Honeywell

FORTRAN MATH LIBRARY

SERIES 16

SOFTWARE

Honeywell

FORTRAN MATH LIBRARY

SERIES 16

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:

December 1973

ORDER NUMBER:

AM74, Rev. 0

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-16 Mod 2 assembly 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. Section VI presents run-time and control subroutines. There are five appendices, included to facilitate access to the library.

Additional information may be obtained from the following manuals:

<u>DAP-16 and DAP-16 Mod 2 Assembly Language</u>, Order Number BY09.
<u>316/516 Programmers' Reference Manual</u>, Order Number BX47.
<u>Series 16 FORTRAN IV</u>, Order Number BX32.
<u>Series 16 Equipment Operators' Manual</u>, Order Number BX48.

> The FORTRAN Math Library consists of coded programs designed to extend the power of Series 16 in the area of program preparation and maintenance. They are supported by comprehensive documentation and training; periodic program maintenance is furnished for the current version of the programs, in accordance with established Honeywell specifications, provided they are not modified by the user.

File No.: 1A23

AM74

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

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-16 Mod 2 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 four magnetic tapes and eight paper tapes available. The first two magnetic tapes are source tapes and contain the sources for the Statistical Library, FORTRAN Library, and Fixed Point Math Library. The third magnetic tape contains the objects for the software version of the three libraries; the fourth tape contains the objects for the hardware version of the three libraries.

---

[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.

The eight object paper tapes contain the FORTRAN Library and are labeled:

LTCF1	Tape 1 of 6	
LTCF2	Tape 2 of 6	
LTCF3S	Tape 3 of 6	Software Version
LTCF3H	Tape 3 of 6	Hardware Version
LTCF4	Tape 4 of 6	
LTCF5S	Tape 5 of 6	Software Version
LTCF5H	Tape 5 of 6	Hardware Version
LTCF6	Tape 6 of 6	

The tapes are order dependent, as many of the subroutines call other subroutines which appear later in the library. The digits 1 through 6 on the label indicate the loading order.

Appendix B lists the math routines by argument type.

Appendix C lists the library subroutines by function.

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 six rolls 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, the first two paper tapes are not required.

Each paper tape has been assembled via the DAP-16 Mod 2 assembly language and should be loaded by the Series 16 Loader, LDR-APM. Refer to the Series 16 Equipment Operators' Manual, Order Number BX48, for information concerning loading object paper tapes.

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
```

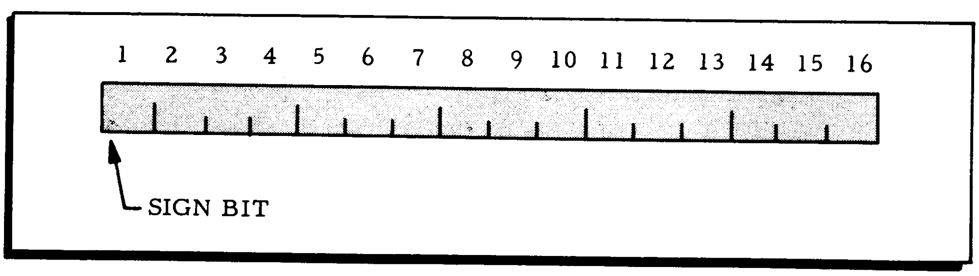

Figure 2-1. Format of Integer

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.

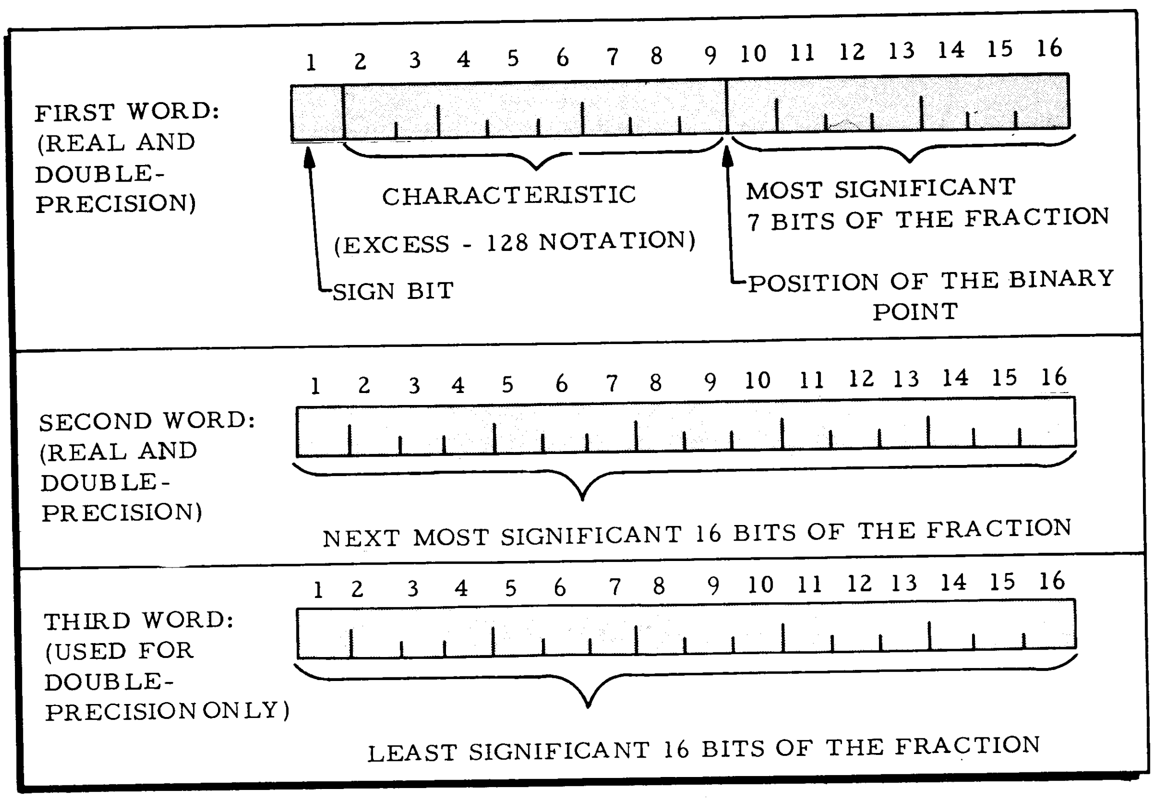

Figure 2-2. Format of Real and Double-Precision Numbers

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-16 MOD 2 PROGRAMMING INFORMATION

LIBRARY CALLS

The DAP-16 Mod 2 calling sequences for entry into the subroutines are shown in the descriptions in Sections IV, V and VI. 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 are two or more arguments, the final address is followed bv 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$AT (Argument Transfer; refer to Section VI). This call immediately follows the entry point and in turn is followed by a word containing a count of the number of arguments as defined in the definition statement, followed by that number of words. The F$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-16 Mod 2 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.

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-16 Mod 2 code:

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

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)
```

To perform the same function as the statement Z = X + Y and to generate the same code.

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

CALL <u>ROUTINE NAME</u> (ARG1)

CALL <u>ROUTINE NAME</u> (ARG1, ARG2)

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

EXAMPLES OF 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 and special FORTRAN subroutines, arranged in alphabetical 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 $\leq$ |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, H$22, L$22, D$22

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 the product

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

such that F*K = 1+G, where G is positive

$$\log_2 X = \frac{\ln(F*K/K)}{\ln(2)} + B$$

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

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

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

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

Since ln(2) = .69314718,

$$\log_2 X = 1.442695141\,G - .7213475704\,G^2 + .4808984995\,G^3$$

$$\cdots \frac{\ln(K)}{\ln(2)} + B$$

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$22X, M$22X, S$22X, F$ER

ALOG10

Purpose To calculate the common (base 10) logarithm.

See ALOG or ALOGX.

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

See MAX0.

AMAX1

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

See MAX1.

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

See MIN0.

AMIN1

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

See MIN1.

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≤N≤10**75 into three intervals:

If N ≤10**(-B), ATAN(N) = N

If N >10**10, ATAN(N) = pi/2

If 10** (-8) < N ≤ 10**10,

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

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

If N ≤1/2, Z = N and base angle = 0

If N≤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

<u>Purpose</u> To calculate the arctangent as the quotient of two real numbers.

See ATAN.

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

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 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

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) = R+I(ϕ)

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

ϕ = (TAN**-1)(Y/X) = ϕ + 2Kpi

where K = 0, ± 1, ± 2, ...

A particular value for ϕ is chosen such that -pi≤ϕ≤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$55, SUB$

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

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 calculate the cosine of a real number expressed in radians.

See SIN.

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 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

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 o 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 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

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 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

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 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

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 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 ≤|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

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 $1/2 \leq F^1 < 1$. $\log_2(X) = \log_2(F^1)$ + 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 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^1*(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

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 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

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 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

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 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

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 the made to achieve full double-precision accuracy.
Data Type of Arguments and Results	This square root function of a double-precision argument result 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 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

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 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.

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

See IFIX.

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 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 uses the characteristic to scale the value to an integer.
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

INT

Purpose

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

See IFIX.

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.

ISTORE

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

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.

MAXO

<u>Purpose</u>

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.

<u>DAP Calling Sequence</u>

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

<u>FORTRAN Reference</u>

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

<u>Method</u>

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.

<u>Data Type of Arguments and Results</u>

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.

<u>Other Routines Used</u>

FLOAT

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

MINO

<u>Purpose</u>

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.

<u>DAP Calling Sequence</u>

```
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)
```

<u>FORTRAN Reference</u>

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

<u>Method</u>

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.

<u>Data Types of Arguments and Results</u>

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.

<u>Other Routines Used</u>

FLOAT

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	See below
FORTRAN Reference	MIN1(R,R,...,R) or AMIN1(R,R,...,R)
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

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)
```

MOD

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

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

REAL

Purpose	To load the real portion of a complex number into the A- and B-register.
DAP Calling Sequence	CALL REAL DAC ARG1 (a complex number) (Return)
FORTRAN Reference	REAL(C)
Method	This subroutine calls ARG$ to place the complex argument, ARG1, into the index register. The real portion, i.e., the first two words, of the complex argument is then loaded into the A- and B-registers.
Data Type of Arguments and Results	This function of a complex number results in a real number.
Other Routines Used	ARG$

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

SIN

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, wnere 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

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

SLITET

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

See SLITE.

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

SQRTX

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 $1/4 \le e < 1$. An initial approximation to the square root of ARG1 is chosen as follows:

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

$$ARG1 = 9/16(F) + 7/16 \text{ if } e \ge 1/2$$

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$22X, A$22X, F$ER

SSWTCH

<u>Purpose</u> To set or reset the pseudo sense switches.

See SLITE.

TANH

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$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

Purpose

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

DAP Calling Sequence

```
CALL  A$22X    (A$22, S$22 or S$22X)
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

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)
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$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

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 by, 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

< implicit double-precision argument> {+ − * /} < double-precision argument> → < double-precision result>

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

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 by, 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> {+, -, *, /} < 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

A$81

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

(AC2, AC3, AC4, AC5)

Purpose	To assign 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.

ARG$

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.

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

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$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 (Return) or CALL SNGL / DAC ARG1 (a double-precision number) / (Return)
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 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 two real numbers. (This subroutine has a high-speed version, D$22X.)

See M$22.

D$22X

Purpose

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

DAP Calling Sequence

```
CALL    D$22X    (or 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

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 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 (ARG2) 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

D$66

<u>Purpose</u> To divide normalized double-precision numbers.

See A$66.

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

E$11X

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 (or E$11) 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

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

E$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

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

E$51

<u>Purpose</u>

To calculate the value of a complex quantity raised to an integer power.

<u>DAP Calling Sequence</u>

```
CALL    E$51
DAC     ARG1      (the integer exponent)
(Return)
```

<u>Method</u>

The number in the complex accumulator is multiplied by itself ARG1-1 times.

<u>Data Type of Arguments and Results</u>

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

<u>Other Routines Used</u>

FAT, H55, IABS, L$55, M$55, D$55

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

E$62

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

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

H$22

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

Purpose	To hold (store) the contents of the complex accumulator in memory.
DAP Calling Sequence	CALL H$55 DAC ARG1 (location in which the contents of the complex accumulator are to be stored) (Return)
Method	The contents of memory at the location specified by the argument address, ARG1, are replaced by the contents of the complex accumulator. The contents of the accumulator remain unchanged.
Data Type of Arguments and Results	This subroutine stores a complex number in the argument address.
Other Routines Used	ARG$, AC1, AC2, AC3, AC4

H$66

Purpose	To hold (store) the contents of the double-precision accumulator in memory.
DAP Calling Sequence	CALL H$66 DAC ARG1 (location in which the contents of the double-precision accumulator are to be stored) (Return)
Method	The contents of memory specified by the argument address, ARG1, are replaced by the contents of the double-precision accumulator. The contents of the accumulator are unchanged.
Data Types of Arguments and Results	This subroutine stores a double-precision number in the argument address.
Other Routines Used	ARG$, AC1, AC2, AC3

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$

L$33

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.

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

L$66

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

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

M$11X

Purpose	To multiply two integers. (This subroutine requires the High-Speed Arithmetic Option.)
DAP Calling Sequence	CALL M$11X (or 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

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)	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.
DAC	ARG2	(multiplier or divisor)	
(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**(B+C)
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, where H(X(1)*Y(1)) is the most significant part of the product X(1)*Y(1) and L(X(1)*Y(1)) is the least significant part of that product.

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 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

M$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

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.

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

<u>Purpose</u>

To subtract real numbers. (This subroutine has a high-speed version, S$22X.)

See A$22.

S$22X

Purpose

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

See A$22X.

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

S$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

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

<u>Purpose</u> To subtract normalized, double-precision numbers.

See A$66.

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

See C$62.

SUB$

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     O          (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(..., (S(N)-1)*D(N-1) +...+ (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

Z$80

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

SECTION VI

RUN-TIME AND CONTROL SUBROUTINES

This section describes the routines which: control input and output by selecting and activating the proper device drivers; provide buffers; and edit and trace all I/O.

F$AR

Purpose

To transfer an array from or to the input or output data list.

DAP Calling Sequence

CALL	F$AR	For use with DAP
OCT	m	Number of words in array
DAC	a	Location of first word
(Return)		
or		
CALL	F$Lx	As used by the compiler
OCT	m	x = 1 for integer
DAC	a	= 2 for real
(Return)		= 3 for logical
		= 5 for complex
		= 6 for double precision

FORTRAN Reference

DIMENSION I(5)
READ (x, f) list or (f = FORTRAN statement number)
WRITE (x, f) list

Method

Whenever the data list requires data from the internal source or data to be stored in the internal source, exit is made from F$10 to the next location in the data list. Elements of the data list may be variables, subscripted variables, or array names. Mode may be integer, real, logical, complex, or double precision. For easier data transmission, all list elements are assumed to be arrays; the mode is determined by appropriate format descriptors. For each item in the list, the three-word calling sequence (above) is generated.

Other Routines Used

F$10, F$CB, F$ER

Purpose	To transfer a variable number of arguments from the calling routine to the called routine.		
DAP Calling Sequence	DAC	**	Entry point
	CALL	F$AT	
	DEC	n	Number of arguments to be transferred
	DAC	ARG1	Address at which first argument is stored
	.		
	.		
	.		
	DAC	ARGn	Address at which last argument is stored
	(Return)		
FORTRAN References	CALL	SUB1	(ARG1, ARG2, ..., ARGn) Where SUB1 is any subroutine and ARG1 through ARGn are any constants, variables, arrays, etc.

Method

When arguments are to be transferred from a calling routine to a subroutine, a call to F$AT is generated by the compiler. The number of arguments specified by the first pseudo-operation following the call are transferred from the calling routine to the subroutine. All levels of indirect addressing are removed before an argument is transferred. ARG1 is the beginning location of the block into which the arguments are to be placed.

Data Type of Arguments

Arguments are direct relative addresses.

F$B5-9

Purpose	To connect the calling program with the magnetic tape rewind routine.		
DAP Calling Sequence	CALL	F$Bx	x = 5, 6, 7, 8, 9 or the previously defined variable n (n = 5, 6, 7, 8, or 9)
FORTRAN Reference	REWIND	x	x = 5, 6, 7, 8, 9 or the variable n
Method	This routine converts the logical magnetic tape unit number (5, 6, 7, 8 or 9) to the corresponding physical magnetic tape unit number (1, 2, 3, 4, or 5) and then calls the REWIND routine to rewind the tape on that unit to the beginning of tape.		
Other Routines Used	C$MR		

Purpose	To close the buffers used for input or output.	
DAP Calling Sequence	CALL F$CB	
FORTRAN Reference	READ (n, m) list READ (n, m) READ (n) list READ (n) WRITE (n, m) list WRITE (n, m) WRITE (n) list	n = device number; m = format statement number
Method	At the end of the data list from a READ or WRITE statement, or when the format statement is exhausted (non-list), a call is issued to F$CB to close the buffer. The address of the buffer was determined by F$IO. F$CB checks for I/O mode and immediately closes the buffer if mode is input. If the buffer is formatted output, F$CB fills the remainder of the buffer with up to 134 spaces. If output is binary, the end of the buffer is filled with up to 120 zeros.	
Other Routines Used	F$IO	

F$D5-9

Purpose	To control the writing of an end-of-file mark on magnetic tape and a STOP code as an end-of-file on paper tape punch.
DAP Calling Sequence	CALL F$D2 — 2 = paper tape punch CALL F$Dx — x = logical tape unit number 5 through 9 CALL F$Dn — n = dummy device number and the A-register contains the value 2, 5, 6, 7, 8, or 9
FORTRAN References	END FILE x — x = 2, 5, 6, 7, 8, 9 or the variable n
Method	This routine converts the logical magnetic tape unit number (5, 6, 7, 8, or 9) to the corresponding physical magnetic tape unit number (1, 2, 3, 4, or 5). It then calls the driver to write an end-of-file mark on the specified magnetic tape. If x is 2, it calls the driver to punch a STOP code on the high-speed paper tape punch.
Other Routines Used	OME, OPS

Purpose	To cause a mnemonic error indicator to be typed on the ASR-33 when an object-time error is encountered in a specified routine.	
DAP Calling Sequence	CALL F$ER DAC ARG1	ARG1 is the address of the indicator to be typed. The routine types the error indicator and halts if Sense Switch 3 is not set. Pressing START after the halt causes the program to continue. If Sense Switch 3 is set, F$ER exits with no typeout and no halt.
FORTRAN References	CALL F$ER (2Hxx)	xx = two ASCII characters to be typed
Method	The location of the object-time error mnemonic indicator is extracted from its relocatable address, AC5. AC5 is then loaded with the error mnemonic indicator and Sense Switch 3 is tested. If the switch is not set, a carriage return and line feed are issued. The error mnemonic is then printed, and the routine halts. If START is pressed at this point, a normal return is made to the calling routine. If Sense Switch 3 is set, return is made immediately to calling routine.	
Data Type of Argument	The argument is the address of any two ASCII characters.	
Other Routines Used	AC5, F$HT	

F$F5-9

Purpose	To control back spacing of a record on magnetic tape.
DAP Calling Sequence	CALL F$Fx x = 5, 6, 7, 8, 9 or the previously defined variable n.
FORTRAN References	BACKSPACE x x = 5, 6, 7, 8, 9 or the variable n.
Method	This routine converts the logical magnetic tape unit number (5, 6, 7, 8, or 9) to the corresponding physical magnetic tape unit number (1, 2, 3, 4, or 5) and then calls the driver to back space one record on the specified magnetic tape.
Error Message	An error message BF is reported if an end-of-file is encountered.
Other Routines Used	CBR, FER

Purpose	To process FORTRAN run-time assigned GO TO statements.

DAP Calling Sequence

```
LDA    PTR    Transfer address in A-register
CALL   F$GA
DEC    n      Number of statements in list
DAC    S1     Address of first statement
DAC    S2     Address of second statement
 .      .
 .      .
 .      .
DAC    Sn     Address of last statement
```

FORTRAN Reference

```
ASSIGN J TO I               J = statement number
GO TO I, (K1, K2, . . . Kn)  I = integer variable name
                            Ks = statement numbers
```

Method

This routine checks the address passed in the A-register against the statement address list that follows the call. If the address is found in that list, control passes to that statement. If not, a GO error is reported. No recovery from this error is possible. If the statement address list is empty (n = 0), the address is not checked.

Data Type of Argument

An address is passed to this routine in the A-register. Control is passed to the statement number at that address.

Other Routines Used

F$ER

F$GC

Purpose	To process FORTRAN run-time computed GO TO statements.

DAP Calling Sequence

```
LDA     PTR     Index to statement list
CALL    F$GC
DEC     n       Number of statements in list
DAC     S1      Address of first statement
DAC     S2      Address of second statement
 .       .
 .       .
 .       .
DAC     Sn      Address of last statement in list
```

FORTRAN Reference

GO TO (K1, K2, . . . Kn), I   I = integer value in the range 1 to n
Ks = statement numbers

Method

The integer variable I or the content of PTR is treated as an index number; F$GC uses it to select the statement number from that position in the calling sequence. For example, if I = 3 then control is shifted to statement K3 or S3 in the above examples. If the index (I) is < 1 or $>n$, the computed GO TO statement is treated as a NOP.

Data Type of Argument

An integer value is passed to this routine in the A-register. Control is passed to the statement at the computed address.

Purpose	To cause the computer to stop and print PA if a PAUSE statement has been encountered, or to print ST if a STOP statement has been encountered.
DAP Calling Sequence	CALL F$HT DAC '151724 Octal notation for ST CALL F$HT DAC '150301 Octal notation for PA
FORTRAN Reference	STOP or PAUSE
Method	This calling sequence is generated by the compiler when the STOP or PAUSE verb is encountered. The mnemonic ST or PA is placed in relocatable address AC5 and printed on the ASR. The A-register is restored and the program halts. Return to the calling program may be made by pressing START.
Data Type of Argument	The binary equivalent to the specified ASCII characters.
Other Routines Used	AC5

F$IO

Purpose	To perform input/output conversion, to edit input/output information, to accommodate the appropriate input/output device, and to provide buffers.	

DAP Calling Sequence

DAC	a	a = location of the format list from the READ statement
CALL	F$IO	
DAC	BUF	BUF = buffer location
DAC*	a	The calling sequence for a FORTRAN WRITE with a and BUF as above
CALL	F$IO	
DAC	BUF	

FORTRAN References

READ (n, x) list	n = device number
WRITE (n, x) list	x = format statement number
x FORMAT ( )	

Method

A FORTRAN READ/WRITE statement starts with a device number and a reference to a format statement, followed by an optional argument list. The first instruction generated by the READ/WRITE statement is a coupling to the appropriate device driver. The device driver then calls on F$IO passing the location of the format list, setting the entry location for the device driver and setting a flag to indicate input or output. F$IO then interprets the format list, character-by-character, taking whatever actions are required. Whenever data is required from or is to be stored in the internal source, exit is made from F$IO to the next location in the data list.

Other Routines Used

FAR, FCB, F$ER

Purpose	To control the typewriter keyboard input routine.	
DAP Calling Sequence	CALL F$R1 DAC n (Return)	n = location of the format descriptor list
FORTRAN References	READ(1, f) list READ(1, f)	f = FORTRAN statement number

Method

This subroutine connects the calling program with the I/O control subroutine (F$IO). Included in F$R1 is the driving logic needed to transmit input from the typewriter keyboard. When F$IO is called, the location of the format descriptor list (if any), the entry location of the driver subroutine, and a flag indicating input are transmitted.

Whenever the F$IO subroutine requires data, return is made to the driver input entry of this subroutine, at which time up to 120 characters (terminated by a carriage return) are entered into the input buffer.

Data Type

Information is in ASCII format.

Other Routines Used

FIO, IAA

F$R2

Purpose	To control the paper tape input routine.

DAP Calling Sequence

CALL	F$R2	
DAC	n	Location of the format descriptor list (or 00000 if input is in binary format)
(Return)		

FORTRAN References

READ(2, f) list	Formatted READ where f is a format statement number
READ(2, f)	
READ(2) list	Unformatted paper tape read
READ(2)	

Method

This subroutine connects the calling program with the I/O control subroutine (F$IO). Included in F$R2 is the driving logic needed to transmit input from the paper tape reader.

When F$IO is called the location of the format descriptor list (if any), the entry location of the driver subroutine, and a flag indicating input are transmitted.

Whenever the F$IO subroutine requires data, it calls on the F$R2 driver subroutine, which assembles data into a 60-word buffer, three characters per word, if the input mode is binary. If the input mode is BCD, 80 characters are assembled, two per word, into a 40-word buffer. A carriage return character is replaced with as many blanks as needed to fill the rest of the input buffer. A tab character is replaced with as many blanks as needed to reach the next predetermined tab position.

A STOP code read in either binary or BCD mode causes the characters ST to be typed followed by a halt. Press START to continue.

Data Type

Information is in ASCII if formatted, or in binary if unformatted.

Other Routines Used

FIO, IPA, I$PB

Purpose	To control the card input routine.	
DAP Calling Sequence	CALL F$R3 DAC n (Return)	 Location of format descriptor list (00000 if input is binary)
FORTRAN References	READ(3, f) list READ(3, f) READ(3) list READ(3)	f = FORTRAN Statement number Unformatted read from a punch card

Method

This subroutine connects the calling program to the I/O control subroutine F$IO. Included in this subroutine is the driving logic required to input from the card reader. When F$IO is called, this subroutine transmits the location of the driver subroutine and a flag indicating input.

When the F$IO subroutine requires data, it calls upon the F$R3 driver subroutine, which assembles 80 characters (two per word) into a 40-word buffer, if mode is Hollerith (formatted), or into a 60-word buffer in column binary format (three words for every four columns) if binary. Return is made to F$IO where the buffer is processed.

Data Type

Information is in Hollerith if formatted, or in column binary if unformatted.

Other Routines Used

FIO, ICA, I$CB

F$R5-9

Purpose	To control reading of magnetic tape.		
DAP Calling Sequence	CALL	F$Rx	x = 5, 6, 7, 8, or 9
	DAC	n	Location of the format descriptor list
	(Return)		if formatted, or zero if unformatted
FORTRAN References	READ(x, f) list		f = FORTRAN statement number and x = 5, 6, 7, 8, or 9
	READ(x, f)		
	READ(x) list		Unformatted read where x = 5, 6, 7, 8, or 9
	READ(x)		

Method

This subroutine connects the calling program with I/O control routine (F$IO) and standard magnetic tape routines.

When F$IO is called, the format descriptor list and a flag indicating input are transmitted.

When the F$IO routine needs a buffer of data, it calls this driver, which in turn calls the appropriate magnetic tape unit and conversion routines (for formatted READ). The number of words read is 60, equivalent to 120 characters.

The appropriate magnetic tape units are physical magnetic tape units 1 through 5 corresponding respectively to logical tape units, numbers 5 through 9.

Data Type

Information is in ASCII if formatted, or in binary if unformatted.

Other Routines Used

FIO, IMA, IMC, C6T08

Purpose	To control the input drivers for variable input device numbers.		
DAP Calling Sequence	LDA	d	Location of device number
	CALL	F$Rn	
	DAC	n	Location of Format Descriptor List = 00000
	(Return)		if format is binary
FORTRAN References	READ(x, f) list		f = FORTRAN statement number;
	READ(x, f)		x = variable device number 1 through 9
	READ(x) list		unformatted read where x = 1, 2, 3, 5, 6, 7, 8, 9
	READ(x)		

Method

The value of d is checked for correct limits and is then used to determine the entry position of a Jump Table. The Jump Table transfers to the proper F$R subroutine. (Note that the entire F$R subroutine must be called into memory along with this subroutine, because there is no way of knowing in advance which drivers are required.)

If d does not equal a number from 1 to 9, the computer halts with a 1 in the A-register. The A-register may be changed manually to another device number; otherwise, the typewriter will be selected as the input device when START is pressed to continue processing.

Other errors, such as parity, end of tape, etc., cause the actions described in the appropriate F$R subroutine.

Data Type

Information is in ASCII if formatted, or in binary if unformatted.

Other Routines Used

F$R1, F$R2, F$R3, F$R5-9

F$TR

Purpose	To aid in debugging programs by printing the following:

- Values of variables or array elements as they are being stored
- Values of the DO parameters as they vary
- Locations of statement numbers as they are encountered
- Content of IF statements as they are evaluated.

DAP Calling Sequence

CALL	F$TR	
OCT		The first three bits of the first argument are:
OCT		000 = 0 = Statement number
OCT		001 = 1 = Integer
(Return)		010 = 2 = Real
		011 = 3 = Logical
		100 = 4 (Not Used)
		101 = 5 = Complex
		110 = 6 = Double precision
		The mode of an IF statement depends on the mode of the expression being evaluated. The remaining 13 bits of ARG1 and all the ARG2 and ARG3 are the name of the variable, the statement number, or blank.

FORTRAN References

TRACE x1, x2, ... xn	Each x represents a variable or array name;
TRACE n	n represents a statement number

Method

The A-register is stored in location zero (index register). The location of the argument is extracted and is interchanged with the index register, restoring the A-register. Sense Switch 4 is interrogated and if set, return is made to the calling program. If Sense Switch 4 is reset, trace mode is entered. The A- and B-registers are saved, and the buffer pointer is reset. The mode jump address is then set up. The 1- to 6-character name is then put in the format statement buffer. The return address for exit is calculated and stored. The 1- to 6-character name is interrogated to determine whether a statement number or a variable is being processed. If a variable name is being processed, an equal sign is stored in the format buffer following the name of the variable. An indirect jump is then taken, contingent upon whether the mode is real, integer, logical, complex, or double-precision. The following formats are moved to the format buffer, depending on the mode, to provide maximum representation of the variable.

REAL	G19.7
INTEGER	I7
LOGICAL	L2
COMPLEX	E15.7,H, E15.7
DOUBLE PRECISION	D19.11

The output device and format location are provided to the appropriate I/O driver and communicated to F$IO. Then the variable name and equal sign, and its appropriate value, are printed. When an IF statement evaluation is being written the four characters IF( ) are printed, followed by an equal sign and the value of the expression in the parentheses.

If a statement number is being printed, the number is bracketed by opening and closing parentheses, and is printed. Upon completion of each printout, the buffer is closed and the A- and B-registers restored. Return is made to the calling program.

Data Type of Arguments

See calling sequence.

Other Routines Used

F$W1, F$AR, F$CB, AC1, AC2, AC3

F$W1

Purpose	To control the typewriter output routine.	
DAP Calling Sequence	CALL F$W1 DAC n (Return)	 Location of the format descriptor list
FORTRAN References	WRITE(1, f)list WRITE(1, f)	f = FORTRAN statement number

Method

This subroutine connects the calling program with the I/O control subroutine (F$IO). Included in this subroutine is the driving logic needed to produce output on the typewriter.

When F$IO is called, the location of the format descriptor list (if any), the entry location of the driver subroutine, and a flag indicating output are transferred.

After the F$IO subroutine has generated a buffer full of data (72 characters), return is made to the driver output entry of this subroutine. At that time, the first character of the buffer is analyzed for proper line feed control as follows:

Blank	— Type a single carriage return followed by characters (2-72)
0	— Issue two carriage returns, then type characters (2-72)
1	— Skip to the top of the next page, then type characters (2-72)
+	— No line advance, type characters (2-72)
Others	— Same as blank, except characters (1-72) are typed

This subroutine prints 60 lines per page and skips six lines to the top of the next page. The operator should start three lines down the first page in order to get correct spacing between pages.

Data Type

Only ASCII information is processed.

Other Routines Used

FIO, OAC, O$AF

Purpose	To control the paper tape output routine.	
DAP Calling Sequence	CALL F$W2 DAC n (Return)	 Location of format descriptor list (00000 if output mode is binary)
FORTRAN References	WRITE(2, f) list WRITE(2, f) WRITE(2) list	f = FORTRAN statement number Unformatted WRITE to paper tape punch

Method

This subroutine connects the calling program to the I/O control subroutine F$IO. Included in the subroutine is the driving logic required to produce output on the paper tape punch. When F$IO is called, this subroutine transmits the location of the format descriptor list (if any) the entry location of the driver subroutine, and a flag indicating input.

After the F$IO subroutine has generated a full buffer of data 60 words at two characters per word, or 40 words at three binary characters per word, return is made to the driver output entry of the subroutine. At that time, the buffer is punched on tape.

Data Type

Information is in ASCII if formatted or in binary if unformatted.

Other Routines Used

FIO, OPF, OPP, OPB

F$W3

Purpose	To control the card punch routines.	
DAP Calling Sequence	CALL F$W3 DAC n (Return)	 Location of format descriptor list (00000 if output mode is binary)
FORTRAN References	WRITE(3, f) list WRITE(3, f) WRITE(3) list	f = format statement number Unformatted WRITE to the card punch

Method

This subroutine connects the calling program (FORTRAN Object Program) to the I/O control subroutine, F$IO, and to the card punch subroutines. When F$IO is called, F$W3 transmits the location of the address of the format descriptor list (if any), including a flag indicating output mode (DAC* for input), and a location for reentrance to F$W3.

After the F$IO subroutine has generated a full buffer of data (40 words at two BCD characters per word or 60 binary words), return is made to F$W3. The appropriate card punch subroutine is called, and a card is punched.

Data Type

Information is in Hollerith if formatted, or in column binary if unformatted.

Other Routines Used

FIO, OCH, O$CB

Note

The Hollerith information can be either 026 or 029 character set, depending on the version of O$CH selected.

Purpose	To control the line printer output routine.

DAP Calling Sequence

CALL	F$W4	
DAC	n	Location of format descriptor list
(Return)		

If n is zero, which normally indicates binary output, the computer halts. Push START to print data in BCD format.

FORTRAN References

WRITE(4, f) list	f = format statement number
WRITE(4, f)	

Method

This subroutine connects the calling program to the I/O control subroutine F$IO. Included in the subroutine is the driving logic required to produce output on the line printer. When F$IO is called, this routine transmits the location of the format descriptor list (if any), the entry location of the driver subroutine, and a flag indicating output mode.

After the F$IO subroutine has generated a full buffer of data (120 characters), return is made to the driver output entry of this subroutine. At that time, the first character of the buffer is analyzed for proper line feed control as follows:

Blank	– Advance one line and print characters 2 through 120.
0	– Advance two lines and print characters 2 through 120.
1	– Advance to top of next page and print characters 1 through 120.
+	– Print characters 1 through 120 without advancing line position.
Others	– Advance one line and print characters 1 through 120.

Data Type

Information is in ASCII format.

Other Routines Used

FIO, OLP, O$LO

F$W5-9

Purpose	To control writing on magnetic tape.		
DAP Calling Sequence	CALL	F$Wx	x = 5, 6, 7, 8, or 9
	DAC	n	Address of format, if any
FORTRAN References	WRITE(x, f) list		f = FORTRAN statement number
	WRITE(x, f)		x = 5, 6, 7, 8, or 9
	WRITE(x) list		Unformatted READ, x = 5, 6, 7, 8, or 9

Method

This program connects the calling program with the I/O control routine (F$IO) and the standard magnetic tape routines. When F$IO is called, the format descriptor list and a flag indicating output mode are transferred.

When the F$IO routine has a buffer of data to write, it calls the driver, which in turn calls the appropriate magnetic tape and conversion routines (for formatted WRITE).

60 words are written, equivalent to 120 characters in either mode (formatted or unformatted).

The appropriate magnetic tape units (physical) are numbers 1 through 5 corresponding respectively to the logical tape units numbers 5 through 9 given for x above.

Data Type

Information on the tape is in ASCII if formatted, or in binary if unformatted.

Other Routines Used

FIO, OMC, OMA, C8TO6

Purpose	To control the output drivers for variable output device numbers.

DAP Calling Sequence

LDA	d	Location of device number
CALL	F$Wn	
DAC	n	Location of format descriptor list
(Return)		(00000 if format is binary)

FORTRAN References

WRITE(x, f) list	f = FORTRAN statement number
WRITE(x, f)	x = variable device number 1 through 9
WRITE(x) list	Unformatted WRITE, where x = 1, 2, 3, 4, 5, 6, 7, 8, or 9

Method

The value of d is checked for correct limits and then used to determine the entry position of a Jump Table. The Jump Table is then transferred to the proper F$W subroutine. (Note that all F$W subroutines must be called into memory along with this subroutine, because there is no way of knowing in advance which drivers are required.

If d does not equal a number from 1 through 9, the computer halts with a 1 in the A-register. The A-register may be changed manually to another device number. Otherwise, the typewriter will be selected as the output device when START is pressed to continue processing.

Other errors, such as parity, end of tape, etc., cause the actions described in the appropriate F$W subroutine.

Data Type

Information is in ASCII if formatted, or in binary if unformatted.

Other Routines Used

F$W1, F$W2, F$W3, F$W4, F$W5-9

APPENDIX A

TAPE CONTENTS

MAGNETIC TAPE 70182805000 - LTCSIS (LIBRARY SOURCES CODED IN FORTRAN)

This tape consists of the individual sources of the following programs in the order listed. This tape is one of two distributed and contains that portion of the FORTRAN Library that is FORTRAN-coded.

FILE NUMBER	NAME	DOC. NO.
1	STMEAN	70181386000
2	STGEOM	70181387000
3	STCORR	70181388000
4	STSPER	70181389000
5	STCRMT	70181390000
6	STMEDT	70181391000
7	STCHI2	70181392000
8	STBNPB	70181394000
9	STLNRG	70181395000
10	STCHIS	70181393000
11	STPLRG	70181396000
12	STANVI	70181397000
13	STANV2	70181398000
14	STANVR	70181399000
15	STANVL	70181400000
16	STANVG	70181401000
17	STANVB	70181402000
18	STANVY	70181403000
19	DEFOAD	70181405000
20	DEFOMA	70181404000
21	DEFOHA	70181406000
22	DEFORK	70181407000
23	DESOAD	70181408000
24	DESOMA	70181409000
25	DESOHA	70181410000
26	DESORK	70181411000
27	PLYMUL	70181416000
28	PLYDIV	70181417000
29	PLYINT	70181420000
30	PLYIRT	70181421000
31	PLYEVL	70181422000
32	PLYDEF	70181423000
33	NACPLY	70181429000
34	NAAITK	70181418000
35	NALAGR	70181419000
36	NABAIR	70181424000
37	MATTRS	70181412000
38	MATMUL	70181413000
39	MARITH	70181414000

FILE NUMBER	NAME	DOC. NO.
40	MATEIG	70181415000
41	MATINV	70181427000
42	NAREGU	70181425000
43	NAMULL	70181426000
44	GSEID	70181428000
45	SORT	70181430000
46	SORT2	70181431000
47	CVPOLR	70181432000
48	E$62	70180053000
49	E$61	70180052000
50	E$26	70182582000
51	E$66	70180054000
52	DSQRT	70182580000
53	DCOS	70180055000
54	DSIN	70182583000
55	DEXP	70182581000
56	DLOGIO	70180051000
57	DLOG	70182579000
58	DLOG2	70182914000
59	DATAN2	70180056000
60	DATAN	70182584000
61	DMOD	70182588000
62	DSIGN	70182589000
63	DABS	70182587000
64	A$62	70180037000
65	S$62	70180038000
66	M$62	70180039000
67	D$62	70180040000
68	C$16	70180059000
69	DBLE	70180058000
70	CSQRT	70182592000
71	CCOS	70180066000
72	CSIN	70182595000
73	CLOG	70182591000
74	CEXP	70182593000
75	CABS	70182596000
76	E$51	70182594000
77	A$52	70180041000
78	S$52	70180042000
79	M$52	70180043000
80	D$52	70180044000
81	A$55	70182544000
82	S$55	70180093000
83	M$55	70182545000
84	D$55	70180034000
85	CONJG	70182598000
86	C$25	70180068000
87	CMPLX	70182597000
88	N$55	70180069000
	END	

MAGNETIC TAPE 70182806000 – LTCS2S
(LIBRARY SOURCES CODED IN DAP)

FILE NUMBER	NAME	DOC. NO.
1	DMAX1	70182585000
2	DMIN1	70182586000
3	DINT	70180850000
4	Z$80	70180851000
5	A$81	70180852000
6	C$61	70182554000
7	A$66	70180853000
8	A$66XRA	70180979000
9	H$66	70180855000
10	C$26	70180857000
11	H$55	70180860000
12	MAX0	70182548000
13	MAX1	70182549000
14	MIN0	70180649000
15	MIN1	70182551000
16	TANH	70182565000
17	SQRT	70182560000
18	SQRTX	70180681000
19	SIN, COS	70182563000
20	ATAN	70182564000
21	E$21	70182562000
22	E$22	70180045000
23	ALOG	70182559000
24	ALOGX	70180682000
25	EXP	70182561000
26	E$11	70182547000
27	E$11X	70180684000
28	ABS	70182570000
29	C$62	70180884000
30	AMOD	70182572000
31	L$66	70180854000
32	AINT	70182571000
33	N$66	70180856000
34	DIM	70182573000
35	SIGN	70182574000
36	AIMAG	70180858000
37	L$55	70180859000
38	IFIX	70182553000
39	FLOAT	70180062000
40	C$12	70182575000
41	C$21	70182558000
42	LOC	70181962000
43	C$81	70180882000
44	ISTORE	70181982000
45	N$33	70180090000
46	IFETCH	70181983000
47	IABS	70182552000
48	F$OE	70181984000
49	MOD	70182555000
50	F$TR-RA	70180827000
51	SUB$	70185150000

FILE NUMBER	NAME	DOC. NO.
52	F$GA	70185151000
53	F$GC	70185152000
54	IDIM	70182556000
55	A$22	70182536000
56	M$22	70182537000
57	A$22X11	70181805000
58	M$22X11	70181806000
59	D$22X11	70181804000
60	ISIGN	70182557000
61	L$22	70182534000
62	H$22	70182535000
63	N$22	70180097000
64	SLITE	70182599000
65	M$11	70180035000
66	D$11	70182546000
67	M$11X	70180685000
68	D$11X	70180686000
69	OVERFL	70180894000
70	F$AT	70180071000
71	L$33	70180065000
72	F$WN	70180089000
73	F$RN	70180088000
74	F$R1	70182610000
75	F$W1	70182611000
76	F$R2	70182612000
77	F$W2	70182613000
78	F$R3	70182614000
79	F$W3	70181667000
80	F$W4	70182616000
81	F$R5-9	70180306000
82	F$F5-9	70180310000
83	F$W5-9	70180307000
84	F$10	70182618000
85	016CHAIN	70180659000
86	ARG$	70180072000
87	F$D5-9	70180308000
88	F$B5-9	70180309000
89	F$ER-RA	70181068000
90	AC1	70180717000
	END	

MAGNETIC TAPE 70182803541 – LTCM1S (LIBRARY OBJECTS - SOFTWARE VERSION)

This magnetic tape consists of the concatenation of the individual objects of the listed programs. They have been translated by the FORTRAN Translator Mod 1 if FORTRAN coded and/or assembled by the DAP-16 Mod 2 Assembler.

FILE NUMBER	NAME	DOC. NO.
1	STMEAN	70181386000
2	STGEOM	70181387000
3	STCORR	70181388000
4	STSPER	70181389000
5	STCRMT	70181390000
6	STMEDT	70181391000
7	STCH12	70181392000

FILE NUMBER	NAME	DOC. NO.
8	STBNPB	70181394000
9	STLNRG	70181395000
10	STCHIS	70181393000
11	STPLRG	70181396000
12	STANV1	70181397000
13	STANV2	70181398000
14	STANVR	70181399000
15	STANVL	70181400000
16	STANVG	70181401000
17	STANVB	70181402000
18	STANVY	70181403000
19	DEFOAD	70181405000
20	DEFOMA	70181404000
21	DEFOHA	70181406000
22	DEFORK	70181407000
23	DESOAD	70181408000
24	DESOMA	70181409000
25	DESOHA	70181410000
26	DESORK	70181411000
27	PLYMUL	70181416000
28	PLYDIV	70181417000
29	PLYINT	70181420000
30	PLYIRT	70181421000
31	OLYEVL	70181422000
32	PLYDIF	70181423000
33	NACPLY	70181429000
34	NAAITK	70181418000
35	NALAGR	70181419000
36	NABAIR	70181424000
37	MATTRS	70181412000
38	MATMUL	70181431000
39	MARITH	70181414000
40	MATEIG	70181415000
41	MATINV	70181427000
42	NAREGU	70181425000
43	NAMULL	70181426000
44	GSEID	70181428000
45	SORT	70181430000
46	SORT2	70181431000
47	CVPOLR	70181432000
48	E$62	70180053000
49	E$61	70180052000
50	E$26	70182582000
51	E$66	70180054000
52	DSQRT	70182580000
53	DCOS	70180055000
54	DSIN	70182583000
55	DEXP	70182581000
56	DLOG10	70180051000
57	DLOG	70182579000
58	DLOG2	70182914000
59	DATAN2	70180056000
60	DATAN	70182584000
61	DMOD	70182588000
62	DSIGN	70182589000
63	DABS	70182587000
64	A$62	70180037000

FILE NUMBER	NAME	DOC. NO.
65	S$62	70180036000
66	M$62	70180039000
67	D$62	70180040000
68	C$16	70180059000
69	DBLE	70180058000
70	CSQRT	70182592000
71	CCOS	70180066000
72	CSIN	70182595000
73	CLOG	70182591000
74	CEXP	70182593000
75	CABS	70182596000
76	E$51	70182594000
77	A$52	70180041000
78	S$52	70180042000
79	M$52	70180043000
80	D$52	70180044000
81	A$55	70182544000
82	S$55	70180093000
83	M$55	70182545000
84	D$55	70180034000
85	CONJG	70182598000
86	C$25	70180068000
87	CMPLX	70182597000
88	N$55	70180069000
89	DMAX1	70182585000
90	DMIN1	70182586000
91	DINT	70180850000
92	Z$80	70180851000
93	A$81	70180852000
94	C$61	70182554000
95	A$66	70180853000
96	H$66	70180855000
97	C$26	70180857000
98	H$55	70180860000
99	MAX0	70182548000
100	MAX1	70182549000
101	MIN0	70180649000
102	MIN1	70182551000
103	TANH	70182565000
104	SQRT	70182560000
105	SIN, COS	70182563000
106	ATAN	70182564000
107	E$21	70182562000
108	E$22	70180045000
109	ALOG	70182559000
110	EXP	70182561000
111	E$11	70182547000
112	ABS	70182570000
113	C$62	70180884000
114	AMOD	70182572000
115	L$66	70180854000
116	AINT	70182571000
117	N$66	70180856000
118	DIM	70182573000
119	SIGN	70182574000
120	AIMAG	70180858000

FILE NUMBER	NAME	DOC. NO.
121	L$55	70180859000
122	IFIX	70182553000
123	FLOAT	70180062000
124	C$12	70182575000
125	C$21	70182558000
126	LOC	70181962000
127	C$81	70180882000
128	ISTORE	70181982000
129	N$33	70180090000
130	IFETCH	70181983000
131	IABS	70182552000
132	F$DE	70181984000
133	MOD	70182555000
134	F$TR-RA	70180827000
135	SUB$	70185150000
136	F$GA	70185151000
137	F$GC	70185152000
138	IDIM	70182556000
139	A$22	70182536000
140	M$22	70182537000
141	ISIGN	70182557000
142	L$22	70182534000
143	H$22	70182535000
144	N$22	70180097000
145	SLITE	70182599000
146	M$11	70180035000
147	D$11	70182546000
148	OVERFL	70180894000
149	F$AT	70180071000
150	L$33	70180065000
151	F$WN	70180089000
152	F$RN	70180088000
153	F$R1	70182610000
154	F$W1	70182611000
155	F$R2	70182612000
156	F$W2	70182613000
157	F$R3	70182614000
158	F$W3	70181667000
159	F$W4	70182616000
160	F$R5-9	70180306000
161	F$F5-9	70180310000
162	F$W5-9	70180307000
163	F$10	70182618000
164	016CHAIN	70180659000
165	ARG$	70180072000
166	F$D5-9	70180308000
167	F$B5-9	70180309000
168	F$ER-RA	70181068000
169	AC1	70180717000
170	SQRX1	70188775000
171	COSX1	70188781000
172	SINX1	70188777000
173	ANTX1	70188779000
174	LGEX1	70188814000
175	LG2X1	70188784000
176	EXEX1	70188786000
177	EX2X1	70188782000

FILE NUMBER	NAME	DOC. NO.
178	DSQRX1	70188788000
179	DCOSX1	70188792000
180	DSINX1	70188790000
181	DATNX1	70188793000
182	DLGEX1	70188801000
183	DLG2X1	70188795000
184	DEXEX1	70188799000
185	DEX2X1	70188797000
186	DMPY	70188808000
187	MPY	70188811000
188	DIV	70188810000
189	DADD	70188812000
190	DSUB	70188813000
191	ROND	70188805000
192	RODD	70188804000
193	TWOS	70188803000
	END	

Files	1-47	Statistical Library
Files	48-169	FORTRAN Library
Files	170-193	Fixed Point Math Libra.

MAGNETIC TAPE 70182804541 – LTCM1H (LIBRARY OBJECTS - HARDWARE VERSION)

This magnetic tape consists of the concatenation of the individual objects of the listed programs. They have been translated by the FORTRAN Translator Mod 1 if FORTRAN code and/or assembled by the DAP-16 Mod 2 Assembler.

FILE NUMBER	NAME	DOC. NO.
1	STMEAN	70181386000
2	STGEON	70181387000
3	STCORR	70181388000
4	STSPER	70181389000
5	STCRMT	70181390000
6	STMEDT	70181391000
7	STCHI2	70181392000
8	STBNPB	70181394000
9	STLNRG	70181395000
10	STCHIS	70181393000
11	STPLRG	70181396000
12	STANV1	70181397000
13	STANV2	70181398000
14	STANVR	70181399000
15	STANVL	70181400000
16	STANVG	70181401000
17	STANVB	70181402000
18	STANVY	70181403000
19	DEFOAD	70181405000
20	DEFOMA	70181404000
21	DEFOHA	70181406000
22	DEFORK	70181407000
23	DESOAD	70181408000

FILE NUMBER	NAME	DOC. NO.
24	DESONA	70181409000
25	DESOHA	70181410000
26	DESORK	70181411000
27	PYLMUL	70181416000
28	PLYDIV	70181417000
29	PLYINT	70181420000
30	PLYIRT	70181421000
31	PLYEVL	70181422000
32	PLYDIF	70181423000
33	NACPLY	70181429000
34	NAAITK	70181418000
35	NALAGR	70181419000
36	NABAIR	70181424000
37	NATTRS	70181412000
38	NATMUL	70181413000
39	MARITH	70181414000
40	NATEIG	70181415000
41	NATINV	70181427000
42	NAREGU	70181425000
43	NAMULL	70181426000
44	GSEID	70181428000
45	SORT	70181430000
46	SORT2	70181431000
47	CVPOLR	70181432000
48	E$62	70180053000
49	E$61	70180052000
50	E$26	70182582000
51	E$66	70180054000
52	DSQRT	70182580000
53	DCOS	70180055000
54	DSIN	70182583000
55	DEXP	70182581000
56	DLOG10	70180051000
57	DLOG	70182579000
58	DLOG2	70182914000
59	DATAN2	70180056000
60	DATAN	70182584000
61	DMOD	70182588000
62	DSIGN	70182589000
63	DABS	70182587000
64	A$62	70180037000
65	S$62	70180038000
66	M$62	70180039000
67	D$62	70180040000
68	C$16	70180059000
69	DBLE	70180058000
70	CSQRT	70182592000
71	CCOS	70180066000
72	CSIN	70182595000
73	CLOG	70182591000
74	CEXP	70182593000
75	CABS	70182596000
76	E$51	70182594000
77	A$52	70180041000
78	S$52	70180042000
79	M$52	70180043000
80	D$52	70180044000

FILE NUMBER	NAME	DOC. NO.
81	A$55	70182544000
82	S$55	70180093000
83	M$55	70182545000
84	D$55	70180034000
85	CONJG	70182598000
86	C$25	70180068000
87	CMPLX	70182597000
88	N$55	70180069000
89	DMAX1	70182585000
90	DMIN1	70182586000
91	DINT	70180850000
92	Z$80	70180851000
93	A$81	70180852000
94	C$61	70182554000
95	A$66XRA	70180979000
96	H$66	70180855000
97	C$26	70180857000
98	H$55	70180860000
99	MAX0	70182548000
100	MAX1	70182549000
101	MIN0	70180649000
102	MIN1	70182551000
103	TANH	70182565000
104	SQRTX	70180681000
105	SIN, COS	70182560000
106	ATAN	70182564000
107	E$21	70182562000
108	E$22	70180045000
109	ALOGX	70180682000
110	EXP	70182561000
111	E$11X	70180684000
112	ABS	70182570000
113	C$62	70180884000
114	AMOD	70182572000
115	L$66	70180854000
116	AINT	70182571000
117	N$66	70180856000
118	DIM	70182573000
119	SIGN	70182874000
120	AIMAG	70180858000
121	L$55	70180859000
122	IFIX	70182553000
123	FLOAT	70180062000
124	C$12	70182575000
125	C$21	70182558000
126	LOC	70181962000
127	C$81	70180882000
128	ISTORE	70181982000
129	N$33	70180090000
130	IFETCH	70181983000
131	IABS	70182552000
132	F$OE	70181984000
133	MOD	70182555000
134	F$TR-RA	70180827000
135	SUB$	70185150000
136	F$GA	70185151000
137	F$GC	70185152000

FILE NUMBER	NAME	DOC. NO.
138	IDIM	70182556000
139	A$22X11	70181805000
140	M$22X11	70181806000
141	D$22X11	70181804000
142	ISIGN	70182557000
143	L$22	70182534000
144	H$22	70182535000
145	N$22	70180097000
146	SLITE	70182599000
147	M$11X	70180685000
148	D$11X	70180686000
149	OVERFL	70180894000
150	F$AT	70180071000
151	L$33	70180065000
152	F$WN	70180089000
153	F$RN	70180088000
154	F$R1	70182610000
155	F$W1	70182611000
156	F$R2	70182612000
157	F$W2	70182613000
158	F$R3	70182614000
159	F$W3	70181667000
160	F$W4	70182616000
161	F$R5-9	70180306000
162	F$F5-9	70180310000
163	F$W5-9	70180307000
164	F$I0	70182618000
165	016CHAIN	70180659000
166	ARG$	70180072000
167	F$D5-9	70180308000
168	F$B5-9	70180309000
169	F$ER-RA	70181068000
170	AC1	70180717000
171	SQRX2	70188776000
172	COSX2	70180761000
173	SINX2	70188778000
174	ATNX2	70188780000
175	LGEX2	70188815000
176	LG2X2	70188785000
177	EXEX2	70188787000
178	EX2X2	70188783000
179	DSQRX2	70188789000
180	DCOSX2	70180762000
181	DSINX2	70188791000
182	DATNX2	70188794000
183	DLGEX2	70188802000
184	DLG2X2	70188796000
185	DEXEX2	70188800000
186	DEX2X2	70188798000
187	DMPYH	70188809000
188	DADD	70188812000
189	DSUB	70188813000
190	ROND	70188805000
191	RODD	70188804000
192	TWOS	70188803000
	END	

Files	1-47	Statistical Library
Files	48-170	FORTRAN Library
Files	171-192	Fixed Point Math Library

PAPER TAPE 70181876000 - LTCF1 (Paper Tape 1 of 6)

This paper tape consists of the concatenation of the individual objects of the listed programs. They have been translated by the FORTRAN Translator MOD 1 and assembled by the DAP-16 Mod 2 Assembler.

FILE NUMBER	NAME	DOC. NO.
1	E$62	70180053000
2	E$61	70180052000
3	E$26	70182582000
4	E$66	70180054000
5	DSQRT	70182580000
6	DCOS	70180055000
7	DSIN	70182583000
8	DEXP	70182581000
9	DLOG10	70180051000
10	DLOG	70182579000
11	DLOG2	70182914000
12	DATAN2	70180056000
13	DATAN	70182584000
14	DMOD	70182588000
15	DSIGN	70182589000
16	DABS	70182587000
17	A$62	70180037000
18	S$62	70180038000
19	M$62	70180039000
20	D$62	70180040000
21	C$16	70180059000
22	DBLE	70180058000
	END	

PAPER TAPE 70181877000 - LTCF2 (Paper Tape 2 of 6)

This paper tape consists of the concatenation of the individual objects of the listed program. They have been translated by the FORTRAN Translator MOD 1 and assembled by the DAP-16 Mod 2 Assembler.

FILE NUMBER	NAME	DOC. NO.
1	CSQRT	70182592000
2	CCOS	70180066000
3	CSIN	70182595000
4	CLOG	70182591000
5	CEXP	70182593000
6	CABS	70182596000
7	E$51	70182594000
8	A$52	70180041000
9	S$52	70180042000
10	M$52	70180043000

FILE NUMBER	NAME	DOC. NO.
11	D$52	70180044000
12	A$55	70182544000
13	S$55	70180093000
14	M$55	70182545000
15	D$55	70180034000
16	CONJG	70182598000
17	C$25	70180068000
18	CMPLX	70182597000
19	N$55	70180069000
	END	

PAPER TAPE 70181882000 - LTCF3S (Paper Tape 3 of 6)

This is the Software Version of Paper Tape 3

FILE NUMBER	NAME	DOC. NO.
1	DMAX1	70182585000
2	DMIN1	70182586000
3	DINT	70180850000
4	Z$80	70180851000
5	A$81	70180852000
6	C$61	70182554000
7	A$66	70180853000
8	H$66	70180855000
9	C$26	70180857000
10	H$55	70180860000
11	MAX0	70182548000
12	MAX1	70182549000
13	MIN0	70180649000
14	MIN1	70182551000
15	TANH	70182565000
16	SQRT	70182560000
17	SIN, COS	70182563000
18	ATAN	70182564000
19	E$21	70182562000
20	E$22	70180045000
21	ALOG	70182559000
22	EXP	70182561000
23	E$11	70182547000
	END	

PAPER TAPE 70181878000 - LTCF3H (Paper Tape 3 of 6)

This is the Hardware Version of Paper Tape 3

FILE NUMBER	NAME	DOC. NO.
1	DMAX1	70182585000
2	DMIN1	70182586000
3	DINT	70180850000
4	Z$80	70180851000
5	A$81	70180852000
6	C$61	70182554000
7	A$66XRA	70180979000
8	H$66	70180855000
9	C$26	70180857000

FILE NUMBER	NAME	DOC. NO.
10	H$55	70180860000
11	MAX0	70182548000
12	MAX1	70182549000
13	MIN0	70180649000
14	MIN1	70182551000
15	TANH	70182565000
16	SQRTX	70180681000
17	SIN, COS	70182563000
18	ATAN	70182564000
19	E$21	70182562000
20	E$22	70180045000
21	ALOGX	70180682000
22	EXP	70182561000
23	E$11X	70180684000
	END	

PAPER TAPE 70181879000 - LTCF4 (Paper Tape 4 of 6)

FILE NUMBER	NAME	DOC. NO.
1	ABS	70182570000
2	C$62	70180884000
3	AMOD	70182572000
4	L$66	70180854000
5	AINT	70182571000
6	N$66	70180856000
7	DIM	70182573000
8	SIGN	70182574000
9	AIMAG	70180858000
10	L$55	70180859000
11	IFIX	70182553000
12	FLOAT	70180062000
13	C$12	70182575000
14	C$21	70182558000
15	LOC	70181962000
16	C$81	70180882000
17	ISTORE	70181982000
18	N$33	70180090000
19	IFETCH	70181983000
20	IABS	70182552000
21	F$OE	70181984000
22	MOD	70182555000
23	F$TR-RA	70180827000
24	SUB$	70185150000
25	F$GA	70185151000
26	F$GC	70185152000
	END	

PAPER TAPE 70181883000 - LTCF5S (Paper Tape 5 of 6)

This is the Software Version of Tape 5

FILE NUMBER	NAME	DOC. NO.
1	IDIM	70182556000
2	A$22	70182536000
3	M$22	70182537000
4	ISIGN	70182557000

FILE NUMBER	NAME	DOC. NO.
5	L$22	70182534000
6	H$22	70182535000
7	N$22	70180097000
8	SLITE	70182599000
9	M$11	70180035000
10	D$11	70182546000
11	OVERFL	70180894000
12	F$AT	70180071000
13	L$33	70180065000
14	F$WN	70180089000
15	F$RN	70180088000
16	F$R1	70182610000
17	F$W1	70182611000
18	F$R2	70182612000
19	F$W2	70182613000
20	F$R3	70182614000
21	F$W3	70181667000
22	F$W4	70182616000
23	F$R5-9	70180306000
24	F$F5-9	70180310000
25	F$W5-9	70180307000
26	F$IO	70182618000
	END	

PAPER TAPE 70181880000 - LTCF5H (Paper Tape 5 of 6)

This is the Hardware Version of Tape 5

FILE NUMBER	NAME	DOC. NO.
1	IDIM	70182556000
2	A$22X11	70181805000
3	M$22X11	70181806000
4	D$22X11	70181804000
5	ISIGN	70182557000
6	L$22	70182534000
7	H$22	70182535000
8	N$22	70180097000
9	SLITE	70182599000
10	M$11X	70180685000
11	D$11X	70180686000
12	OVERFL	70180894000
13	F$AT	70180071000
14	L$33	70180065000
15	F$WN	70180089000
16	F$RN	70180088000
17	F$R1	70182610000
18	F$W1	70182611000
19	F$R2	70182612000
20	F$W2	70182613000
21	F$R3	70182614000
22	F$W3	70181667000
23	F$W4	70182616000
24	F$R5-9	70180306000
25	F$F5-9	70180310000
26	F$W5-9	70180307000
27	F$IO	70182618000
	END	

PAPER TAPE 70181881000 - LTCF6 (Paper Tape 6 of 6)

FILE NUMBER	NAME	DOC. NO.
1	016CHAIN	70180659000
2	ARG$	70180072000
3	F$D5-9	70180308000
4	F$B5-9	70180309000
5	F$ER-RA	70181068000
6	AC1	70180717000
	END	

APPENDIX B

MATHEMATICAL ROUTINES

Function	Routine
Complex	
Absolute value	CABS
Add	A$55
Add real argument	A$52
Conjugate	CONJG
Convert imaginary part to real	AIMAG
Cosine	CCOS
Divide	D$55
Divide by real argument	D$52
Exponential, base e	CEXP
Load	L$55
Load real part	REAL
Logarithm, base e	CLOG
Multiply	M$55
Multiply by real argument	M$52
Negate	N$55
Raise to integer power	E$51
Sine	CSIN
Square root	CSQRT
Store (hold)	H$55
Subtract	S$55
Subtract single-precision argument	S$52
Double-Precision	
Absolute value	DABS
Add	A$66
Add single-precision argument	A$62
Add integer to exponent	A$81
Arctangent, principal value	DATAN

Function	Routine
Double-Precision	
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 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 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 real argument	S$62
Transfer sign of second argument to first	DSIGN
Truncate fractional bits	DINT
Truncate fractional bits and convert to integer	IDINT
Real	
Absolute value	ABS
Add	A$22
Arctangent, principal value	ATAN
Arctangent, X/Y	ATAN2
Convert pair to complex	CMPLX

Function	Routine
Real	
Convert to complex format	C$25
Convert to double-precision	C$26
Convert to integer	C$21
Divide	D$22
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
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
Transfer sign of second argument to first	SIGN
Truncate fractional bits	AINT
Truncate fractional bits and convert to integer	IFIX, INT
TWOs complement	N$22
Integer	
Absolute value	IABS
Convert to double-precision	C$16
Convert to real (FORTRAN-generated)	FLOAT
Convert to real	C$12
Divide	D$11
Maximum value	AMAX0

Function	Routine
Integer	
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
Transfer sign of second argument to first	ISIGN
Logical	
Complement	N$33
OR with A-register	L$33

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	
MIN10	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

Special Subroutines for FORTRAN Use

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

COMPILER SUPPORT SUBROUTINES

Conversion Routines

Name	Argument Data Type	Result Data Type	Function
C$12	I	R	Convert integer to real
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$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

Arithmetic Routines

Name	Function	Name	Function
A$22	R = R+R	E$62	D = D**R
A$52	C = C+R	E$66	D = D**D
A$55	C = C+C	M$11	I = I*I
A$62	D = D+R	M$22	R = R*R
A$66	D = D+D	M$52	C = C*R
A$81	D = D*(2**I)	M$55	C = C*C
D$11	I = I/I	M$62	D = D*R
D$22	R = R/R	M$66	D = D*D
D$52	C = C/R	N$22	R = -R
D$55	C = C/C	N$33	L = -L
D$62	D = D/R	N$55	C = -C
D$66	D = D/D	N$66	D = -D
E$11	I = I**I	S$22	R = R-R
E$21	R = R**I	S$52	C = C-R
E$22	R = R**R	S$55	C = C-C
E$26	D = R**D	S$62	D = D-R
E$51	C = C**I	S$66	D = D-D
E$61	D = D**I	Z$80	Replace binary exponent with zero

Miscellaneous Routines

Name	Function
AC1	Pseudo accumulators
ARG$	Convert indirect address to direct address
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
A$22	A$22	ARG$	1	150	5S
	S$22	N$22	1		
		F$ER	1		
A$22X	A$22	N$22	1	140	5H
	S$22	F$ER	1		
A$52	A$52	F$AT	1	20	2
		H$55	1		
		L$22	1		
		A$22	1		
		H$22	1		
		L$55	1		
A$55	A$55	F$AT	1	60	2
		H$55	1		
		SUB$	4		
		L$22	2		
		A$22	2		
		H$22	2		
		L$55	1		
A$62	A$62	F$AT	1	20	1
		H$66	1		
		DBLE	1		
		A$66	1		
A$66	A$66	N$66	11	580	3S
	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	3H
	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		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words $_{10}$)	Tape Number
A$81	A$81	N$22	2	70	3
		F$ER	1		
		AC1	1		
		AC2	1		
ABS	ABS	L$22	1	10	4
		N$22	1		
AC1	AC1			5	6
	AC2				
	AC3				
	AC4				
	AC5				
AIMAG	AIMAG	L$55	1	10	4
		L$22	1		
		AC3	1		
AINT	AINT	L$22	1	30	4
		N$22	2		
		A$22	1		
		S$22	1		
ALOG	ALOG10	ARG$	1	120	3S
	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	3H
	ALOG	C$12	1		
	ALOGX	A$22	4		
		M$22	4		
		S$22	1		
		F$ER	1		
ALOG10	See ALOG or ALOGX				
AMAX0	See MAX0				
AMAX1	See MAX1				
AMIN0	See MIN0				
AMIN1	See MIN1				
AMOD	AMOD	L$22	1	30	4
		D$22	1		
		AINT	1		
		M$22	1		
		N$22	1		
		A$22	1		
ARG$	ARG$			20	6

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number
ATAN	ATAN2	ARG$	3	340	3
	ATAN	D$22	6		
		N$22	7		
		M$22	5		
		A$22	11		
		S$22	2		
ATAN2	See ATAN				
C$12	C$12	A$22	1	30	4
		N$22	1		
C$16	C$16	C$12	1	5	1
		C$26	1		
C$21	C$21	N$22	1	30	4
		A$22	1		
		F$ER	1		
C$25	C$25	H$22	1	20	2
		CMPLX	1		
C$26	C$26	AC1	1	10	3
		AC2	1		
		AC3	1		
C$61	C$61	C$62	1	4	3
		C$21	1		
C$62	C$62	L$22	1	20	4
	SNGL	N$66	1		
		N$22	1		
		L$66	1		
		AC1	1		
		AC2	1		
C$81	C$81	AC1	1	10	4
CABS	CABS	F$AT	1	40	2
		SUB$	2		
		L$22	2		
		M$22	2		
		H$22	2		
		A$22	1		
		SQRT	1		
CCOS	CCOS	F$AT	1	40	2
		L$55	1		
		A$55	1		
		H$55	1		
		CSIN	1		
CEXP	CEXP	F$AT	1	60	2
		SUB$	7		
		EXP	1		
		H$22	3		
		COS	1		
		M$22	2		
		SIN	1		
		L$55	1		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number
CLOG	CLOG	F$AT	1	90	2
		SUB$	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	2
		SUB$	2		
		L$22	2		
		H$22	2		
		L$55	1		
CONJG	CONJG	F$AT	1	40	2
		SUB$	4		
		L$22	2		
		H$22	2		
		N$22	1		
COS	See SIN	L$55			
CSIN	CSIN	F$AT	1	90	2
		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$55	1		
CSQRT	CSQRT	F$AT	1	90	2
		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	5S
		F$ER	1		
D$11X	D$11	ARG$	1	40	5H
	D$11X	F$ER	1		
D$22	See M$22				
D$22X	D$22	N$22	3	110	5H
		F$ER	2		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage $(Words_{10})$	Tape Number
D$52	D$52	F$AT	1	50	2
		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	2
		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$62	D$62	F$AT	1	20	1
		H$66	2		
		DBLE	1		
		L$66	1		
		D$66	1		
D$66	See A$66				
D$66X	See A$66X				
DABS	DABS	F$AT	1	10	1
		L$66	1		
		N$66	1		
DATAN	DATAN	F$AT	1	180	1
		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
		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
		L$22	1		
		C$26	1		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words $_{10}$)	Tape Number
DCOS	DCOS	F$AT	1	20	1
		L$66	1		
		A$66	1		
		H$66	1		
		DSIN	1		
DEXP	DEXP	F$AT	1	160	1
		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	4
		S$22	1		
DINT	DINT	L$66	1	20	3
		N$66	2		
		A$66	1		
		S$66	1		
		AC1	1		
DIV$	See M$22				
DLOG	DLOG	F$AT	1	10	1
		DLOG2	1		
		M$66	1		
DLOG2	DLOG2	F$AT	1	100	1
		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
		DLOG2	1		
		M$66	1		
DMAX1	DMAX1	L$66	3	40	3
		H$66	2		
		S$66	1		
DMIN1	DMIN1	L$66	3	40	3
		H$66	2		
		S$66	1		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words $_{10}$)	Tape Number
DMOD	DMOD	F$AT	1	20	1
		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
		L$66	3		
		N$66	1		
DSIN	DSIN	F$AT	1	130	1
		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
		L$66	2		
		C$62	1		
		H$22	1		
		SQRT	1		
		C$26	1		
		H$66	1		
		D$66	1		
		A$66	1		
		A$81	1		
E$11	E$11	ARG$	1	100	3S
		M$11	2		
		F$ER	1		
E$11X	E$11	ARG$	1	110	3H
	E$11X	F$ER	1		
E$21	E$21	ARG$	1	50	3
		M$22	1		
		D$22	1		
E$22	E$22	ARG$	1	30	3
		ALOG	1		
		M$22	1		
		EXP	1		
E$26	E$26	F$AT	1	30	1
		C$26	1		
		H$66	2		
		DLOG	1		
		M$66	1		
		DEXP	1		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words$_{10}$)	Tape Number
E$51	E$51	F$AT	1	60	2
		H$55	3		
		IABS	1		
		L$55	4		
		M$55	1		
		D$55	1		
E$61	E$61	F$AT	1	70	1
		H$66	5		
		L$66	5		
		D$66	1		
		D$11	2		
		M$11	1		
		M$66	2		
E$62	E$62	F$AT	1	30	1
		H$66	2		
		DLOG	1		
		M$62	1		
		DEXP	1		
E$66	E$66	F$AT	1	30	1
		H$66	2		
		DLOG	1		
		M$66	1		
		DEXP	1		
EXP	EXP	ARG$	1	230	3
		N$22	2		
		M$22	6		
		S$22	3		
		A$22	2		
		D$22	2		
		F$ER	1		
F$AR	See F$IO				5
F$AT	F$AT			58	5
F$B5-9	F$B5	C$MR	6	26	6
	F$B6				
	F$B7				
	F$B8				
	F$B9				
	F$BN				
F$CB	See F$IO				5
F$D5-9	F$D2	O$PS	2	34	6
	F$D5	O$ME	6		
	F$D6				
	F$D7				
	F$D8				
	F$D9				
	F$DN				

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number
F$ER	F$ER	AC5	2	37	6
	F$HT				
F$F5-9	F$F5	C$BR	1	41	5
	F$F6	F$ER	1		
	F$F7				
	F$F8				
	F$F9				
	F$FN				
F$GA	F$GA	F$ER	1	18	4
F$GC	F$GC			14	4
F$HT	See F$ER				6
F$IO	F$IO	F$ER	2	1356	5
	F$CB				
	F$L1				
	F$L2				
	F$L3				
	F$L5				
	F$L6				
	F$AR				
F$R1	F$R1	F$IO	1	21	5
		I$AA	1		
		I$AB	1		
F$R2	F$R2	F$IO	1	21	5
		I$PA	1		
		I$PB	1		
F$R3	F$R3	F$IO	1	21	5
		I$CA	1		
		I$CB	1		
F$R5-9	F$R5	F$IO	1	80	5
	F$R6	I$MA	1		
	F$R7	I$MC	1		
	F$R8				
	F$R9				
F$Rn	F$Rn	F$R1	1	45	5
		F$R2	1		
		F$R3	1		
		F$R5	1		
		F$R6	1		
		F$R7	1		
		F$R8	1		
		F$R9	1		
F$TR	F$TR	F$W1	5	198	4
		F$AR	4		
		F$CB	1		
		F$L6	1		
		AC1	2		
		AC2	1		
		AC3	1		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number
F$W1	F$W1	F$IO	1	94	5
		O$AP	1		
		O$AC	1		
		O$AF	1		
		O$AB	1		
F$W2	F$W2	F$IO	1	80	5
		O$PF	1		
		O$PP	1		
		O$PC	1		
		O$PB	1		
F$W3	F$W3	F$IO	1	39	5
		O$CH	1		
		O$CB	1		
F$W4	F$W4	F$IO	1	36	5
		O$LF	3		
		O$LP	1		
		O$LO	1		
F$W5-9	F$W5	F$IO	1	57	5
	F$W6	O$MC	1		
	F$W7	C$8TO6	1		
	F$W8	O$MA	1		
	F$W9				
F$Wn	F$Wn	F$W1	1	41	5
		F$W2	1		
		F$W3	1		
		F$W4	1		
		F$W5	1		
		F$W6	1		
		F$W7	1		
		F$W8	1		
		F$W9	1		
FLOAT	FLOAT	C$12	1	10	4
H$22	H$22	ARG$	1	10	5
H$55	H$55	ARG$	1	20	3
		AC1	1		
		AC2	1		
		AC3	1		
		AC4	1		
H$66	H$66	ARG$	1	20	3
		AC1	1		
		AC2	1		
		AC3	1		
IABS	IABS			10	4
IDIM	IDIM			20	5
IDINT	See IFIX				
IFETCH	IFETCH	ARG$	1	10	4

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words_{10})	Tape Number
IFIX	IDINT	L$22	1	10	4
	INT	C$21	1		
	IFIX				
INT	See IFIX				
ISIGN	ISIGN			20	5
ISTORE	ISTORE	F$AT	1	10	4
L$22	REAL	ARG$	1	10	5
	L$22				
L$33	L$33			10	5
L$55	L$55	ARG$	1	20	4
		AC1	1		
		AC2	1		
		AC3	1		
		AC4	1		
L$66	L$66	ARG$	1	20	4
		AC1	1		
		AC2	1		
		AC3	1		
LOC	LOC			10	4
M$11	M$11	ARG$	1	110	5S
		F$ER	1		
M$11X	M$11	ARG$	1	50	5H
	M$11X	F$ER	1		
M$22	M$22	N$22	5	330	5S
	D$22	ARG$	2		
	DIV$	F$ER	3		
M$22X	M$22	F$ER	1	130	5H
M$52	M$52	F$AT	1	50	2
		H$55	1		
		SUB$	2		
		L$22	2		
		M$22	2		
		H$22	2		
		L$55	1		
M$55	M$55	F$AT	1	110	2
		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		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage ($Words_{10}$)	Tape Number
M$62	M$62	F$AT	1	20	1
		H$66	1		
		DBLE	1		
		M$66	1		
M$66	See A$66				35
M$66X	See A$66X				34
MAX0	AMAX0	FLOAT	1	40	3
	MAX0				
MAX1	AMAX1	L$22	2	50	3
	MAX1	H$22	2		
		S$22	1		
		IFIX	1		
MIN0	AMIN0	FLOAT	1	30	3
	MIN0				
MIN1	AMIN1	L$22	2	50	3
	MIN1	H$22	2		
		S$22	1		
		IFIX	1		
MOD	MOD	D$11	1	20	4
		M$11	1		
N$22	N$22			10	5
N$33	N$33			10	4
N$55	N$55	H$55	1	30	2
		SUB$	2		
		L$22	2		
		N$22	2		
		H$22	2		
		L$55	1		
N$66	N$66	AC1	1	30	4
		AC2	1		
		AC3	1		
OVERFL	OVERFL	AC5	1	20	5
REAL	See L$22				5
S$22	See A$22				5S
S$22X	See A$22X				
S$52	S$52	F$AT	1	30	2
		H$55	1		
		L$22	1		
		S$22	1		
		H$22	1		
		L$55	1		
S$55	S$55	F$AT	1	40	2
		H$55	1		
		SUB$	4		
		L$22	2		
		S$22	2		

Primary Name	Entry Points	Subroutines Called	Number of References	Approx. Storage (Words $_{10}$)	Tape Number
		N$22	2		
		H$22	2		
		L$55	1		
S$62	S$62	F$AT	1	20	1
		H$66	1		
		DBLE	1		
		S$66	1		
		N$66	1		
S$66	See A$66				3S
S$66X	See A$66X				3H
SIGN	SIGN	L$22	2	20	4
		N$22	1		
SIN	COS	ARG$	1	190	3
	SIN	N$22	2		
		M$22	7		
		S$22	1		
		A$22	4		
SLITE	SLITE	ARG$	3	70	5
	SLITET	L$33	1		
	SSWTCH				
SLITET	See SLITE				
SNGL	See C$62				
SQRT	SQRT	ARG$	1	70	3S
		DIV$	1		
		D$22	1		
		A$22			
		F$ER			
SQRTX	SQRT	ARG$	1	80	3H
	SQRTX	D$22	1		
		A$22	1		
		F$ER	1		
SSWTCH	See SLITE				
SUB$	SUB$	M$11	3	130	4
		F$ER	1		
TANH	TANH	L$22	1	60	3
		EXP	1		
		A$22	2		
		H$22	1		
		D$22	1		
Z$80	Z$80	AC1	1	20	3

APPENDIX E

ERROR MESSAGES

Error Message	Condition	Subroutine
AD	Over/underflow in double-precision	A$66, S$66, A$66X, S$66X
BF	End-of-file mark encountered while unit backspacing a record.	F$F5-9
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$81
EX	Exponential overflow during exponentiation	EXP
FE	Format error	F$IO
GO	Incorrect control variable in a GO TO statement	F$GA
II	First argument zero, second argument negative $I > 2$ and $J \geq 15$, or $I \leq -2$ and $J \geq 15$	E$11, E$11X
IM	Over/underflow during integer multiplication	M$11, M$11X
IN	Input error	F$AR
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 $\geq$ 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

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form*

TITLE: SERIES 16 FORTRAN MATH LIBRARY

ORDER No.: AM74, REV. 0

DATED: DECEMBER 1973

ERRORS IN PUBLICATION:

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

(Please Print)

FROM: NAME ____________________ DATE: ____________

COMPANY ____________________

TITLE ____________________

____

____

*Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply.

CUT ALONG LINE

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS,
MASS. 02181

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

POSTAGE WILL BE PAID BY:

HONEYWELL INFORMATION SYSTEMS
60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATTN: PUBLICATIONS, MS 050

FOLD ALONG LINE

Honeywell

The Other Computer Company:

Honeywell

HONEYWELL INFORMATION SYSTEMS

9125
11273
Printed in U.S.A.

In the U.S.A.: 200 Smith Street, MS 061, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario

AM74, Rev. 0