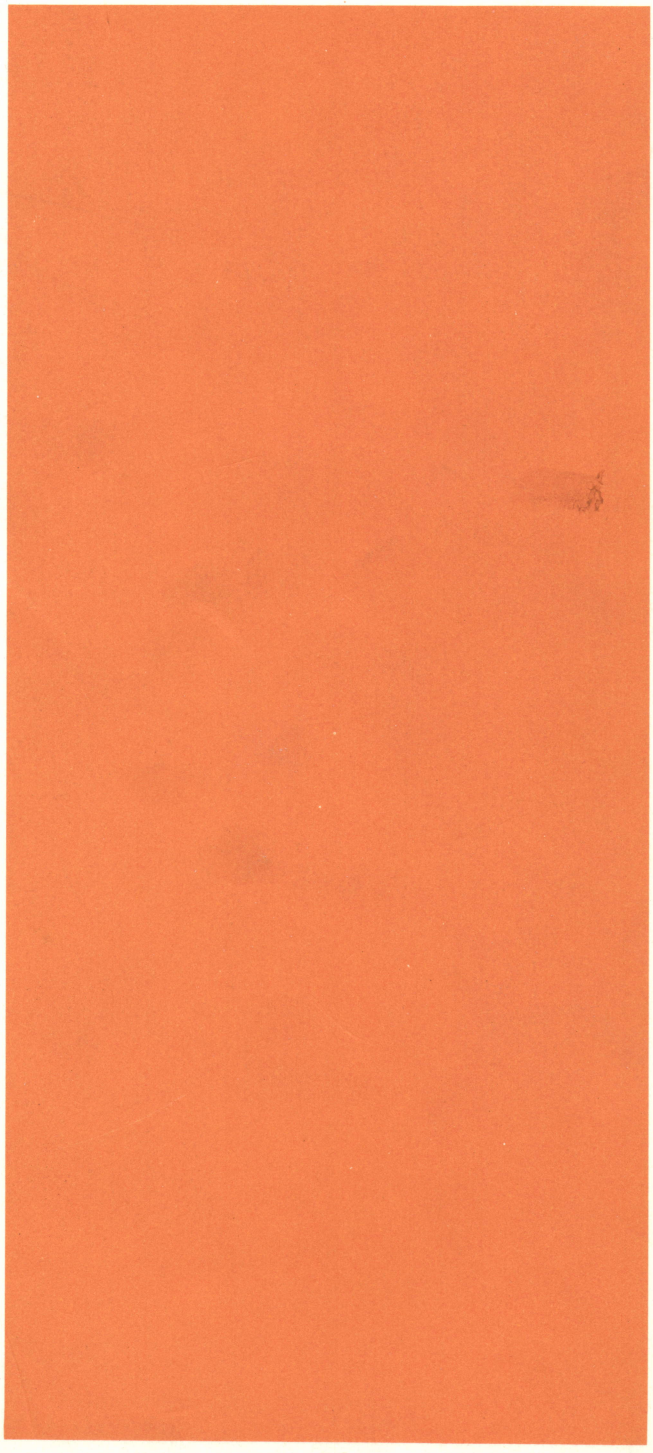
Honeywell

FORTRAN MATH LIBRARY

SYSTEM 700

OS/700

SOFTWARE



Honeywell

FORTRAN MATH LIBRARY

SYSTEM 700

OS/700

SUBJECT:

Conventions, Loading Information, Library Use, Programming Information, Description of Intrinsic and External Functions and Subroutines and of Compiler Support Subroutines, and Error Messages.

DATE:

May 1972

ORDER NUMBER:

AG16, Rev. 0

DOCUMENT NUMBER:

70130072602A

PREFACE

The Fortran Math Library consists of Fortran-callable subroutines. Section I introduces the library. Section II contains information for a programmer using the various subroutines. Section III shows how to call them from a DAP/700 program and gives examples. Section IV describes the standard ANSI and ISA Fortran subroutines in the library, and Section V describes the compiler support subroutines. There are five appendices, included to facilitate access to the library.

Additional information may be obtained from the following manuals:

DAP/700 Macro Assembler, Order Number AG17.

OS/700 Linkage Editor, Order Number AG08.

System 700 Programmers' Reference Manual, Order Number AC72.

Fortran 700, Order Number AH99. (This manual contains a description of the run-time library.)

The Fortran Math Library consists of coded programs designed to extend the power of System 700 in the area of program preparation and maintenance. They are supported by comprehensive documentation and training; periodic program maintenance and, where feasible, improvements are furnished for the current version of the programs, provided they are not modified by the user.

File No.: 1923

CONTENTS

ILLUSTRATIONS

SECTION I
INTRODUCTION

The Fortran Math Library consists of an extensive assortment of subroutines to aid the programmer in performing mathematical and trigonometric operations and functions, conversions between data types, bit string operations, logical relations, and other functions. The math routines included are for single-(real) and double-precision, complex, integer, and logical calculations.

This library may be loaded in either normal or extended mode and will run in the same mode.

SUBROUTINE DESCRIPTIONS

The descriptions, in Sections IV and V, of the Fortran external and intrinsic functions and the compiler support subroutines give the name of the subroutine, its purpose, the DAP/700 calling sequence, the Fortran calling sequence (where appropriate), the method used to compute the result, the data types of the arguments and the result (where applicable), error messages generated by the subroutine, if any, and other routines used by the subroutines, if any.[1]

APPENDICES

There are five appendices to this manual:

- APPENDIX A lists the contents of the library tapes. There are three paper tapes and one magnetic tape. The first tape listed is the source magnetic tape, containing the routines on paper tape 1 and the two versions of paper tape 2, in source rather than in object form. Paper tape 1, FTNLB1, contains only those subroutines which use complex and/or double-precision variables. The first version of paper tape 2, contains the remaining subroutines for systems equipped with the High-Speed Arithmetic Option. The second version of paper tape 2, FTNLB2S, contains the subroutines for those systems without the hardware option.
- APPENDIX B lists the math routines by argument type.
- APPENDIX C lists the library subroutines by function.

[1] The list of "Other Routines Used" is given in the order in which they are called. If a routine is called more than once, it is listed only once, the first time it is called.

- APPENDIX D is an alphabetical list of all the subroutines with their entry points, approximate storage required, subroutines referenced and the number of times referenced, the library tape on which they are located, and the page in this manual on which they are described.

- APPENDIX E lists the error messages produced by the subroutines and the interpretation of these messages.

SYMBOLS

The following symbols and letters are used in many of the subroutine descriptions:

*	multiplication
/	division
**n	raised to the exponential power of n
C	complex
D	double-precision
I	integer
L	logical
R	real

NAMING CONVENTIONS

The intrinsic and external functions are named according to the American National Standards Institute (ANSI) or the Instrument Society of America (ISA) naming rules.

The compiler support subroutines are named, for the most part, according to the following naming convention: The first letter of the name denotes the operation to be performed (see the list below). It is followed by a dollar sign having no significance and then by two numbers. The first number (see the list below) represents the operand initially in the accumulator (except in load operations) and the second number represents the second operand or the type of result. If there is a High-Speed Arithmetic Option version of these subroutines, an X is appended to the name.

Operation	Argument Type
A - Add	1 - Integer
C - Convert	2 - Real
D - Divide	3 - Logical
E - Exponential	5 - Complex
H - Hold (store)	6 - Double-precision
L - Load	8 - Double-precision exponent
M - Multiply	
N - Negate	
S - Subtract	
Z - Zero (clear)	

Examples

A$22 - add two real numbers

D$52 - divide a complex number by a real number

E$61 - calculate the value of a double-precision number to an integer power

M$22X - multiply two real numbers, using the High-Speed Arithmetic Option

LOADING INFORMATION

There are two sets of library subroutines, one for installation with the High-Speed Arithmetic Option and one for those systems without this hardware option. Each set is contained on two reels of paper tape. Customers who purchase the library in source form (on magnetic tape) receive both sets of library subroutines.

The organization of the library is modular, thus making it possible to load only those routines which will be used. This concept of modularity extends to the paper tape. If complex or double-precision variables are not used, only the second of the two paper tapes is needed.

Each reel has been assembled via DAP/700 and should be loaded by use of the Linkage Editor program. Reel one (FTNLB1) must be loaded before reel two (either FTNLB2S or FTNLB2H).

SECTION II

USE OF FORTRAN MATH LIBRARY

DATA TYPES AND REPRESENTATIONS

The representation of a negative number in any of the following formats (excluding logical) is the TWOs complement of the equivalent positive number. The complement is taken for the entire representation, including all subfields. The TWOs complement is taken by reversing all bits in the representation (ONEs complement) and adding one to the low-order position, propagating carries as required.

Integer

This is a 16-bit (single-precision only) word with an implied decimal point after bit 16; bit 1 is a sign bit (see Figure 2-1). An integer value may range from -32,768 to +32,767.

Example: +5 = 0 000 000 000 000 101 = '000005[1]
-5 = 1 111 111 111 111 011 = '177773

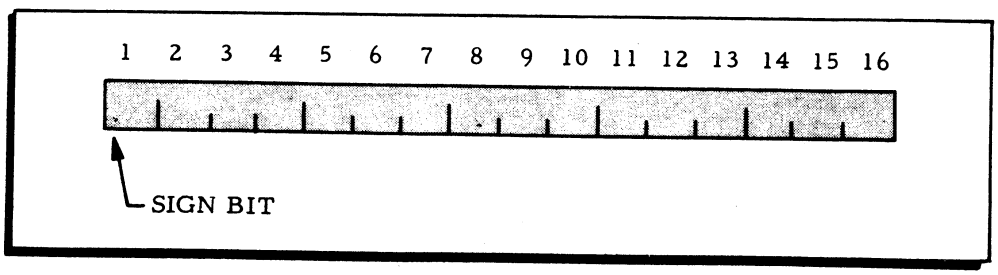


Figure 2-1. Format of Integer

Real

This is a 32-bit word in the format shown in Figure 2.2. Bit 1 is the sign bit (0 for positive, 1 for negative). Bits 2-9 contain a binary number (N) with a maximum decimal value of 255 (377 octal) representing the 8-bit characteristic. This number is "biased" by 128 (200 octal). The remaining 23 bits represent a binary fraction (F) with a value less than 1. The value represented is F*2**(N-128). A number is considered "normalized" when the fraction F is at least 1/2 (i.e., the leading bit is set for a positive number). Within this representation the largest representable number in normalized form is just under 2**127, or approximately 10**(38.5). The smallest number is 2**(-129), or approximately 10**(-38.5). The 23 magnitude bits give a precision of one part in 2**23, or approximately 6.9 digits of accuracy. Zero

[1] The apostrophe before a number indicates octal code.

is shown by all zeros in these 23 bits. (Throughout this manual the word "real" is used to reference real single-precision numbers.)

Example: +5. = 0 100 000 111 010 000, 0 000 000 000 000 000 = '040720, 0
-5. = 1 011 111 000 110 000, 0 000 000 000 000 000 = '137060, 0

Double-Precision

This three-word format is identical to the real number format with the exception of an additional 16 magnitude bits (see Figure 2-2). The 39 magnitude bits give a precision of one part in 2**39, or approximately 11.7 digits of accuracy. This data type should not be confused with hardware double-precision.

Complex

This is represented by two real number pairs, each having the format of a real number (see Figure 2-2). A real number takes two words of storage; the complex format requires four words. The first two words represent the real portion of the complex number, and the last two words represent the imaginary portion.

Logical

A logical value is shown as a word of all zeros for false and a value of one for true. In logical operations, any nonzero value is interpreted as true. The complement of a logical value changes it from 0 to 1 or 1 to 0.

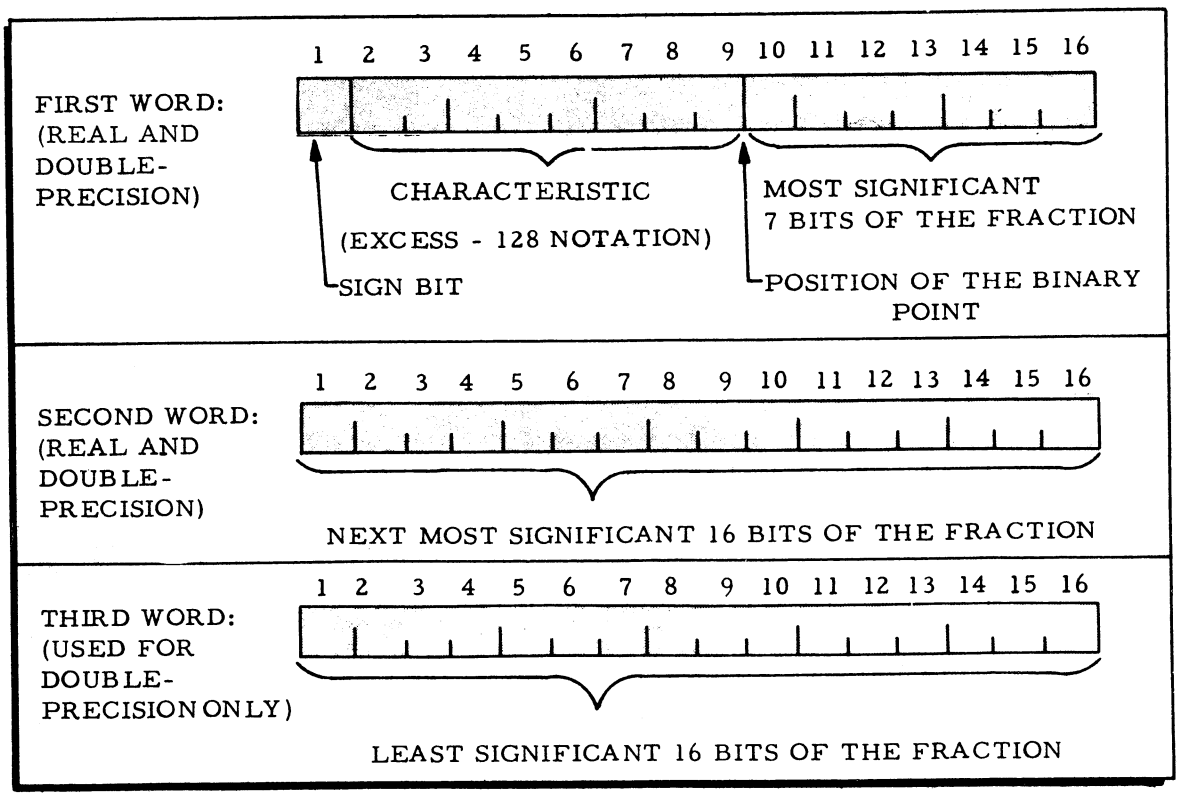


Figure 2-2. Format of Real and Double-Precision Numbers

NORMALIZATION

A real, double-precision, or complex number is defined as normalized when the fractional part has a value between 1/2 and 1. For instance, $3/8 \times 2^3$ and $3/4 \times 2^2$ both have the same value, but the latter is the normalized form.

REGISTER USE

All registers are presumed to be available to the subroutine library. and the user is cautioned not to expect any of them to be preserved, whether or not the arguments or results are stored in them. That is, any registers not specifically described as containing a particular result upon exit from the subroutine must be considered as having become undefined by the execution of the subroutine.

ACCUMULATORS

Integer Accumulator

The A-register is used in all integer operations.

Real Accumulator

The A- and B-registers are used in all real operations.

Complex (pseudo) Accumulator

This four-word area in memory (AC1-AC4) is provided by the library to be used in all complex operations. The real portion of the complex number is stored in locations AC1 and AC2; the imaginary portion is stored in locations AC3 and AC4.

Double-Precision (pseudo) Accumulator

This three-word area in memory (AC1-AC3) is provided by the library to be used in all double-precision operations.

RESULTS

Results are stored according to their data types. Complex numbers are found in the complex accumulator upon exit from any of the compiler support subroutines; double-precision numbers are found in the double-precision accumulator; real numbers are found in the A- and B-registers; and integer and logical values are found in the A-register.

SECTION III
DAP/700 PROGRAMMING INFORMATION

LIBRARY CALLS FROM DAP/700

The DAP/700 calling sequences for entry into the subroutines are shown in the descriptions in Sections IV and V. When the Fortran compiler encounters either a function reference or a call to a subroutine, the following steps are performed:

1. A call to the function or subroutine is generated.
2. The address of each argument is determined and saved, in the order in which it is retrieved. In the case of expressions, this address is the location containing the current value of the expression.
3. If there is more then one argument, the final address is followed by a word of zeros to serve as an argument list terminator.

The code generated by a subprogram definition written in Fortran includes a call to the special subroutine F$AT (Argument Transfer, found in the Run-Time Library). This call immediately follows the entry point and in turn is followed by a word containing a count of the number of arguments as defined in the definition statement, followed by that number of words. The F$AT subroutine fills in those words with the argument addresses (from the call to the subprogram) and sets the return to the word following the argument terminator word (zeros). All levels of indirect addressing are removed in passing these addresses. In the case of a single argument, the terminator word is eliminated, the argument to F$AT shows a single argument, and the search for the terminator is not performed.

Null arguments may be included in a calling sequence by use of DAC* 0 as the address in the call. Subroutines serviced by F$AT find the address DAC*0 placed in the list of addresses and therefore know that the parameter was null. It is equally effective to use a DAC*PTR, where PTR is a DAC*0. This permits a dummy argument to be null, i.e., an argument passed through an intermediate subroutine call.

The DAP/700 programmer can generate his own code, performing the same functions as the F$AT subroutine.

Some of the Fortran Math Library subroutines have additional arguments in the A- and B-registers, or the C-register, or the pseudo-accumulators AC1-AC4. When this is the case, the description references an "implicit" argument, i.e., one whose address is not explicitly part of the calling sequence.

The compiler support subroutines are those which are not normally explicitly called by the Fortran programmer. For example, the statement

```
Z = X + Y
```

produces the following DAP/700 code:

```
CALL    L$22    }
DAC     X       }  loads the value of X in the A- and B-registers
CALL    A$22    }
DAC     Y       }  adds the value of Y to the A- and B-registers
CALL    H$22    }
DAC     Z       }  stores the result in the A- and B-registers in
                   location Z
```

The three subroutines, L$22, A$22, and H$22, are compiler support subroutines. They may be called explicitly by the Fortran programmer, if desired, as follows:

```
CALL    L$22(X)
CALL    A$22(Y)
CALL    H$22(Z)
```

This group of instructions performs the same function as the statement Z = X + Y and generates the same DAP/700 code.

Any of the compiler support subroutines may be called by the Fortran programmer in the following manner:

CALL ROUTINE NAME (ARG1)

CALL ROUTINE NAME (ARG1, ARG2)

CALL ROUTINE NAME (ARG1, ARG2, ... ARGn)

EXAMPLES OF DAP/700 CALLS TO LIBRARY

```
CALL    M$55
DAC     ARG1
Return
```

This call enters the complex multiplication subroutine, multiplying the contents of the complex pseudo-accumulator by the complex value in locations ARG1-ARG1+3 in the standard format for complex numbers. The result is stored in the complex accumulator (AC1-AC4), and any of the other registers should be presumed to have become undefined.

```
CALL    AMIN0
DAC     I
DAC     J
DAC     K
OCT     0
Return
```

This subroutine compares the three integer arguments I, J, and K (no implicit arguments) and returns with the value of the smallest of these, converted to data type real, in the A- and B-registers. Other registers are now presumed to be undefined.

SECTION IV
INTRINSIC AND EXTERNAL FUNCTIONS AND SUBROUTINES

This section describes the mathematical and trigonometric functions, bit string operations, and other special Fortran subroutines, arranged in alphabetic order by subroutine name.

ABS

Purpose	To generate the absolute value of a real number.
DAP Calling Sequence	CALL ABS DAC ARG1 (a real number) (Return)
Fortran Reference	ABS(R)
Method	This subroutine checks the real agrument, ARG1, for its algebraic sign. If the sign is negative, the TWOs complement of ARG1 is calculated. If the sign is positive, the number remains unchanged.
Data Type of Arguments and Results	This absolute value function of a real number gives a real result.
Other Routines Used	L$22, N$22

Purpose	To obtain the imaginary part of a complex argument and convert it to real format.
DAP Calling Sequence	CALL AIMAG DAC ARG1 (a complex number) (Return)
Fortran Reference	AIMAG(C)
Method	The complex argument, ARG1, is placed in the complex accumulator. The imaginary part of the complex number (AC3 and AC4) is then loaded into the A- and B-registers.
Data Type of Arguments and Results	The imaginary part of the complex argument, ARG1, is converted to a real number and placed in the A- and B-registers.
Other Routines Used	L$55, L$22, AC3

AINT

Purpose

To truncate the fractional bits of a real number.

DAP Calling Sequence

```
CALL   AINT
DAC    ARG1  (a real number)
  (Return)
```

Fortran Reference

AINT(R)

Method

A constant (2**22) is successively added and subtracted from ARG1. The available precision of real numbers is such that the fractional part of this result is lost. If ARG1 is negative, its TWOs complement is taken before the addition and subtraction take place and it is recomplemented before the subroutine exits. The resultant value is effectively the largest integer ≤ |ARG1| with the sign of ARG1.

Data Type of Arguments and Results

The real argument remains a real number.

Other Routines Used

L$22, N$22, A$22, S$22

Purpose

To calculate the natural (base e) or common (base 10) logarithm of a real number.

DAP Calling Sequence

```
CALL   ALOG   (or ALOG10)
DAC    ARG1   (a real number)
  (Return)
```

Fortran Reference

ALOG(R) or ALOG10(R)

Method

The $\log_2$ of the argument, ARG1, is computed. This value is then converted to the desired base by multiplication by an appropriate constant.

$$\log_2 ARG1 = F^1*(C1+T(C3+T(C5+T(C7+T(C9)))))+B-.5,$$

where $T = F^1*F^1$ and

C1 = .28853901E1
C3 = .96179665E0
C5 = .57708664E0
C7 = .41153510E0
C9 = .34280712E0

$$F^1 = \frac{F-\frac{\sqrt{2}}{2}}{F+\frac{\sqrt{2}}{2}}$$

F is the fractional part of the normalized argument and B is the binary exponent of the original argument which has been converted to a real number.

Data Type of Arguments and Results

The argument and the results are both real numbers.

Error Messages

The message "LG" is reported if a negative or zero-valued argument is used, and the result is undefined.

Other Routines Used

ARG$, C$12, A$22, M$22, S$22, F$ER

ALOGX

Purpose

To calculate the natural (base e) or common (base 10) logarithm of a real number.

DAP Calling Sequence

```
CALL   ALOGX   (or ALOG or ALOG10)
DAC    ARG1    (a real number)
 (Return)
```

Fortran Reference

ALOG(R) or ALOG10(R)

Method

$\log_A Z = (\log_2 Z)*(\log_A 2)$, where Z=ARG1. Thus for the natural logarithm,

$\ln Z = (\log_2 Z)*(\log_2 2)$; for the common logarithm,

$\log_{10} Z = (\log_2 Z)*(\log_{10} 2)$. The calculation simplifies in both cases to a computation of $\log_2 Z$. Remembering that the floating-point number Z can be expressed as Z = F*2**B, where F is the fractional part and B the binary exponent of the normalized argument Z,

$\log_2 Z = (\ln(F)/\ln(2))+B$.

Now, let F = F*K/K, where K may be

$$\prod_{i=1}^{t} K_i$$

such that F*K = 1+G

$$\text{LOG}_2 X = \frac{\ln F*K/K)}{\ln(2)} + B$$

$$= \frac{\ln(F*K - \ln(K)}{\ln(2)} + B \quad \text{then } \ln(K) \text{ is } \sum_{i=1}^{t} \ln(K_i)$$

$$= \frac{\ln(F*K)}{\ln(2)} - \frac{\ln(K)}{\ln(2)} + B$$

$$= \frac{\ln(1+G)}{\ln(2)} - \frac{\ln(K)}{\ln(2)} + B$$

$$= \frac{G - 1/2G^2 + 1/3G^3 -}{\ln(2)} \quad \cdots\cdots \quad \frac{\ln(K)}{\ln(2)} + B$$

Since ln(2) = .69314718,

$$\log_2 X = 1.442695141\ G - .7213475704\ G^2 + .4808984995\ G^3 - \frac{\ln(K)}{\ln(2)} + B$$

Data Type of Arguments and Results	This function with a real argument results in a real number.
Error Messages	The message "LG" is reported if a negative or zero-valued argument is used, and an undefined result is returned.
Other Routines Used	ARG$, C$12, A$22, M$22, S$22, F$ER

ALOG10

Purpose To calculate the common (base 10) logarithm.

See ALOG, p. 4-5.

Purpose — To find the maximum real value in a list of integers.

See MAX0, p. 4-61.

AMAX1

Purpose To find the maximum real value in a list of real arguments.

See MAX1, p. 4-62.

Purpose To find the minimum real value in a list of integers.

See MIN0, p. 4-63.

AMIN1

Purpose — To find the minimum real value in a list of real arguments.

See MIN1, p. 4-64.

Purpose

To compute the remainder resulting from the division of two real numbers.

DAP Calling Sequence

```
CALL   AMOD
DAC    ARG1     (real dividend)
DAC    ARG2     (real divisor)
OCT    0        (end of arguments flag)
 (Return)
```

Fortran Reference

AMOD(R, R)

Method

This subroutine divides ARG1 by ARG2 by calling D$22. The function AMOD (ARG1, ARG2) is defined as:

A1 - (A1/A2) * A2, where A1=ARG1 and A2=ARG2

(A1/A2) is the integer whose magnitude does not exceed the magnitude of A1/A2 and whose sign is the same as that of A1/A2.

Data Type of Arguments and Results

This function with two real arguments results in a real number for a remainder.

Other Routines Used

L$22, D$22, AINT, M$22, N$22, A$22

ATAN

Purpose

To calculate the principal value of the arctangent (i.e., 1st or 4th quadrant angle) of a real number or to compute and adjust for quadrant the arctangent of a real number expressed as a ratio (X/Y).

DAP Calling Sequence

```
CALL   ATAN                    or CALL  ATAN2
DAC    ARG1 (a real number)       DAC   ARG1  (both arguments
 (Return)                         DAC   ARG2   are real numbers)
                                  OCT   0  (end of arguments flag)
                                 (Return)
```

Fortran Reference

ATAN(R) or ATAN2(R, R)

Method

For ATAN, let N = ABS(ARG1). The arctangent of N is evaluated by dividing the total range $0 \leq N \leq 10^{**}75$ into three intervals:

If $N \leq 10^{**}(-B)$, ATAN(N) = N

If $N > 10^{**}10$, ATAN(N) = pi/2

If $10^{**}(-8) < N \leq 10^{**}10$,

ATAN(N) = base angle + P(Z)

= base angle + C1*Z+ C3*Z**3+C5*Z**5...

If $N \leq 1/2$, Z = N and base angle = 0

If $N \leq 2$, Z = (N-1)/(N+1) and base angle = pi/4

If $N < 2$, Z = (-1/N) and base angle = pi/2

For ATAN2, the arctangent of the quotient of ARG1/ARG2 (ARG1 = side opposite, ARG2 = side adjacent, or sin/cos) is computed as in ATAN and adjusted for quadrant by examination of the signs of the numerator and denominator.

Results

Quadrant	ARG1	ARG2	Quotient	Results (radians)
1	+	+	0 to ∞	0 to pi/2
2	+	-	- to 0	pi/2 to pi
3	-	-	0 to ∞	-pi to -pi/2
4	-	+	- to 0	-pi/2 to 0

Data Type of Arguments and Results	This arctangent function of a real number results in a real number.
Other Routines Used	ARG$, D$22, N$22, M$22, A$22, S$22

ATAN2

See ATAN, p. 4-14.

MIN1

Purpose

To find the smallest value in a list of real arguments and exit with this value (AMIN1) or convert it to an integer (MIN1) and exit.

DAP Calling Sequence

```
CALL    MIN1    (or AMIN1)
DAC     ARG1    (a real number)
DAC     ARG2    (A real number)
 .
 .
DAC     ARGn    (last real argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

MIN1(R, R, . . . , R_n) or AMIN1(R, R, . . . , R_n)

Method

Compare the arguments and retain the smallest value.

Data Type of Arguments and Results

The arguments are real numbers for either call (MIN1 or AMIN1). The result is real if AMIN1 is called; the result is integer if MIN1 is called.

Other Routines Used

L$22, H$22, S$22, IFIX

Purpose

To find the smallest value in a given set of integers and exit with this value or convert this value to a real number and exit.

DAP Calling Sequence

```
CALL    MIN0    (or AMIN0)
DAC     ARG1    (an integer value)
DAC     ARG2    (an integer value)
  .
  .
DAC     ARGn    (last integer argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

MIN0(I, I, . . . , I_n) or AMIN0(I, I, . . . , I_n)

Method

This subroutine compares the arguments and retains the smallest value. If AMIN0 is called, the result is converted to a real number before the subroutine exits.

Data Types of Arguments and Results

The arguments are integers in either call (MIN0 or AMIN0). The result is integer if MIN0 is called; the result is a real number if AMIN0 is called.

Other Routines Used

FLOAT

MAX1

Purpose

To find the largest value in a list of real arguments and exit with this value or convert it to an integer (MAX1) and exit.

DAP Calling Sequence

```
CALL    MAX1    (or AMAX1)
DAC     ARG1    (a real number)
DAC     ARG2    (a real number)
  .
  .
DAC     ARGn    (last real argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

MAX1(R, R, . . .) or AMAX1(R, R, . . .)

Method

This subroutine compares the arguments and retains the largest value. If MAX1 is called, the result is converted to integer by calling IFIX before the subroutine exits.

Data Type of Arguments and Results

The arguments are real numbers in either call (AMAX1 or MAX1). The result is real if AMAX1 is called; the result is an integer if MAX1 is called.

Other Routines Used

L$22, H$22, S$22, IFIX

Purpose

To find the largest value in a list of integer arguments and exit with this value or convert it to real format (AMAX0) and exit.

DAP Calling Sequence

```
CALL    MAX0    (or AMAX0)
DAC     ARG1    (integer value)
DAC     ARG2    (integer value)
 .
 .
DAC     ARGn    (last integer argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

MAX0(I,I,...) or AMAX0(I,I,...)

Method

This subroutine compares the arguments and retains the largest value. If AMAX0 is called, the result is converted to real by calling FLOAT before the subroutine exits.

Data Type of Arguments and Results

The arguments are integers in either call (MAX0 or AMAX0). The result is integer if MAX0 is called; the result is a real number if AMAX0 is called.

Other Routines Used

FLOAT

LOC

Purpose

To determine the address of the argument.

DAP Calling Sequence

```
CALL   LOC
DAC    ARG1
  (Return)
```

Fortran Reference

LOC(ARG1)

Method

Fetch the argument address (direct or indirect) and load it into the A-register.

Purpose

To EXCLUSIVELY OR two integers

DAP Calling Sequence

```
CALL    IXOR    (or CALL IEOR)
DAC     ARG1    (an integer value)
DAC     ARG2    (an integer value)
OCT     0       (end of arguments flag)
  (Return)
```

Fortran Reference

IXOR(I, I) or IEOR(I, I)

Method

An EXCLUSIVE OR (ERA) of ARG2 (which is placed in the A-register) and ARG1 is performed.

Data Type of Arguments and Results

This function uses two integer arguments and gives an integer result.

ITEST

Purpose

To test the status of a specified bit.

DAP Calling Sequence

```
CALL   ITEST
DAC    ARG1    (memory location to be tested)
OCT    N       (bit of the word to be tested, 0-15)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

CALL ITEST(I, J)

Method

This subroutine loads a value of 1 into the A-register, shifts it the number of places specified by the value N, and ANDs the A-register with ARG1 to determine whether the specified bit is set. A value of 0 is returned in the A-register if the bit is not set; a 1 is returned if the bit is set.

Other Routines Used

F$AT, ISHFT

Purpose

To store the contents of the second argument in the location specified as the first argument.

DAP Calling Sequence

```
CALL   ISTORE
DAC    ARG1    (target word address)
DAC    ARG2    (word to be stored)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

ISTORE(ARG1, ARG2)

Method

Fetch the target word address (ARG1) and save it.

Fetch the word to be stored (ARG2) and use it to replace the contents of the target location. Effectively, the contents of ARG2 are stored in location ARG1.

Other Routines Used

F$AT

ISIGN

Purpose

To generate a value consisting of the sign of the second integer argument and the magnitude of the first integer argument.

DAP Calling Sequence

```
CALL   ISIGN
DAC    ARG1    (an integer value)
DAC    ARG2    (an integer value)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

ISIGN(I, I)

Method

ARG2 is tested for its algebraic sign and, depending on the sign of ARG1, the procedure is as follows:

ARG1	ARG2	Result
+	+	+ |ARG1|
+	-	- |ARG1|
-	+	+ |ARG1|
-	-	- |ARG1|

Data Type of Arguments and Results

Both arguments and the result are integers.

Purpose

To shift an integer N places to the right or left in the A-register.

DAP Calling Sequence

```
CALL   ISHFT
DAC    ARG1    (the integer to be shifted)
DAC    ARG2    (the number of places to shift the integer)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

ISHFT (I, ±N)

Method

This routine loads the value of ARG1 into the A-register and shifts it the number of places specified by ARG2 (maximum value =± 16). If ARG2 is negative, the number is shifted to the right; if ARG2 is positive, the number is shifted to the left.

Data Type of Arguments and Results

The integer argument is shifted and return is made with the shifted integer in the A-register.

ISET

Purpose

To set a specified bit in the word specified by the first argument.

DAP Calling Sequence

```
CALL   ISET
DAC    ARG1    (memory location of word)
OCT    N       (bit of word to be set)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

CALL ISET(I, J)

Method

This subroutine loads a value of 1 into the A-register, shifts it the number of places specified by the value N, and save this value in a temporary location. An EXCLUSIVE OR of the A-register with ARG1, an AND with the temporary storage word, and another EXCLUSIVE OR are performed with ARG1. This value is stored in ARG1 and the subroutine exits.

Other Routines Used

F$AT, ISHFT

Purpose

To logically OR two integers.

DAP Calling Sequence

```
CALL   IOR
DAC    ARG1    (An integer value)
DAC    ARG2    (an integer value)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

IOR(I, I)

Method

The complement of the second argument is ANDed with the first argument, and this value is EXCLUSIVELY ORed with the first argument. The result is found in the A-register.

Data Type of Arguments and Results

This routine uses two integer arguments and gives an integer result.

INT

Purpose

To truncate the fractional bits from a real argument, thus converting it to integer format.

See IFIX, p. 4-51.

Purpose

To truncate the fractional bits from a real or double-precision argument, thus converting it to integer format.

DAP Calling Sequence

```
CALL   IFIX    (or CALL INT)
DAC    ARG1    (a real number)
 (Return)

 or

CALL   IDINT
DAC    ARG1    (a double-precision number)
 (Return)
```

Fortran Reference

IFIX(R), INT(R), IDINT(D)

Method

This subroutine truncates the fractional bits of ARG1, shifts it to the right until the binary point is at the end of the register, and normalizes the result. It then scales the value to an integer using the characteristic.

Data Type of Arguments and Results

If either IFIX or INT is called, the argument is a real number and the result is an integer. If IDINT is called, the argument is a double-precision number and the result is an integer.

Other Routines Used

L$22, C$21

IFETCH

Purpose

To fetch the contents of the memory location specified by ARG1.

DAP Calling Sequence

```
CALL   IFETCH
DAC    ARG1
 (Return)
```

Fortran Reference

IFETCH(ARG1)

Method

The A-register is loaded with the contents of the location specified by ARG1.

Other Routines Used

ARG$

Purpose

To truncate the fractional bits from a double precision argument, thus converting it to integer format.

See IFIX, p. 4-51.

IDIM

Purpose

To compute the positive difference between two integer arguments.

DAP Calling Sequence

```
CALL   IDIM
DAC    ARG1    (an integer value)
DAC    ARG2    (an integer value)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

IDIM(I, I)

Method

Compute DIF = ARG1-ARG2. If DIF is positive, the result of this function is the value of DIF. If DIF is negative, the result of this function is zero.

DIF = ARG1 - MIN(ARG1, ARG2)

Data Type of Arguments and Results

The result of this function with two integer arguments is an integer.

Purpose

To clear a specified bit.

DAP Calling Sequence

```
CALL   ICLR
DAC    ARG1    (memory location with bit to be cleared)
OCT    N       (bit to be cleared, 0-15)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

CALL ICLR(I, N)

Method

This subroutine loads a value of 1 into the A-register, shifts it the number of places specified by the value N, complements the A-register, ANDs the A-register with ARG1, stores this value back in ARG1, and exits.

Other Routines Used

F$AT, ISHFT

IAND

Purpose

To logically AND two integers

DAP Calling Sequence

```
CALL   IAND
DAC    ARG1   (an integer value)
DAC    ARG2   (an integer value)
OCT    0      (end of arguments flag)
  (Return)
```

Fortran Reference

IAND(I, I)

Method

A logical AND (ANA) of the two integer arguments is performed and the routine exits.

Data Type of Arguments and Results

This function uses two integer arguments and gives an integer result.

Purpose	To generate the absolute value of an integer.
DAP Calling Sequence	CALL IABS DAC ARG1 (An integer value) (Return)
Fortran Reference	IABS(I)
Method	This subroutine checks the integer argument, ARG1, for its algebraic sign. If the sign is negative, the TWOs complement of ARG1 is calculated. If the sign is positive, the number remains unchanged.
Data Type of Arguments and Results	This absolute value function with an integer argument results in an integer.

FLOAT

Purpose	To convert an integer argument to real format.
DAP Calling Sequence	CALL FLOAT DAC ARG1 (an integer value) (Return)
Fortran Reference	FLOAT (I)
Method	This routine extracts the integer and converts it to real format, leaving the result in the A- and B-registers.
Data Type of Arguments and Results	This routine converts an integer argument to a real number.
Other Routines Used	C$12

Purpose

To calculate e**x, where x is a real number.

DAP Calling Sequence

```
CALL   EXP
DAC    ARG1  (a real number)
 (Return)
```

Fortran Reference

EXP(R)

Method

e**ARG1 = 2**(ARG1 * $\log_2$(e)) = 2**(I+F), where I is the integer and F is the fractional portion of the product ARG1 $\log_2$(e). The value of F is used to define the quantities I', F(1), and F(2):

F	I'	F(1)	F(2)
-1 < F < -1/2	I - I	1/4	F + 3/4
-1/2 < F < 0	I - I	3/4	F + 1/4
0 < F < 1/2	I	1/4	F - 1/4
1/2 < F < 1	I	3/4	F - 3/4

From the above table, e**ARG1=2**(I'+F1+F2) = 2**(I'+F1) * (2**F2)

where

2**F2 = e**(F2*ln(2)) = e**F = (A(F))/(A(F)-B(F))

A(F) = C1+(F*F), B(F) = C2*F

Data Type of Arguments and Results

This exponential function with a real argument (e^R) results in a real number.

Error Message

When overflow occurs, the error message "EX" is reported and the answer returned is the maximum value possible (1.7E38). When underflow occurs, the value 0 is returned without an error message.

Other Routines Used

ARG$, N$22, M$22, S$22, A$22, D$22, F$ER

DSQRT

Purpose	To calculate the square root of a double-precision number.
DAP Calling Sequence	CALL DSQRT DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DSQRT(D)
Method	A first approximation to the double-precision square root of the double-precision argument is obtained by calling the real square root routine (SQRT). One more Newton-Raphson iteration is then made to achieve full double-precision accuracy.
Data Type of Arguments and Results	This square root function of a double-precision argument results in a double-precision number.
Other Routines	FAT, L66, C$62, H$22, SQRT, C$26, H$66, D$66, A$66, A$81

Purpose

To calculate the sine of a double-precision number expressed in radians.

DAP Calling Sequence

```
CALL   DSIN
DAC    ARG1  (a double-precision number)
  (Return)
```

Fortran Reference

DSIN(D)

Method

An arbitrary angle X expressed in radian measure can be reduced to the range $0 \leq Y \leq \frac{pi}{2}$ through the relation X = Y + N(pi/2). Adjustment is made for quadrant before using a modified Taylor's expansion.

Data Type of Arguments and Results

This sine function with a double-precision argument results in a double-precision number.

Other Routines Used

F$AT, DABS, M$66, H$66, C$61, C$16, N$66, A$66, MOD, L$66, S$66

DSIGN

Purpose

To generate a value consisting of the sign of the second double-precision argument and the magnitude of the first double-precision argument.

DAP Calling Sequence

```
CALL    DSIGN
DAC     ARG1    (a double-precision number)
DAC     ARG2    (a double-precision number)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

DSIGN(D, D)

Method

ARG2 is tested for its algebraic sign and, depending on the sign of ARG1, the procedure is as follows:

ARG1	ARG2	Result
-	+	+/ARG1/
-	-	-/ARG1/
+	+	+/ARG1/
+	-	-/ARG1/

Data Type of Arguments and Results

Both arguments for this call are double-precision numbers and the result is a double-precision number.

Other Routines Used

FAT, L66, N$66

Purpose

To compute the remainder resulting from the division of two double-precision numbers.

DAP Calling Sequence

```
CALL    DMOD
DAC     ARG1    (a double precision number)
DAC     ARG2    (a double-precision number)
OCT     0       (end of arguments flag)
  (Return)
```

Fortran Reference

DMOD(D, D)

Method

This subroutine divides ARG1 by ARG2 by calling D$66. The function DMOD (A1, A2) is defined as A1-(A1/A2)*A2, where (A1, A2) is the integer whose magnitude does not exceed the magnitude of A1/A2 and whose sign is the same as that of A1/A2.

Data Type of Arguments and Results

This function with two double-precision arguments results in a double-precision number for a remainder.

Other Routines Used

FAT, L66, D$66, H$66, DINT, M$66, S$66, N$66

DMIN1

Purpose

To find the smallest value in a list of double-precision arguments.

DAP Calling Sequence

```
CALL    DMIN1
DAC     ARG1    (a double-precision argument)
DAC     ARG2    (a double-precision argument)
 .
 .
DAC     ARGn    (last double-precision argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

DMIN1(D, D, . . . D)

Method

Compare the arguments and retain the smallest value.

Data Type of Arguments and Results

Both of the arguments are double-precision and the result of this function is a double-precision number.

Other Routines Used

L$66, H$66, S$66

Purpose

To find the largest value in a list of double-precision arguments.

DAP Calling Sequence

```
CALL    DMAX1
DAC     ARG1    (first double-precision argument)
DAC     ARG2    (a double-precision number)
 .
 .
DAC     ARGn    (last double-precision argument)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

DMAX1(D, D, . . . D)

Method

Compare the arguments and retain the largest value.

Data Type of Arguments and Results

The largest double-precision argument is stored in the double-precision accumulator.

Other Routines Used

L$66, H$66, S$66

DLOG10

Purpose	To calculate the common (base 10) logarithm of a double-precision number.
DAP Calling Sequence	CALL DLOG10 DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DLOG10(D)
Method	See "Method" for DLOG.
Data Type of Arguments and Results	This logarithm function with a double-precision argument results in a double-precision number.
Error Messages	The message "DL" is reported if a negative or zero-valued argument is found. The result is undefined.
Other Routines Used	F$AT, DLOG2, M$66

Purpose	To calculate the common (base 2) logarithm of a double-precision number.
DAP Calling Sequence	CALL DLOG2 DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DLOG2(D)
Method	This routine is used by DLOG and DLOG10 to calculate $\log_2(X)$, where X is equal to F*(2**B) and $1/2 \leq F \leq 1$. See "Method" for DLOG.
Data Type of Arguments and Results	This common logarithm function with a double-precision argument results in a double-precision number.
Error Messages	The message "DL" is reported if a negative or zero-valued argument is found. The result is undefined.
Other Routines Used	FAT, L66, FER, C81, C$16, H$66, Z$80, A$66, S$66, D$66, M$66

DLOG

Purpose

To calculate the natural (base e) logarithm of a double-precision number.

DAP Calling Sequence

```
CALL   DLOG
DAC    ARG1      (a double-precision number)
  (Return)
```

Fortran Reference

DLOG(D)

Method

This routine is also used by DLOG2 and DLOG10. Log A (X), where X = ARG1, is calculated as $\log_2(X)/\log_2(A)$. To calculate $\log_2(X)$, X is considered as the number $F^1*(2**B)$, where $1/2 \leq F < 1$. $\log_2(X) = \log_2(F^1)$ + the binary exponent of F^1, and $\log_2(F^1) = 1/2 + C1*Z + C3(Z**3) + \ldots$ where

$$Z = \frac{(F^1 - \sqrt{2})}{(F^1 + \sqrt{2})}$$

```
C1 = 2.885390081845024D0
C3 = .9617966484737566D0
C5 = .577086624639535D0
C7 = .4115350984570017D0
C9 = .3428071228932386D0
```

Data Type of Arguments and Results

This natural logarithm function of a double-precision argument results in a double-precision number.

Error Messages

The message "DL" is reported if a negative or zero-valued argument is found. The result in the double-precision accumulator is undefined.

Other Routines Used

F$AT, DLOG2, M$66

Purpose	To truncate the fractional bits of a double-precision number.
DAP Calling Sequence	CALL DINT DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DINT(D)
Method	A constant (2**38) is successively added and subtracted from the argument, ARG1. The available precision of double-precision numbers (39 bits) is such that the fractional part of this result is lost. If ARG1 is negative, its TWOs complement is taken before the addition and subtraction take place and it is recomplemented before the subroutine exits. The resultant value is effectively the largest integer $\leq$|ARG1| with the sign of ARG1.
Data Type of Arguments and Results	The double-precision argument after truncation remains a double-precision number.
Other Routines Used	L$66, N$66, A$66, S$66, AC1

DIM

Purpose

To compute the positive difference between two real arguments.

DAP Calling Sequence

```
CALL    DIM
DAC     ARG1    (a real number)
DAC     ARG2    (a real number)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

DIM(R, R)

Method

ARG1 - ARG2 is computed. If the result is positive, this value is the result given. If ARG1 - ARG2 is a negative quantity, the result of this function is zero.

Data Type of Arguments and Results

This routine to calculate the difference between two real numbers results in a real number.

Other Routines Used

L$22, S$22

Purpose

To calculate e**X, where X is a double-precision number.

DAP Calling Sequence

```
CALL   DEXP
DAC    ARG1      (a double-precision number)
  (Return)
```

Fortran Reference

DEXP(D)

Method

In calculating e**ARG1, the following method is used: e**ARG1 = 2**(ARG1*$\log_2$(e)) = 2**(I+F), where I and F are the integer and fractional portions, respectively, of the product ARG1*$\log_2$(e).

Data Type of Arguments and Results

This function raises e to the power of a double-precision argument and gives a double-precision result.

Other Routines Used

FAT, L66, M$66, H$66, C$61, C$16, N$66, A$66, S$66, D$66, A$81

DCOS

Purpose	To calculate the cosine of a double-precision number expressed in radians.
DAP Calling Sequence	CALL DCOS DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DCOS(D)
Method	The cosine function is transformed into the sine function using the trigonometric identity cos (X) = sin (pi/2+X). Sin (pi/2+X) is then evaluated, with X = ARG1.
Data Type of Arguments and Results	This function with a double-precision argument gives a double-precision result.
Other Routines Used	FAT, L66, A$66, H$66, DSIN

Purpose	To convert a real number to double-precision format.
DAP Calling Sequence	CALL DBLE DAC ARG1 (a real number) (Return)
Fortran Reference	DBLE(R)
Method	This subroutine stores the real argument, ARG1, in AC1 and AC2. A word of zeros is appended to the real number as the least significant word of the double-precision fraction and stored in AC3.
Data Type of Arguments and Results	The real argument is converted to a double-precision number.
Other Routines Used	FAT, L22, C$26

DATAN2

Purpose

To calculate the arctangent of the quotient of two double-precision numbers.

DAP Calling Sequence

```
CALL    DATAN2
DAC     ARG1    (a double-precision number (X))
DAC     ARG2    (a double-precision number (Y))
OCT     0       (end of arguments flag)
  (Return)
```

Fortran Reference

DATAN2(D, D)

Method

The arctangent of the quotient (X/Y) is adjusted for the quadrant by examining the signs of the numerator and denominator. See "Method" for ATAN.

Data Type of Arguments and Results

This arctangent function of a double-precision quantity gives a double-precision result.

Error Messages

The error message "DT" is reported if the second argument is zero. The result in the double-precision accumulator is undefined.

Other Routines Used

FAT, L66, H$66, F$ER, D$66, DATAN, S$66, A$66

Purpose	To calculate the arctangent of a double-precision number.
DAP Calling Sequence	CALL DATAN DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DATAN(D)
Method	The principal value is computed. See "Method" for ATAN.
Data Type of Arguments and Results	This function with a double-precision argument results in a double-precision number.
Other Routines Used	F$AT, DABS, H$66, C$81, L$66, A$66, D$66, M$66, N$66

DABS

Purpose	To generate the absolute value of a double-precision number.
DAP Calling Sequence	CALL DABS DAC ARG1 (a double-precision number) (Return)
Fortran Reference	DABS(D)
Method	This subroutine checks the double-precision argument, ARG1, for its algebraic sign. If the sign is negative, the TWOs complement of ARG1 is calculated. If the sign is positive, the number remains unchanged.
Data Type of Arguments and Results	This function with a double-precision argument results in a double-precision number.
Other Routines Used	FAT, L66, N$66

Purpose

To calculate the square root of a complex number.

DAP Calling Sequence

```
CALL   CSQRT
DAC    ARG1   (a complex number)
 (Return)
```

Fortran Reference

CSQRT(C)

Method

If the complex argument is positive, (A+B)**.5 = C+DI is determined as follows:

C = (((A**2+B**2)**.5+A)/2)**.5

D = B/(2*C)

If the argument is negative, ABS(D) = (((A**2+B**2) -A)/2)**.5.

The sign of the real part of the result will be positive and the sign of the imaginary part of the result will be the same as the sign of the imaginary part of the argument. That is, the results will lie in quadrants I or IV of the complex plane.

Data Type of Arguments and Results

This square root function of a complex number results in a complex number.

Other Routines Used

FAT, SUB, CABS, H$22, ABS, A$22, M$22, SQRT, L$22, D$22, L$55

CSIN

Purpose

To calculate the sine of a complex number with the real part in radian measure.

DAP Calling Sequence

```
CALL   CSIN
DAC    ARG1   (a complex number)
  (Return)
```

Fortran Reference

CSIN(C)

Method

The sine function of the complex number ARG1 (X+IY) is computed as follows:

SIN (X+IY) = SIN(X) * COSH (Y) + I * (COS(X) * SINH(Y))

where SINH(Y) = 1/2 * (E**Y -E**-Y)

COSH(Y) = 1/2 * (E**Y+E**-Y)

Data Type of Arguments and Results

The argument and the result of this function are complex numbers.

Other Routines Used

FAT, SUB, EXP, H$22, L$22, D$22, A$22, SIN, M$22, S$22, COS, L$55

Purpose To calculate the cosine of a real number expressed in radians.

See SIN, p. 4-69.

CONJG

Purpose	To obtain the conjugate of a complex number.
DAP Calling Sequence	CALL CONJG DAC ARG1 (a complex number) (Return)
Fortran Reference	CONJG(C)
Method	This subroutine reverses the sign of the imaginary part of the complex argument (ARG1).
Data Type of Arguments and Results	The complex argument in this function remains a complex number.
Other Routines Used	FAT, SUB, L$22, H$22, N$22, L$55

Purpose

To combine two real numbers into one complex quantity.

DAP Calling Sequence

```
CALL   CMPLX
DAC    ARG1    (a real number)
DAC    ARG2    (a real number)
OCT    0       (end of arguments flag)
  (Return)
```

Fortran Reference

CMPLX(R, R)

Method

The first real argument (ARG1) is stored in the real portion of the complex accumulator (AC1 and AC2). The second real argument (ARG2) is stored in the complex portion of the complex accumulator (AC3 and AC4).

Data Type of Arguments and Results

The two real arguments are combined into one complex number and stored in the complex accumulator.

Other Routines Used

FAT, SUB, L$22, H$22, L$55

CLOG

Purpose

To calculate a particular value of the natural logarithm (base e) of a complex number.

DAP Calling Sequence

```
CALL   CLOG
DAC    ARG1   (a complex number)
 (Return)
```

Fortran Reference

CLOG(C)

Method

The following algorithm is used to calculate ln(ARG1), where ARG1 = X + IY:

$$\ln (X+IY) = \ln (X+IY) = R+I(\phi)$$

where R = ln (X**2+Y**2)**.5 = 1/2 ln (X**2+Y**2)

$$\phi = ((TAN^{**}-1)(Y/X) = \phi + 2K\pi$$

where K = 0, ± 1, ± 2, ...

A particular value for ϕ is chosen such that $-\pi \leq \phi \leq \pi$ by entering the arctangent routine ATAN2.

Data Type of Arguments and Results

This logarithm function of a complex number gives a complex result.

Other Routines Used

FAT, L22, M$22, H$22, A$22, ALOG, ATAN2, L$55

Purpose

To calculate the exponential of a complex number with the imaginary part in radian measure.

DAP Calling Sequence

```
CALL   CEXP
DAC    ARG1  (a complex number)
 (Return)
```

Fortran Reference

CEXP(C)

Method

The following algorithm is used to calculate the value of e**ARG1, where ARG1 is a complex number:

If ARG1 = X+IY,

e**(X+IY) = (e**X) * (e**IY) = (e**X) *COS(Y) + I * (e**X) *SIN(Y)

Data Type of Arguments and Results

This function raises e to a complex power and gives a complex result.

Other Routines Used

FAT, SUB, EXP, H$22, COS, M$22, SIN, L$55

CCOS

Purpose

To calculate the cosine of a complex number with the real part in radian measure.

DAP Calling Sequence

```
CALL   CCOS
DAC    ARG1    (a complex number)
  (Return)
```

Fortran Reference

CCOS(C)

Method

The cosine function is transformed into the sine function by use of the trigonometric identity cos (Z) = sin (Z+pi/2), where Z= Y+IY. Sin (Z+pi/2) is then evaluated.

Data Type of Arguments and Results

This cosine function of a complex number results in a complex number.

Other Routines Used

FAT, L55, A$55, H$55, CSIN

Purpose	To generate the absolute value of a complex number.
DAP Calling Sequence	CALL CABS DAC ARG1 (a complex number) (Return)
Fortran Reference	CABS(C)
Method	The argument is squared and its square root is taken to arrive at its absolute value; e.g., if ARG1 = X+IY, CABS(ARG1) = SQRT (X**2+Y**2).
Data Type of Arguments and Results	This absolute value function of a complex number gives a real result.
Other Routines Used	FAT, SUB, L$22, M$22, H$22, A$22, SQRT

Purpose

To compute the remainder resulting from the division of two integers.

DAP Calling Sequence

```
CALL    MOD
DAC     ARG1    (an integer value)
DAC     ARG2    (an integer value)
OCT     0       (end of arguments flag)
  (Return)
```

Fortran Reference

MOD(I, I)

Method

This subroutine divides ARG1 by ARG2 by calling D$11. The function MOD(A1, A2) is defined as A1-(A1/A2)*A2, where (A1/A2) is the integer whose magnitude does not exceed the magnitude of A1/A2 and whose sign is the same as that of A1/A2.

Data Type of Arguments and Results

This function with two integer arguments results in an integer for a remainder.

Other Routines Used

D$11, M$11

NOT

Purpose	To ONEs complement an integer argument.
DAP Calling Sequence	CALL NOT DAC ARG1 (integer value) (Return)
Fortran Reference	NOT(I)
Method	This subroutine performs the ONEs complement (CMA) of the integer argument, ARG1, and exits.
Data Type of Arguments and Results	The argument is an integer and the result is an integer.

Purpose

To check for an error condition.

DAP Calling Sequence

```
CALL   OVERFL
DAC    J        (an integer value)
 (Return)
```

Fortran Reference

OVERFL(J)

Method

This subroutine checks error flag AC5 for a nonzero value, which indicates that an entry to the error subroutine, F$ER, was made since the last call to OVERFL. If AC5 is nonzero, the variable J is set to 1 and AC5 is cleared. If AC5 is zero, J is set to 2

Other Routines Used

AC5

SIGN

Purpose

To generate a value consisting of the sign of the second real argument and the magnitude of the first real argument.

DAP Calling Sequence

```
CALL    SIGN
DAC     ARG1    (a real number)
DAC     ARG2    (a real number)
OCT     0       (end of arguments flag)
 (Return)
```

Fortran Reference

SIGN(R, R)

Method

ARG2 is tested for its algebraic sign and, depending on the sign of ARG1, the procedure is as follows:

ARG1	ARG2	Result
+	+	+ | ARG1 |
+	-	- | ARG1 |
-	+	+ | ARG1 |
-	-	- | ARG1 |

Data Type of Arguments and Results

Both arguments are real numbers and the result is a real number.

Other Routines Used

L$22, N$22

Purpose

To calculate the sine or cosine of a real number expressed in radians.

DAP Calling Sequence

```
CALL    SIN     (or COS)
DAC     ARG1    (a real number)
  (Return)
```

Fortran Reference

(SIN(R) or COS(R)

Method

The angle is reduced to the first quadrant by the use of the relation X = Y+N*(pi/2) and the identities sin(Y) = cos(pi/2-Y) and cos(Y) = sin(pi/2-Y). A modified Taylor's expansion is then used to calculate the sine of the first quadrant angle.

The cosine function is transformed into the sine function by the use of the identity cos(X) = sin(pi/2-X); sin(pi/2-X) is then evaluated, where X=ARG1.

Data Type of Arguments and Results

This sine function with a real argument results in a real number.

Other Routines Used

ARG$, N$22, M$22, S$22, A$22

SLITE

Purpose

To set or reset the pseudo sense lights and switches.

DAP Calling Sequence

```
CALL    SLITE
DAC     ARG1    (where ARG1 is the address of the variable con-
 (Return)        taining the sense light number).

CALL    SLITET  (or CALL  SSWTCH)
DAC     ARG1    (where ARG1 is the address of the variable con-
DAC     ARG2     taining the sense light or switch (SSWTCH)
OCT     0        number to be interrogated, and ARG2 is the
 (Return)        address of the location in which to store the
                 "set or reset" indicator;  (1=set, 2= reset).
```

Fortran Reference

CALL SLITE (I), CALL SLITET(I, J), CALL SSWTCH(I, J)

Method

SLITE --- The ARG$ routine is used to place the variable address in the index register. The argument (I) is tested for zero. If zero, all sense light positions are reset; otherwise, the sense light specified is shifted to its appropriate position and INCLUSIVELY ORed with current settings, leaving them undisturbed.

SLITET --The ARG$ routine is used to place the sense light number in the A-register and the location of the variable in the index register. If the sense light number is 0, a 2 is inserted into the variable J, signifying a reset condition. Otherwise, the sense light bit is moved to its proper position in the A-register. A logical AND is executed with the sense light register. If the result of the AND is zero, the sense light is reset and a 2 is placed in J. If the result of the AND is not zero, an EXCLUSIVE OR is carried out with the sense light register, resetting the sense light specified and storing a 1 in J to signify that the sense light was set on entry.

SSWTCH - The ARG$ routine is used to place the sense switch number in the A-register and the variable location in the index register. If the sense switch number is 0 (no real switch), J is set to 1. If the sense switch number is valid (1 to 4), J is set to 1 if the external switch is set and set to 2 if the external switch is not set.

Other Routines Used

ARG$, L$33

Purpose — To set or reset the pseudo sense lights and switches.

See SLITE, p. 4-70.

SQRT

Purpose

To calculate the square root of a real number. (This subroutine has a high-speed version, SQRTX.)

DAP Calling Sequence

```
CALL    SQRT
DAC     ARG1    (a real number)
 (Return)
```

Fortran Reference

SQRT(R)

Method

Given the argument N = F(2**e), the mantissa is adjusted so that e is even and $1/4 \leq e < 1$. An initial approximation to the square root (Y) is chosen as follows:

$Y = 7/8(F) + 9/32$ if $e < 1/2$

$Y = 9/16(F) + 7/16$ if $e \geq 1/2$

Two Newton-Raphson iterations are then made to obtain full single-precision accuracy.

Data Type of Arguments and Results

This square root function of a real number results in a real number.

Error Messages

The error message "SQ" is reported if a negative argument is found. An undefined result is returned in the A-and B-registers.

Other Routines Used

ARG$, DIV$, D$22, A$22, F$ER

Purpose

To calculate the square root of a real number. (This routine requires the High-Speed Arithmetic Option.)

DAP Calling Sequence

```
CALL SQRTX      (or SQRT)
DAC     ARG1    (a real number)
  (Return)
```

Fortran Reference

SQRT(R)

Method

Given the argument N=F*(2**e), the mantissa is adjusted so that e is even and $1/4 \leq e < 1$. An initial approximation to the square root of ARG1 is chosen as follows:

ARG1 = 7/8(F) + 9/32 if $e < 1/2$

ARG1 = 9/16(F) + 7/16 if $e \geq 1/2$

Two Newton-Raphson iterations are then made to obtain full single-precision accuracy.

Data Type of Arguments and Results

This square root function of a real number results in a real number.

Error Messages

The error message "SQ" is reported if a negative argument is found. An undefined result is returned in the A-and B-registers.

Other Routines Used

ARG$, D$22, A$22, F$ER

SSWTCH

Purpose — To set or reset the pseudo sense switches.

See SLITE, p. 4-70.

Purpose	To calculate the hyperbolic tangent of a real number.
DAP Calling Sequence	CALL TANH DAC ARG1 (a real number) (Return)
Fortran Reference	TANH(R)
Method	TANH = (e**(2*X)-1)/(e**(2*X)+1), where X = ARG1.
Data Type of Arguments and Results	This tangent function with a real argument results in a real number.
Other Routines Used	L$22 EXP, A$22, H$22, D$22

SECTION V
COMPILER SUPPORT SUBROUTINES

This section describes the compiler support subroutines, i.e., those subroutines which are not normally explicitly called by the Fortran programmer. These subroutines perform conversions between data types, logical relationals, arithmetic operations, and miscellaneous functions.

A$21

Purpose

To add an integer argument to a real number.

DAP Calling Sequence

```
CALL  A$21
DAC   ARG2    (an integer value)
(Return)
```

Method

This subroutine adds an integer argument, ARG2, to a real number (in the A- and B-registers). The integer is converted to a real number by calling FLOAT, and the real addition routine (A$22) is called.

Data Type of Arguments and Results

< implicit real argument > + < integer argument > → < real result >

Other Routines Used

FAT, H22, FLOAT, A$22

Purpose

To add or subtract real numbers. This subroutine has a high-speed version, A$22X.

DAP Calling Sequence

```
CALL  A$22    (or S$22)
DAC   ARG2    (a real number)
(Return)
```

Method

A$22 (Add) – The contents of ARG2 are added to the contents of the A- and B-registers after both numbers are unpacked and scaled. The result is normalized and the characteristic is adjusted.

S$22 (Subtract) – The value contained in ARG2 is negated and the add routine, A$22, is entered.

Data Type of Arguments and Results

< implicit real argument> ± < real argument> → < real result>

Error Messages

The error message "SA" is reported if an arithmetic overflow occurs, i.e., the result is ≥ 2**127. An undefined result is returned.

Other Routines Used

ARG$, N$22, F$ER

A$22X

Purpose

To add or subtract real numbers. (This routine requires the High-Speed Arithmetic Option.)

DAP Calling Sequence

```
CALL  A$22    (or S$22)
DAC   ARG2    (a real number)
(Return)
```

Method

A$22 (Add) – The contents of ARG2 are added to the contents of the A- and B-registers after both numbers are unpacked and scaled. The result is normalized and the characteristic is adjusted.

S$22 (Subtract) – The value contained in ARG2 is negated and the add routine, A$22, is entered.

Data Type of Arguments and Results

< implicit real argument> ± < real argument> → < real result>

Error Messages

The error message "SA" is reported if an arithmetic overflow occurs, i.e., the result is ≥ 2**127. An undefined result is returned.

Other Routines Used

N$22, F$ER

Purpose	To add an integer argument to a complex number.
DAP Calling Sequence	CALL A$51 DAC ARG2 (an integer value) (Return)
Method	This routine converts the integer value to a real number by calling C$12 and it calls the complex/real addition routine (A$52).
Data Type of Arguments and Results	< implicit complex argument > + < integer argument > → < complex result >
Other Routines Used	FAT, H55, C$12, H$22, L$55, A$52

A$52

Purpose

To add a real argument to a complex number.

DAP Calling Sequence

```
CALL  A$52
DAC   ARG2    (a real number)
(Return)
```

Method

The following is the algorithm used to compute the operation of adding a real argument (ARG2) to the contents of the complex accumulator (Y):

$$Y + ARG2 = A + B * I + ARG2$$
$$= (A + ARG2) + B * I)$$
$$\text{where } Y = A + B * I$$

Data Type of Arguments and Results

< implicit complex argument > + < real argument > → < complex result >

Other Routines Used

FAT, H55, L$22, A$22, H$22, L$55

Purpose	To add complex numbers.
DAP Calling Sequence	CALL A$55 DAC ARG2 (a complex number) (Return)
Method	The following is the algorithm used in the addition of two complex numbers (the contents of ARG2 and the complex accumulator): X*ARG2 = (A+B*I) + (M+N*I) = (A+M) + (B+N) * I where X = A+B*I and ARG2 = M+N*I
Data Type of Arguments and Results	< implicit complex argument> + < complex argument> → < complex result>
Other Routines Used	FAT, H55, SUB$, L$22, A$22, H$22, L$55

A$61

Purpose

To add an integer argument to a double-precision number.

DAP Calling Sequence

```
CALL  A$61
DAC   ARG2    (an integer value)
(Return)
```

Method

This subroutine calls C$12 to convert the integer argument to real and calls the double-precision/real addition routine (A$62).

Data Type of Arguments and Results

< implicit double-precision argument > + < integer argument > → < double-precision result >

Other Routines Used

FAT, H66, C$12, H$22, L$66, A$62

Purpose	To add a real number to a double-precision number.
DAP Calling Sequence	CALL A$62 DAC ARG2 (a real number) (Return)
Method	This subroutine calls DBLE to convert the real argument to a double-precision number and calls A$66 to perform the double-precision addition.
Data Type of Arguments and Results	< implicit double-precision argument > + < real argument > → < double-precision result >
Other Routines Used	FAT, H66, DBLE, A$66

A$66

Purpose

To add, subtract, multiply, or divide normalized, double-precision numbers. (This subroutine has a high-speed version, A$66X.)

DAP Calling Sequence

```
CALL  A$66    (or S$66, M$66, or D$66)
DAC   ARG2    (a double-precision number)
(Return)
```

Method

The contents of ARG2 are added to, subtracted from, multiplied with, or divided into the contents of the double-precision accumulator (X).

Add (A$66) – The numbers are unpacked and scaled to coincident places. The addition process takes place (X+ARG2), and the result is normalized.

Subtract (S$66) – The numbers are unpacked and scaled to coincident places. The subtraction process takes place (X-ARG2), and the result is normalized.

Multiply (M$66) – X*ARG2 = (X*2**E1) * (Y*2**E2)

= X*ARG2*2** (E1+E2)

Let X = (A+B*2** (-N))

and ARG2 = (C+D*2** (=N))

X*ARG2 = A*C+(A*D+B*C) * 2** (-N))

The term B*D*2** (-2N) is ignored.

The least significant bits of the product are:

L*(A*C)+H*(A*D)+H*(B*C)

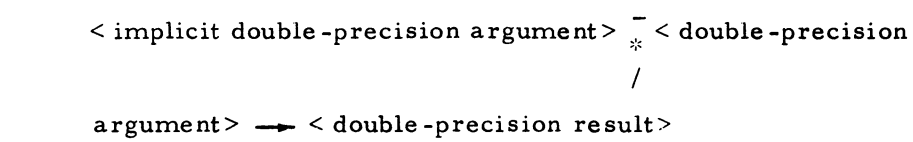
Divide (D$66) – The quotient X/ARG2 is obtained by the binomial expansion of 1/X = X**(=1). The high-order and low-order parts (H and L) of the quotient are computed as follows:

(A+B*2** (-N))/(C+D*2** (-N)) = (A+B=A*D/C)/C

H = (A+B-A*D/C)/C

L = remainder (H)/C

Data Type of Arguments and Results



Error Messages

1. The error message "AD" is printed if an addition or subtraction over/underflow occurs.
2. The error message "PZ" is printed if a division by zero is attempted.
3. The error message "MD" is printed if a multiplication or division over/underflow occurs.

After an error message is reported, the double-precision accumulator is loaded with the maximum ((2**128)-1) or minimum (2**(-128)) value (as determined by the correct sign) before returning to the calling program.

Other Routines Used

N$66, F$ER, H$66, L$66, ARG$, AC1, AC2, AC3

A$66X

Purpose

To add, subtract, multiply, or divide normalized, double-precision numbers. (This routine requires the High-Speed Arithmetic Option.)

DAP Calling Sequence

```
CALL   A$66X   (or A$66, S$66, S$66X, M$66, M$66X, D$66, D$66X)
DAC    ARG2    (a double-precision number)
(Return)
```

Method

The contents of ARG2 are added to, subtracted from, multiplied with, or divided into the contents of the double-precision accumulator. See A$66, described on the preceding pages, for a detailed description of the methods used.

Data Type of Arguments and Results

< implicit double-precision argument > $\begin{matrix}+\\-\\*\\/\end{matrix}$ < double-precision argument > → < double-precision result >

Error Messages

See Error Messages for A$66.

Other Routines Used

N$66, F$ER, H$66, L$66, ARG$, AC1, AC2, AC3

Purpose

To add an integer value (I) to the characteristic of the variable in the double-precision accumulator (effectively, multiplication by 2^I).

DAP Calling Sequence

```
CALL   A$81
DAC    ARG2    (an integer value)
(Return)
```

Method

The characteristic (base 2) of the value in the double-precision accumulator is increased (or decreased) by an integral value, ARG2. For example, if ARG2 = 2 and the value in the double-precision accumulator is 8.0 ($2^{3.0}$), the result of this call would be $2^{3.0+2}$ or $2^{5.0}$ = 32.0 (8.0*2^2). If the absolute value of the result is less than 2**(-128), a value of zero is returned.

Data Type of Arguments and Results

< implicit double-precision argument> * (2** < integer argument>) → < double-precision result>

Error Messages

If there is exponent overflow, an "EQ" error message is reported and external locations AC1 and AC2 are loaded with the maximum value possible ((2**128)-1) with the sign of ARG2.

Other Routines Used

N$22, F$ER, AC1, AC2

AC1

(AC2, AC3, AC4, AC5)

Purpose

Locations to be used as a double-precision or complex accumulator by the Fortran library routines.

Use

AC1, AC2, AC3:	double-precision accumulator.
AC1, AC2:	complex accumulator, real portion.
AC3, AC4:	complex accumulator, imaginary portion.
AC5:	error flag.

Purpose

To convert the indirect address of an argument to its corresponding direct address.

DAP Calling Sequence

```
CALL   ARG$
DAC*   ARG2    (usually a subroutine entry)
(Return)
```

Method

The address of the argument is returned in the index register. This subroutine may be used upon entering a subroutine to set up the return address.

C$12

Purpose	To convert an integer to a real number.
DAP Calling Sequence	CALL C$12 (Return)
Method	The integer value in the A-register is placed in the B-register and the A-register is set to 045600 (octal), representing a characteristic such that the number fits the description given for a real number except that it is not "normalized." A$22 (with argument = 0 (040000,000000), also unnormalized) is called to normalize the result.
Data Type of Arguments and Results	The integer value in the A-register is converted to a real number and placed in the A- and B-registers.
Other Routines Used	A$22, N$22

Purpose	To convert an integer to a complex number.
DAP Calling Sequence	CALL C$15 (Return)
Method	This subroutine converts the integer to a real number by calling C$12 and converts the real number to a complex number by calling C$25.
Data Type of Arguments and Results	An integer value in the A-register is converted to a complex value and the result is placed in the complex accumulator.
Other Routines Used	C$12, C$25

C$16

Purpose	To convert an integer to a double-precision number.
DAP Calling Sequence	CALL C$16 (Return)
Method	The integer in the A-register is normalized and converted to real by calling C$12. This real value is then converted to a double-precision number by calling C$26. The result is placed in the double-precision accumulator. AC1 contains the contents of the B-register (the real exponent), AC2 contains the contents of the A-register (the most significant word of the fraction), and AC3 contains a word of zeros.
Data Type of Arguments and Results	The integer value in the A-register is converted to a double-precision number and placed in the double-precision accumulator.
Other Routines Used	C$12, C$26

Purpose	To convert a real number to an integer.
DAP Calling Sequence	CALL C$21 (Return)
Method	This subroutine scales the real number in the A- and B-registers to 23 bits by adding the octal value 045700 (2**22) to truncate the fractional part of the real number. The result is in the A-register.
Data Type of Arguments and Results	The real number in the A- and B-registers is converted to an integer and returned in the A-register.
Error Messages	The message "RI" is reported if the integer (I) is too large when converted from real to integer. The integer must be in the following range: $-2^{15} \leq I \leq 2^{15}-1$. An undefined result is returned in the A-register.
Other Routines Used	N$22, A$22, F$ER

C$25

Purpose	To convert a real number to a complex number.
DAP Calling Sequence	CALL C$25 (Return)
Method	The A- and B- registers are stored in AC1 and AC2, respectively (the real part of the complex number), and AC3 and AC4 (the imaginary part of the complex number) are set to zeros.
Data Type of Arguments and Results	The real argument in the A- and B- registers is converted to a complex number and stored in the complex accumulator (AC1, AC2, AC3, and AC4).
Other Routines Used	H$22, CMPLX

Purpose	To convert a real number to a double-precision number.
DAP Calling Sequence	CALL C$26 (Return)
Method	The number in the A- and B-registers is placed in AC1 and AC2. AC3 is cleared and the routine exits.
Data Type of Arguments and Results	The real number in the A- and B-registers is converted to double-precision and placed in the double-precision accumulator.
Other Routines Used	AC1, AC2, AC3

C$51

Purpose	To convert a complex number to an integer.
DAP Calling Sequence	CALL C$51 (Return)
Method	This subroutine calls C$52 to convert the value in the complex accumulator to a real number and calls C$21 to convert from real to integer.
Data Type of Arguments and Results	The complex number in the complex accumulator is converted to an integer and placed in the A-register.
Other Routines Used	C$52, C$21

Purpose	To convert a complex number to a real number.
DAP Calling Sequence	CALL C$52 (Return)
Method	The real part of the complex number (in AC1 and AC2) is loaded into the A and B registers as a real number.
Data Type of Arguments and Results	The complex number in the complex accumulator is converted to real format and placed in the A- and B-registers.
Other Routines Used	H$55, L$22

C$61

Purpose	To convert a double-precision number to an integer.
DAP Calling Sequence	CALL C$61 (Return)
Method	This subroutine calls C$62 to convert the number in the double-precision accumulator to real and calls C$21 to convert the real number to integer.
Data Type of Arguments and Results	The double-precision value in the double-precision accumulator is converted to an integer and placed in the A-register.
Other Routines Used	C$62, C$21

Purpose

To convert a double-precision number to a real number.

DAP Calling Sequence

```
CALL  C$62          or    CALL SNGL
(Return)                  DAC  ARG1     (a double-precision
                          (Return)       number)
```

Method

AC1 and AC2 (the exponent and the most significant part of the fraction of the number in the double-precision accumulator) or the first two words of ARG1 (if SNGL is called) are loaded into the A- and B-registers. The least significant part of the fraction (AC3, word 3) is not considered in the result.

Data Type of Arguments and Results

The double-precision value in the double-precision accumulator or in ARG1 is converted to a real number and placed in the A- and B-registers.

Other Routines Used

L$22, N$66, N$22, L$66, AC1, AC2

C$81

Purpose	To convert the exponent of the value in the double-precision accumulator to an integer.
DAP Calling Sequence	CALL C$81 (Return)
Method	Extract the characteristic (base 2) from the value in the double-precision accumulator (AC1) and convert it to an integer.
Data Type of Arguments and Results	The characteristic of the double-precision argument is converted to an integer.
Other Routines Used	AC1

Purpose

To determine whether two complex numbers are equal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. It may be called, however, if desired:

```
CALL   C$EQ
(Return)
```

Fortran Reference

< complex expression> .EQ. < complex expression>

Method

Note for Complex Relationals: Both C$EQ and C$NE expect that a Fortran programmer has used a complex relational expression (C .EQ. C or C .NE. C, where C is a complex expression) which would cause the Fortran compiler to subtract the two complex expressions and leave the result in the complex accumulator (AC1-AC4).

This subroutine, C$EQ, checks the result of the subtraction in AC1 and AC3, the most significant parts of the real and imaginary portions of the complex result, for zero.[1] If they are both zero, the two complex expressions are equal and the relation is true; the A-register is set to 1. If the result of the subtraction is nonzero, the relation is false and the A-register is set to 0.

Caution: Two expressions that are mathematically equal may not be exactly equal when compared if they were calculated in a different manner.

Data Type of Arguments and Results

The two complex expressions are compared and a logical 0 and 1 is returned in the A-register.

Other Routines Used

AC1, AC3

[1] AC2 and AC4 need not be checked, since AC1 and AC3 cannot be zero unless AC2 and AC4 are zero, respectively. (The numbers are normalized, thus moving any nonzero value in AC2 or AC4 to AC1 or AC3, respectively.)

C$NE

Purpose

To determine whether two complex arguments are unequal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. It may, however, be called if desired:

```
CALL  C$NE
(Return)
```

Fortran Reference

< complex expression> .NE. < complex expression>

Method

See "Note for Complex Relationals," page 5-27. This subroutine checks the result of the subtraction in AC1 and AC3, the most significant parts of the real and imaginary portions of the complex result, for zero. (See footnote, page 5-27.) If they are both zero, the two complex expressions are equal and the relation is false; the A-register is set to 0. If either AC1 or AC3 is not zero, the relation is true and the A-register is set to 1.

Caution: Two expressions that are mathematically equal may not be exactly equal when compared if they were calculated in a different manner.

Data Type of Arguments and

The two complex expressions are compared and a logical 0 or 1 is returned in the A-register.

Other Routines Used

AC1, AC3

Purpose

To divide two integers. (This subroutine has a high-speed version, D$11X.)

DAP Calling Sequence

```
CALL   D$11
DAC    ARG2    (integer divisor)
(Return)
```

Method

The numerator (an integer value) should be in the A-register upon entrance to this subroutine. If the denominator, ARG2, is zero, an overflow occurs and an error message is reported. If both arguments are nonzero, the numerator is positioned in the A- and B-registers and the division is performed. The results are examined for the special case (-32,768/-1) which is treated as an overflow. If the results are in the range of -32,768 to +32,767, D$11 returns to the calling program with the quotient in the A-register and the remainder in the B-register. The integer answer is in the A-register.

Data Type of Arguments and Results

< implicit integer argument> / < integer argument> → < integer result>

Error Messages

The error message "IZ" is reported if a division by zero is attempted. The maximum value is output (-32,768 if negative or +32,767 if positive). A division of -32,768 by -1 also causes "IZ" to be reported; D$11 returns a value of +32,767, the maximum value possible.

Other Routines Used

ARG$, F$ER

D$11X

Purpose	To divide two integers. (This routine requires the High-Speed Arithmetic Option.)
DAP Calling Sequence	CALL D$11X (or D$11) DAC ARG2 (integer divisor) (Return)
Method	See "Method" for D$11.
Data Type of Arguments and Results	< implicit integer argument> / < integer argument> → < integer result>
Error Messages	See "Error Messages" for D$11.
Other Routines Used	ARG$, F$ER

Purpose

To divide a real number by an integer argument.

DAP Calling Sequence

```
CALL   D$21
DAC    ARG2    (an integer value)
(Return)
```

Method

This subroutine divides the real number in the A- and B-registers by the integer ARG2. ARG2 is converted to a real number by calling FLOAT, and the real division routine (D$22) is called to perform the division.

Data Type of Arguments and Results

< implicit real argument > / < integer argument > → < real result >

Other Routines Used

FAT, H22, FLOAT, L$22, D$22

D$22

Purpose

To divide two real numbers. (This subroutine has a high-speed version, D$22X.)

See M$22, p. 5-69.

Purpose

To divide two real numbers. (This subroutine requires the High-Speed Arithmetic Option.)

DAP Calling Sequence

```
CALL   D$22
DAC    ARG2    (the real divisor)
(Return)
```

Method

This subroutine divides the real number in the A- and B-registers (X) by the real argument, ARG2 (Y). The division is performed by multiplying X by the reciprocal of Y, i.e., X*1/Y. Newton's method for 1/Y is:

R(1) = R(0) * (2-R(0)*Y

where

R(0) = 1/H(Y), H(Y) being the high-order 15 bits of Y

X * (1/Y = X * R(1) = X * R(0) * (2-R(0)*Y)

Data Type of Arguments and Results

< implicit real argument> / < real argument> → < real result>

Error Messages

A "DZ" error message is typed if division by zero is attempted. A value of 0 is returned if the dividend is also 0. The signed maximum value (±1.7E38) is returned if the dividend is nonzero.

An "SM" error message is reported if an arithmetic overflow occurs. The signed maximum value (±1.7E38) is returned.

A value of 0 is returned for an overflow.

Other Routines Used

N$22, F$ER

D$51

Purpose	To divide a complex number by an integer argument.
DAP Calling Sequence	CALL D$51 DAC ARG2 (an integer value) (Return)
Method	This subroutine calls C$12 to convert the integer to a real number and calls the complex/real division routine (D$52) to perform the division.
Data Type of Arguments and Results	< implicit complex argument> / < integer argument> → < complex result>
Other Routines Used	FAT, H55, C$12, H$22, L$55, D$52

Purpose

To divide a complex number by a real number.

DAP Calling Sequence

```
CALL   D$52
DAC    ARG2 (a real number)
(Return)
```

Method

This subroutine divides the complex value in the complex accumulator (Y) by the real argument, ARG2.

Y/ARG2 = (A+ B*I)/ARG2 = A/ARG2+ B*I, where Y = A+ B*I

Data Type of Arguments and Results

< implicit complex argument> / < real argument> → <complex result>

Other Routines Used

FAT, H55, SUB$, L$22, D$22, H$22, L$55

D$55

Purpose

To divide two complex numbers.

DAP Calling Sequence

```
CALL  D$55
DAC   ARG2 (complex divisor)
(Return)
```

Method

The following algorithm is used to compute the operation of dividing two complex numbers. The contents of the complex accumulator (X) are divided by the contents of ARG2 (Y).

$$X/Y = (A+ B*I)/(M+ N*I)$$

where $X = A+ B*I$ and $Y = M+ N*I$

$$= (A+ B*I)*(M-N*I)/(M+ N*I)*(M-N*I)$$
$$= (A+ B*I)*(M-N*I)/(M**2+ N**2)$$
$$= (A*M+ B*N+ B*M*I- A*N*I/(M**2+N**2)$$
$$= (A*M+ B*N)/(M**2+ N**2)+ (B*M*I-A*N*I)/(M**2+ N**2)$$
$$= (A*M+ B*N)/(M**2+ N**2)+ (I*(B*M-A*N))/(M**2+ N**2)$$

Data Type of Arguments and Results

< implicit complex argument> / < complex argument> → < complex result>

Other Routines Used

FAT, H55, SUB$, L$22, M$22, H$22, A$22, D$22, S$22, N$22, L$55

Purpose	To divide a double-precision number by an integer argument.
DAP Calling Sequence	CALL D$61 DAC ARG2 (an integer value) (Return)
Method	This routine calls C$12 to convert the integer argument to a real number and calls D$62 to perform the double-precision/real division.
Data Type of Arguments and Results	<implicit double-precision argument> / < integer argument> → < double-precision result>
Other Routines Used	FAT, H66, C$12, H$22, L$66, D$62

D$62

Purpose	To divide a double-precision number by a real number.
DAP Calling Sequence	CALL D$62 DAC ARG2 (a double-precision number) (Return)
Method	This subroutine calls DBLE to convert the real divisor (ARG-2) to a double-precision number and calls the double-precision divide routine (D$66).
Data Type of Arguments and Results	< implicit double-precision argument> / < real argument> → < double-precision result>
Other Routines Used	FAT, H66, DBLE, L$66, D$66

Purpose To divide normalized double-precision numbers.

See A$66, page 5-10.

E$11

Purpose

To calculate the value of an integer raised to an integer power. (This subroutine has a high-speed version, E$11X.)

DAP Calling Sequence

```
CALL   E$11
DAC    ARG2      (the integer exponent)
(Return)
```

Method

The implicit integer argument in the A-register and the integer exponent, ARG2, are first examined for the combinations listed below. If one of these combinations is found, the answer is loaded in the A-register for return to the calling program.

Value in A-Register	Exponent	Answer
I	0	1
0	0	1
0	-	+32767
0	+	0
1	J	1
-1	even	1
-1	odd	-1
1	-	0

Otherwise, the value of the expression is calculated and returned in the A-register. The maximum or minimum value computed may not exceed +32,767 or -32,768.

Data Type of Arguments and Results

< implicit integer argument> ** < integer argument> → < integer result>

Error Messages

The error message "II" is reported and +32,767 is returned if overflow occurs or if I = 0 and J is negative (1/0). The value -32,768 is returned if I ≤ -2, J is odd, and overflow occurs.

Other Routines Used

ARG$, M$11, F$ER

Purpose	To calculate the value of an integer raised to an integer power. (This subroutine requires the High-Speed Arithmetic Option.)
DAP Calling Sequence	CALL E$11X DAC J (the integer exponent) (Return)
Method	See "Method" for E$11.
Data Type of Arguments and Results	< implicit integer argument> ** <integer argument> → < integer result>
Error Messages	See "Error Messages" for E$11.
Other Routines Used	ARG$, F$ER

E$21

Purpose	To calculate the value of a real number raised to an integer power.
DAP Calling Sequence	CALL E$21 DAC ARG2 (the integer exponent) (Return)
Method	A**ARG2 is evaluated by multiplying A by itself ARG2-1 times. The sign is determined by the sign of the number in the A- and B- registers and whether I is odd or even.
Data Type of Arguments and Results	<implicit real argument> ** <integer argument> → <real result>
Other Routines Used	ARG$, M$22, D$22

Purpose	To calculate the value of a real argument raised to a real power.
DAP Calling Sequence	CALL E$22 DAC ARG2 (the real exponent) (Return)
Method	X**ARG2 is evaluated as e**(ARG2*log(X)).
Data Type of Arguments and Results	< implicit real argument> ** < real argument> → < real result>
Other Routines Used	ARG$, ALOG M$22, EXP

E$26

Purpose	To calculate the value of a real number raised to a double-precision power.
DAP Calling Sequence	CALL E$26 DAC ARG2 (the double-precision exponent) (Return)
Method	B**ARG2 is evaluated as e**ARG2*log(B)).
Data Type of Arguments and Results	< implicit real argument> ** < double-precision argument> → < double-precision result>
Other Routines Used	FAT, C26, H$66, DLOG, M$66, DEXP

Purpose	To calculate the value of a complex quantity raised to an integer power.
DAP Calling Sequence	CALL E$51 DAC ARG1 (the integer exponent) (Return)
Method	The number in the complex accumulator is multiplied by itself ARG1-1 times.
Data Type of Arguments and Results	<implicit complex argument> ** < integer argument> → < complex result>
Other Routines Used	FAT, H55, IABS, L$55, M$55, D$55

E$52

Purpose	To raise a complex number to a real power.
DAP Calling Sequence	CALL E$52 DAC ARG2 (the real exponent) (Return)
Method	This subroutine converts the real argument (ARG2) to a complex number by calling C$25 and calls E$55 to raise a complex number to a complex power.
Data Type of Arguments and Results	< implicit complex argument> ** < real argument> → < complex result>
Other Routines Used	FAT, H55, L$22, C$25, L$55, E$55

Purpose	To raise a complex number to a complex power.
DAP Calling Sequence	CALL E$55 DAC ARG2 (the complex exponent) (Return)
Method	If the absolute magnitude of the implicit complex number (A) is zero, the routine is exited with an error message. Otherwise the complex result is computed by the following algorithm: CEXP(ARG2 * CLOG(A)).
Data Type of Arguments and Results	< implicit complex argument> ** < complex argument> → < complex result>
Error Messages	The error message "CE" is reported if the absolute value of the implicit argument in the complex accumulator is zero.
Other Routines Used	FAT, H55, CABS, F$ER, CLOG, C$25, M$55, CEXP

E$61

Purpose	To calculate the value of a double-precision number raised to an integer power.
DAP Calling Sequence	CALL E$61 DAC ARG2 (the integer exponent) (Return)
Method	This routine checks for an even-numbered exponent, squares the number in the double-precision accumulator, and divides the integer argument (the exponent) by 2 until the exponent divided by 2 = 1. If the exponent is odd, the computed value (D^{I-1}) is multiplied by the original double-precision number before exiting.
Data Type of Arguments and Results	< implicit double-precision argument> ** < integer argument> → < double-precision result>
Other Routines Used	FAT, H66, L$66, D$66, D$11, M$11, M$66

Purpose	To calculate the value of the number in the double-precision accumulator raised to a real power.
DAP Calling Sequence	CALL E$62 DAC ARG2 (the real exponent) (Return)
Method	B**ARG2 is evaluated as e **(ARG2*DLOG(B)), where B = the contents of the double-precision accumulator.
Data Type of Arguments and Results	< implicit double-precision argument> ** < real argument> → < double-precision result>
Other Routines Used	FAT, H66, DLOG, M$62, DEXP

E$66

Purpose

To calculate the value of a double-precision value raised to a double-precision result.

DAP Calling Sequence

```
CALL   E$66
DAC    ARG2      (the double-precision exponent)
(Return)
```

Method

B**ARG2 is evaluated as e **(ARG2 * LOG (B)), where B = the contents of the double-precision accumulator.

Data Type of Arguments and Results

< implicit double-precision argument> ** < double-precision argument> → < double-precision result>

Other Routines Used

FAT, H66, DLOG, M$66, DEXP

Purpose Argument transfer.

This subroutine is part of the Run-Time Library.

F$ER

Purpose To print error messages.

This subroutine is part of the Run-Time Library.

Purpose	To store (hold) the contents of the A- and B-registers in memory.
DAP Calling Sequence	CALL H$22 DAC ARG1 (location in which the contents of the A- and B-registers are to be stored) (Return)
Method	The contents of memory at the location specified by the argument address, ARG1, are replaced by the contents of the A- and B-registers. The contents of the A- and B-registers remain unchanged.
Data Type of Arguments and Results	This subroutine stores a real number in the argument address.
Other Routines Used	ARG$

H$55

Purpose

To hold (store) the contents of the complex accumulator in memory.

DAP Calling Sequence

```
CALL    H$55
DAC     ARG1    (location in which the contents of the complex
(Return)        accumulator are to be stored)
```

Method

The contents of memory at the location specified by the argument address, ARG1, are replaced by the contents of the complex accumulator. The contents of the accumulator remain unchanged.

Data Type of Arguments and Results

This subroutine stores a complex number in the argument address.

Other Routines Used

ARG$, AC1, AC2, AC3, AC4

Purpose

To hold (store) the contents of the double-precision accumulator in memory.

DAP Calling Sequence

```
CALL   H$66
DAC    ARG1     (location in which the contents of the double-
(Return)         precision accumulator are to be stored)
```

Method

The contents of memory specified by the argument address, ARG1, are replaced by the contents of the double-precision accumulator. The contents of the accumulator are unchanged.

Data Types of Arguments and Results

This subroutine stores a double-precision number in the argument address.

Other Routines Used

ARG$, AC1, AC2, AC3

I$EQ

Purpose

To determine whether two integer expressions are equal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL    I$EQ
(Return)
```

Fortran Reference

< integer expression> .EQ. < integer expression>

Method

Note for Integer Relationals: Subroutines IEQ, IGE, IGT, ILE, I$LT, and I$NE all expect that a Fortran programmer has used an integer relational expression (I_1 .OP. I_2, where I is an integer expression and OP = EQ, GE, GT, LE, LT, or NE). The Fortran compiler subtracts the two integer expressions (I_1-I_2) and leaves the result in the A-register. If overflow occurs, the C-bit is set.

This subroutine, I$EQ, checks the C-bit and the result of the subtraction in the A-register for zero. If the A-register is zero and the C-bit is not set, the relation is true and the A-register is set to 1. Otherwise, the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned in the A-register.

Purpose

To determine whether one integer expression is greater than or equal to another integer expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   I$GE
```

Fortran Reference

< integer expression> .GE. < integer expression>

Method

See "Note for Integer Relationals," p. 5-56.
This subroutine checks the C-bit and the result of the subtraction A-register. If the C-bit is not set and the result in the A-register is greater than or equal to 0, the relation is true and the A-register is set to 1. Otherwise the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned to the A-register.

I$GT

Purpose

To determine whether one integer expression is greater than another integer expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   I$GT
(Return)
```

Fortran Reference

< integer expression> .GT. < integer expression>

Method

See "Note for Integer Relationals," on p. 5-56.
This subroutine checks the C-bit and the result of the subtraction in the A-register. If the C-bit is not set and the result in the A-register is a positive number, the relation is true and the A-register is set to 1. Otherwise the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned in the A-register.

Purpose

To determine whether one integer expression is less than or equal to another integer expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   I$LE
(Return)
```

Fortran Reference

< integer expression> .LE. < integer expression>

Method

See "Note for Integer Relationals," p. 5-56.
This subroutine checks the C-bit and the result of the subtraction in the A-register. If the C-bit is not set and the result in the A-register is less than or equal to 0, the relation is true and a logical 1 is returned in the A-register. Otherwise the relation is false and a 0 is returned in the A-register.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned in the A-register.

I$LT

Purpose

To determine whether one integer expression is less than another.

DAP Calling Sequence

This subroutine is not intended for use by DAP programmers. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called, if desired:

```
CALL   I$LT
(Return)
```

Fortran Reference

< integer expression> .LT. < integer expression>

Method

See "Note for Integer Relationals," on p. 5-56.
This subroutine checks the C-bit and the result of the subtraction in the A-register. If the C-bit is not set and the result in the A-register is negative, the relation is true and the A-register is set to 1. Otherwise the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned in the A-register.

Purpose

To determine whether two integer expressions are unequal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   I$NE
(Return)
```

Fortran Reference

< integer expression > .NE. < integer expression >

Method

See "Note for Integer Relationals," p. 5-56. This subroutine checks the C-bit and the result of the subtraction in the A-register. If the C-bit is not set and the result in the A-register is 0, the relation is false and a 0 is returned in the A-register. Otherwise the relation is true and the A-register is set to 1.

Data Type of Arguments and Results

The two integer arguments are compared and a logical 0 or 1 is returned in the A-register.

L$22

Purpose

To load a real number into the A- and B-registers.

DAP Calling Sequence

```
CALL   L$22   or   CALL REAL
DAC    ARG1        (a real number)
(Return)
```

Method

This subroutine calls ARG$ to place the address of the argument, ARG1, into the index register. ARG1 is then loaded into the A- and B-registers.

Other Routines Used

ARG$

Purpose

To form an INCLUSIVE OR from memory with the value in the A-register.

DAP Calling Sequence

```
CALL    L$33
DAC     ARG1    (an integer value)
(Return)
```

Method

The value in the A-register is EXCLUSIVELY ORed, ANDed and EXCLUSIVELY ORed again with the argument, ARG1.

L$55

Purpose

To load a complex number into the complex accumulator.

DAP Calling Sequence

```
CALL   L$55
DAC    ARG1    (a complex number)
(Return)
```

Method

This subroutine calls ARG$ to place the address of the argument, ARG1, into the index register. ARG1 is then loaded into the complex accumulator.

Other Routines Used

ARG$, AC1, AC2, AC3, AC4

Purpose	To load a double-precision number into the double-precision accumulator.
DAP Calling Sequence	CALL L$66 DAC ARG1 (a double-precision number) (Return)
Method	This subroutine calls ARG$ to place the address of the argument, ARG1, into the index register. ARG1 is then loaded into the double-precision accumulator.
Other Routines Used	ARG$, AC1, AC2, AC3

M$11

Purpose

To multiply two integers. (This subroutine has a high-speed version, M$11X.)

DAP Calling Sequence

```
CALL   M$11
DAC    ARG2    (integer multiplier)
(Return)
```

Method

This subroutine multiplies the value in the A-register by the integer argument, ARG2. If either or both are negative, a sign counter is incremented and the negative value(s) are made positive. The multiplier, ARG2, is loaded into the B-register and shifted to place the low-order bit of the multiplier in the C-register. The C-bit is tested and if it is set, the multiplicand is added to the A-register. The A- and B-registers are shifted together 1 bit, with the new low-order bit going into the C-register, and so forth, for 16 shifts. When these right shifts are completed, the bits are shifted back into the A-register, one at a time, checking for overflow. The positive or negative result is returned in the A-register.

Data Type of Arguments and Results

< implicit integer argument> * < integer argument> → < integer result>

Error Messages

When an over/underflow occurs, the error message "IM" is reported. The subroutine returns with +32,767 in the A-register if the answer is positive, or -32,768 if it is negative.

Other Routines Used

ARG$, F$ER

Purpose

To multiply two integers. (This subroutine requires the High-Speed Arithmetic Option.)

DAP Calling Sequence

```
CALL    M$11
DAC     ARG2    (an integer value)
(Return)
```

Method

This subroutine multiplies the value in the A-register by ARG2. The result is then examined for over/underflow (see "Error Messages"). If the result is in the proper range, the signed result is returned to the calling program in the A-register.

Data Type of Arguments and Results

<implicit integer argument> * <integer argument> → <integer result>

Error Messages

See "Error Messages" for M$11.

Other Routines Used

ARG$, F$ER

M$21

Purpose

To multiply a real number by an integer.

DAP Calling Sequence

```
CALL   M$21
DAC    ARG2    (an integer value)
(Return)
```

Method

This subroutine calls FLOAT to convert the integer argument to a real number and calls the real multiplication routine (M$22).

Data Type of Arguments and Results

<implicit real argument> * <integer argument> → <real result>

Other Routines Used

FAT, H22, FLOAT, M$22

Purpose

To multiply or divide two real numbers. (This subroutine has a high-speed version, M$22X.)

DAP Calling Sequence

```
CALL   M$22   (or D$22)
DAC    ARG2   (multiplier
(Return)      or divisor)
```

The multiplicand (M$22) or dividend (D$22) must be in the A- and B-registers. The sign, exponent, and most significant bits will be in the B-register.

Method

X*Y = (X*2**B)*(Y*2**C), where X = the value in the A- and B-registers
Y = ARG 2

= ABS(X)*ABS(Y)*2**(BC)
ABS(X)*ABS(Y) = X(1)*Y(1)+(X(1)*Y(2)+X(2)*Y(1))*2**-15
The most significant part of the product is H(X(1)*Y(1)) and the least significant part is L(X(1)*Y(1))+H(H(1)*Y(2))+H(X(2)*Y(1))*2**-15.
Newton's method for 1/Y is R(1) = R(0)*(2-R(0)*Y), where R(0) = 1/H(Y), H(Y) being the high-order 15 bits of Y.
X(1/Y) = X*R(1) = X*R(0)*(2-R(0)*Y).

Data Type of Arguments and Results

< implicit real argument> * < real argument> → < real result>

Error Messages

Multiplication - If there is underflow, a value of zero is returned with no error message.

If there is overflow, an "SM" error message is reported and the maximum value ((2**128)-1) is returned in the A- and B-registers.

Division - If division by zero is attempted, a "DZ" error message is reported and the result in the A- and B-registers is undefined.

If the divisor is unnormalized, an "SD" error message is reported and the result in the A- and B-registers is undefined.

Other Routines Used

N$22, ARG$, F$ER

M$22X

Purpose

To multiply two real numbers.

DAP Calling Sequence

```
CALL   M$22
DAC    ARG2    (a real number)
(Return)
```

Method

X*Y = (X*2**B)*(Y*2**C), where X = the value in the A- and B-registers
Y = ARG2

= ABS(X)*ABS(Y)*2**(BC)

ABS(X)*ABS(Y) = X(1)*Y(1)*(X(1)*Y(2)+X(2)*Y(1))*2**-15

The most significant part of the product is H((X(1)*Y(1)) and the least significant part is L(X(1)*Y(1))+H(H(1)*Y(2))+H(X(2)*Y(1)) *2**-15.

Data Type of Arguments and Results

< implicit real argument> * < real argument> → < real result>

Error Messages

Underflow - A value of zero is returned with no error message.
Overflow - An "SM" error message is reported and a signed maximum value (±1.7E38) is returned.

Other Routines Used

F$ER

Purpose	To multiply a complex number by an integer.
DAP Calling Sequence	CALL M$51 DAC ARG2 (an integer value) (Return)
Method	This subroutine calls C$12 to convert the integer argument to a real number and calls the complex/real multiplication routine (M$52).
Data Type of Arguments and Results	< implicit complex argument> * < integer argument> → < complex result>
Other Routines Used	FAT, H55, C$12, H$22, L$55, M$52

M$52

Purpose

To multiply a complex number by a real number.

DAP Calling Sequence

```
CALL    M$52
DAC     ARG2    (a real number)
(Return)
```

Method

Y*X = (A+B*I)*X = A*X+(B*X)*I
where Y = A+B*I (in the complex accumulator)
X = ARG2

Data Type of Arguments and Results

< implicit complex argument > * < real argument > → < complex result >

Other Routines Used

FAT, H55, SUB$, L$22, M$22, H$22, L$55

Purpose

To multiply complex numbers.

DAP Calling Sequence

```
CALL   M$55
DAC    ARG2    (a complex value)
(Return)
```

Method

This routine multiplies the contents of the complex accumulator (X) by the value in ARG2 (Y).
X*Y = (A+B*I) (M+N*I) = A*M-B*N+(A*N+B*M)*I
where X = A+B*I and Y = M+N*I.

Data Type of Arguments and Results

< implicit complex argument> * < complex argument> → < complex result>

Other Routines Used

FAT, H55, SUB$, L$22, M$22, H$22, S$22, N$22, A$22, L$55

M$61

Purpose

To multiply a double-precision number by an integer argument.

DAP Calling Sequence

```
CALL    M$61
DAC     ARG2    (an integer value)
(Return)
```

Method

This subroutine calls C$12 to convert the integer argument to a real number and calls the double-precision/real multiplication routine (M$62).

Data Type of Arguments and Results

< implicit double-precision argument > * < integer argument > → < double-precision result >

Other Routines Used

FAT, H66, C$12, H$22, L$66, M$62

Purpose

To multiply a double-precision number by a real number.

DAP Calling Sequence

```
CALL    M$62
DAC     ARG2    (real multiplier)
(Return)
```

Method

This subroutine calls DBLE to convert the real multiplier to a double-precision number and calls the double-precision multiply routine (M$66).

Data Type of Arguments and Results

< implicit double-precision argument > * < real argument > → < double-precision result >

Other Routines Used

FAT, H66, DBLE, M$66

M$66

Purpose

To multiply normalized, double-precision numbers.

See A$66, on p. 5-10.

Purpose

To determine the TWOs complement of a real number.

DAP Calling Sequence

```
CALL   N$22
DAC    ARG1    (a real number)
(Return)
```

Method

The C-bit is preset on entrance to this routine to provide a true TWOs complement if the low-order word is found to be zero. The C-bit is reset when this is not the case, and the A- and B-registers are TWOs complemented normally.

Data Type of Arguments and Results

The TWOs complement of the real argument is computed and the routine exits with the the real result in the A- and B-registers.

N$33

Purpose

To obtain the complement of a logical value.

DAP Calling Sequence

```
CALL   N$33
(Return)
```

Method

The least significant bit of the argument in the A-register is logically complemented, changing its value from true to false (1 to 0) or false to true (0 to 1).

Purpose	To negate a complex quantity.
DAP Calling Sequence	CALL N$55 (Return)
Method	The signs of the real part and the complex part of the complex number are negated. The result is in the complex accumulator.
Data Type of Arguments and Results	The complex argument is negated and the subroutine exits, with the complex result in the complex accumulator.
Other Routines Used	H$55, SUB$, L$22, N$22, H$22, L$55

N$66

Purpose

To negate a double-precision number.

DAP Calling Sequence

```
CALL   N$66
(Return)
```

Method

This subroutine negates the value in the double-precision accumulator. The double-precision word is effectively TWOs complemented, as follows:

1. The lowest order word, AC3, word 3, is tested for zero. If it is not zero, the word is TWOs complemented. If it is zero, the C-bit is set.

2. AC2, word 2, is tested for zero. If it is not zero, the word is ONEs complemented and the C-bit is added. If it is zero and the C-bit is set, no action is taken. If the C-bit is not set, the word is ONEs complemented.

3. AC1, word 1, is ONEs complemented, and the C-bit, if set, is added. The negated result is left in the double-precision accumulator.

Data Type of Arguments and Results

The double-precision argument is negated and the routine exits with a double-precision result in AC1, AC2, and AC3.

Other Routines Used

AC1, AC2, AC3

Purpose

To determine whether two integer expressions are equal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   R$EQ
(Return)
```

Fortran Reference

< real or double-precision expression> .EQ. < real or double-precision expression>

Method

Note for Real or Double-Precision Relationals:

Subroutines REQ, RGE, RGT, RLE, R$LT, and R$NE all expect that a Fortran programmer has used a real or double-precision relational expression (R_1 or D_1 .OP. R_2 or D_2), where R is a real expression, D is a double-precision expression, and OP = EQ, GE, GT, LE, LT, or NE. The Fortran compiler subtracts the two expressions (R_1 or D_1 - R_2 or D_2) and leaves the characteristic and most significant part of the result (the contents of AC1 for double-precision expressions) in the A-register. Note that this result in the A-register can be zero only if the entire double-precision number (AC1-AC3) is zero, since the number is normalized and any nonzero values in AC2 and AC3 are moved to AC1.

This subroutine, R$EQ, checks the result of the subtraction in the A-register for 0. If zero, the relation is true and the A-register is set to 1. Otherwise the relation is false and the A-register is set to 0. Caution: Two expressions that are mathematically equal may not be exactly equal when compared if they were calculated in a different manner.

Data Type of Arguments and Results

Two real or double-precision numbers are compared and a logical 0 or 1 is returned in the A-register.

R$GE

Purpose

To determine whether one real or double-precision expression is greater than or equal to another real or double-precision expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   R$GE
(Return)
```

Fortran Reference

< real or double-precision expression> .EQ. < real or double-precision expression>

Method

See "Note for Real or Double-Precision Relationals", on p. 5-81. This subroutine checks the result of the subtraction in the A-register for a value greater than or equal to zero and sets the A-register to 1 if it finds such a value. Otherwise the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

Two real or double-precision numbers are compared and a logical value of 0 or 1 is returned in the A-register.

Purpose

To determine whether one real or double-precision number is greater than another real or double-precision number.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   R$GT
(Return)
```

Fortran Reference

< real or double-precision expression> .GT. < real or double-precision expression>

Method

See "Note for Real or Double-Precision Relationals," on p. 5-81. This subroutine checks the result of the subtraction in the A-register. If the value in the A-register is greater than 0, the relation is true and the A-register is set to 1. Otherwise the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

Two real or double-precision arguments are compared and a logical 0 or 1 is returned in the A-register.

R$LE

Purpose

To determine whether one real or double-precision expression is less than or equal to another real or double-precision expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   R$LE
(Return)
```

Fortran Reference

<real or double-precision expression>.LE.<real or double-precision expression>

Method

See "Note for Real or Double-Precision Relationals," p. 5-81. This subroutine checks the result of the subtraction in the A-register for a value less than or equal to 0 and sets the A-register to 1 if it finds a value in that range. Otherwise, the relation is false and the A-register is set to 0.

Data Type of Arguments and Results

Either argument may be real or double-precision and a logical 0 or 1 is returned in the A-register.

Purpose

To determine whether one real or double-precision expression is less than another real or double-precision expression.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL   R$LT
(Return)
```

Fortran Reference

<real or double-precision expression> .LT. < real or double-precision expression>

Method

See "Note for Real or Double-Precision Relationals," p. 5-81. This subroutine checks the result of the subtraction in the A-register for a value greater than or equal to 0 and sets the A-register to 0 if it finds such a value. Otherwise, the relation is true and the A-register is set to 1.

Data Type of Arguments and Results

The two real or double-precision arguments are compared and a logical 0 or 1 is returned in the A-register.

R$NE

Purpose

To determine whether two real or double-precision expressions are unequal.

DAP Calling Sequence

This subroutine is not intended for use by a DAP programmer. A CAS instruction with appropriate jumps would be more efficient code. This subroutine may, however, be called if desired:

```
CALL    R$NE
(Return)
```

Fortran Reference

<real or double-precision expression> .NE. < real or double-precision expression>

Method

See "Note for Real or Double-Precision Relationals," page 5-81.
This subroutine checks the result of the subtraction in the A-register for 0. if zero, the relation is false and the subroutine exits with a 0 in the A-register. Otherwise the relation is true and the A-register is set to 1. Caution: Two expressions that are mathematically equal may not be exactly equal when compared if they were calculated in a different manner.

Data Type of Arguments and Results

Two real or double-precision arguments are compared and a logical 0 or 1 is returned in the A-register.

Purpose	To subtract an integer from a real number.
DAP Calling Sequence	CALL S$21 DAC ARG2 (an integer value) (Return)
Method	This subroutine converts the integer argument to a real number by calling FLOAT and calls the real subtraction routine (S$22).
Data Type of Arguments and Results	<implicit real argument> – <integer argument> → <real result>
Other Routines Used	FAT, H22, FLOAT, S$22, N$22

S$22

Purpose — To subtract real numbers.

See A$22, page 5-3.

Purpose	To subtract an integer argument from a complex number.
DAP Calling Sequence	CALL S$51 DAC ARG2 (an integer value) (Return)
Method	This subroutine calls C$12 to convert the integer argument to a real number and calls the complex/real subtraction routine (S$52).
Data Type of Arguments and Results	< implicit complex argument > − < integer argument > → < complex result >
Other Routines Used	FAT, H55, C$12, H$22, L$55, S$52

S$52

Purpose

To subtract a real number from a complex number to obtain a complex result.

DAP Calling Sequence

```
CALL    S$52
DAC     ARG2      (a real number)
(Return)
```

Method

Y-X = A+B*I-X = (A-X)+B*I

where Y = A+B*I, X = ARG2

Data Type of Arguments and Results

< implicit complex argument> — < real argument> → < complex result>

Other Routines Used

FAT, H55, L$22, S$22, H$22, L$55

Purpose

To subtract two complex numbers.

DAP Calling Sequence

```
CALL    S$55
DAC     ARG2    (the complex subtrahend)
(Return)
```

Method

This subroutine subtracts ARG2(Y) from the value in the complex accumulator (X):

X - Y = (A+B*I) - (M+N*I) = A*M+(B*N)*I

where X = A+B*I, Y = M+N*I

Data Type of Arguments and Results

<implicit complex argument> — <complex argument> → <complex result>

Other Routines Used

FAT, H55, SUB$, L$22, S$22, N$22, H$22, L$55

S$61

Purpose

To subtract an integer argument from a double-precision number.

DAP Calling Sequence

```
CALL    S$61
DAC     ARG2    (an integer value)
(Return)
```

Method

This subroutine calls C$12 to convert the integer argument to a real number and calls the double-precision/real subtraction routine (S$62).

Data Type of Arguments and Results

< implicit double-precision argument > — < integer argument > → < double-precision result >

Other Routines Used

FAT, H66, C$12, H$22, L$66, S$62

Purpose	To subtract a real argument from a double-precision number.
DAP Calling Sequence	CALL S$62 DAC ARG2 (a real number) (Return)
Method	This subroutine calls DBLE to convert the real argument to a double-precision number and enters the double-precision subtraction routine (S$66).
Data Type of Arguments and Results	< implicit double-precision argument > — < real argument > → < double-precision result >
Other Routines Used	FAT, H66, DBLE, S$66, N$66

S$66

Purpose — To subtract normalized, double-precision numbers.

See A$66, page 5-10.

Purpose To calculate an array size.

See SUB$, page 5-97.

SNGL

Purpose — To convert a double-precision number to a real number.

See C$62, page 5-25.

Purpose

To calculate the address of a referenced array element or to calculate the array size.

DAP Calling Sequence

```
CALL   SUB$
DAC or DAC* ARRAY TABLE1
DAC or DAC* SUBSCRIPT 1
DAC or DAC* SUBSCRIPT 2
  .
  .
DAC or DAC* SUBSCRIPT N
(Return)
```

or

```
CALL   SIZ$
DAC or DAC* ARRAY TABLE2
(Return)
```

Method

ARRAY TABLE1 Layout

```
DAC or DAC* ARRAY
OCT     L       (number of words per array element)
DEC     DIMENSION 1
DEC     DIMENSION 2
  .
  .
DEC     DIMENSION N
OCT     0       (end of dimension list)
```

ARRAY TABLE2 Layout

```
DAC or DAC* ARRAY
OCT     KEY
OCT or DAC* DIMENSION 1
OCT or DAC* DIMENSION 2
  .
  .
OCT or DAC*    DIMENSION N    or    OCT   ARRAY SIZE
  or omitted
```

The KEY bit pattern is CVDDDDDDDDDDDLLL, where

C = 0 - no array bounds checking
C = 1 - array bounds checking
V = 0 - last word of array table is array size
V = 1 - last word of array table is dimension

If C = 0 and V = 0, the last dimension word of the array table is omitted.

D = dimensionality - limited to 2047
L = number of words per array element

Note that L is determined by the data type of the array as follows:

Data Type of Array	
L =	1 - integer or logical
	2 - real
	3 - double-precision
	4 - complex

Let S denote the array starting address, L the number of words per array element, S(I) the Ith subscript value, and D(I) the Ith dimension for an N-dimensional array A where $N \geq 1$.

The address of the array element A(S(1), S(2), ...S(N) is given by

$$S + L(\ldots, (S(N)-1)*D(N-1) + \ldots + (S(2)-1)*D(1) + (S(1)-1)$$

Error Messages

The error message "AO" (array overflow) is reported if the array element referenced is outside the bounds of the array. Only the final array element referenced is checked for legality, not individual subscript values.

Other Routines Used

M$11, F$ER

Purpose	To clear (zero-out) the exponent of the variable in the double-precision accumulator.
DAP Calling Sequence	CALL Z$80 (Return)
Method	Extract the value in AC1 and replace the characteristic (base 2) in bits 2-9 with zeros.
Other Routines Used	AC1

APPENDIX A
TAPE CONTENTS

MAGNETIC TAPE 70185015000A (LIBRARY SOURCES)

This tape consists of the concatenation of individual source magnetic tapes of the following programs in the order listed:

NAME:	DOC NO.	REV
LTCSMT	70185288000	A
LTCL1	70185285000	A
LTCL2S	70185286000	A
LTCL2H	70185287000	A
C$15	70185249000	A
A$51	70185254000	A
C$51	70185255000	A
D$51	70185256000	A
M$51	70185257000	A
S$51	70185258000	A
C$52	70185258000	A
E$52	70185259000	A
E$55	70185260000	A
A$61	70185261000	A
D$61	70185262000	A
M$61	70185263000	A
S$61	70185264000	A
E$62	70180053000	C
E$61	70180052000	D
E$26	70182582000	C
E$66	70180054000	C
DSQRT	70182580000	C
DCOS	70180055000	C
DSIN	70182583000	D
DEXP	70182581000	C
DLOG10	70180051000	C
DLOG	70182579000	D
DLOG2	70182914000	B
DATAN2	70180056000	C
DATAN	70182584000	C
DMOD	70182588000	C
DSIGN	70182589000	C
DABS	70182587000	C
A$62	70180037000	C
S$62	70180038000	C
M$62	70180039000	C
D$62	70180040000	C
C$16	70180059000	C
DBLE	70180058000	C
CSQRT	70182592000	D
CCOS	70180066000	C
CSIN	70182595000	D
CLOG	70182591000	D
CEXP	70182593000	C

CABS	70182596000	C
E$51	70182594000	C
A$52	70180041000	C
S$52	70180042000	C
M$52	70180043000	C
D$52	70180044000	C
A$55	70182544000	C
S$55	70180093000	C
M$55	70182545000	C
D$55	70180034000	C
CONJG	70182598000	C
C$25	70180068000	C
CMPLX	70182597000	C
N$55	70180069000	C
C$EQ	70185276000	A
C$NE	70185277000	A
DMAX1	70182585000	D
DMIN1	70182586000	D
DINT	70180850000	C
Z$80	70180851000	C
A$81	70180852000	C
C$61	70182554000	C
H$55	70180860000	C
AIMAG	70180858000	D
L$55	70180859000	C
A$21	70185250000	A
D$21	70185251000	A
M$21	70185252000	A
S$21	70185253000	A
IAND	70185266000	A
IXOR	70185267000	A
ITEST	70185046000	A
ICLR	70185047000	A
ISET	70185048000	A
ISHFT	70185268000	A
IOR	70185269000	A
NOT	70185284000	A
I$GE	70185270000	A
I$GT	70185271000	A
I$LE	70185272000	A
I$NE	70185273000	A
I$LT	70185274000	A
I$EQ	70185275000	A
R$EQ	70185278000	A
R$GE	70185279000	A
R$GT	70185280000	A
R$LE	70185281000	A
R$LT	70185282000	A
R$NE	70185283000	A
A$66	70180853000	D
A$66X	70180979000	C
H$66	70180855000	C
C$26	70180857000	C
MAX0	70182548000	D
MAX1	70182549000	D
MIN0	70180649000	B
MIN1	70182551000	D
TANH	70182565000	D
SQRT	70182560000	C
SQRTX	70180681000	C
SIN,COS	70182563000	D
ATAN	70182564000	D
E$21	70182562000	C
E$22	70180045000	C
ALOG	70182559000	E
ALOGX	70180682000	C
EXP	70182561000	D

E$11	70182547000	D
E$11X	70180684000	B
ABS	70182570000	C
C$62	70180884000	E
AMOD	70182572000	C
L$66	70180854000	C
AINT	70182571000	C
N$66	70180856000	C
DIM	70182573000	C
SIGN	70182574000	C
IFIX	70182553000	C
FLOAT	70180062000	C
C$12	70182575000	C
C$21	70182558000	E
LOC	70181962000	B
C$81	70180882000	C
ISTORE	70181982000	B
N$33	70180090000	C
IFETCH	70181983000	B
IABS	70182552000	C
MOD	70182555000	D
SUB$	70185150000	A
IDIM	70182556000	C
A$22	70182536000	F
M$22	70182537000	E
A$22X	70181805000	B
M$22X	70181806000	B
D$22X	70181804000	B
ISIGN	70182557000	C
L$22	70182534000	D
H$22	70182535000	D
N$22	70180097000	C
SLITE	70182599000	E
M$11	70180035000	D
D$11	70182546000	D
M$11X	70180685000	B
D$11X	70180686000	B
OVERFL	70180894000	C
F$AT	70180071000	D
L$33	70180065000	C
ARG$	70180072000	C
AC1	70180717000	C

PAPER TAPE 70185012000A (FTNLB1)

This tape consists of the concatenation of individual object paper tapes of the following programs in the order listed (each object has been assembled via the DAP/700 Macro Assembler and may be linked to a calling program by the Linkage Editor):

NAME:	DOC NO.	REV
C$15	70185249000	A
A$51	70185254000	A
C$51	70185255000	A
D$51	70185256000	A
M$51	70185257000	A
S$51	70185258000	A
C$52	70185258000	A
E$52	70185259000	A
E$55	70185260000	A
A$61	70185261000	A
D$61	70185262000	A
M$61	70185263000	A
S$61	70185264000	A

E$62	70180053000	C
E$61	70180052000	D
E$26	70182582000	C
E$66	70180054000	C
DSQRT	70182580000	C
DCOS	70180055000	C
DSIN	70182583000	D
DEXP	70182581000	C
DLOG10	70180051000	C
DLOG	70182579000	D
DLOG2	70182914000	B
DATAN2	70180056000	C
DATAN	70182584000	C
DMOD	70182588000	C
DSIGN	70182589000	C
DABS	70182587000	C
A$62	70180037000	C
S$62	70180038000	C
M$62	70180039000	C
D$62	70180040000	C
C$16	70180059000	C
DBLE	70180058000	C
CSQRT	70182592000	D
CCOS	70180066000	C
CSIN	70182595000	D
CLOG	70182591000	D
CEXP	70182593000	C
CABS	70182596000	C
E$51	70182594000	C
A$52	70180041000	C
S$52	70180042000	C
M$52	70180043000	C
D$52	70180044000	C
A$55	70182544000	C
S$55	70180093000	C
M$55	70182545000	C
D$55	70180034000	C
CONJG	70182598000	C
C$25	70180068000	C
CMPLX	70182597000	C
N$55	70180069000	C
C$EQ	70185276000	A
C$NE	70185277000	A
DMAX1	70182585000	D
DMIN1	70182586000	D
DINT	70180850000	C
Z$80	70180851000	C
A$81	70180852000	C
C$61	70182554000	C
H$55	70180860000	C
AIMAG	70180858000	D
L$55	70180859000	C

PAPER TAPE 70185013000A (FTNLB2S)

This tape consists of the concatenation of individual object tapes of the following programs in the order listed (each object has been assembled via the DAP/700 Macro Assembler and may be linked to a calling program by the Linkage Editor):

NAME:	DOC NO.	REV
A$21	70185250000	A
D$21	70185251000	A
M$21	70185252000	A
S$21	70185253000	A

IAND	70185266000	A
IXOR	70185267000	A
ITEST	70185046000	A
ICLR	70185047000	A
ISET	70185048000	A
ISHFT	70185268000	A
IOR	70185269000	A
NOT	70185284000	A
I$GE	70185270000	A
I$GT	70185271000	A
I$LE	70185272000	A
I$NE	70185273000	A
I$LT	70185274000	A
I$EQ	70185275000	A
R$EQ	70185278000	A
R$GE	70185279000	A
R$GT	70185280000	A
R$LE	70185281000	A
R$LT	70185282000	A
R$NE	70185283000	A
A$66	70180853000	D
H$66	70180855000	C
C$26	70180857000	C
MAX0	70182548000	D
MAX1	70182549000	D
MIN0	70180649000	B
MIN1	70182551000	D
TANH	70182565000	D
SQRT	70182560000	C
SIN,COS	70182563000	D
ATAN	70182564000	D
E$21	70182562000	C
E$22	70180045000	C
ALOG	70182559000	E
EXP	70182561000	D
E$11	70182547000	D
ABS	70182570000	C
C$62	70180884000	E
AMOD	70182572000	C
L$66	70180854000	C
AINT	70182571000	C
N$66	70180856000	C
DIM	70182573000	C
SIGN	70182574000	C
IFIX	70182553000	C
FLOAT	70180062000	C
C$12	70182575000	C
C$21	70182558000	E
LOC	70181962000	B
C$81	70180882000	C
ISTORE	70181982000	B
N$33	70180090000	C
IFETCH	70181983000	B
IABS	70182552000	C
MOD	70182555000	D
SUB$	70185150000	A
IDIM	70182556000	C
A$22	70182536000	F
M$22	70182537000	E
ISIGN	70182557000	C
L$22	70182534000	D
H$22	70182535000	D
N$22	70180097000	C
SLITE	70182599000	E
M$11	70180035000	D
D$11	70182546000	D
OVERFL	70180894000	C

FSAT	70180071000	D
L$33	70180065000	C
ARGS	70180072000	C
AC1	70180717000	C

PAPER TAPE 70185014000A (FTNLB2H) FOR HIGH-SPEED ARITHMETIC OPTION

This tape consists of the concatenation of individual object paper tapes of the following programs in the order listed (each object has been assembled via the DAP/700 Macro Assembler and may be linked to a calling program by the Linkage Editor):

NAME:	DOC NO.	REV
A$21	70185250000	A
D$21	70185251000	A
M$21	70185252000	A
S$21	70185253000	A
IAND	70185266000	A
IXOR	70185267000	A
ITEST	70185046000	A
ICLR	70185047000	A
ISET	70185048000	A
ISHFT	70185268000	A
IOR	70185269000	A
NOT	70185284000	A
I$GE	70185270000	A
I$GT	70185271000	A
I$LE	70185272000	A
I$NE	70185273000	A
I$LT	70185274000	A
I$EQ	70185275000	A
R$EQ	70185278000	A
R$GE	70185279000	A
R$GT	70185280000	A
R$LE	70185281000	A
R$LT	70185282000	A
R$NE	70185283000	A
A$66X	70180979000	C
H$66	70180855000	C
C$26	70180857000	C
MAX0	70182548000	D
MAX1	70182549000	D
MIN0	70180649000	B
MIN1	70182551000	D
TANH	70182565000	D
SQRTX	70180681000	C
SIN,COS	70182563000	D
ATAN	70182564000	D
E$21	70182562000	C
E$22	70180045000	C
ALOGX	70180682000	C
EXP	70182561000	D
E$11X	70180684000	B
ABS	70182570000	C
C$62	70180884000	E
AMOD	70182572000	C
L$66	70180854000	C
AINT	70182571000	C
N$66	70180856000	C
DIM	70182573000	C
SIGN	70182574000	C
IFIX	70182553000	C
FLOAT	70180062000	C

```
C$12        70182575000 C
C$21        70182558000 E
LOC         70181962000 B
C$81        70180882000 C
ISTORE      70181982000 B
N$33        70180090000 C
IFETCH      70181983000 B
IABS        70182552000 C
MOD         70182555000 D
SUB$        70185150000 A
IDIM        70182556000 C
A$22X       70181805000 B
M$22X       70181806000 B
D$22X       70181804000 B
ISIGN       70182557000 C
L$22        70182534000 D
H$22        70182535000 D
N$22        70180097000 C
SLITE       70182599000 E
M$11X       70180685000 B
D$11X       70180686000 B
OVERFL      70180894000 C
FSAT        70180071000 D
L$33        70180065000 C
ARG$        70180072000 C
AC1         70180717000 C
```

APPENDIX B
MATHEMATICAL ROUTINES

Function	Routine
Complex	
Absolute value	CABS
Add	A$55
Add integer argument	A$51
Add real argument	A$52
Conjugate	CONJG
Convert imaginary part to real	AIMAG
Convert to integer	C$51
Cosine	CCOS
Divide	D$55
Divide by integer argument	D$61
Divide by real argument	D$62
Exponential, base e	CEXP
Load	L$55
Logarithm, base e	CLOG
Multiply	M$55
Multiply by integer argument	M$51
Multiply by real argument	M$52
Negate	N$55
Raise to integer power	E$51
Raise to real power	E$52
Raise to complex power	E$55
Sine	CSIN
Square root	CSQRT
Store (hold)	H$55
Subtract	S$55
Subtract integer argument	S$51
Subtract single-precision argument	S$52

Function	Routine
Double-Precision	
Absolute value	DABS
Add	A$66
Add integer argument	A$61
Add single-precision argument	A$62
Add integer to exponent	A$81
Arctangent, principal value	DATAN
Arctangent, X/Y	DATAN2
Clear (zero) exponent	Z$80
Convert exponent to integer	C$81
Convert to integer	C$61
Convert to single-precision	C$62
Cosine	DCOS
Divide	D$66
Divide by integer argument	D$61
Divide by real argument	D$62
Exponential, base e	DEXP
Load	L$66
Logarithm, base e	DLOG
Logarithm, base 2	DLOG2
Logarithm, base 10	DLOG10
Maximum value	DMAX1
Minimum value	DMIN1
Multiply	M$66
Multiply by integer argument	M$61
Multiply by real argument	M$62
Negate	N$66
Raise to double-precision power	E$66
Raise to integer power	E$61
Raise to real power	E$62
Remainder	DMOD
Sine	DSIN
Square root	DSQRT
Store (hold)	H$66
Subtract	S$66
Subtract integer argument	S$61
Subtract real argument	S$62
Transfer sign of second argument to first	DSIGN

Function	Routine
Truncate fractional bits	DINT
Truncate fractional bits and convert to integer	IDINT
Real	
Absolute value	ABS
Add	A$22
Add integer argument	A$21
Arctangent, principal value	ATAN
Arctangent, X/Y	ATAN2
Convert pair to complex	CMPLX
Convert to complex format	C$25
Convert to double-precision	C$26
Convert to integer	C$21
Divide	D$22
Divide by integer argument	D$21
Exponential, base e	EXP
Hyperbolic tangent	TANH
Load	L$22
Logarithm, base e	ALOG
Logarithm, base 10	ALOG10
Maximum integer value	MAX1
Maximum value	AMAX1
Minimum integer value	MIN1
Minimum value	AMIN1
Multiply	M$22
Multiply by integer argument	M$21
Positive difference	DIM
Raise to double-precision power	E$26
Raise to integer power	E$21
Raise to real power	E$22
Remainder	AMOD
Sine, cosine	SIN, COS
Square root	SQRT
Store (hold)	H$22
Subtract	S$22
Subtract integer argument	S$21
Transfer sign of second argument to first	SIGN
Truncate fractional bits	AINT

Truncate fracional bits and convert to integer	IFIX, INT
TWOs complement	N$22

Integer

Absolute value	IABS
Convert to complex	C$15
Convert to double-precision	C$16
Convert to real (Fortran-generated)	FLOAT
Convert to real	C$12
Divide	D$11
Logically AND integer	IAND
Logically EXCLUSIVE OR integer	IXOR
Logically OR integer	IOR
Maximum value	AMAX0
Maximum integer value	MAX0
Minimum value	AMIN0
Minimum integer value	MIN0
Multiply	M$11
Positive difference	IDIM
Raise to integer power	E$11
Remainder	MOD
Shift integer	ISHFT
Transfer sign of second argument to first	ISIGN

Logical

Complement	N$33
OR with A-register	L$33
Relationals	
Equal to	C$EQ I$EQ R$EQ
Greater than	I$GT R$GT
Greater than or equal to	I$GE R$GE
Less than	I$LT R$LT
Less than or equal to	I$LE R$LE
Not equal to	C$NE I$NE R$NE

APPENDIX C
SUBROUTINE FUNCTIONS

INTRINSIC AND EXTERNAL FUNCTIONS

Mathematical and Trigonometric Functions

Name	Argument Data Type	Result Data Type	Function
SIN	R	R	Sine (radians)
DSIN	D	D	
CSIN	C	C	
COS	R	R	Cosine (radians)
DCOS	D	D	
CCOS	C	C	
ATAN	R	R	Arctangent (radians)
DATAN	D	D	
ATAN2	R	R	
DATAN2	D	D	
TANH	R	R	Hyperbolic tangent (radians)
SQRT	R	R	Square root
DSQRT	D	D	
CSQRT	C	C	
EXP	R	R	Exponential
DEXP	D	D	
CEXP	C	C	
ALOG	R	R	Natural logarithm
DLOG	D	D	
CLOG	C	C	
ALOG10	R	R	Common logarithm
DLOG2	D	D	
DLOG10	D	D	
ABS	R	R	Absolute value
IABS	I	I	
DABS	D	D	
CABS	C	R	
AMOD	R	R	Remainder
MOD	I	I	
DMOD	D	D	
AINT	R	R	Truncate fractional bits
DINT	D	I	
IDINT	D	I	
IFIX	R	I	
INT	R	I	

Name	Argument Data Type	Result Data Type	Function
AMAX0	I	R	Choose largest argument
AMAX1	R	R	
MAX0	I	I	
MAX1	R	I	
DMAX1	D	D	
AMIN0	I	R	Choose smallest argument
AMIN1	R	R	
DMIN1	D	D	
MIN0	I	I	
MIN1	R	I	
FLOAT	I	R	Change data type or argument
AIMAG	C	R	
DBLE	R	D	
CMPLX	C	R	
REAL	C	R	
SNGL	R	D	
SIGN	R	R	Value of first argument, sign of second
DSIGN	D	D	
ISIGN	I	I	
DIM	R	R	Positive difference
IDIM	I	I	
CONJG	C	C	Complex conjugate

Bit String Operations

Name	Argument Data Type	Result Data Type	Function
IAND	I	I	Logical AND
ICLR	I	I	Clear a specified bit
IOR	I	I	Logical OR
ISET	I	I	Set a specified bit
ITEST	I	I	Test specified bit
IXOR	I	I	Logical EXCLUSIVE OR
NOT	I	I	ONEs complement

SPECIAL SUBROUTINES FOR FORTRAN USE

Name	Argument Data Type	Result Data Type	Function
IFETCH (I)			Get contents of location I
ISHFT	I	I	Shift integer in A-register
ISTORE(I, J)			Store value of J in location I
LOC			Find address of argument
OVERFL			Check for error condition
SLITE SLITET SSWTCH			Set and reset sense lights or switches

COMPILER SUPPORT SUBROUTINES

Conversion Routines

Name	Argument Data Type	Result Data Type	Function
C$12	I	R	Convert integer to real
C$15	I	C	Convert integer to complex
C$16	I	D	Convert integer to double-precision
C$21	R	I	Convert real to integer
C$25	R	C	Convert real to complex
C$26	R	D	Convert real to double-precision
C$51	C	I	Convert complex to integer
C$52	C	R	Convert complex to real
C$61	D	I	Convert double-precision to integer
C$62	D	R	Convert double-precision to real
C$81	D	D	Convert exponent of double-precision number to integer

Logical Relationals

Name	Argument Data Type	Result Data Type	Function
C$EQ	C	L	Equal to
I$EQ	I	L	
R$EQ	R	L	
I$GE	I	L	Greater than or equal to
R$GE	R	L	
I$GT	I	L	Greater than
R$GT	R	L	
I$LE	I	L	Less than or equal to
R$LE	R	L	
I$LT	I	L	Less than
R$LT	R	L	
C$NE	C	L	Not equal to
I$NE	I	L	
R$NE	R	L	

Arithmetic Routines

Name	Function	Name	Function
A$21	R = R+I	D$51	C = C/I
A$22	R = R+R	D$52	C = C/R
A$51	C = C+I	D$55	C = C/C
A$52	C = C+R	D$61	D = D/I
A$55	C = C+C	D$62	D = D/R
A$61	D = D+I	D$66	D = D/D
A$62	D = D+R	E$11	I = I**I
A$66	D = D+D	E$21	R = R**I
A$81	D = D*(2**I)	E$22	R = R**R

Name	Function	Name	Function
D$11	I = I/I	E$26	D = R**D
D$21	R = R/I	E$51	C = C**I
D$22	R = R/R	E$52	C = C**R
E$55	C = C**C	N$22	R = -R
E$61	D = D**I	N$33	L = -L
E$62	D = D**R	N$55	C = -C
E$66	D = D**D	N$66	D = -D
M$11	I = I*I	S$21	R = R-I
M$21	R = R*I	S$22	R = R-R
M$22	R = R*R	S$51	C = C-I
M$51	C = C*I	S$52	C = C-R
M$52	C = C*R	S$55	C = C-C
M$55	C = C*C	S$61	D = D-I
M$61	D = D*I	S$62	D = D-R
M$62	D = D*R	S$66	D = D-D
M$66	D = D*D	Z$80	Replace binary exponent with zero

Miscellaneous Routines

Name	Function
AC1	Pseudo accumulators
ARG$	Convert indirect address to direct address
F$AT	Transfer variable number of arguments
F$ER	Print error messages
H$22	Store real number in memory
H$55	Store complex number in memory
H$66	Store double-precision number in memory
L$22	Load real number into A- and B-registers
L$55	Load complex number into complex accumulator
L$66	Load double-precision number into double-precision accumulator
L$33	INCLUSIVE OR with A-register
SUB$	Calculate address of array element

APPENDIX D

LIBRARY INDEX

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
A$21	A$21	F$AT	1	20	2	5-2
		H$22	1			
		FLOAT	1			
		A$22	1			
A$22	A$22	ARG$	1	150	2S	5-3
	S$22	N$22	1			
		F$ER	1			
A$22X	A$22	N$22	1	140	2H	5-4
	S$22	F$ER	1			
A$51	A$51	F$AT	1	20	1	5-5
		H$55	1			
		C$12	1			
		H$22	1			
		L$55	1			
		A$52	1			
A$52	A$52	F$AT	1	20	1	5-6
		H$55	1			
		L$22	1			
		A$22	1			
		H$22	1			
		L$55	1			
A$55	A$55	F$AT	1	60	1	5-7
		H$55	1			
		SUB$	4			
		L$22	2			
		A$22	2			
		H$22	2			
		L$55	1			
A$61	A$61	F$AT	1	20	1	5-8
		H$66	1			
		C$12	1			
		H$22	1			
		L$66	1			
		A$62	1			
A$62	A$62	F$AT	1	20	1	5-9
		H$66	1			
		DBLE	1			
		A$66	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number	Page
A$66	A$66	N$66	11	580		5-10
	S$66	F$ER	3			
	M$66	H$66	1			
	D$66	L$66	1			
		ARG$	1			
		AC1	1			
		AC2	1			
		AC3	1			
A$66X	A$66	N$66	11	530	1	5-12
	A$66X	F$ER	3			
	S$66	H$66	1			
	S$66X	L$66	1			
	M$66	ARG$	1			
	M$66X	AC1	1			
	D$66	AC2	1			
	D$66X	AC3	1			
A$81	A$81	N$22	2	70	1	5-13
		F$ER	1			
		AC1	1			
		AC2	1			
ABS	ABS	L$22	1	10	2	4-2
		N$22	1			
AC1	AC1			5	2	5-14
	AC2					
	AC3					
	AC4					
AIMAG	AIMAG	L$55	1	10	1	4-3
		L$22	1			
		AC3	1			
AINT	AINT	L$22	1	30	2	4-4
		N$22	2			
		A$22	1			
		S$22	1			
ALOG	ALOG10	ARG$	1	120	2S	4-5
	ALOG	C$12	1			
		H$22	5			
		L$22	3			
		A$22	6			
		S$22	2			
		D$22	1			
		M$22	7			
		F$ER	1			
ALOGX	ALOG10	ARG$	1	180	2H	4-6
	ALOG	C$12	1			
	ALOGX	A$22	4			
		M$22	4			
		S$22	1			
		F$ER	1			
ALOG10	See ALOG or ALOGX					4-8
AMAX0	See MAX0					4-9

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
AMAX1	See MAX1					4-10
AMIN0	See MIN0					4-11
AMIN1	See MIN1					4-12
AMOD	AMOD	L$22	1	30	2	4-13
		D$22	1			
		AINT	1			
		M$22	1			
		N$22	1			
		A$22	1			
ARG$	ARG$			20	2	5-15
ATAN	ATAN2	ARG$	3	340	2	4-14
	ATAN	D$22	6			
		N$22	7			
		M$22	5			
		A$22	11			
		S$22	2			
ATAN2	See ATAN					4-16
C$12	C$12	A$22	1	30	2	5-16
		N$22	1			
C$15	C$15	C$12	1	4	1	5-17
		C$25	1			
C$16	C$16	C$12	1	5	1	5-18
		C$26	1			
C$21	C$21	N$22	1	30	2	5-19
		A$22	1			
		F$ER	1			
C$25	C$25	H$22	1	20	1	5-20
		CMPLX	1			
C$26	C$26	AC1	1	10	1	5-21
		AC2	1			
		AC3	1			
C$51	C$51	C$52	1	4	1	5-22
		C$21	1			
C$52	C$52	H$55	1	10	1	5-23
		L$22	1			
C$61	C$61	C$62	1	4	1	5-24
		C$21	1			
C$62	C$62	L$22	1	20	1	5-25
	SNGL	N$66	1			
		N$22	1			
		L$66	1			
		AC1	1			
		AC2	1			
C$81	C$81	AC1	1	10	2	5-26
C$EQ	C$EQ	AC1	1	20	2	5-27
		AC3	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number	Page
C$NE	C$NE	AC1	1	10	2	5-28
		AC3	1			
CABS	CABS	F$AT	1	40	1	4-17
		SUB$	2			
		L$22	2			
		M$22	2			
		H$22	2			
		A$22	1			
		SQRT	1			
CCOS	CCOS	F$AT	1	40	1	4-18
		L$55	1			
		A$55	1			
		H$55	1			
		CSIN	1			
CEXP	CEXP	F$AT	1	60	1	4-19
		SUB$	7			
		EXP	1			
		H$22	3			
		COS	1			
		M$22	2			
		SIN	1			
		L$55	1			
CLOG	CLOG	F$AT	1	90	1	4-20
		SUB$	6			
		L$22	3			
		M$22	3			
		H$22	5			
		A$22	1			
		ALOG	1			
		ATAN2	1			
		L$55	1			
CMPLX	CMPLX	F$AT	1	40	1	4-21
		SUB$	2			
		L$22	2			
		H$22	2			
		L$55	1			
CONJG	CONJG	F$AT	1	40	1	4-22
		SUB$	4			
		L$22	2			
		H$22	2			
		N$22	1			
CSIN	CSIN	F$AT	1	90	1	4-24
		SUB$	5			
		EXP	1			
		H$22	6			
		L$22	3			
		D$22	1			
		A$22	1			
		SIN	1			
		M$22	4			
		S$22	1			
		COS	1			
		L$55	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
CSQRT	CSQRT	F$AT	1	90	1	4-25
		SUB$	7			
		CABS	1			
		H$22	8			
		ABS	1			
		A$22	1			
		M$22	2			
		SQRT	1			
		L$22	6			
		D$22	1			
		L$55	1			
D$11	D$11	ARG$	1	80	2S	5-29
		F$ER	1			
D$11X	D$11	ARG$	1	40	2H	5-30
	D$11X	F$ER	1			
D$21	D$21	F$AT	1	20	2	5-31
		H$22	2			
		FLOAT	1			
		L$22	1			
		D$22	1			
D$22	See M$22					5-32
D$22X	D$22	N$22	3	110	2H	5-33
		F$ER	2			
D$51	D$51	F$AT	1	20	1	5-34
		H$55	1			
		C$12	1			
		H$22	1			
		L$55	1			
		D$52	1			
D$52	D$52	F$AT	1	50	1	5-35
		H$55	1			
		SUB$	2			
		L$22	2			
		D$22	2			
		H$22	2			
		L$55	1			
D$55	D$55	F$AT	1	140	1	5-36
		H$55	1			
		SUB$	12			
		L$22	8			
		M$22	6			
		H$22	8			
		A$22	2			
		D$22	2			
		S$22	1			
		N$22	1			
		L$55	1			
D$61	D$61	F$AT	1	20	1	5-37
		H$66	1			
		C$12	1			
		H$22	1			
		L$66	1			
		D$62	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
D$62	D$62	F$AT	1	20	1	5-38
		H$66	2			
		DBLE	1			
		L$66	1			
		D$66	1			
D$66	See A$66					5-39
D$66X	See A$66X					5-12
DABS	DABS	F$AT	1	10	1	4-26
		L$66	1			
		N$66	1			
DATAN	DATAN	F$AT	1	180	1	4-27
		DABS	1			
		H$66	9			
		C$81	1			
		L$66	13			
		A$66	10			
		N$66	3			
		D$66	2			
		M$66	9			
DATAN2	DATAN2	F$AT	1	70	1	4-28
		L$66	9			
		H$66	3			
		F$ER	1			
		D$66	1			
		DATAN	1			
		S$66	1			
		A$66	1			
DBLE	DBLE	F$AT	1	20	1	4-29
		L$22	1			
		C$26	1			
DCOS	DCOS	F$AT	1	20	1	4-30
		L$66	1			
		A$66	1			
		H$66	1			
		DSIN	1			
DEXP	DEXP	F$AT	1	160	1	4-31
		L$66	12			
		M$66	8			
		H$66	11			
		C$61	1			
		C$16	1			
		N$66	1			
		A$66	8			
		S$66	3			
		D$66	1			
		A$81	1			
DIM	DIM	L$22	1	20	2	4-32
		S$22	1			
DINT	DINT	L$66	1	20	1	4-33
		N$66	2			
		A$66	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
		S$66	1			
		AC1	1			
DIV$	See M$22					5-69
DLOG	DLOG	F$AT	1	10	1	4-34
		DLOG2	1			
		M$66	1			
DLOG2	DLOG2	F$AT	1	100	1	4-35
		L$66	5			
		F$ER	1			
		C$81	1			
		C$16	1			
		H$66	6			
		Z$80	1			
		A$66	6			
		S$66	2			
		D$66	1			
		M$66	6			
DLOG10	DLOG10	F$AT	1	10	1	4-36
		DLOG2	1			
		M$66	1			
DMAX1	DMAX1	L$66	3	40	1	4-37
		H$66	2			
		S$66	1			
DMIN1	DMIN1	L$66	3	40	1	4-38
		H$66	2			
		S$66	1			
DMOD	DMOD	F$AT	1	20	1	4-39
		L$66	1			
		D$66	1			
		H$66	1			
		DINT	1			
		M$66	1			
		S$66	1			
		N$66	1			
DSIGN	DSIGN	F$AT	1	20	1	4-40
		L$66	3			
		N$66	1			
DSIN	DSIN	F$AT	1	130	1	4-41
		DABS	1			
		M$66	9			
		H$66	5			
		C$61	1			
		C$16	1			
		N$66	3			
		A$66	7			
		MOD	1			
		L$66	8			
		S$66	2			
DSQRT	DSQRT	F$AT	1	40	1	4-42
		L$66	2			
		C$62	1			
		H$22	1			
		SQRT	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
		C$26	1			
		H$66	1			
		D$66	1			
		A$66	1			
		A$81	1			
E$11	E$11	ARG$	1	100	2S	5-40
		M$11	2			
		FSER	1			
E$11X	E$11	ARG$	1	110	2H	5-41
	E$11X	F$ER	1			
E$21	E$21	ARG$	1	50	2	5-42
		M$22	1			
		D$22	1			
E$22	E$22	ARG$	1	30	2	5-43
		ALOG	1			
		M$22	1			
		EXP	1			
E$26	E$26	F$AT	1	30	1	5-44
		C$26	1			
		H$66	2			
		DLOG	1			
		M$66	1			
		DEXP	1			
E$51	E$51	F$AT	1	60	1	5-45
		H$55	3			
		LABS	1			
		L$55	4			
		M$55	1			
		D$55	1			
E$52	E$52	F$AT	1	30	1	5-46
		H$55	2			
		L$22	1			
		C$25	1			
		L$55	1			
		E$55	1			
E$55	E$55	F$AT	1	40	1	5-47
		H$55	2			
		CABS	1			
		F$ER	1			
		CLOG	1			
		C$25	1			
		M$55	1			
		CEXP	1			
E$61	E$61	F$AT	1	70	1	5-48
		H$66	5			
		L$66	5			
		D$66	1			
		D$11	2			
		M$11	1			
		M$66	2			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
E$62	E$62	F$AT	1	30	1	5-49
		H$66	2			
		DLOG	1			
		M$62	1			
		DEXP	1			
E$66	E$66	F$AT	1	30	1	5-50
		H$66	2			
		DLOG	1			
		M$66	1			
		DEXP	1			
EXP	EXP	ARG$	1	230	2	4-43
		N$22	2			
		M$22	6			
		S$22	3			
		A$22	2			
		D$22	2			
		F$ER	1			
F$AT	F$AT			50	2	5-51
FLOAT	FLOAT	C$12	1	10	2	4-44
H$22	H$22	ARG$	1	10	2	5-53
H$55	H$55	ARG$	1	20	1	5-54
		AC1	1			
		AC2	1			
		AC3	1			
		AC4	1			
H$66	H$66	ARG$	1	20	2	5-55
		AC1	1			
		AC2	1			
		AC3	1			
I$EQ	I$EQ			5	2	5-56
I$GE	I$GE			10	1	5-57
I$GT	I$GT			10	2	5-58
I$LE	I$LE			10	2	5-59
I$LT	I$LT			10	2	5-60
I$NE	I$NE			10	2	5-61
IABS	IABS			10	2	4-45
IAND	IAND			10	2	4-46
ICLR	ICLR	F$AT	1	10	2	4-47
		ISHFT	1			
IDIM	IDIM			20	2	4-48
IDINT	See IFIX					4-49
IEOR	See IXOR					4-59
IFETCH	IFETCH	ARG$	1	10	2	4-50
IFIX	IDINT	L$22	1	10	2	4-51
	INT	C$21	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number	Page
	IFIX					
INT	See IFIX					4-52
IOR	IOR			20	2	4-53
ISET	ISET	F$AT	1	10	2	4-54
		ISHFT	1			
ISHFT	ISHFT			30	2	4-55
ISIGN	ISIGN			20	2	4-56
ISTORE	ISTORE	F$AT	1	10	2	4-57
ITEST	ITEST	F$AT	1	10	2	4-58
		ISHFT	1			
IXOR	IEOR			10	2	4-59
	IXOR					
L$22	REAL	ARG$	1	10	2	5-62
	L$22					
L$33	L$33			10	2	5-63
L$55	L$55	ARG$	1	20	1	5-64
		AC1	1			
		AC2	1			
		AC3	1			
		AC4	1			
L$66	L$66	ARG$	1	20	2	5-65
		AC1	1			
		AC2	1			
		AC3	1			
LOC	LOC			10	2	4-60
M$11	M$11	ARG$	1	110	2S	5-66
		F$ER	1			
M$11X	M$11	ARG$	1	50	2H	5-67
	M$11X	F$ER	1			
M$21	M$21	F$AT	1	20	2	5-68
		H$22	1			
		FLOAT	1			
		M$22	1			
M$22	M$22	N$22	5	330	2S	5-69
	D$22	ARG$	2			
	DIV$	F$ER	3			
M$22X	M$22	F$ER	1	130	2H	5-70
M$51	M$51	F$AT	1	20	1	5-71
		H$55	1			
		C$12	1			
		H$22	1			
		L$55	1			
		M$52	1			
M$52	M$52	F$AT	1	50	1	5-72
		H$55	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number	Page
		SUB$	2			
		L$22	2			
		M$22	2			
		H$22	2			
		L$55	1			
M$55	M$55	F$AT	1	110	1	5-73
		H$55	1			
		SUB$	10			
		L$22	4			
		M$22	4			
		H$22	4			
		S$22	1			
		N$22	1			
		A$22	1			
		L$55	1			
M$61	M$61	F$AT	1	20	1	5-74
		H$66	1			
		C$12	1			
		H$22	1			
		L$66	1			
		M$62	1			
M$62	M$62	F$AT	1	20	1	5-75
		H$66	1			
		DBLE	1			
		M$66	1			
M$66	See A$66					5-76
M$66X	See A$66X					5-12
MAX0	AMAX0 MAX0	FLOAT	1	40	2	4-61
MAX1	AMAX1 MAX1	L$22	2	50	2	4-62
		H$22	2			
		S$22	1			
		IFIX	1			
MIN0	AMIN0 MIN0	FLOAT	1	30	2	4-63
MIN1	AMIN1 MIN1	L$22	2	50	2	4-64
		H$22	2			
		S$22	1			
		IFIX	1			
MOD	MOD	D$11	1	20	2	4-65
		M$11	1			
N$22	N$22			10	2	5-77
N$33	N$33			10	2	5-78
N$55	N$55	H$55	1	30	1	5-79
		SUB$	2			
		L$22	2			
		N$22	2			
		H$22	2			
		L$55	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
N$66	N$66	AC1	1	30	2	5-80
		AC2	1			
		AC3	1			
NOT	NOT			10	2	4-66
OVERFL	OVERFL	AC5	1	20	2	4-67
R$EQ	R$EQ			10	2	5-81
R$GE	R$GE			10	2	5-82
R$GT	R$GT			10	2	5-83
R$LE	R$LE			10	2	5-84
R$LT	R$LT			10	2	5-85
R$NE	R$NE			5	2	5-86
S$21	S$21	F$AT	1	20	2	5-87
		H$22	1			
		FLOAT	1			
		S$22	1			
		N$22	1			
S$22	See A$22					5-88
S$51	S$51	F$AT	1	40	1	5-89
		H$55	1			
		C$12	1			
		H$22	1			
		L$55	1			
		S$52	1			
S$52	S$52	F$AT	1	30	1	5-90
		H$55	1			
		L$22	1			
		S$22	1			
		H$22	1			
		L$55	1			
S$55	S$55	F$AT	1	40	1	5-91
		H$55	1			
		SUB$	4			
		L$22	2			
		S$22	2			
		N$22	2			
		H$22	2			
		L$55	1			
S$61	S$61	F$AT	1	20	1	5-92
		H$66	1			
		C$12	1			
		H$22	1			
		L$66	1			
		S$62	1			
S$62	S$62	F$AT	1	20	1	5-93
		H$66	1			
		DBLE	1			
		S$66	1			
		N$66	1			

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number	Page
S$66	See A$66					5-94
S$66X	See A$66X					5-12
SIGN	SIGN	L$22	2	20	2	4-68
		N$22	1			
SIN	COS	ARG$	1	190	2	4-69
	SIN	N$22	2			
		M$22	7			
		S$22	1			
		A$22	4			
SIZ$	See SUB$					5-95
SLITE	SLITE	ARG$	3	70	2	4-70
	SLITET	L$33	1			
	SSWTCH					
SLITET	See SLITE					4-71
SNGL	See C$62					5-96
SQRT	SQRT	ARG$	1	70	2S	4-72
		DIV$	1			
		D$22	1			
SQRTX	SQRT	ARG$	1	80	2H	4-73
	SQRTX	D$22	1			
		A$22	1			
		F$ER	1			
SSWTCH	See SLITE					4-74
SUB$	SUB$	M$11	3	130	2	5-97
		F$ER	1			
TANH	TANH	L$22	1	60	2	4-75
		EXP	1			
		A$22	2			
		H$22	1			
		D$22	1			
Z$80	Z$80	AC1	1	20	1	5-99

APPENDIX E
ERROR MESSAGES

Error Message	Condition	Subroutine
AD	Over/underflow in double-precision	A$66, S$66, A$66X, S$66X
AO	Array element referenced is outside array boundary	SUB$
CE	Absolute magnitude of complex number is zero	E$55
DL	Negative or zero argument	DLOG, DLOG10, DLOG2
DT	Both arguments are zero	DATAN2
DZ	Division by zero	D$22, D$22X
EQ	Exponential overflow adding integer to double-precision exponent	A$81
EX	Exponential overflow during exponentiation	EXP
II	First argument zero, second argument negative I > 2 and J ≥ 15, or I ≤ -2 and J ≥ 15	E$11, E$11X
IM	Over/underflow during integer multiplication	M$11, M$11X
IZ	Integer division by zero or -32,768/-1	D$11, D$11X
LG	Log of negative or zero argument	ALOG, ALOG10, ALOGX
MD	Double-precision multiplication or division over/underflow	D$66, M$66, D$66X, M$66X
PZ	Double-precision division by zero	D$66, D$66X
RI	Integer too large when converted from real to integer	C$21
SA	Arithmetic overflow (result ≥ 2**127)	A$22, A$22X
SD	Divisor unnormalized	D$22
SM	Arithmetic overflow during multiplication or division	M$22, M$22X, D$22X
SQ	Negative argument	SQRT, SQRTX

COMPUTER GENERATED INDEX

Honeywell

HONEYWELL INFORMATION SYSTEMS

AG16, Rev. 0