File No. S360-29
Order No. GC28-2046-1

TSS

IBM Systems Reference Library

IBM System/360 Time Sharing System

PL/I Library

Computational Subroutines

This publication gives details of the computational subroutines available in the PL/I Library. These subroutines are used by the PL/I compiler in the implementation of PL/I built-in functions and of the operators used in the evaluation of PL/I expressions. Not all PL/I built-in functions and expression operators are

PREFACE

This publication provides the PL/I user with detailed information about the computational subroutines which are part of the IBM System/360 Time Sharing System PL/I Library.

The reader is assumed to be a TSS/360 user with a particular concern for performance information associated with individual subroutines. The numerical analyst is provided with a description of the algorithms, and a specification of accuracy and range, where these are considered to be significant.

Useful background reading is provided in the following IBM publications:

IBM System/360 Principles of Operation, Order No. GA22-6821

IBM System/360 Time Sharing System:

Concepts and Facilities, Order No. GC28-2003

Assembler Language, Order No. GC28-2000

PL/I Reference Manual, Order No. GC28-2045

Second Edition (June 1970)

This is a major revision of, and makes obsolete, the previous edition, Form C28-2046-0.

This edition is current with Version 7, Modification 0, of IBM System/360 Time Sharing System (TSS/360) and will remain in effect for all subsequent versions or modifications of TSS/360. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to IBM Corporation, Programming Publications, Department 636, Neighborhood Road, Kingston, New York 12401

CONTENTS

FIGURES

The PL/I Library computational subroutines provide support for the operators and built-in functions of the PL/I language in four major categories:

1. Bit and Character Strings
2. Arithmetic
3. Mathematical
4. Arrays

This publication gives detailed information in each of the four sections mentioned above with respect to accuracy, choice of algorithm, and range of values handled (where appropriate).

A number of exceptional conditions may arise in the execution of the library subroutines. Many of these are not directly related to PL/I ON conditions. The method of treatment in these cases is to write a diagnostic message and raise the ERROR condition. This allows the user the opportunity to investigate the error by use of the ONCODE built-in function in his ON ERROR unit and to program the action he wants taken.

Module Names

The module name for each of these subroutines is IHEWxxx, where xxx is usually a mnemonic group indicating the module function.

The seventh character usually defines the base, scale, mode and precision of the arguments for a given module. In the arithmetic, mathematical and array subroutines, this suffix is usually one of the characters shown in Figure 1; the only exceptions to this are the array indexing subroutines, where the suffixes are mnemonic only, and the ALL(x), ANY(x) subroutines, where the suffix is 1 or 2.

In the string subroutines, the seventh character in each module name has only a mnemonic significance. In some cases the seventh character may be one of those given in Figure 1. This is purely coincidental; the meanings in Figure 1 do not apply to the string subroutines.

Seventh Character	Argument Attributes	Argument (or element of argument) Passed in	Maximum Precision
B	Real fixed-point binary	Fullword	31
D	Real fixed-point decimal	Up to 8 bytes	15
F	Real fixed-point binary or decimal	Binary: fullword Decimal: up to 8 bytes	Binary: 31 Decimal: 15
G	Real or complex short floating-point	Real: 1 fullword Complex: 2 fullwords	Binary: 21 Decimal: 6
H	Real or complex long floating-point	Real: 1 doubleword Complex: 2 doublewords	Binary: 53 Decimal: 16
L	Real long floating-point	Doubleword	Binary: 53 Decimal: 16
S	Real short floating-point	Fullword	Binary: 21 Decimal: 6
U	Complex fixed-point binary	2 fullwords	31
V	Complex fixed-point decimal	Up to 16 bytes	15
X	Complex fixed-point binary or decimal	Binary: 2 fullwords Decimal: Up to 16 bytes	Binary: 31 Decimal: 15
W	Complex short floating-point	2 fullwords	Binary: 21 Decimal: 6
Z	Complex long floating-point	2 doublewords	Binary: 53 Decimal: 16

Figure 1. Interpretation of Seventh Character in Module Names

The library string package contains modules for handling bit and character string operations. Generally, a string function or operator is supported by only one module, but in the interests of efficiency some of the bit string operators are provided with additional modules to deal with byte-aligned input data.

A complete list of the modules provided in the Library string package is given in Figure 2.

BIT STRING OPERATIONS

The 'And' Operator (&) (Bit Strings)

Module Name: IHEWBSA

Entry Point: IHEBSA0

Function:

To implement the 'and' operator between two byte-aligned bit strings, placing the result in a byte-aligned target field.

Method:

The current length of the target string is set equal either to the maximum length of the operands, or to the maximum length of the target field (when truncation is necessary to avoid exceeding the length of this field). The strings are 'and'ed together for a length equal to the minimum of the lengths of the operands, and the result is extended with zeros, if necessary, up to the current length calculated for the target field.

The 'Or' Operator (|)(Bit Strings)

Module Name: IHEWBSO

Entry Point: IHEBSO0

Function:

To implement the 'or' operation between two byte-aligned bit strings, placing the result in a byte-aligned target field.

Method:

The current length of the target string is set equal to either the maximum length of the operands or to the maximum length of the target field (when truncation is necessary to avoid exceeding the length of this field). The strings are 'or'ed together for a length equal to the minimum of the lengths of the operands and the remainder of the longer string is moved into the target field up to the current length; the remainder of the target field is left unchanged.

The 'Not' Operator (¬)(Bit Strings)

Module Name: IHEWBSN

Entry Point: IHEBSN0

Function:

To implement the 'not' operator for a byte-aligned bit string, placing the result in a byte-aligned target field.

PL/I Operation	PL/I Function	Bit String		Character String
		General	Byte-aligned	
'And' (&)	-	Use BOOL	IHEWBSA	-
'Or' (|)	-	Use BOOL	IHEWBSO	-
'Not' (¬)	-	Use BOOL	IHEWBSN	-
Concatenate (||)	REPEAT	IHEWBSK	-	IHEWCSK
Compare	-	IHEWBSD	IHEWBSC	IHEWCSC
Assign	-	IHEWBSK	IHEWBSM	IHEWCSM
Fill	-	IHEWBSM	-	IHEWCSM
-	HIGH/LOW	-	-	IHEWCSM
-	SUBSTR	IHEWBSS	-	IHEWCSS
-	INDEX	IHEWBSI	-	IHEWCSI
-	BOOL	IHEWBSF	-	-

Figure 2. Bit and Character String Operations and Functions

Method:

The current length of the target string is set equal to either the current length of the operand or to the maximum length of the target field (when truncation is necessary to avoid exceeding the length of this field). The target field is set to a string of 1's for a length equal to its calculated current length and the result is obtained by an 'exclusive or' with the operand. The remainder of the target field beyond the calculated current length is left unchanged.

Concatenate/REPEAT/General Assign (Bit Strings)

Module Name: IHEWBSK

Entry Points:

Operation	Entry Point
Concatenate (||)	IHEBSKK
REPEAT(Bit string, n)	IHEBSKR
General assign	IHEBSKA

Function:

IHEBSKK: to concatenate two bit strings into a target field.

IHEBSKR: to concatenate n + 1 instances of the single source string into a target field. If n ≤ 0, the result is the string itself.

IHEBSKA: to assign a bit string to a target field without zero filling.

Method:

The current length of the target field is made equal to the smaller of two values:

- the sum of the current lengths of the source strings

- the maximum length of the target field

All entry points use a subroutine that obtains data from a source, aligns it correctly and moves it to the target field:

IHEBSKK: uses this subroutine twice to move the source strings to the target field.

IHEBSKR: uses the subroutine to concatenate the contents of the target field with itself (whenever possible) as well as concatenating the contents of this field with the source string. Direct concatenation of the source string n + 1 times is not used.

IHEBSKA: Uses the subroutine once to move the source string to the target field.

For all entry points, the remainder of the target field beyond the calculated current length is left unaltered.

Comparison (Bit Strings, Byte-aligned)

Module Name: IHEWBSC

Entry Point: IHEBSC0

Function:

To compare two byte-aligned bit strings and to return a condition code as bits 2 and 3 of a fullword target field as follows:

00 if strings are equal
01 if first string compares low at the first inequality
10 if first string compares high at the first inequality

The shorter string is treated as though extended with zeros to the length of the longer.

The first byte of the target field is also used to preserve the program mask in the PSW for the calling routine. This byte contains:

Bits	Contents
0 to 1	Instruction length code 01
2 to 3	Condition code as above
4 to 7	Program mask (calling routine)

Method:

The two strings are compared up to the current length of the shorter string. The remainder of the longer string is compared with zeros.

General Comparison (Bit Strings)

Module Name: IHEWBSD

Entry Point: IHEBSD0

Function:

To compare two bit strings and return a condition code as bits 2 and 3 of a fullword target field as follows:

00 if strings are equal

01 if first string compares low at the first inequality

10 if first string compares high at the first inequality

The shorter string is treated as though extended with zeros to the length of the longer.

The first byte of the target field is also used to preserve the program mask in the PSW for the calling routine. This byte contains:

Bits	Contents
0 to 1	Instruction length code 01
2 to 3	Condition code as above
4 to 7	Program mask (calling routine)

Method:

The two strings are compared up to the current length of the shorter string. The remainder of the longer string is compared with zeros.

Assign/Fill (Bit Strings)

Module Name: IHEWBSM

Entry Points:

Operation	Entry Point
Fixed-length assign	IHEBSMF
Variable-length assign	IHEBSMV
Zero fill only	IHEBSMZ

Function:

IHEBSMF: to assign a byte-aligned string to a byte-aligned fixed-length target, filling out with zero bits if necessary.

IHEBSMV: to assign a byte-aligned string to a byte-aligned variable-length target.

IHEBSMZ: to fill out the target area from its current length to its maximum length with zero bits.

Method:

IHEBSMF: the minimum of the source current length and the target maximum length is calculated and the source string is moved to the target for a length equal to this length. Zero filling of the target is performed if necessary. The current length of the target is set equal to the maximum length.

IHEBSMV: the source string is moved to the target field as above, but without zero filling. The current length of the target is set appropriately.

IHEBSMZ: zeros are propagated in the target from the current length to the maximum length. The current length of the target is set equal to the maximum length.

Other Information:

This routine supplies assignment of byte-aligned bit strings of both fixed and variable lengths. Non-aligned strings may be assigned by using the general assign module (entry point IHEBSKA). Any filling required for fixed length strings can then be obtained using the IHEBSMZ entry described above.

BIT STRING FUNCTIONS

SUBSTR (Bit Strings)

Module Name: IHEWBSS

Entry Points:

Operation	Entry Point
SUBSTR(Bit-string,i)	IHEBSS2
SUBSTR(Bit-string,i,j)	IHEBSS3

Function:

To produce a string dope vector describing the SUBSTR pseudo-variable and function of a bit-string.

Method:

Arithmetic is performed according to the function definition, using the current length of the argument string. The result describes a fixed-length string.

Error and Exceptional Conditions:

STRINGRANGE

INDEX (Bit Strings)

Module Name: IHEWBSI

Entry Point: IHEBSIO

Function:

To compare two bit strings to see if the second is identical to a substring of the first, and, if it is, to produce a binary integer (the index) which indicates the first bit position in the first string at which such a substring begins. If no such index is found, or if either string is null, the function value is zero.

Method:

The index is found by shifting and comparing portions of the two strings in registers.

BOOL (Boolean Function)(Bit Strings)

Module Name: IHEWBSF

Entry Point: IHEBSF0

Function:

To take two source strings and perform one of the sixteen possible logical operations between corresponding bits. The particular operation performed is defined by inserting the bit pattern - $n_1n_2n_3n$ - yielded by the third argument into the table below:

First field	0	0	1	1
Second field	0	1	0	1
Target field	n_1	n_2	n_3	n_4

Method:

The current length of the target string is set equal to either the maximum of the current lengths of the source strings or to the maximum length of the target field (when truncation is necessary to avoid exceeding the length of this field). The necessary operation is performed on the strings and the result stored in the target field. If one string is shorter than the other, it is regarded as being extended on the right with zeros up to the length of the longer. The field between the calculated current length and the maximum length of the target is left unchanged.

CHARACTER STRING OPERATIONS

Concatenate/REPEAT (Character Strings)

Module Name: IHEWCSK

Entry Points:

Operation	Entry Point
Concatenate (||)	IHECSKK
REPEAT (Character string,n)	IHECSKR

Function:

IHECSKK: to concatenate two character strings into a target field.

IHECSKR: to concatenate n + 1 instances of the single source string into a target field. If $n \leq 0$, the result is the string itself.

Method:

The current length of the target field is made equal to the smaller of two values:

- the sum of the current lengths of the source fields.
- the maximum length of the target field.

Both entry points use a subroutine that moves characters from a source to the target:

IHECSKK: Uses the subroutine to perform the required number of source moves.

IHECSKR: Uses the subroutine to concatenate the source string with the target field and also to concatenate the target field with itself (whenever possible).

For both entry points, characters beyond the range of the target current length remain unaltered.

Compare (Character Strings)

Module Name: IHEWCSC

Entry Point: IHECSC0

Function:

To compare two character strings and to return a condition code as bits 2 and 3 of a fullword target field as follows:

00 if strings are equal
01 if first string compares low at the first inequality
10 if the first string compares high at the first inequality

The shorter string is treated as though extended with blanks to the length of the longer one.

The first byte of the target field is also used to preserve the program mask in the PSW for the calling routine. This byte contains:

Bits	Contents
0 to 1	Instruction length code 01
2 to 3	Condition code as above
4 to 7	Program mask (calling routine)

Method:

The two strings are compared in storage. If the strings are of different lengths and are identical up to the length of the shorter, the remainder of the longer is compared with blanks.

Assign/Fill/HIGH/LOW (Character Strings)

Module Name: IHEWCSM

Entry Points:

Operation	Entry Point
Fixed-length assign	IHECSMF
Variable-length assign	IHECSMV
Blank fill only	IHECSMB
HIGH	IHECSMH
LOW	IHECSML

Function:

IHECSMF: to assign a character string to a fixed-length target, filling out with blanks if necessary.

IHECSMV: to assign a character string to a variable-length target.

IHECSMB: to fill out the target field from its current length to its maximum length with blanks.

IHECSMH: to fill a target field with the highest character in the collating sequence, up to its current length.

IHECSML: to fill the target field with the lowest character in the collating sequence, up to its current length.

Method:

IHECSMF: The minimum of the source current length and the target maximum length is calculated and the source string is moved to the target for a length equal to this length. Filling of the target with blanks up to the target maximum length is performed if necessary. The current length of the target is set equal to its maximum length.

IHECSMV: moves the string as above, but without blank filling. The current length of the target is set appropriately.

IHECSMB: propagates blanks and sets the current length of the target equal to its maximum length.

IHECSMH, IHECSML: uses part of the blank fill routine to propagate the highest or lowest character in the collating sequence up to the current length of the target.

CHARACTER STRING FUNCTIONS

SUBSTR (Character Strings)

Module Name: IHEWCSS

Operation	Entry Point
SUBSTR(Character-string,i)	IHECSS2
SUBSTR(Character-string,i,j)	IHECSS3

Function:

To produce a string dope vector describing the SUBSTR pseudo-variable and function of a character string.

Method:

Arithmetic is performed according to the function definition, using the current length of the argument string. The result describes a fixed-length string.

Error and Exceptional Conditions:

STRINGRANGE

INDEX (Character Strings)

Module Name: IHEWCSI

Entry Point: IHECSIO

Function:

To compare two character strings to see if the second is identical to a substring of the first, and, if it is,to produce a binary integer (the index) which indicates the first character position in the first string at which such a substring begins. If no such index is found, or if either string is null, the function value is zero.

Method:

The first string is scanned from left to right for a character equal to the first character in the second string. If a match is found, the whole of the second string is compared with a substring of the first string beginning at the matching character. If they are equal, an index is produced. The scanning continues until either an index is produced or the end of the first string is reached.

Library arithmetic modules support all those arithmetic generic functions and operators for which the compilers neither produce in-line code nor (as for the functions FIXED, FLOAT, BINARY and DECIMAL) use the conversion package. The names of the library modules which support the arithmetic operations are given in Figure 3; the names of those which support the arithmetic functions are given in Figure 4.

Statistics for accuracy of floating-point modules are given where considered meaningful and helpful; an explanation of their use is given in the chapter on mathematical routines. Precise results are obtained from all fixed-point modules except complex division and complex ABS, where small truncation errors inevitably occur, and the ADD function (fixed decimal), in which the effect of truncation errors depends on the relative values of the scale factors of the arguments.

Any restrictions on the admissibility of arguments are noted under the headings 'Range' and 'Error and Exceptional Conditions'.

Range: This states any ranges of arguments for which a module is valid. Arguments outside the ranges given are assumed to have been excluded before the module is called.

Error and Exceptional Conditions: These cover conditions which may result from the use of a routine; they are listed in four categories:

P -- Programmed conditions in the module concerned. Programmed tests are made where this is not too costly and, if an invalid argument is found, a branch is taken to the entry point IHEERRC of the execution error package (EXEP). This results in the printing of an appropriate message and in the ERROR condition being raised.

I -- Interruption conditions in the module concerned. For those routines where SIZE and FIXEDOVERFLOW are detected by programmed tests or where hardware interruptions may occur, the OVERFLOW, UNDERFLOW, FIXEDOVERFLOW, SIZE and ZERODIVIDE conditions pass to the ON handler (IHEERR) and are treated in the normal way. The machine is assumed to be enabled for all interruptions except significance, which is masked off.

O -- Programmed conditions in modules called by the module concerned. These occur when invalid arguments are detected in the module called.

H -- As I, but the interruption conditions occur in the modules called by the module concerned.

ARITHMETIC OPERATIONS				
Operation	Binary fixed	Decimal fixed	Short float	Long float
Real Operations				
Integer exponentiation: x**n	IHEWXIB	IHEWXID	IHEWXIS	IHEWXIL
General exponentiation: x**y	-	-	IHEWXXS	IHEWXXL
Shift-and-assign, Shift-and-load	-	IHEWAPD	-	-
Complex Operations				
Multiplication/division: $z_1*z_2, z_1/z_2$	IHEWMZU	IHEWMZV	-	-
Multiplication: z_1*z_2	-	-	IHEWMZW	IHEWMZZ
Division: z_1/z_2	-	-	IHEWDZW	IHEWDZZ
Integer exponentiation: z**n	IHEWXIU	IHEWXIV	IHEWXIW	IHEWXIZ
General exponentiation: z_1**z_2	-	-	IHEWXXW	IHEWXXZ

Figure 3. Arithmetic Operations

ARITHMETIC FUNCTIONS				
Function	Binary fixed	Decimal fixed	Short float	Long float
Real Arguments				
MAX, MIN	IHEWMXB	IHEWMXD	IHEWMXS	IHEWMXL
ADD	-	IHEWADD	-	-
Complex Arguments				
ADD	-	IHEWADV	-	-
MULTIPLY	IHEWMPU	IHEWMPV	-	-
DIVIDE	IHEWDVU	IHEWDVV	-	-
ABS	IHEWABU	IHEWABV	IHEWABW	IHEWABZ

Figure 4. Arithmetic Functions

REAL OPERATIONS

Positive Integer Exponentiation (fixed binary)

Module Name: IHEWXIB

Entry Point: IHEXIB0

Function:

To calculate x**n, where n is a positive integer.

Method:

The result is set initially to the value of the argument. The final result is then obtained by repeated squaring of this value or squaring and multiplying by the argument.

Range:

$0 < n < 2^{31}$

The precision rules of PL/I impose a further restriction in that if x has a precision (p,q), this module will be called only if $n*(p + 1) - 1 \le 31$. This implies that $n \le 32/(p + 1) \le 16$ for all such cases.

Positive Integer Exponentiation (fixed decimal)

Module Name: IHEWXID

Entry Point: IHEXID0

Function:

To calculate x**n, where n is a positive integer.

Method:

The result is set initially to the value of the argument. The final result is then obtained by repeated squaring of this value or squaring and multiplying by the argument.

Range:

The precision rules of PL/I impose the restriction that if x has a precision (p,q), this module will be called only if $n*(p + 1) - 1 \le 15$. This implies that $n \le 16/(p + 1) \le 8$ for all such cases and, in fact, this module will operate only for the range $0 < n \le 8$.

Integer Exponentiation (floating-point)

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWXIS	IHEXIS0
Long float	IHEWXIL	IHEXIL0

Function:

To calculate x**n, where n is an integer between -2**31 and 2**31 - 1 inclusive.

Method:

If the exponent is zero and the argument nonzero, the result 1 is returned immediately. Otherwise the result is set initially to the value of the argument and the exponent is made positive. The argument is raised to this positive power by repeated squaring of the contents of the result field or squaring and multiplying by the argument. Then, if the exponent is negative, the reciprocal of the result is taken, otherwise it is left unchanged.

Accuracy:

The values given here are for the relative error divided by the exponent for exponents between 2 and 1023; the arguments are uniformly distributed over the full range for each exponent for which neither OVERFLOW nor UNDERFLOW occurs. There are 2**(10 - k) arguments for each exponent in the range $2^{k} \le$ exponent $\le 2^{(k+1)} - 1$, where k has integral values from 1 to 9 inclusive.

IHEWXIS

R.M.S. relative error/exponent *10**6	Maximum relative error/exponent *10**6
0.00871	0.692

IHEWXIL

R.M.S. relative error/exponent *10**15	Maximum relative error/exponent *10**15
0.0995	1.73

Error and Exceptional Conditions:

P : x = 0 with n ≤ 0

I : OVERFLOW, UNDERFLOW
Since x**(-m), where m is a positive integer, is evaluated as 1/(x**m), the OVERFLOW condition may occur when m is large, and the UNDERFLOW condition when x is very small.

Other Information:

IHEWXIS: For large exponents, for example, those greater than 1023, it is generally faster and more accurate to use the module IHEWXXS rather than IHEWXIS, passing the exponent as a floating-point argument. However, it should be noted that IHEWXXS will not accept a negative first argument, and thus it is necessary to pass the absolute value of this argument, and also, in cases where the exponent is odd, to test the sign of the argument in order to be able to attach the correct sign to the numerical result returned.

General Floating-Point Exponentiation

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWXXS	IHEXXS0
Long float	IHEWXXL	IHEXXL0

Function:

To calculate x**y, where x and y are floating-point numbers.

Method:

When x = 0, the result x**y = 0 is given if y > 0, and an error message if y ≤ 0. When x ≠ 0 and y = 0, the result x**y = 1 is given. Otherwise x**y is computed as EXP(y*LOG(x)), using the appropriate mathematical function routines.

Error and Exceptional Conditions:

P : x = 0 with y ≤ 0

O : a. x < 0 with y ≠ 0 : error caused in LOG routine

b. y*LOG(x) > 174.673: error caused in EXP routine

Shift-and-assign, Shift-and-load (fixed decimal)

Module Name: IHEWAPD

Entry Points:

Operation	Entry Point
Shift and assign	IHEAPDA
Shift and load	IHEAPDB

Function:

IHEAPDA: To convert a real fixed decimal number with precision (p_1,q_1) to precision (p_2,q_2), where $p_1 \leq 31$ and $p_2 \leq 15$.

IHEAPDB: To convert a real fixed decimal number with precision (p_1,q_1) to precision $(31,q_2)$, where $p_1 \leq 31$.

Method:

The argument scale factor is subtracted from the target scale factor. The argument is converted to precision 31 in a field with a shift equal to the magnitude of the difference between the scale factors; the shift is to the left if the difference is positive and to the right if negative.

If entry point IHEAPDB is used, the field is moved unchanged to the target. If entry point IHEAPDA is used, the result is checked for FIXEDOVERFLOW and then assigned to the target with the specified precision. The assignment may cause the SIZE condition to be raised.

Error and Exceptional Conditions:

I : FIXEDOVERFLOW or SIZE

COMPLEX OPERATIONS

Multiplication/Division (fixed binary)

Module Name: IHEWMZU

Entry Points:

Mathematical Operation	Entry Point
z_1*z_2	IHEMZUM
z_1/z_2	IHEMZUD

Function:

To calculate z_1*z_2 or z_1/z_2, where z_1 and z_2 are fixed-point binary complex numbers.

Method:

Let $z_1 = a + bI$ and $z_2 = c + dI$. Then, for multiplication, an incorporated subroutine is used to compute a*c - b*d and b*c + a*d; these are tested for FIXEDOVERFLOW and then stored as the real and imaginary parts of the result.

For division, the subroutine is used to compute a*c + b*d and b*c - a*d. The expression c**2 + d**2 is computed and the real and imaginary parts of the result are then obtained by division.

The subroutine computes the expressions u*x + v*y and v*x - u*y.

Error and Exceptional Conditions:

I : FIXEDOVERFLOW in either routine, ZERODIVIDE in the division routine.

Multiplication/Division (fixed decimal)

Module Name: IHEWMZV

Entry Points:

Mathematical Operation	Entry Point
z_1*z_2	IHEMZVM
z_1/z_2	IHEMZVD

Function:

To calculate z_1*z_2 or z_1/z_2 where z_1 and z_2 are fixed-point decimal complex numbers.

Method:

Let $z_1 = a + bI$ and $z_2 = c + dI$. The products a*c, b*c, a*d and b*d are computed. Then the required result is obtained as follows:

Multiplication:

Real part	a*c - b*d
Imaginary part	b*c + a*d

Division:

Real part	(a*c + b*d)/(c*c + d*d)
Imaginary part	(b*c - a*d)/(c*c + d*d)

Error and Exceptional Conditions:

I : FIXEDOVERFLOW in either routine, ZERODIVIDE in the division routine.

Other Information:

Where the operands differ in precision, it is faster to present the longer operand as the second argument rather than the first.

Multiplication (floating-point)

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWMZW	IHEMZW0
Long float	IHEWMZZ	IHEMZZ0

Function:

To compute z_1*z_2 in floating-point, when $z_1 = a + bI$ and $z_2 = c + dI$.

Method:

The real and imaginary parts of the result are computed as a*c - b*d and b*c + a*d, respectively.

Error and Exceptional Conditions:

I : Exponent OVERFLOW and UNDERFLOW

Division (floating-point)

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWDZW	IHEDZW0
Long float	IHEWDZZ	IHEDZZ0

Function:

To compute z_1/z_2 in floating-point, when $z_1 = a + bI$ and $z_2 = c + dI$.

Method:

1. ABS(c) ≥ ABS(d)

Compute q = d/c
then REAL (z_1/z_2) = (a + b*q)/(c + d*q)
IMAG (z_1/z_2) = (b - a*q)/(c + d*q)

2. ABS(c) < ABS(d)

(a + bI)/(c + dI) = (b - aI)/(d - cI), which reduces to the first case.

The comparison between ABS(c) and ABS(d) is adequately performed in short precision in both modules.

Error and Exceptional Conditions:

I : OVERFLOW, UNDERFLOW and ZERODIVIDE

Positive Integer Exponentiation (fixed binary)

Module Name: IHEWXIU

Entry Point: IHEXIU0

Function:

To calculate z**n, where n is a positive integer less than 2**31.

Method:

The contents of the target field are set to the value of z. The final result is obtained by repeated squaring of the contents of the target field or squaring and multiplying by z. Multiplication is performed by the complex multiplication routine IHEWMZU.

Range:

$0 < n < 2^{31}$.

The precision rules of PL/I impose a further restriction in that if z has a precision (p, q), this module may only be called if $n*(p + 1) - 1 \le 31$. This implies that $n \le 32/(p + 1) \le 16$ for all such cases.

Positive Integer Exponentiation (fixed decimal)

Module Name: IHEWXIV

Entry Point: IHEXIV0

Function:

To calculate z**n, where n is a positive integer less than 2**31.

Method:

The contents of the target field are set to the value of the argument. The final result is obtained by repeated squaring of the contents of the target field or squaring and multiplying by the argument. Multiplication is performed by the complex multiplication routine IHEWMZV.

Range:

The precision rules of PL/I impose the restriction that if z has a precision (p,q), this module may only be called if $n*(p + 1) - 1 \le 15$. This implies that $n \le 16/(p + 1) \le 8$ for all such cases and, in fact, this module will operate only for the range $0 < n \le 8$.

Integer Exponentiation (floating-point)

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWXIW	IHEXIW0
Long float	IHEWXIZ	IHEXIZ0

Function:

To calculate z**n, where n is an integer between -2**31 and 2**31 - 1 inclusive.

Method:

If the exponent is 0 and the argument non-zero, the result 1 is returned immediately. If the exponent is non-zero, the contents of the target field are set to the argument value. The exponent is made positive and the argument raised to this positive power by repeated squaring of the contents of the target field or squaring and multiplying by the argument. Multiplication is performed by a branch to the complex multiplication subroutine. Then, if the exponent was negative, the reciprocal of the result is taken, otherwise it is left unchanged.

Error and Exceptional Conditions:

P : $z = 0$ with $n \le 0$

I : OVERFLOW, UNDERFLOW
Since x**(-m), where m is a positive integer, is evaluated as 1/(x**m), the OVERFLOW condition may occur when m is large and the UNDERFLOW condition when x is very small.

H : OVERFLOW or UNDERFLOW in complex multiplication routine (IHEWMZW or IHEWMZZ)

General Floating-Point Exponentiation

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWXXW	IHEXXW0
Long float	IHEWXXZ	IHEXXZ0

Function:

To calculate z_1**z_2, where z_1 and z_2 are complex numbers of the same precision.

Method:

When $z_1 = 0$, the result 0 is returned if REAL(z_2) > 0 and IMAG(z_2) = 0. Otherwise, z_1**z_2 is computed as

EXP(z_2*LOG(z_1)),

with the proviso that if IMAG(z_1) = 0 then LOG(ABS(z_1)) is calculated by a call to the real LOG routine, not to the complex LOG routine.

Error and Exceptional Conditions:

P : z_1 = 0 with either REAL(z_2) ≤ 0 or IMAG(z_2) ≠ 0

O : a. REAL(z_2*LOG(z_1)) > 174.673: error caused in IHEWEXS or IHEWEXL

b. IHEWXXW: ABS(IMAG(z_2*LOG(z_1))) ≥ 2**18*pi: error caused in SIN routine (IHEWSNS)

IHEWXXZ: ABS(IMAG(z_2*LOG(z_1))) ≥ 2**50*pi: error caused in SIN routine (IHEWSNL)

FUNCTIONS WITH REAL ARGUMENTS

ADD (Fixed decimal)

Module Name: IHEWADD

Entry Point: IHEADD0

Function:

ADD(x_1,x_2,p,q) where x_1 and x_2 are real fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method:

If both arguments are non-zero, a call to the module IHEWAPD is used to shift the one with the larger scale factor to give it the scale factor of the other, and convert it to precision 31. The arguments are added together, and IHEWAPD is used to convert the sum to the specified precision and to assign it to the target field.

If one of the arguments is zero, the other is treated as the sum above.

Error and Exceptional Conditions:

H : FIXEDOVERFLOW or SIZE may occur in IHEWAPD.

MAX,MIN

Module Names and Entry Points:

Argument	PL/I Function	Module Name	Entry Point
Fixed binary	MAX	IHEWMXB	IHEMXBX
	MIN		IHEMXBN
Fixed decimal	MAX	IHEWMXD	IHEMXDX
	MIN		IHEMXDN
Short float	MAX	IHEWMXS	IHEMXSX
	MIN		IHEMXSN
Long float	MAX	IHEWMXL	IHEMXLX
	MIN		IHEMXLN

Function:

To find the maximum or the minimum of a group of arithmetic values.

All arguments must have the same base, scale and precision.

Method:

IHEWMXB,IHEWMXS,IHEWMXL: The value of the current maximum or minimum is set to the value of the first argument; it is then compared algebraically with the next argument and replaced by it if appropriate. The process is repeated until a test on the argument list indicates that all source items have been processed, when the current value is stored as the result.

IHEWMXD: The address of the current maximum or minimum is set to the address of the first argument; this argument is then compared algebraically with the next argument, and the address of the latter replaces that of the former if appropriate. The process is repeated until a test on the argument list indicates that all source items have been processed, when the result is moved into the target field.

FUNCTIONS WITH COMPLEX ARGUMENTS

ADD (Fixed decimal)

Module Name: IHEWADV

Entry Point: IHEADV0

Function:

ADD(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method:

The real parts of each argument are added and the sum is assigned to the target field by using the real fixed decimal ADD module (IHEWADD). The imaginary parts are treated similarly.

Error and Exceptional Conditions:

H : FIXEDOVERFLOW or SIZE may occur in IHEWAPD.

MULTIPLY (fixed binary)

Module Name: IHEWMPU

Entry Point: IHEMPU0

Function:

MULTIPLY(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point binary numbers, and (p,q) is the required precision of the result.

Method:

Let the arguments be z_1 = a + bI and z_2 = c + dI.

Then REAL(z_1*z_2) = a*c - b*d
IMAG(z_1*z_2) = b*c + a*d

The real and imaginary parts of the product are computed. These numbers are then shifted to give them the required scale factor(q).

The results of the shifts are tested for FIXEDOVERFLOW and truncated by left shifts.

Error and Exceptional Conditions:

I : FIXEDOVERFLOW

MULTIPLY (fixed decimal)

Module Name: IHEWMPV

Entry Point: IHEMPV0

Function:

MULTIPLY(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method:

Let z_1 = a + bI and z_2 = c + dI, then:

REAL(z_1*z_2) = a*c - b*d.
IMAG(z_1*z_2) = b*c + a*d.

The real and imaginary parts are calculated and then each is assigned to the target with precision (p,q) by separate calls to the entry point IHEAPDA of the decimal shift and assign module IHEWAPD.

Error and Exceptional Conditions:

H : FIXEDOVERFLOW or SIZE in IHEWAPD.

DIVIDE (fixed binary)

Module Name: IHEWDVU

Entry Point: IHEDVU0

Function:

DIVIDE(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point binary numbers, and (p,q) is the required precision of the result.

Method:

Let z_1 = a + bI, and z_2 = c + dI, then:

REAL(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)
IMAG(z_1/z_2) = (b*c - a*d)/(c**2 + d**2)

The expressions a*c + b*d, b*c - a*d, and c**2 + d**2 are computed with a precision of 63. The denominator, c**2 + d**2 is shifted to precision 31 by either a right or left shift.

Two calls are then made to an incorporated subroutine which accepts a numerator and shifts it so that it has two insignificant leading digits. It then divides by c**2 + d**2 and shifts the quotient to the required scale factor (q).

Error and Exceptional Conditions:

I : FIXEDOVERFLOW or ZERODIVIDE

DIVIDE (fixed decimal)

Module Name: IHEWDVV

Entry Point: IHEDVV0

Function:

DIVIDE(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method:

Let z_1 = a + bI, and z_2 = c + dI, then

REAL(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)
IMAG(z_1/z_2) = (b*c - a*d)/(c**2 + d**2)

The expressions a*c + b*d, b*c - a*d, and c**2 + d**2 are computed. Leading zeros are removed from the denominator (c**2 + d**2) by truncation on the left and a left shift if necessary. If the denominator is still more than 15 digits long it is truncated on the right to 15 digits.

Two calls are then made to an incorporated subroutine which accepts a numerator and shifts it to precision 31 with 2 leading zeros by calling IHEWAPD (via entry point IHEAPDB). It then divides by c**2 + d**2 and calls IHEWAPD (via entry point IHEAPDA) to assign the quotient to the target field with the required precision (p,q).

Error and Exceptional Conditions:

I : ZERODIVIDE

H : FIXEDOVERFLOW or SIZE in IHEWAPD

ABS (fixed binary)

Module Name: IHEWABU

Entry Point: IHEABU0

Function:

To calculate ABS(z) = SQRT(x**2 + y**2), where z = x + yI.

Method:

If x = y, result is x*SQRT(2). Otherwise,

let X1 = MAX(ABS(x),ABS(y))

Y1 = MIN(ABS(x),ABS(y)).

Then ABS(z) is computed as

X1*SQRT(1 + (Y1/X1)**2),

where the fixed binary calculation of SQRT(g) for $1 \le g < 2$ is included within the module.

The first approximation to the square root is taken as

g/(1+g) + (1+g)/4,

with maximum relative error 1.8*2**-10. One Newton-Raphson iteration gives maximum relative error 1.6*2**-20, and suffices if X1 < 2**(15-q) where q is the scale factor of z.

Otherwise a second iteration is used, with theoretical maximum relative error of 1.3*2**-40.

Error and Exceptional Conditions:

I : FIXEDOVERFLOW

ABS (fixed decimal)

Module Name: IHEWABV

Entry Point: IHEABV0

Function:

To calculate ABS(z) = SQRT (x**2 + y**2) where z = x + yI.

Method:

x and y are converted to binary, with appropriate scaling if either exceeds 9 significant decimal digits.

Let X1 be the maximum, and Y1 the minimum, of the absolute values of the two binary numbers thus obtained.

Then if X1 = Y1 = 0, result 0 is returned. Otherwise, an approximation to ABS(z) is computed as

X1*SQRT(1 + (Y1/X1)**2),

where the fixed binary calculation of SQRT(g) for $1 \le g \le 2$ is included within the module.

The first approximation to the square root is taken in the form

A + B*(1 + g) - A/(1 + g)

with maximum relative error 2.17*10**-4, and one Newton-Raphson iteration then gives a value with maximum relative error 2.35*10**-8.

Multiplication by X1 produces a value for ABS(z) which is rounded and converted to decimal, and this suffices if it has not more than 7 significant decimal digits. Otherwise, this approximation is scaled if necessary and used in a final Newton-Raphson iteration for SQRT(x**2 + y**2) in decimal, with theoretical maximum relative error 2.76*10**-16.

Error and Exceptional Conditions:

I : FIXEDOVERFLOW

ABS (floating-point)

Module Names and Entry Points:

Argument	Module Name	Entry Point
Short float	IHEWABW	IHEABW0
Long float	IHEWABZ	IHEABZ0

Function:

To calculate ABS(z) = SQRT (x**2 + y**2), where z = x + yI.

Method:

Let $z = x + yI$. If $x = y = 0$, answer is 0.

Otherwise let z_1 = MAX(ABS(x),ABS(y))
and z_2 = MIN(ABS(x),ABS(y)).

Then the answer is computed as

$$ABS(z) = z_1 * SQRT(1 + (z_2/z_1)**2).$$

Accuracy:

IHEWABW

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
Full range	Exponential radially, uniform round origin	0.833	2.02

IHEWABZ

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
Full range	Exponential radially, uniform round origin	0.828	3.38

Error and Exceptional Conditions:

I : OVERFLOW

The Library supports all floating point arithmetic generic functions and has separate modules for short and long precision real arguments. Additionally, the Library has separate modules for short and long precision complex arguments where these are admissible.

Since the calling sequence generated in compiled code is the same as that required for passing the same arguments to a PL/I procedure, it is permissible to pass the names of any of the float arithmetic generic functions as arguments between procedures, according to the normal rules for entry names.

Any restrictions on the admissibility of arguments are noted under the heading 'Error and Exceptional Conditions.'

Error and Exceptional Conditions: These cover conditions which may result from the use of a routine; they are listed in four categories:

P -- Programmed conditions in the module concerned. Programmed tests are made where this is not too costly and, if an invalid argument is found, a branch is taken to the entry point IHEERRC of the execution error package (EXEP). This results in the printing of an appropriate message and in the ERROR condition being raised.

I -- Interruption conditions in the module concerned. For those routines where an OVERFLOW interruption may occur, the condition is passed to the ON condition error handler (IHEWERR) and is treated in the normal way. For those routines where an UNDERFLOW may occur, the condition is disabled and both intermediate and terminal underflows are accepted as true zero. In certain circumstances, however, where intermediate underflow may cause severe deterioration in the accuracy of the result, the condition is avoided by programmed tests.

O -- Programmed conditions in modules called by the module concerned. These occur when invalid arguments are detected in the module called.

H -- As I, but the interruption conditions occur in the modules called by the module concerned.

Accuracy

In order to appreciate properly the meaning of the statistics for accuracy given with each module, some consideration of the limits and implications of these statistics is required. Because the size of a machine word is limited, small errors may be generated by mathematical routines. In an elaborate computation, slight inaccuracies can accumulate and become large errors. Thus, in interpreting final results, errors introduced during the various intermediate stages must be taken into account.

The accuracy of an answer produced by a routine is influenced by two factors: (1) the accuracy of the argument and (2) the performance of the routine.

Most arguments contain errors. An error in a given argument may have accumulated over several steps prior to the use of the routine. Even data fresh from input conversion may contain slight errors. The effect of an argument error on the accuracy of an answer depends solely on the nature of the mathematical function involved and not on the particular coding by which that function is computed within a routine. In order to assist users in assessing the accumulation of errors, a guide on the propagational effect of argument errors is provided for each function. Wherever possible, this is expressed as a simple formula.

The performance statistics supplied in this document are based upon the assumption that the arguments are perfect (i.e., without errors, and therefore having no argument error propagation effect upon answers). Thus the only errors in answers are those introduced by the routines themselves.

For each routine, accuracy figures are given for the valid argument range or for representative segments of this. In each case the particular statistics given are those most meaningful to the function and range under consideration.

For example, the maximum relative error and the root-mean-square of the relative error of a set of answers are generally useful and revealing statistics, but are useless for the range of a function where its value becomes 0, since the slightest error of the argument value can cause an unbounded fluctuation in the relative magnitude of the answer. Such is the case with SIN(x) for values of x close to pi; in this range it is more appropriate to discuss absolute errors.

The results were derived from random distributions of 5000 arguments per segment, generated to be either uniform or exponential, as appropriate. It must be emphasized that each value quoted for the maximum error refers to a particular test using the method described above, and should be treated only as a guide to the true maximum error.

This explains, for example, why it is possible that the maximum error quoted for a segment may be greater than that found from a distribution of different arguments over a larger range which includes that segment.

Hexadecimal Truncation Errors

While the use of hexadecimal numbers in System/360 has led to increased efficiency and flexibility, the effect of the variable number of significant digits carried by the floating-point registers must be noted in making allowance for truncation errors. In the production of the PL/I Library, special care was taken to minimize such errors, whenever this could be accomplished at minor cost. As a result, the relative errors produced by some of the Library routines may be considerably smaller than the relative error produced in some instances by a single operation such as multiplication.

Representations of finite length entail truncation errors in any number system. With binary normalization, the effect of truncation is roughly uniform. With hexadecimal normalization, however, the effect varies by a factor of 16 depending on the size of the mantissa; in a chain of computations, the worst error committed in the chain usually prevails at the end.

In short-precision representation, a number has between 21 and 24 significant binary digits. Therefore, the truncation errors range from 2**-24 to 2**-20 (5.96*10**-8 to 9.5*10**-7). Assuming exact operands, a product or quotient is correct to the 24th binary digit of the mantissa. Hence truncation errors contributed by multiplication or division are no more than 2**-20. The same is true for the sum of two operands of the same sign. Subtraction, on the other hand, is the commonest cause of loss of significant digits in any number system. For short-precision operations, therefore, a guard digit is provided which helps to reduce such loss.

In long-precision representation, a number has between 53 and 56 significant binary digits. Therefore truncation errors range from 2**-56 to 2**-52 (1.39*10**-17 to 2.22*10**-16).

Normal care in numerical analysis should be exercised for addition and subtraction. In particular, when two algorithms are theoretically equivalent, it usually pays to choose the one which avoids subtraction between operands of similar size.

Hexadecimal Constants

Many of the modules described below discriminate between algorithms or test for errors by comparisons involving hexadecimal constants; it must be realized that where decimal fractions are used in the descriptions the fractions are only quoted as convenient approximations to the hexadecimal values actually employed.

Algorithms

The algorithms are the methods by which the mathematical functions are computed. The presentation of each algorithm is divided into its major computational steps, with the formulas necessary for each step supplied. Some of the formulas are widely known; those that are not so widely known are derived from more common formulas. The process leading from the common formula to the computational formula is sketched in enough detail so that the derivation can be reconstructed by anyone who has an understanding of college mathematics and access to the common texts on numerical analysis.[1]

Many of the approximations were derived by the so-called "minimax" methods. The approximation sought by these methods can be characterized as follows: given a function f(x), an interval I, the form of the approximation (such as the rational form with specified degrees), and the type of error to be minimized (such as the relative error), there is normally a unique approximation to f(x) whose maximum error over I is the smallest among all possible approximations of the given form. Details of the theory and the various methods of deriving such approximations are left to the reference.

----

[1]Any of the modern numerical texts may be used as a reference. One such text is A. Ralston's A First Course in Numerical Analysis (McGraw-Hill Book Company, Inc., New York, 1965). Background information for algorithms that use continued fractions may be found in H. S. Wall's Analytic Theory of Continued Fractions (D. VanNostrand Co. Inc., Princeton, N.J., 1948).

Terminology

Maximum and root-mean-square values for the relative and (where necessary) the absolute errors are given for each module. These are defined thus:

Let f(x) = the correct value for a function
g(x) = the result obtained from the module in question

Then the absolute error of the result is

ABS(f(x) - g(x)),

and the relative error of the result is

ABS((f(x) - g(x))/f(x)).

Let the number of sample results obtained be N; then the root-mean-square of the absolute error is

$SQRT(\sum_i (ABS(f(x_i) - g(x_i))**2)/N)$,

and the root-mean-square of the relative error is

$SQRT(\sum_i (ABS((f(x_i) - g(x_i))/f(x_i))**2)/N)$.

The Library mathematical modules are summarized in Figures 5 and 6.

Function	Real Arguments	
	Short Float	Long Float
SQRT	IHEWSQS	IHEWSQL
EXP	IHEWEXS	IHEWEXL
LOG,LOG2,LOG10	IHEWLNS	IHEWLNL
SIN, COS,SIND,COSD	IHEWSNS	IHEWSNL
TAN, TAND	IHEWTNS	IHEWTNL
ATAN, ATAND	IHEWATS	IHEWATL
SINH, COSH	IHEWSHS	IHEWSHL
TANH	IHEWTHS	IHEWTHL
ATANH	IHEWHTS	IHEWHTL
ERF, ERFC	IHEWEFS	IHEWEFL

Figure 5. Mathematical Functions With Real Arguments

Function	Complex Arguments	
	Short Float	Long Float
SQRT	IHEWSQW	IHEWSQZ
EXP	IHEWEXW	IHEWEXZ
LOG	IHEWLNW	IHEWLNZ
SIN,COS,SINH,COSH	IHEWSNW	IHEWSNZ
TAN, TANH	IHEWTNW	IHEWTNZ
ATAN, ATANH	IHEWATW	IHEWATZ

Figure 6. Mathematical Functions With Complex Arguments

FUNCTIONS WITH REAL ARGUMENTS

SQRT (short floating-point real)

Module Name: IHEWSQS

Entry Point: IHESQS0

Function:

To calculate the square root of x.

Method:

If x = 0, SQRT(x) = 0. Otherwise, let

X = 16**(2*p - q)*f,

where p is an integer, q = 0 or 1, and 1/16 ≤ f < 1. Then

SQRT(x) = 16**p*4**-q*SQRT(f)

The first approximation, y_i, of SQRT(x) is obtained by the hyperbolic fit

y_0 = 16**p*4**-q*(1.681595-1.288973/(0.8408065+f))

This approximation attains the minimax relative error. The maximum relative error is 2**-5.748.

Two Newton-Raphson iterations then yield

$y_1 = (y_i+x/y_i)/2$

$y_2 = (y_1-x/y_1)/2+x/y_1$

with a partial rounding. The maximum relative error of y_2 is theoretically 2**-25.9.

Effect of Argument Error:

The relative error caused in the result is approximately half the relative error in the argument.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
Full Range	Exponential	0.133	0.477

Error and Exceptional Conditions:

P : x < 0

SQRT (long floating-point real)

Module Name: IHEWSQL

Entry Point: IHESQL0

Function:

To calculate the square root of x.

Method:

If x = 0, SQRT(x) = 0. Otherwise, let x = 16**(2*p - q)*f, where p is an integer, q = 0 or 1, and 1/16 ≤ f < 1. Then

SQRT(x) = 16**p*4**-q *SQRT(f).

The first approximation of SQRT(f) is computed as:

y = 16**p*4**(1-q)*0.2202(f+0.2587)

This approximation was chosen in order to permit the use of single precision instructions in the final iteration by making the quantity x/y_3-y_3 below less than 16**(p-8).

Four Newton-Raphson iterations of the form $y = (y_n + x/y_n)/2$ are then applied, two in short precision and two in long precision, the last being computed as

SQRT(x) = $y_3 + (x/y_3 - y_3)/2$

with an appropriate truncation maneuver to obtain virtual rounding.

The maximum relative error of the final result is theoretically 2**-63.23.

Effect of an Argument Error:

The relative error caused in the result is approximately half of the relative error in the argument.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
Full range	Exponential	0.0310	0.109

Error and Exceptional Conditions:

P : x < 0

EXP (short floating-point real)

Module Name: IHEWEXS

Entry Point: IHEEXS0

Function: To calculate e to the power x.

Method:

If x < -180.218, a zero result is returned immediately.

Otherwise EXP(x) is calculated as follows:

1. Divide x by LOG(2) and write

   y = x/LOG(2) = 4*a-b-d

   where a and b are integers, 0 ≤ b≤ 3 and 0 ≤ d < 1.

   Then EXP(x) = 2**y = 16**a*2**-b*2**-d

2. Compute 2**-d by the following fractional approximation:

   2**-d = 1-2*d/(0.034657359*d**2+d+9.9545948-617.97227/(d**2+87.417497))

   This formula can be obtained by the transformation of the Gaussian continued fraction

   EXP(-z)=1-z/(1+z/(2-z/(3+z/(2-z/(5+z/(2-z/(7+z/2-...)))))))

   The maximum relative error of this approximation is 2**-29.

3. Multiply 2**-d by 2**-b

4. Finally multiply by 16**a by adding a to the characteristic of the result of step 3.

Effect of Argument Error:

The relative error caused in the result is approximately equal to the absolute error in the argument, i.e., to the argument relative error multiplied by x. Thus for large values of x, even the round-off error of the argument causes a substantial relative error in the answer.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
-1 < x < 1	Uniform	0.129	0.444
Full Range	Uniform	0.115	0.459

Error and Exceptional Conditions:

I : OVERFLOW if x > 174.673

EXP (long floating-point real)

Module Name: IHEWEXL

Entry Point: IHEEXL0

Function: To calculate e to the power x.

Method:

If x < -180.2187, return zero as the result.

Otherwise EXP(x) is calculated as follows:

1. Divide x by LOG(2) and let

   y=x/LOG(2) = 4*a-b-c/16

where a, b, and c are integers, $0 \le b \le 3$, and $0 \le c \le 15$. Then, as an exact representation for x, obtain

x = (4*a-b-c/16)*LOG(2)-d

where the remainder d is in the range $0 \le d <$ LOG(2)/16. This reduction is carried out in extra precision. Then

EXP(x) = 16**a*2**-b*2**(-c/16)*EXP(-d)

2. Compute EXP(-d) by using a minimax polynomial approximation of degree 6 over the range $0 \le d <$ LOG(2)/16. The coefficients of this approximation were obtained by taking the minimax of relative errors under the constraint that the constant term shall be exactly one. The relative error is less than 2**-56.87.

3. Multiply EXP(-d) by 2**(-c/16), then halve the result b times.

4. Finally, multiply by 16**a by adding a to the characteristic of the result of step 3.

Effect of an Argument Error:

The relative error caused in the result is approximately equal to the absolute error in the argument, i.e., to the argument relative error multiplied by x. Thus for large values of x, even the round-off error of the argument causes a substantial relative error in the answer.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
-1 < x < 1	Uniform	0.0543	0.209
Full range	Uniform	0.0472	0.426

Error and Exceptional Conditions:

I : OVERFLOW if x > 174.673

LOG, LOG2, LOG10 (short floating-point real)

Module Name: IHEWLNS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Log x to the base e	LOG(x)	IHELNSE
Log x to the base 2	LOG2(x)	IHELNS2
Log x to the base 10	LOG10(x)	IHELNSD

Function: To calculate log x.

Method:

Let x = 16**p*2**(-q)*m where p and q are integers, $0 \le q \le 3$, and $1/2 \le m < 1$.

Two constants, a (= base point) and b (= -LOG2(a)), are defined as follows:

If $1/2 \le m <$ 1/SQRT(2): then a = 1/2, b = 1

If 1/SQRT(2) $\le m < 1$: then a = 1, b = 0

Let y = (m-a)/(m+a).

Then m = a*(1+y)/(1-y) and ABS(y) < 0.1716.

Now x = (2**(4*p-q-b))*((1+y)/(1-y)). Therefore

LOG(x) = (4*p-q-b)*LOG(2) + LOG((1+y)/(1-y)).

To obtain LOG((1+y)/(1-y)) first w = 2*y = (m-a)/(0.5m+0.5a) is computed (which is represented in System/360 with more significant digits than y itself), then the following approximation is performed:

LOG((1+y)/(1-y)) = w*(c_0 +c_1*w**2/(c_2-w**2))

The coefficients were obtained by the minimax rational approximation of LOG((1+y)/(1-y))/(2*y), in relative error, under the constraint that the first term shall be one. The maximum relative error of this approximation is less than 2**-25.33.

LOG2(x) or LOG10(x) is calculated by multiplying the above result by LOG2(e) or LOG10(e) respectively.

Effect of Argument Error:

The absolute error caused in the result is approximately equal to the relative error in the argument. Thus if the argument is close to 1, even the round-off error of the argument causes a substantial relative error in the answer, since the function value there is very small.

Accuracy:

Arguments: Range	Arguments: Distribution	Relative Error *10**6: RMS	Relative Error *10**6: Maximum
IHELNSE			
Excluding 0.5 < x < 2.0	Exponential	0.122	0.841
IHELNS2			
Excluding 0.5 < x < 2.0	Exponential	0.340	0.980
IHELNSD			
Excluding 0.5 < x < 2.0	Exponential	0.219	1.10

Arguments: Range	Arguments: Distribution	Absolute Error *10**6: RMS	Absolute Error *10**6: Maximum
IHELNSE			
0.5 < x < 2.0	Uniform	0.0255	0.0679
IHELNS2			
0.5 < x < 2.0	Uniform	0.228	0.479
IHELNSD			
0.5 < x < 2.0	Uniform	0.0228	0.0720

Error and Exceptional Conditions:

P : x ≤ 0

LOG, LOG2, LOG10 (long floating-point real)

Module Name: IHEWLNL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Log x to the base e	LOG(x)	IHELNLE
Log x to the base 2	LOG2(x)	IHELNL2
Log x to the base 10	LOG10(x)	IHELNLD

Function: To calculate log x.

Method:

Let x = 16**p*2**(-q)*m where p and q are integers, 0 ≤ q ≤ 3, and 1/2 ≤ m < 1.

Two constants, a (= base point) and b (= -LOG2(a)), are defined as follows:

if 1/2 ≤ m ≤ 1/SQRT(2): then a = 1/2, b = 1

if 1/SQRT(2) ≤ m < 1: then a = 1, b = 0.

Let y = (m - a)/(m + a).

Then m = a*(1 + y)/(1 - y) and ABS(y) < 0.1716.

Now x = 2**(4*p - q - b)*(1 + y)/(1 - y) Therefore

LOG(x) = (4*p - q - b)*LOG(2) + LOG((1 + y)/(1 - y)).

To obtain LOG((1+y)/(1-y)) first w = 2*y = (m-a)/(0.5m+0.5a) is computed (which is represented in System /360 with more significant digits than y itself), then the following approximation is performed:

$$LOG((1+y)/(1-y)) = w*(c_0+c_1*w**2/(c_2-w**2))$$

The coefficients were obtained by the minimax rational approximation of LOG((1+y)/(1-y))/(2*y), in relative error, over the range y**2 ≤ 0.02944 under the constraint that the first term shall be 1. The maximum relative error of this approximation is less than 2**-60.55.

LOG2(x) or LOG10(x) is calculated by multiplying the above result by LOG2(e) or LOG10(e) respectively.

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the relative error in the argument. Thus if the argument is close to 1, even the round-off error of the argument causes a substantial relative error in the answer, since the function value there is very small.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHELNLE			
Excluding 0.5 < x < 2.0	Exponential	0.0544	0.339
IHELNL2			
Excluding 0.5 < x < 2.0	Exponential	0.0881	0.425
IHELNLD			
Excluding 0.5 < x < 2.0	Exponential	0.0659	0.322

Arguments		Absolute Error *10**15	
Range	Distribution	RMS	Maximum
IHELNLE			
0.5 < x < 2.0	Uniform	0.0239	0.0472
IHELNL2			
0.5 < x < 2.0	Uniform	0.0291	0.0576
IHELNLD			
0.5 < x < 2.0	Uniform	0.0125	0.0294

Error and Exceptional Conditions:

P : x ≤ 0

<u>SIN, SIND, COS, COSD (short floating-point real)</u>

Module Name: IHEWSNS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Sin(x radians)	SIN(x)	IHESNSS
Sin(x degrees)	SIND(x)	IHESNSZ
Cos(x radians)	COS(x)	IHESNSC
Cos(x degrees)	COSD(x)	IHESNSK

Function: To calculate sin x or cos x.

Method:

Let k = pi/4

Evaluate p = ABS(x)*(1/k) if x is in radians
or p = ABS(x)*(1/45) if x is in degrees,
using long-precision multiplication to safeguard accuracy.

Separate p into integer part q and fractional part r, i.e., p = q + r where 0 ≤ r < 1.

Define q_1 = q if SIN or SIND is required and x ≥ 0;
q_1 = q + 2 if COS or COSD is required;
q_1 = q + 4 if SIN or SIND is required and x < 0.

Then for all values of x each case has been reduced to the computation of SIN(k*(q_1+r)) = SIN(t) say, where t ≥ 0.

Let q_2 = MOD(q_1,8).
If q_2 = 0, SIN(t) = SIN(k*r)
If q_2 = 1, SIN(t) = COS(k*(1-r))
If q_2 = 2, SIN(t) = COS(k*r)
If q_2 = 3, SIN(t) = SIN(k*(1-r))
If q_2 = 4, SIN(t) = -SIN(k*r)
If q_2 = 5, SIN(t) = -COS(k*(1-r))
If q_2 = 6, SIN(t) = -COS(k*r)
If q_2 = 7, SIN(t) = -SIN(k*(1-r)).

Thus it is necessary to compute only SIN(k*r_1) or COS(k*r_1) where r_1 = r or 1 - r and 0 ≤ r_1 ≤ 1, as follows:

1. SIN(k*r_1) = r_1*(a_0 +a_1r_1**2+a_2r_1**4+a_3r_1**6)

   The coefficients were obtained by the Chebyshev interpolation. The maximum relative error is less than 2**-28.1.

2. cos(k*r_1) = 1+b_1r_1**2+b_2r_1**4+b_3r_1**6

   The coefficients were obtained by a variation of the minimax approximation which provides partial rounding for the short precision computation. The maximum absolute error is 2**-24.57.

Effect of an Argument Error:

The absolute error of the answer is approximately equal to the absolute error in the argument. Hence, the larger the argument, the larger its absolute error and the larger the absolute error of the result. Since the function diminishes periodically for both sine and cosine, no consistent control of the relative error can be maintained outside the range -pi/2 to pi/2 radians (or -90 to +90 degrees).

Accuracy:

Arguments		Absolute Error *10**6	
Range	Distribution	RMS	Maximum
IHESNSS			
ABS(x) ≤ pi/2	Uniform	0.0467	0.119
pi/2 < ABS(x) ≤ 10	Uniform	0.0400	0.125
10 < ABS(x) ≤ 100	Uniform	0.0401	0.124
IHESNSC			
0 ≤ x ≤ pi	Uniform	0.0408	0.119
-10 ≤ x < 0, pi < x ≤ 10	Uniform	0.0402	0.120
10 < ABS(x) ≤ 100	Uniform	0.0398	0.113

Error and Exceptional Conditions:

P : IHESNSS, IHESNSC:
ABS(x) ≥ 2**18*pi

IHESNSZ, IHESNSK:
ABS(x) ≥ 2**18*180

SIN, SIND, COS, COSD (long floating-point real)

Module Name: IHEWSNL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Sin(x radians)	SIN(x)	IHESNLS
Sin(x degrees)	SIND(x)	IHESNLZ
Cos(x radians)	COS(x)	IHESNLC
Cos(x degrees)	COSD(x)	IHESNLK

Function: To calculate sin x or cos x.

Method:

Let y = ABS(x)/(pi/4) for x in radians,
or y = ABS(x)/45 for x in degrees,
and y = q + r, q integral, 0 ≤ r < 1.

Take q_1 = q for SIN or SIND with positive or zero argument,
q_1 = q + 2 for COS or COSD,
q_1 = q + 4 for SIN or SIND with negative argument,
and q_2 = MOD(q_1,8).

Since COS(x) = SIN(ABS(x) + pi/2)
and SIN(-x) = SIN(ABS(x) + pi),

it is only necessary to find

SIN(pi/4*(q_2 + r)), for 0 ≤ q_2 ≤ 7.

Therefore compute:

SIN(pi/4*r), if q_2 = 0 or 4,
COS(pi/4*(1 - r)), if q_2 = 1 or 5,
COS(pi/4*r), if q_2 = 2 or 6,
SIN(pi/4*(1 - r)) if q_2 = 3 or 7.

SIN(pi/4*r_1)/r_1, where r_1 is r or (1 - r), is computed by using the Chebyshev interpolation polynomial of degree 6 in r_1**2, in the range 0 ≤ r_1**2 ≤ 1, with maximum relative error 2**(-58).

COS(pi/4*r_1) is computed by using the Chebyshev interpolation polynomial of degree 7 in r_1**2, in the range 0 ≤ r_1**2 ≤ 1, with maximum relative error 2**(-64.3).

Finally, if q_2 ≥ 4 a negative sign is given to the result.

Effect of an Argument Error:

The absolute error of the answer is approximately equal to the absolute error in the argument. Hence, the larger the argument, the larger its absolute error and the larger the absolute error of the result. Since the function diminishes periodically for both sine and cosine, no consistent control of the relative error can be maintained outside the range -pi/2 to pi/2 radians (or -90 to +90 degrees).

Accuracy:

IHESNLS

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHESNLS			
ABS(x) ≤ pi/2	Uniform	0.0181	0.0771
pi/2 < ABS(x) ≤ 10	Uniform	0.317	2.36
10 < ABS(x) ≤ 100	Uniform	0.928	2.65
IHESNLC			
0 ≤ x ≤ pi	Uniform	0.0739	0.266
-10 ≤ x < 0, pi < x ≤ 10	Uniform	0.0683	0.266
10 < ABS(x) ≤ 100	Uniform	1.02	2.68

Error and Exceptional Conditions:

P : IHESNLS, IHESNLC:
ABS(x) ≥ 2**50*pi

IHESNLZ, IHESNLK:
ABS(x) ≥ 2**50*180

TAN, TAND (short floating-point real)

Module Name: IHEWTNS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Tan(x radians)	TAN(x)	IHETNSR
Tan(x degrees)	TAND(x)	IHETNSD

Function: To calculate tan x.

Method:

Evaluate p = (4/pi)*ABS(x) if x is in radians,
or p = (1/45)*ABS(x) if x is in degrees,

using long-precision multiplication to safeguard accuracy.

Let q and r be respectively the integral and fractional parts of p.

If q is even, put s = r;
if q is odd, put s = 1-r.

Let q_1 = MOD(q,4). Then

If q_1 = 0, TAN(ABS(x)) = TAN(pi*s/4)
If q_1 = 1, TAN(ABS(x)) = COT(pi*s/4)
If q_1 = 2, TAN(ABS(x)) = -COT(pi*s/4)
If q_1 = 3, TAN(ABS(x)) = -TAN(pi*s/4)

Compute TAN(pi*s/4) and COT(pi*s/4) as the ratio of two polynomials:

TAN(pi*s/4) = s*P(u)/Q(u)
COT(pi*s/4) = Q(u)/(s*P(u))

where u = s**2/2 and

P(u) = -8.460901+u and

Q(u) = 10.772754+5.703366*u
-0.159321*u**2

These coefficients were obtained by the minimax rational approximation in relative error of the above form. The maximum relative error of this approximation is 2**-26. The variable u, rather than s**2, was chosen for P and Q in order to improve the rounding effect of the coefficients.

Finally, if x < 0, put

TAN(x) = -TAN(ABS(x)).

Effect of an Argument Error:

The absolute error of the answer is approximately equal to the absolute error of the argument multiplied by (1 + TAN(x) **2). Hence if x is near an odd multiple of pi/2, an argument error will produce a large absolute error in the answer.

The relative error in the result is approximately equal to twice the absolute error in the argument divided by SIN(2* x). Hence, if x is near a multiple of pi/2, an argument error will produce a large relative error in the result.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHETNSR			
ABS(x) ≤ pi/4	Uniform	0.290	1.64
pi/4 < ABS(x) < pi/2	Uniform	0.369	1.54
pi/2 < ABS(x) ≤ 10	Uniform	0.321	4.81
10 < ABS(x) ≤ 100	Uniform	0.310	1.38

Error and Exceptional Conditions:

P : IHETNSR: ABS(x) ≥ 2**18*pi
    IHETNSD: ABS(x) ≥ 2**18*180

I : IHETNSR: OVERFLOW
    IHETNSD: OVERFLOW

TAN, TAND (long floating-point real)

Module Name: IHEWTNL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Tan(x radians)	TAN(x)	IHETNLR
Tan(x degrees)	TAND(x)	IHETNLD

Function: To calculate tan x.

Method:

Evaluate

p = (4/pi)*ABS(x) if x is in radians
or p = (1/45)*ABS(x) if x is in degrees.

Let q and r be respectively the integral and fractional parts of p.

If q is even, put s = r;
If q is odd, put s = 1 - r.

Let q_1 = MOD(q,4). Then

If q_1 = 0, TAN(ABS(x)) = TAN(pi*s/4)
If q_1 = 1, TAN(ABS(x)) = COT(pi*s/4)
If q_1 = 2, TAN(ABS(x)) = -COT(pi*s/4)
If q_1 = 3, TAN(ABS(x)) = -TAN(pi*s/4)

Compute TAN(pi*s/4) and COT(pi*s/4) as the ratio of two polynomials:

TAN(pi*s/4) = s*P(s**2)/Q(s**2))
COT(pi*s/4) = Q(s**2)/(s*P(s**2))

where both P and Q are polynomials of degree 3 in s**2. The coefficients of P and Q were obtained by the minimax rational approximation (in relative error) of TAN(pi*s/4) of the indicated form. The maximum relative error of this approximation is 2**-55.6.

Finally, if x < 0, TAN(x) = -TAN(ABS(x)).

Effect of an Argument Error:

The absolute error in the result is approximately equal to the absolute error in the argument multiplied by (1+TAN(x)**2). Hence, if x is near an odd multiple of pi/2, an argument error will produce a large absolute error in the result.

The relative error in the result is approximately equal to twice the absolute error in the argument divided by SIN(2*x). Hence, if x is near a multiple of pi/2, an argument error will produce a large relative error in the result.

Accuracy:

IHETNLR

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
ABS(x) ≤ pi/4	Uniform	0.646	0.571
pi/4 < ABS(x) < pi/1.5	*Uniform	0.471	2.26
pi/1.5 < ABS(x) ≤ 10	*Uniform	84.2	4730
10 < ABS(x) ≤ 100	*Uniform	78.8	2710
*The errors quoted are those encountered in a sample of 5000 points; each figure depends very much on the particular points encountered near the singularities of the function, where no error control can be maintained.			

Error and Exceptional Conditions:

P : IHETNLR: ABS(x) ≥ 2**50*pi
IHETNLD: ABS(x) ≥ 2**50*180

I : IHETNLR: OVERFLOW
IHETNLD: OVERFLOW

ATAN(X), ATAND(X), ATAN (Y,X), ATAND (Y,X) (short floating-point real)

Module Name: IHEWATS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Arctan x (radians)	ATAN(x)	IHEATS1
Arctan(y/x) (radians)	ATAN(y,x)	IHEATS2
Arctan x (degrees)	ATAND(x)	IHEATS3
Arctan(y/x) (degrees)	ATAND(y,x)	IHEATS4

Function:

To calculate arctan x or arctan(y/x). The result range is:

Arctan x (radians)	± pi/2
Arctan(y/x) (radians)	± pi
Arctan x (degrees)	± 90°
Arctan(y/x) (degrees)	± 180°

Method:

1. ATAN(y,x)

If x = 0 or ABS(y/x) ≥ 2**24, the answer SIGN(y)*pi/2 is returned except for the error case x = y = 0. Otherwise

ATAN(y,x) = ATAN(y/x) if x > 0
or ATAN(y,x) = ATAN(y/x) + SIGN(y)*pi if x < 0.

Hence the computation is now reduced to the single argument case.

2. ATAN(x)

The general case may be reduced to the range 0 ≤ x ≤ 1 since

ATAN(-x) = -ATAN(x), and
ATAN(1/ABS(x)) = pi/2 - ATAN(ABS(x)).

A further reduction to the range ABS(x) ≤ TAN(pi/12) is made by using

ATAN(x) = pi/6 + ATAN((SQRT(3)*x - 1)/(x + SQRT(3))).

Care is taken to avoid the loss of significant digits in computing

SQRT(3)*x - 1.

For the basic range ABS(x) ≤ TAN(pi/12), use an approximation formula of the form

ATAN(x)/x = a + b*x**2 + c/(d + x**2)

with relative error less than 2**-27.1.

3. ATAND(x) and ATAND(y,x)

The treatment is as above with the addition of a final conversion of the result to degrees.

Effect of an Argument Error:

Let t = x or y/x; then the absolute error of the answer approximates to the absolute error in t divided by (1 + t**2). Hence, for small values of t, the two errors are approximately the same; however, as t becomes larger the effect of the argument error on the answer error diminishes.

Accuracy:

Arguments: Range	Arguments: Distribution	Relative Error *10**6: RMS	Relative Error *10**6: Maximum
IHEATS1			
ABS(x) < 1	Uniform	0.127	0.898
Full range	Exponential	0.246	0.994
IHEATS2			
ABS(y) ≤ 1, ABS(x) ≤ 1	Exponential / Uniform	0.291	1.62

Error and Exceptional Conditions:

P : IHEATS2, IHEATS4: x = y = 0

ATAN(X), ATAND(X), ATAN (Y,X), ATAND (Y,X) (long floating-point real)

Module Name: IHEWATL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Arctan x (radians)	ATAN(x)	IHEATL1
Arctan(y/x) (radians)	ATAN(y,x)	IHEATL2
Arctan x (degrees)	ATAND(x)	IHEATL3
Arctan(y/x) (degrees)	ATAND(Y,X)	IHEATL4

Function:

To calculate arctan x or arctan(y/x). The result range is:

Arctan x (radians) ± pi/2
Arctan(y/x) (radians) ± pi
Arctan x (degrees) ± 90°
Arctan(y/x) (degrees) ± 180°

Method:

1. ATAN(y,x)

If x = 0 or ABS(y/x) ≥ 2**56, the answer SIGN(y)*pi/2 is returned except for the error case x = y = 0. Otherwise

ATAN(y,x) = ATAN(y/x) if x > 0
or ATAN(y,x) = ATAN(y/x) + SIGN(y)*pi if x < 0.

Hence the computation is now reduced to the single argument case.

2. ATAN(x)

The general case may be reduced to the range 0 ≤ x ≤ 1 since

ATAN(-x) = - ATAN(x), and
ATAN(1/ABS(x)) = pi/2 - ATAN(ABS(x)).

A further reduction to the range ABS(x) ≤ TAN(pi/12) is made by using

ATAN(x) = pi/6 + ATAN((SQRT(3)*x - 1)/(x + SQRT(3)))

Care is taken to avoid the loss of significant digits in computing

SQRT(3)*x - 1

For the basic range ABS(x) ≤ TAN(pi/12), use a continued fraction of the form

$$ATAN(x)/x = 1+u*(b_0-a_1/(b_1+u-a_2/(b_2+u-a_3/(b_3+u))))$$

where u = x**2.

The relative error of this approximation is less than 2**-60.7.

The coefficients of this formula were derived by transforming a minimax rational approximation in relative error over the range 0 ≤ u ≤ 0.071797 for ATAN(x)/x of the following form:

$$ATAN(x)/x = a_0+u*((c_0+c_1*u+c_2*u*u+c_3*u*u*u)/(d_0+d_1*u+d_2*u*u+u*u*u)).$$

under the constraint that a = 1.

3. ATAND(x) and ATAND(y,x)

The treatment is as above with the addition of a final conversion of the result to degrees.

Effect of an Argument Error:

Let t = x or y/x; then the absolute error of the answer approximates to the absolute error in t divided by (1 + t**2). Hence, for small values of t, the two errors are approximately the same; however, as t becomes larger the effect of the argument error on the answer error diminishes.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHEATL1			
ABS(x) < 1	Uniform	0.0415	0.206
Full range	Exponential	0.0526	0.206
IHEATL2			
ABS(y) ≤ 1, ABS(x) ≤ 1	Exponential / Uniform	0.0688	0.358

Error and Exceptional Conditions:

P : IHEATL2, IHEATL4: x = y = 0

SINH, COSH (short floating-point real)

Module Name: IHEWSHS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Hyperbolic sin x	SINH(x)	IHESHSS
Hyperbolic cos x	COSH(x)	IHESHSC

Function:

To calculate hyperbolic sin x or hyperbolic cos x.

Method:

1. ABS(x) < 1

Compute SINH(x) as:

$$SINH(x)=x+c_1*x**3+c_2*x**5+c_3*x**7$$

The coefficients were obtained by the minimax approximation (in relative error) of SINH(x)/x as a function of x**2. The maximum relative error of this approximation is 2**(-25.6).

2. x≥1

Compute SINH(x) as:

```
SINH(x)=(1+D)*(EXP(x+LOG(V))-
        V**2/EXP(x+LOG(V)))
```

Using module IHEWEXS.

Here 1+D=1/(2*V), so that this expression is theoretically equivalent to (EXP(x)-EXP(-x))/2. The value of V (and consequently those of LOG(V) and D) was so chosen as to satisfy the following conditions:

a) V is slightly less than 1/2, so that D is positive and small

b) LOG(V) is an exact multiple of 2**(-16).

Condition (b) ensures that the addition x+LOG(V) is carried out exactly.

3. x≤-1

Use the identity

```
SINH(x)=-SINH(ABS(x))
```

to reduce to case (2), above.

4. COSH(x)

For all legal values of arguments, use the identity

```
COSH(x)=(1+D)*(EXP(x+LOG(V))+
        V**2/EXP(x+LOG(V)))
```

Here the notation and considerations are identical to those used in the computation of SINH(x), in (2) above.

Effect of Argument Error:

The relative error caused in the result is approximately as follows:

SINH: The absolute error in the argument divided by TANH(x), i.e., of the order of the absolute error in the argument for large x, or of the relative error in the argument for small x.

COSH: The absolute error in the argument multiplied by TANH(x), i.e., of the order of the absolute error in the argument.

Thus, for large values of x, even the round-off error of the argument causes a substantial relative error in the answer.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHESHSS			
0 < ABS(x) ≤ 1	Uniform	0.198	0.877
1 < ABS(x) < 2	Uniform	0.255	1.03
ABS(x)≤170	Uniform	0.201	0.816
IHESHSC			
ABS(x) ≤ 1	Uniform	0.406	0.962
1 < ABS(x) < 2	Uniform	0.248	0.720
ABS(x)≤170	Uniform	0.202	0.816

Error and Exceptional Conditions:

H : OVERFLOW in real EXP routine (IHEWEXS).

COSH, SINH (long floating-point real)

Module Name: IHEWSHL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Hyperbolic cos x	COSH(x)	IHESHLC
Hyperbolic sin x	SINH(x)	IHESHLS

Function:

To calculate hyperbolic sin x or hyperbolic cos x.

Method:

1. ABS(x)<0.881374

Compute SINH(x) as

$$SINH(x)=c_0*x+c_1*x**3+c_2*x**5+...+c_6*x**13$$

The coefficients were obtained by the minimax approximation (in relative error) of SINH(x)/x as a function of x**2. The maximum relative error of this approximation is 2**(-55.7).

2. x≥0.881374

Compute SINH(x) as

SINH(x)=(1+D)*(EXP(x+LOG(V))-
V**2/EXP(x+LOG(V)))

using module IHEEXL

Here 1+D=1/(2*V) so that this expression is theoretically equivalent to (EXP(x)-EXP(-x))/2. The value of V (and consequently those of LOG(V) and D) was so chosen as to satisfy the following conditions:

a) V is slightly less than 1/2 so that D is positive and small.

b) LOG(V) is an exact multiple of 2**(-16).

Condition (b) ensures that the addition x+LOG(V) is carried out exactly.

3. x≤-0.881374

Use the identity

SINH(x)=-SINH(ABS(x))

to reduce the case to that of step (2).

4. COSH(x)

For all legal values of arguments, use the identity:

COSH(x)=(1+D)*(EXP(x+LOG(V))+
V**2/EXP(x+LOG(V)))

Here the notation and considerations are identical to those used in the computation of SINH (x) in step (2) above.

Effect of an Argument Error:

The relative error caused in the result is approximately as follows:

SINH: The absolute error in the argument divided by TANH(x), i.e., of the order of the absolute error in the argument for large x, or of the relative error in the argument for small x.

COSH: The absolute error in the argument multiplied by TANH(x), i.e., of the order of the absolute error in the argument.

Thus, for large values of x, even the round-off error of the argument causes a substantial relative error in the answer.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHESHLC			
ABS(x) ≤ 5	Uniform	0.106	0.376
ABS(x) ≤ 170	Uniform	0.109	0.390
IHESHLS			
ABS(x) < 0.881374	Uniform	0.0373	0.203
0.881374 < ABS(x) ≤ 5	Uniform	0.100	0.354
ABS(x) ≤ 170	Uniform	0.102	0.361

Error and Exceptional Conditions:

H : OVERFLOW in real EXP routine (IHEWEXL).

TANH (short floating-point real)

Module Name: IHEWTHS

Entry Point: IHETHS0

Function: To calculate hyperbolic tan x.

Method :

1. ABS(x) ≤ 2**-12

Return x as result.

2. 2**-12 < ABS(x) ≤ 0.7

Use a fractional approximation of the form:

TANH(x)/x = 1 -
x**2*(.0037828+.8145651/(x**2+2.471749))

The coefficients of this approximation were obtained by taking the minimax of relative error, over the range x**2 < 0.49, of approximations of this form under the constraint that the first term shall be 1. The maximum relative error of this approximation is 2**-26.4.

3. 0.7 ≤ x < 9.011

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

4. $x \geq 9.011$

Return result 1.

5. $x < -0.7$

Use the identity:

TANH(x) = -TANH(-x).

and apply 3 or 4 above, as appropriate.

Effect of an Argument Error:

The relative error caused in the result is approximately twice the absolute error in the argument divided by SINH(2*x). Thus for small values of x it is of the order of the relative error in the argument, and as x increases the effect of the argument error is diminished.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
ABS(x)≤0.7	Uniform	0.149	0.781
0.7<ABS(x) ≤9.011	Uniform	0.0389	0.288

TANH (long floating-point real)

Module Name: IHEWTHL

Entry Point: IHETHL0

Function: To calculate hyperbolic tan x.

Method:

1. ABS(x) ≤ 2**-28

Return x as result

2. 2**-12 < ABS(x) < 0.54931

Use a transformed minimax approximation of the form

$$TANH(x)/x=c_0+d_1*x**2/(x**2+c_1)+d_2/(x**2+c_2)+d_3/(x**2+c_3)$$

The minimax of relative error was taken over the range x**2 ≤ 0.30174 under the constraint that the first term is 1.

The maximum relative error is 2**-63

3. 0.54931 ≤ x < 20.101

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

4. $x \geq 20.101$

Return result 1.

5. $x \leq -0.54931$

Effect of an Argument Error:

The relative error caused in the result is approximately twice the absolute error in the argument divided by SINH(2*x). Thus for small values of x it is of the order of the relative error in the argument, and as x increases the effect of the argument error is diminished.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
ABS(x) ≤ 0.54931	Uniform	0.0385	0.192
0.54931 < ABS(x) ≤ 5	Uniform	0.0109	0.160

ATANH (short floating-point real)

Module Name: IHEWHTS

Entry Point: IHEHTS0

Function: To calculate hyperbolic arctan x.

Method:

1. ABS(x) ≤ 0.2

Use a rational approximation of the form:

ATANH(x) = x + x ** 3/ (a + b*x**2)

2. 0.2 < ABS(X) < 1

ATANH(x) = -SIGN(x)*0.5*LOG((0.5 - ABS(x/2))/(0.5 + ABS(x/2)))

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the absolute error in the argument divided by (1 - x**2). Thus as x approaches +1 or -1, relative error increases rapidly. Near x = 0, the relative error in the result is of the order of that in the argument.

Accuracy:

Arguments		Relative Error (*10**6)	
Range	Distribution	RMS	Maximum
$-0.2 \le x \le 0.2$	Uniform	0.456	1.07
$-0.9 < x < 0.9$	Uniform	0.391	1.18

Error and Exceptional Conditions:

P : ABS(x) ≥ 1

ATANH (long floating-point real)

Module Name: IHEWHTL

Entry Point: IHEHTL0

Function: To calculate hyperbolic arctan x.

Method:

1. ABS(x) ≤ 0.25

   Use a Chebyshev polynomial of degree 8 in x**2 to compute ATANH(x)/x.

2. 0.25 < ABS(x) < 1

   ATANH(x) = -SIGN(x)*0.5*LOG((0.5 - ABS(x/2))/(0.5 + ABS(x/2)))

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the absolute error in the argument divided by (1 - x**2). Thus as x approaches +1 or -1, relative error increases rapidly. Near x = 0, the relative error in the result is of the order of that in the argument.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
ABS(x) ≤ 0.25	Uniform	0.0638	0.223
ABS(x) ≤ 0.95	Uniform	0.0913	0.253

Error and Exceptional Conditions:

P : ABS(x) ≥ 1

ERF, ERFC (short floating-point real)

Module Name: IHEWEFS

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Error function (x)	ERF(x)	IHEEFSF
Complement of error function(x)	ERFC(x)	IHEEFSC

Function:

To calculate the error function of x or the complement of this function.

Method:

1. $0 \le x \le 1$

   Compute ERF(x) by the following approximation:

   ERF(x)=x*(a +a_1*x**2+a_2*x**4+....+a_5*x**10

   The coefficients were obtained by the minimax approximation in relative error of ERF(x)/x as a function of x**2 over the range 0 ≤ x**2 ≤ 1. The relative error of this approximation is less than 2**-24.6. The value of ERFC(x) is computed as

   ERFC(x) = 1-ERF(x)

2. $1 < x < 2.040452$

   Compute ERFC(x) by the following approximation:

   ERFC(x) = b_0+b_1*z+b_2*z**2+..... + b_9*z**9

   where z = x-T_0 and T_0 = 1.709472. The coefficients were obtained by the minimax approximation in absolute error of the function f(z) = ERFC(z+T_0) over the range -0.709472 ≤ z ≤ 0.33098. The absolute error of this approximation is less than 2**-31.5. The limits of this range and the value of the origin, T_0, were chosen to minimize the hexadecimal rounding errors.

   The value of ERFC(x) within this range is between 1/256 and 0.1573.

   The value of ERF(x) is computed as

   ERF(x) = 1-ERFC(x)

3. $2.040452 \le x < 13.306$

   Compute ERFC(x) by the following approximation:

ERFC(x) = EXP(-z) *F/x

where z = x**2 and

$$F = c_0+(c_1+c_2*z+c_3*z**2)/(d_1*z+d_2*z**2+z**3)$$

The coefficients of F were obtained by transforming a minimax rational approximation in absolute error of the function f(w) = ERFC(x)*x*EXP(x**2)over the range 13.306**-2 ≤ w ≤ 2.040452**-2 (where w=x**2). This approximation is of the form

$$f(w) = (a_0+a_1*w+a_2*w**2+a_3*w**3)/(b_0+b_1*w+w**2)$$

The absolute error of this approximation is less than 2**-26.1.

If 2.040452 ≤ x < 3.9192, ERF(x) = 1 - ERFC(x)

If 13.306 > x ≥ 3.9192, ERF(x) = 1

4. x ≥ 13.306

Results 1 and 0 are returned for ERF(x) and ERFC(x) respectively.

5. x < 0

Reduce to a case involving a positive argument by use of the identities:

ERF(x) = - ERF(-x)
and ERFC(x) = 2 - ERFC(-x).

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the absolute error in the argument multiplied by EXP(-x**2).

ERF(x): As the magnitude of the argument increases from 1, the effect of an argument error diminishes rapidly. For small x, the relative error of the result is of the order of the relative error of the argument.

ERFC(x): For x > 1, ERFC(x) is approximately EXP(-x**2)/(2*x). Thus the relative error in the result is approximately equal to the relative error in the argument multiplied by 2*x**2. For negative, or small positive, values of x, the relative error in the result is approximately equal to the absolute error in the argument multiplied by EXP(-x**2).

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHEEFSF			
ABS(x) ≤ 1	Uniform	0.115	0.853
1 < ABS(x) ≤ 2.04	Uniform	0.0370	0.107
2.04 < ABS(x) ≤ 3.9192	Uniform	0.0348	0.0597
IHEEFSC			
-3.8 < x < 0	Uniform	0.297	0.941
0 ≤ x ≤ 1	Uniform	0.126	0.692
1 < x ≤ 2.04	Uniform	0.374	1.98
2.04 < x ≤ 4	Uniform	0.369	1.27
4 < x ≤ 13.3	Uniform	8.22	15.1

<u>ERF, ERFC (long floating-point real)</u>

Module Name: IHEWEFL

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Error function (x)	ERF(x)	IHEEFLF
Complement of error function(x)	ERFC(x)	IHEEFLC

Function:

To calculate the error function of x or the complement of this function.

Method:

1. 0 ≤ x < 1

Compute ERF(x) by the following approximation:

$$ERF(x) = x*(a_0+a_1*x**2+a_2*x**4+....+a_{11}*x**22)$$

The coefficients were obtained by the minimax approximation in relative error of ERF(x)/x as a function of x**2 over the range 0 ≤ x**2 ≤1. The relative error of this approximation is less than 2**-56.9. The value of ERFC is computed as

ERFC(x) = 1-ERF(x)

2. 1 ≤ x < 2.040452

Compute ERFC(x) by the following approximation:

$$ERFC(x) = b_0+b_1*z+b_2z**2+....+b_{18}*z**18$$

where $z = x-T_0$ and $T_0 = 1.709472$. The coefficients were obtained by the minimax approximation in absolute error of the function $f(z) = ERFC(z+T_0)$ over the range -0.709472 ≤ z ≤ 0.330948. The absolute error of this approximation is less than 2**-60.3. The limits of this range and the value of the origin, T_0, were chosen to minimize the hexadecimal rounding errors.

The value of ERFC(x) within this range is between 1/256 and 0.1573. The value of ERF(x) is computed as

ERF(x) = 1-ERFC(x)

3. 2.040452 ≤ x < 13.306

Compute ERFC(x) by the following approximation:

ERFC(x) = EXP(-z)*F/x

where z = x**2 and

$$F = c_0+d_1/(z+c_1)+d_2/(z+c_2)+....+d_7/(z_7+c)$$

The coefficients of F were obtained by transforming a minimax rational approximation in absolute error of the function f(w) = ERFC(x)*x*EXP(x**2) over the range 13.306**-2 ≤ w ≤ 2.040452**-2 (where w=x**-2). This approximation is of the form

$$f(w)=(a_0+a_1*w+a_2*w**2+....a\ *w**7)/(b_7+b_1*w+b_2*w**2+....+b_6*w**6+w**7)$$

The absolute error of this approximation is less than 2**-57.9.

If 2.040452 ≤ x < 6.092368, ERF(x) = 1-ERFC(x)

If 13.360 > x ≥ 6.092368, ERF(x) = 1

4. x ≥ 13.306

Results 1 and 0 are returned for ERF(x) and ERFC(x) respectively.

5. x < 0

Reduce to a case involving positive arguments by use of the identities:

ERF(x) = -ERF(-x)
and ERFC(x) = 2-ERFC(-x).

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the absolute error in the argument multiplied by EXP(-x**2).

ERF(x): As the magnitude of the argument increases from 1, the effect of an argument error diminishes rapidly. For small x, the relative error of the result is of the order of the relative error of the argument.

ERFC(x): For x > 1, ERFC(x) is approximately EXP(-x**2)/(2*x). Thus the relative error in the result is approximately equal to the relative error in the argument multiplied by 2*x**2. For negative, or small positive, values of x, the relative error in the result is approximately equal to the absolute error in the argument multiplied by EXP(-x**2).

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHEEFLF			
ABS(x)≤1	Uniform	0.0257	0.193
1 < ABS(x) ≤ 2.04	Uniform	0.00946	0.0287
2.04 < ABS(x) < 6.092	Uniform	0.00802	0.0139
IHEEFLC			
-6 < x < 0	Uniform	0.0652	0.208
0 ≤ x ≤ 1	Uniform	0.0266	0.146
1 < x ≤ 2.04	Uniform	0.0913	0.426
2.04 < x < 4	Uniform	0.0865	0.326
4 ≤ x < 13.3	Uniform	1.96	3.51

FUNCTIONS WITH COMPLEX ARGUMENTS

SQRT (short floating-point complex)

Module Name: IHEWSQW

Entry Point: IHESQW0

Function:

To calculate the principal value of the square root of z, i.e., -pi/2 < argument of result ≤ pi/2.

Method:

1. Let SQRT(x+yI) = a+bI

2. Let SQRT((ABS(x) + ABS(x+yI))/2) = k*SQRT(w_1+w_2) = c

   v_1 = MAX(ABS(x),ABS(y)) and

   v_2 = MIN(ABS(x),ABS(y))

3. In the special case when either v_2 = 0 or v_1 >> v_2 let

   w_1 = v_2 and w_2 = v_1

   Let k = 1 if v_1 = ABS(x)

   k = 1/SQRT(2) if v_1 = ABS(y)

4. In the general case compute:

   F = SQRT(1/4+(1/4)*(v_1/v_2)**2)

   If ABS(x) is near the underflow threshold, then take

   w_1 = ABS(x), w_2 = v_1*2*F, and k = 1/SQRT(2)

   If v_1*F is near the overflow threshold, then take

   w_1 = ABS(x)/4, w_2 = v_1*F/2 and k = SQRT(2)

   In all other cases take

   w_1 = ABS(x)/2, w_2 = v_1*F, and k = 1

5. If c = 0 then a = b = 0

   If c ≠ 0 and x ≥ 0, then

   a = c, and
   b = y/(2*c)

   if c ≠ 0 and x < 0, then

   a = ABS(y/(2*c)), and
   b = SIGN(y)*c

Effect of an Argument Error:

Let z = r*EXP(hI), and
SQRT(z) = s*EXP(kI).

Then the relative error in s is approximately half the relative error in r, and the relative error in k is approximately equal to the relative error in h.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
Full range	Exponential	0.540	2.18

SQRT (long floating-point complex)

Module Name: IHEWSQZ

Entry Point: IHESQZ0

Function:

To calculate the principal value of the square root of z, i.e., -pi/2 < argument of result ≤ pi/2.

Method:

1. Let SQRT(x+yI) = a+bI

2. Let SQRT((ABS(x)+ABS(x+yI))/2)= k*SQRT(w_1+w_2) = c

   v_1 = MAX(ABS(x),ABS(y)) and
   v_2 = MIN(ABS(x),ABS(y))

3. In the special case when either v_2 = 0 or v_1 >> v_2 let

   w_1 = v_2 and w_2 = v_1

   Let k = 1 if v_1 = ABS(x)

   k = 1/SQRT(2) if v_1 = ABS(y)

4. In the general case compute:

   F = SQRT(1/4+(1/4)*(v_1/v_2)**2)

   If ABS(x) is near the underflow threshold, then take

   w_1 = ABS(x), w_2 = v_1*2*F, and k = 1/SQRT(2)

   If v_1*F is near the overflow threshold, then take

   w_1 = ABS(x)/4, w_2 = v_1*F/2 and k = SQRT(2)

In all other cases take

w_1 = ABS(x)/2, w_2 = v_1*F, and k = 1

5. If c = 0 then a = b = 0

If c ≠ 0 and x ≥ 0, then

a = c, and
b = y/(2*c)

if c ≠ 0 and x < 0, then

a = ABS(y/(2*c)), and
b = SIGN(y)*c

Effect of an Argument Error:

Let z = r*EXP(hI), and
SQRT(z) = s*EXP(kI).

Then the relative error in s is approximately half the relative error in r, and the relative error in k is approximately equal to the relative error in h.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
Full range	Exponential	0.131	0.492

EXP (short floating-point complex)

Module Name: IHEWEXW

Entry Point: IHEEXW0

Function: To calculate e to the power z.

Method:

Let z = x + yI.

Then REAL(EXP(z)) = EXP(x)*COS(y)
and IMAG(EXP(z)) = EXP(x)*SIN(y).

Effect of an Argument Error:

Let EXP(x + yI) = s*EXP(kI).

Then k = y, and the relative error in s is approximately equal to the absolute error in x.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
ABS(x) ≤ 170 ABS(y) ≤ pi/2	Uniform	0.646	2.40
ABS(x) ≤ 170 pi/2 < ABS(y)≤ 20	Uniform	0.628	2.28

Error and Exceptional Conditions:

O : ABS(y) ≥ 2**18*pi : error caused in real SIN routine (IHEWSNS)

H : OVERFLOW in real EXP routine (IHEWEXS)

EXP (long floating-point complex)

Module Name: IHEWEXZ

Entry Point: IHEEXZ0

Function: To calculate e to the power z.

Method:

Let z = x + yI.

Then REAL(EXP(z)) = EXP(x)*COS(y)
and IMAG(EXP(z)) = EXP(x)*SIN(y).

Effect of an Argument Error:

Let EXP(x + yI) = s*EXP(kI).

Then k = y, and the relative error in s is approximately equal to the absolute error in x.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
ABS(x) < 1 ABS(y) < pi/2	Uniform	0.187	0.614
ABS(x) < 20 ABS(y) < 20	Uniform	0.200	0.819

Error and Exceptional Conditions:

O : ABS(y) ≥ 2**50*pi : error caused in real SIN routine (IHEWSNL)

H : OVERFLOW in real EXP routine (IHEWEXL)

LOG (short floating-point complex)

Module Name: IHEWLNW

Entry Point: IHELNW0

Function:

To calculate the principal value of the natural log of z, i.e., -pi < imaginary part of result ≤ pi.

Method:

1. Let LOG(x+yI) = a+bI

2. Then, a = LOG(ABS(x+yI)) and b = ATAN(y,x)

3. LOG(ABS(x+yI)) is computed as follows:

Let v_1 = MAX(ABS(x),ABS(y)) and

v_2 = MIN(ABS(x),ABS(Y))

Let t be the exponent of v_1 (i.e., v_1 = m*16**t, 1/16 ≤ m < 1)

Let t_1 = t if t ≤ 0 or

t_1 = t-1 if t > 0 and

s = 16**t_1

Then LOG(ABS(x+yI)) = 4*t_1*LOG(2) + LOG((v_1/s)**2 + (v_2/s)**2)/2

Computation of v_1/s and v_2/s are carried out by suitable adjustment of the characteristics of v_1 and v_2; in particular, if v_2/s << 1, it is taken to be 0.

Effect of an Argument Error:

Let z = r*EXP(hI) and LOG(z) = u + vI.

Then the absolute error in u is approximately equal to the relative error in r. For the absolute error in v(= h = ATAN(y, x)), see corresponding paragraph for module IHEWATS.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
Full range	Exponential	0.396	1.89

Error and Exceptional Conditions:

O : x = y = 0, error in real LOG routine (IHEWLNS)

LOG (long floating-point complex)

Module Name: IHEWLNZ

Entry Point: IHELNZ0

Function:

To calculate the principal value of natural log of z, i.e., -pi < imaginary part of result ≤ pi.

Method:

1. Let LOG(x+yI) = a+bI

2. Then, a = LOG(ABS(x+yI)) and b = ATAN(y,x)

3. LOG(ABS(x+yI)) is computed as follows:

Let v_1 = MAX(ABS(x),ABS(y)) and

v_2 = MIN(ABS(x),ABS(y))

Let t be the exponent of v_1 (i.e., v_1 = m*16**t, 1/16 ≤ m < 1)

Let t_1 = t if t ≤ 0 or

t_1 = t-1 if t > 0 and

s = 16**t_1

Then LOG(ABS(x+yI)) = 4*t_1*LOG(2)+LOG((v_1/s)**2+(v_2/s)**2)/2

Computation of v_1/s and v_2/s are carried out by suitable adjustment of the characteristics of v_1 and v_2; in particular, if v_2/s << 1, it is taken to be 0.

Effect of an Argument Error:

Let z = r*EXP(hI) and LOG(z) = u + vI.

Then the absolute error in u is approximately equal to the relative error in r. For the absolute error in v(= h = ATAN(y, x)) see the corresponding paragraph for module IHEWATL.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
Full range	Exponential	0.125	0.542

Error and Exceptional Conditions:

O : x = y = 0, error caused in log routine (IHEWLNL)

SIN, SINH, COS, COSH (short floating-point complex)

Module Name: IHEWSNW

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Sin z	SIN(z)	IHESNWS
Hyperbolic sin z	SINH(z)	IHESNWZ
Cos z	COS(z)	IHESNWC
Hyperbolic cos z	COSH(z)	IHESNWK

Function:

To calculate sin z or hyperbolic sin z, or cos z or hyperbolic cos z.

Method:

Let z = x + yI.

```
Then REAL(SIN(z))  =  SIN(x)*COSH(y)
and  IMAG(SIN(z))  =  COS(x)*SINH(y);

     REAL(COS(z))  =  COS(x)*COSH(y)
and  IMAG(COS(z))  = -SIN(x)*SINH(y);

     REAL(SINH(z)) =  COS(y)*SINH(x)
and  IMAG(SINH(z)) =  SIN(y)*COSH(x);

     REAL(COSH(z)) =  COS(y)*COSH(x)
and  IMAG(COSH(z)) =  SIN(y)*SINH(x).
```

To avoid making calls to evaluate SINH and COSH separately, and thus frequently having to evaluate EXP twice for the same argument, SINH(u) is computed as follows:

1. u > 0.3465736

   SINH(u) = (EXP(u) - 1/EXP(u))/2.

2. 0 ≤ u ≤ 0.3465736

SINH(u)/u is approximated by a polynomial of the form a_0 + a_1*u**2 + a_2*u**4 (which has a relative error of less than 2**-26.4).

The coefficients were obtained by the minimax approximation in relative error of SINH(x)/x over the range 0 ≤ x**2 ≤ 0.12011 under the constraint that the first term shall be exactly 1.

3. u ≤ 0

   SINH(u) = -SINH(-u). Then
   COSH(u) = SINH(ABS(u)) + 1/EXP(ABS(u)).

Effect of an Argument Error:

Combine the effects on SIN, COS, SINH and COSH according to the method of evaluation described in the above paragraph.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHESNWS			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	1.17	3.37
IHESNWZ			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.878	2.75
IHESNWC			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	1.18	3.23
IHESNWK			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.968	3.11

Error and Exceptional Conditions:

O : IHESNWS, IHESNWC:
  ABS(x) ≥ 2**18*pi : error caused in real SIN routine (IHEWSNS)

  IHESNWZ, IHESNWK:
  ABS(y) ≥ 2**18*pi : error caused in real SIN routine (IHEWSNS)

H : OVERFLOW in real EXP routine (IHEWEXS)

SIN, SINH, COS, COSH (long floating-point complex)

Module Name: IHEWSNZ

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Sin z	SIN(z)	IHESNZS
Hyperbolic sin z	SINH(z)	IHESNZZ
Cos z	COS(z)	IHESNZC
Hyperbolic cos z	COSH(z)	IHESNZK

Function:

To calculate sin z or hyperbolic sin z, or cos z or hyperbolic cos z.

Method:

Let z = x + yI.

Then REAL(SIN(z)) = SIN(x)*COSH(y)
and IMAG(SIN(z)) = COS(x)*SINH(y);

REAL(COS(z)) = COS(x)*COSH(y)
and IMAG(COS(z)) = -SIN(x)*SINH(y);

REAL(SINH(z)) = COS(y)*SINH(x)
and IMAG(SINH(z)) = SIN(y)*COSH(x);

REAL(COSH(z)) = COS(y)*COSH(x)
and IMAG(COSH(z)) = SIN(y)*SINH(x).

To avoid making calls to evaluate SINH and COSH separately, and thus frequently having to evaluate EXP twice for the same argument, SINH(u) is computed as follows:

1. u ≥ 0.481212

   SINH(u) = (EXP(u) - 1/EXP(u))/2

2. 0 ≤ u < 0.481212

SINH(u)/u is approximated by a polynomial of the fifth degree in u**2 which has a relative error of less than 2**-56.07

The coefficients were obtained by the minimax approximation in relative error of SINH(x)/x over the range 0 ≤ x**2 ≤ 0.23156 under the constraint that the first term shall be exactly 1.

3. u < 0

   SINH(u) = -SINH(-u). Then
   COSH(u) = SINH(ABS(u)) + 1/EXP(ABS(u)).

Effect of an Argument Error:

Combine the effects on SIN,COS,SINH and COSH according to the method of evaluation described in the above paragraph.

Accuracy:

Arguments: Range	Arguments: Distribution	Relative Error *10**15: RMS	Relative Error *10**15: Maximum
IHESNZS			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	2.01	113
IHESNZZ			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.229	0.641
IHESNZC			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.311	3.83
IHESNZK			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.250	0.730

Error and Exceptional Conditions:

O : IHESNZS, IHESNZC:
  ABS(x) ≥ 2**50*pi: error caused in real SIN routine (IHEWSNL)

  IHESNZZ, IHESNZK:
  ABS(y) ≥ 2**50*pi: error caused in real SIN routine (IHEWSNL)

H : OVERFLOW in real EXP routine (IHEWEXL)

TAN, TANH (short floating-point complex)

Module Name: IHEWTNW

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Tan z	TAN(z)	IHETNWN
Hyperbolic tan z	TANH(z)	IHETNWH

Function:

To calculate tan z or hyperbolic tan z.

Method:

Let z = x + yI.

Then REAL(TAN(z)) =
TAN(x)*(1 - TANH(y)**2)/
(1 + (TAN(x)*TANH(y))**2),

IMAG(TAN(z)) =
TANH(y)*(1 + TAN(x)**2)/
(1 + (TAN(x)*TANH(y))**2).

TANH(z) = - (TAN(zI))I.

Effect of an Argument Error:

The absolute error caused in the result is approximately equal to the absolute error in the argument divided by ABS(COS(z)**2) for IHETNWN, or divided by ABS(COSH(z)**2) for IHETNWH. The relative error caused in the result is approximately twice the absolute error in the argument divided by ABS(SIN(2*z)) for IHETNWN, or divided by ABS(SINH(2*z)) for IHETNWH.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHETNWN			
ABS(x) < 1 ABS(y) < 9	Uniform	0.532	2.87
IHETNWH			
ABS(x) < 9 ABS(y) < 1	Uniform	0.524	2.74

Error and Exceptional Conditions:

I : OVERFLOW

O : ABS(u) ≥ 2**18*pi, where
u = x for IHETNWN,
u = y for IHETNWH.

H : OVERFLOW or ZERODIVIDE in real TAN routine (IHEWTNS)

TAN, TANH (long floating-point complex)

Module Name: IHEWTNZ

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Tan z	TAN(z)	IHETNZN
Hyperbolic tan z	TANH(z)	IHETNZH

Function:

To calculate tan z or hyperbolic tan z.

Method:

Let z = x + yI.

Then REAL(TAN(z)) =
TAN(x)*(1 - TANH(y)**2)/
(1 + (TAN(x)*TANH(y))**2),

IMAG(TAN(z)) =
TANH(y)*(1 + TAN(x)**2)/
(1 + (TAN(x)*TANH(y))**2).

TANH(z) = - (TAN(zI))I.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHETNZN			
ABS(x) < 1 ABS(y) < 9	Uniform	0.172	0.709
IHETNZH			
ABS(x) < 9 ABS(y) < 1	Uniform	0.174	0.692

Error and EXCEPTIONAL Conditions:

I : OVERFLOW

O : ABS(u) ≥ 2**50*pi, where
u = x for IHETNZN,
u = y for IHETNZH.

H : OVERFLOW or ZERODIVIDE in real TAN routine (IHEWTNL)

ATAN, ATANH (short floating-point complex)

Module Name: IHEWATW

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Arctan z	ATAN(z)	IHEATWN
Hyperbolic arctan z	ATANH(z)	IHEATWH

Function:

To calculate arctan z or hyperbolic arctan z.

Method:

Let z = x + yI.

Then REAL(ATANH(z)) = (ATANH(2*x/(1+x*x+y*y)))/2

IMAG(ATANH(z)) = (ATAN(2*y,(1-x*x-y*y)))/2

and ATAN(z) = -(ATANH(zI))I.

Effect of an Argument Error:

The absolute error in the result is approximately equal to the absolute error in the argument divided by (1 + z**2) in the case of IHEATWN, or by (1 - z**2) in the case of IHEATWH. Thus the effect may be considerable in the vicinity of z = ± 1I (IHEATWN) or ± 1 (IHEATWH).

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHEATWN			
Full range	Exponential	0.205	1.05
IHEATWH			
Full range	Exponential	0.224	1.22

Error and Exceptional Conditions:

P : IHEATWN: z = ±1I
IHEATWH: z = ±1

ATAN, ATANH (long floating-point complex)

Module Name: IHEWATZ

Entry Points:

Mathematical Function	PL/I Name	Entry Point
Arctan z	ATAN(z)	IHEATZN
Hyperbolic arctan z	ATANH(z)	IHEATZH

Function:

To calculate arctan z or hyperbolic arctan z.

Method:

Let z = x + yI.

Then REAL(ATANH(z)) = (ATANH(2*x/(1+x*x+y*y)))/2

IMAG(ATANH(z)) = (ATAN(2*y,(1-x*x-y*y)))/2

and ATAN(z) = -(ATANH(zI))I.

Effect of an Argument Error:

The absolute error in the result is approximately equal to the absolute error in the argument divided by (1 + z**2) in the case of IHEATZN, or by (1 - z**2) in the case of IHEATZH. Thus the effect may be considerable in the vicinity of z = ± I (IHEATZN) or ± 1 (IHEATZH).

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
IHEATZN			
Full range	Exponential	0.0517	0.438
IHEATZH			
Full range	Exponential	0.0562	0.409

Error and Exceptional Conditions:

P : IHEATZN: z = ±1I
IHEATZH: z = ±1

The Library supports the array built-in functions SUM, PROD, POLY, ALL and ANY, and also provides indexing routines for handling simple (i.e., consecutively stored) and interleaved arrays.

Input Data

The array function modules are distinguished from the other Library modules in that they all accept array arguments and perform their own indexing, whereas the other modules require that indexing should be handled by compiled code. Calls to conversion routines are included in the SUM, PROD and POLY modules with fixed-point arguments, so that these arguments are converted to floating-point as they are accessed (it should be noted that it is a requirement of the language that the results from these modules be in floating-point). On the other hand, the conversions necessary for the ALL and ANY modules (the arguments must be converted to bit string arrays) are not part of the modules and must be carried out before the modules are invoked.

Any restrictions on the admissibility of arguments are noted under the headings 'Range' and 'Error and Exceptional Conditions'.

Range: This states any ranges of arguments for which a module is valid. Arguments outside the ranges given are assumed to have been excluded before the module is called.

Error and Exceptional Conditions: These cover conditions which may result from the use of a routine; they are listed in four categories:

P -- Programmed conditions in the module concerned. Programmed tests are made where this is not too costly and, if an invalid argument is found, a branch is taken to the entry point IHEERRC of the execution error package(EXEP). This results in the printing of an appropriate message and in the ERROR condition being raised.

I -- Interruption conditions in the module concerned. For those routines where SIZE and FIXEDOVERFLOW are detected by programmed tests or where hardware interruptions may occur, the OVERFLOW, UNDERFLOW, and (when the conversion package is called) SIZE conditions pass to the ON condition error handler (IHEWERR) and are treated in the normal way. The machine is assumed to be enabled for all interruptions except significance, which is masked off.

O -- Programmed conditions in modules called by the module concerned. These occur when invalid arguments are detected in the module called.

H -- As I, but the interrupt conditions occur in the modules called by the module concerned.

Effect of Hexadecimal Truncation

See the corresponding section in the introduction to Chapter 3 for guidance to the accuracy of SUM, PROD, and POLY. If fixed-point arguments are passed to these functions, further errors may be introduced by conversions.

A summary of the Library array modules is given in Figures 7 and 8.

	Simple arrays, and interleaved arrays of variable-length strings	Interleaved string arrays with fixed-length elements
Indexers ALL, ANY	IHEWJXS IHEWNL1	IHEWJXI IHEWNL2
Note: IHEWJXI is used for indexing through interleaved arithmetic arrays.		

Figure 7. Bit String Array Functions and Array Indexers

PL/I function		Fixed-point arguments		Floating-point arguments			
				Short precision		Long precision	
		Simple	Interleaved	Simple	Interleaved	Simple	Interleaved
SUM	real	IHEWSSF	IHEWSMF	IHEWSSG	IHEWSMG	IHEWSSH	IHEWSMH
	complex	IHEWSSX	IHEWSMX	IHEWSSG	IHEWSMG	IHEWSSH	IHEWSMH
PROD	real	IHEWPSF	IHEWPDF	IHEWPSS	IHEWPDS	IHEWPSL	IHEWPDL
	complex	IHEWPSX	IHEWPDX	IHEWPSW	IHEWPDW	IHEWPSZ	IHEWPDZ
POLY	real	IHEWYGF		IHEWYGS		IHEWYGL	
	complex	IHEWYGX		IHEWYGW		IHEWYGZ	

Figure 8. Arithmetic Array Functions

ARRAY INDEXERS

Indexer for Simple Arrays

Module Name: IHEWJXS

Entry Points:

Element Address	Entry Point
Bit addresses	IHEJXSI
Byte addresses	IHEJXSY

Function:

To find the first and last elements of an array. Their addresses are returned, in general registers 0 and 1 respectively, as bit addresses (IHEJXSI) or byte addresses (IHEJXSY).

Method:

The address of the virtual origin B of the array (i.e., the address that would correspond to the element A(0,..0)) is obtained as a byte address for IHEJXSY, or a bit address for IHEJXSI, by referring to the first word of the array dope vector (ADV).

$$\text{Address of first element} = B + \sum_{i=1}^{n} M_i L_i$$

$$\text{Address of last element} = B + \sum_{i=1}^{n} M_i U_i$$

where M is the multiplier for the ith dimension

L is the lower bound for the ith dimension

U is the upper bound for the ith dimension, and

n is the number of dimensions.

Range:

0 < number of dimensions < 2**22

Indexer for Interleaved Arrays

Module Name: IHEWJXI

Entry Points:

Operation	Entry Point
Initialization for bit addresses	IHEJXII
Initialization for byte addresses	IHEJXIY
Elements after the first	IHEJXIA

Function:

To find the next element of an array and to return its bit or byte address in general register 1.

Entry point IHEJXII is used to initialize the routine for bit addresses and to provide the address of the first element in the array; IHEJXIY does the same for byte addresses. Entry point IHEJXIA is used thereafter to obtain the addresses of subsequent elements of the array; one address is returned for each entry into the routine.

Method:

Arrays are stored in row major order. Let L_i be the lower bound and U_i the upper bound of the ith dimension, and n the number of dimensions. Starting with the element $A(L_1, L_2, \ldots\ldots L_n)$, the routine varies the subscripts through their ranges to $A(U_1, U_2, \ldots\ldots U_n)$, changing the nth subscript most rapidly; in this way the elements are referenced in the order in which they are stored.

The routine does not deal with actual subscript values but calculates the extent E_i(= U_i - L_i + 1) of each dimension and uses this as a count that varies from E_i to 1 for subscript values L_i to U_i. A 'base address' for each dimension is maintained and, for the ith dimension, is defined as the address of the element with ith subscript equal to its lowest bound L_i and with all other subscripts at their current values.

Thus initially the base addresses are all equal to the address of A(L_1,L_2,...... L_n). Each subsequent element address is generated from the previous one by adding the multiplier M_n from the array dope vector (ADV) and reducing the subscript count by 1. When the count for the ith dimension has been reduced from E_i to 1 it is reset to E_i, M_{i-1} is added to the (i-1)th dimension's base address and the count for this dimension is decreased by one.

This new base is the starting point for further increments by M_n. When a new base address is calculated, the base addresses for all higher dimensions ((i+1), (i+2),......n) is set equal to the ith base address.

Range:

0 < number of dimensions < 2**22

ARRAY FUNCTIONS

ALL (X), ANY (X)

Module Names:

Arguments	Module Name
Simple arrays and interleaved arrays with variable-length elements	IHEWNL1
Interleaved arrays with fixed-length elements	IHEWNL2

Entry Points:

PL/I Function	Entry Point
ALL(X), ANY(X), byte-aligned	IHENL1A IHENL2A
ALL(X), any alignment	IHENL1L IHENL2L
ANY(X), any alignment	IHENL1N IHELN2N

Function:

The argument X is a bit string array (any necessary conversion having been performed prior to the invocation of these modules). The result is a scalar bit string of length equal to the greatest of the current lengths of the elements of X.

ALL(X): the ith bit of the result is 1 if the ith bits of all the elements of X exist and are 1; otherwise it is 0.

ANY(X): the ith bit of the result is 1 if the ith bit of any element of X exists and is 1; otherwise it is 0.

Method:

For byte-aligned string arrays, AND (IHEWBSA) and OR (IHEWBSO) are used for ALL and ANY respectively; for string arrays with any alignment BOOL (IHEWBSF) is used with appropriate parameter bits.

The elements of the array are passed to IHEWBSA, IHEWBSO, or IHEWBSF one at a time, and the result is developed in the target field. For the first call to any of these logical modules the first element of the array serves as both first and second source arguments. For subsequent calls, the result already developed in the target field is the first argument and the next element of the array is the second argument.

Range:

Bit strings are limited to a maximum of 32,767 bits.

SUM (X)

Module Names and Entry Points:

Simple Arrays

Arguments	Module Name	Entry Point
Fixed, real	IHEWSSF	IHESSF0
Fixed, complex	IHEWSSX	IHESSX0
Short float		
real	IHEWSSG	IHESSGR
complex	IHEWSSG	IHESSGC
Long float		
real	IHEWSSH	IHESSHR
complex	IHEWSSH	IHESSHC

Interleaved Arrays

Arguments	Module Name	Entry Point
Fixed, real	IHEWSMF	IHESMF0
Fixed, complex	IHEWSMX	IHESMX0
Short float		
real	IHEWSMG	IHESMGR
complex	IHEWSMG	IHESMGC
Long float		
real	IHEWSMH	IHESMHR
complex	IHEWSMH	IHESMHC

Function:

To produce a scalar with a value which is the sum of all the elements of the array argument.

Method:

The elements of the array are added to the current sum in row major order.

For fixed-point arguments each element is converted to floating-point by using the PL/I Library conversion package.

For a complex argument, the summation of the real parts is performed before the summation of the imaginary parts is begun in modules IHEWSSG and IHEWSSH, while the two sums are developed concurrently in other modules.

Error and Exceptional Conditions:

I : OVERFLOW, UNDERFLOW

H : IHEWSSF, IHEWSSX, IHEWSMF, IHEWSMX: ABS(element of the array) > 7.2*10** 75: SIZE condition caused in conversion package

PROD (X)

Module Names and Entry Points:

Simple Arrays

Arguments	Module Name	Entry Point
Fixed, real	IHEWPSF	IHEPSF0
Fixed complex	IHEWPSX	IHEPSX0
Short float		
real	IHEWPSS	IHEPSS0
complex	IHEWPSW	IHEPSW0
Long float		
real	IHEWPSL	IHEPSL0
complex	IHEWPSZ	IHEPSZ0

Interleaved Arrays

Arguments	Module Name	Entry Point
Fixed, real	IHEWPDF	IHEPDF0
Fixed, complex	IHEWPDX	IHEPDX0
Short float		
real	IHEWPDS	IHEPDS0
complex	IHEWPDW	IHEPDW0
Long float		
real	IHEWPDL	IHEPDL0
complex	IHEWPDZ	IHEPDZ0

Function:

To produce a scalar with a value which is the product of all the elements in the array argument.

Method:

The elements of the array are used in row major order to multiply the current product.

For fixed-point arguments, each element is converted to floating-point by using the PL/I Library conversion package.

Error and Exceptional Conditions:

I : OVERFLOW, UNDERFLOW

H : IHEWPSF, IHEWPSX, IHEWPDF, IHEWPDX: ABS(element of the array) > 7.2*10** 75: SIZE condition caused in conversion package

POLY (A,X)

Module Names and Entry Points:

Arguments	Module Name	Entry Point
Fixed, real		
vector X	IHEWYGF	IHEYGFV
scalar X	IHEWYGF	IHEYGFS
Fixed, complex		
vector X	IHEWYGX	IHEYGXV
scalar X	IHEWYGX	IHEYGXS
Short float, real		
vector X	IHEWYGS	IHEYGSV
scalar X	IHEWYGS	IHEYGSS
Short float, complex		
vector X	IHEWYGW	IHEYGWV
scalar X	IHEWYGW	IHEYGWS
Long float, real		
vector X	IHEWYGL	IHEYGLV
scalar X	IHEWYGL	IHEYGLS
Long float, complex		
vector X	IHEWYGZ	IHEYGZV
scalar X	IHEWYGZ	IHEYGZS

Function:

Vector X:

Let the arguments be arrays declared as A(m:n) and X(p:q). Then the function computed is:

$$A(m) + \sum_{j=1}^{n-m} A(m+j) * \prod_{i=0}^{j-1} X(p+i)$$

unless n = m, when result is A(m).

If q - p < n - m - 1, then, for p + i > q, X(p + i) = X(q).

Scalar X:

This may be interpreted as a special case of vector X, that is, a vector with one element, X(1), which is equal to X. Then the function computed is:

$$\sum_{j=0}^{n-m} A(m + j)*X**j$$

A floating-point result is obtained in both cases.

Method:

1. Vector X, $(q - p \geq n - m - 1)$:

POLY(A,X) is evaluated by nested multiplication and addition, i.e.,

```
(...(A(n)*X(k) + A(n-1))*X(k-1) +
      A(n-2))* ... + A(m+1))*X(p) + A(m)
```

where $k = p + n - m - 1$.

2. Vector X, $(q - p < n - m - 1)$:

In the expression above, the terms in X with subscript ranging from k down to q are all made equal to X(q). The evaluation is treated as for scalar X until sufficient terms in X have been made equal to X(q), when the computation continues as in (1.).

3. Scalar X:

Terms in X with subscript ranging from k to p are equal to X.

For fixed-point arguments each element is converted to floating-point, by using the PL/I Library conversion package.

Error and Exceptional Conditions:

I : OVERFLOW, UNDERFLOW

H : IHEWYGF, IHEWYGX:
ABS(element of the array) > 7.2*10**75: SIZE condition caused in conversion package

INDEX

GC28-2046-1

PL/I Lib. Subroutines Printed in U.S.A. GC28-2046-1

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]