File No. S360-29
Form C28-6590-0

OS

Systems Reference Library

IBM System/360 Operating System

PL/I Subroutine Library

Computational Subroutines

Program Number 360S-LM-512

This publication gives details of the computational subroutines available in the PL/I Library. These subroutines are used by the PL/I (F) 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 supported by the PL/I Library; the compiler generates in-line code for a small number of them. The details provided include timing figures, summaries of the mathematical methods used, and (where appropriate) figures for range and accuracy. This information is intended to be of interest chiefly to those programmers concerned with the performance of computational subprograms.

PREFACE

This publication provides the PL/I programmer with detailed information about the computational subroutines which are part of the OS/360 PL/I Library.

The reader is assumed to be a programmer with a particular concern for performance information associated with individual modules. 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, Form A22-6821

IBM System/360 Operating System PL/I: Language Specifications, Form C28-6571

IBM System/360 Operating System: PL/I (F) Programmer's Guide, Form C28-6594

IBM System/360 Operating System: Assembler Language, Form C28-6514

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Department D39,1271 Avenue of the Americas, New York, N.Y. 10020.

INTRODUCTION

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

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

These subroutines have been designed to allow their use in a multi-tasking environment.

This publication gives detailed information in each of the four sections mentioned above with respect to performance, 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 of investigating the error by use of the ONCODE built-in function in his ON ERROR unit and of making a choice on the action which he wishes to take. Full details of the diagnostic messages printed at object time and of the ONCODE values associated with them are specified in IBM System/360 Operating System: PL/I (F) Programmer's Guide, Form C28-6594.

The Library string package contains modules for handling bit and character strings. 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.

Execution times are based on information in IBM System/360 Instruction Timing Information, Form A22-6825. They are intended to be simple enough to give a good general guide to performance while averaging out many logical variations, the effect of which is comparatively small.

A complete list of the modules provided in the Library string package is given in Figure 1.

PL/I Operation	PL/I Function	Bit String General	Bit String Byte-aligned	Character String
'And' (&)	-	Use BOOL	IHEBSA	-
'Or' (|)	-	Use BOOL	IHEBSO	-
'Not' (¬)	-	Use BOOL	IHEBSN	-
Concatenate (||)	REPEAT	IHEBSK	-	IHECSK
Compare	-	IHEBSD	IHEBSC	IHECSC
Assign	-	Use IHEBSK	IHEBSM	IHECSM
Fill	-	IHEBSM	-	IHECSM
-	HIGH/LOW	-	-	IHECSM
-	SUBSTR	IHEBSS	-	IHECSS
-	INDEX	IHEBSI	-	IHECSI
-	BOOL	IHEBSF	-	-

Figure 1. Bit and Character String Operations and Functions

BIT STRING OPERATIONS

The 'And' Operator (&)

Module Name: IHEBSA

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 of those 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.

Implementation:

- Module size: 296 bytes
- Execution time:

  Let L_1 = the length of the shorter string

  L_2 = the difference between the string lengths

  M_i = FLOOR((L_i - 1)/2048)

  N_i = MOD(L_i, 2048) - 1

  P_i = FLOOR(N_i/8)

  S = SIGN(L_2)

  i = 1,2

  Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

  $$a + b*M_1 + c*P_1 + S*(d + e*M_2 + f*P_2)$$

	30	40	50	65	75
a	2184	693	296	86	57
b	2569	1543	757	243	202
c	9	5.6	2.8	0.9	0.7
d	363	117	48	15	9
e	1171	696	316	107	95
f	4	2.5	1.1	0.4	0.3
a'	1437	466	200	60	43

Note: If the two strings are equal in length and end on a byte boundary, then a is replaced by a'.

The 'Or' Operator (|)

Module Name: IHEBSO

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 of those 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 calculated for it; the remainder of the target field is left unchanged.

Implementation:

- Module size: 312 bytes
- Execution time:

  Let L_1 = the length of the shorter string

  L_2 = the difference between the string lengths

  M_i = FLOOR((L_i - 1)/2048)

  N_i = MOD(L_i,2048) - 1

  P_i = FLOOR(N_i/8)

  S = SIGN(L_2)

  i = 1,2

  Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

  $$a + b*M_1 + c*P_1 + S*(d + e*M_2 + f*P_2)$$

	30	40	50	65	75
a	2294	724	316	97	65
b	2569	1543	757	243	202
c	9	5.6	2.8	0.9	0.7
d	364	117	53	18	15
e	1146	688	311	103	92
f	4	2.5	1.1	0.4	0.3
a'	1437	466	200	60	43

Note: If the two strings are equal in length and end on a byte boundary, then a is replaced by a'.

The 'Not' Operator (¬)

Module Name: IHEBSN

Entry Point: IHEBSN0

Function:

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

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.

Implementation:

- Module size: 192 bytes
- Execution time:

Let L_1 = the string length

M_1 = FLOOR((L_1 - 1)/2048)

N_1 = MOD(L_1,2048) - 1

P_1 = FLOOR(N_1/8)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*M_1 + c*P_1$$

or, if MOD(L_1,8) = 0, from:

$$a' + b*M_1 + c*P_1$$

	30	40	50	65	75
a	1599	530	245	72	52
b	2544	1533	754	242	201
c	9	5.6	2.8	0.9	0.7
a'	1183	429	192	55	40

Bit String Concatenate/REPEAT

Module Name: IHEBSK

Entry Points:

Operation	Entry point
Concatenate (||)	IHEBSKK
REPEAT(Bit string,n)	IHEBSKR

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.

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, and the maximum length of the target field. Both entry points use a loop which obtains data from the source fields, aligns it correctly, and moves it to the target field. The remainder of the target field beyond the calculated current length is left unchanged.

Implementation:

- Module size: 328 bytes
- Execution time:

Let L_1,L_2 = the lengths of the two strings

F_i = FLOOR(L_i/32)

R = n + 1, if n ≥ 0
  = 1, if n < 0

i = 1,2

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the formula for the appropriate entry point:

IHEBSKK: $a + b*(F_1 + F_2)$

IHEBSKR: $c + R*(d + b*F_1)$

	30	40	50	65	75
a	3497	1311	445	126	74.7
b	345	111	42	11.8	8.1
c	594	187	73	16.4	11.0
d	1505	479	196	56.3	33.2

Bit String Byte-aligned Comparison

Module Name: IHEBSC

Entry Point: IHEBSC0

Function:

To compare two byte-aligned bit strings and to return a condition code as bits 2 and 3 of a full-word 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.

Implementation:

- Module size: 272 bytes
- Execution time:

Let L_1 = the number of bytes compared up to the first inequality

L_2 = the number of bytes in the additional part of the longer string, compared to zeros if necessary

M_1 = FLOOR $((L_1 - 1)/256)$

N_1 = MOD $(L_1 - 1,256)$

S = SIGN (L_2)

Then the appropriate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*M_1 + c*N_1 + S*(d + e*L_2)$$

	30	40	50	65	75
a	981	304	128	40.3	27.0
b	1477	794	289	114	110
c	5	2.8	1.0	0.4	0.4
d	846	256	107	29.2	18.8
e	99	40	16	4.9	3.6

Bit String General Comparison

Module Name: IHEBSD

Entry Point: IHEBSD0

Function:

To compare two bit strings and return a condition code as bits 2 and 3 of a full-word 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.

Implementation:

- Module size: 192 bytes
- Execution time:

Let F_1 = the number of 32-bit words compared up to the first inequality

F_2 = the number of 32-bit words compared to zero if necessary

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*F_1 + c*F_2$$

	30	40	50	65	75
a	1113	344	120	31.7	18.2
b	624	194	76	21.3	14.9
c	437	139	55	15.3	11.4

Bit String Assign/Fill

Module Name: IHEBSM

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.

Implementation:

- Module size: 384 bytes
- Execution time:

Let B_1 = the bit length of the field to be assigned

B_2 = the bit length for zero filling

I_i = FLOOR(P_i/8)

M_i = FLOOR (L_i/256)

N_i = MOD(L_i,256)

S = SIGN (L_2)

i = 1,2

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the formula for the appropriate entry point:

IHEBSMF: $f + b*M_1 + c*N_1 + S*(g + e*M_2 + c*N_2)$

IHEBSMV: $a + b*M_1 + c*N_1$

IHEBSMZ: $d + e*M_2 + c*N_2$

	30	40	50	65	75
a	1379	442	192	58.9	41.2
b	1196	706	319	108	95.8
c	4	2.5	1.1	0.4	0.3
d	1865	589	243	73.6	48.0
e	1171	696	316	107	92.9
f	1897	623	266	81.1	56.6
g	1086	349	150	46.7	31.8
a'	1079	343	140	42.9	29.1
f'	1597	524	214	65.1	44.5

Note: a' and f' replace a and f respectively if MOD $(B_1, 8) = 0$

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 REPEAT entry IHEBSKR with n equal to 0. Any filling required for fixed length strings can then be obtained using the IHEBSMZ entry described above.

BIT STRING FUNCTIONS

Bit String SUBSTR

Module Name: IHEBSS

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.

Implementation:

- Module size: 192 bytes
- Execution times:

  Approximate execution time in microseconds for the System/360 models given below is shown for each entry point:

Entry Point	30	40	50	65	75
IHEBSS2	1184	378	152	42.6	28.8
IHEBSS3	1315	415	168	46.7	31.7

Bit String INDEX

Module Name: IHEBSI

Entry Point: IHEBSI0

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.

Implementation:

- Module size: 296 bytes
- Execution time:

Let F_i = the length (in words) actually processed at the ith comparison (the length up to the first inequality or the length of the second string if no inequality is found)

B = (length of first string) - (length of second string)

I = the resulting index, except when this is 0 with B ≥ 0, in which case I = B + 1

J = MOD(I,32)

i = 1,n

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*I + c*J + d*\sum_{i=1}^{I} (F_i - 1)$$

	30	40	50	65	75
a	2026	646	246	63.6	37.4
b	422	124	48	13.9	9.2
c	104	41	18	4.3	3.1
d	543	180	73	18.9	13.2

Bit String BOOL (Boolean Function)

Module Name: IHEBSF

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 - $n_1n_2n_3n_4$- 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.

Implementation:

- Module size: 480 bytes
- Execution time:

Let B_1 = the bit length of the shorter string

B_2 = the difference in bit lengths of the strings

F_i = CEIL(B_i/32)

i = 1,2

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*F_1 + c*F_2$$

	30	40	50	65	75
a	2275	767	291	75.0	46.5
b	728	230	95	30.0	19.4
c	501	162	68	21.8	15.2

CHARACTER STRING OPERATIONS

Character String Concatenate/REPEAT

Module Name: IHECSK

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 $n \le 0$, 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, and the maximum length of the target field. The source strings are then moved to the target. Characters beyond the range of the target current length remain unaltered.

Implementation:

- Module size: 208 bytes
- Execution time:

Let L_1, L_2 = the lengths of the source strings

M_i = FLOOR($(L_i - 1)/256$)

N_i = MOD($L_i - 1, 256$)

$R = n + 1$, if $n \ge 0$
$= 1$, if $n < 0$

$i = 1,2$

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHECSKK: $a + b*(M_1 + M_2) + c*(N_1 + N_2)$

IHECSKR: $R*(b*M_1 + c*N_1 + d) + e$

	30	40	50	65	75
a	1407	491	209	63.7	46.4
b	1196	706	319	108	95.8
c	4	2.5	1.1	0.4	0.3
d	703	236	101	31.7	22.0
e	108	54.6	26.0	5.4	6.2

Character String Compare

Module Name: IHECSC

Entry Point: IHECSC0

Function:

To compare two character strings and to return a condition code as bits 2 and 3 of a full-word 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.

Implementation:

- Module size: 200 bytes
- Execution time:

Let L_1 = the length of the strings compared up to the first inequality (proceeding left to right)

L_2 = the length of the additional part of the longer string compared with blanks if necessary

M_i = FLOOR($(L_i - 1)/256$)

N_i = MOD($L_i - 1, 256$)

S = SIGN(L_2)

$i = 1,2$

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*M_1 + c*N_1 + S*(d + e*M_2 + c*N_2)$$

	30	40	50	65	75
a	849	284	116	36.8	27.1
b	1469	790	289	114	110
c	5	2.8	1.0	0.4	0.4
d	620	210	88	27.3	21.7
e	1474	794	290	114	110

Character String Assign/Fill/HIGH/LOW

Module Name: IHECSM

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.

Implementation:

- Module size: 280 bytes
- Execution time:

Let L_1 = either the specified length (for IHECSMH/L/B) or the length for blank filling (IHECSMF)

L_2 = the length of the shorter of the source and target fields (IHECSMF/V)

M_1 = FLOOR((L_1 - 2)/256)

N_1 = MOD(L_1 - 2,256)(where $L_1 \geq 2$ for both M_1 and N_1)

M_2 = FLOOR((L_2 - 1)/256)

N_2 = MOD(L_2 - 1,256)

S = SIGN(L_1)

Then the approximate execution times in microseconds for the System /360 models given below are obtained from the following formulas:

IHECSMF: $h + g*M_2 + c*N_2 + S*(i + b*M_1 + c*N_1)$

IHECSMV: $f + g*M_2 + c*N_2$

IHECSMB: $e + b*M_1 + c*N_1$

IHECSMH: $a + b*M_1 + c*N_1$

IHECSML: $d + b*M_1 + c*N_1$

	30	40	50	65	75
a	830	282	120	37.0	26.8
b	1171	696	316	107	95.2
c	4	2.5	1.1	0.4	0.3
d	799	268	116	35.5	25.6
e	892	300	130	40.5	28.0
f	790	267	114	34.6	24.9
g	1196	706	319	108	95.8
h	1298	446	190	57.4	40.6
i	400	129	57.6	18.5	13.7

CHARACTER STRING FUNCTIONS

Character String SUBSTR

Module Name: IHECSS

Entry Points:

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.

Implementation:

- Module size: 176 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are as shown for each entry point:

Entry Point	30	40	50	65	75
IHECSS2	887	310	127	36.9	26.2
IHECSS3	1018	347	143	41.0	29.1

Character String INDEX

Module Name: IHECSI

Entry Point: IHECSI0

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 point required is located by comparing in storage the second string with a corresponding number of characters in the first string.

Implementation:

- Module size: 168 bytes
- Execution time:

Let L_i = the length processed at the ith comparison (the length up to the first inequality, or the length of the second string if no inequality is found)

M_i = FLOOR(L_i/256)

N_i = MOD(L_i, 256)

B = (length of first string) - (length of second string)

I = the resulting index, except when this is 0 with B ≥ 0, in which case I = B + 1

i = 1,n

Then the approximate execution times in microseconds for the System/360 models given below is obtained from the following formula:

$$a + b*I + \sum_{i=1}^{I} (c*M_i + d*N_i)$$

	30	40	50	65	75
a	954	314	127	38.2	24.5
b	160	59.9	28.8	10.1	9.3
c	1427	776	280	110	107
d	5	2.8	1.0	0.4	0.4

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 parts of the conversion package.

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 which a module assumes to have been excluded prior to its being 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 -- Interrupt 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 interrupt conditions occur in the modules called by the module concerned.

Speed

The average execution times given are based on the IBM System/360 Instruction Timing Information, Form A22-6825. These times include the times taken by the modules called.

A summary of the Library arithmetic modules is given in Figures 2 and 3.

ARITHMETIC OPERATIONS				
Operation	Binary fixed	Decimal fixed	Short float	Long float
Real Operations				
Integer exponentiation: x**n	IHEXIB	IHEXID	IHEXIS	IHEXIL
General exponentiation: x**y	-	-	IHEXXS	IHEXXL
Shift-and-assign, Shift-and-load	-	IHEAPD	-	-
Complex Operations				
Multiplication/division: $z_1*z_2, z_1/z_2$	IHEMZU	IHEMZV	-	-
Multiplication: z_1*z_2	-	-	IHEMZW	IHEMZZ
Division: z_1/z_2	-	-	IHEDZW	IHEDZZ
Integer exponentiation: z**n	IHEXIU	IHEXIV	IHEXIW	IHEXIZ
General exponentiation: z_1**z_2	-	-	IHEXXW	IHEXXZ

Figure 2. Arithmetic Operations

ARITHMETIC FUNCTIONS				
Function	Binary fixed	Decimal fixed	Short float	Long float
Real Arguments				
MAX, MIN	IHEMXB	IHEMXD	IHEMXS	IHEMXL
ADD	-	IHEADD	-	-
Complex Arguments				
ADD	-	IHEADV	-	-
MULTIPLY	IHEMPU	IHEMPV	-	-
DIVIDE	IHEDVU	IHEDVV	-	-
ABS	IHEABU	IHEABV	IHEABW	IHEABZ

Figure 3. Arithmetic Functions

REAL OPERATIONS

Positive Integer Exponentiation (fixed binary)

Module Name: IHEXIB

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.

Implementation:

- Module size: 88 bytes
- Execution time:

  Let M = number of significant bits in the exponent

  N = number of 1 bits in the exponent

  Then the approximate execution times in microseconds for the System /360 models given below are obtained from the following formula:

  -a + b*M + c*N

	30	40	50	65	75
a	238	9	5.5	-1.46	-0.6
b	708	188	63.8	15.6	9.8
c	335	94	33.0	6.1	3.9

<u>Positive Integer Exponentiation (fixed decimal)</u>

Module Name: IHEXID

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 $n*(p + 1) - 1 \leq 15$. This implies that $n \leq 16/(p + 1) \leq 8$ for all such cases and, in fact, this module will operate only for the range $0 < n \leq 8$.

Implementation:

- Module size: 136 bytes
- Execution time:

  Let M = number of significant bits in the exponent

  N = number of 1 bits in the exponent

  Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

  -a + b*M + c*N

	30	40	50	65	75
a	580	48	-16.3	37.1	44.5
b	1113	279	91	45.5	40.3
c	752	192	65	38.8	37.2

<u>Integer Exponentiation (floating-point)</u>

Module Names and Entry Points:

Argument	Module name	Entry point
Short float	IHEXIS	IHEXIS0
Long float	IHEXIL	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 non-zero, 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 was 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 ≤ exponent ≤ 2**(k + 1) - 1, where k has integral values from 1 to 9 inclusive.

IHEXIS

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

IHEXIL

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.

Implementation:

- Module size: IHEXIS 152 bytes
  IHEXIL 152 bytes

- Execution time:

  Let M = number of significant bits in the exponent

  N = number of 1 bits in the exponent

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

a + b*M + c*N for positive exponents

a'+ b*M + c*N for negative exponents

IHEXIS

	30	40	50	65	75
a	-104	23	29	10.7	7.6
b	701	176	56	14.3	8.7
c	342	90	26	5.7	3.2
a'	552	171	57	18.6	12.7

IHEXIL

	30	40	50	65	75
a	-1535	-322	0.7	4.1	3.7
b	1441	355	73	17.5	10.7
c	1082	269	42	8.9	5.2
a'	1055	180	78	20.0	11.9

Other Information:

IHEXIS: For large exponents, for example, those greater than 1023, it is generally faster and more accurate to use the module IHEXXS rather than IHEXIS, passing the exponent as a floating-point argument. However, it should be noted that IHEXXS 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	IHEXXS	IHEXXS0
Long float	IHEXXL	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 $y \le 0$. When $x \ne 0$ 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 $y \le 0$

O : a. $x < 0$ with $y \ne 0$: error caused in LOG routine

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

Implementation:

- Module size: IHEXXS: 144 bytes
  IHEXXL: 152 bytes

- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEXXS	9809	2861	902	236	143
IHEXXL	30444	7453	1579	358	203

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

Module Name: IHEAPD

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 $p_1 \le 31$ and $p_2 \le 15$.

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

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

Implementation:

- Module size: 360 bytes

- Execution times:

  Let S = (Target scale factor) (argument scale factor)

  $f_1 = 0$ if $S \ge 31$
  $\quad = 1$ if $S < 31$

  $f_2 = 0$ if $ABS(S) \ge p$
  $\quad = 1$ if $ABS(S) < p$
  (where p is the precision of the argument)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHEAPDA:

(i) $S > 0$: $t_6 + t_2*f_1$

(ii) $S = 0$: t_7

(iii) $S < 0$: $t_8 + t *f_2$

IHEAPDB:

(i) $S > 0$: $t_1 + t_2*f_1$

(ii) $S = 0$: t_3

(iii) $S < 0$: $t_4 + t_5*f_2$

	30	40	50	65	75
t_1	1314	513	241	68.4	53.5
t_2	451	191	64.3	17.1	15.1
t_3	1117	446	202	60.1	47.2
t_4	1220	386	171	49.5	36.9
t_5	502	221	94.2	37.3	21.8
t_6	1996	736	345	95.6	70.9
t_7	1799	670	305	87.3	64.5
t_8	1902	610	274	76.7	54.4

COMPLEX OPERATIONS

Multiplication/Division (fixed binary)

Module Name: IHEMZU

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.

Implementation:

- Module size: 240 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas and tables:

Entry Point	30	40	50	65	75
IHEMZUM	2421	689	256	56.6	37.1

IHEMZUD

Let M = number of significant bits in Z_2*CONJG(Z_2)

N = FLOOR(M/4 - 8)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

a + b + c + N*d

	30	40	50	65	75
a	3021	1340	420	94.1	64.2
b	338	79	30	5.7	2.7
c	78	24	18	6.6	2.1
d	213	56	22	6.2	3.8

Note: $b = 0$ if $M \le 31$
$c = d = 0$ if $M \le 32$

Multiplication/Division (fixed decimal)

Module Name: IHEMZV

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.

Implementation:

- Module size: 672 bytes
- Execution time:

Let (p,q), (r,s) = precisions of the operands

L_1 = FLOOR(p/2) + 1

L_2 = FLOOR(r/2) + 1

T = c**2 + d**2 ≥ 10**(15 - 2*s) (i.e., T = 1 or 0 depending on whether the relation is true or false)

Then approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHEMZVM: $a + b*L_1 + c*L_2 + d*L_1*L_2$

IHEMZVD: $e + f*L_1 + g*L_2 + h*L_1*L_2 + j*L_2**2 + T*(k + m*L_2)$

	30	40	50	65	75
a	2605	956	457	128	103
b	246	105	49	17	13
c	30	33	19	5.3	3.5
d	112	15	0	4.0	4.7
e	14525	4203	1144	626	508
f	246	105	49	17	12
g	168	103	41	16	11
h	112	15	0	4.0	4.7
j	56	7.5	0	2.0	2.3
k	731	251	131	41.2	30.2
m	12	16	4.3	0.8	0.3

Other Information:

It should be noted from the timings for multiplication that 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	IHEMZW	IHEMZW0
Long float	IHEMZZ	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

Implementation:

- Module size: IHEMZW 64 bytes
  IHEMZZ 64 bytes

- Execution times:

  Approximate execution time in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEMZW	1979	550	172	41.9	23.3
IHEMZZ	5115	1307	251	62.3	31.3

Division (floating-point)

Module Names and Entry Points:

Argument	Module name	Entry point
Short float	IHEDZW	IHEDZW0
Long float	IHEDZZ	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

Implementation:

- Module size: IHEDZW 104 bytes
  IHEDZZ 104 bytes

- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEDZW	3546	875	221	60.8	35.7
IHEDZZ	11741	2515	234	92.5	51.1

Positive Integer Exponentiation (fixed binary)

Module Name: IHEXIU

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 IHEMZU.

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 ≤ 31. This implies that n ≤ 32/(p + 1) ≤ 16 for all such cases.

Implementation:

- Module size: 120 bytes

- Execution time:

  Let M = number of significant bits in the exponent

  N = number of 1 bits in the exponent

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

-a + b*M + c*N

	30	40	50	65	75
a	4169	1183	405	101	69.5
b	2409	822	306	73.3	48.3
c	2553	738	276	62.7	42.1

Positive Integer Exponentiation (fixed decimal)

Module Name: IHEXIV

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 IHEMZV.

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 ≤ 15. This implies that n ≤ 16/(p + 1) ≤ 8 for all such cases and, in fact, this module will operate only for the range 0 < n ≤ 8.

Implementation:

- Module size: 192 bytes
- Execution time:

Let M = number of significant bits in the exponent

N = number of 1 bits in the exponent

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

-a + b*M + c*N

	30	40	50	65	75
a	9200	3100	1300	450	370
b	5000	1700	730	250	200
c	5500	1800	750	250	200

Integer Exponentiation (floating-point)

Module Names and Entry Points:

Argument	Module name	Entry point
Short float	IHEXIW	IHEXIW0
Long float	IHEXIZ	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 answer 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 routine. 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 ≤ 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 (IHEMZW or IHEMZZ)

Implementation:

* Module size: IHEXIW 256 bytes
  IHEXIZ 256 bytes

* Execution time:

  Let M = number of significant bits in the exponent

  N = number of 1 bits in the exponent

  Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

  -a + b*M + c*N for positive exponents

  -a'+ b*M + c*N for negative exponents

IHEXIW

	30	40	50	65	75
a	1446	582	128	35.2	17.6
b	2125	565	174	45.8	26.9
c	1782	484	147	36.1	21.1
a'	-142	-31	-12	4.6	-6.3

IHEXIZ

	30	40	50	65	75
a	8524	2042	275	60.0	30.8
b	5393	1397	289	59.9	34.9
c	4918	1245	226	50.2	29.1
a'	95	-374	-53	-5.1	-4.8

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 IHEEXS or IHEEXL

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

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

General Floating-Point Exponentiation

Module Names and Entry Points:

Argument	Module name	Entry point
Short float	IHEXXW	IHEXXW0
Long float	IHEXXZ	IHEXXZ0

Function:

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

Implementation:

- Module size: IHEXXW 280 bytes
  IHEXXZ 280 bytes

- Execution times:

  Approximate execution times in microseconds for System/360 models given below are obtained from the table:

  $a = \mathrm{IMAG}(z_1)$ $b = \mathrm{IMAG}(z_2)$

  $c = \mathrm{REAL}(z_1)$ $d = \mathrm{REAL}(z_2)$

	30	40	50	65	75
IHEXXW					
$a = 0$ $c > 0$	20606	5834	1816	480	291
$a = 0$ $d < 0$	21750	6171	1929	509	311
$a \neq 0$ $b = 0$	27448	8022	2414	687	417
$a \neq 0$ $b \neq 0$	28263	8229	2576	711	424
IHEXXZ					
$a = 0$ $c > 0$	62440	15325	3206	745	426
$a = 0$ $d < 0$	65208	16062	3366	781	450
$a \neq 0$ $b = 0$	90418	21611	4524	1056	604
$a \neq 0$ $b \neq 0$	92809	22197	4623	1077	616

FUNCTIONS WITH REAL ARGUMENTS

ADD (Fixed decimal)

Module Name: IHEADD

Entry Point: IHEADD0

Function:

$\mathrm{ADD}(x_1,x_2,p,q)$ 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 IHEAPD 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 IHEAPD 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 IHEAPD.

Implementation:

- Module size: 216 bytes

- Execution time:

  Let S_1 = time for IHEAPDA with argument equal to the sum of the two arguments to IHEADD, precision 31 and scale factor equal to the minimum of the scale factors of the two arguments

  S_2 = time for IHEAPDB with argument equal to the argument to IHEADD with the larger scale factor

  Then the approximate execution time in microseconds for the System/360 models given below are obtained from the following formulas:

  (i) Both arguments non-zero:

  $$t_1 + S_1 + S_2$$

  (ii) At least one argument zero:

  $$t_2 + S_1$$

	30	40	50	65	75
t_1	1895	648	320	85.0	65.6
t_2	1619	534	260	71.5	53.0

MAX, MIN

Module Names and Entry Points:

Argument	PL/I function	Module name	Entry point
Fixed binary	MAX MIN	IHEMXB	IHEMXBX IHEMXBN
Fixed decimal	MAX MIN	IHEMXD	IHEMXDX IHEMXDN
Short float	MAX MIN	IHEMXS	IHEMXSX IHEMXSN
Long float	MAX MIN	IHEMXL	IHEMXLX 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:

IHEMXB,IHEMXS,IHEMXL: 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.

IHEMXD: 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.

Implementation:

- Module sizes: IHEMXB 96 bytes
  IHEMXD 120 bytes
  IHEMXS 96 bytes
  IHEMXL 96 bytes

- Execution time:

IHEMXB

Let N = the number of source arguments

Then the average execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHEMXBX

$$a + b*N + c* \sum_{i=2}^{N} (1/i)$$

IHEMXBN

$$a' + b*N + c* \sum_{i=2}^{N} (1/i)$$

	30	40	50	65	75
a	391	135.7	51.8	12.1	7.4
b	186	70.7	30.0	11.0	8.06
c	54	21.3	8.0	2.5	0.9
a'	356	123.8	47.3	10.7	6.3

IHEMXD

Let N = the number of source arguments

(p,q) = the precision

L = FLOOR(p/2) + 1

Then the average execution times in microseconds for the System 360 models given below are obtained from the following formulas:

IHEMXDX

$$a + (b + d*L)*N + c* \sum_{i=2}^{N} (1/i)$$

IHEMXDN

$$a' + (b + d*L)*N + c* \sum_{i=2}^{N} (1/i)$$

	30	40	50	65	75
a	670	217	84.1	20.9	12.3
b	232	92.5	46.4	16.4	13.5
c	44	16.9	6.5	1.9	1.5
d	5	2.5	2.1	0.4	0.4
a'	635	205	79.6	19.6	11.3

IHEMXS,IHEMXL

Let N = the number of source arguments

Then the average execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHEMXSX,IHEMXLX

$$a + b*N + c* \sum_{i=2}^{N} (1/i)$$

IHEMXSN,IHEMXIN

$$a' + b*N + c* \sum_{i=2}^{N} (1/i)$$

IHEMXS

	30	40	50	65	75
a	367	131.2	44.6	11.5	7.3
b	219	73.3	29.1	10.5	7.7
c	55	21.3	7.3	2.5	1.9
a'	332	118.6	40.1	10.2	6.2

IHEMXL

	30	40	50	65	75
a	367	133.2	46.4	11.7	7.3
b	251	81.2	31.4	10.6	7.71
c	71	26.3	9.3	2.7	1.9
a'	332	120.6	41.9	10.4	6.2

FUNCTIONS WITH COMPLEX ARGUMENTS

ADD (Fixed decimal)

Module Name: IHEADV

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 (IHEADD). The imaginary parts are treated similarly.

Error and Exceptional Conditions:

H : FIXEDOVERFLOW or SIZE may occur in IHEAPD.

Implementation:

- Module size: 96 bytes
- Execution time:

Let T_1 = execution time for IHEADD with the real parts of the arguments for IHEADV as arguments

T_2 = execution time for IHEADD with the imaginary parts of the arguments for IHEADV as arguments

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$t + T_1 + T_2$$

	30	40	50	65	75
t	1094	396	156	42.6	26.1

MULTIPLY (fixed binary)

Module Name: IHEMPU

Entry Point: IHEMPU0

Function:

MULTIPLY(z_1,z_2,p,q) where z, 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

Implementation:

- Module size: 240 bytes
- Execution time:

Let q_1 = scale factor of the first argument

q_2 = scale factor of the second argument

Q = scale factor of the target

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the table:

$k = q_1 + q_2$

	30	40	50	65	75
Q = k	2816	847	329	75.7	49.3
Q > k	3098	910	354	81.5	54.9
Q < k	3394	1052	404	96.0	66.8

Implementation:

- Module size: 288 bytes
- Execution time:

Let $(p_1,q_1)(p_2,q_2)$ = the precisions of the arguments·

L_1 = FLOOR(p_1/2)

L_2 = FLOOR(p_2/2)

T = Time to shift-and-assign the result using IHEAPDA, with argument of precision $(p_1 + p_2 + 1, q_1 + q_2)$ and result (p,q)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$2*T + a + b*L_1 + c*L_2 + d*L_1*L_2$$

	30	40	50	65	75
a	3164	1170	540	168.1	131.7
b	358	89.7	46.4	25.2	17.3
c	142	18.4	18.3	8.8	7.9
d	112	15.0	0	4.0	4.7

MULTIPLY (fixed decimal)

Module Name: IHEMPV

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 IHEAPD.

Error and Exceptional Conditions:

H : FIXEDOVERFLOW or SIZE in IHEAPD.

DIVIDE (fixed binary)

Module Name: IHEDVU

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

Implementation:

- Module size: 408 bytes
- Execution time:

Let N_1 = number of significant bits in the expression a*c + b*d

N_2 = number of significant bits in the expression b*c - a*d

N_3 = number of significant bits in the expression c**2 + d**2

F_1 = FLOOR ((61 - N_1)/4) if $N_1 \le 61$
= 0 if $N_1 > 61$

F_2 = FLOOR ((61 - N_2)/4) if $N_2 \le 61$
= 0 if $N_2 > 61$

F_3 = FLOOR ((N_3 - 32)/4)

S_1 = (61 - N_1) - F_1*4 if $N_1 \le 61$
= 0 if $N_1 > 61$

S_2 = (61 - N_2) - F_2*4 if $N_2 \le 61$
= 0 if $N_2 > 61$

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

(i) $n_3 > 32$: $t_1 + t_2*(F_1 + F_2) + t_3*(S_1 + S_2) + t_4*F_3$

(ii) $n_3 = 32$: $t_5 + t_2*(F_1 + F_2) + t_3*(S_1 + S_2)$

(iii) $n_3 < 32$: $t_6 + t_2*(F_1 + F_2) + t_3*(S_1 + S_2) + t_7*N_3$

	30	40	50	65	75
t_1	5924	1835	609	144	104
t_2	261	73.5	26.0	7.6	5.1
t_3	251	73.5	25.5	7.8	5.0
t_4	213	55.6	21.5	6.0	3.8
t_5	5714	1780	582	138	100
t_6	3712	1288	473	115	86.5
t_7	-111	-38.8	-13.3	-3.4	-2.6

<u>DIVIDE (fixed decimal)</u>

Module Name: IHEDVV

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 IHEAPD (via entry point IHEAPDB). It then divides by c**2 + d**2 and calls IHEAPD (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 IHEAPD

Implementation:

- Module size: 576 bytes
- Execution time:

Let p_1, p_2 = the precisions of the arguments

L_1 = FLOOR(p_1/2)

L_2 = FLOOR(p_2/2)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formula:

$$a + b*L_1 + c*L_2 + d*L_1*L_2 + e*L_2**2$$

	30	40	50	65	75
a	28016	7646	3495	1146	975
b	368	94.7	48.7	25.8	17.9
c	382	67.5	60.4	23.2	19.9
d	112	15.0	0	4.0	4.7
e	56	7.5	0	2.0	2.3

ABS (fixed binary)

Module Name: IHEABU

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

Implementation:

- Module size: 184 bytes
- Execution times:

Approximate execution times in microseconds for System/360 models given below are obtained from the table:

a = 2**(15-q)

X1	30	40	50	65	75
< a	3809	1218	320	79.4	52.3
≥ a	4601	1473	372	93.1	59.8

ABS (fixed decimal)

Module Name: IHEABV

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

Implementation:

- Module size: 544 bytes
- Execution times:

Let (p,q) = the precision of the argument

L = CEIL((p+1)/2), i.e., the length in bytes of each of the real and imaginary parts of the argument

D_1 = maximum number of significant digits in real and imaginary parts of the argument

D_2 = number of significant digits in result

Then the approximate execution times in microseconds for the System/360 models shown below are obtained from the following formulas:

$L \le 5$ and $D_2 \le 7$: a

$L \le 5$, $7 < D_1 < 10$ and $D_2 > 7$: b+f*L+g*L**2

$5 < L \le 8$ and $D_2 \le 7$: c

$5 < L \le 8$, $7 < D_1 < 10$ and $D_2 > 7$: d+f*L+g*L**2

$5 < L \le 8$ and $10 \le D_1 \le 15$: e+f*L+g*L**2

	30	40	50	65	75
a	6220	1971	656	169	116
b	13001	3785	1101	460	352
c	6666	2200	737	190	132
d	13447	4014	1182	481	368
e	13918	4194	1279	509	391
f	82	61.6	40.1	7.9	5.2
g	56	7.5	0.0	2.0	2.3

ABS (floating-point)

Module Names and Entry Points:

Argument	Module name	Entry point
Short float	IHEABW	IHEABW0
Long float	IHEABZ	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 Z1 = MAX(ABS(x),ABS(y)) and Z2 = MIN(ABS(x),ABS(y)).

Then the answer is computed as

Z1*SQRT(1 + (Z2/Z1)**2).

Accuracy:

IHEABW

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

IHEABZ

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

Implementation:

- Module size: IHEABW 128 bytes
  IHEABZ 128 bytes

- Execution times:

  Approximate execution times in micro-seconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEABW	5595	1493	447	129	79.9
IHEABZ	14318	3191	695	174	104

The Library supports all float arithmetic generic functions and has separate modules for short and long precision real arguments and also 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 headings 'Range' and 'Error and Exceptional Conditions.'

Range: This states any ranges of arguments which a module assumes to have been excluded prior to its being 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 -- Interrupt 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 and UNDERFLOW 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 interrupt conditions occur in the modules called by the module concerned.

Speed

The average execution times given are based on the IBM System/360 Instruction Timing Information, Form A22-6825. These times include times for the modules called.

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 the former.

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). Assuming exact operands, a quotient is correct to the 56th binary digit of the mantissa. Therefore, truncation errors resulting from division are no more than 2**-52. The accuracy of a product, on the other hand, depends on the necessity for post-normalization. If the mantissas of both operands are close to 1, the truncation error of a product is about 2**-56. If the product of the mantissas is about 1/16, the truncation error is about 2**-52. On the other hand, if the mantissas of both operands are close to 1/16, the intermediate product has 7 leading zeros, and post-normalization introduces 4 trailing zeros. In this case, the truncation error can be close to 2**-48 (3.55*10**-15). In particular, multiplication by 1 in the long-precision form has the effect of erasing the last hexadecimal digit of the multiplicand.

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. There is no guard digit for long-precision additions and subtractions.

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.

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

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

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

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

The Library mathematical modules are summarized in Figures 4 and 5.

Function	Real Arguments	
	Short Float	Long Float
SQRT	IHESQS	IHESQL
EXP	IHEEXS	IHEEXL
LOG,LOG2,LOG10	IHELNS	IHELNL
SIN, COS,SIND,COSD	IHESNS	IHESNL
TAN, TAND	IHETNS	IHETNL
ATAN, ATAND	IHEATS	IHEATL
SINH, COSH	IHESHS	IHESHL
TANH	IHETHS	IHETHL
ATANH	IHEHTS	IHEHTL
ERF, ERFC	IHEEFS	IHEEFL

Figure 4. Mathematical Functions with Real Arguments

Function	Complex Arguments	
	Short Float	Long Float
SQRT	IHESQW	IHESQZ
EXP	IHEEXW	IHEEXZ
LOG	IHELNW	IHELNZ
SIN,COS,SINH,COSH	IHESNW	IHESNZ
TAN, TANH	IHETNW	IHETNZ
ATAN, ATANH	IHEATW	IHEATZ

Figure 5. Mathematical Functions with Complex Arguments

FUNCTIONS WITH REAL ARGUMENTS

SQRT (short floating-point real)

Module Name: IHESQS

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 $1/16 \le f < 1$. Then

SQRT(x) = 16**(p + q)*z,
where z = SQRT(f) if q = 0,
z = SQRT(f)/4 if q = 1.

An initial approximation, y_0, is taken in the hyperbolic form a + b/(c + f) with different sets of constants for the two cases:

1. q = 0 a = 1.80713
   b = -1.57727
   c = 0.954182

The maximum relative error in this range is then less than 2**(-5.44), with an exact fit at f = 1 to guard as far as possible against loss of the last hexadecimal digit when f is nearly 1.

2. q = 1 a = 0.428795
   b = -0.3430368
   c = 0.877552

The maximum relative error in this range is less than 2**(-6)*f**(-1/8).
Then y_1 = 16**(p + q)*y_0.

Two Newton-Raphson iterations then yield:

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

For case q = 0, the final relative error from this algorithm is less than 2**(-24.7), and, for case q = 1, less than 2**(-29).

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	0.230	0.924

Error and Exceptional Conditions:

P : x < 0

Implementation:

- Module size: 168 bytes
- Execution time:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHESQS	3140	793	227	68.4	40.7

SQRT (long floating-point real)

Module Name: IHESQL

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*2**(-2*q)*SQRT(f).

An initial approximation, y_0, is taken by using (2/9 + 8/9*f) for SQRT(f). Multiplication by 2**(-2) when q = 1 is accomplished by using the HALVE instruction twice. The maximum relative error of this approximation is 1/9.

Four Newton-Raphson iterations of the form $y_{n+1} = (y_n + x/y_n)/2$ are then applied, two in short precision and two in long precision, the last being computed as

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

to minimize the truncation error.

The maximum relative error in the result from this algorithm is 2**(-65.7).

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
x > 10** (-52)	Exponential	0.0276	0.124

Error and Exceptional Conditions:

P : x < 0

Implementation:

- Module size: 160 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHESQL	8282	1733	376	97.7	57.2

EXP (short floating-point real)

Module Name: IHEEXS

Entry Point: IHEEXS0

Function: To calculate e to the power x.

Method:

If x < -180.2, a zero result is returned immediately.

Otherwise, EXP(x) is calculated as:

2**(x*LOG2(e))

The calculation is performed as follows:

x*LOG2(e) = f + N,

where N = 4h + g, h is an integer such that g = 0, 1, 2 or 3, and 0 ≤ f < 1.0.

Then, by subtracting 0.5, this is reduced to the range -0.5 ≤ f < 0.5. Next, 2**f is calculated as:

(a + b/(c + x*x)+x)/(a + b/(c + x*x)-x).

This is multiplied by 2**0.5 and then shifted in the appropriate direction to give the effect of multiplication by 2**g. Finally, the exponent of the result is obtained from h.

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	0.132	0.490
Full Range	Uniform	1.29	2.61

Error and Exceptional Conditions:

I : OVERFLOW if x > 174.673

Implementation:

- Module size: 232 bytes
- Execution times:

  Approximate execution times in micro-seconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEEXS	3847	1172	356	90.0	58.0

EXP (long floating-point real)

Module Name: IHEEXL

Entry Point: IHEEXL0

Function: To calculate e to the power x.

Method:

If x < -180.2183, return zero result.

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

where a, b and c are integers, 0 ≤ b ≤ 3, 0 ≤ c ≤ 15 and 0 ≤ d < 1/16.

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

Compute 2**(-d) by using the Chebyshev interpolation polynomial of degree 6 over the range 0 ≤ d < 1/16, with maximum relative error 2**(-57).

If c > 0, multiply 2**(-d) by 2**(-c/16). The constants 2**(-c/16), 1 ≤ c ≤ 15, are included in the subroutine.

If b > 0, halve the result b times.

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

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	0.0674	0.216
Full range	Uniform	0.867	2.30

Error and Exceptional Conditions:

I : OVERFLOW if x > 174.673

Implementation:

- Module size: 448 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEEXL	12131	2901	616	343	194

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

Module Name: IHELNS

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 = m*16**p where 1/16 ≤ m < 1 and p is an integer.

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

```
1/16 ≤ m < 1/8,   a = 1/16,  b = 4;
1/8  ≤ m < 1/2,   a = 1/4,   b = 2;
1/2  ≤ m < 1,     a = 1,     b = 0.
```

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

Then m = a*(1+y)/(1-y) and ABS(y) ≤ 1/3.

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

Therefore

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

LOG((1+y)/(1-y)) is computed by using the Chebyshev interpolation polynomial of degree 4 in y**2 for the range 0 ≤ y**2 ≤ 1/9, with maximum relative error 2**-27.8. LOG2(x) or LOG10(x) are calculated by multiplying LOG(x) 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:

Range	Distribution	RMS (Relative Error *10**6)	Maximum (Relative Error *10**6)
IHELNSE			
Excluding 0.5 < x < 2.0	Exponential	0.0320	0.577
IHELNS2			
Excluding 0.5 < x < 2.0	Exponential	0.342	0.754
IHELNSD			
Excluding 0.5 < x < 2.0	Exponential	0.170	0.963

Arguments		Absolute Error *10**6	
Range	Distribution	RMS	Maximum
IHELNSE			
0.5 < x < 2.0	Uniform	0.0960	0.394
IHELNS2			
0.5 < x < 2.0	Uniform	0.177	0.842
IHELNSD			
0.5 < x < 2.0	Uniform	0.0526	0.164

Error and Exceptional Conditions:

P : x ≤ 0

Implementation:

- Module size: 256 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

Entry Point	30	40	50	65	75
IHELNSE	4669	1238	385	173	95.7
IHELNS2	5041	1342	417	180	101.3
IHELNSD	5054	1366	417	180	101.3

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

Module Name: IHELNL

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 is the exponent, q is an integer such that 0 ≤ q ≤ 3, and 1/2 ≤ m < 1.

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

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

LOG((1 + y)/(1 - y)) is computed by using the Chebyshev interpolation polynomial of degree 7 in y**2 for the range 0 ≤ y**2 ≤ 0.02944, with maximum relative error 2**(-59.6).

LOG2(x) or LOG10(x) is calculated by multiplying the 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	0.0530	0.329
IHELNL2			
Excluding 0.5 < x < 2.0	Exponential	0.443	2.60
IHELNLD			
Excluding 0.5 < x < 2.0	Exponential	0.155	0.402

Arguments		Absolute Error *10**15	
Range	Distribution	RMS	Maximum
IHELNLE			
0.5 < x < 2.0	Uniform	0.192	0.507
IHELNL2			
0.5 < x < 2.0	Uniform	0.245	0.466
IHELNLD			
0.5 < x < 2.0	Uniform	0.0318	0.0625

Error and Exceptional Conditions:

P : $x \leq 0$

Implementation:

- Module size: 360 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

Entry Point	30	40	50	65	75
IHELNLE	16216	3926	788	178	98.5
IHELNL2	17315	4196	834	190	107
IHELNLD	17284	4192	828	188	105

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

Module Name: IHESNS

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 $0 \leq r < 1$.

Define q_1 = q if SIN or SIND is required and $x \geq 0$;
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 $t \geq 0$.

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 $0 \le r_1 \le 1$.

This is performed by using the Chebyshev interpolation polynomials of degree 3 in r_1**2, with maximum relative error of 2**-28.1 in the sine polynomial and 2**-24.6 in the cosine polynomial.

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	0.0557	0.126
pi/2 < ABS(x) ≤ 10	Uniform	0.0553	0.148
10 < ABS(x) ≤ 100	Uniform	0.0560	0.143
IHESNSC			
0 ≤ x ≤ pi	Uniform	0.0553	0.149
-10 ≤ x < 0, pi < x ≤ 10	Uniform	0.0571	0.154
10 < ABS(x) ≤ 100	Uniform	0.0553	0.142

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHESNSS			
ABS(x) ≤ pi/2	Uniform	0.198	1.40

Error and Exceptional Conditions:

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

IHESNSZ, IHESNSK:
ABS(x) ≥ 2**18*180

Implementation:

- Module size: 320 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHESNSS					
< pi/4	4091	1120	333	85.0	50.6
≥ pi/4	4386	1190	362	92.5	53.8
IHESNSC					
< pi/4	4078	1115	329	83.6	49.9
≥ pi/4	4373	1184	357	91.0	53.1
IHESNSZ					
< 45	4026	1132	338	86.3	51.6
≥ 45	4421	1202	366	93.7	54.8
IHESNSK					
< 45	3693	1127	334	84.8	51.0
≥ 45	4408	1196	362	92.3	54.1

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

Module Name: IHESNL

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, $0 \le r < 1$.

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 $q_2 = \mathrm{MOD}(q_1,8)$.

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 $0 \le q_2 \le 7$.

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 $0 \le r_1$**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 $0 \le r_1$**2 ≤ 1, with maximum relative error 2**(-64.3).

Finally, if $q_2 \ge 4$ 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
-pi/2 < x < pi/2	Uniform	0.0542	0.381

IHESNLC

Arguments		Absolute Error *10**15	
Range	Distribution	RMS	Maximum
-pi/2 < x < pi/2	Uniform	0.0604	0.168

Error and Exceptional Conditions:

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

IHESNLZ, IHESNLK:
ABS(x) ≥ 2**50*180

Implementation:

- Module size: 416 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

Entry Point	30	40	50	65	75
IHESNLS	13654	3290	661	155	85.3
IHESNLC	13641	3288	654	153	84.3
IHESNLZ	13689	3302	665	157	86.3
IHESNLK	13676	3300	659	155	85.3

TAN, TAND (short floating-point real)

Module Name: IHETNS

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

where P(s**2) = 212.58037 - 12.559912 *s**2
Q(s**2) = 270.665736 - 71.645273 *s**2 + s**4.

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	0.319	1.92
pi/4 < ABS(x) < 1.5	Uniform	0.465	1.24
pi/4 < ABS(x) < pi/2	Uniform	3.14	170*
pi/2 < ABS(x) ≤ 10	Uniform	1.25	70.6*
10 < ABS(x) ≤ 100	Uniform	3.57	205*

*These maximum errors 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.

Error and Exceptional Conditions:

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

I : IHETNSR: OVERFLOW
IHETNSD: OVERFLOW

Implementation:

- Module size: 280 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHETNSR					
< pi/4	4429	1172	336	85.8	51.0
≥ pi/4	4788	1262	368	95.1	55.0
IHETNSD					
< 45	4464	1184	341	87.0	52.1
≥ 45	4823	1274	373	96.3	56.0

TAN, TAND (long floating-point real)

Module Name: IHETNL

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 P(s**2) is of degree 3 and Q(s**2) is of degree 4 in s**2, and maximum relative error is 3.4*10**-19.

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	0.091	0.530
pi/4 < ABS(x) < 1.5	Uniform	0.437	2.31
pi/4 < ABS(x) < pi/2	Uniform	7.75	416*
pi/2 < ABS(x) ≤ 10	Uniform	18.3	1140*
10 < ABS(x) ≤ 100	Uniform	271	13400*

*These maximum errors 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.

Error and Exceptional Conditions:

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

I : IHETNLR: OVERFLOW
IHETNLD: OVERFLOW

Implementation:

- Module size: 352
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHETNLR					
< pi/4	15440	3622	687	154	87.2
≥ pi/4	16130	3817	747	169	93.9
IHETNLD					
< 45	15475	3634	691	155	88.2
≥ 45	16165	3829	751	170	94.9

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

Module Name: IHEATS

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		Relative Error *10**6	
Range	Distribution	RMS	Maximum
IHEATS1			
Full range	Tangents of numbers uniformly distributed between -pi/2 and pi/2	0.443	0.958
IHEATS2			
Full range	y = sin and x = cos of numbers uniformly distributed between -pi/2 and pi/2	0.449	1.42

Error and Exceptional Conditions:

P : IHEATS2, IHEATS4: x = y = 0

Implementation:

- Module size: 408 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the tables:

  k = TAND(15)

ABS(x)	30	40	50	65	75
IHEATS1					
≤ k	3162	858	279	78.4	48.4
k < ABS(x) < 1	4345	1136	347	97.8	58.0
1 ≤ ABS(x) < 1/k	5051	1301	381	108	64.1
≥ 1/k	3868	1023	313	88.7	54.5
IHEATS2					
≤ k	4193	1138	363	106	67.3
k < ABS(x) < 1	5376	1416	431	126	77.0
1 ≤ ABS(x) < 1/k	6082	1581	465	136	83.1
≥ 1/k	4899	1303	496	117	73.4
IHEATS3					
≤ k	3521	948	305	83.9	51.3
k < ABS(x) < 1	4704	1226	374	103	61.0
1 ≤ ABS(x) < 1/k	5410	1391	408	114	67.1
≥ 1/k	4227	1114	339	94.2	57.4
IHEATS4					
≤ k	4552	1228	389	112	70.2
k < ABS(x) < 1	5735	1506	458	131	69.9
1 ≤ ABS(x) < 1/k	6441	1671	492	142	86.0
≥ 1/k	5358	1393	323	122	76.3

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

Module Name: IHEATL

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 + a_1*x*x/(b_1 + x*x + a_2/(b_2 + x*x + a_3/(b_3 + x*x + a_4/(b_4 + x*x))))

with relative error less than 2**(-57.9).

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:

IHEATL1

Arguments		Relative Error *10**15	
Range	Distribution	RMS	Maximum
-1 < x < 1	Uniform	0.0438	0.207

Error and Exceptional Conditions:

P : IHEATL2, IHEATL4: x = y = 0

Implementation:

- Module size: 544 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

Entry Point	30	40	50	65	75
IHEATL1	20472	4389	826	181	100
IHEATL2	23523	5042	967	217	123
IHEATL3	21575	4667	871	190	105
IHEATL4	24602	5305	1008	226	128

SINH, COSH (short floating-point real)

Module Name: IHESHS

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:

For IHESHSS, if ABS(x) ≤ 1, use a polynomial approximation of the seventh degree.

Otherwise,

SINH(x) = EXP(x)/2 - 0.5/EXP(x),
COSH(x) = EXP(x)/2 + 0.5/EXP(x).

These two versions of EXP(x)/2 ± 0.5/EXP(x) are preferable to the equivalent versions of (EXP(x) - 1/EXP(x))/2 because, in floating-point, 0.5 has three more significant bits than 1.0.

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	0.200	0.932
1 < ABS(x) < 2	Uniform	0.221	0.950
IHESHSC			
ABS(x) ≤ 1	Uniform	0.367	0.908
1 < ABS(x) < 2	Uniform	0.192	0.700

Error and Exceptional Conditions:

H : OVERFLOW in real EXP routine (IHEEXS).

Implementation:

- Module size: 216 bytes
- Execution times:

  Approximate execution times in micro-seconds for the System/360 models given below are obtained from the appropriate entry point in the table:

Entry Point	30	40	50	65	75
IHESHSS					
ABS(x)≤1	2544	692	228	56.3	33.0
ABS(x)>1	5647	1693	526	144	91.8
IHESHSC	5500	1648	509	139	88.4

SINH, COSH (long floating-point real)

Module Name: IHESHL

Entry Points:

Mathematical function	PL/I name	Entry point
Hyperbolic sin x	SINH(x)	IHESHLS
Hyperbolic cos x	COSH(x)	IHESHLC

Function:

To calculate hyperbolic sin x or hyperbolic cos x.

Method:

For IHESHSS, if ABS(x) < 0.3465736, compute SINH(x)/x using a polynomial approximation of degree 5 in x**2, with relative error less than 2**-61.9

Otherwise, compute s = EXP(ABS(x)); then

COSH(x) = (s + 1/s)/2
SINH(x) = SIGN(x)*(s - 1/s)/2

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 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
IHESHLS			
ABS(x) < 0.34657	Uniform	0.0530	0.217
0.34567 < ABS(x) ≤ 5	Uniform	0.0870	0.359
IHESHLC			
ABS(x) ≤ 5	Uniform	0.123	0.429

Error and Exceptional Conditions:

H : OVERFLOW in real EXP routine (IHEEXL).

Implementation:

- Module size: 264 bytes
- Execution times:

  Approximate execution times in microseconds for System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHESHLS					
< 0.347	9024	2279	450	101	59.0
0.347 ≤ ABS(x) ≤ 174.6	18634	4338	938	215	125
IHESHLC					
≤ 174.6	18493	4300	924	211	123

<u>TANH (short floating-point real)</u>

Module Name: IHETHS

Entry Point: IHETHS0

Function: To calculate hyperbolic tan x.

Method :

1. ABS(x) ≤ 2**-12

   Return x as result.

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

   Use a transformed continued fraction of the form:

   TANH(x)/x = 1-((x**2 + a)/(x**2 + b + c/x**2))

   with relative error less than 2**-27.

3. 0.54931 ≤ x < 9.011

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

4. x ≥ 9.011

   Return result 1.

5. x ≤ -0.54931

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

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
-0.5 < x < 0.5	Uniform	0.174	0.867
-9 < x < 9	Uniform	0.0720	0.782

Implementation:

- Module size: 200 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

	30	40	50	65	75
ABS(x) ≤ 2**-12	791	263	102	28.7	21.7
2**-12 < ABS(x)<0.5	3033	785	231	64.1	43.9
0.5 ≤ x < 9	5934	1805	562	152	117
x ≥ 9	1095	363	139	40.5	35.2

TANH (long floating-point real)

Module Name: IHETHL

Entry Point: IHETHL0

Function: To calculate hyperbolic tan x.

Method:

1. ABS(x) < 0.54931

   Compute TANH(x)/x using a rational approximation, with relative error less than 2**-64.5

2. 0.54931 ≤ X < 20.101

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

3. x ≥ 20.101

Return result 1.

4. x ≤ -0.54931

   TANH(x) = -TANH(-x)

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	0.0440	0.211
0.54931 < ABS(x) ≤ 5	Uniform	0.0250	0.199

Implementation:

- Module size: 280 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

ABS(x)	30	40	50	65	75
< 0.549	12745	3030	564	123	67.9
0.549 ≤ ABS(x) < 20.1	16400	3918	878	205	119
≥ 20.1	1239	372	135	39.3	25.5

ATANH (short floating-point real)

Module Name: IHEHTS

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
-0.8 ≤ x ≤ 0.8	Uniform	0.440	1.32
-0.9 < x < 0.9	Uniform	0.389	1.14

Error and Exceptional Conditions:

P : ABS(x) ≥ 1

Implementation:

- Module size: 192 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

ABS(x)	30	40	50	65	75
≤ 0.2	2520	667	208	52.0	31.3
0.2 < ABS(x) < 1	7091	1829	606	163	94.8

ATANH (long floating-point real)

Module Name: IHEHTL

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	0.0650	0.223
ABS(x) ≤ 0.95	Uniform	0.133	0.397

Error and Exceptional Conditions:

P : ABS(x) ≥ 1

Implementation:

- Module size: 272 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

ABS(x)	30	40	50	65	75
≤ 0.25	12252	3037	562	121	68
0.25 < ABS(x) < 1	20448	4900	1040	242	137

ERF, ERFC (short floating-point real)

Module Name: IHEEFS

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 ≤ x ≤ 1.317

Compute ERF(x)/x by using a Chebyshev interpolation polynomial of degree 6 in x**2, with relative error less than 2**-24.

ERFC(x) = 1 - ERF(x)
(ERFC(x) > 1/16 in this range).

2. 1.317 < x ≤ k, where k = 2.04000092

Compute ERFC(x) by using a Chebyshev interpolation polynomial of degree 7 in (x-k), with absolute error less than 1.3 * 2**-30.

ERF(x) = 1 - ERFC(x).
(ERFC(x) > 1/256 in this range).

3. k < x < 13.306

ERFC(x)*x*EXP(x**2) is computed by using a Chebyshev interpolation polynomial of degree 6 in x**-2, with relative error less than 1.2 * 2**-23.

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

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) ≤ 1.3	Uniform	0.139	0.934
1.3 < ABS(x) ≤ 2	Uniform	0.0372	0.263
2 < ABS(x) ≤ 3.9	Uniform	0.0347	0.0605

IHEEFSC

$-3.8 < x < 0$	Uniform	0.297	0.930
$0 < x \le 1.3$	Uniform	0.505	3.80
$1.3 < x \le 2$	Uniform	0.314	4.08
$2 < x \le 3.9$	Uniform	0.367	1.45
$3.9 < x \le 13.3$	Uniform	9.09	17.1

Implementation:

- Module size: 376 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHEEFSF					
≤ 1.32	4354	1191	392	100	57.2
$1.32 <$ ABS(x) ≤ 2.04	4613	1266	418	110	62.0
$2.04 <$ ABS(x) < 3.92	10013	2843	868	228	140
≥ 3.92	1530	473	183	50.7	32.0
IHEEFSC					
≤ 1.317	4412	1204	385	103	58.8
$1.32 <$ ABS(x) ≤ 2.04	4582	1262	405	110	63.4
$2.04 <$ ABS(x) < 3.92	9982	2839	854	228	142
$3.92 \le$ ABS(x) < 13.31	10168	2897	879	236	147
≥ 13.31	1598	499	182	54.5	35.5

ERF, ERFC (long floating-point real)

Module Name: IHEEFL

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. $0 \le x \le 1$

Compute ERF(x)/x by using a Chebyshev interpolation polynomial of degree 11 in x**2, with relative error less than 1.07*2**-57.

ERFC(x) = 1-ERF(x).
(ERFC(x) > 1/16 in this range).

2. $1 < x \le 2.04000092$

Compute ERFC(x) by using a Chebyshev interpolation polynomial of degree 18 in (x - 1.999999), with absolute error less than 1.5*2**-61.

ERF(x) = 1-ERFC(x).
(ERFC(x) > 1/256 in this range).

3. $2.04000092 < x < 13.306$

ERFC(x) is computed by using a Chebyshev interpolation polynomial of degree 20 in x**-2 for ERFC(x)*x*EXP(x**2), with relative error ranging from 2**-53 at 2.04000092 to 2**-51 at 13.306.

If x < 6.092, ERF(x) = 1-ERFC(x).
If x ≥ 6.092, ERF(x) = 1

4. $x \ge 13.306$

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

5. $x < 0$

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) ≤ 1.317	Uniform	0.0280	0.202
1.317 < ABS(x) ≤ 2.04	Uniform	0.0107	0.0291
2.04 < ABS(x) < 6.092	Uniform	0.00803	0.0170
IHEEFLC			
-6 < x < 0	Uniform	0.0684	0.188
0 ≤ x ≤ 1.317	Uniform	0.0762	0.352
1.317 < x ≤ 2.04	Uniform	0.127	0.445
2.04 < x < 4	Uniform	1.24	4.02
4 ≤ x < 13.3	Uniform	1.40	5.02

Implementation:

- Module size: 744 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(x)	30	40	50	65	75
IHEEFLF					
≤ 1.00	16567	4154	805	180	103
1.00 < ABS(x) ≤ 2.04	24413	6095	1175	263	147
2.04 < ABS(x) < 6.09	45574	1105	2142	477	269
≥ 6.09	2493	707	222	58.0	36.3
IHEEFLC					
≤ 1.00	16905	4240	837	188	106
1.00 < ABS(x) ≤ 2.04	24263	6065	1166	261	147
2.04 < ABS(x) < 6.09	45424	10974	2133	474	270
6.09 ≤ ABS(x) < 13.3	45624	11040	2160	482	274
≥ 13.3	2408	693	219	57.4	37.5

FUNCTIONS WITH COMPLEX ARGUMENTS

SQRT (short floating-point complex)

Module Name: IHESQW

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:

Let z = x + yI, and
SQRT(z) = u + vI.

1. x = y = 0

Then u = v = 0.

2. x ≥ 0

Then u = SQRT((ABS(x) + ABS(x + yI))/2)
and v = y/(2*u).

3. x < 0

Then u = y/(2*v)
and v = S(y)*SQRT((ABS(x) + ABS(x + yI))/2)

where S(y) = 1 if y ≥ 0
= -1 if y < 0

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 radially, uniform round origin	0.513	1.51

Error and Exceptional Conditions:

I : OVERFLOW

H : OVERFLOW in complex ABS routine (IHEABW)

Implementation:

- Module size: 152 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHESQW	11130	3023	1006	265	164

SQRT (long floating-point complex)

Module Name: IHESQZ

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:

Let z = x + yI, and
SQRT(z) = u + vI.

1. x = y = 0

Then u = v = 0.

2. x ≥ 0

Then u = SQRT((ABS(x) + ABS(x + yI))/2)
and v = y/(2*u).

3. x < 0

Then u = y/(2*v)
and v = S(y)*SQRT((ABS(x) + ABS(x + yI))/2)

where S(y) = 1 if y ≥ 0
= -1 if y < 0

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 radially, uniform round origin	0.263	1.54

Error and Exceptional Conditions:

I : OVERFLOW

H : OVERFLOW in complex ABS routine (IHEABZ)

Implementation

- Module size: 144 bytes
- Execution times:

  Approximate execution times in microseconds for System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHESQZ	26996	5957	1352	341	203

EXP (short floating-point complex)

Module Name: IHEEXW

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	1.32	3.14
ABS(x) ≤ 170 pi/2 < ABS(y) ≤ 20	Uniform	1.31	3.34

Error and Exceptional Conditions:

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

H : OVERFLOW in real EXP routine (IHEEXS)

Implementation:

- Module size: 136 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEEXW	14240	4058	1244	326	200

EXP (long floating-point complex)

Module Name: IHEEXZ

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	0.136	0.478
ABS(x) < 20 ABS(y) < 20	Uniform	1.28	2.29

Error and Exceptional Conditions:

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

H : OVERFLOW in real EXP routine (IHEEXL)

Implementation:

- Module size: 136 bytes
- Execution times:

  Approximate execution times in microseconds for System/360 models given below are obtained from the table:

Module Name	30	40	50	65	75
IHEEXZ	42838	10560	2174	505	287

LOG (short floating-point complex)

Module Name: IHELNW

Entry Point: IHELNW0

Function:

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

Method:

Let LOG(x + yI) = u + vI.

```
Then u = LOG(ABS(x + yI))
       = LOG(SQRT(x**2 + y**2))
       = LOG(x**2 + y**2)/2
and  v = ATAN(y,x).
```

In computing u, the exponents of x and y are modified if necessary to avoid OVERFLOW or UNDERFLOW, with the appropriate correction being applied after the logarithm has been taken.

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 IHEATS.

Accuracy:

Arguments		Relative Error *10**6	
Range	Distribution	RMS	Maximum
Full range except within 10**-6 of 1+0I	Exponential radially, uniform round origin	0.150	2.27

Error and Exceptional Conditions:

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

Implementation:

- Module size: 272 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the table:

	30	40	50	65	75
(i)	11511	3414	1078	308	183
(ii)	11688	3489	1104	318	190
(iii)	11800	3520	1117	321	193

(i) ABS(x) and ABS(y) < SQRT(8)*16**31 and either ABS(x) or ABS(y) ≥ 16**-30

(ii) Either ABS(x) or ABS(y) ≥ SQRT(8)*16**31

(iii) ABS(x) and ABS(y) < 16**-30

LOG (long floating-point complex)

Module Name: IHELNZ

Entry Point: IHELNZ0

Function:

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

Method:

Let LOG(x + yI) = u + vI.

```
Then u = LOG(ABS(x + yI))
       = LOG(SQRT(x**2 + y**2))
       = LOG(x**2 + y**2)/2
and  v = ATAN(y,x).
```

In computing u, the exponents of x and y are modified if necessary to avoid OVERFLOW or UNDERFLOW, with the appropriate correction being applied after the logarithm has been taken.

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 IHEATL.

Accuracy:

Arguments		Relative Error *10**15	
Range	Distribution	R.M.S.	Maximum
Full range except within 10**-6 of 1+0I	Exponential radially, uniform round origin	0.0558	1.46

Error and Exceptional Conditions:

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

Implementation:

- Module size: 288 bytes
- Execution times:

  Approximate execution times in microseconds for System/360 models given below are obtained from the table:

	30	40	50	65	75
(i)	44101	10166	2086	480	274
(ii)	44398	10316	2155	507	290
(iii)	44438	10325	2156	507	290

(i) ABS(x) and ABS(y) < SQRT(8)*16**31 and either ABS(x) or ABS(y) ≥ 16**-26

(ii) Either ABS(x) or ABS(y) ≥ SQRT(8)*16**31

(iii) ABS(x) and ABS(y) < 16**-26

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

Module Name: IHESNW

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 + a_1*u**2 + a_2*u**4 (which has a relative error of less than 2**-26.4).

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**6: RMS	Relative Error *10**6: Maximum
IHESNWS			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.721	2.07
IHESNWZ			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.561	1.86
IHESNWC			
ABS(x) ≤ 10, ABS(y) ≤ 20	Uniform	0.546	2.00
IHESNWK			
ABS(x) ≤ 10, ABS(y) ≤ 20	Uniform	0.558	2.35

Error and Exceptional Conditions:

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

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

H : OVERFLOW in real EXP routine (IHEEXS)

Implementation:

- Module size: 320 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the tables:

SIN, COS : ABS(y) > 0.3465736
SINH,COSH: ABS(x) > 0.3465736

Entry Point	30	40	50	65	75
IHESNWS	15826	1508	648	363	223
IHESNWC	15898	1518	653	366	225
IHESNWZ	15930	1520	653	366	226
IHESNWK	15900	1519	655	367	227

SIN, COS : ABS(y) ≤ 0.3465736
SINH,COSH: ABS(x) ≤ o.3465736

IHESNWS	16896	1585	674	381	232
IHESNWC	16968	1595	679	384	234
IHESNWZ	17000	1596	679	384	234
IHESNWK	16970	1595	681	384	235

<u>SIN, SINH, COS, COSH (long floating-point complex)</u>

Module Name: IHESNZ

Entry Points:

Mathematical <u>function</u>	PL/I <u>name</u>	Entry <u>point</u>
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 > 0.3465736

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

2. 0 ≤ u ≤ 0.3465736

SINH(u)/u is approximated by a polynomial of the fifth degree in u**2 which has a relative error of less than 2**-61.8

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**15	
Range	Distribution	RMS	Maximum
IHESNZS			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	2.11	82.4
IHESNZZ			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.180	2.31
IHESNZC			
ABS(x) ≤ 10, ABS(y) ≤ 1	Uniform	0.389	6.33
IHESNZK			
ABS(x) ≤ 10, ABS(y) ≤ 20	Uniform	1.11	19.4

Error and Exceptional Conditions:

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

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

H : OVERFLOW in real EXP routine (IHEEXL)

Implementation:

- Module size: 368 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the tables:

SIN, COS : ABS(y) > 0.3465736
SINH,COSH: ABS(x) > 0.3465736

Entry Point	30	40	50	65	75
IHESNZS	46584	11267	2363	552	313
IHESNZC	46656	11294	2373	555	315
IHESNZZ	46726	11317	2378	557	316
IHESNZK	46640	11304	2374	556	316

SIN, COS : ABS(Y) ≤ 0.3465736
SINH,COSH: ABS(X) ≤ 0.3465736

IHESNZS	54173	13141	2656	612	345
IHESNZC	54245	13168	2666	615	347
IHESNZZ	54325	13191	2671	617	347
IHESNZK	54247	13177	2667	616	348

TAN, TANH (short floating-point complex)

Module Name: IHETNW

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	0.430	1.67
IHETNWH			
ABS(x) < 9 ABS(y) < 1	Uniform	0.430	1.45

Error and Exceptional Conditions:

I : OVERFLOW

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

H : OVERFLOW in real TAN routine (IHETNS)

Implementation:

- Module size: 184 bytes
- Execution times:

Approximate execution times in microseconds for the System/360 models given below are obtained from the formulas:

IHETNWN: a + time for execution of IHETHS with argument y

IHETNWH: b + time for execution of IHETHS with argument x

	30	40	50	65	75
a	9094	2310	696	186	111
b	9197	2454	716	191	115

TAN, TANH (long floating-point complex)

Module Name: IHETNZ

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	0.139	1.11
IHETNZH			
ABS(x) < 9 ABS(y) < 1	Uniform	0.137	0.980

Error and EXCEPTIONAL Conditions:

I : OVERFLOW

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

H : OVERFLOW in real TAN routine (IHETNL)

Implementation:

- Module size: 184 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table:

ABS(y)	30	40	50	65	75
IHETNZN					
< 0.549	40843	9625	1844	411	233
0.549 ≤ ABS(y) < 20.1	44498	10513	10365	493	284
≥ 20.1	29337	6967	1415	327	190

ABS(x)	30	40	50	65	75
IHETNZH					
< 0.549	41122	9709	1871	419	236
0.549 ≤ ABS(x) < 20.1	44777	10597	2185	501	287
≥ 20.1	29616	7051	1442	334	193

ATAN, ATANH (short floating-point complex)

Module Name: IHEATW

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	0.216	2.88
IHEATWH			
Full range	Exponential	0.208	1.18

Error and Exceptional Conditions:

P : IHEATWN: z + ±1I
IHEATWH: z = ±1

Implementation:

- Module size: 304 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table, where

  a = ABS(2*u/(1+x*x+y*y))
  u = y for IHEATWN
  = x for IHEATWH

	30	40	50	65	75
IHEATWN					
a ≤ 0.2	12235	3306	1033	279	173
0.2 < a < 1	16056	4454	1408	462	276
IHEATWH					
a ≤ 0.2	12100	3267	1017	275	171
0.2 < a < 1	15921	4415	1392	458	274

ATAN, ATANH (long floating-point complex)

Module Name: IHEATZ

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	0.141	7.93
IHEATZH			
Full range	Exponential	0.0826	1.20

Error and Exceptional Conditions:

P : IHEATZN: z = ±1I
IHEATZH: z = ±1

Implementation:

- Module size: 296 bytes
- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate entry point in the table, where

  a = ABS(2*u/(1+x*x+y*y))
  u = y for IHEATZN
  = x for IHEATZH

	30	40	50	65	75
IHEATZN					
a≤0.25	43477	9977	2006	455	260
0.25<a < 1	51673	11840	2406	576	329
IHEATZH					
a≤0.25	43293	9923	1987	450	258
0.25<a < 1	51489	11786	2466	571	327

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 which a module assumes to have been excluded prior to its being 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 -- Interrupt 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 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 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.

Speed

The average execution times given are based on IBM System/360 Instruction Timing Information, Form A22-6825.

A summary of the Library array modules is given in Figures 6 and 7.

	Simple arrays, and interleaved arrays of variable length strings	Interleaved string arrays with fixed-length elements
Indexers ALL, ANY	IHEJXS IHENL1	IHEJXI IHENL2

Note: IHEJXI is used for indexing through interleaved arithmetic arrays

Figure 6. 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	IHESSF	IHESMF	IHESSG	IHESMG	IHESSH	IHESMH
	complex	IHESSX	IHESMX	IHESSG	IHESMG	IHESSH	IHESMH
PROD	real	IHEPSF	IHEPDF	IHEPSS	IHEPDS	IHEPSL	IHEPDL
	complex	IHEPSX	IHEPDX	IHEPSW	IHEPDW	IHEPSZ	IHEPDZ
POLY	real	IHEYGF		IHEYGS		IHEYGL	
	complex	IHEYGX		IHEYGW		IHEYGZ	

Figure 7. Arithmetic Array Functions

ARRAY INDEXERS

Indexer for Simple Arrays

Module Name: IHEJXS

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_i is the multiplier for the ith dimension

L_i is the lower bound for the ith dimension

U_i is the upper bound for the ith dimension, and

n is the number of dimensions.

Range:

0 < number of dimensions < 2**22

Implementation:

- Module size: 104 bytes
- Execution time:

  Let n = the number of dimensions

  Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

  IHEJXSI: a + c*n

  IHEJXSY: b + c*n

	30	40	50	65	75
a	720	229	85.8	22.0	13.4
b	555	183	71.3	18.2	11.4
c	377	148	64.8	16.0	10.0

Indexer for Interleaved Arrays

Module Name: IHEJXI

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 $A(L_1,L_2,\ldots\ldots L_n)$. 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

Implementation:

- Module size: 328 bytes
- Execution time:

Let n = the number of dimensions of the array

$$E_i = U_i - L_i + 1$$

$$T = \prod_{i=1}^{n-1} E_i$$

$$S = \sum_{i=1}^{n-1} \prod_{j=1}^{n-i} E_j$$

V_1 = time to get a VDA (IHESADF)

V_2 = time to free a VDA (IHESAFD)

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHEJXII $a + V_1 + c*n$

IHEJXIY $b + V_1 + c*n$

IHEJXIA:

n = 1: $d + V_2 + e*E_1$

n > 1: $k + V_2 + h*S + T*(f + e*E_n + g*n*(n-1))$

The total time required to index through the complete array is the sum of:

i) the time for either IHEJXII or IHEJXIY, and

ii) the time from the appropriate IHEJXIA formula

	30	40	50	65	75
a	1620	526	203	55.9	34.4
b	1398	472	182	50.8	31.8
c	508	192	75.8	20.6	14.3
d	122	51.5	8.8	2.5	1.6
e	786	267	107	28.8	19.7
f	204	66.7	25.8	7.7	5.3
g	42	14.4	4.8	1.4	1.1
h	260	95.3	36.8	11.1	8.2
k	220	91.9	24.0	7.6	5.3

Note: the fastest overall execution time will occur when the extent of the nth dimension is the largest of the subscript extents of the array dimensions.

ARRAY FUNCTIONS

ALL (X), ANY (X)

Module Names:

Arguments	Module name
Simple arrays and interleaved arrays with variable-length elements	IHENL1
Interleaved arrays with fixed-length elements	IHENL2

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 (IHEBSA) and OR (IHEBSO) are used for ALL and ANY respectively; for string arrays with any alignment BOOL (IHEBSF) is used with appropriate parameter bits.

The elements of the array are passed to IHEBSA, IHEBSO or IHEBSF 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.

Implementation:

- Module size: IHENL1: 280 bytes
  IHENL2: 192 bytes

- Execution time:

Let R = number of elements in the array

T_1 = time required to execute appropriate bit string routine via IHEBSA0, IHEBSO0 or IHEBSF0

T_2 = time to index via IHEJXSI

T_3 = time to index via IHEJXSY

T_4 = sum of times required to index via IHEJXII and IHEJXIA

Then the approximate execution times in microseconds for the System/360 models given below are obtained from the following formulas:

IHENL1

Fixed-length elements:
$$a + R*(b + T_1 + T_2)$$

Varying-length elements:
$$a + R*(c + T_1 + T_3)$$

	30	40	50	65	75
a	1318	470	189	51.6	36.9
b	837	258	95	27.3	18.0
c	375	134	49.8	15.2	10.7

IHENL2

$$a + T_4 + R*(b + T_1)$$

	30	40	50	65	75
a	1341	479	188	53.5	36.6
b	428	133	54.0	15.2	11.5

SUM (X)

Module Names and Entry Points:

Simple Arrays

Arguments	Module name	Entry point
Fixed, real	IHESSF	IHESSF0
Fixed, complex	IHESSX	IHESSX0
Short float		
real	IHESSG	IHESSGR
complex	IHESSG	IHESSGC
Long float		
real	IHESSH	IHESSHR
complex	IHESSH	IHESSHC

Interleaved Arrays

Arguments	Module name	Entry point
Fixed, real	IHESMF	IHESMF0
Fixed, complex	IHESMX	IHESMX0
Short float		
real	IHESMG	IHESMGR
complex	IHESMG	IHESMGC
Long float		
real	IHESMH	IHESMHR
complex	IHESMH	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 IHESSG and IHESSH, while the two sums are developed concurrently in other modules.

Error and Exceptional Conditions:

I : OVERFLOW, UNDERFLOW

H : IHESSF, IHESSX, IHESMF, IHESMX: ABS(element of the array) > 7.2*10**75: SIZE condition caused in conversion package

Implementation:

- Module Sizes: 75

Module	Bytes
IHESSF	168
IHESSX	216
IHESSG	104
IHESSH	104
IHESMF	136
IHESMX	224
IHESMG	128
IHESMH	128

- Execution times:

Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate formula.

Constants used in these formulas are:

R = number of elements in the array

T_1 = sum of times required to execute IHEJXI using IHEJXIY and IHEJXIA

T_2 = time to execute IHEJXS by means of IHEJXSY

T_3 = time for the appropriate conversion using IHEDMA

The binary and decimal source data is always fixed-point; target data is short or long floating-point.

IHESSF

Source	Target	
binary	short	$a + T_2 + R*(e + T_3)$
decimal	short	$b + T_2 + R*(e + T_3)$
binary	long	$c + T_2 + R*(f + T_3)$
decimal	long	$d + T_2 + R*(f + T_3)$

	30	40	50	65	75
a	1281	446	177	46.6	31.6
b	1367	469	186	48.7	32.2
c	1308	454	179	46.6	31.4
d	1394	477	188	48.6	32.0
e	119	35.5	13.4	12.4	8.3
f	137	35.2	12.2	11.2	7.2

IHESSX

Source	Target	
binary	short	$a + T_2 + R*(e+2*T_3)$
decimal	short	$b + T_2 + R*(e+2*T_3)$
binary	long	$c + T_2 + R*(f+2*T_3)$
decimal	long	$d + T_2 + R*(f+2*T_3)$

	30	40	50	65	75
a	1309	492	208	54.9	40.7
b	1395	515	216	58.9	41.3
c	1385	518	315	58.0	41.5
d	1471	540	224	60.0	42.0
e	776	259	94.5	24.0	15.2
f	834	267	96.1	22.9	14.1

IHESSG

Real	$a + T_2 + R*b$
Complex	$c + T_2 + R*d$

	30	40	50	65	75
a	887	301	118	32.0	21.4
b	127	34.9	12.4	4.1	2.1
c	1033	347	134	36.7	24.6
d	254	67.8	24.8	8.3	4.2

IHESSH

Real	$a + T_2 + R*b$
Complex	$c + T_2 + R*d$

	30	40	50	65	75
a	935	314	121	32.0	21.4
b	167	43.9	15.2	4.2	2.1
c	1129	372	142	36.8	24.6
d	334	87.8	30.4	8.3	4.2

IHESMF

<u>Target</u>

short $a + T_1 + R*(e + T_3)$

long $c + T_1 + R*(f + T_3)$

	30	40	50	65	75
a	1074	363	141	37.3	24.6
c	1123	381	147	38.5	25.6
e	400	139	54.4	16.2	10.9
f	418	139	53.2	15.1	9.8

IHESMH

Real $a + T_1 + R*b$

Complex $c + T_1 + R*d$

	30	40	50	65	75
a	887	298	111	30.9	20.4
b	366	116	42.4	12.3	7.9
c	993	337	125	34.4	23.5
d	514	157	55.9	16.0	9.8

IHESMX

<u>Source</u>	<u>Target</u>	
binary	short	$a + T_1 + R*(e+2*T_3)$
decimal	short	$b + T_1 + R*(e+2*T_3)$
binary	long	$c + T_1 + R*(f+2*T_3)$
decimal	long	$d + T_1 + R*(f+2*T_3)$

	30	40	50	65	75
a	1327	511	229	63.8	47.8
b	1413	533	238	65.9	48.4
c	1319	515	227	63.7	47.4
d	1405	537	235	65.8	48.0
e	712	232	87	25.1	16.4
f	770	240	89	24.0	15.2

IHESMG

Real $a + T_1 + R*b$

Complex $c + T_1 + R*d$

	30	40	50	65	75
a	839	286	107	30.3	20.4
b	294	97.4	35.6	11.9	7.9
c	929	320	119	33.9	23.5
d	402	129	46.3	15.6	9.8

<u>PROD (X)</u>

Module Names and Entry Points:

<u>Simple Arrays</u>

<u>Arguments</u>	Module <u>name</u>	Entry <u>point</u>
Fixed, real	IHEPSF	IHEPSF0
Fixed complex	IHEPSX	IHEPSX0
Short float		
real	IHEPSS	IHEPSS0
complex	IHEPSW	IHEPSW0
Long float		
real	IHEPSL	IHEPSL0
complex	IHEPSZ	IHEPSZ0

<u>Interleaved Arrays</u>

<u>Arguments</u>	Module <u>name</u>	Entry <u>point</u>
Fixed, real	IHEPDF	IHEPDF0
Fixed, complex	IHEPDX	IHEPDX0
Short float		
real	IHEPDS	IHEPDS0
complex	IHEPDW	IHEPDW0
Long float		
real	IHEPDL	IHEPDL0
complex	IHEPDZ	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 : IHEPSF, IHEPSX, IHEPDF, IHEPDX: ABS(element of the array) > 7.2*10**75: SIZE condition caused in conversion package

Implementation:

- Module sizes:

Module	Bytes
IHEPSF	160
IHEPSS	72
IHEPSL	72
IHEPSX	256
IHEPSW	96
IHEPSZ	96
IHEPDF	144
IHEPDS	88
IHEPDL	88
IHEPDX	272
IHEPDW	120
IHEPDZ	120

- Execution times:

  Approximate execution times in microseconds for the System/360 models given below are obtained from the appropriate formula.

  Constants used in the formulas are:

  R = number of elements in the array

  T_1 = sum of times required to execute IHEJXI using IHEJXIY and IHEJXIA

  T_2 = time to execute IHEJXS via IHEJXSY

  T_3 = time for the appropriate conversion using IHEDMA

  The binary and decimal source data is always fixed-point; target data is short or long floating-point.

IHEPSF

Source	Target	
binary	short	$a + T_2 + R*(e+T_3)$
decimal	short	$b + T_2 + R*(e+T_3)$
binary	long	$c + T_2 + R*(f+T_3)$
decimal	long	$d + T_2 + R*(f+T_3)$

	30	40	50	65	75
a	1208	416	163	43.8	29.3
b	1294	439	172	45.9	29.9
c	1257	434	169	45.0	30.2
d	1343	456	178	47.1	30.8
e	613	188	62.3	16.1	10.3
f	1331	357	74.8	18.1	11.2

IHEPSS

$a + T_2 + R*b$

	30	40	50	65	75
a	456	191	82.3	24.6	17.7
b	372	96.9	27.0	5.9	3.3

IHEPSL

$a + T_2 + R*b$

	30	40	50	65	75
a	-252	22.5	41.5	16.0	12.9
b	1112	276	43.5	9.1	5.3

IHEPSX

Source	Target	
binary	short	$a + T_2 + R*(e+2*T_3)$
decimal	short	$b + T_2 + R*(e+2*T_3)$
binary	long	$c + T_2 + R*(f+2*T_3)$
decimal	long	$d + T_2 + R*(f+2*T_3)$

	30	40	50	65	75
a	1405	494	192	52.1	35.5
b	1491	517	200	54.1	36.0
c	1481	520	199	53.2	36.2
d	1567	542	208	55.2	36.8
e	1993	553	173	41.6	24.1
f	5043	1285	241	54.1	31.0

IHEPSW

$a + T_2 + R*b$

	30	40	50	65	75
a	-614	-69.8	7.7	8.9	10.6
b	1508	382	108	24.1	12.1

IHEPSZ

$a + T_2 + R*b$

	30	40	50	65	75
a	-3622	-791	-56.2	-4.3	2.6
b	4580	1124	180	37.7	20.1

IHEPDF

Target	
short	$a + T_1 + R*(e + T_3)$
long	$c + T_1 + R*(f + T_3)$

	30	40	50	65	75
a	1075	365	141	37.8	25.0
c	1124	382	147	39.0	25.9
e	645	201	69.0	18.2	12.1
f	1363	371	81.5	20.2	13.0

IHEPDS

$a + T_1 + R*b$

	30	40	50	65	75
a	418	178	80.0	23.7	17.5
b	492	143	43.3	11.4	7.2

IHEPDL

$a + T_1 + R*b$

	30	40	50	65	75
a	-290	8.8	67.5	20.7	15.5
b	1264	331	63.8	14.8	9.2

IHEPDX

Source	Target	
binary	short	$a + T_1 + R*(e+2*T_3)$
decimal	short	$b + T_1 + R*(e+2*T_3)$
binary	long	$c + T_1 + R*(f+2*T_3)$
decimal	long	$d + T_1 + R*(f+2*T_3)$

	30	40	50	65	75
a	1382	497	208	56.8	41.2
b	1468	520	216	58.9	41.8
c	1374	501	205	56.7	40.8
d	1460	523	214	58.8	41.4
e	2047	574	182	44.4	26.3
f	5097	1306	250	56.9	33.2

IHEPDW

$a + T_1 + R*b$

	30	40	50	65	75
a	-814	-96	12.1	8.9	12.5
b	1694	452	132	32.0	17.7

IHEPDZ

$a + T_1 + R*b$

	30	40	50	65	75
a	-3841	-852	-61.5	-4.7	4.2
b	4830	1214	211	46.1	25.7

POLY (A,X)

Module Names and Entry Points:

Arguments	Module name	Entry point
Fixed, real		
vector X	IHEYGF	IHEYGFV
scalar X	IHEYGF	IHEYGFS
Fixed, complex		
vector X	IHEYGX	IHEYGXV
scalar X	IHEYGX	IHEYGXS
Short float, real		
vector X	IHEYGS	IHEYGSV
scalar X	IHEYGS	IHEYGSS
Short float, complex		
vector X	IHEYGW	IHEYGWV
scalar X	IHEYGW	IHEYGWS
Long float, real		
vector X	IHEYGL	IHEYGLV
scalar X	IHEYGL	IHEYGLS
Long float, complex		
vector X	IHEYGZ	IHEYGZV
scalar X	IHEYGZ	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 : IHEYGF, IHEYGX: ABS(element of the array) > 7.2*10**75: SIZE condition caused in conversion package

Implementation:

- Module sizes:

Module	Bytes
IHEYGF	432
IHEYGS	240
IHEYGL	240
IHEYGX	688
IHEYGW	280
IHEYGZ	280

- Execution times:

  Let the arguments be declared as A(m:n) and X(p:q), or X, and T be the time for one conversion using the arithmetic conversion director IHEDMA. Then the approximate execution times in microseconds for the System/360 models given are obtained from the appropriate formula. 'Short' or 'long' refers to the floating-point result.

IHEYGF

Scalar X:

short a + 2*T + (n-m)*(b+T)

long c + 2*T + (n-m)*(d+T)

Vector X, (q - p ≥ n - m - 1):

short e + T + (n-m)*(f+2*T)

long g + T + (n-m)*(h+2*T)

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

short i + 2*T + (n-m)*(b+T) + (q-p+1)*(j+T)

long k + 2*T + (n-m)*(d+T) + (q-p+1)*(l+T)

	30	40	50	65	75
a	2297	834	338	99.0	67.1
b	904	387	100	26.0	17.1
c	2408	870	351	102	69.2
d	1706	484	124	30.4	20.2
e	2629	910	370	97.1	64.6
f	1480	459	155	41.2	26.3
g	2740	946	383	99.4	66.7
h	3044	844	200	50.0	32.6
i	3370	1197	491	140	94.0
j	258	91	35	9.5	6.0
k	3581	1233	504	143	96.1
l	280	101	39	10.7	7.1

IHEYGX

Scalar X:

short a + 4*T + (n-m)*(b+2*T)

long c + 4*T + (n-m)*(d+2*T)

Vector X, (q - p ≥ n - m - 1):

short e + 2*T + (n-m)*(f+4*T)

long g + 2*T + (n-m)*(h+4*T)

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

short i + 4*T + (n-m)*(b+2*T) + (q-p+1)*(j+2*T)

long k + 4*T + (n-m)*(d+2*T) + (q-p+1)*(l+2*T)

	30	40	50	65	75
a	3245	1174	478	142	95.4
b	2345	664	220	58.3	33.5
c	3471	1221	496	145	97.6
d	5519	1433	301	73.2	42.6
e	3187	1114	447	127	83.5
f	4368	1226	399	103	58.1
g	3533	1161	465	129	85.7
h	10636	2746	556	133	76.4
i	4087	1459	592	171	114
j	545	187	72.8	21.1	12.7
k	4243	1506	610	176	116
l	567	196	76.8	22.3	13.6

IHEYGS, IHEYGL, IHEYGW, IHEYGZ

Scalar X: a + (n-m)*b

Vector X:

(q-p≥n-m-1): c + (n-m)*d

(q-p<n-m-1): e + (n-m)*b + (q-p+1)*f

IHEYGS

	30	40	50	65	75
a	1232	430	182	49.5	33.3
b	461	121	37.6	9.8	5.3
c	1871	623	259	69.6	45.4
d	490	128	40.9	10.5	5.7
e	2140	733	304	83.6	54.7
f	29	7.5	3.3	0.7	0.4

IHEYGL

	30	40	50	65	75
a	1320	451	189	49.6	33.3
b	1241	308	56.9	13.1	7.3
c	1959	644	266	69.7	45.4
d	1270	316	60.2	13.7	7.7
e	2228	755	311	83.6	54.7
f	29	7.5	3.3	0.7	0.4

IHEYGW

	30	40	50	65	75
a	1396	475	198	54.9	36.1
b	1672	425	126	30.5	15.0
c	2035	667	275	75.0	48.1
d	1701	432	129	31.2	15.4
e	2304	775	320	88.9	57.4
f	29	7.5	3.3	0.7	0.4

IHEYGZ

	30	40	50	65	75
a	1572	517	211	54.9	36.1
b	4824	1184	203	44.2	23.0
c	2211	710	288	75.0	48.1
d	4853	1192	207	44.9	23.4
e	2480	821	333	89.0	57.4
f	29	7.5	3.3	0.7	0.4

READER'S COMMENTS

Title: IBM System/360 Operating System
PL/I Subroutine Library
Computational Subroutines

Form: C28-6590-0

Is the material:

	Yes	No
Easy to read?	---	---
Well organized?	---	---
Complete?	---	---
Well illustrated?	---	---
Accurate?	---	---
Written for your technical level?	---	---

How did you use this publication?

-----As an introduction to the subject

-----For additional knowledge

Other------------------------

Please check the items that describe your position:

-----Customer personnel	----Operator	-----Sales Representative
-----IBM personnel	----Programmer	-----Systems Engineer
-----Manager	----Customer Engineer	-----Trainee
-----Systems Analyst	----Instructor	-----Other---------------

Please check specific criticisms, give page numbers, and explain below:

-----Clarification on pages
-----Addition on pages
-----Deletion on pages
-----Error on pages

Explanation:

If you wish a reply, be sure to include your name and address.

C28-6590-0

fold

fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION

1271 AVENUE OF THE AMERICAS
NEW YORK, N.Y. 10020

ATTENTION: PUBLICATIONS, DEPT. D39

fold

fold

Printed in U.S.A. – C28-6590-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

Printed in U.S.A. C28-6590-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601