IN=XN*10^{-n}$ where IN and XN represent internal and external numbers, respectively.
2. F,E,G,D, input with exponent field in external field: no effect.
3. F output: external number equals internal number times ten raised to the nth power. ie,

   $$XN = IN*10^{n}$$
4. E,D output: mantissa is multiplied by 10^{n} and the exponent is reduced by n. If $n \leq 0$, there will be -n leading zeroes and d + n significant digits to the right of the decimal point. If n>o, there will be n significant digits to the left of the decimal point and d-n + 1 to the right. The scale factor when applied to E and D output has the effect of shifting the decimal point to the left or right and adjusting the exponent accordingly. Note that when n > 0, there are d + 1 significant digits in the external field.
5. G output: If F conversion is used, the scale factor has no effect. If E conversion is used, the scale factor has the same effect as with E output.

Examples of
Input conversion:

External field	Format	Internal number
528.6	1PF10.3	52.86
.5286E+03	1PG10.3	528.6
528.6	-2PD10.3	52860.

Examples of
Output conversion:

Internal number	Format	External field
528.6	1PF8.2	‸5286.00
.5286	2PE10.4	52.860E-02
5.286	-1D10.4	‸.0529D+02
52.86	1PG10.3	‸‸52.9‸‸‸‸
-5286.	1PG10.3	-5.286E+03

I SPECIFICATION

The I specification defines a field for decimal integer.

Output

On output, the I specification converts numbers (integer, real, or double precision) in memory into character form. The I field is defined in a format by the presence of the I specification (Iw). The field occupies w positions in the output record. The variable is converted to an integer, if necessary, and printed out right-justified in the field (spaces to the left) as:

$$\underbrace{\cdots \wedge \overset{\wedge}{-}\, x,\ \ldots x_d}_{w}$$

where

x, ... x_d are the digits of the value, (max = 5), w is the width of the field in characters, and the minus sign (-) is present if the number is negative.

If the output field is too short, the field is filled with dollar signs ($).

Format	Data Item	Result
I5	-1234	-1234
I5	+12345	12345
I4	+12345	$$$$
I6	+12345	‸12345

Input

The I specification on input (Iw) is equivalent to an Fw.0 specifications. The input field is read in, the number is converted to the form suitable to the variable (integer, real, double real), and the binary value is stored in the variable location.

During input, if a value is less than -32768_{10}, the value is converted to +32767.

Examples:

Format	Input Field	Internal Result
I5	-‸123	-123
I5	12003	12003
I4	‸102	102
I1	3	3

O,K,@ SPECIFICATION (NOT AVAILABLE WITH 4K FORMATTER)

These three specification types (O,K,@) are equivalent; they are all used to convert octal (base eight) numbers.

Output

On output, the octal specification (O,K,@) converts an integer value in memory into octal digits for output. The octal field is defined in a format by the presence of the O(Ow), K(Kw), or @(@w) specification. The field is w octal digits wide. The integer value is converted and right justified in the field as:

$$\underbrace{\ldots\ldots_{\wedge\wedge} d_1 \ldots\ldots d_n}_{w}$$

where

$d_1 \ldots\ldots d_n$ are the octal digits (6 maximum), $\ldots_{\wedge\wedge}$ are lead spaces, and w is the width.

If w is less than 6, the w least significant octal digits are written.

Input

On input, the octal specification tells the formatter to interpret the next w positions in the input record as an octal number. The formatter converts the digits into an octal integer and stores it into an integer variable.

If w is greater than or equal to six, up to six octal digits are stored; non-octal digits with the field are ignored.

If w is less than six or if less than six octal digits occur in the field, the result is right-justified in the variable with zeroes (0) to the left.

If the value of the octal digits in the field is greater than 177777, the results are unpredictable.

Examples:	Format	Input Field	Internal Result
	@6	123456	123456
	@7	-123456	123456
	2K5	2342342342	023423 and 042342
	2@4	,396E-05	000036 and 000005

L SPECIFICATION

The L specification is available only with the FORTRAN IV formatter and allows input or output of logical values:

TRUE = T (external), negative (internal)
FALSE = F (external), non-negative (internal)

Output

On output, the L specification converts numbers (integer, real, or double precision) in memory into their external logical value (T or F). The L field is defined by the presence of the L specification (Lw). The field is w spaces wide, consisting of w-1 blanks followed by a T or F.

Input

On input, the L specification converts an external character field into the internal representation of true or false. The L specification (Lw) specifies a field w spaces wide, consisting of optional blank, a T or F and optional trailing characters. A T is converted to -32,768 (100000_8) and an F is converted to 0.

A AND R SPECIFICATIONS

The A and R specifications define a field of one two eight ASCII characters. ASCII characters are stored as two 8-bit codes per integer variable, four 8-bit codes per real variable, six per extended real, and eight per double real.

The number of characters per variable will always be referred to as "v".

Output

On output, the A and R specifications transfer ASCII character codes from memory to an external medium. The field is defined by an A or R specification (Aw or Rw). The field is w positions wide in the output record. For $w \geq v$, A and R are equivalent: the field is blank filled to the left of the data. For w > v, the A specification uses the left-most characters in the variable, and the R specification (and A if OLDIO) uses the right-most.

Examples:	Variable	Format	Output Format
	ABCD	A4 & R4	ABCD
	ABCD	A6 & R6	^^ABCD
	ABCD	A3	ABC
	ABCD	R2	CD

A string of n*v characters may be output from (or input to) n variables (e.g. using an array of length n) using a repeat factor.

Examples:

Variable Type	Variables	Format	Input or Output
4 integers	AB, CD, EF, GH	4A2	ABCDEFGH
2 reals	ABCD, EFGH	2A4	ABCDEFCH
1 double real	ABCDEFGH	A8	ABCDEFGH

Input

On input, the A and R specifications transfer ASCII character codes from an external medium to internal memory. The field is defined by an A or R specification (Aw or Rw). The field is w positions wide. If w ≥ v, the right most two characters are taken from the input field.

For the A specification with w<v, data is left-justified and blank filled in the variable. For the R specification (and A if OLDIO) with w<v, data is right-justified and zero-filled.

Examples:

Input Field	Format	Real Variable	
MN	A2	MN^^	
MN	R2	zzMN	z = binary zero
MNOP	A4,R4	MNOP	
MNOPQRS	A7,R7	PQRS	

In order to read in a string of more characters than fit in the data type used, the repeat factor must be used.

Examples:	Input Field	Format	Variable
real	MNOPQRSTUVWX	3A4	MNOP,QRST,UVWX
integer	FGHIJK	3A2	FG,HI,JK

For w < the variable size, the FORTRAN IV and FORTRAN Formatter differ.

FORTRAN Formatter

In FORTRAN the A is the same as the R. For w = 1, A and R read in one character and places it in the right half of the variable with binary zeroes in the left.

Example:	Input	Format	Variable	
	X	A1 or R1	00000000_2	X
			left	right

computer word

FORTRAN IV Formatter

The R specification is the same as in the FORTRAN Formatter.

For A1, one character is read in and placed in the left half of the computer word. An ASCII blank is placed in the right half.

Example:	Input	Format	Variable
	X	A1	X‸

To Insure Compatibility with previous software:

The Formatter can be modified at run-time to interpret the A specification as the R specification. This is done by calling the OLDIO entry point:

```
CALL  OLDIO
```

To change back to a FORTRAN IV A specification call NEWIO:

```
CALL  NEWIO
```

The Formatter always begins operation in the NEWIO state.

X SPECIFICATION

The X specification produces spaces on output and skips characters on input. The comma (,) following X in the format is optional.

Output

On output, the X specification causes spaces to be inserted in the output record. The X field is defined by the presence of an X specification (nX) in the format, where n is the number of spaces to be inserted. (X alone = 1X; 0X is not permitted.)

Examples:

Format

`E8.3,5X,F6.2,5X,I4`

Data Values

+123.4, -12.34, -123

Output Field

`.123E+03‸‸‸‸‸-12.34‸‸‸‸‸-123`

Input

On input, the X specification causes characters to be skipped in the input record. The X field is defined by the presence of an X specification (nX) in the format, where n is the number of characters to be skipped. (X alone = 1X; 0X is not permitted.)

Examples:

Format

`8X,I2,10X,F4.2,10X,F5.2`

Input Field

`WEIGHT ‸‸10‸‸PRICE‸‸$1.98‸‸TOTAL‸‸$19.80`

Internal Values

10, 1.98, 19.80

' ', " ", H SPECIFICATIONS (LITERAL STRINGS)

The H and quotation mark specifications provide for the transfer, without conversion, of a series of ASCII characters (except that quotation marks cannot be transferred using " " or ' '). A comma after this specification is optional.

Output

On output, the ASCII characters in the format specification (there is no associated variable since this is only an editing specification) are output as headings, comments, titles, etc. The specifications are of the form:

$$nHc_1c_2\ldots c_n \quad \text{or} \quad "c_1c_2\ldots c_n" \quad \text{or} \quad 'c_1c_2\ldots c_n'$$

where

n is the numbers of characters to be transmitted, $c_1c_2\ldots c_n$ are the characters themselves, and H or the quotation marks are the specification types.

(H alone = 1H; 0H is not permitted.)

Note that with quotation marks, the field length is not specified; that is determined by the number of characters between the quotation marks.

Examples:

Format	Result
20H‸THIS‸IS‸AN‸EXAMPLE	‸THIS IS AN EXAMPLE
"THIS‸ALSO‸IS‸AN‸EXAMPLE"	THIS‸ALSO‸IS‸AN‸EXAMPLE
3"ABC"	ABCABCABC
3("ABC")	ABCABCABC
2'ABCD'	ABCDABCD

Input

If H is used on input, the number of characters needed to fill the specification is transmitted from the input record to the format. A subsequent output statement will transfer the new heading to the output record. In this way, headings can be altered at run-time.

If quotation marks are used on input, the number of characters within the quotation marks is skipped on the input field.

Example:

Format

```
31H‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸‸
```

Input

```
H‸INPUT‸ALLOWS‸VARIABLE‸HEADERS
```

Result

```
31HH‸INPUT‸ALLOWS‸VARIABLE‸HEADERS
```

/ SPECIFICATION

The / specification terminates the current record. The / may appear anywhere in the format and need not be set off by commas. Several records may be skipped by preceding the slash with a repetition factor (r-1 records are skipped for r/).

On output, a new record means a new line (list device), a carriage return-linefeed (punch device), or an end-of-record (magnetic tape). Formatted I/O records can be up to 67 words (134 characters) long.

On input, a new record is a new "unit record" (card reader), is terminated by a carriage return-linefeed (teleprinter), or is terminated by an end-of-record (magnetic tape).

> *NOTE: When the formatter reaches the end of a format and still has values to output, it starts a new record.*

Examples:

Format

```
22X,6HBUDGET/// 6HWEIGHT,6X, 5HPRICE,9X,
5HTOTAL,8X
```

Result

```
(line 1) ^^^^^^^^^^^^^^^^^^^^^^BUDGET
(line 2)
(line 3)
(line 4) WEIGHT ^^^^^^PRICE^^^^^^^^^TOTAL^^^^^^^^
```

HOW TO PUT FORMATS TOGETHER

1. When two specifications follow each other they are concatenated.

   Format: E9.4,I6

E field	I field
9 characters	6 characters

2. To leave space between numbers use X.

   Format: E9.4,3X,I6

E field	X	I field
9 characters	3 characters	6 characters

3. To start a new Line, use /

   Format: E9.4/I6

E field
9 characters

I field
6 characters

4. Specifications can be gathered together into groups and surrounded by parentheses.

   Example: (E9.3, 2X, I6)

E	X	I

These groups can be nested one level deep, except in the FORTRAN IV Formatter they can be four levels deep. For example,

(E9.3,3(2X,I6))

E	X	I	X	I	X	I

(E9.3,3(2X,I6),2(2X,I8))

E	X	I	X	I	X	I	X	I	X	I

5. Use the repetition factor to repeat single specifications (except nH) or groups of specifications. This is done by preceding the specification or parenthetical groups with a repeat count, r. The conversion is repeated up to r times, unless the list of variables is exhausted first.

   3(E9.3,2X,I6,2X)/

E	X	I	X	E	X	I	X	E	X	I	X

E	X	I	X	E	X	I	X	E	X	I	X

6. Use the principle of unlimited groups -- when the formatter has exhausted the specifications of a format and still has list items left, it inputs a new record for a READ or outputs the present record for a WRITE and returns to the last, outer-most unlimited group within the format. An unlimited group is a set of specifications enclosed In parenthesis. If the format has no unlimited groups, the formatter returns to the beginning of the format.

Example: Format = (I5,2(3X,F8.4,8(I2))

Format = (I5,2(3X,F8.4,8(12I2)),4X,3(I6))

Format = (I5,3X,4F8.4,3X)

7. Keep in mind the accuracy limitations of your data. Although the formatter will print out or read in as many digits as specified, only certain digits are significant:

   Integer variables can be between $-32,768_{10}$ and $+32,767_{10}$.
   Floating-point numbers can guarantee 6 digits of accuracy (plus exponent).
   Double precision can guarantee 11 digits of accuracy (plus exponent).

8. On input to the FORTRAN IV formatter blanks are interpreted as zero digits, while on input to the FORTRAN Formatter, blanks are not evaluated as part of the data item.

   The FORTRAN IV Formatter can be made to act exactly as the FORTRAN Formatter does by calling entry point OLDIO. This condition can be reversed by calling entry point NEWIO. These calls are made in FORTRAN as:

```
CALL OLDIO
CALL NEWIO
```

   In Assembly Language as:

```
JSB OLDIO     JSB NEWIO
DEF *+1       DEF *+1
→             →
```

FREE FIELD INPUT

When free field input is used, a format specification is not used. Special symbols are included within the input data to direct the conversion process:

space or,	Data item delimiters
/	Record terminator
+ -	Sign of item
. E + - D	Floating point number
@	Octal integer
"..."	Comments

All other ASCII non-numeric characters are treated as spaces (and delimiters). Free field input may be used for numeric data only.

DATA ITEM DELIMITERS

Any contiguous string of numeric and special formatting characters occurring between two commas, a comma and a space, or two spaces, is a data item whose value corresponds to a list element. A string of consecutive spaces is equivalent to one space. Two consecutive commas indicate that no data item is supplied for the corresponding list element; the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

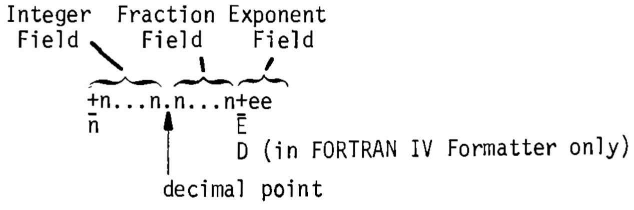
Example:

1) Input data: 1720, 1966, 1980, 1392
Result in memory: 1720
1966
1980
1392

2) Input data: 1266,, 1794, 2000
Result in memory: 1266
1966 (Value of 1966 is unchanged)
1794
2000

FLOATING POINT INPUT

The symbols used to indicate a floating point data item are the same as those used in representing floating point data for Format specification directed input:



If the decimal point is not present, it is assumed to follow the last digit.

Example: Input Data: 3.14, 314E-2, 3140-3, .0314+2, .314E1

All are equivalent to 3.14

OCTAL INPUT

An octal input item has the following format:

$@x_1 \ldots x_d$

The symbol @ defines an octal integer. The x's are octal digits each in the range of 0 through 7. List elements corresponding to the octal data items must be type integer.

RECORD TERMINATOR

A slash within a record causes the next record to be read as a continuation of the data list; the remainder of the current record is skipped as comments.

Example: Input data: 987, 654, 321, 123/DESCENDING
456

Result in memory: 987 654 321 123 456

COMMENTS WITHIN INPUT

All characters appearing between a pair of quotation marks in the same line are considered to be comments and are ignored.

Examples:	"6.7321"	is a comment and ignored
	6.7321	is a real number

INTERNAL CONVERSION

The Formatter provides the programmer with the option of using the conversion parts of the Formatter only without any input or output. This process is called "internal conversion."

On "input", ASCII data is read from a buffer and converted according to a format (or free field) into a variable list. (This is known as decoding.)

On "output", binary data is converted to ASCII according to a format and stored in a buffer. (This is known as encoding.)

Internal conversion ignores "/" specifications or unlimited groups. The concept of records does not apply during internal conversion.

OUTPUT CALLING SEQUENCE (BINARY TO ASCII CONVERSION): ENCODING

```
CLA
CLB
JSB .DIO.
DEF buffer (destination)
DEF format
DEF end of list
 :
Calls to define each variable
     (Same as regular calls)
 :
Termination Call
     (Same as regular calls)
```

where *buffer* is a storage area for the ASCII "output" to be stored into.

INPUT CALLING SEQUENCE (ASCII TO BINARY CONVERSION): DECODING

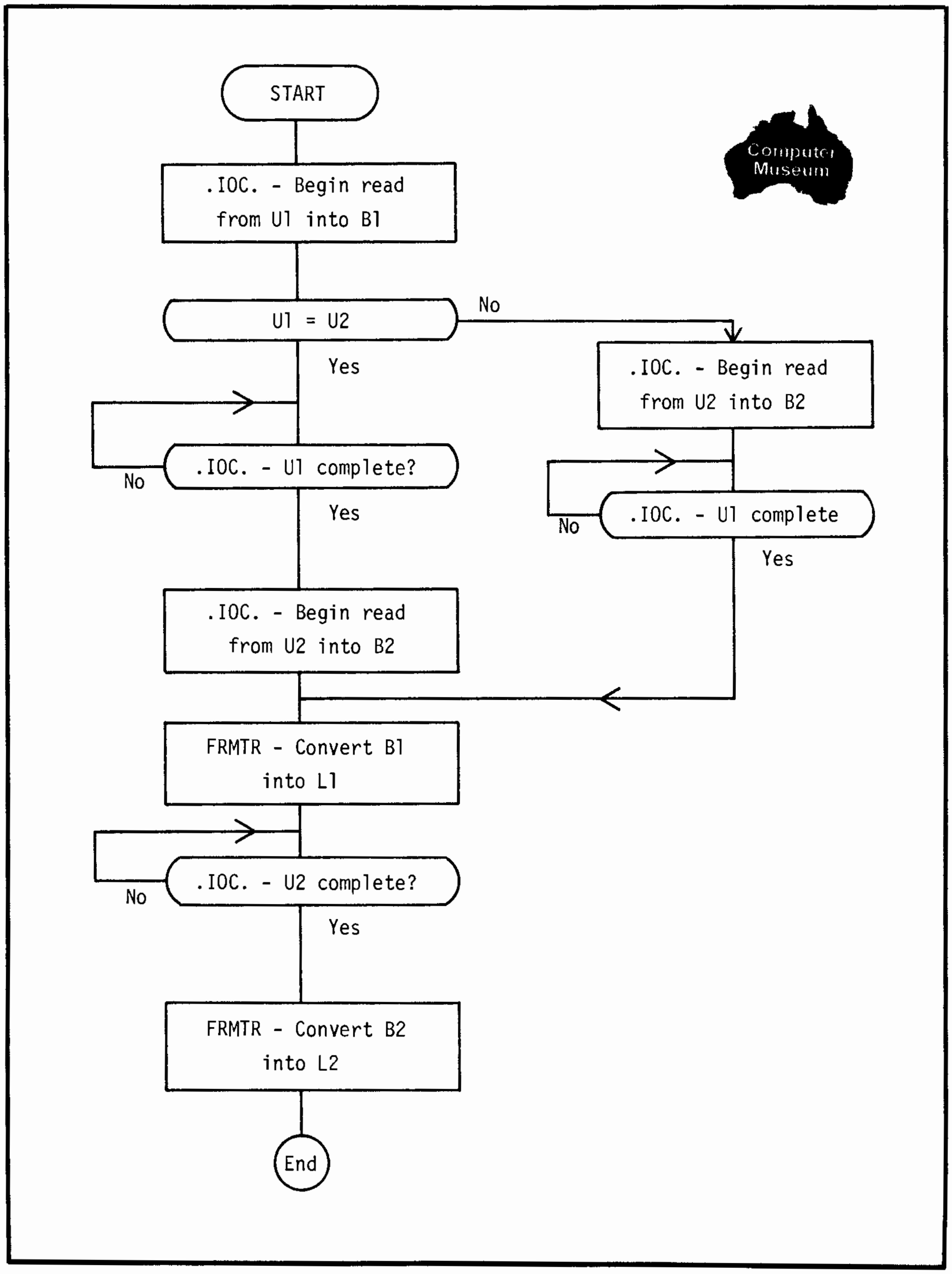
```
Formatter             Free Field

CLA                   CLA
CLB, INB              CLB, INB
JSB  .DIO.            JSB  .DIO.
DEF  buffer           DEF  buffer
DEF  format           ABS  0
DEF  end of list      DEF  end of list
 :                     :
Calls to define each variable
     (Same as regular calls)
```

where *buffer* is a storage area containing ASCII characters which will be converted by the Formatter into binary values.

BUFFERED I/O WITH THE FORMATTER

Normally, when a program uses the Formatter, it can only execute one I/O operation at a time. However, the internal conversion feature of the Formatter can be used with direct calls to .IOC. (through the MAGTP subroutine) to provide both buffered and formatter I/O.

The following flowchart shows how a program can read in data from two units (U1 and U2) into two buffers (B1 and B2) at the same time by calling .IOC.. When unit U1 is complete, buffer B1 is converted into list L1 by the Formatter (while input continues on unit U2).



EXAMPLE CALLING SEQUENCES

EXAMPLE 1: FORMATTED INPUT

Purpose

A 20 character double precision number and a 10 character integer are read and converted from the first record. 80 characters are read from the second record and stored in ASCII form in the array ALPHA. Execution continues with the instruction at ENDLS.

```
          LDA     INPUT                          Input unit number
          CLB,INB                                Input flag
          JSB     .DIO.                          Initialization enterance
          DEF     FMT                            Location of format
          DEF     ENDLS                          End of list
          JSB     .XIO.                          Declare double precision variable
          DEF     DP                             Location of variable
          JSB     .IIO.                          Declare integer variable
          DEF     I                              Location
          JSB     .IAY.                          Declare integer array
          DEF     ALPHA                          Location
          DEC     80                             Number of elements
    ENDLS →                                      (Continue program here)
          :
    INPUT DEC     1                              Unit number
    DP    BSS     3                              Double precision variable
    I     BSS     1                              Integer variable
    ALPHA BSS     80                             Integer array
    FMT   ASC     9,(D20.12,I10/80A1)            Format specification
```

EXAMPLE 2: UNFORMATTED OUTPUT

Purpose

1000 2-word elements in the array ARRAY are punched on the standard punch unit. The output will consist of 60 word records (59 data words and 1 control word) until the entire array is punched.

```
       LDA  PUNCH          Output unit number
       CLB                 Output flag
       JSB  .BIO           Binary initialization enterance
       LDA  =D1000         Number of elements in array
       LDB  ADRES          Location of array
       JSB  .RAR.          Real (2-word) array enterance
       JSB  .DTA.          Output termination
       →
        .
        .
        .
PUNCH  DEC  4              Unit number
ADRES  DEF  ARRAY          Location of ARRAY
ARRAY  BSS  2000           Defines 1000 2-word elements.
```

EXAMPLE 3: INTERNAL CONVERSION AND FREE FIELD INPUT

Purpose

The ASCII data starting at BUFFR is converted in free field form to binary. R will contain the binary representation of .0001234 and I will contain the binary representation of 28.

```
       CLA                           Internal conversion flag
       CLB,INB                       ASCII to binary flag
       JSB      .DIO.                Initialization enterance
       DEF      BUFFR                Location of ASCII data
       ABS      0                    Specifies ASCII data is in free-field form
       DEF      ENDLS                End of list
       JSB      .IOR.                Declare real variable
       DST      R                    Store binary item in R
       JSB      .IOI.                Declare integer variable
       STA      I                    Store in I
ENDLS  →

       .
       .
       .
R      BSS      2                    Real variable
I      BSS      1                    Integer variable
BUFFR  ASC      6,123.4E-6,28        ASCII data to be converted to binary.
```

FMT.E

PURPOSE: Provides ability to change output LU # for FMTIO
Routine .FMT.E is defaulted by 6.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS: FMT.E

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
EXT FMT.E
LDA LU  Desired LU
STA FMT.E
```

METHOD: Method: A zero value for FMT.E will cause error messages to be inhibited.

ATTRIBUTES:

ENTRY POINTS:

	FMT.E
Parameters:	Logical Unit Number
Result:	
FORTRAN:	Not Callable
FORTRAN IV:	Not Callable
ALGOL:	Not Callable
Errors:	

FMTIO

PURPOSE: Provides internal conversion according to a FORMAT from one memory area to another memory area.

PROGRAM TYPE = 7 **ROUTINE IS: U**

ENTRY POINTS:	(FMTIO ENTRY POINTS - see page 4-30)		
EXTERNAL REFERENCES:	(FMTIO EXTERNAL REFERENCES - see page 4-30)		
CALLING SEQUENCES:	JSB CODE DEF *+1 Read or write request (see Note 1)	or	JSB CODE DEF *+2 DEF ICHRS

METHOD: Utilizes the internal conversion capability of the Formatter.

ATTRIBUTES:

ENTRY POINTS:

	CODE	ACODE
Parameters:	None	None
Result:	None	None
FORTRAN:	Callable	Callable
FORTRAN IV:	Callable	Callable
ALGOL:	Not callable (Note 2)	Callable (Note 2)
Errors:	None	None

NOTES:

1. The call to CODE must immediately precede a READ or WRITE request where the identifier of an ASCII record buffer replaces the logical unit number. Any labels must be attached to the CODE call, as the CODE call and the READ/WRITE call are treated as one statement.

   In FORTRAN the calling sequences are:

CALL CODE (ICHRS)	CALL CODE
READ (*v*,*n*)*L*	WRITE (*v*,*n*)*L*

COMMENT

where *v* is the unsubscripted identifier of an ASCII record buffer;

n is the number of a FORMAT Statement; and

L is an Input/Output List of variables.

ICHRS is an optional parameter which limits the size of the buffer the Formatter will read to satisfy the variable list. Typically ICHRS would equal the number of ASCII characters in buffer V. If ICHRS is not specified the Formatter will search all of memory, if necessary, to satisfy the external.

On read, the contents of the ASCII record v are converted according to the FORMAT n and are stored in the variables listed in L.

On write, the contents of the variables listed in L are converted to ASCII according to FORMAT n and the ASCII characters are stored in v.

1a. Two other interesting routines: ITLOG & ISTAT

```
    JSB ITLOG                        ICHRS = ITLOG(IXXXX)
    DEF *+1
    STA ICHRS
WHERE:
  ICHRS = THE NUMBER OF CHARACTORS READ OR WRITTEN BY THE FORMATTER
    BY ITS LAST INPUT/OUTPUT REQUEST TO THE SYSTEM.  " ICHRS " VALUE
    WILL BE 0 TO 134 (120 OF BINARY) REGARDLESS OF THE SPECIFIED
    BUFFER SIZE IN THE READ OR WRITE STATEMENT.
  IXXXX = THE SAME AS " ICHRS "
    JSB ISTAT                        ISTUS = ISTAT(IXXXX)
    DEF *+1
    STA ISTUS

WHERE:
    ISTUS = THE STATUS WORD RETURNED FROM THE EXEC IN THE LAST
    INPUT/OUTPUT CALL THE FORMATTER DID.
    IXXXX = SAME AS " ISTUS "
```

1b. EXAMPLES

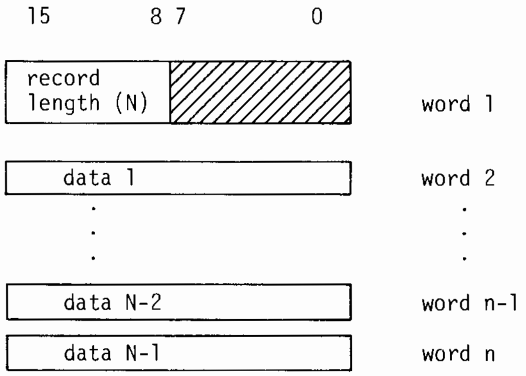
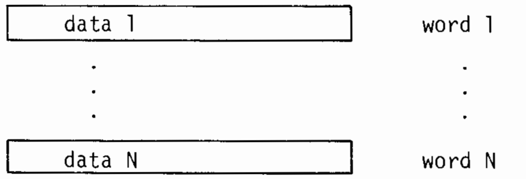
```
EXAMPLE:   CODE
    CALL EXEC (1,401B,IBUFR,-80)
    CALL ABREG(IA,ICHRS)
    CALL CODE(ICHRS)
    READ(IBUFR,*) A,B,C,D

EXAMPLE:   ITLOG
  5 READ (1,10) (IBUF(I),I=1,36)
 10 FORMAT (36A2)
    IF (ITLOG(ICHRS)) 20,5,20
 20 ISTRC = 1
    CALL NAMR(IPBUF,IBUF,ICHRS,ISTRC)

NOTE:  ICHRS CAN BE AS LARGE AS 134 IF 134 CHARACTERS ARE INPUT.

EXAMPLE:   ISTAT
    READ (8,10) (IBUF(I),I=1,80)
 10 FORMAT (40A2)
    IF (IAND(ISTAT(ISTUS),240B)) 99,20,99
 20 CONTINUE
     ---
 99 CONTINUE (END OF FILE OR END TAPE DETECTED)
```

1c.

Note 1: The result of ITLOG is always given as the number of bytes transferred. For unformatted (binary) I/O to type 0 - 17B devices (teletype, cartridge tape, paper tape punches), the transmission log includes two bytes of record length information. (See Binary Record Format, p 4-30a.)

Note 2: Both the transmission log and the device status are meaningless if the device is buffered.

Binary Record Format

Device Type	Format
0 - 17B (0 < N < 256)	 15 8 7 0 record length (N) — word 1 data 1 — word 2 . . . data N-2 — word n-1 data N-1 — word n
20 - 77 (0 < N < 32767)	 data 1 — word 1 . . . data N — word N

Integer, logical, or ASCII variables require one data word. Real variables require two data words. Extended precision reals require three data words. Double precision reals require four data words.

2. ALGOL programmers must use the entry point ACODE instead of CODE. ACODE ROUTINE does not handle ARRAYS. The following is an example of how to handle ARRAYS.

```
HPAL,L,"TEST"
BEGIN
INTEGER ARRAY B[1:3]; INTEGER I, INPUT:=12345;
OUTPUT LST1(INPT);
FORMAT F1(I6);
PROCEDURE ACODE;
   CODE;
PROCEDURE ACODEWRITE(BUFFER, FRMT, LIST);
   INTEGER BUFFER; FORMAT FRMT; OUTPUT LIST;
   BEGIN
   ACODE;
   WRITE(BUFFER, FRMT, LIST)
   END;
ACODEWRITE(B[1], F1, LST1);
WRITE(1, #(" RESULT:",3A2), FOR I:=1 TO 3 DO B[I])
END$
```

FMTIO Entry Points

The entry points are:

.RIO.
.IIO.
.XIO.
.XAY.
.RAY.
.IAY.
.DIO.
.BIO.
.IOI.
.IOR.
.IAR.
.RAR.
.DTA.
.TIO.
NEWIO
OLDIO
CODE
ACODE
ITLOG
ISTAT
.TAY.
.XAR.
.TAR.
LGBUF

FMTIO External References

The external references are:

EXEC
.FRMN
.LS2F
.INPN
.DTAN
FMT.E
REIO
PNAME
.SBT

FRMTR

PURPOSE: This routine is the re-entrant portion of the Formatter. Its entry points are only callable by the routine FMTIO. FRMTR contains the various type conversion routines (D, R, K, E, L, etc.)

PROGRAM TYPE = 6 **ROUTINE IS: R**

ENTRY POINTS:	.FRMN,.LS2F,.INPN,.DTAN
EXTERNAL REFERENCES:	.ZRNT,.XPAK, .LBT,.SBT
CALLING SEQUENCES:	Only callable from FMTIO

LGBUF

PURPOSE: Can be used to specify the address and length of the I/O buffer.

ENTRY POINTS: None

EXTERNAL REFERENCES: None

CALLING SEQUENCES:

```
JSB LGBUF
DEF *+3
DEF IBUF
DEF ILNG
→
ILNG DEC N
IBUF BSS N
```

ATTRIBUTES:

	LGBUF
Parameters:	
Result:	
Fortran:	Callable
Fortran IV:	Callable
Algol:	Callable

Note 1: For devices type 0 - 17B, ILNG should not exceed 255. For other devices, ILNG may be in the range 1 - 32767. (See Binary Record Format, p. 4-31.)

Note 2: If a line of I/O exceeds the buffer size ILNG, the access is lost. Unformatted output, however is an exception; data cannot be lost (multiple records not exceeding ILNG will be output).

Note 3: If LGBUF is to be called in a segment, the buffer must be dimensioned in common in the main program and all the segments.

Example:

```
      DIMENSION IBUF(500)
      DIMENSION JOB(100)
       :
      CALL LGBUF(IBUF,500)
       :
      WRITE(8,100)(JOB(I),I=1,100)
 100  FORMAT(100I10)
```

APPENDIX A

RUN TIME ERROR MESSAGES

APPENDIX A
RUN TIME ERROR MESSAGES

During execution of programs referencing Relocatable Library Subroutines, error messages may be generated. Error messages are listed together with the subroutine involved. The list LU is defaulted to LU6. To change the list LU, refer to the routine ER0.E.

Mathematical Subroutines

Error messages are printed in the form:

program name nn xx

program name	is the name of the user program where the error was encountered.
nn	is a number in the range 02 through 15 which identifies the subroutine involved in the error condition.
xx	is the error type, as follows: OF = Integer or Floating Point Overflow OR = Out of Range UN = Floating Point Underflow

These error messages can occur when system intrinsics are called or during an exponentiation operation. Suppose X and Y are real values and I and J are integers. Then, the following relocatable subroutines are called for these computations:

X**Y	.RTOR	(real to real)
X**I	.RTOI	(real to integer)
I**J	.ITOI	(integer to integer)

The following is a summary of possible error messages:

Error Message	Issuing Subroutine	Where Used	Error Condition
02-UN	ALOG	ALOG	$X \le 0$
		ALOGT	$X \le 0$
		CLOG	$X = 0$
		DLOG	$X \le 0$
		DLOGT	$X \le 0$
		.LOG	$X \le 0$
		.LOG0	$X \le 0$
		.LOGT	$X \le 0$
03-UN	SQRT	SQRT	$X < 0$
	DSQRT	DSQRT	
	.SQRT	.SQRT	

Error Message	Issuing Subroutine	Where Used	Error Condition
04-UN	.RTOR	.RTOR	$X = 0, Y \le 0$ $X < 0, Y \ne 0$
05-OR	SIN COS	SIN CSNCS CEXP COS	X outside [-8192*π, +8191.75*π]
06-UN	.RTOI	.RTOI	$X = 0, Y \le 0$
07-OF	EXP	EXP	$X * \log_2 e \ge 127$
	.EXP	CEXP	$X_1 * \log_2 e \ge 127$
		.RTOR	$\mid X * ALOG(X) \mid \ge 127$
		CSNCS	$X_2 * \log_2 e \ge 127$
		.EXP	$X \cdot \log_2 e \ge 127$
		.TTOT .TTOR .RTOT	$X^Y \ge 2^{127}$
08-UN	.ITOI	.ITOI	$I = 0, J \le 0$
08-OF	.ITOI	.ITOI	$I^J \ge 2^{23}$
09-OR	TAN	DTAN TAN .TAN	$X > 2^{14}$
10-OF	DEXP	DEXP	$e^X > (1-2^{-39})\ 2^{127}$
		.DTOD .DTOR .RTOD	$X > (1-2^{-39})\ 2^{127}$
11-UN	DLOG	DLOG DLOGT	$X \le 0$ $X < 0$
12-UN	.DTOI	.DTOI .TTOI	$X = 0, I \le 0$
13-UN	.DTOD	.DTOD .DTOR .RTOD .RTOT .TTOR .TTOT	$X = 0, Y \le 0$ $X < 0$
14-UN	.CTOI	.CTOI	$X = 0, I \le 0$
15-UN	.ATN2	.ATN2	

Format Errors

During execution of the object program error messages may be printed on the output unit by the input/output system supplied for FORTRAN programs. The error message is printed in the form:

FMT ERR *nn* program name

nn	is the error code.
program name	is the name of the user program.

The following is a summary of the FMT error codes:

Error Code	Explanation	Action
01	FORMAT ERROR: a) w or d field does not contain proper digits. b) No decimal point after w field. c) w - d <= 4 for E-specification.	Irrecoverable error; program must be recompiled.
02	a) FORMAT specifications are nested more than one level deep. b) A FORMAT statement contains more right parentheses than left parentheses.	Irrecoverable error; program must be recompiled.
03	a) Illegal character in FORMAT statement. b) Format repetition factor of zero. c) FORMAT statement defines more character positions than possible for device.	Irrecoverable error; program must be recompiled.
04	Illegal character in fixed field input item or number not right-justified in field.	Verify data.
05	A number has an illegal form (e.g., two Es, two decimal points, two signs, etc.).	Verify data.

APPENDIX B

RTE DEBUG LIBRARY SUBROUTINE

DEBUG, a utility subroutine of the RTE-DOS Relocatable Library is appended to the user's main program and to each segment by the loader when the appropriate loader option is set, and allows programs to be checked for logical errors during execution.

After the user's program is loaded with DEBUG appended to it, the user turns his program on with either of the following commands:

RU,*name,lu*   ON,*name,lu*

Where:

name	is the program name.
lu	is the logical unit of the console to be used for interactive commands.

Programs that expect starting parameters or that call RMPAR may require a special version of the module DBGLU, which determines the console lu.

The primary entry point of the program and of each segment (the location where execution begins) is set to DEBUG so that when the program is turned on, or a segment is entered, DEBUG takes control and printes a message:

```
    BEGIN "DEBUG"
or:
    BEGIN SEGMENT
```

You can then enter any legal debug operation. Illegal requests are ignored and a message is printed.

```
    ENTRY ERROR
```

The following commands describe DEBUG operation.

ABORT

A	Abort DEBUG operation. The program is set dormant.

BREAKPOINT

B,*n*	Instruction breakpoint at octal address *n*.

When the program reaches the breakpoint, execution is interrupted and the following message is printed:

P = v_1 I = v_2 A = v_3 B = v_4 E = v_5 O = v_6 MA = v_7 MC = v_8

The v's are octal values of registers and memory locations as follows:

P - P-Register (instruction address)
I - Instruction (contents)
A - A-Register
B - B-Register
E - E-Register
O - Overflow
MA - Effective operand address of a memory reference instruction
MC - Contents of effective address of a memory reference instruction

The breakpoint address *n* is relative to the program relocation base. P and MA are relative to the program relocation base if preceded by 'M+'. (See "M" command). Any legal DEBUG control statement may then be entered. The displayed instruction will be executed when the "R" command is entered.

The instruction may be modified with the "S" or "W" commands prior to entering the "R" command.

Three possible cases will prevent the instruction's execution until the breakpoint is cleared:

1) If the instruction will cause a memory protect violation (for example, JSB EXEC, DST 1B, JMP 100B, etc.), then the message 'MEM PROTECT' is displayed.
2) If the instruction is not in the HP 2100 instruction set (for example CAX, MWF, user-defined microcode, unimplemented instruction, etc.), then the message '?INSTR?' is displayed.
3) If the memory address cannot be resolved (more than 24 levels of indirect addressing), then the message 'INDIRECT LOOP' is displayed.

A maximum of fifteen breakpoints may be set at a time in the main program. An additional fifteen may be set in the current segment. Note that when one segment is overlayed by another, any memory modifications ('S' or 'M' commands), or breakpoints set within it are lost. The copy of DEBUG appended to the main should not be used to set breakpoints in the segments. Likewise, the copy of DEBUG appended to a segment should not be used to set breakpoints in the main or another segment.

DUMP MEMORY

D,A,n_1 (,n_2) ASCII dump of octal main memory address n_1 or from n_1 through n_2
D,B,n_1 (,n_2) Binary dump of octal main memory address n_1 or from n_1 through n_2

The second parameter indicates the format of the print-out: A specifies ASCII, B specifies octal. The address n_1 designates the location of the word or the first of a series of words that is to be dumped. If the second address, n_2, is greater than n_1, a block of memory, n_1 through n_2, is printed. If n_2 is the same as n_1, only one location is printed. All addresses are relative to the program relocation base. (See "M" command.)

The Dump output record format consists of the contents up to 8 consecutive words preceded by the address of the first word:

```
                addr.        word1        word2 . . .     word8
     Octal:     M+aaaaa      000000       000000 . . .    000000
     ASCII:     M+aaaaa        cc           cc              cc
```

The system BR command can be used to stop the listing.

PROGRAM RELOCATION BASE

M,n Sets absolute base of relocatable program unit at octal address n

The statement defines the program relocation base, n, as the absolute origin in memory of the user's relocatable program. This address may be obtained from the listing produced by the Relocating Loader during loading. If not specified, a value of zero is assumed. The value is added to all address parameters entered by the operator. It is subtracted from all addresses displayed by DEBUG.

Specification of this value allows subsequent reference in the control statements to addresses as shown on the program listing produced by the Assembler or the FORTRAN compiler. If this control statement is not used, program address parameters for other control statements must be absolute.

RUN

R(,n) Execute user program starting at octal address n or execute starting at next location in user program (used after a breakpoint or to initiate the program at the transfer point in the user program).

If the letter R only is entered, execution starts with the next sequential instruction in the user's program. To start at another location, the operator enters the address, n. The address n specified relative to the program relocation base (see the M command). The breakpoint message can be repeated by setting n equal to the location of the breakpoint.

SET MEMORY

S,n,d	Set octal value d in octal address n
S,n,d_1,d_2, . . . , d_n	Set octal values d_1 through d_n in successive memory locations beginning at octal address n

The above statement allows the user to set one or more values into locations defined by the first address, n. The value specified for d_1 is stored in location n; the value for d_2, in location $n + 1$; and so forth. To specify that an existing value in memory is to remain unchanged, two consecutive commas are used in the control statement. Any number of values may be entered via one control statement provided the length of the statement does not exceed 72 characters. The address n is relative to the program relocation base. (See "M" command.) If the address is outside of the program's area the message:

ADDR n ILLEGAL

is displayed and the store is not allowed.

SET REGISTER

W,A,d	Set A-register to octal value d
W,B,d	Set B-register to octal value d
W,E,d	Set E-register to octal value d (0 = off; non-zero = on)
W,O,d	Set Overflow to octal value d (0 = off; non-zero = on)

Since the Debugging routine simulates the register, the results of a Set Register operation are not reflected on the computer front panel.

CLEAR BREAKPOINT

X,n	Clear breakpoint at octal address n. The address is relative to the program relocation base. (See "M" command.)

INDEX I

This index lists the names and page reference of all the Relocatable Library entry points.

INDEX II

This index lists the subroutines in the DOS/RTE Relocatable Library by function. The functional categories are:

- Absolute Value
- Complex Number Arithmetic
- Conditional Branch
- DOS/RTE Utilities
- Exponents, Logs, and Roots
- General I/O
- Integer Arithmetic
- Miscellaneous
- Number Conversion
- Parameters, Formats, and Addresses
- Program Error and Termination
- Real Number Arithmetic
- Register Test
- Trigonometry
- Double Integer

CONDITIONAL BRANCH

Name	Function	Page
.GOTO	Transfer control to the location indicated by a FORTRAN computed GOTO statement: GOTO (K_1, K_2, . . .K_n)	3-62
.SWCH	Switch execution control to the Ith label in a sequence of N labels (implements ALGOL switch statement)	3-68

DOS/RTE UTILITIES

DBKPT	Process breakpoints for DEBUG	3-5
DEBUG	Provide debug aids for relocatable programs	3-6
SREAD	Read a source record or sector from a specified device	3-25
%WRIS	Write a disc source file (RTE only)	3-58
%WRIT	Write load-and-go file on disc	3-57

EXPONENTS, LOGS, AND ROOTS

ALOG	Ln (real x)	2-4
ALOGT	Log_{10} (real x)	2-5
CLOG	Ln (complex x)	2-13
CSQRT	Complex complex x	2-19
DEXP	Extended real e (extended real x)	2-27
DLOG	Ln (extended real x)	2-29
DLOGT	Log_{10} (extended real x)	2-30
DSQRT	Square root of x, where x is extended real value	2-34
CEXP	Complex e^x, where x is complex value	2-12
EXP	e^x, where x is real value	2-39
SQRT	Square root of x, where x is real value	2-58
.CTOI	x^I, where x is a complex value	2-84
.DTOD	x^y, where x and y are extended real values	2-91
.DTOI	x^I, where x is extended real and I is an integer	2-92
.DTOR	x^y, where x is extended and y is real value; result is extended real	2-93
.EXP	Calculate e^x where x is double real	2-94
.FPWR	Calculates x^I for real x	2-98
.ITOI	I^J, where I and J are integers	2-103
.LOG	Calculates $\log_e$ x for double real x	2-105
.LOGO	Calculates $\log_{10}$ x for double real x	2-106
.PWR2	$x.2^n$, where x is real and n is an integer value	2-114
.RTOD	x^y, where x is real value, y is a extended real value; result is extended real	2-115
.RTOI	x^I, where x is a real value and I is an integer	2-115
.RTOR	x^y, where x and y are real values	2-117
.RTOT	Calculate x^Y, where x is real and Y is double real	2-118
.SQRT	Calculate the square root of double real x	2-122
.TPWR	Calculates x^I, where x is double real and I is unsigned	2-130
.TTOI	Calculates X^I, where X is double real and I is an integer	2-132
.TTOR	Calculates X^Y, where X is double real and Y is real	2-133
.TTOT	Calculate x^Y, where x and Y are double reals	2-134
#EXP	Complex e^x, where x is complex value; no error return	3-27
#LOG	Ln (complex x); no error return	3-28
$EXP	Extended real e^x, where x is extended real value; no error return	3-30
$LOG	Ln (extended real x); no error return	3-31

EXPONENTS, LOGS, AND ROOTS (Continued)

Name	Function	Page
$LOGT	Log_{10} (extended real x); no error return	3-32
$SQRT	Square root of x, where x is a extended real value; no error return	3-34
%LOG	Ln (real x); call-by-name	3-46
%LOGT	Log_{10} (real x); call-by-name	3-47
%QRT	Square root of x, where x is a real value; call-by-name	3-52
%XP	e^x, where x is a real value; call-by-name	3-58
/EXP	.EXP with no error return	3-74
/EXTH	Compute 2^N x 2^Z for small double real z	3-75
/LOG	.LOG with error return	3-76
/LOG0	.LOG0 with no error return	3-77
/SQRT	.SQRT with no error return	3-79

GENERAL I/O

BINRY	Read or write on disc	X-XX
CLRIO	Compatibility routine	3-3
MAGTP	Perform utility functions on magnetic tape unit	3-15
PTAPE	Position magnetic tape	3-22
.TAPE	Rewind, back space, or end-of-file operation on magnetic tape unit	3-69

INTEGER ARITHMETIC

ISIGN	I • sign (z); transfer the sign of a real or integer z to an integer I	2-49
%SIGN	I • sign (z); transfer the sign of a real or integer z to an integer I; call-by-name	3-53

MISCELLANEOUS

DPOLY	Evaluate the quotient of two polynomials in double precision	2-63
IAND	Calculate the logical product of integers I and J	2-43
IOR	Calculate the logical inclusive or of integers I and J	2-48
IXOR	Calculate integer exclusive OR	2-50
MXMND	Calculate the maximum or minimum of a series of extended real values	2-52
MXMNI	Calculate the maximum or minimum of a series of integer values	2-53
MXMNR	Calculate the maximum or minimum of a series of real values	2-54
TRNL	See DPOLY	2-63
XPOLY	Evaluate the extended real polynomial: $C_1X^{n-1}+C_2X^{n-2}+ \ldots +C_{n-1}X+C_n$	2-68
.CFER	Move four words from address x to address y. (Complex transfer)	2-77
.CHEB	Evaluate chebyshev series	2-78
.FLUN	Unpack a real x; place exponent in A-register, lower mantissa in B-register	2-96
.MANT	Extract the mantissa of a real x	2-108
.XFER	Move three words from address x to address y (extended real transfer)	2-137
%AND	Calculate the logical product of integers I and J; call-by-name	3-38
%OR	Calculate the logical inclusive "or" of integers I and J; call-by-name	3-49
%OT	Complement integer I; call-by-name	3-51
.MXMN	Finds maximum of a list of double reals	2-111
.MXMN	Finds minimum of a list of double reals	2-111
.OPSY	Determine which disc operating system is in control	3-64
.4ZRO	Common double real zero	2-141
..TCM	Negate a double real	2-146
$SETP	Set up a list of pointers	3-33

NUMBER CONVERSON

Name	Function	Page
ACODE	Internal number conversion	4-32
AINT	Truncate a real x	2-3
AMOD	x modulo y, where x and y are real values	2-6
CODE	Internal number conversion	4-29
DBLE	Convert real x to extended real y	2-24
DDINT	Truncate an extended real x	2-26
DMOD	x modulo y, where x and y are extended real values	2-31
ENTIE	Calculate greatest integer I that is not greater than real x	2-37
ENTIE	Round a real x to the nearest integer I	2-37
FLOAT	Convert integer I to real x	2-41
FMTIO	Provides internal conversion according to a FORMAT from one memory area to another memory area	4-29
IDINT	Truncate an extended real x to an integer	2-45
IFIX	Convert a real x to an integer I	2-46
INT	Truncate a real x to an integer J	2-47
MOD	I modulo J, where I and J are integers	2-51
SNGM	Convert extended real x to real y without rounding	2-59
SNGL	Convert an extended real x to a real y	2-58
.BLE	Convert real to double real	2-73
.CMRS	Reduce argument for SIN, COS, TAN, EXP	2-81
.CTBL	Converts a complex real to double real	2-83
.CINT	Convert a complex x to an integer	2-79
.DCPX	Convert an extended real x to a complex y	2-85
.DINT	Convert an extended real x to an integer	2-87
.ICPX	Convert integer I to complex value	2-99
.IDBL	Convert integer I to extended real value	2-100
.ITBL	Converts integer to double real	2-102
.IENT	Calculate the greatest integer I that is not greater than real x	2-101
.NGL	Convert double real to real	2-112
.PACK	Convert signed mantissa of a real x into normalized real format	2-113
.TCPX	Convert double real to complex real	2-125
.TINT	Convert double real to integer	2-128
%FIX	Convert a real x to an integer I; call-by-name	3-41
%INT	Truncate a real x; call-by-name	3-44
%LOAT	Convert integer I to a real x; call-by-name	3-45
%NT	Truncate a real x to an integer J; call-by-name	3-48
/CMRT	Range reduction for .SIN, .COS, .TAN, .EXP, and .TAN	3-73
/TINT	Conversion of double precision to integer	3-81

PARAMETERS, FORMATS, AND ADDRESSES

GETAD	Determine the true address of a parameter passed to a subroutine and store address	3-9
INDEX	Determine address or value of an ALGOL array	3-9
IGET	Read the contents of a memory address	3-12
ISTAT	The Status word returned from the EXEC in the last I/O call the FORMATTER did	4-30
ITLOG	Number of characters read or written by last formatter I/O request	4-30
LGBUF	Can be used to specify address and length of I/O buffer	4-34
NAMR	Read input buffer, produce 10-word parameter buffer	3-16

PARAMETERS, FORMATS, AND ADDRESSES (Continued)

Name	Function	Page
RMPAR	Move five words into a designated array from address pointed to by B-register	3-23
RSFLG	Set the save-resource flag for RTE-BASIC	3-24
.DFER	Move three words from address y to x (extended real transfer)	2-86
.DIV	DOS-III only: replace subroutine call with hardware instruction to divide 2-word integer I by 1-word integer J	2-88
.DLD	DOS-III only: replace subroutine call with hardware instruction to load memory locations x+1 into A- and B-registers, respectively	2-89
.DST	DOS-III only: replace subroutine call with hardware instruction to store A- and B-register contents into address x and x+1, respectively	2-90
.ENTC	Transfers true addresses of parameters from a calling sequence into a subroutine; adjusts return addresses to the true return point	3-59
.ENTR	Transfer the true address of parameters from a calling sequence into a subroutine; adjust return addresses to the true return point	3-60
.LBT	Replaces 21MX microcoded instruction LBT	2-104
.MAC.	DOS-III only: replace subroutine call with hardware instruction to initiate firmware	2-107
.MAP.	Return actual address of a particular element of a two-dimensional FORTRAN array	3-63
.MPY	DOS-III only: replace subroutine call with hardware instruction to multiple integer I by integer J	2-110
.RCNG	Converts calls using .ENTR to .ENTC conventions	3-67
.PCAD	Return the true address of a parameter passed to a subroutine	3-65
.PRAM	Process parameter values and/or addresses passed to Assembly language subroutines by ALGOL programs	3-66
.SBT	Replaces 21MX microcoded instruction SBT	2-119
..MAP	Compute the address of a specified element of a 2 or 3 dimensional array	3-70

PROGRAM TERMINATION AND ERROR

Name	Function	Page
ERR0	Print a 4 character error code on the list device	3-8
IND.E	Select output LU for error messages	3-11
ISTAT	Status word returned from EXEC in last I/O call done by formatter	4-30
PAUSE	Halt program execution and print message	3-20
PAU.E	Select output LU for PAUSE messages	3-19

REAL NUMBER ARITHMETIC

Name	Function	Page
DSIGN	x • sign (y); transfer the sign of a extended real y to a extended real x	2-32
ENTIX	Calculate greatest integer that is not greater than a extended real x; result that is extended real	2-38
FADSB	x + y, where x and y are real	2-40
FADSB	x − y, where x and y are real	2-40
SIGN	x • sign (z); transfer the sign of a real or integer z to a real x	2-56
XADD	FORTRAN II Interface to .XADD	2-64
XADSB	Handles floating point addition and subtraction in extended precision	2-65
XDIV	FORTRAN II Interface to .XDIV	2-66
XMPY	FORTRAN II Interface to .XMPY	2-67
XSUB	FORTRAN II Interface to XADSB	2-69
.FDV	Divide real x by real y	2-95
.FMP	Multiply real x by real y	2-97
.MOD	Calculates double real remainder of x/y	2-109
.SIGN	Transfer the sign of a double real y to a double real x	2-120
.TADD	Double real add	2-129

REAL NUMBER ARITHMETIC (Continued)

Name	Function	Page
.TINT	Convert double real to integer	2-128
.TSUB	Double real subtract	2-129
.TMPY	Double real multiply	2-129
.TDIV	Double real divide	2-129
.YINT	Truncate fractional part of double real	2-140
.XCOM	Complement an extended real unpacked mantissa in place	2-135
.XDIV	Divide extended real x by extended real y	2-136
.XMPY	Multiply extended real x by extended real y	2-138
.XPAK	Normalize, round, and pack with the exponent an extended real mantissa	2-139
..DCM	Complement an extended real x	2-143
..DLC	Load and complement a real x	2-144
..FCM	Complement a real x	2-145
%IGN	x • sign (z); transfer the sign of a real or integer z to a real	3-42
/ATLG	Compute (1 −x)/(1 +x) in double precision	3-71

REGISTER TEST

ISSR	Set S-register to value N	3-13
ISSW	Set sign bit of A-register according to bit n of Switch Register	3-14
OVF	Set sign bit of A-register according to overflow bit	3-18
%SSW	Set sign bit of A-register according to bit n of Switch Register; call-by-name	3-54

TRIGONOMETRY

ATAN	Arctangent (real x)	2-7
ATAN2	Arctangent (real x/real y)	2-8
COS	COS (real x)	2-17
CSNCS	Complex sin (complex x); Complex cos (Complex x)	2-18
DATAN	Arctangent (extended real x)	2-22
DATN2	Arctangent (extended real x/double real y)	2-23
DCOS	Cos (extended real x)	2-25
DSIN	Sin (extended real x)	2-33
DTAN	Calculate tangent of extended real x	2-35
DTANH	Calculate hyperbolic tangent of real x	2-36
SIN	Sin (real x)	2-57
TAN	Tan (real x)	2-61
TANH	Tanh (real x); hyperbolic tangent	2-62
.ATAN	Calculate the arctangent of a double real	2-71
.ATN2	Calculates arctangent of double real quotient x/y	2-72
.TAN	Calculates tangent of double real x (radians)	2-123
.TANH	Calculates hyperbolic tangent of double real x	2-124
.TSCS	Calculates cosine of double precision Z	2-131
.TSCS	Calculates sine of double precision Z	2-131
#COS	Complex cos (complex x); no error return	3-26
#SIN	Complex sin (complex x); no error return	3-27
$TAN	DTAN with no error return	3-36

TRIGONOMETRY (Continued)

Name	Function	Page
%AN	Tan (real x); call-by-name	3-37
%ANH	Tanh (real x); call-by-name	3-39
%IN	Sin (real x); call-by-name	3-43
%OS	Cos (real x); call-by-name	3-50
%TAN	Arctangent (real x); call-by-name	3-55
/COS	.COS with no error return	3-72
/SIN	.SIN with no error return	3-78
/TAN	.TAN with no error return	3-80

DOUBLE INTEGER

FIXDR	Convert real to double-length record number	2-148
FLTDR	Convert double-length record number to real	2-149
.DADS	Double integer add and subtract	2-150
.DCO	Compare two double integers	2-151
.DDE	Decrement double integer in A & B registers	2-152
.DDI	Double integer divide; Z=X/Y	2-153
.DDS	Double integer decrement and skip if zero	2-154
.DIN	Increment double integer in A & B registers	2-155
.DIS	Double integer increment and skip if zero	2-156
.DMP	Double integer multiply; Z=X*Y	2-157
.DNG	Negate double integer x; Z=−x	2-158
.FIXD	Convert real to double integer	2-159
.FLTD	Convert double integer to real	2-160
.TFTD	Convert double integer to double real	2-161
.XFTD	Convert double integer to extended real	2-163
.XFXD	Convert entended real to double integer	2-164
.TFXD	Convert double real to double integer	2-162

HEWLETT
PACKARD

MANUAL PART NO. 24998-90001
Printed in U.S.A. October 1981

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014