IBM Application Program

GY20-0568-0

CALL/360-OS
PL/I System Manual Volume II

Program Number 360A-CX-42X

The CALL/360-OS PL/I compiler (to be used with the CALL/360-OS System on an IBM System/360 Model 50 or higher) is described in the four volumes of this publication. The publication is addressed to system programmers and customer engineers who require a detailed knowledge of the compiler. It contains a general overview of the compiler and detailed information on the compiler and runtime routines and macros that perform required functions. Additional information required to understand CALL/360-OS PL/I compiler operations is provided in several appendices.

Volume II contains information on runtime support modules and the first part of a directory to runtime routines.

First Edition (August 1970)

This edition applies to Version 1, Modification Level 0, of CALL/360-OS (360A-CX-42X) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for reader's comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 112 East Post Road, White Plains, New York 10601.

© Copyright International Business Machines Corporation 1970

CONTENTS - VOLUME II

FIGURES - VOLUME II

SECTION 4 - RUNTIME SUPPORT SUMMARY

General information pertaining to the modules which support the run-time function is presented in this section. The modules are divided into two main services:

1. Library Interface Services (LIBINT)
2. Library Computational Services (LIBCOMP)

Individual routines (modules) are explained in detail in the section that follows.

LIBRARY INTERFACE SERVICES (LIBINT)

The CALL/360-OS PL/I library is the primary interface between the CALL/360-OS PL/I object code and the system at object time. Most system macro instructions are issued via library calls; this minimizes the compiler's dependency on stability of the structure of SVC's (Supervisor Calls).

The Library Interface Services (LIBINT) modules perform the following general functions:

1. Handle stream I/O requests from the CALL/360-OS PL/I object program by interfacing with the I/O access methods. LIBINT modules which perform this function are grouped into the I/O Management Package (IOMP).

2. Handle conditions which cause interruption to the main flow of a program at object time by interfacing with the system. LIBINT modules which perform this function are grouped into the Handling of Interrupts Package (HIP). The execution error package (EXEP) is a subpackage of this package.

   Note: In the text of this manual, all lowercase letters (rather than initial uppercase letters followed by lowercase letters) are used for subpackages to distinguish them from packages.

3. Achieve dynamic management of the CALL/360-OS PL/I object program (housekeeping, including GO TO interpretation, and allocation and freeing of dynamic storage). LIBINT modules which perform this function are grouped into the Management of Object Program Package (MOPP).

I/O MANAGEMENT PACKAGE (IOMP)

The following I/O Management Package routines support CALL/360-OS PL/I stream-oriented statements, options, and built-in functions:

IHECLOSE	IHEIOA	IHEIOX
IHEDDI	IHEIOB	IHELDI
IHEDDO	IHEIOD	IHELDO
IHEDDP	IHEIOG	IHEOPEN
IHEDIO	IHEIOP	IHERSET

I/O Record Format

Each record of a CALL/360-OS disk file is 3440 bytes long. The first seven bytes are used for record description. The remaining bytes are used for data.

The record description information is coded as follows:

bytes 1-4:	number of bytes used in record (includes first seven bytes)		
byte 5:	bit 0:	=0	This is not last logical record in file.
		=1	This is last logical record in file.
	bits 1-7:		zero (Not currently used.)
byte 6:	bit 0:	=0	Internally coded file
		=1	Externally coded file
	bit 1:	=0	All items in record of same arithmetic type (<u>Note</u>: cannot be used if strings in record)
		=1	Not all items in record of same arithmetic type or for externally coded file
	bits 2-7:		zero (Not currently used.)
byte 7:	zero (Not currently used.)		

Externally Coded Files

Bytes 8 to 3440 contain a stream of EBCDIC characters. The number of characters in the file is determinable from the record length.

Internally Coded Files

If the record is of the type that all items in the record are of the same arithmetic type, then byte 8 contains the code and the data items follow in a continuous stream. The record count (bytes 1 to 4) is used to determine the number of items in the record. (Notice that this gives proper boundary alignment for all arithmetic items in the record.) Though internal records formatted as described above are processed correctly, CALL/360-OS PL/I internal I/O formats all internal records as described below.

If all items are not of the same type, then starting in byte 8, each record consists of a series of specifications concatenated together (the record count is used to determine the number of specifications). Each specification consists of at least three fields. The syntax of a specification is:

$$\text{Code}\left\{\begin{array}{l}\text{replication item}\ldots\\ \text{length item}\ldots\end{array}\right\}$$

where:

<u>Code</u> is one byte in length and contains:

bits 0-4: zero

bits 4-7:	=0001 if character string	
	=0010 if fixed item	=1010 if fixed complex item
	=0011 if short float item	=1011 if short float complex item
	=0111 if long float item	=1111 if long float complex item

<u>Replication</u> is one byte indicating the number of fixed or float items following it.

<u>Length</u> is one byte indicating the length of the character string item following it.

In this record format, specifications are not split across records, and boundary alignment for arithmetic items is not assured.

Input End of File Detection

On input, end-of-file can be recognized either by a record length of 8 or by bit 0 of byte 5 being set to one. Before obtaining the next record from disk, this bit is checked. If it is one, there is no need to read the next record because the record just processed was the last record in the file.

Coding of Byte 6

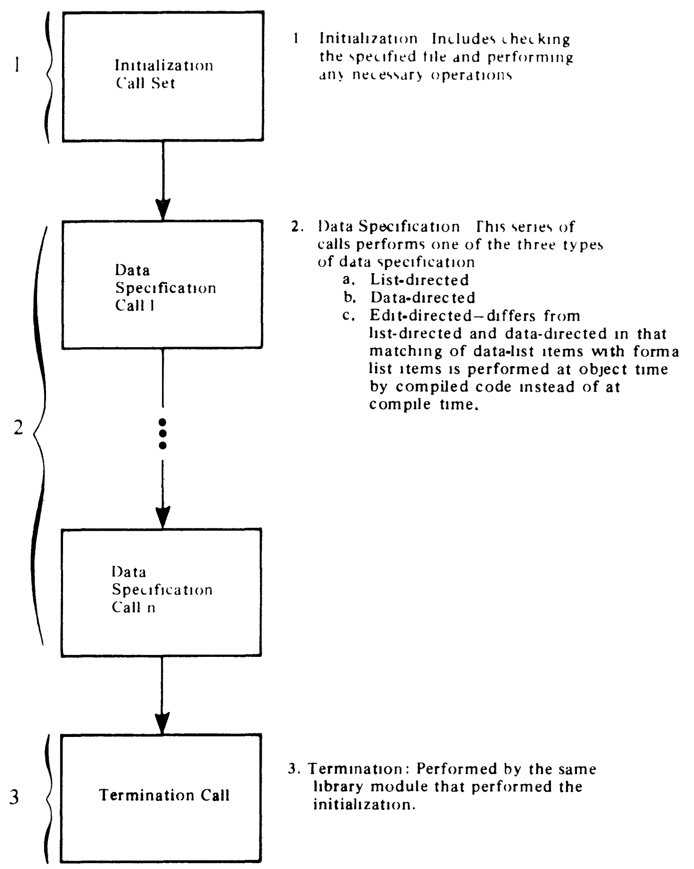
The codes in byte 6 are set so that a test under mask of bits 0 and 1 together determines the type of record. An all ones result identifies an externally coded file. Mixed ones and zeros denotes internally coded, not of same type. All zeros signifies internally coded, all of the same type.

Coding of Code Field for Internal Format

If bit 6 of the code field is one, the type is arithmetic. If arithmetic and bit 7 is one, the scale is float. Bits 5 through 7 give the length of the floating item. If the scale is fixed, then making bit 7 a one gives the length of the fixed item. (All lengths are length - 1 as required for MVC command.)

<u>Stream-Oriented I/O</u>

The use of stream-oriented I/O allows a program to ignore record boundaries and handle data fields instead. The GET and PUT statements are used to transmit data between storage and records which exist in a buffer. IOMP modules assume that the compiled code for the GET and PUT statements will have the general structure of three call sets to the library, as illustrated in Figure 4-1.



1
Initialization Call Set
1 Initialization Includes checking the specified file and performing any necessary operations
Data Specification Call 1
2. Data Specification This series of calls performs one of the three types of data specification
a. List-directed
b. Data-directed
c. Edit-directed—differs from list-directed and data-directed in that matching of data-list items with forma list items is performed at object time by compiled code instead of at compile time.
2
Data Specification Call n
3
Termination Call
3. Termination: Performed by the same library module that performed the initialization.

Figure 4-1. GET and PUT Compiled Code Structure

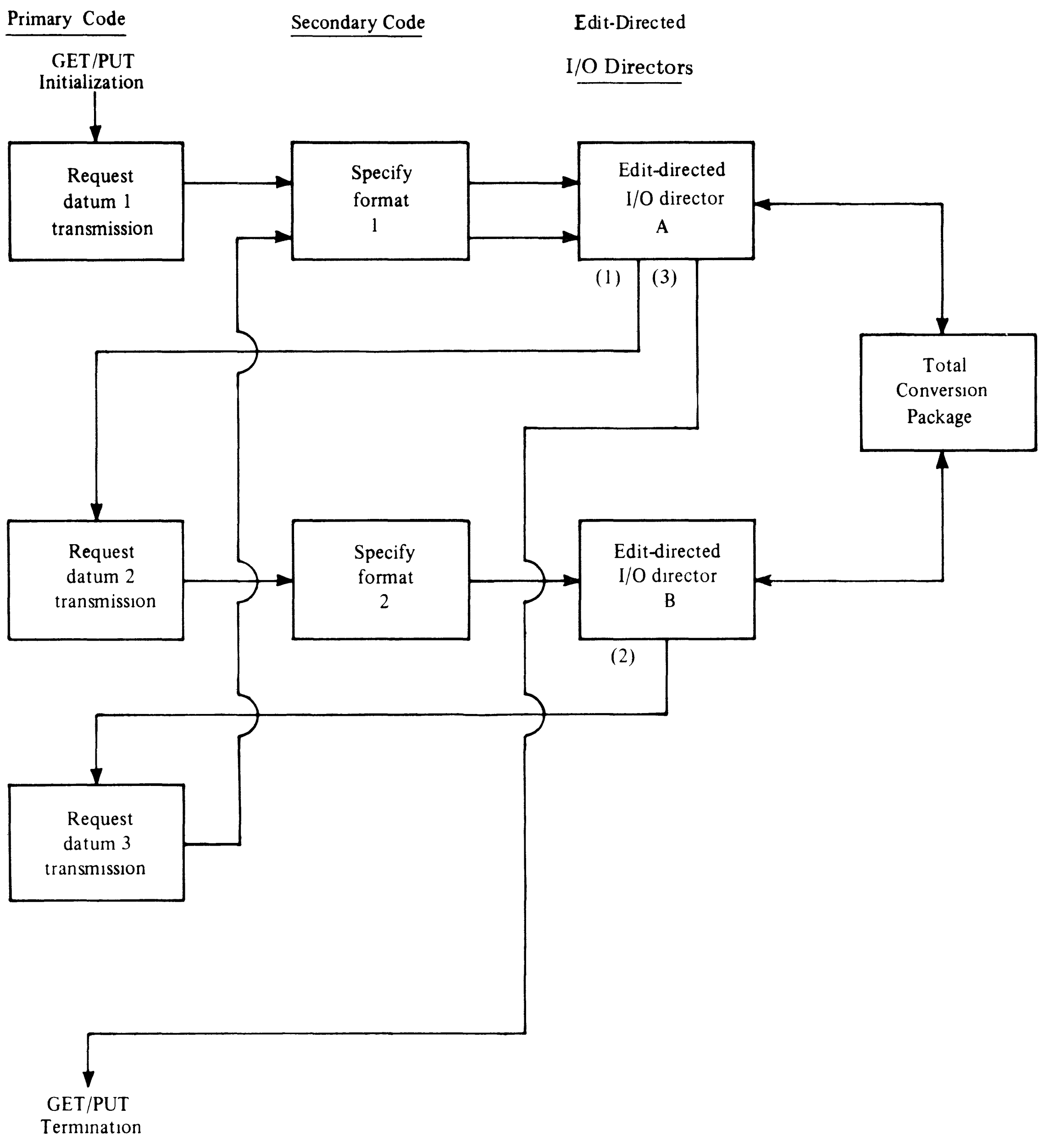
The data-specification call-set is handled differently for edit-directed I/O than for data-directed or list-directed I/O. The matching of data-list items and format-list items at object time by compiled code is necessitated by the CALL/360-OS PL/I language specifications for edit-directed I/O. Format items may have expressions for replication factors and format subfields, and these expressions may be evaluated with values read in during the same, or another, GET operation. Furthermore, due to this use of dynamic replication factors and to the use of array data-list items with variable bounds, there may not be a predeterminable matching between data-list and format-list items.

To implement dynamic matching, the "executable format scheme" is used. Basically, this scheme calls for two location counters, one for a compiled series of data-list item requests (the "primary code") and the other for a compiled series of format-list item specifications (the "secondary code").

The executable format scheme depends upon the edit-directed I/O directors (IHEDIA, IHEDOA, IHEDIM, IHEDOM, IHEDIB, and IHEDOB) in the Total Conversion Package (TCP) of the library to determine and supervise the performance of the necessary conversion, as dictated by the internal representation of the data item (described by its data element descriptor) and its specified external representation (described by a format element descriptor).

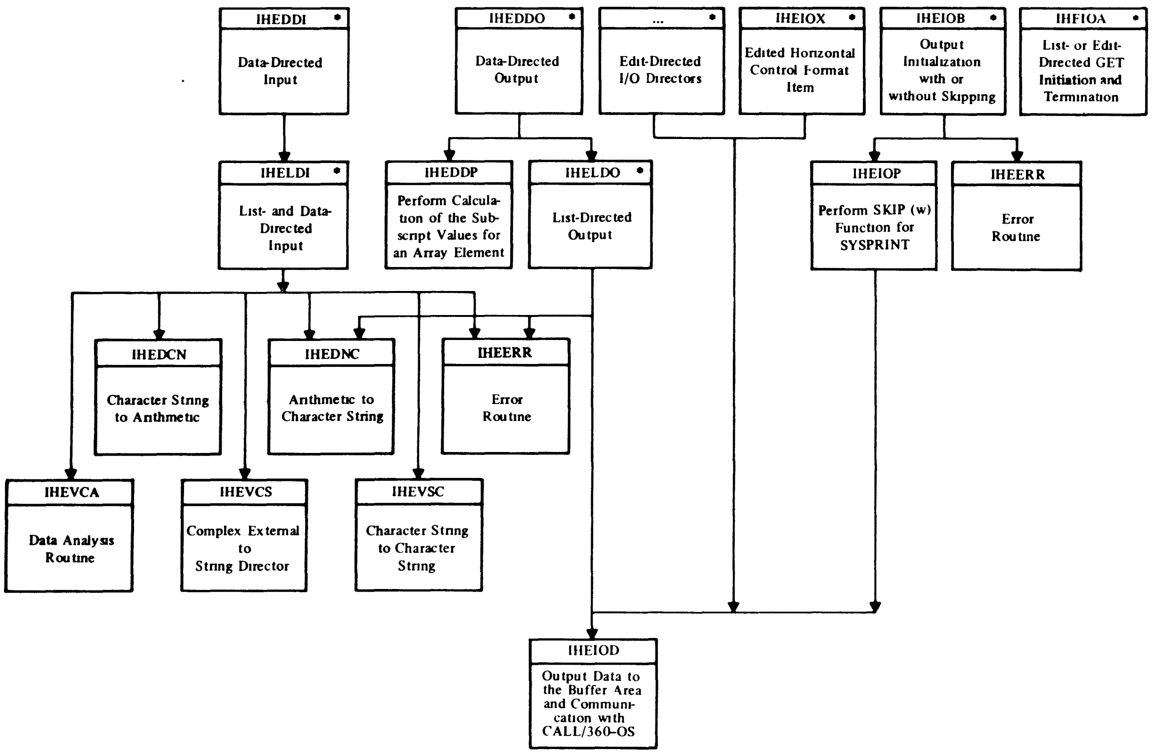
Figure 4-2 illustrates the executable format scheme when there are three data items in the data list and two format items in the format list. Note that the number of data-list items determines the duration of the operation of the executable format scheme. The secondary code is reused from the beginning until all the primary code is exhausted.

Figure 4-3 describes the modular linkage through stream-oriented input and output.



Primary Code
Secondary Code
Edit-Directed I/O Directors
GET/PUT Initialization
Request datum 1 transmission
Specify format 1
Edit-directed I/O director A
(1)
(3)
Total Conversion Package
Request datum 2 transmission
Specify format 2
Edit-directed I/O director B
(2)
Request datum 3 transmission
GET/PUT Termination

Figure 4-2. Executable Format Scheme



IHEDDI •
Data-Directed Input
IHEDDO •
Data-Directed Output
... •
Edit-Directed I/O Directors
IHEIOX •
Edited Horizontal Control Format Item
IHEIOB •
Output Initialization with or without Skipping
IHEIOA •
List- or Edit-Directed GET Initiation and Termination
IHELDI •
List- and Data-Directed Input
IHEDDP
Perform Calculation of the Subscript Values for an Array Element
IHELDO •
List-Directed Output
IHEIOP
Perform SKIP (w) Function for SYSPRINT
IHEERR
Error Routine
IHEDCN
Character String to Arithmetic
IHEDNC
Arithmetic to Character String
IHEERR
Error Routine
IHEVCA
Data Analysis Routine
IHEVCS
Complex External to String Director
IHEVSC
Character String to Character String
IHEIOD
Output Data to the Buffer Area and Communication with CALL/360-OS

Note 1) An asterisk indicates that the module can be entered directly from the compiled code
2) The edit-directed I/O directors are IHEDOB, IHEDOM, and IHEDOA They and IHEDCN, IHEDNC, IHEVCA, IHEVCS, and IHEVSC are part of the Total Conversion Package of the CALL/360-OS PL/I library.

Figure 4-3. Modular Linkage through Stream-Oriented I/O

HANDLING OF INTERRUPTS PACKAGE (HIP)

The Handling of Interrupts Package routines handle conditions that cause interruption to the main flow of a program at object time. These conditions are CALL/360-OS PL/I-defined execution error conditions (for which it is possible to specify an on-unit).

Execution error handling is performed by the routines IHEERR and IHEERN. These two routines and the routine IHEDUM make up the execution error package (EXEP). EXEP and the routine IHEONREV constitute HIP.

Control is passed to IHEERR as the result of an execution error. If IHEERR determines that an error message must be printed, it calls IHEERN to do the printing.

For CALL/360-OS PL/I-defined conditions, the following main sequence of events takes place in IHEERR:

1. Recognize the condition.

2. If condition disabled, return to interrupted program.

3. If there is an on-unit, generate code to call the on-unit.

   If there is no on-unit, take standard system action for the condition and return to the interrupted object code or terminate.

To carry out these operations, IHEERR depends on information passed when the actual interrupt occurs, along with information set by the compiled code in the DSA for each procedure. The information passed when the actual interrupt occurs is contained in the user's communications area. The information set by the compiled code is contained in a fixed portion of the DSA for each procedure. The compiled code prologue allocates space in the DSA for every ON statement appearing in the block, and for each ON-condition which is disabled within the block. Standard system actions are performed for the enabled conditions which have no on-units associated with them.

For errors in the library modules, an error-code is set in the library communications area (LCA). This error code is used by IHEERR to determine what action is to be taken.

MANAGEMENT OF OBJECT PROGRAM PACKAGE (MOPP)

The Management of Object Program Package (MOPP) is concerned with the dynamic management of a CALL/360-OS PL/I program. The general housekeeping requirements are performed, along with the allocation, freeing, and control of dynamic storage.

The requirements for block housekeeping are as follows.

1. Prologues and Epilogues. Prologues and epilogues are the routines executed on entry to and exit from a procedure or begin block. The MOPP modules support certain sections of the prologue and epilogue which are common to all prologues and epilogues. These common functions are:

   a. To preserve the environment of the invoking block.

   b. To obtain AUTOMATIC storage for the block.

   c. To provide chaining mechanisms to enable the program's progress to be traced.

   The epilogue functions performed by the GO TO Interpreter (IHESAF) release storage for the block and recover the environments of the invoking block before returning control to it. Since there are two types of block, a means of identifying the storage associated with each one is provided.

2. Treatment of GO TO. In CALL/360-OS PL/I, a GO TO statement involves not only the transfer of control to a particular label in a block, but also the termination of contained blocks. Therefore, both a return address and a means of identifying the AUTOMATIC storage associated with the block to be made current are required. A pointer address constant area for each block is required. The logic for interpreting GO TO statements, in order to perform these housekeeping functions, is contained in IHESAF.

3. AUTOMATIC Variables. A special type of AUTOMATIC storage area is necessary for variables whose extents are not known at compile time. The storage obtained for a particular block must be associated with that block for epilogue purposes.

LIBRARY COMPUTATIONAL SERVICES (LIBCOMP)

The Library Computational Services (LIBCOMP) modules operate on data and shape it to the user's requirements.

A library computational services module exists for any one of the following reasons:

1. To support one of the built-in function names listed under "Mathematical Built-In Functions" in Appendix A of the CALL/360-OS PL/I Language Reference Manual.
2. To support one of the built-in function names listed under "Arithmetic Built-In Functions" in Appendix A of the CALL/360-OS PL/I Language Reference Manual.
3. To support often-used arithmetic operations.
4. To support one of the built-in function names listed under "String Functions" in Appendix A of the CALL/360-OS PL/I Language Reference Manual.
5. To support often-used string operations.
6. To support one of the built-in function names listed under "Generic Functions for Manipulation of Arrays" in Appendix A of the CALL/360-OS PL/I Language Reference Manual.
7. To support data representation conversion.
8. To perform services for other LIBCOMP routines.

The LIBCOMP modules can be organized in two groups:

Conversion
- Total Conversion Package (TCP)
- --includes arithmetic conversion package (ACP)
- String Manipulation Package (SIMP)

Function Support
- Arithmetic Function Package (AFUNC)
- Mathematical Function Package (MFUNC)
- Aggregate Manipulation Package (AMP)

Note: In text of this manual, all lowercase letters (rather than initial uppercase letters followed by lowercase letters) are used for subpackages to distinguish them from packages.

Some built-in function names are supported by in-line code generation. Built-in function names in this category are:

SIGN	HBOUND
FLOOR	DIM
CEIL	CHAR
TRUNC	COMPLEX
MOD	IMAG
CONJ	REAL
LBOUND	ABS(real argument)

TOTAL CONVERSION PACKAGE (TCP)

CALL/360-OS PL/I restricts the user in selecting the form of representation of his data, both on the external medium and in storage internally. However, considerable flexibility is permitted in specifying changes of data type and form. The Total Conversion Package controls editing and data conversion. It is designed to implement the full set of editing and conversion functions. The complete language range is specified in Figure 4-4, except for complex items, for which the real and imaginary parts are treated separately.

FROM \ TO		Internal Data Type			Output Format		
		FIX	FLT	CS	F	E	A
Internal Data	FIX	X	X	X	X	X	X
	FLT	X	X	X	X	X	X
	CS	X	X	X	X	X	X
Format Items	F	X	X	X	X	X	
	E	X	X	X	X	X	
	A	X	X	X			X

Figure 4-4. Changes of Data Type and Form

To avoid unnecessary duplication of code, use is made of standard intermediate forms. This reduces the number of library routines required to cover logical converstions to less than 25. All the routines contained within the Total Conversion Package are called by means of the CALL/360-OS PL/I standard calling sequence.

All data conversions are handled by the Total Conversion Package. The routines in this package differ from the routines in the other four LIBCOMP packages in three major ways:

1. The TCP routines do not exist to support built-in functions specified in Appendix A of the CALL/360-OS PL/I Language Reference Manual, but rather to implement the rule that conversion should be possible from any one CALL/360-OS PL/I data representation to any other.

2. The TCP routines do not return a new value; rather, they return a new representation of the same value they were presented with as an argument.

3. Some of the TCP routines function as directors. Given the input representation and the desired returned representation, a director routine is incapable of performing this conversion within itself. However, it does supervise the linkage to other TCP routines (including other directors) which can perform this conversion. This linkage to other routines is transparent to the user.

The TCP routines can be organized in the following functional groups:

1. Edit-directed I/O directors
2. Type conversion directors
3. Mode conversion director
4. String conversion routines
5. Arithmetic conversion routines and director
6. Constant analysis routine

The edit-directed I/O directors supervise the conversion that is necessary during edit-directed stream I/O. On input, a character-string in the data stream (which is described by a format item in the format list of the GET statement) must be converted to an internal data representation (which is described by the attributes associated through a DED with the identifier which appears in the data list of the GET statement). On output, the process is reversed.

The director used for a particular I/O operation depends on the type of format item being used and whether the operation is input or output. There are three general types of format items: real arithemtic, complex arithmetic, and string.

The TCP routines which are directors for real arithmetic format items are:

System Name	Format Item Type
IHEDIAA	Input with F-format item
IHEDIAB	Input with E-format item
IHEDOAA	Output with F-format item
IHEDOAB	Output with E-format item

The TCP routines which are directors for complex arithmetic format items are:

System Name	Format Item Type
IHEDIMA	Input with C-format item
IHEDOMA	Output with C-format item

A C-format item is two real arithmetic format items, the first for the real part and the second for the imaginary part. They need not be of the same type.

The TCP routines which are directors for string format items are:

System Name	Format Item Type
IHEDIBA	Input with A-format item
IHEDOBA	Output with A-format item (field-width specified)
IHEDOBB	Output with A-format item (no field-width specified)

The type conversion directors supervise conversion between string and arithmetic data. These directors have an application during list-directed and data-directed I/O operations where character-string representations of arithmetic and string constants are converted to internal data representations. The TCP routines which perform type conversion are:

System Name	Conversion Function
IHEDCNA	From a character-string representation (which includes a character representation of a valid arithmetic constant or complex expression) to an arithmetic representation (of specified mode, scale, and precision)
IHEDNCA	From an arithmetic source (of specified mode, scale, and precision) to a character string

The mode conversion director handles complex arithmetic data.

System Name	Function
IHEUPAA	For a coded complex representation, zero the real part and return a pointer to the imaginary part
IHEUPAB	For a coded complex representation, return a pointer to the imaginary part and/or zero the imaginary part

The string conversion routines perform representation conversion from one string type to another string type. There is one string type from the point of view of these routines, fixed-length character-string (FLC).

There are two string conversion routines:

System Name	Conversion Function
IHEVCSA	Complex character to internal string
IHEVSCA	Character-string assignment (from FLC)

The arithmetic conversion routines perform all conversions where both the input representation and the returned representation are arithmetic. An arithmetic representation includes not only the internal arithmetic representations (defined in the discussions of MFUNC and AFUNC), but also character strings in the data stream that are arithmetic representations by virtue of being in the form of arithmetic constants under list-directed and data-directed data transmission, or by virtue of being paired with arithmetic format elements under edit-directed data transmission.

The Arithmetic Conversion Director (IHEDMA) performs the function of director for a given conversion. IHEDMA sets up a sequence of calls to two or three other arithmetic conversion routines. At the end of this sequence, the desired arithmetic conversion will have been performed. However, this sequence is transparent to the user and IHEDMA appears to return the returned representation.

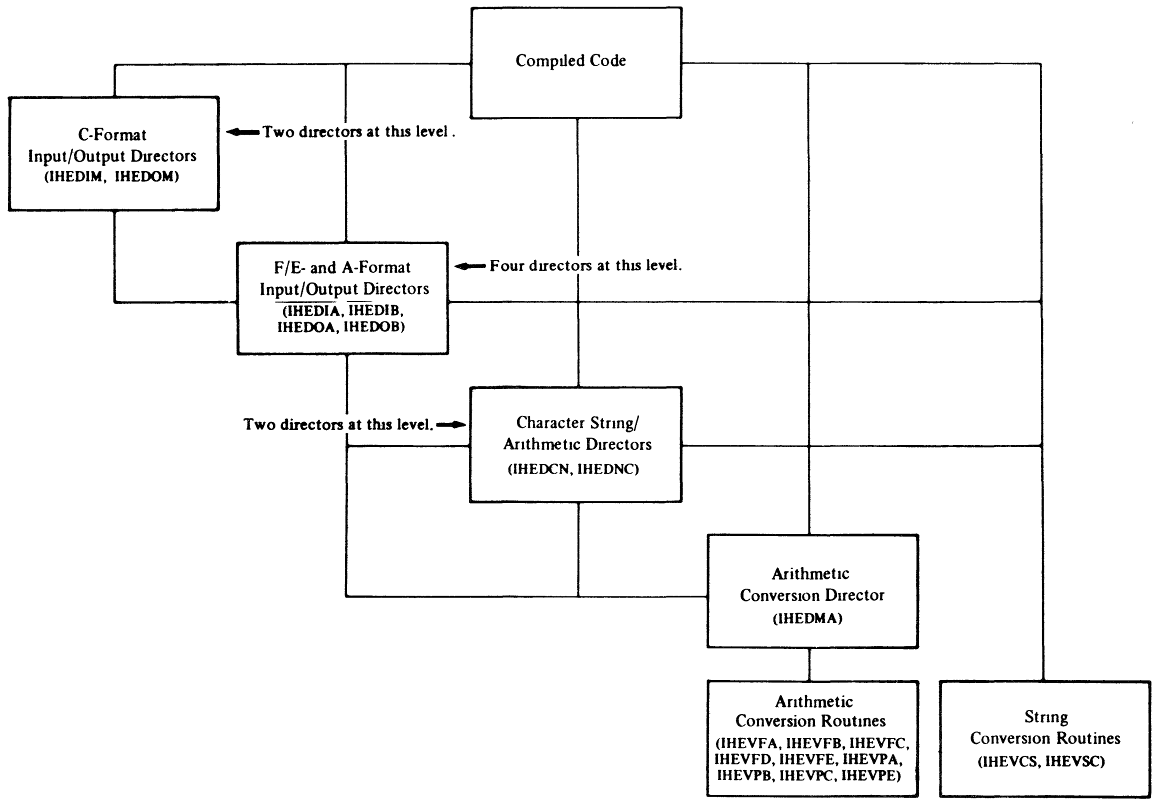
The user, compiler, or library cannot invoke any of the other arithmetic conversion modules directly; instead they must be invoked indirectly through director entry point IHEDMAA. The functions of these modules are not specified here, but each is described in the next section.

The constant analysis routine scans the character representation of an arithmetic constant, determines its mode, scale, and precision, and then creates a DED that reflects these attributes.

System Name	Function
IHEVCAA	Create a DED for an arithmetic constant

Structure of Total Conversion Package

To perform a change from a source data item to a target data item may involve a succession of steps and the use of several individual library routines within the Total Conversion Package. The structure of the package is shown in Figure 4-5.



Compiled Code
C-Format Input/Output Directors (IHEDIM, IHEDOM)
Two directors at this level.
F/E- and A-Format Input/Output Directors (IHEDIA, IHEDIB, IHEDOA, IHEDOB)
Four directors at this level.
Two directors at this level.
Character String/ Arithmetic Directors (IHEDCN, IHEDNC)
Arithmetic Conversion Director (IHEDMA)
Arithmetic Conversion Routines (IHEVFA, IHEVFB, IHEVFC, IHEVFD, IHEVFE, IHEVPA, IHEVPB, IHEVPC, IHEVPE)
String Conversion Routines (IHEVCS, IHEVSC)

Figure 4-5. Total Conversion Package Structure

In association with each individual step, knowledge of the attributes must be available for the source fields, target fields, or both. The required information is provided in the calling sequences with each data item, by means of the FED or DED.

In certain cases, when the form of the data on the external medium is not known until object time, it is necessary for the library to generate dynamically the control information it needs in this interpretive scheme.

There are director routines at four levels, as illustrated by Figure 4-5.

1. Complex Format Directors.
2. Input/Output Format Directors.
3. String to Arithmetic and Arithmetic to String Directors.
4. Arithmetic Conversion Director.

These routines are used for two main purposes:

1. The matching of source element with target element, which may not be known at compile time.

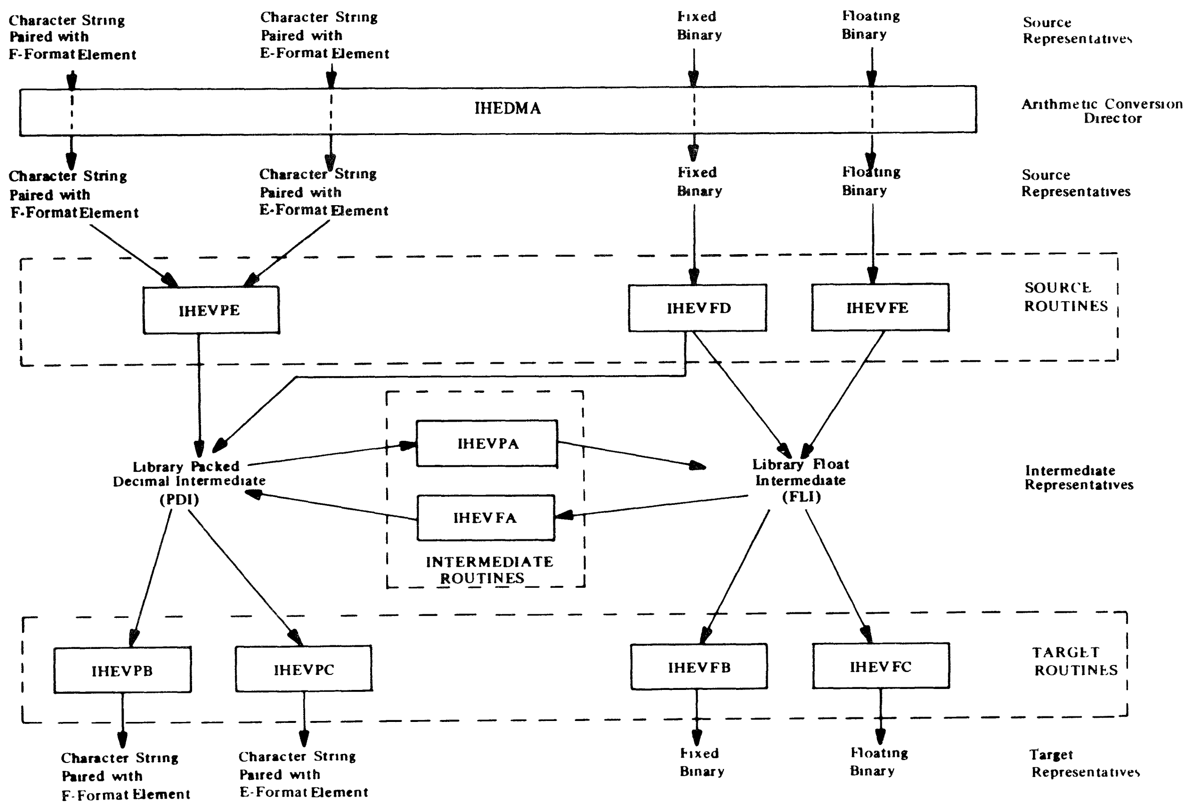
2. The controlling of the flow at object time by means of interpretive information passed to them.

The latter function is best illustrated by the Arithmetic Conversion Director (IHEDMA); a single call to IHEDMA determines the flow through a subpackage of nine arithmetic conversion routines.

Each group of directors can be called directly from compiled code or other library modules, and any director can call any other below it in the structure. Further details are provided in the explanations of the individual routines in the following section.

Conversions - Type Arithmetic

The library subpackage for arithmetic type conversions carries out editing and conversion activity for all source fields of type arithmetic that have target fields also of type arithmetic. Included in the scheme are format items and constants, which are representations on the external medium of arithmetic type data. The flow control through this subpackage is achieved by the Arithmetic Conversion Director. The method employed is to use an intermediate form of representation, according to the form of the source data, and to relate this intermediate form to the target data, either by direct conversion or by use of a second intermediate form. The latter case implies radix change. The two intermediate forms in use are referred to as packed decimal intermediate (or decimal intermediate) and float intermediate. In machine terms, the first is 16 digits and a sign together with a one-word scale factor (SWCF) in binary, representing powers of ten; the second is the standard long floating-point internal representation. The logical flow through the package is best indicated by Figure 4-6.


Character String Paired with F-Format Element
Character String Paired with E-Format Element
Fixed Binary
Floating Binary
Source Representatives
IHEDMA
Arithmetic Conversion Director
Character String Paired with F-Format Element
Character String Paired with E-Format Element
Fixed Binary
Floating Binary
Source Representatives
IHEVPE
IHEVFD
IHEVFE
SOURCE ROUTINES
IHEVPA
Library Packed Decimal Intermediate (PDI)
Library Float Intermediate (FLI)
IHEVFA
Intermediate Representatives
INTERMEDIATE ROUTINES
IHEVPB
IHEVPC
IHEVFB
IHEVFC
TARGET ROUTINES
Character String Paired with F-Format Element
Character String Paired with E-Format Element
Fixed Binary
Floating Binary
Target Representatives

Note In addition, module IHEVTB is used by radix conversion routines IHEVPA and IHEVFA.

Figure 4-6. Arithmetic Conversion Subpackage Structure

The Arithmetic Conversion Director links the routines required for a particular arithmetic conversion. It is called either by compiled code directly or by other director routines.

The flag bytes in the source and target DED's are interrogated to determine which routines are required for the current conversion and their order of execution. The library communications area (LCA) is used to record information required by successive modules, as follows:

WBR1	Address of entry point of second routine.
WBR2	Address of entry point of third routine (if required).
WRCD	Target information.

Control is then passed by the Arithmetic Conversion Director to the first routine in the chain. The first transfers to the second, and so on, until the conversion is complete. The last routine returns to the program which called the Arithmetic Conversion Director.

All the routines which can be first in the chain set up by Arithmetic Conversion Director use the source parameters passed to it. With one exception, the first conversion is always to the intermediate form of the same radix as the source, and this result is stored in LCA in the slot named WINT for binary radix, and in slots WINT and WSFC for decimal radix. (The exception is converting from a fixed-point binary to an F/E-format data item, in which case the intermediate form is packed decimal direct.)

Three routines in the arithmetic conversion package deal with data on the external medium to handle the output of F- and E-format items from packed decimal intermediate, and conversion from F- or E-format items to packed decimal intermediate. In each case, a corresponding format element descriptor is required, and its address is contained in the LCA slot WFED.

There are nine routines in the arithmetic conversion package (ACP). To perform one arithmetic conversion, three or four of these routines are executed in a sequence, called the <u>ACP sequence</u>. The first routine in the sequence is <u>always</u> the Arithmetic Conversion Director (IHEDMA). The only way a user can call the ACP to perform an arithmetic conversion is by calling IHEDMA. The function of IHEDMA is to choose which of the other routines in the ACP will be the second, third, and (possibly) fourth routines in the ACP sequence.

The user need not know which routines are used as the second, third, or fourth routines, or even that there are other routines in the ACP beside IHEDMA. From the user's point of view, he need only call IHEDMA with a list of four parameters (the source address, the source descriptor address, the target address, and the target descriptor address), and the target representation desired by the user will be available in the parameter list target address when control is returned to the user.

The source representations recognized by the ACP and the target representations that it can produce are shown in Figure 4-6. The conversion from any one of the representations in the source column to any of the representations in the target column is performed (from the user's viewpoint) by simply calling <u>one</u> routine--IHEDMA.

The conversion logic flow is shown in Figure 4-7. Each box indicates the entry point name(s) and the library level assigned to the module, together with a brief description of the function performed. The arrows denote the direction of logic flow. Segments connected by dual arrows indicate that program control returns to the calling module upon completion of the function for which the called module was invoked. The invocation direction is downward. The return direction, if any, is upward.

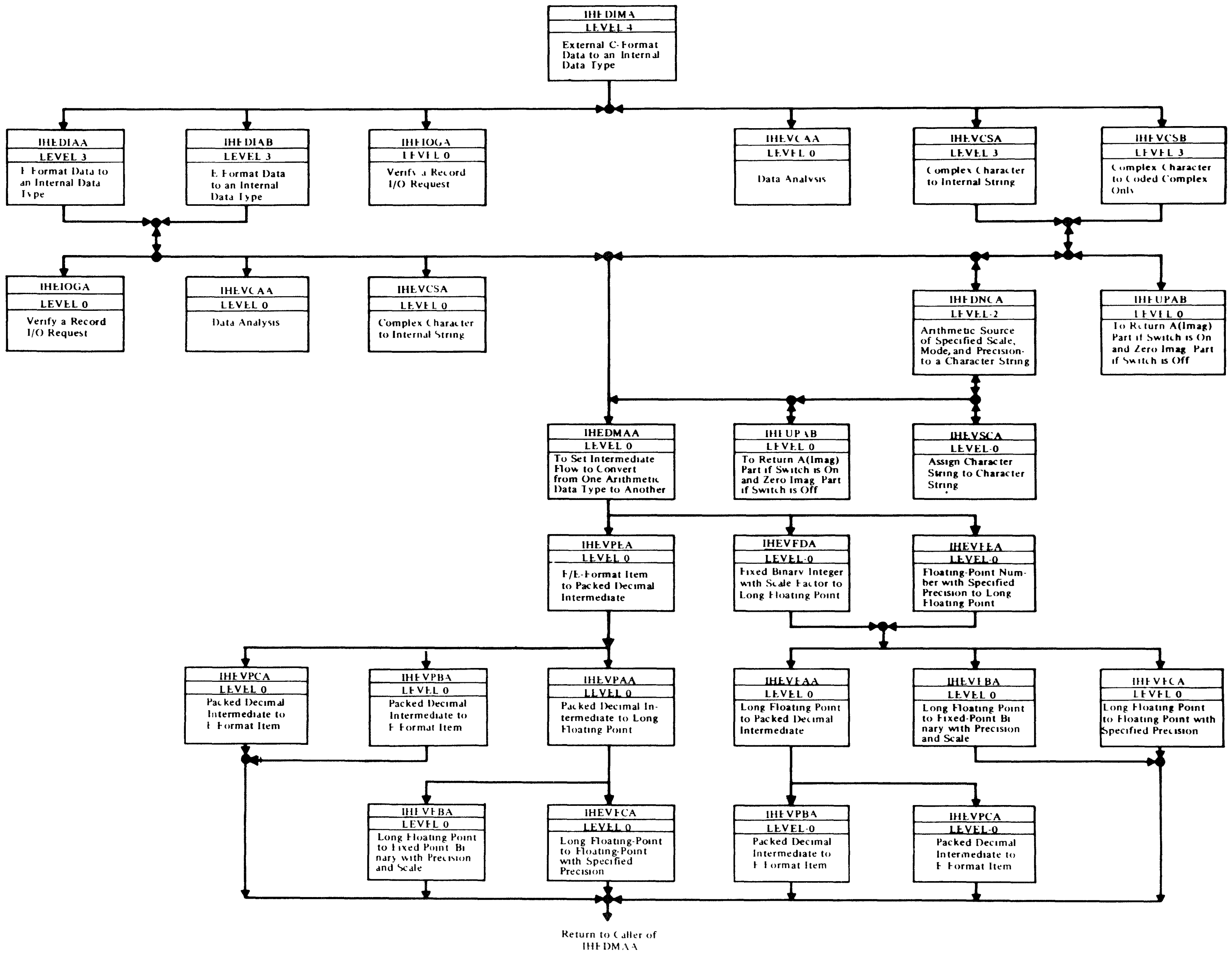

IHEDIMA LEVEL 4 External C-Format Data to an Internal Data Type
IHEDIAA LEVEL 3 F Format Data to an Internal Data Type
IHEDIAB LEVEL 3 E Format Data to an Internal Data Type
IHEIOGA LEVEL 0 Verify a Record I/O Request
IHEVCAA LEVEL 0 Data Analysis
IHEVCSA LEVEL 3 Complex Character to Internal String
IHEVCSB LEVEL 3 Complex Character to Coded Complex Only
IHEIOGA LEVEL 0 Verify a Record I/O Request
IHEVCAA LEVEL 0 Data Analysis
IHEVCSA LEVEL 0 Complex Character to Internal String
IHEDNCA LEVEL 2 Arithmetic Source of Specified Scale, Mode, and Precision to a Character String
IHEUPAB LEVEL 0 To Return A(Imag) Part if Switch is On and Zero Imag Part if Switch is Off
IHEDMAA LEVEL 0 To Set Intermediate Flow to Convert from One Arithmetic Data Type to Another
IHEUPAB LEVEL 0 To Return A(Imag) Part if Switch is On and Zero Imag Part if Switch is Off
IHEVSCA LEVEL 0 Assign Character String to Character String
IHEVPEA LEVEL 0 F/E-Format Item to Packed Decimal Intermediate
IHEVFDA LEVEL 0 Fixed Binary Integer with Scale Factor to Long Floating Point
IHEVFEA LEVEL 0 Floating-Point Number with Specified Precision to Long Floating Point
IHEVPCA LEVEL 0 Packed Decimal Intermediate to E Format Item
IHEVPBA LEVEL 0 Packed Decimal Intermediate to F Format Item
IHEVPAA LEVEL 0 Packed Decimal Intermediate to Long Floating Point
IHEVFAA LEVEL 0 Long Floating Point to Packed Decimal Intermediate
IHEVFBA LEVEL 0 Long Floating Point to Fixed-Point Binary with Precision and Scale
IHEVFCA LEVEL 0 Long Floating Point to Floating Point with Specified Precision
IHEVFBA LEVEL 0 Long Floating Point to Fixed Point Binary with Precision and Scale
IHEVFCA LEVEL 0 Long Floating-Point to Floating-Point with Specified Precision
IHEVPBA LEVEL 0 Packed Decimal Intermediate to F Format Item
IHEVPCA LEVEL 0 Packed Decimal Intermediate to E Format Item
Return to Caller of IHEDMAA

Figure 4-7. Flow through Total Conversion Package (Page 1 of 4)

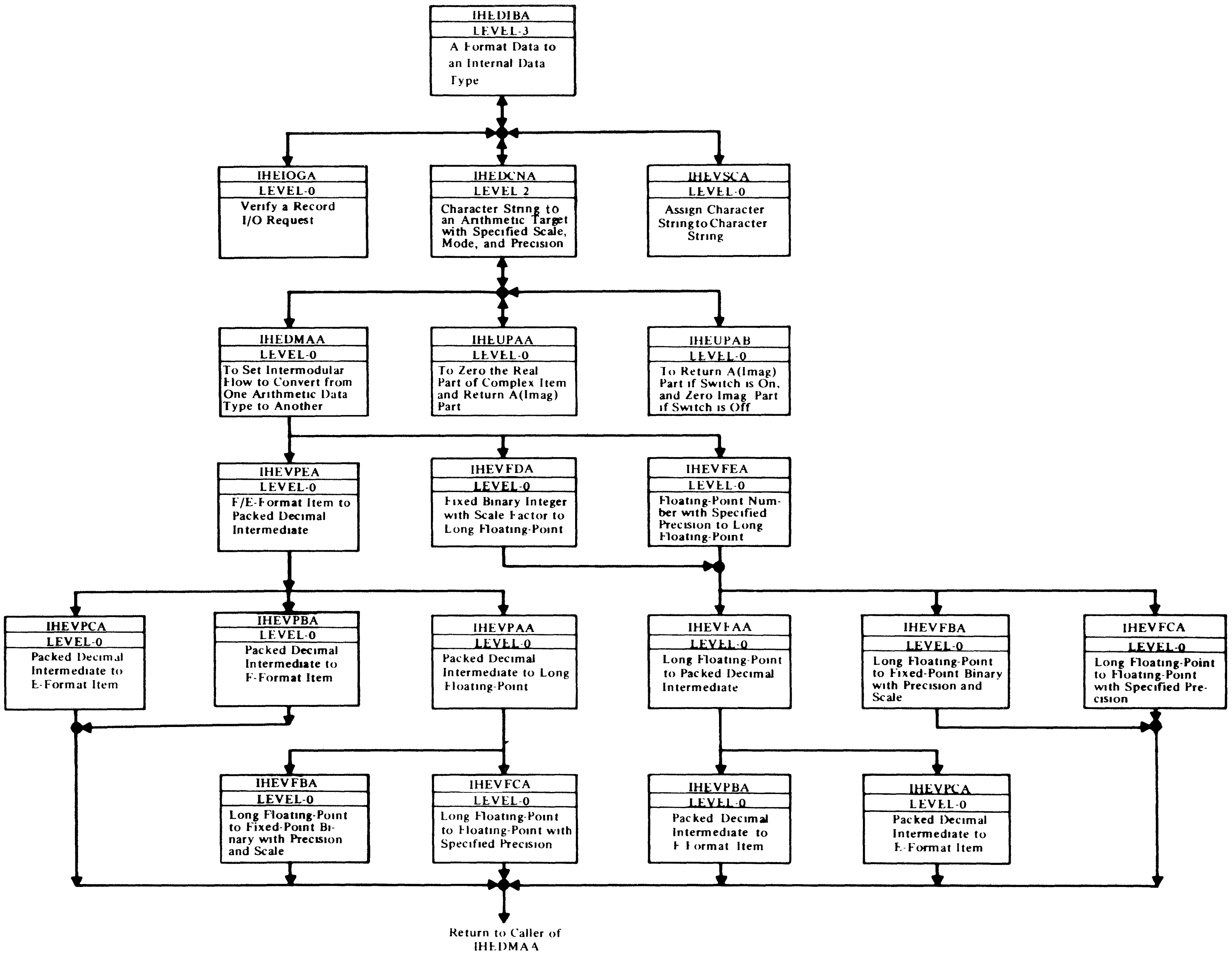

IHEDIBA
LEVEL-3
A Format Data to an Internal Data Type
IHEIOGA
LEVEL-0
Verify a Record I/O Request
IHEDCNA
LEVEL 2
Character String to an Arithmetic Target with Specified Scale, Mode, and Precision
IHEVSCA
LEVEL-0
Assign Character String to Character String
IHEDMAA
LEVEL-0
To Set Intermodular Flow to Convert from One Arithmetic Data Type to Another
IHEUPAA
LEVEL-0
To Zero the Real Part of Complex Item and Return A(Imag) Part
IHEUPAB
LEVEL-0
To Return A(Imag) Part if Switch is On, and Zero Imag Part if Switch is Off
IHEVPEA
LEVEL-0
F/E-Format Item to Packed Decimal Intermediate
IHEVFDA
LEVEL-0
Fixed Binary Integer with Scale Factor to Long Floating-Point
IHEVFEA
LEVEL-0
Floating-Point Number with Specified Precision to Long Floating-Point
IHEVPCA
LEVEL-0
Packed Decimal Intermediate to E-Format Item
IHEVPBA
LEVEL-0
Packed Decimal Intermediate to F-Format Item
IHEVPAA
LEVEL-0
Packed Decimal Intermediate to Long Floating-Point
IHEVFAA
LEVEL-0
Long Floating-Point to Packed Decimal Intermediate
IHEVFBA
LEVEL-0
Long Floating-Point to Fixed-Point Binary with Precision and Scale
IHEVFCA
LEVEL-0
Long Floating-Point to Floating-Point with Specified Precision
IHEVFBA
LEVEL-0
Long Floating-Point to Fixed-Point Binary with Precision and Scale
IHEVFCA
LEVEL-0
Long Floating-Point to Floating-Point with Specified Precision
IHEVPBA
LEVEL-0
Packed Decimal Intermediate to F Format Item
IHEVPCA
LEVEL-0
Packed Decimal Intermediate to E-Format Item
Return to Caller of IHEDMAA

Figure 4-7. Flow through Total Conversion Package (Page 2 of 4)

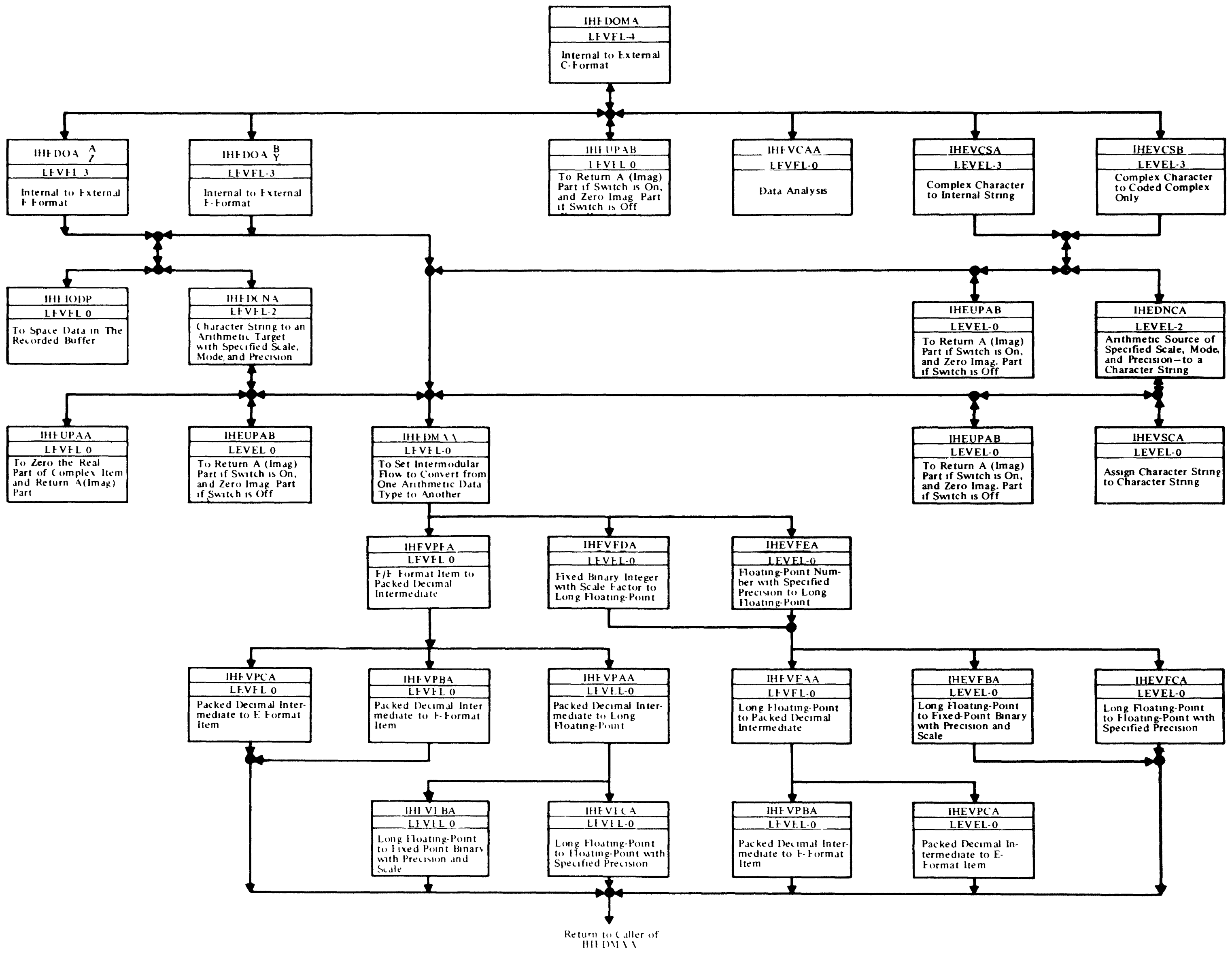

IHEDOMA
LEVEL-4
Internal to External C-Format
IHEDOA A/Z
LEVEL-3
Internal to External F-Format
IHEDOA B/Y
LEVEL-3
Internal to External E-Format
IHEUPAB
LEVEL-0
To Return A (Imag) Part if Switch is On, and Zero Imag. Part if Switch is Off
IHEVCAA
LEVEL-0
Data Analysis
IHEVCSA
LEVEL-3
Complex Character to Internal String
IHEVCSB
LEVEL-3
Complex Character to Coded Complex Only
IHEIODP
LEVEL-0
To Space Data in The Recorded Buffer
IHEDCNA
LEVEL-2
Character String to an Arithmetic Target with Specified Scale, Mode, and Precision
IHEUPAB
LEVEL-0
To Return A (Imag) Part if Switch is On, and Zero Imag. Part if Switch is Off
IHEDNCA
LEVEL-2
Arithmetic Source of Specified Scale, Mode, and Precision—to a Character String
IHEUPAA
LEVEL-0
To Zero the Real Part of Complex Item and Return A(Imag) Part
IHEUPAB
LEVEL-0
To Return A (Imag) Part if Switch is On, and Zero Imag. Part if Switch is Off
IHEDMAA
LEVEL-0
To Set Intermodular Flow to Convert from One Arithmetic Data Type to Another
IHEUPAB
LEVEL-0
To Return A (Imag) Part if Switch is On, and Zero Imag. Part if Switch is Off
IHEVSCA
LEVEL-0
Assign Character String to Character String
IHEVPEA
LEVEL-0
F/E Format Item to Packed Decimal Intermediate
IHEVFDA
LEVEL-0
Fixed Binary Integer with Scale Factor to Long Floating-Point
IHEVFEA
LEVEL-0
Floating-Point Number with Specified Precision to Long Floating-Point
IHEVPCA
LEVEL-0
Packed Decimal Intermediate to E Format Item
IHEVPBA
LEVEL-0
Packed Decimal Intermediate to F-Format Item
IHEVPAA
LEVEL-0
Packed Decimal Intermediate to Long Floating-Point
IHEVFAA
LEVEL-0
Long Floating-Point to Packed Decimal Intermediate
IHEVFBA
LEVEL-0
Long Floating-Point to Fixed-Point Binary with Precision and Scale
IHEVFCA
LEVEL-0
Long Floating-Point to Floating-Point with Specified Precision
IHEVFBA
LEVEL-0
Long Floating-Point to Fixed Point Binary with Precision and Scale
IHEVFCA
LEVEL-0
Long Floating-Point to Floating-Point with Specified Precision
IHEVPBA
LEVEL-0
Packed Decimal Intermediate to F-Format Item
IHEVPCA
LEVEL-0
Packed Decimal Intermediate to E-Format Item
Return to Caller of IHEDMAA

Figure 4-7. Flow through Total Conversion Package (Page 3 of 4)

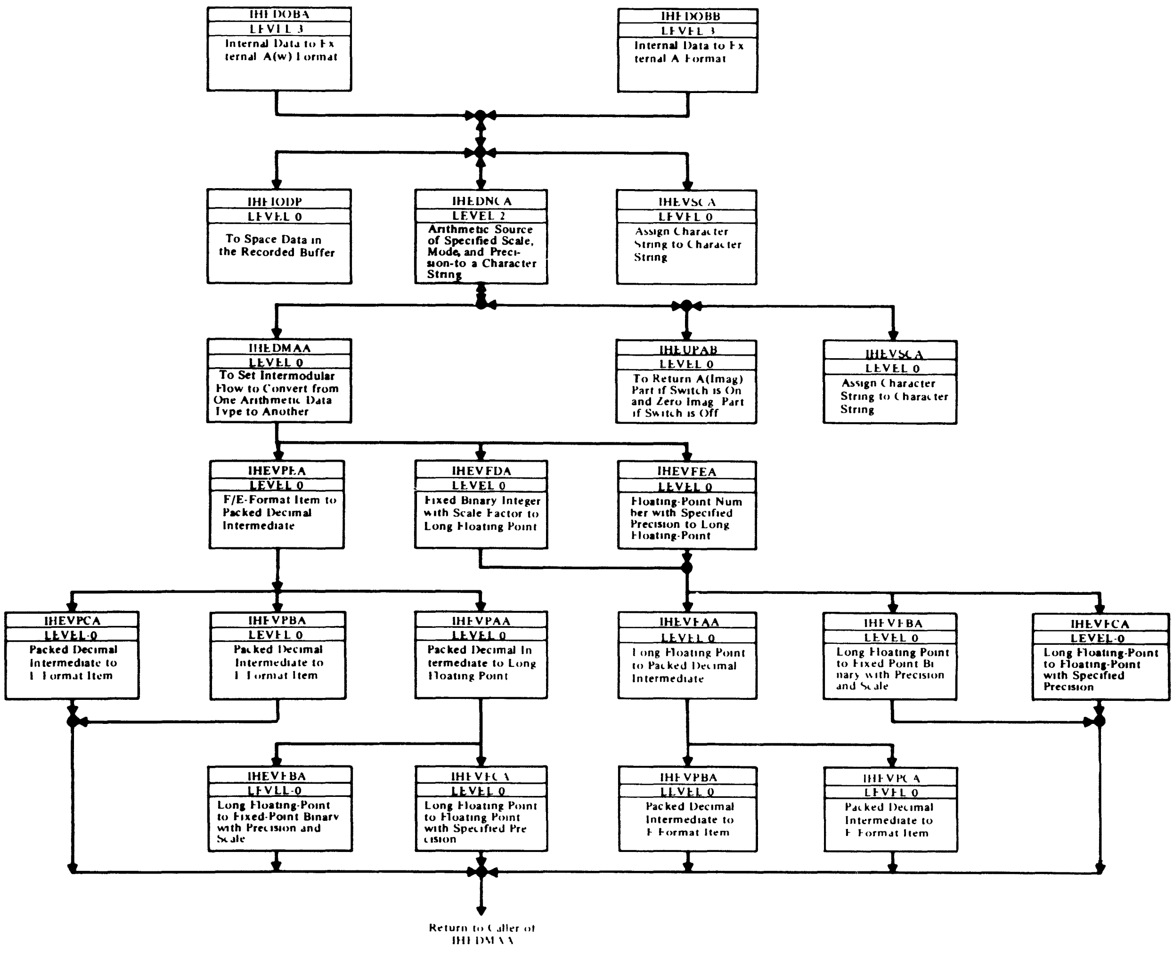

IHEDOBA
LEVEL 3
Internal Data to Ex ternal A(w) Format
IHEDOBB
LEVEL 3
Internal Data to Ex ternal A Format
IHEIODP
LEVEL 0
To Space Data in the Recorded Buffer
IHEDNCA
LEVEL 2
Arithmetic Source of Specified Scale, Mode, and Preci- sion-to a Character String
IHEVSCA
LEVEL 0
Assign Character String to Character String
IHEDMAA
LEVEL 0
To Set Intermodular Flow to Convert from One Arithmetic Data Type to Another
IHEUPAB
LEVEL 0
To Return A(Imag) Part if Switch is On and Zero Imag Part if Switch is Off
IHEVSCA
LEVEL 0
Assign Character String to Character String
IHEVPEA
LEVEL 0
F/E-Format Item to Packed Decimal Intermediate
IHEVFDA
LEVEL 0
Fixed Binary Integer with Scale Factor to Long Floating Point
IHEVFEA
LEVEL 0
Floating-Point Num ber with Specified Precision to Long Floating-Point
IHEVPCA
LEVEL-0
Packed Decimal Intermediate to E Format Item
IHEVPBA
LEVEL 0
Packed Decimal Intermediate to F Format Item
IHEVPAA
LEVEL 0
Packed Decimal In termediate to Long Floating Point
IHEVFAA
LEVEL 0
Long Floating Point to Packed Decimal Intermediate
IHEVFBA
LEVEL 0
Long Floating Point to Fixed Point Bi nary with Precision and Scale
IHEVFCA
LEVEL-0
Long Floating-Point to Floating-Point with Specified Precision
IHEVFBA
LEVEL-0
Long Floating-Point to Fixed-Point Binary with Precision and Scale
IHEVFCA
LEVEL 0
Long Floating Point to Floating Point with Specified Pre cision
IHEVPBA
LEVEL 0
Packed Decimal Intermediate to F Format Item
IHEVPCA
LEVEL 0
Packed Decimal Intermediate to E Format Item
Return to Caller of IHEDMAA

Figure 4-7. Flow through Total Conversion Package (Page 4 of 4)

STRING MANIPULATION PACKAGE (SIMP)

The String Manipulation Package contains modules for handling character strings. The modules are listed below.

CALL/360-OS PL/I Operation	Character String
Compare	IHECSC
Assign	IHECSM
Character String SUBSTR	IHECSS

The operation of comparison of two strings is supported by IHECSC. Two strings are compared for equality and a condition code returned, differentiating the lower-valued string from the higher-valued one.

The general design of the String Manipulation Package assumes that complete evaluation of the rightmost side of an assignment statement occurs before the assignment. There is usually an intermediate stage, in which a partial result is placed in a field acting as a temporary result field.

When an aggregate of strings is processed, indexing is handled by compiled code, which passes the individual string elements to the required library routine.

Some concepts that are basic to string manipulation in CALL/360-OS PL/I follow:

1. Character strings are allowed.

2. Internal to IBM System/360 computers, character strings are always byte-aligned. One byte represents one character in the string. There is one type of character string.

   Fixed Length - The number of characters in the string never varies, and the field width (in bytes) that the string occupies is equal to the number of characters in the string at all times.

3. The description of the string representation itself is stored in a data element descriptor (DED) for the string.

4. The description of the field the string occupies is stored in a string dope vector (SDV) for the string. Information contained in the SDV includes:

   a. The byte address of the beginning of the string
   b. The maximum length of the string, m

Linkage to the string routines is by external and internal calling sequences. String information is passed to the library by means of string dope vectors. All string lengths supplied in string dope vectors are assumed to be valid and nonnegative. Certain length fields in the string dope vector are ignored by these string routines and need not be completed by the caller.

The CALL/360-OS PL/I Language Reference Manual lists string built-in function names under "String Functions" in Appendix A.

Function Name	Function
CHAR	Converts an argument to a character string of specified length.
SUBSTR(S,i,j)	Creates a substring of string S, starting at position i and extending to j position (to position i+j-1).

CHAR and SUBSTR are more efficiently implemented by compiled code than by calls to the library.

ARITHMETIC FUNCTION PACKAGE (AFUNC)

Library arithmetic routines support all arithmetic generic functions and operations for which the compiler does not produce in-line code. Linkage between the object code and the library arithmetic routines is performed by means of external (all functions) and internal (all operators) standard calling sequences.

Where the functions and operators are applied to aggregates of data, indexing is handled by compiled code which passes the individual elements to the required library routine. Where evaluation or conversion of an argument is necessary, it is done before the library is called.

Fixed-point data often require data element descriptors (DED) to convey information about precision. The calling sequence sets up addresses of arguments in a standard order with each fixed-point item followed by its DED. If the DED is not needed by a routine, it need not be set up.

Floating-point arguments are assumed to be normalized in aligned full-word or doubleword fields for short or long precision respectively; the results returned are similarly normalized.

Complex arguments are assumed to have real and imaginary parts stored next to each other in that order, so that the address of a real part suffices for both of them. Both parts are also covered by the same DED.

Definitions

Some of the definitions required to understand arithmetic function evaluation in CALL/360-OS PL/I are discussed below:

1. There are four internal representations of *floating-point data*:

   a. Real short float
   b. Real long float
   c. Complex short float
   d. Complex long float

   These representations are defined under "Mathematical Function Package (MFUNC)."

2. There two *fixed-point data* representations:

   a. Real fixed
   b. Complex fixed

   These representations are defined as follows:

   Real fixed representations occupy four bytes as shown:

```
r----------------------------------,
| |                                |
|S|             ...                |
| |                                |
| |                                |
L----------------------------------J
 0                              31
```

   The binary digits are right-adjusted, and the four-byte field is always word-aligned. Halfword (2 bytes) real fixed binary representation is allowed in the IBM System/360, but it is not supported by any of the LIBCOMP routines.

   Complex fixed representations are two real fixed-point representations, one representing the real part and one representing the complex part. The two real representations must have the same precision and scale factor. They must be contiguous in storage and the real part must precede the imaginary part; thus, a pointer to the real part is needed to obtain both parts.

3. The description of an arithmetic representation is stored in a *data element descriptor* (DED) for the arithmetic datum. The attributes specified in the DED are:

   a. Real or complex (mode).

   b. Fixed or float (scale).

c. Short or long (applicable to floating-point representations only; fixed-point representations are always short, that is, four bytes long).

d. Precision (p) (which is the total number of digits in a fixed-point representation and the total number of digits in the fractional part of a floating-point representation).

e. Scale factor (q) (which is applicable only to fixed-point representations and is the number of digits to the right of the implied radix point).

4. In functions where two arithmetic representations are involved and conversion to the highest characteristics is specified, the following attributes are the highest of their pair:

   a. Complex (over real)
   b. Float (over fixed)

Module Description

The CALL/360-OS PL/I Language Reference Manual lists the following built-in function names under "Arithmetic Built-In Functions" in Appendix A.

Function Name	Arguments and Function Value
SIGN(x)	Given a real argument, returns an integer indicating the sign of the argument (+1 for positive, -1 for negative, and 0 for zero).
FLOOR(x)	Given a real argument, returns the largest of the group of integers not exceeding the argument.
CEIL(x)	Given a real argument, returns the smallest of the group of integers not exceeded by the argument.
TRUNC(x)	Given a real argument, returns the integer part (FLOOR(x) for arguments ≥ 0, CEIL(x) for arguments < 0).
MOD(x1,x2)	Given two real arguments, returns the positive remainder left after obtaining the integer quotient (x1/x2). The scale is the highest characteristic of x1 and x2.
CONJ(z)	Given a complex argument, returns the conjugate of the argument.
COMPLEX (x,y)	Mode conversion. Given two real arguments, returns a complex representation. The scale and precision of the complex representation are the highest characteristics of x and y.
REAL(z)	Mode conversion. Given a complex argument, returns the real part. The scale and precision are unchanged.

Function Name	Arguments and Function Value
IMAG(z)	Mode conversion. Given a complex argument, returns the imaginary part. The scale and precision are unchanged.
ABS(t)	Given a real argument, returns its positive value. Given a complex argument (for example, x+yI), returns its positive magnitude: $\left(\sqrt{x^2 + y^2}\right)$ The mode and scale of the result are the same as for the argument.
MAX(x1,x2,...,xn)	Given a list of real arguments, returns the value of the maximum argument. The scale is the highest characteristic of all the arguments in the list.
MIN(x1,x2,...,xn)	Same as for MAX, except returns the value of the minimum argument.

The following built-in functions are more efficiently supported by the compiled code than by a library call:

SIGN	CONJ
FLOOR	COMPLEX
CEIL	REAL
TRUNC	IMAG
MOD	ABS(real argument)

The following built-in functions in the area of arithmetic function evaluation are supported by AFUNC:

ABS(complex argument)
MAX
MIN

These built-in functions may specify any expression as an argument. However, the AFUNC modules which support these built-in functions accept only arithmetic scalar values having the following characteristics:

1. Real fixed arguments must be stored in four bytes, word-aligned, with $0 < p \leq 9$.

2. Floating arguments can be short or long, and must be normalized (same as arguments for MFUNC modules).

3. Complex arguments have both parts stored contiguously and the same DED describes both parts.

The compiler is responsible for:

1. Evaluating the expression argument presented by the source program's built-in function and converting the resulting scalar value to one of the representations that will be accepted by the AFUNC module.

2. Choosing, from the several entry points which support a given built-in function, the proper AFUNC module entry point to invoke. This choice is strictly dependent on the representation of the value of the input argument(s).

3. Constructing a list of arguments to present to the AFUNC module. In general, if B is the number of arguments presented by the built-in function, the list presented to AFUNC module will contain 2B+2 arguments if DED is needed, or 1B+1 if no DED is needed.

Following is a list of the AFUNC entry points which support ABS, MAX, and MIN. The built-in function ABS is supported in AFUNC only for complex arguments. The built-in functions MAX and MIN are supported in AFUNC for real arguments only.

Function Name	Entry Point	Argument Attributes
ABS(z)	IHEABG0	Fixed (complex)
	IHEABT0	Short float (complex)
	IHEABM0	Long float (complex)
MAX(x1,x2,...,xn)	IHEMXF0	Fixed (real)
	IHEMXS0	Short float (real)
	IHEMXL0	Long float (real)
MIN(x1,x2,...,xn)	IHEMNF0	Fixed (real)
	IHEMNS0	Short float (real)
	IHEMNL0	Long float (real)

In addition to supporting the built-in functions listed in the CALL/360-OS PL/I Language Reference Manual, AFUNC supports four often-used arithmetic operations. They are:

1.	Integer exponentiation	For both real and complex operands
2.	General exponentiation	
3.	Multiplication	For complex operands only
4.	Division	

Integer exponentiation, general exponentiation, and multiplication and division are supported as follows:

Operation	System Name	Argument Attributes
	IHEXIFI	Real fixed
	IHEXISI	Real short float
Integer exponentiation (t**n)	IHEXILI	Real long float
	IHEXIGI	Complex fixed
	IHEXITI	Complex short float
	IHEXIMI	Complex long float
General exponentiation (t1**t2)	IHEXISF	Real short (float)
	IHEXILF	Real long (float)
	IHEXITF	Complex short (float)
	IHEXIMF	Complex long (float)
Multiplication (z1*z2)	IHEMZG0	Fixed binary (complex)
	IHEMZT0	Short float (complex)
	IHEMZM0	Long float (complex)
Division (z1/z2)	IHEDZG0	Fixed bonary (complex)
	IHEDZT0	Short float (complex)
	IHEDZM0	Long float (complex)

Summary

The library arithmetic modules are summarized in Figures 4-8 and 4-9.

Operation	Binary Fixed	Short Float	Long Float
Real Operations			
Integer exponentiation: x**n	IHEXIB	IHEXIS	IHEXIL
General exponentiation: x**y	--	IHEXXS	IHEXXL
Shift-and-assign, shift-and-load	--	--	--
Complex Operations			
Multiplication/Division: z1*z2, z1/z2	IHEMZU	--	--
Multiplication: z1*z2	--	IHEMZW	IHEMZZ
Division: z1/z2	--	IHEDZW	IHEDZZ
Integer exponentiation: z**n	IHEXIU	IHEXIW	IHEXIZ
General exponentiation: z1**z2	--	IHEXXW	IHEXXZ

Figure 4-8. Arithmetic Operations

Function	Binary Fixed	Short Float	Long Float
Real Arguments			
MAX, MIN	IHEMXB	IHEMXS	IHEMXL
ADD	--	--	--
Complex Arguments			
ADD	--	--	--
MULTIPLY	IHEMZU	IHEMZW	IHEMZZ
DIVIDE	IHEMZU	IHEDZW	IHEDZZ
ABS	IHEABU	IHEABW	IHEABZ

Figure 4-9. Arithmetic Functions

Figures 4-10 through 4-12 show the modular interaction within the library and indicate the modular level assignments.

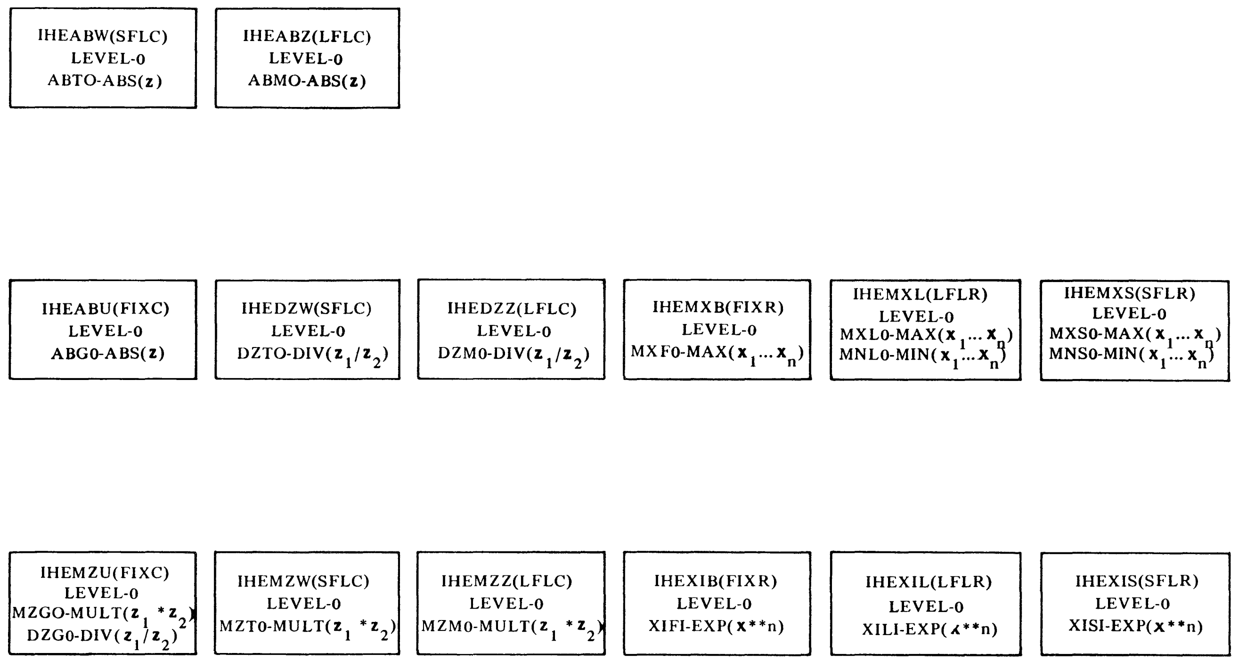

IHEABW(SFLC) LEVEL-0 ABTO-ABS(z)
IHEABZ(LFLC) LEVEL-0 ABMO-ABS(z)
IHEABU(FIXC) LEVEL-0 ABG0-ABS(z)
IHEDZW(SFLC) LEVEL-0 DZTO-DIV(z1/z2)
IHEDZZ(LFLC) LEVEL-0 DZM0-DIV(z1/z2)
IHEMXB(FIXR) LEVEL-0 MXF0-MAX(x1...xn)
IHEMXL(LFLR) LEVEL-0 MXL0-MAX(x1...xn) MNL0-MIN(x1...xn)
IHEMXS(SFLR) LEVEL-0 MXS0-MAX(x1...xn) MNS0-MIN(x1...xn)
IHEMZU(FIXC) LEVEL-0 MZGO-MULT(z1*z2) DZG0-DIV(z1/z2)
IHEMZW(SFLC) LEVEL-0 MZT0-MULT(z1*z2)
IHEMZZ(LFLC) LEVEL-0 MZM0-MULT(z1*z2)
IHEXIB(FIXR) LEVEL-0 XIFI-EXP(x**n)
IHEXIL(LFLR) LEVEL-0 XILI-EXP(x**n)
IHEXIS(SFLR) LEVEL-0 XISI-EXP(x**n)

Figure 4-10. AFUNC Level 0

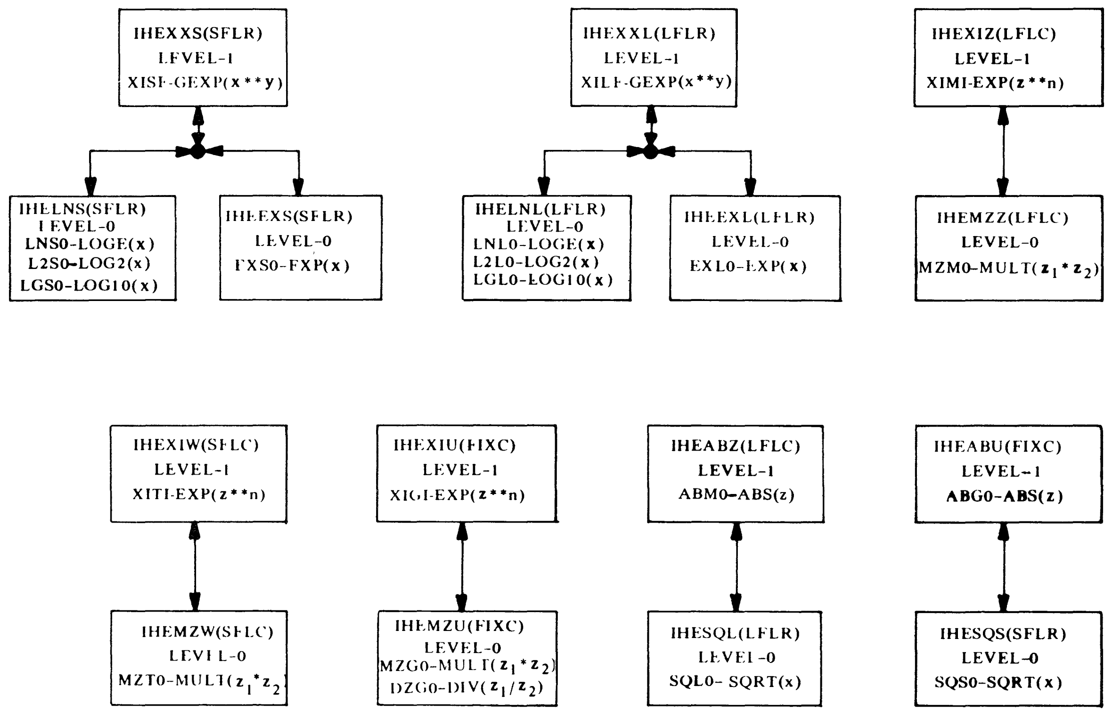
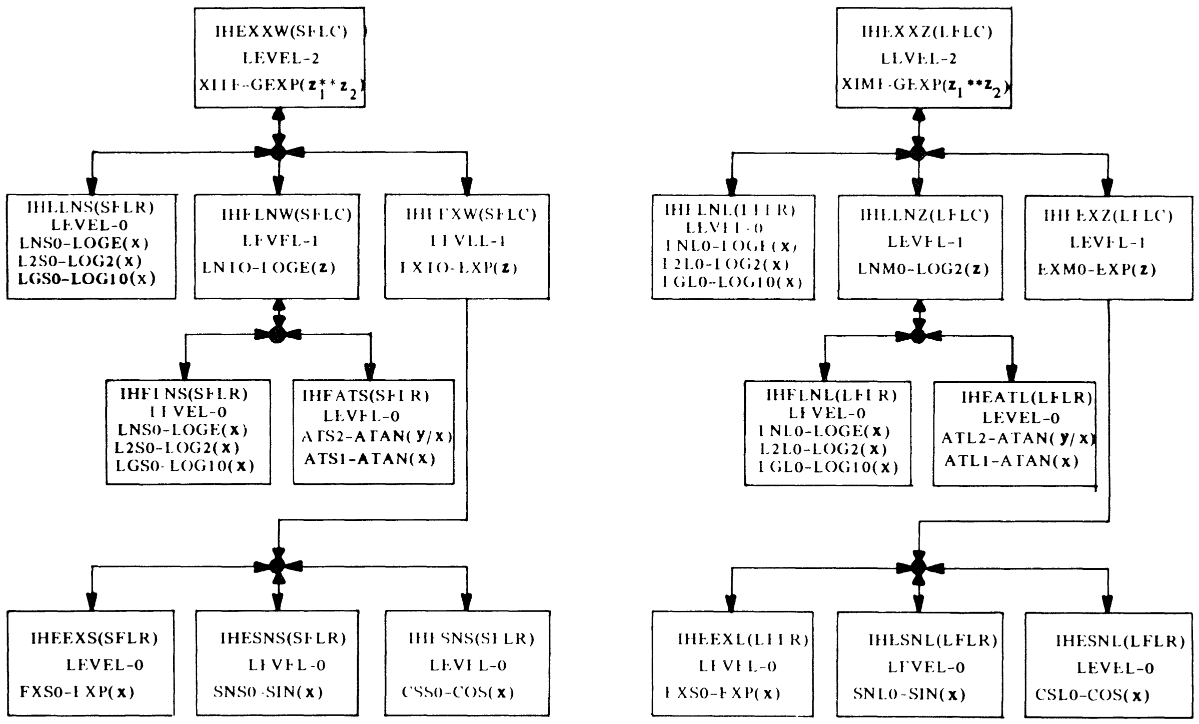

IHEXXS(SFLR) LEVEL-1 XISI-GEXP(x**y)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x) L2S0-LOG2(x) LGS0-LOG10(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHEXXL(LFLR) LEVEL-1 XILI-GEXP(x**y)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x) L2L0-LOG2(x) LGL0-LOG10(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)
IHEXIZ(LFLC) LEVEL-1 XIMI-EXP(z**n)
IHEMZZ(LFLC) LEVEL-0 MZM0-MULT(z1*z2)
IHEXIW(SFLC) LEVEL-1 XITI-EXP(z**n)
IHEMZW(SFLC) LEVEL-0 MZT0-MULT(z1*z2)
IHEXIU(FIXC) LEVEL-1 XIGI-EXP(z**n)
IHEMZU(FIXC) LEVEL-0 MZG0-MULT(z1*z2) DZG0-DIV(z1/z2)
IHEABZ(LFLC) LEVEL-1 ABM0-ABS(z)
IHESQL(LFLR) LEVEL-0 SQL0-SQRT(x)
IHEABU(FIXC) LEVEL-1 ABG0-ABS(z)
IHESQS(SFLR) LEVEL-0 SQS0-SQRT(x)

Figure 4-11. AFUNC Level 1



IHEXXW(SFLC) LEVEL-2 XIIE-GEXP(z1**z2)
IHEXXZ(LFLC) LEVEL-2 XIMF-GEXP(z1**z2)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x) L2S0-LOG2(x) LGS0-LOG10(x)
IHELNW(SFLC) LEVEL-1 LNI0-LOGE(z)
IHEEXW(SFLC) LEVEL-1 EXI0-EXP(z)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x) L2L0-LOG2(x) LGL0-LOG10(x)
IHELNZ(LFLC) LEVEL-1 LNM0-LOG2(z)
IHEEXZ(LFLC) LEVEL-1 EXM0-EXP(z)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x) L2S0-LOG2(x) LGS0-LOG10(x)
IHEATS(SFLR) LEVEL-0 ATS2-ATAN(y/x) ATS1-ATAN(x)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x) L2L0-LOG2(x) LGL0-LOG10(x)
IHEATL(LFLR) LEVEL-0 ATL2-ATAN(y/x) ATL1-ATAN(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHESNS(SFLR) LEVEL-0 SNS0-SIN(x)
IHESNS(SFLR) LEVEL-0 CSS0-COS(x)
IHEEXL(LFLR) LEVEL-0 EXS0-EXP(x)
IHESNL(LFLR) LEVEL-0 SNL0-SIN(x)
IHESNL(LFLR) LEVEL-0 CSL0-COS(x)

Figure 4-12. AFUNC Level 2

MATHEMATICAL FUNCTION PACKAGE (MFUNC)

The library supports all floating-point arithmetic generic functions, and has separate routines for short- and long-precision real arguments, and also for short- and long-precision complex arguments where these are admissible. Linkage between the main program and the library mathematical routines is performed by means of the external standard calling sequence.

The calling sequence generated in compiled code is the same as that required for passing the same arguments to a CALL/360-OS PL/I procedure. Therefore, the names of any of the floating-point arithmetic generic functions can be passed as arguments between procedures, according to the normal rules for entry names.

Where functions are applied to aggregates of data, indexing is handled by compiled code, which passes the individual elements to the required library routine. Where evaluation or conversion of an argument is necessary, it is carried out before the library routine is called.

The arguments are assumed to be normalized in aligned fullword or doubleword fields for short or long precision respectively; the results returned are normalized similarly.

Complex arguments are assumed to have real and imaginary parts stored contiguously in that order, so that the address of the real part suffices for both of them.

Source fields may also be used as target fields in all cases where this is not explicitly forbidden in a routine description (see next section).

Definitions

Some definitions which are basic to understanding mathematical function evaluation in CALL/360-OS PL/I follow.

1. The floating-point representation of a numeric value in IBM System/360 computers occupies a fixed-length field, but this field may be either of two lengths.

   Short floating-point representations occupy four bytes as shown:

```
┌──────────────────────────────────┐
│ │Exponent│                       │
│S│  Part  │    Fractional Part    │
│ │(Powers │    (6 Hex Digits)     │
│ │  of 16)│                       │
└──────────────────────────────────┘
0         7 8                     31
```

   The fractional part value represents six decimal digits (short float decimal).

   Long floating-point representations occupy eight bytes as shown:

```
┌──────────────────────────────────────────────────────────┐
│ │Exponent│                                               │
│S│  Part  │              Fractional Part                  │
│ │(Powers │              (14 Hex Digits)                  │
│ │  of 16)│                                               │
└──────────────────────────────────────────────────────────┘
0         7 8                                             63
```

   The fractional part value represents a maximum of sixteen decimal digits (long float decimal).

2. Both short and long floating-point representations are normalized. This means that the leftmost hexadecimal digit (leftmost four bits) of the fractional part is nonzero.

3. The representation of a complex floating-point number is two real floating-point representations, one representing the real part and the other representing the imaginary part. The two real representations must both be long or short. They must be contiguous in storage and the real part must precede the imaginary part. Thus, a pointer to the real part is needed to obtain both parts.

4. Most mathematical functions are multivalued for complex arguments, but a principal value can be chosen as the value returned from the function.

Module Description

The CALL/360-OS PL/I Language Reference Manual lists mathematical built-in function names under "Mathematical Built-In Functions." All of these built-in function names are supported by the LIBCOMP Mathematical Function Package (MFUNC). All of the modules in MFUNC are devoted to supporting them.

The specification of a built-in function name may be supported by multiple MFUNC entry points. The entry point used in a particular case depends strictly upon the mode and length attributes of the normalized floating-point scalar argument that is presented to the module.

Four of the built-in function names (LOG2, LOG10, ERF, and ATAN with two arguments) are supported only for real arguments, so there are two MFUNC entry points for each of these names: one accepts a real short argument and the other accepts a real long argument. The other names (EXP, LOG, SQRT, ATAN, ATANH, TAN, TANH, SIN, SINH, COS, and COSH) are supported for both real and complex arguments, so there are four MFUNC entry points for each of these names: one each for the acceptance of real short, real long, complex short, and complex long arguments.

The compiler is responsible for knowing the mode and length attributes of the input argument and calling the proper MFUNC entry point accordingly. Figure 4-13 shows the relationship between built-in function names and MFUNC entry points for each of the argument types. The following symbol definitions apply:

x	=	real argument, or real part of complex argument
y	=	imaginary part of argument
z	=	complex argument
u	=	real part of complex returned value
v	=	imaginary part of complex returned value

All MFUNC routine entry points (with the exceptions of ATS4, ATL4, ATS2, and ATL2) require the same two arguments: a pointer to the normalized floating-point scalar input value and a pointer to the field in which to put the returned value. The returned value is always a normalized floating-point scalar with the same mode and length as the input argument.

Built-In Function Name	Entry Point Name	Argument Type	Function
SIN(x)	SNS0 SNL0	real short real long	Sine of x x expressed in radians

SIN(z)	SNT0
	SNM0
		complex short
		complex long
			Principal value of sine of
			z=SIN(x) COSH(y)
			+ ICOS(x)SINH(y)
			x and y expressed in radians

SINH(x)	SHS0 SHL0	real short real long	Hyperbolic sine of x x expressed in radians
SINH(z)	SHT0 SHM0	complex short complex long	Principal value of hyperbolic sine of z=SINH(x)COS(y) + ICOSH(x)SIN(y) x and y expressed in radians
COS(x)	CSS0 CSL0	real short real long	Cosine of x x expressed in radians
COS(z)	CST0 CSM0	complex short complex long	Principal value of cosine of z = COS(x)COSH(y) - ISIN(x)SINH(y) x and y expressed in radians
COSH(x)	CHS0 CHL0	real short real long	Hyperbolic cosine of x x expressed in radians
COSH(z)	CHT0 CHM0	complex short complex long	Principal value of hyperbolic cosine of z = COSH(x)COS(y) + ISINH(x)SIN(y) x and y expressed in radians

Figure 4-13. Mathematical Built-In Functions (Page 1 of 3)

Built-In Function Name	Entry Point Name	Argument Type	Function
ATAN(x)	ATS1 ATL1	real short real long	Arctangent of x x expressed in radians $\left(-\frac{\pi}{2} \leqslant \text{returned value} \leqslant \frac{\pi}{2}\right)$
ATAN(z)	ATT0 ATM0	complex short complex long	Principal value of arctangent of z = (LOG(1+z)/(1-z))/2 (Error if z = +1 or -1)
ATANH(x)	AHS0 AHL0	real short real long	Hyperbolic arctangent of x (Error if ABS(x) ⩾ 1)
ATANH(z)	ATT0 ATM0	complex short complex long	Principal value of hyperbolic arctangent of z = IATANH(Iz) (Error if z = +I or -I)
TAN(x)	TNS0 TNL0	real short real long	Tangent of x x expressed in radians
TAN(z)	TNT0 TNM0	complex short complex long	Tangent of z x and y expressed in radians
TANH(x)	THS0 THL0	real short real long	Hyperbolic tangent of x x expressed in radians
TANH(z)	THT0 THM0	complex short complex long	Hyperbolic tangent of z x and y expressed in radians

Figure 4-13. Mathematical Built-In Functions (Page 2 of 3)

Built-In Function Name	Entry Point Name	Argument Type	Function
EXP(x)	EXSO EXLO	real short real long	e^x
EXP(z)	EXTO EXMO	complex short complex long	e^z
LOG(x)	LNSO LNLO	real short real long	$\text{Log}_e\ x$ (Error if $x \leqslant 0$)
LOG(z)	LNTO LNMO	complex short complex long	Principal value of $\log_e z$ $= -\pi < z \leqslant \pi$ (Error if $z = 0$)
SQRT(x)	SQSO SQLO	real short real long	Positive square root of x (Error if $x < 0$)
SQRT(z)	SQTO SQMO	complex short complex long	Principal value of z^2 $= u$, if $u > 0$, or $= v$, if $u = 0$ and $v \geqslant 0$
LOG2(x)	L2SO L2LO	real short real long	$\text{Log}_2\ x$ (Error if $x \leqslant 0$)
LOG10(x)	LGSO LGLO	real short real long	$\text{Log}_{10}\ x$ (Error if $x \leqslant 0$)
ERF(x)	EFSO EFLO	real short real long	Error function of x $= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$
ATAN (x1, x2)	ATS2 ATL2	real short real long	Arctangent of (x1/x2) x1 and x2 are in radians (Error if $x1 = 0$, $x2 = 0$)

Figure 4-13. Mathematical Built-In Functions (Page 3 of 3)

Summary

The library mathematical modules are summarized in Figures 4-14 and 4-15. These figures identify routines for real and complex arguments, respectively.

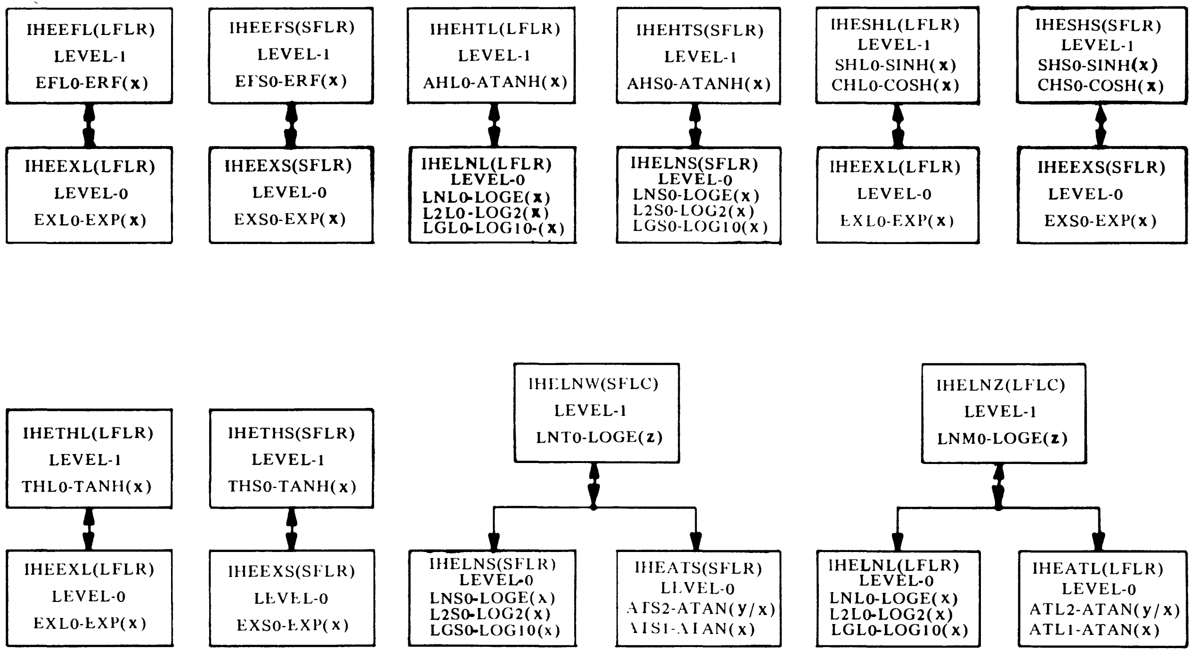
Function	Real Arguments	
	Short Float	Long Float
SQRT	IHESQS	IHESQL
EXP	IHEEXS	IHEEXL
LOG, LOG2, LOG10	IHELNS	IHELNL
SIN, COS	IHESNS	IHESNL
TAN	IHETNS	IHETNL
ATAN	IHEATS	IHEATL
SINH, COSH	IHESHS	IHESHL
TANH	IHETHS	IHETHL
ATANH	IHEHTS	IHEHTL
ERF, ERFC	IHEEFS	IHEEFL

Figure 4-14. 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 4-15. Mathematical Functions with Complex Arguments

Figures 4-16 through 4-18 show the modular interaction within the library and indicate the modular level assignments.


IHEEFL(LFLR) LEVEL-1 EFL0-ERF(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)
IHEEFS(SFLR) LEVEL-1 EFS0-ERF(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHEHTL(LFLR) LEVEL-1 AHL0-ATANH(x)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x) L2L0-LOG2(x) LGL0-LOG10-(x)
IHEHTS(SFLR) LEVEL-1 AHS0-ATANH(x)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x) L2S0-LOG2(x) LGS0-LOG10(x)
IHESHL(LFLR) LEVEL-1 SHL0-SINH(x) CHL0-COSH(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)
IHESHS(SFLR) LEVEL-1 SHS0-SINH(x) CHS0-COSH(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHETHL(LFLR) LEVEL-1 THL0-TANH(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)
IHETHS(SFLR) LEVEL-1 THS0-TANH(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHELNW(SFLC) LEVEL-1 LNT0-LOGE(z)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x) L2S0-LOG2(x) LGS0-LOG10(x)
IHEATS(SFLR) LEVEL-0 ATS2-ATAN(y/x) ATS1-ATAN(x)
IHELNZ(LFLC) LEVEL-1 LNM0-LOGE(z)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x) L2L0-LOG2(x) LGL0-LOG10(x)
IHEATL(LFLR) LEVEL-0 ATL2-ATAN(y/x) ATL1-ATAN(x)

Note:		
	x	denotes either a real argument or real part of a complex argument.
	y	denotes either a real argument or imaginary part of a complex argument.
	z	denotes a complex argument.
	(XXXX)	describes the arguments. The first letter specifies the precision (Long or Short) the next two, the scale (Float or Fixed), the last, the mode (Real or Complex).

Figure 4-16. MFUNC Level 0

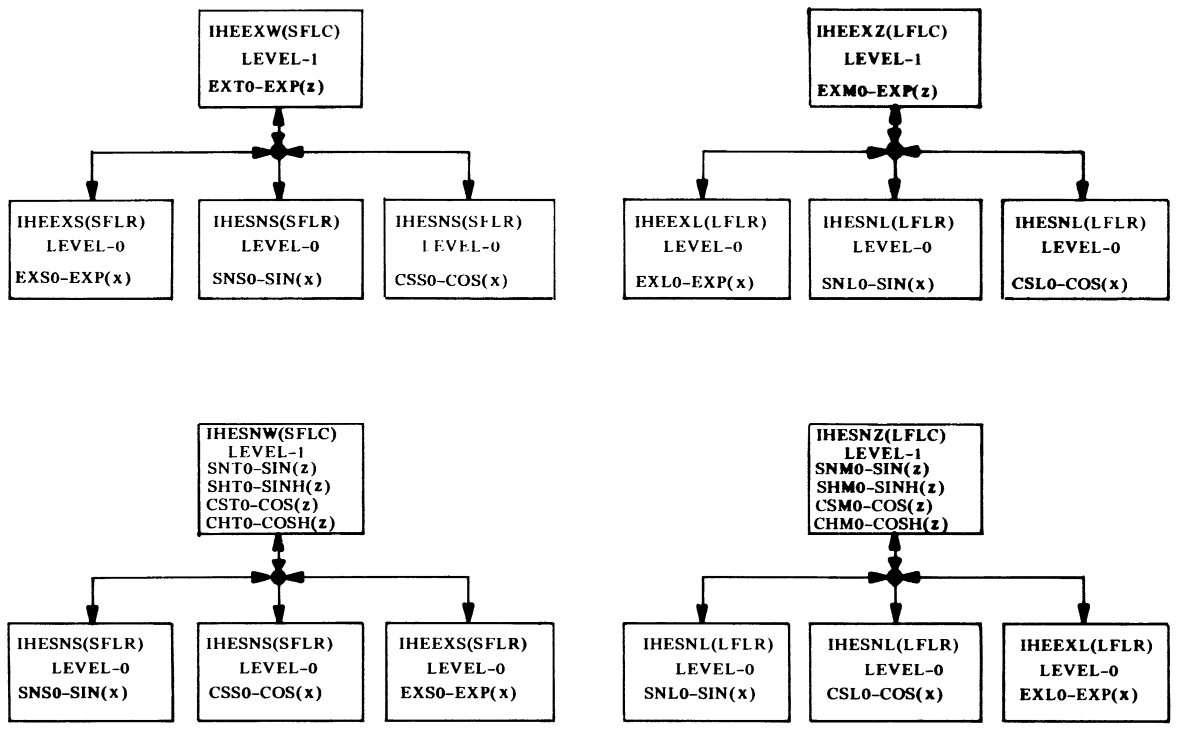

IHEEXW(SFLC) LEVEL-1 EXT0-EXP(z)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHESNS(SFLR) LEVEL-0 SNS0-SIN(x)
IHESNS(SFLR) LEVEL-0 CSS0-COS(x)
IHEEXZ(LFLC) LEVEL-1 EXM0-EXP(z)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)
IHESNL(LFLR) LEVEL-0 SNL0-SIN(x)
IHESNL(LFLR) LEVEL-0 CSL0-COS(x)
IHESNW(SFLC) LEVEL-1 SNT0-SIN(z) SHT0-SINH(z) CST0-COS(z) CHT0-COSH(z)
IHESNS(SFLR) LEVEL-0 SNS0-SIN(x)
IHESNS(SFLR) LEVEL-0 CSS0-COS(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHESNZ(LFLC) LEVEL-1 SNM0-SIN(z) SHM0-SINH(z) CSM0-COS(z) CHM0-COSH(z)
IHESNL(LFLR) LEVEL-0 SNL0-SIN(x)
IHESNL(LFLR) LEVEL-0 CSL0-COS(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)

Figure 4-17. MFUNC Level 1

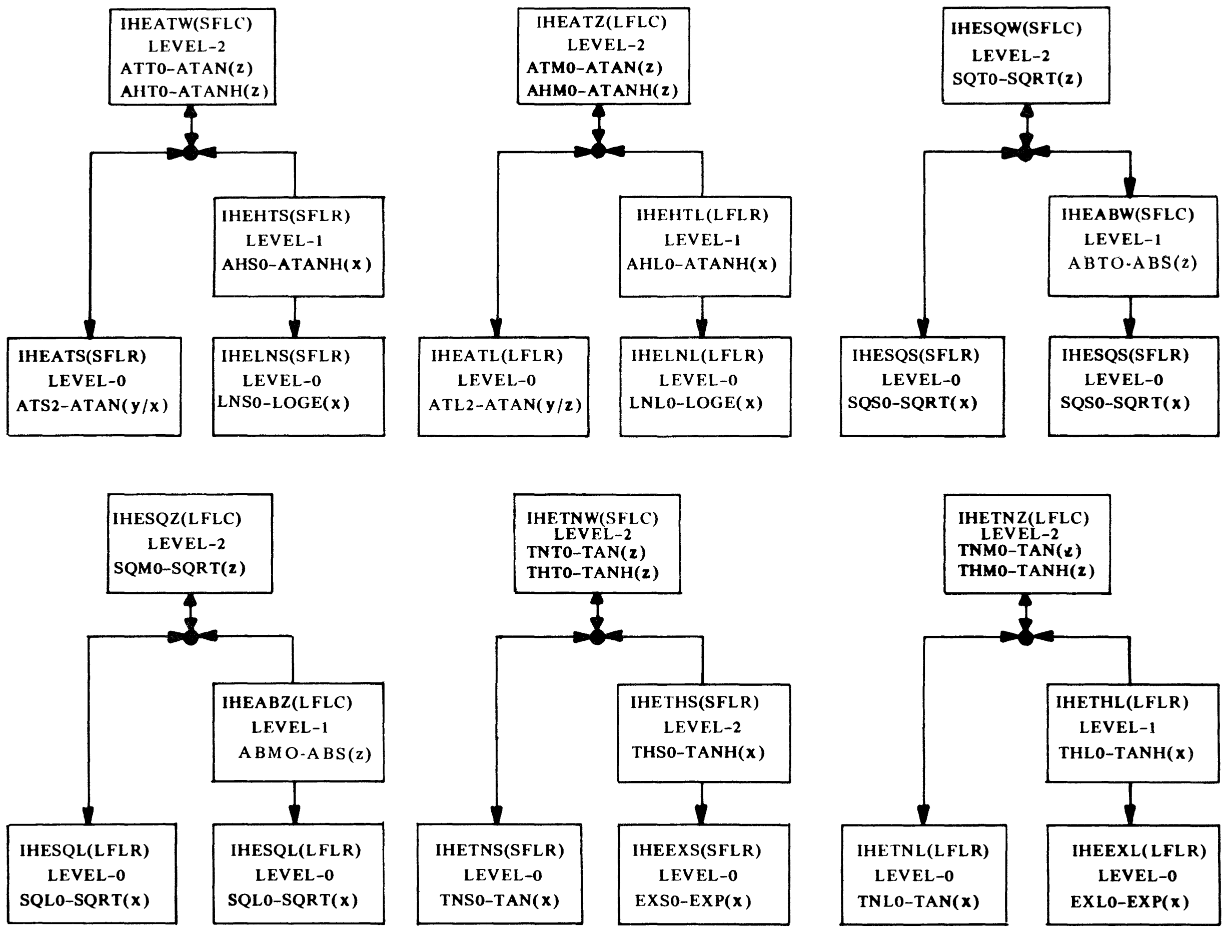

IHEATW(SFLC) LEVEL-2 ATT0-ATAN(z) AHT0-ATANH(z)
IHEHTS(SFLR) LEVEL-1 AHS0-ATANH(x)
IHEATS(SFLR) LEVEL-0 ATS2-ATAN(y/x)
IHELNS(SFLR) LEVEL-0 LNS0-LOGE(x)
IHEATZ(LFLC) LEVEL-2 ATM0-ATAN(z) AHM0-ATANH(z)
IHEHTL(LFLR) LEVEL-1 AHL0-ATANH(x)
IHEATL(LFLR) LEVEL-0 ATL2-ATAN(y/z)
IHELNL(LFLR) LEVEL-0 LNL0-LOGE(x)
IHESQW(SFLC) LEVEL-2 SQT0-SQRT(z)
IHEABW(SFLC) LEVEL-1 ABT0-ABS(z)
IHESQS(SFLR) LEVEL-0 SQS0-SQRT(x)
IHESQS(SFLR) LEVEL-0 SQS0-SQRT(x)
IHESQZ(LFLC) LEVEL-2 SQM0-SQRT(z)
IHEABZ(LFLC) LEVEL-1 ABM0-ABS(z)
IHESQL(LFLR) LEVEL-0 SQL0-SQRT(x)
IHESQL(LFLR) LEVEL-0 SQL0-SQRT(x)
IHETNW(SFLC) LEVEL-2 TNT0-TAN(z) THT0-TANH(z)
IHETHS(SFLR) LEVEL-2 THS0-TANH(x)
IHETNS(SFLR) LEVEL-0 TNS0-TAN(x)
IHEEXS(SFLR) LEVEL-0 EXS0-EXP(x)
IHETNZ(LFLC) LEVEL-2 TNM0-TAN(z) THM0-TANH(z)
IHETHL(LFLR) LEVEL-1 THL0-TANH(x)
IHETNL(LFLR) LEVEL-0 TNL0-TAN(x)
IHEEXL(LFLR) LEVEL-0 EXL0-EXP(x)

Figure 4-18. MFUNC Level 2

AGGREGATE MANIPULATION PACKAGE (AMP)

The library supports the array built-in functions SUM, PROD, and POLY, and also provides indexing routines for handling simple (that is, consecutively stored) and interleaved arrays.

Linkage between the main program and the library array function routines is performed by means of the external standard calling sequence. Calls to the indexing routines are made using the internal standard calling sequence.

The array routines accept array arguments and perform their own indexing, unlike other routines which require that indexing be handled by compiled code. Calls to conversion routines are included in the SUM, PROD, and POLY routines with fixed-point arguments, so that these arguments are converted to floating point as they are accessed.

Information about arrays is passed to the library routines in the form of array dope vectors (ADV); fixed-point arguments need an additional data element descriptor (DED). The number of dimensions of the array is contained in the ADV. No overlapping of source and target fields is permitted for any of these routines.

Definitions

Some definitions required to understand aggregate manipulation in CALL/360-OS PL/I follow:

1. Arrays are always stored in row-major order, that is, the rightmost subscript varies more rapidly in choosing the element of the array to be stored. For example:

   The elements of the array A(2,2,3) would be stored in the following order:

   A(1,1,1), A(1,1,2), A(1,1,3), A(1,2,1), A(1,2,2), A(1,2,3), A(2,1,1), A(2,1,2), A(2,1,3), A(2,2,1), A(2,2,2), A(2,2,3).

2. Simple arrays are arrays whose elements are stored contiguously; interleaved arrays are arrays whose elements are not stored contiguously.

3. If the elements of an array are arithmetic, the array as an aggregate is described by an array dope vector (ADV). Information contained in the ADV includes:

   a. The virtual origin of the array. This is the byte address of the element whose subscript values are zero, that is, A(0,0,...,0). This origin is called virtual because this element may be hypothetical and its address not included in the area of storage actually allocated to the array.

   b. The upper bound and lower bound for each dimension of the array.

   c. A multiplier for each dimension i as a function of the multiplier for dimension i+1, the upper and lower bounds of dimension i+1, and the length of array elements (which is the same for all elements in the array).

   d. The number of dimensions in the array.

   Given the information in the ADV, three functions can be performed:

   a. Given an array, the elements can be stepped through in row-major order.

   b. Given the subscript values of an element, the element address can be obtained.

   c. Given an element address, its subscript values can be obtained.

4. All elements of an array must have the same attributes, so the data element descriptor (DED) which describes one of these scalar elements describes each of the elements in the array.

Module Description

The CALL/360-OS PL/I Language Reference Manual lists the following built-in function names under "Generic Functions for Manipulation of Arrays."

Function Name	Function Value
LBOUND(x,n)	Returns current lower bound of nth dimension of array x.
HBOUND(x,n)	Returns current upper bound of nth dimension of array x.
DIM(x,n)	Returns current extent of nth dimension of array x.
SUM(x)	Returns sum of all elements of array x.
PROD(x)	Returns product of all elements of array x.
POLY(a,x)	Given two vectors, a and x; evaluates them as if a were the coefficient vector and x the variable vector in a polynomial equation.

For the built-in function POLY, both vector arguments are assumed to be stored in a simple manner, so only one set of AMP routines is provided. However, alternate entry points are provided to differentiate the case where argument x is a scalar from the case where it is a vector. These entry points are shown below.

Function Name	System Name	Element Attributes of Array x
	IHEYGFV	Real, Fixed Vector
	IHEYGFS	Real, Fixed Scalar
	IHEYGXV	Complex, Fixed Vector
	IHEYGXS	Complex, Fixed Scalar
	IHEYGSV	Real, Short Float Vector
POLY(A,X)	IHEYGSS	Real, Short Float Scalar
	IHEYGWV	Complex, Short Float Vector
	IHEYGWS	Complex, Short Float Scalar
	IHEYGLV	Real, Long Float Vector
	IHEYGLS	Real, Long Float Scalar
	IHEYGZV	Complex, Long Float Vector
	IHEYGZS	Complex, Long Float Scalar

The AMP routines are unique; they accept entire arrays as arguments, rather than one element of the array at a time. An AMP service routine, IHEJXI, performs the operation of indexing through the input arrays for other routines in AMP. One call must be made to IHEJXI in order to obtain one element.

LBOUND, HBOUND, and DIM are not supported by the library. These functions can be most efficiently performed by in-line code, rather than by a library call. The remainder of the built-in functions (SUM, PROD, ánd POLY) are supported by the LIBCOMP Aggregate Manipulation Package (AMP).

For each of the built-in functions SUM and PROD, a set of AMP routines is provided for interleaved arrays; separate routines are provided for each of the possible combinations of attributes the array elements may have. (The library may be simplified by always assuming interleaved arrays. This would sacrifice execution efficiency for these routines.)

Function Name	Interleaved Set System Name	Element Attributes of Array x
SUM(x)	IHESMFO	Real, Fixed
	IHESMXO	Complex, Fixed
	IHESMGR	Real, Short Float
	IHESMGC	Complex, Short Float
	IHESMHR	Real, Long Float
	IHESMHC	Complex, Long Float
PROD(x)	IHEPDFO	Real, Fixed
	IHEPDXO	Complex, Fixed
	IHEPDSO	Real, Short Float
	IHEPDWO	Complex, Short Float
	IHEPDLO	Real, Long Float
	IHEPDZO	Complex, Long Float

<u>Summary</u>

The library array modules are summarized in Figures 4-19 and 4-20.

	Interleaved string arrays with fixed-length elements
Indexers	IHEJXI

Figure 4-19. Array Indexers

<u>Note:</u> IHEJXI is used for indexing through interleaved arithmetic arrays.

	Fixed-Point Arguments	Floating-Point Arguments	
		Short Precision	Long Precision
Function	Interleaved	Interleaved	Interleaved
SUM real	IHESMF	IHESMG	IHESMH
complex	IHESMX	IHESMG	IHESMH
PROD real	IHEPDF	IHEPDS	IHEPDL
complex	IHEPDX	IHEPDW	IHEPDZ
POLY real	IHEYGF	IHEYGS	IHEYGL
complex	IHEYGX	IHEYGW	IHEYGZ

Figure 4-20. Arithmetic Array Functions

SECTION 5 - RUNTIME ROUTINE DIRECTORY

As noted in the preceding section, runtime support for CALL/360-OS PL/I applications is provided by the Library Interface Services (LIBINT) and Library Computational Services (LIBCOMP) modules. The individual routines that form these modules are explained in detail in this section. The routines are grouped within the organizational framework introduced during the general discussion. Thus, runtime support is described in detail in this section as follows.

Library Interface Services (LIBINT):

- I/O Management Package
- Handling of Interrupts Package
- Management of Object Program Package

Library Computational Services (LIBCOMP):

- Total Conversion Package
- String Manipulation Package
- Arithmetic Function Package
- Mathematical Function Package
- Aggregate Manipulation Package

I/O MANAGEMENT PACKAGE

The following routines constitute the I/O Management Package (IOMP). Descriptions of these routines are given on succeeding pages of this manual. The routines are discussed in alphabetic order, according to their mnemonics, as indicated.

- Close (IHECLOSE)
- Data-Directed Input (IHEDDI)
- Data-Directed Output (IHEDDO)
- Perform Calculation of the Subscript Values for an Array Element (IHEDDP)
- Edit I/O Director (IHEDIO)
- List- or Edit-Directed GET Initiation and Termination (IHEIOA)
- Output Initialization with or without Skipping (IHEIOB)
- Output Data to the Buffer Area and Communication with CALL/360-OS (IHEIOD)
- Get Data Field from Input Buffer (IHEIOG)
- Perform SKIP(w) Function for SYSPRINT (IHEIOP)
- Edited Horizontal Control Format Item (IHEIOX)
- List- and Data-Directed Input (IHELDI)
- List-Directed Output (IHELDO)
- Open (IHEOPEN)
- Reset Disk Files (IHERSET)

TITLE: CLOSE (IHECLOSE)

Program Definition

Purpose and Usage

The Close routine is used to close a disk file.

Description

If the file is an output file, this routine is used to write current buffer of data and close the file.

Errors Detected

ILLEGAL FILENAME. (123)
*DIRECTORY MISSING. (136)
INEXPLICABLE I/O ERROR. (301)

Local Variables

None

Program Interface

Entry Points

IHECLOSE	Linkage:	P7 - A(FCIB)
		P8 - RETURN
		P9 - ENTRY
	Called by:	Compiled Code

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Exit to IHEERRB.

Routines Called

IHEERR	Error Routine
IHESVC	Library SVC Director

Global Variables

None

TITLE: DATA-DIRECTED INPUT (IHEDDI)

Program Definition

Purpose and Usage

The Data-Directed Input routine handles initiation of data-directed input operation and assignment of input data to internal variables according to symbol table information conventions.

Description

An item is read from the specified input stream according to the rules specified for data-directed input. The item is scanned as follows:

1. Any leading blanks are ignored.
2. A search is made for an equal sign or a left parenthesis.
3. If an equal sign is found, the input contains a scalar item and control passes to 4.

   If a left parenthesis is found, the input contains an array item and scanning is continued until an equal sign is found.
4. The List- and Data-Directed Input routine is called to scale the value part. (Entry Point: IHELDIC.)

The address of the symbol table is provided in the FCB (file control block). The symbol table is searched for a name the same as that just scanned. If there is no such name, an error is recognized. (For details of the symbol table, see Appendix E.)

Dimensionality must be correct:

1. If the name is a scalar and if subscripts appear in the input stream, an error is recognized.
2. If the name is an array and if no subscripts appear in the input stream, an error is recognized. This routine checks the subscripts of the input item and addresses the specified element of the array.

The value of the input item is assigned to the internal variables using list-directed input (Entry Point: IHELDID).

If the transmission terminator is found, return is made to caller. If a NL character is found before the terminator, another item is scanned as described above.

Errors Detected

EXTRA INPUT DATA IGNORED. (025)
NOT OPENED. (124)
NOT FILE TYPE. (132)
SUBSCRIPT RANGE. (500)
IMPROPER NUMBER OF SUBSCRIPTS FOR DATA INPUT VARIABLE. (803)
DATA NAME NOT FOUND IN SYMBOL TABLE. (805)
SUBSCRIPT NOT IN USER AREA. (806)
DATA I/O ON INTERNAL FILE. (808)

Local Variables

None

Program Interface

Entry Points

IHEDDIB	Data-directed input with or without data-list Linkage: P7: A(FCB) Called by: Compiled Code

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If end-of-file condition has been raised, call IHEERR.

Routines Called

IHELDI	List- and Data-Directed Input
IHEERR	Error Routine
IHEIOA	List- or Edit-Directed GET Initiation and Termination

Global Variables

BUFLTH	Terminal Buffer Length (Communications Area)
BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)
FCBDEF	File Control Block Definition
UTTLOC	Address of User Terminal Table (Communications Area)
L#INFLA	Status of List Links (UTT)
SYMTABLE	Symbol Table
FCIBDEF	File Control Interface Block Definition

TITLE: DATA-DIRECTED OUTPUT (IHEDDO)

Program Definition

Purpose and Usage

The Data-Directed Output routine is used to convert data according to data-directed output conventions.

Description

Scalar Variable (Entry Point: IHEDDOA):

The output string is created as follows:

1. Obtain the variable name from the symbol table and place it on the output string.
2. Insert = immediately following the name.
3. Call IHELDO to direct the conversion and place the converted data in the output string.
4. Step down the symbol table. If the variable is not the last name, then repeat 1, else return to the caller.

Array Element (Entry Point: IHEDDOB):

1. Call IHEDDP to evaluate the subscript values of the array element.
2. Obtain array name from the symbol table and place it with its subscript values into the output string.
3. Insert = following the array name.
4. Call IHELDO to direct the conversion and place the converted data in the output string.

Termination (Entry Point: IHEDDOC):

Insert ; in the output string. This is used to terminate a data list.

Errors Detected

None

Local Variables

WORKREL	Offset from the relocatable library work space where SDV (string dope vector) is to be created.
WORKDREL	Offset from nonrelocatable library work space which is used as working storage area.

Program Interface

Entry Points

IHEDDOA	Linkage: Called by:	P7: A(FCIB) Compiled Code
IHEDDOB	Linkage: Called by:	P7: A(FCIB) Compiled Code

IHEDDOC	Linkage:	P7: A(FCIB)
	Called by:	Compiled Code

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEIOD	Output Data to the Buffer Area and Communication with CALL/360-OS
IHEDDP	Perform Calculation of the Subscript Values for an Array Element
IHELDO	List-Directed Output
IHEERR	Error Routine

Global Variables

FCB	File Control Block
SYMTABLE	Symbol Table

TITLE: PERFORM CALCULATION OF THE SUBSCRIPT VALUES FOR AN ARRAY ELEMENT (IHEDDP)

Program Definition

Purpose and Usage

The Perform Calculation of the Subscript Values for an Array Element routine is used to calculate subscript values of an array element with the address information provided by FCB (file control block) and an ADV (array dope vector).

Description

The algorithms for calculating the subscript values are given as follows:

$Si = FLOOR(Ri/Mi)$ for $i=1,2,...,(n-1)$
$Sn = Rn$

where:

Si = ith subscript value

$R1$ = array element address - virtual origin

$Ri = MOD(Ri-1, Mi)$

n = the dimensionality of the array

i = the dimension under consideration

Mi = ith multiplier

$$M = \prod_{i=1}^{n} Mi$$

The array name with its calculated subscript values will be placed in the output string.

Errors Detected

None

Local Variables

LNGTH (Register G0)	Length of array element
DIMENS (Register G2)	Dimension of the array

Program Interface

Entry Points

IHEDDPD	Linkage:	P7 (FCB)
	Called by:	IHEDDO

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEIOD	Output Data to the Buffer Area and Communication with CALL/360-OS

Global Variables

FCB	File Control Block
SYMTABLE	Symbol Table
ADV	Array Dope Vector

TITLE: EDIT I/O DIRECTOR (IHEDIO)

Program Definition

Purpose and Usage

The Edit I/O Director accepts a format code in register G1 and directs control to the proper library routine.

Description

Using the format code as an index, either an input or output director table is accessed, depending on the file type. The director is then called.

Errors Detected

EDIT I/O ON INTERNAL FILE. (810)

Local Variables

None

Program Interface

Entry Points

IHEDIOA

Exit Conditions

Call to proper I/O director.

Routines Called

IHEDOM	C-Format Output Director
IHEDOA	F/E-Format Output Director
IHEDOB	A-Format Output Director
IHEIOX	Edited Horizontal Control Format Item
IHEIOP	Perform SKIP(w) Function for SYSPRINT
IHEDIM	C-Format Input Director
IHEDIA	F/E-Format Input Director
IHEDIB	A-Format Input Director
IHEERR	Error Routine

Global Variables

None

TITLE: LIST- OR EDIT-DIRECTED GET INITIATION AND TERMINATION (IHEIOA)

<u>Program Definition</u>

Purpose and Usage

The List- or Edit-Directed GET Initiation and Termination routine initiates or terminates list- or edit-directed GET statements.

Description

<u>Initiation</u> (Entry Point: IHEIOAA):

If the input file is a disk file, the following tests are performed:

1. If the current buffer address is not equal to the buffer address given in the file control block (FCB), return is made to the caller.

2. If the two addresses are equal, a new record is read. If no error, then return is made to caller. If error, the error routine is entered.

If the input file is a terminal file, a new line is read from the terminal. Then return is made to a caller.

<u>Termination</u> (Entry Point: IHEIOAT):

If the input file is a terminal file, a scan is required up to the NL character. If nonblank characters are found, an error condition is raised. Processing continues.

Errors Detected

UNRECOVERABLE I/O ERROR. (125)
DECLARED ENVIRONMENT NOT COMPATIBLE WITH INPUT FILE. (811)

Local Variables: None

<u>Program Interface</u>

Entry Points

```
IHEIOAA    Initiation of an input operation
               Linkage:     P7: A(FCIB)
               Called by:   Compiled Code, IHELDI, IHEDDI

IHEIOAT    Termination of an input operation
               Linkage:     P7: A(FCIB)
               Called by:   Compiled Code
```

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEERR	Error Routine
IHESVC	Library SVC Director

Global Variables

FCB	File Control Block
BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)

TITLE: OUTPUT INITIALIZATION WITH OR WITHOUT SKIPPING (IHEIOB)

Program Definition

Purpose and Usage

The Output Initialization with or without Skipping routine initializes PUT statements with or without SKIP option.

Description

If the output file is OUTFILE and the current buffer address is equal to the buffer address in the FCB, the control bytes (first seven bytes for each disk file record) are set according to information provided by FCB. If no SKIP function is to be performed, return is made to the caller; otherwise, IHEIOP is called to perform the SKIP function.

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHEIOBA    To initialize the PUT operation
                   Linkage:     P7: A(FCIB)
                   Called by:   Compiled Code

    IHEIOBC    To initialize PUT, and perform SKIP
                   Linkage:     P7: A(FCIB)
                   Called by:   Compiled Code
```

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEIOP	Perform SKIP(w) Function for SYSPRINT
IHEERR	Error Routine

Global Variables

FCB	File Control Block

TITLE: OUTPUT DATA TO THE BUFFER AREA AND COMMUNICATION WITH CALL/360-OS (IHEIOD)

Program Definition

Purpose and Usage

The Output Data to the Buffer Area and Communication with CALL/360-OS routine is used to place the converted data string in the proper location in the buffer area and update the current buffer pointer provided by FCB. An SVC (Supervisor Call) to the Executive is issued when the buffer is filled. The routine determines when an NL character should be placed on the output string for terminal buffer.

Description

If the output file is a disk file:

1. If the current buffer pointer is equal to the buffer starting address, set the first seven bytes of the buffer according to information provided by FCB. (The first seven bytes are used for record description for disk file.)

2. If the disk file is an external print file, use the procedures described for the terminal file (SYSPRINT).

3. If the length of the data plus total characters generated on the buffer is greater than the buffer size:

   a. Place an EOF at the location given by the current buffer pointer.
   b. Request an SVC for writing the record to disk.
   c. Set current buffer address to buffer address and repeat 1.

4. If the disk file is an internal disk file (data stored in internal form), two additional bytes for each data element are set to describe the data type and replicator. The data element is transferred to the output buffer according to information given in the string dope vector (SDV).

If the output file is a terminal unit:

1. If the length of the converted data string + number of bytes used for one line > the width of the terminal unit:

   a. Insert an NL to terminate the line.
   b. Set number of bytes used to zero.
   c. Update the buffer pointer.
   d. If the remaining buffer size is less than the width of the terminal unit, request an SVC to transmit the data to output terminal.
   e. Place the converted data string to the terminal buffer and update the number of bytes used and the buffer pointer.

2. If the length of the converted data string + number of bytes used for the line = the width of the terminal unit:

   a. Place the converted data string to the terminal buffer and insert an NL to terminate the line. Set number of bytes used for the line to zero.
   b. Do 1.d.

3. If the length of the converted data string + number of bytes used for the line < the width of the terminal unit, do 1.e.

Errors Detected

INEXPLICABLE I/O ERROR. (301)
MAXIMUM STRING LENGTH EXCEEDED. (706)
END OF OUTPUT FILE. (802)

Local Variables

COUNT1 (Register G3)	Counter for number of characters generated in a line for print file
LENGTH (G1)	Length of source string
TOTCHAR (G5)	Total number of characters generated in the buffer
BUFFSIZE (G0)	Buffer size
LINEWDTH (G4)	Line width for print file
BUFPOINT (P2)	Current buffer pointer
ADSDV (P5)	Address of SDV
BUFADD (P8)	Buffer starting address
ADDED (P4)	Address of the data element descriptor
SOURCEAD (P8)	Buffer starting address
NUSPEC (G3)	Number of specifications for internal disk file
PUSPEC (P4)	Address of old or last data specification

Program Interface

Entry Points

IHEIODP	Linkage:	P7: A(FCB) P5: A(SDV)
	Called by:	Edit-directed I/O directors, IHEIOX, IHEIOP, IHELDO, IHEDDO, and IHEDDP.

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Call IHEERR on I/O error.

Routines Called

IHEERR	Error Routine

Global Variables

FCB	File Control Block
SDV	String Dope Vector
WTOTCHAR	Total characters generated for the SYSPRINT buffer (LCA)
WTOTCHDK	Total characters generated for the disk output buffer (LCA)
WCOUNT1	Number of characters generated in a line for SYSPRINT (LCA)
WCOUNTDK	Number of characters generated in a line for disk print file (LCA)
WDISBUFS	Disk Buffer Size (LCA)
WTERBUFS	Terminal Buffer Size (LCA)
BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)
OPFLAG	Output Inhibit Flag for SYSPRINT (Communications Area)
SPEC	Pointer to old or last data specification field in buffer

TITLE: GET DATA FIELD FROM INPUT BUFFER (IHEIOG)

Program Definition

Purpose and Usage

The Get Data Field from Input Buffer routine is used to collect the data field from the input buffer. If more than one record is to be read, the data is stacked in the library communications area (LCA).

Description

If the input file is a disk file, the buffer is scanned from the current buffer address to the current buffer address plus the data length. If no EOF character is encountered, P7 (register 13) is set to the current buffer address. Then the current buffer address in FCB is updated and return is made to the caller. If an EOF character is encountered, the current part of the data is stacked in LCA. A new record is read and the remaining data is stacked in LCA. P7 is set to the address of LCA where the data is stacked.

If the input file is a terminal file, a similar procedure is performed. However, the NL character is scanned instead of the EOF character.

Errors Detected

UNRECOVERABLE I/O ERROR. (125)
END OF FILE ENCOUNTERED. (140)

Local Variables

BUFPOINT (P2)	Current buffer pointer
STARTAD (P4)	Start address for the data field in buffer
STACKAD (P5)	Address in LCA where data is to be stacked
COUNT (G0)	Counter for the number of characters scanned
LENGTH (G1)	Length of data field to be scanned
FLAG1 (G3)	Zero sufficient characters in buffer. For new record read, data is stacked in LCA.

Program Interface

Entry Points

IHEIOGA	Linkage:	P7: A(FCB) G1: Length of data field
	Called by:	Edited input director. Upon return, P7 contains (source), and G1 (length).

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Call IHEERR.

Routines Called

IHEERR	Error Routine

Global Variables

FCB	File Control Block
WFCS	Area where data is stacked (LCA)

TITLE: PERFORM SKIP(W) FUNCTION FOR SYSPRINT (IHEIOP)

<u>Program Definition</u>

Purpose and Usage

The Perform SKIP(w) Function for SYSPRINT routine is used to perform the SKIP function for output print file.

Description

W-NL characters are created in the nonrelocatable library work space. IHEIOD is called to transfer these characters to the output buffer.

Errors Detected

None

Local Variables

ADSOURCE (P5)	Address where NL characters are created
LENGTH (G1)	Length of the number of NL characters to be created

<u>Program Interface</u>

Entry Points

IHEIOPB	Linkage:	P7: A(FCB)
	Called by:	IHEIOBC

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEIOD	Output Data to the Buffer Area and Communication with CALL/360-OS

Global Variables

FCB	File Control Block

TITLE: EDITED HORIZONTAL CONTROL FORMAT ITEM (IHEIOX)

Program Definition

Purpose and Usage

The Edited Horizontal Control Format Item routine is used in two ways:

- Input: Spaces over next w characters in input stream.
- Output: For control format item, inserts w blanks in the output string. For COLUMN(w), inserts blanks up to w-th character of current or next line.

Description

Input (Entry Point: IHEIOXA):

```
If disk-file then:

     Current Buffer Pointer for Disk File = Current Buffer Pointer
     for Disk File + w

else:

     Current Buffer Pointer for SYSIN = Current Buffer Pointer for
     SYSIN + w
```

Output:

1. Edit-Directed X(w) Format (Entry Point: IHEIOXB): Insert w blanks in the output buffer and update current buffer address in FCB and the global variables in LCA.

2. COLUMN(w) Format (Entry Point: IHEIOXC): If w is less than or equal to the number of characters generated in the current line:

   a. Insert an NL character in the output string.
   b. Insert w blanks after the NL character.

   If w is greater than the number of characters generated for the current line, insert w (number of characters generated) blanks in the output buffer.

Errors Detected

PRINT OPTION FORMAT ITEM FOR NON PRINT FILE. (024)

Local Variables

COUNT1 (G1)	Number of characters generated in a line
W (G2)	Width of data field in characters
LNEWDTH (G0)	Line width for print file
BUFSIZE (G3)	Buffer size
TOTCHAR (G4)	Total number of characters generated in the buffer
ADUTT (G5)	Address of user terminal table
NEWIDTH (G5)	Secondary count of width of data field in characters

Program Interface

Entry Points

IHEIOXA	Linkage:	P7: A(FCB)
	Called by:	Compiled Code

IHEIOXB	Linkage:	P7: A(FCB)
	Called by:	Compiled Code
IHEIOXC	Linkage:	P7: A(FCB)
	Called by:	Compiled Code

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEERR	Error Routine

Global Variables

FCB	File Control Block
BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)

TITLE: LIST- AND DATA-DIRECTED INPUT (IHELDI)

<u>Program Definition</u>

Purpose and Usage

The List- and Data-Directed Input routine works in two ways:

- For list-directed input, it scans one item in the input string and assigns it to internal variable according to rules specified.
- For data-directed input, the constant part of the assignment is scanned and assigned to internal variable.

Description

<u>List-Directed Input</u> (Entry Point: IHELDIB):

1. External Files: Data in the stream is scanned as follows:

    a. Any leading blanks are ignored.

    b. If the first character is a quote mark, a string constant is assumed to follow, and a search is made for a subsequent single quote mark. Otherwise, an arithmetic constant is assumed, and a search is made for an item delimiter (blank or comma).

    c. According to the DED of the internal variable and the type of constant (or string), TCP (Total Conversion Package) modules are called to do the conversion and assign the converted data to internal variable.

2. Internal Files: Source and target specifications are checked for compatibility with respect to type (arithmetic or string) and, if arithmetic, to scale (fixed or float). Should the source and target be incompatible, an appropriate error message is printed. Otherwise, source data is moved to the target. Account is taken of any differences in the precision and mode of the source and target to preserve the validity of the data moved and the integrity of the user area.

<u>Data-Directed Input Constant Scanning</u> (Entry Point: IHELDIC):

The scanning process used for list-directed input can be used for data-directed input. However, a return code is set as follows:

Return Code = 0:	Not last item.
Return Code = 4:	Last item.
Return Code = 8:	End of file encountered before complete data field collected.

Errors Detected

END OF FILE ENCOUNTERED. (140)
INEXPLICABLE I/O ERROR. (301)
ERROR IN CONVERSION FROM ARITHMETIC TO CHARACTER STRING. (605)
ERROR IN CONVERSION FROM FIXED TO FLOAT. (606)
ERROR IN CONVERSION FROM FLOAT TO FIXED. (607)

Local Variables

DBFLAG (G6)	Used to flag the existence of double quotation marks in string for non-print file: 0 = No double quotation mark 4 = Double quotation mark
COUNT1	Length of real part of source
COUNT2	Length of imaginary part of source
COUNT11	Length of real part of target
COUNT22	Length of imaginary part of target
OFSWC	Length adjustment switch used to effect proper update of buffer source pointer

Local Offsets

PLIST	Offset of parameter list in the relocatable library work space
WORK	Offset of string dope vector describing the source string

<u>Program Interface</u>

Entry Points

IHELDIB	Linkage:	P7: A(FCIB)
	Called by:	Compiled Code
IHELDIC	Linkage:	P7: A(FCB) with word 3 of FCB containing a (VARIABLE) and word 4, a (DED).
	Called by:	IHEDDI

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Call IHEERR.

Routines Called

IHESVC	Library SVC Director
IHEDCN	Character String to Arithmetic
IHEDNC	Arithmetic to Character String
IHEERR	Error Routine
IHEVCA	Data Analysis Routine
IHEVSC	Character String to Character String
IHEIOA	List- or Edit-Directed GET Initiation and Termination

Global Variables

BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)
FCB	File Control Block
WFCI	Addresses of source, target, source DED, and target DED (LCA)

TITLE: LIST-DIRECTED OUTPUT (IHELDO)

<u>Program Definition</u>

Purpose and Usage

The List-Directed Output routine is used to output data under the rules for list-directed output.

Description

If the file is an internal disk file, the output file is created without proper conversion. If it is an external file, a TCP routine is called to convert arithmetic data to external form before placing it in the output buffer. For string variables, transfer of the string to the output buffer is performed by this routine.

Errors Detected

SUBSTRING NOT IN DATA AREA. (705)

Local Variables

LENGTH (G0)	Length of source string

<u>Program Interface</u>

Entry Points

IHELDOB — List-directed output (single variable)
- Linkage: P7: A(FCIB)
- Called by: Compiled Code

IHELDOC — Data-directed output
- Linkage: P7: A(FCB)
  - <u>Note</u>: FCB has been modified similar to FCB for list-directed output.
- Called by: IHEDDO

Exit Conditions: Normal exit. Return to caller.

Routines Called

IHEDNC	Arithmetic to Character String
IHEVSC	Character String to Character String
IHEIOD	Output Data to the Buffer Area and Communication with CALL/360-OS
IHEERR	Error Routine

Global Variables

FCB	File Control Block
BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)
WCNP	Location where parameter list is passed while calling TCP routine (LCA)
WCOUNTDK	Number of characters generated in a line for disk print file (LCA)
WCOUNT1	Number of characters generated in a line for SYSPRINT (LCA)
WTOTCHAR	Total characters generated for the SYSPRINT buffer (LCA)
WTOTCHDK	Total characters generated for the disk output buffer (LCA)

TITLE: OPEN (IHEOPEN)

Program Definition

Purpose and Usage

The Open routine is used to open a disk file.

Description

If the file is already open, the OPEN is ignored. An I/O buffer and a file control block (FCB) are assigned to the file.

Errors Detected

UNRECOVERABLE I/O ERROR. (125)
DOES NOT EXIST. (126)
LOCKED. (127)
IN USE. (128)
PROTECTED. (129)
NOT A DATA FILE. (130)
A SHARED FILE. (131)
ILLEGAL ATTRIBUTES. (134)
EXCEEDS FOUR FILES OPEN. (135)
*DIRECTORY MISSING. (136)

Local Variables

None

Program Interface

Entry Points

IHEOPEN	Linkage:	P7 - A(FCIB)
		P8 - RETURN
		P9 - ENTRY
	Called by:	Compiled Code

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Exit to IHEERRB.

Routines Called

IHEERR	Error Routine
IHESVC	Library SVC Director

Global Variables

FCIB	File Control Interface Block
FCB	File Control Block

TITLE: RESET DISK FILES (IHERSET)

Program Definition

Purpose and Usage

For output files, write current half-track, reset disk pointers to start of file, and reset current buffer pointer to start address.

For input files, reset current buffer pointer to start address and reset disk pointers to start of file.

Description

$RESET attributes relative to the statement:

CALL $RESET [(filename, filename)];

1.	Contextual	-	DECLARE $RESET ENTRY Do not use IHERSET
2.	Implicit	-	Used as a variable Do not use IHERSET
3.	Explicit	-	DCL $RESET [ATTRIBUTE]; Do not use IHERSET
		-	LABEL: PROC Do not use IHERSET
		-	LABEL: STATEMENT Do not use IHERSET
4.	Tentative	-	ENTRY Use IHERSET
		-	Other Do not use IHERSET

Errors Detected

ILLEGAL FILENAME. (123)
NOT OPENED. (124)
INEXPLICABLE I/O ERROR. (301)
END OF OUTPUT FILE. (802)

Local Variables

MINUS	Used by SVC 3 and SVC 4 to reset disk pointers to one less than their current values.

Program Interface

Entry Points

IHERSET	Linkage:	P7: A(FCIB) P8: Return Address P9: Entry Point Address
	Called by:	Compiled Code

Exit Conditions

Normal exit. Return to caller via P8.
Abnormal exit. Exit via IHEERRB.

Routines Called

IHEERR	Error Routine

Global Variables

FCBFNAME	File Codes: 00 SYSIN 01 INPUT 10 SYSPRINT 11 OUTPUT
FCBUFAD	Address of Buffer
FCBCUBUF	Current Buffer Pointer

HANDLING OF INTERRUPTS PACKAGE

The following routines constitute the Handling of Interrupts Package (HIP). Descriptions of these routines are given on succeeding pages of this manual. The routines are discussed in alphabetic order, according to their mnemonics, as indicated.

Program Termination (IHEDUM)
Table of Error Messages and Indicators (IHEERN)
Error Routine (IHEERR)
On-ENDFILE and REVERT Initializer (IHEONREV)

TITLE: PROGRAM TERMINATION (IHEDUM)

Program Definition

Purpose and Usage

The Program Termination routine is used to terminate the program.

Description

All open disk files are closed. Final exit is made via SVC 0.

Errors Detected

EXCEEDS FOUR FILES OPEN. (135)
INEXPLICABLE I/O ERROR. (301)

Local Variables

None

Program Interface

Entry Points

IHEDUMP	Linkage:	None
	Called by:	Compiled Code or IHEERRC

Exit Conditions

Control is not returned to the caller.

Routines Called

RTSSVC	Runtime support macro to transmit any information contained in the terminal buffer or disk buffer and to terminate the program (see Appendix D)

Global Variables

FCB	File Control Block

TITLE: TABLE OF ERROR MESSAGES AND INDICATORS (IHEERN)

Program Definition

Purpose and Usage

The Table of Error Messages and Indicators (IHEERN) contains the action code of the execution errors and the runtime error messages.

Description

IHEERN contains no executable statement. The routine contains four tables:

1.	EINDEX	Error Index Table
2.	ERTABL	Error Table
3.	MESTAB	Message Table
4.	ROUT	Routine Name Table

EINDEX contains the offset to ERTABL according to type of error, that is, I/O, CONVERSION, OVERFLOW, etc. ERTABL contains the offset to MESTAB for each individual error and action indicator. For routines in the Mathematical Function Package (MFUNC), ERTABL also contains the offset to the routine name table. MESTAB contains all runtime error messages and an error index associated with each message. ROUT contains all routine names used by the MFUNC routines.

During execution, when an error is detected, an error code is set in the nonrelocatable library work area (LWE). The format of the error code is given below:

1	2	3	Byte
Index To EINDEX	Index To ERTABL	File-Type Indicator	

The specific entrance in ERTABL can be located with the two indexes given in the error code. The file-type indicator is meaningful only when I/O errors occur. The action indicator given in ERTABL provides all information required for IHEERR to direct specific action to be taken (on-unit action or standard system action to be taken after the error message is printed).

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

IHEERNA	Linkage:	None
	Called by:	IHEERRB

Exit Conditions: None

Routines Called: None

Global Variables: None

TITLE: ERROR ROUTINE (IHEERR)

Program Definition

Purpose and Usage

Error Routine is used to determine the identity of the error or condition that has been raised, and to determine what action must be taken on account of it. Several actions are possible, including combinations:

1. Entry into an on-unit.
2. Print error message and terminate.
3. Print error message and continue.
4. No action. Return to program.

Description

Arithmetic Interrupts (Entry Point: IHEERRA): There are four types of arithmetic interrupts:

1. FIXEDOVERFLOW (interrupt code = 8 in PSW).
2. OVERFLOW (interrupt code = 12 in PSW).
3. UNDERFLOW (interrupt code = 13 in PSW).
4. ZERODIVIDE (interrupt code = 9 or 15 in PSW).

PSW2SV (location hex 88 in the communications area) contains the PSW when arithmetic interrupt occurs. In the fixed portion of each DSA, there is a corresponding word for each interrupt, which is used to specify the action for the interrupt. The format for each test-word is as follows:

C O D E	On-Unit Object Code Pointer

Test-Word

Code = 0:	Not specified.
Code = 1:	Standard system action.
Code = 3:	User's action (on-unit).

Note: On-unit object code pointer is meaningful only for code = 3.

Depending on the interrupt code given in PSW, the proper word in the DSA is examined:

1. If code = 0 and the DSA containing the test-word is the first DSA or code = 1, standard system action is performed:

a.	FIXEDOVERFLOW:	Comment and raise the error condition.
b.	OVERFLOW:	Comment and raise the error condition.
c.	UNDERFLOW:	Comment and continue.
d.	ZERODIVIDE:	Comment and raise the error condition.

2. If code = 0 and this DSA is not the first DSA, the corresponding word in the previous DSA is used as the test-word and step 1 is repeated.

3. If code = 3, a call is generated to the on-unit. (Saving and restoring information of all registers is done by the on-unit.) The last three bytes of the test-word contain an offset into the adcon area where the on-unit adcon area is located; the address of the on-unit entry point is stored there.

ON or Non-ON Execution Errors (Entry Point: IHEERRB): All execution errors are handled by IHEERRB. Possible actions to be taken are:

1.	On-Unit Error Standard System Action:	Comment and raise error condition.
2.	On-Unit Error Standard System Action:	Comment and terminate.
3.	On-Unit Error Standard System Action:	Comment and continue.
4.	Comment and raise error condition.	
5.	Comment and terminate.	
6.	Comment and continue.	

For on-unit errors, the on-unit test-word in the DSA is examined. If no on-unit is specified, standard system action is taken as specified in the error table in IHEERN. The error messages and routine names are also given in IHEERN.

Error Conditions (Entry Point: IHEERRC): Same procedure as arithmetic interrupt is used to determine whether standard system action (terminate the major task) or user's action is to be performed for error condition.

Restore Registers and PSW for an On-Unit (Entry Point: IHEERRR): This routine is used to restore the second word of PSW2SV in the user's communications area and general and floating-point registers for an on-unit. Control is then returned to CALL/360-OS.

Restore Registers and PSW for On-ENDFILE Unit (Entry Point: IHEERRN): This routine reinitializes P0 through P5 and transfers via P9 to the next statement following the statement which caused the on-ENDFILE condition.

Subscript Not in User Area (Entry Point: IHEERRZ): This routine is scheduled by the first six bytes of object code in the user's area when a string or array reference is made to a location greater than the user's area or less than the start of the static string storage area.

Errors Detected

PROGRAM ERROR - EXECUTION TERMINATED. (902)

Local Variables

LNGTHMES (Register G5)	Length of the error message in IHEERN
ADBUFFER (P2)	Output buffer address
ADL≠TAB (P8)	Address of object code address-line number table

Program Interface

Entry Points

IHEERRA	Linkage:	None
	Called by:	EXEC on Arithmetic Interrupt
IHEERRB	Linkage:	RA: A(Error Code)
	Called by:	Library Modules
IHEERRC	Linkage:	None
	Called by:	Compiled Code and Library Modules
IHEERRN	Linkage:	P7: A(On-Unit Adcon Area)
	Called by:	On-Unit
IHEERRR	Linkage:	P7: A(On-Unit Adcon Area)
	Called by:	On-Unit
IHEERRZ	Linkage:	None
	Called by:	Compiled Code

Exit Conditions

Depend on the type of errors.

Routines Called

IHEERN	Table of Error Messages and Indicators
IHEDUM	Program Termination

Global Variables

BUFPTR	Buffer Pointer for SYSPRINT (Communications Area)
OPFLG	Output Inhibit Flag (Communications Area)

TITLE: ON-ENDFILE AND REVERT INITIALIZER (IHEONREV)

Program Definition

Purpose and Usage

The On-ENDFILE and REVERT Initializer is used to initialize the on-ENDFILE condition unit to the current unit in effect.

Description

All ENDFILE filenames are searched to find the one in this block with the same FCIB. If none is found, an entry is created and the new on-unit information word is stored.

Local Variables

None

Errors Detected

None

Program Interface

Entry Points

```
    IHEONUN    Linkage:    P7:  A(Parameter list)
               Parameter list:  Address of on-unit adcon
                                Address of FCIB
    IHEREVT    Linkage:    P7:  A(FCIB)
```

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEERR	Error Routine

Global Variables

None

MANAGEMENT OF OBJECT PROGRAM PACKAGE

The following routines constitute the Management of Object Program Package (MOPP). Descriptions of these routines are given on succeeding pages of this manual. The routines are discussed in alphabetic order, according to their mnemonics, as indicated.

Output Director (IHEGPUT)
Initial Prologue, Expand DSA, End Prologue, Object Program
  Initiation (IHESAD)
GO TO Interpreter (IHESAF)
Library SVC Director (IHESVC)

TITLE: OUTPUT DIRECTOR (IHEGPUT)

Program Definition

Purpose and Usage

The Output Director places a 120-character line in the terminal buffer. It also removes trailing blanks and checks line width.

Description

A 120-character line is processed and placed in the terminal buffer. If there is insufficient space, the buffer is emptied. Trailing blanks are removed from the line. If the line exceeds the line width, it is broken into segments of the maximum length. Before return, the line is cleared to blanks.

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

Normal linkage to IHEGPUT. Register P2 has address of line.

Exit Conditions

Normal exit. All registers restored. Line cleared.

Routines Called

IHESVC	Library SVC Director

Global Variables

Communications Area

TITLE: INITIAL PROLOGUE, EXPAND DSA, END PROLOGUE, OBJECT PROGRAM INITIATION (IHESAD)

Program Definition

Purpose and Usage

The Initial Prologue, Expand DSA, End Prologue, Object Program Initiation routine has the following functions:

1. Initial prologue: Provides fixed part of the dynamic storage area for a begin or procedure block.

2. Expand DSA: Obtains automatic storage for elements declared within the block.

3. End prologue: Checks to see if sufficient space is available for the object program. If not, an SVC for more space is issued.

4. Object program initiation:
   a. Calculates space required for the object program.
   b. Sets certain global variables in LCA.
   c. If disk file has been declared, checks to see if links have been provided.

Description

Initial Prologue (Entry Point: IHESADA): DSA is aligned to a double-word boundary. If the fixed space required is greater than the size of the program, additional space is requested.

For procedure blocks, nonvolatile, nonrelocatable general registers and floating-point registers are saved in DSA. All on-unit information words in DSA are set to zeros.

Expand DSA (Entry Point: IHESADB): The automatic storage required for string variables and subscript variables is allocated. For subscript variables, the array dope vector and string array dope vector are set. The algorithms used are given as follows:

Mi = Ui - Li + 1

where:

Mi = ith multiplier

Ui = ith upper bound

Li = ith lower bound

Virtual Origin = Address of current location of DSA-base address of object program address:

(...(L1*M1 + L2)*M2 + ...) + Ln)*Mn

$$\text{Size of Array} = \prod_{i=1}^{n} Mi \qquad \text{where } Mi = Ui - Li + 1$$

End-Prologue (Entry Point: IHESADC): The size of the program is compared with the current DSA address. If more space is required, the RTSSVC macro performs this function (see Appendix D).

Object Program Initiation (Entry Point: IHESADD): The size of the program is calculated with the information given in UTT. The address of the last byte of a user's program is stored in the adcon area.

Time Function (Entry Point: IHESADE): Convert time from binary to EBCDIC.

Errors Detected

RECURSIVE BLOCK OR ON-UNIT. (807)

Local Variables

DIMENS (G1)	Dimension of the array
VIRTUAL (G0)	Virtual origin
ADDOPEV (P4)	Address of array or string dope vector
ADDED (P5)	Address of data element descriptor
ADFCB (P2)	Address of file control block
ADUTT (G4)	Address of user terminal table
ADCOMMUN (P5)	Address of user's communications area

Program Interface

Entry Points

IHESADA	Linkage:	P7: A(BAA)
		G1: Length of DSA
	Called by:	Compiled Code
IHESADB	Linkage:	P7: A(Parameter List)
	Parameter List:	A(Dope Vector)
		A(DED)
	Called by:	Compiled Code
IHESADC	Linkage:	None
	Called by:	Compiled Code
IHESADD	Linkage:	None
	Called by:	Compiled Code
IHESADE	Linkage:	P7: A(TARGET)
	Called by:	Compiled Code (Entered at IHESADE)

Exit Conditions

Normal exit. Return to caller.
Abnormal exit. Call IHEERRB.

Routines Called

IHEERR	Error Routine

Global Variables

BAA	Block Adcon Area
L#2048	Number of 2048-byte blocks allocated (UTT)

TITLE: GO TO INTERPRETER (IHESAF)

Program Definition

Purpose and Usage

The GO TO Interpreter frees all chain elements up to the DSA to which the label belongs.

Description

If the pointer to BAA (block adcon area) is not equal to the pointer to BAA in the present DSA, this routine:

1. Sets the BAA's DSA address to the base address.
2. Updates the current DSA address.
3. Repeats the test.

If the pointer to BAA is equal to the pointer to BAA in the current DSA, a branch is made to the specified label.

Errors Detected

ILLEGAL LABEL VARIABLE GO TO. (809)

Local Variables

None

Program Interface

Entry Points

IHESAFC	Linkage:	P7: A(Parameter List)
	Parameter List:	Offset to Label
		Offset to BAA

Exit Conditions

Normal exit. Return to caller.

Routines Called

None

Global Variables

CDSA	Current DSA Address

TITLE: LIBRARY SVC DIRECTOR (IHESVC)

Program Definition

Purpose and Usage

The Library SVC Director handles all SVC interfaces with the CALL/360-OS System for the library except for SVC 8.

Description

The SVC code is picked up from the parameter list in the halfword following the return point, and the proper SVC is executed. Return is to the location immediately following the parameter list.

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

Normal linkage and entry at IHESVCA. Halfword SVC code immediately after BALR to this routine.

Exit Conditions

Return is to the location two bytes after BALR.

Routines Called

None

Global Variables

None

TOTAL CONVERSION PACKAGE

The following routines constitute the Total Conversion Package (TCP). The routines can be organized in functional groups, as explained in the previous section of this manual and detailed below. Descriptions of the routines are given on succeeding pages. Within each functional group, the routines are discussed in alphabetic order, according to their mnemonics.

Edit-directed I/O directors:

F/E-Format Input Director (IHEDIA)
A-Format Input Director (IHEDIB)
C-Format Input Director (IHEDIM)
F/E-Format Output Director (IHEDOA)
A-Format Output Director (IHEDOB)
C-Format Output Director (IHEDOM)

Type conversion directors:

Character String to Arithmetic (IHEDCN)
Arithmetic to Character String (IHEDNC)

Mode conversion director:

Zero Real or Imaginary Part (IHEUPA)

String conversion routines:

Complex External to String Director (IHEVCS)
Character String to Character String (IHEVSC)

Arithmetic conversion routines and director:

Arithmetic Conversion Director (IHEDMA)
Float Intermediate to Packed Decimal Intermediate (IHEVFA)
Float Intermediate to Fixed Binary (IHEVFB)
Float Intermediate to Float Short or Long (IHEVFC)
Fixed Binary to Float Intermediate (IHEVFD)
Float Source to Float Intermediate (IHEVFE)
Packed Decimal Intermediate to Float Intermediate (IHEVPA)
Packed Decimal Intermediate to F-Format (IHEVPB)
Packed Decimal Intermediate to E-Format (IHEVPC)
String with Format to Packed Decimal Intermediate (IHEVPE)
Table of Powers of Ten (IHEVTB)

Constant analysis routine:

Data Analysis Routine (IHEVCA)

TITLE: F/E-FORMAT INPUT DIRECTOR (IHEDIA)

Program Definition

Purpose and Usage

The F/E-Format Input Director directs the conversion of external data with F/E-format to an internal data type.

Description

Functionally speaking, entry points IHEDIAA and IHEDIAZ are equivalent, as are IHEDIAB and IHEDIAY. IHEDIAA and IHEDIAB receive parameter requirements via a parameter list; IHEDIAZ and IHEDIAY receive parameter requirements via a file control block.

The LCA switch byte WSWA is used to direct processing:

Bit X'40'	identifies C-Format Input Director as the caller of the module, and
Bit X'01'	indicates conversion to involve the imaginary part of a complex data item.

A source dope vector with a string length of zero or less leaves the target unchanged. Acceptable DED flag byte patterns follow in hexadecimal format:

C8-C9	F-format character string
CA-CB	E-format character string (single precision)
DA-DB	E-format character string (double precision)
2B-2B	A-format character string
8C-8D	Fixed-point binary
8E-8F	Short floating-point binary
9E-9F	Long floating-point binary

Errors Detected

None

Work Area

Library work area is obtained from level three.

Local Variables

NWRK	Nonrelocatable library work area
PWRK PLIST PRAMS	Relocatable library work area

Program Interface

Entry Points

IHEDIAA - Entry for F-format input string
P7 = A (Parameter List)

where Parameter List:

A (Source)
A (Target)
A (Target DED)
A (Source FED)

IHEDIAB - Entry for E-format input string. Calling sequence is as described for entry IHEDIAA.

IHEDIAZ - Functionally equivalent to IHEDIAA

```
   P7  = A (FCB)
   PCB = A (Buffer)
         A (Current Buffer)
         A (Target)
         A (Target DED)
         A (Real FED)
```

IHEDIAY - Functionally equivalent to IHEDIAB; calling sequence is as described for entry IHEDIAZ.

Note: Library Common Variables:

WSWA = X' 40'	If module is called by C-Format Input Director.
WSWA = X' 01'	If component to be converted is the imaginary part of a complex data item.

Exit Conditions

Normal exit. Return to caller.

Routines Called

IHEDMA	Arithmetic Conversion Director
IHEDNC	Arithmetic to Character String
IHEUPA	Zero Real or Imaginary Part
IHEVCA	Data Analysis Routine
IHEVSC	Character String to Character String

Global Variables

WCNP
WCN1
WFCB
WFED
WSDV
WSWA
WTEMP

TITLE: A-FORMAT INPUT DIRECTOR (IHEDIB)

Program Definition

Purpose and Usage

The A-Format Input Director supervises the conversion necessary during edit-directed stream I/O to convert an external A-format data item (described by an A-format) to any internal data representation (specified by a DED) in the data stream.

Description

1. FED field width w is tested as follows.

   a. If w is less than or equal to zero, test target type.

      (1) An arithmetic target type is considered an error.
      (2) If target is a character string, call Character String to Character String routine (IHEVSC) to effect transfer of source string to target.

   b. If w is greater than zero, test target type.

      (1) If target type is character string, call Character String to Character String routine (IHEVSC) to effect transfer of source string to target.
      (2) If target type is arithmetic, call the Character String to Arithmetic routine (IHEDCN) to effect conversion from source string to target.

2. IHEIOG is called to obtain data from buffer. WAFORMAT is set to X'FF' before the call to allow IHEIOG to accept a carrier return as a valid input character. WAFORMAT is reset to X'00' on return.

3. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level three.

Local Variables

PLIST	Relocatable library work area

Program Interface

Entry Points

IHEDIBA - Entry for A-Format
   P7 = A (File Control Block)

where File Control Block:

A (Buffer)
A (Current Buffer)
A (Target/Target Dope Vector)
A (Target DED)
A (Real FED)

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

IHEDCN	Character String to Arithmetic
IHEIOG	Get Data Field from Input Buffer
IHEVSC	Character String to Character String

Global Variables

WFCB
WSDV
WTEMP
WAFORMAT

Comments

Called by compiled code.

TITLE: C-FORMAT INPUT DIRECTOR (IHEDIM)

Program Definition

Purpose and Usage

The C-Format Input Director supervises the conversion necessary during edit-directed stream I/O to convert an external data C-format data item (described by two F/E-format elements) to an internal C-format representation (specified by a DED) in the data stream.

Description

1. The C-format switch in LCA (address WSWA) is set and the target data type is tested.

   a. If data type is string, construct DED's describing the real and imaginary parts of the source complex number string. The Data Analysis Routine (IHEVCA) is used to initialize the DED. For complex components for which the F/E-Format Input Director (IHEDIA) is specified, the precision and scale factor are placed in the respective DED. If any director other than IHEDIA is specified, call the Complex External to String Director (IHEVCS) to effect the conversion. Otherwise, call the F/E-Format Input Director (IHEDIA) to effect conversion.

   b. If data type is not string, construct DED and compute precisions and scale factors as described for item a, above. Call the F/E-Format Input Director (IHEDIA) to effect conversion of source strings to target.

2. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level four and level E.

Local Variables

DOPV DEDS SWIT	Nonrelocatable library work area
PLIST RFPSV RGPSV QLIST	Relocatable library work area

Program Interface

Entry Points

IHEDIMA - Entry for C-Format
P7 = A (File Control Block)

where File Control Block:

A (Buffer)
A (Current Buffer)
A (Target/Target Dope Vector)

A (Target DED)
A (Flag Byte, Real FED)
A (Flag Byte, Imaginary FED)

Exit Conditions

Normal exit. Return to user via the link register.

Routines Called

IHEDIA	F/E-Format Input Director
IHEIOG	Get Data Field from Input Buffer
IHEVCA	Data Analysis Routine
IHEVCS	Complex External to String Director

Global Variables

WCNP
WCN1
WFCB
WRCD
WSDV
WSWA
WTEMP

Comments

Called by compiled code.

TITLE: F/E-FORMAT OUTPUT DIRECTOR (IHEDOA)

Program Definition

Purpose and Usage

The F/E-Format Output Director supervises the conversion necessary during edit-directed stream I/O to convert an internal data representation (described by a DED) to an external F/E-format data item (specified by an F/E-format element) in the data stream.

Description

1. E- or F-format output is indicated, according to the entry point by which the module is accessed. Entering at IHEDOAA specifies F-format output; IHEDOAB specifies E-format output.

2. The source data type is tested.

   a. If source is a character string, call Character String to Arithmetic routine (IHEDCN).

   b. If source is arithmetic, call Arithmetic Conversion Director (IHEDMA).

3. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level three.

Local Variables

PLIST	Relocatable library work area

Program Interface

Entry Points

```
    IHEDOAA - Entry for F-Format Output
       P7 = A (Parameter List)

    where Parameter List:

            A (Source/Source SDV)
            A (Source DED)
            A (FED)

    IHEDOAB - Entry for E-Format Output
              Arguments as for IHEDOAA

    IHEDOAZ - F-Format Output with File Control Block
            P7 = A (File Control Block)

    where File Control Block:

            A (Buffer)
            A (Current Buffer)
            A (Source/Source SDV)
            A (Source DED)
            A (Real FED)
```

IHEDOAY - E-Format Output with File Control Block
Arguments as for IHEDOAZ

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

IHEDMA	Arithmetic Conversion Director
IHEDCN	Character String to Arithmetic
IHEIOD	Output Data to the Buffer Area and Communication with CALL/360-OS

Global Variables

WBUFF
WFCB
WFDT
WSDV
WTEMP

TITLE: A-FORMAT OUTPUT DIRECTOR (IHEDOB)

Program Definition

Purpose and Usage

The A-Format Output Director supervises the conversion necessary during edit-directed stream I/O, to convert an internal data representation (described by a DED) to an external A-format data item (specified by an A-format element) in the data stream. If the internal representation is a character string, the A-format element may be implied.

Description

1. The FED's w specification is tested.

   a. If the w specification is less than or equal to zero, test the source data type. An arithmetic data type is considered an error. Otherwise, test length of character string. If a string length is less than or equal to zero, the target remains unchanged. Otherwise, set w equal to the length of the data string and set up character DED.

   b. If the w specification is greater than zero, set up character DED.

2. A target SDV is set up, and the source data type is tested.

   a. If data type is a character string, call the Character String to Character String routine (IHEVSC) to effect assignment of the source string.

   b. If data type is arithmetic, call the Arithmetic to Character String routine (IHEDNC) to effect the conversion and assignment.

3. Return is made to caller.

Errors Detected

A FORMAT WIDTH UNSPECIFIED AND LIST ITEM NOT TYPE STRING. (702)
A FORMAT WIDTH UNSPECIFIED ON INPUT. (704)

Work Area

Library work area is obtained from level three.

Local Variables

PLIST	Relocatable library work area

Program Interface

Entry Points

IHEDOBA - A(w) Format Output
P7 = A (File Control Block)

where File Control Block:

A (Buffer)
A (Current Buffer)
A (Source or Source SDV)
A (Source DED)
A (Real FED) IHEDOBA only

IHEDOBB - A-Format Output

Arguments as for IHEDOBA

Exit Conditions

Normal exit. Return to caller via the link register.
Abnormal exit. Call IHEERRB to raise error condition.

Routines Called

IHEDNC	Arithmetic to Character String
IHEVSC	Character String to Character String
IHEIOG	Get Data Field from Input Buffer

Global Variables

WBUFF
WFCB
WSDV
WTEMP

TITLE: C-FORMAT OUTPUT DIRECTOR (IHEDOM)

Program Definition

Purpose and Usage

The C-Format Output Director supervises the conversion necessary during edit-directed stream I/O to convert an internal data representation (described by DED) to an external C-format data item (specified by two F/E-format elements) in the data stream.

Description

1. The source data type is tested.

   a. If data type is string, create DED's for each part of the complex number to be output. Call the Data Analysis Routine (IHEVCA) to initialize the DED(s). Compute FED parameters as required. Test mode of source.

      (1) If source string is real, set for zero imaginary part. Output the C-complex via the specified real and imaginary output directors (IHEDOA, IHEDMA, and IHEDCN).

      (2) If source string is complex, call the Zero Real or Imaginary Part routine (IHEUPA) to delimit the real and imaginary parts of the C-format. Output the C-complex as described in item (1), above.

   b. If data type is arithmetic, compute FED parameters. Test mode of source. Output as described in items (1) and (2).

2. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level four.

Local Variables

WORK PSWT DED1 DED2 DED3 NFED PSTRG	Nonrelocatable library work area
PWRK PARAM PLIST QLIST PNOW PNXT	Relocatable library work area

Program Interface

Entry Points

IHEDOMA - Entry
  P7 = A (File Control Block)

where File Control Block:

```
        A (Buffer)
        A (Current Buffer)
        A (Source/Source SDV)
        A (Source DED)
        A (Flag Byte, Real FED)
        A (Flag Byte, Imaginary FED)
```

Exit Conditions

Normal exit. Return to user via the link register.

Routines Called

IHEUPA	Zero Real or Imaginary Part
IHEVCA	Data Analysis Routine
IHEVCS	Complex External to String Director
IHEDOA	F/E-Format Output Director

Global Variables

WCNP
WCN1
WCN2
WFCB
WORK
WRCD
WSWZ
WTEMP

TITLE: CHARACTER STRING TO ARITHMETIC (IHEDCN)

Program Definition

Purpose and Usage

Character String to Arithmetic converts a fixed-length character string containing a valid arithmetic constant or complex expression to an arithmetic target with specified scale, mode, and precision.

Description

1. Source string length is tested as follows.

   a. A null string constitutes a valid string.

   b. A string containing all blanks is considered an error.

   c. A string containing embedded blanks in data is considered an error.

   d. A string type not compatible with target data type is considered an error.

   e. A valid string is converted to target specifications via the Arithmetic Conversion Director (IHEDMA). Prior to calling IHEDMA, description parameter values required by the module must be computed. The Zero Real or Imaginary Part routine (IHEUPA) is used to supply zero real or imaginary components, as may be required by complex target data types.

2. Error exit is to IHEERRB with the error code set to IHEERRCD 54.

3. Return is made to caller.

Errors Detected

CONVERSION. (600)
ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC. (604)

Work Area

Library work area is obtained from level two.

Local Variables

TESTC	Beginning of string-scan section
ENDCN	Beginning of conversion section
WORK	Nonrelocatable library work area
PRAMS PLIST HPSAV	Relocatable library work area

Program Interface

Entry Points

```
IHEDCNA -  Initialize on source information and then convert
   P7   =  A (Parameter List)
```

where Parameter List:

A (Character SDV)
A (Character DED)
A (Target)
A (Target DED)

Exit Conditions

Normal exit. Return to caller via the link register.
Abnormal exit. Call IHEERRB to raise conversion error.

Routines Called

IHEDMA	Arithmetic Conversion Director
IHEUPA	Zero Real or Imaginary Part
IHEERR	Error Routine

Global Variables

WCN1
WCN2
WFED

TITLE: ARITHMETIC TO CHARACTER STRING (IHEDNC)

<u>Program Definition</u>

Purpose and Usage

Arithmetic to Character String converts any arithmetic source of specified scale, mode, and precision to a character string.

Description

1. The scale and mode of the source are tested.

   a. If source is real arithmetic, calls Arithmetic Conversion Director (IHEDMA) to convert real source to an F/E-format character string; then calls the Character String to Character String module (IHEVSC) to move the generated F/E-format character string to the target area.

   b. If source is complex arithmetic, makes multiple calls to the Arithmetic Conversion Director and Character String to Character String modules (IHEDMA and IHEVSC, respectively) to effect the required conversion. Calls Zero Real or Imaginary Part module (IHEUPA) as required to generate intermediate real/imaginary zero F/E-format character strings.

2. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level two.

Local Variables

PSWT EDIT DOPV	Nonrelocatable library work area
NWRK SAV1 SAV2 SAV3 PLIST	Relocatable library work area

<u>Program Interface</u>

Entry Points

IHEDNCA - Entry
   P7 = A (Parameter List)

where Parameter List:

A (Source)
A (Source DED)
A (Target Dope Vector)
A (Target DED)

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

IHEDMA	Arithmetic Conversion Director
IHEUPA	Zero Real or Imaginary Part
IHEVSC	Character String to Character String

Global Variables

WCFD
WFDT
WINT
WRCD
WSWA
WTEMP

TITLE: ZERO REAL OR IMAGINARY PART (IHEUPA)

Program Definition

Purpose and Usage

Zero Real or Imaginary Part sets the real or imaginary part of complex arithmetic data item equal to zero and moves a pointer from the real part to the imaginary part or gets the address of the imaginary part only.

Description

Entry at IHEUPAA zeroes the real part and moves pointer to end of real part. Entry at IHEUPAB zeroes the imaginary part and/or moves pointer to the imaginary part of the complex, depending upon the contents of LCA address WSWA. If WSWA contains an X'04', only an update occurs.

Errors Detected: None

Work Area

Library work area is obtained from level zero.

Local Variables

Standard relocatable and nonrelocatable library work areas

Program Interface

Entry Points

```
    IHEUPAA - Entry to zero real part of complex and move pointer
              to imaginary part of complex target
       P7 = A (Parameter List)

    where Parameter List:

               A (Real Part of Data)
               A (DED)

    IHEUPAB - Entry to zero imaginary part of complex and/or
              move pointer to end of imaginary part of complex target

              Arguments as for IHEUPAA
```

Input Parameter

    WSWA = X'04' Update Only Switch

Exit Conditions

Normal exit. Return to caller via link register. WRCD is set to the address of imaginary part of complex.

Routines Called: None

Global Variables

    WRCD
    WSWA
    WSWC

TITLE: COMPLEX EXTERNAL TO STRING DIRECTOR (IHEVCS)

Program Definition

Purpose and Usage

The Complex External to String Director directs the conversion of character representation of complex data to internal string data. The character data is first converted to complex with attributes derived from the real and imaginary parts of the source data (according to arithmetic conversion package rules) and then converted to string.

Description

Compute the scale and precision of the resulting complex as follows:

$s = \max(q1, q2)$
$p = \max(p1 - q1, p2 - q2) + s + 1$

where (p1,q1) and (p2,q2) denote the precision and scale of the real and complex components of the source complex, respectively.

Set scale equal to 128+s.

1. If $m \leq p \leq n$ (where m and n denote the minimum and maximum precision values possible for the mode, respectively), set the precision equal to p.
2. If p is less than m, set precision to m.
3. If p is greater than n, set precision equal to n.

Test caller's point of entry.

1. If entered via a call to IHEVCSA, call the Arithmetic Conversion Director (IHEDMA) to output F/E-format data strings. Two calls to IHEDMA are required, one for the real component and one for the imaginary. The Zero Real or Imaginary Part routine (IHEUPA) is used to locate address of the imaginary part of the complex source.
2. If entered via a call to IHEVCSB, a single call to the Character String to Character String routine (IHEVSC) effects the conversion and transfer of the source to target.

Errors Detected

None

Work Area

Library work area is obtained from level three.

Local Variables

DED1 DED2 PSWT ENTY INTR	Nonrelocatable library work area
PRMS PLIST	Relocatable library work area

Program Interface

Entry Points

IHEVCSA - Complex external to string conversion
P7 = A (Parameter List)

where Parameter List:

A (Start/End Addresses of Real Data)
A (Real DED)
A (Start/End Addresses of Imaginary Data)
A (Imaginary DED)
A (Target Dope Vector)
A (Real FED)
A (Imaginary FED)

IHEVCSB - Complex external to coded complex conversion
P7 = A (Parameter List)

where Parameter List:

A (Start/End Addresses of Real Data)
A (Real DED)
A (Start/End Addresses of Imaginary Data)
A (Imaginary DED)
A (Target)
A (Target DED)
A (Real FED)
A (Imaginary FED)

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

IHEDMA	Arithmetic Conversion Director
IHEDNC	Arithmetic to Character String
IHEUPA	Zero Real or Imaginary Part

Global Variables

WFED
WRCD
WSWA

TITLE: CHARACTER STRING TO CHARACTER STRING (IHEVSC)

Program Definition

Purpose and Usage

Character String to Character String assigns a fixed or varying length character string to a fixed or varying length character string.

Description

1. The length of the source string is tested.

   a. If source string length is greater than or equal to target string length, set source string length to target string length.

   b. If source string length is less than target string length, compute the blank fill required to pad target string.

2. Source string is moved to target string.

3. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

Standard relocatable and nonrelocatable library work areas.

Program Interface

Entry Points

```
    IHEVSCA - Entry
       P7 = A (Parameter List)

    where Parameter List:

       A (Source SDV)
       A (Source DED)
       A (Target SDV)
       A (Target DED)
```

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

None

Global Variables

None

TITLE: ARITHMETIC CONVERSION DIRECTOR (IHEDMA)

Program Definition

Purpose and Usage

The Arithmetic Conversion Director sets up the intermodular flow to effect conversion from one arithmetic data type to another. (See Figure 4-6 in the general discussion of the Total Conversion Package.)

Description

This module sets up one of two transfer vector patterns, depending upon the source and target formats. They are:

1. Type 1

   a. Source to intermediate (PDI or FLI)
   b. Intermediate (PDI or FLI) to intermediate (FLI or PDI)
   c. Intermediate (FLI or PDI) to target

2. Type 2

   a. Source to intermediate (PDI or FLI)
   b. No operation
   c. Intermediate (PDI or FLI) to target

Note: PDI refers to Packed Decimal Intermediate. FLI refers to Float Intermediate (a long-precision number).

The execution of the first module in the chain is effected through a direct transfer from module IHEDMA. Subsequent executions are effected indirectly through the setting of appropriate pointers in the LCA.

The first routine to be executed is determined by inspecting the DED describing the source data item. The third routine is determined by inspection of the DED describing the target data item. The selection of the second routine, if required, is determined by the need to convert from one intermediate format to another in order to go from source to target format.

Acceptable DED flag byte patterns follow in hexadecimal format:

7E E-format	Reserved for use by other total conversion directors
7F F-format	
8C-8D Fixed-point binary	
8E-8F Short floating-point binary	
9E-9F Long floating-point binary	
C8-C9 F-format character string	
CA-CB E-format character string (single precision)	
DA-DB E-format character string (double precision)	

Possible source to intermediate routines are:

1. F/E-Format to PDI (IHEVPE)
2. Float to FLI (IHEVFE)
3. Fixed Binary to FLI (IHEVFD)

Possible intermediate to target routines are:

1. PDI to F-Format (IHEVPB)
2. PDI to E-Format (IHEVPC)
3. FLI to Float (IHEVFC)
4. FLI to Fixed Binary (IHEVFB)

Possible intermediate conversion modules are:

1. PDI to FLI (IHEVPA)	The modules require the use of routine IHEVTB which is a radix conversion table.
2. FLI to PDI (IHEVFA)	

The last routine in the chain transfers control back to caller of module IHEDMA via the link register.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

AC01	Length of the table containing the list of binary processors
AT01	Table of target processor pointers
AT21	Table of source processor pointers
XT01	Table of function index values
AT11	Table of base-conversion processor pointers

<u>Program Interface</u>

Entry Points

```
    IHEDMAA
       P7 = A (Parameter List)

    where Parameter List:

               A (Source)
               A (Source DED)
               A (Target)
               A (Target DED)
```

Input Parameters

```
    WFED = A (Input FED)
    WFDT = A (Output FED)
```

Exit Conditions

Normal exit. Transfer control to IHEVFE, IHEVPE, or IHEVFD. This module sets WRB1 to address of second routine, WRB2 to address of third routine, and WRDC to contents of target and target DED.

Routines Called

IHEVPE	String with Format to Packed Decimal Intermediate
IHEVFD	Fixed Binary to Float Intermediate
IHEVFE	Float Source to Float Intermediate

Global Variables

    WRCD

Comments

Called by:

Compiled Code
IHEDIA (F/E-Format Input Director)
IHEDIB (A-Format Input Director)
IHEDIM (C-Format Input Director)
IHEDOA (F/E-Format Output Director)
IHEDOB (A-Format Output Director)
IHEDOM (C-Format Output Director)
IHEDCN (Character String to Arithmetic)
IHEDNC (Arithmetic to Character String)
IHEVCS (Complex External to String Director)

TITLE: FLOAT INTERMEDIATE TO PACKED DECIMAL INTERMEDIATE (IHEVFA)

Program Definition

Purpose and Usage

Float Intermediate to Packed Decimal Intermediate directs the conversion of a floating-point intermediate number to a packed decimal intermediate number and stores the result into the library communications area (LCA) to make it available to the routine scheduled next for execution by the Arithmetic Conversion Director. (Intermediate arithmetic values are long-precision numbers.)

Description

The long floating-point number currently residing in the LCA address WINT is converted to a packed decimal number with scale factor and stored into WINT and WSCF, respectively.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

WORK	Nonrelocatable library work area

Program Interface

Entry Points

IHEVFAA

Input Parameters

WINT = Long floating-point intermediate number
WBR2 = A (Next routine entry point)

Exit Conditions

Normal exit. A branch is made to the address contained in LCA address WBR2. This routine sets WINT to seventeen-digit (nine-byte) packed decimal number and WSCF to scale factor in a binary word.

Routines Called

None

Global Variables

WBR2
WINT
WSCF
WSWC

TITLE: FLOAT INTERMEDIATE TO FIXED BINARY (IHEVFB)

Program Definition

Purpose and Usage

Float Intermediate to Fixed Binary assists in arithmetic and/or string manipulations.

Description

Floating-point intermediate numbers are converted to fixed-point binary and stored in a target item.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

Standard relocatable and nonrelocatable library work areas

Program Interface

Entry Points

    IHEVFBA

Input Parameters

    WRCD = A (Target Field), A (Target DED)
    WINT = Long Floating-Point Intermediate Number

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

    WINT
    WRCD

TITLE: FLOAT INTERMEDIATE TO FLOAT SHORT OR LONG (IHEVFC)

Program Definition

Purpose and Usage

The Float Intermediate to Float Short or Long routine moves a floating-point intermediate into a floating-point short or long target data item. (Intermediate arithmetic values are long-precision numbers.)

Description

If the target item is a short floating-point data item, the floating-point intermediate is truncated to short floating-point binary and moved into the short floating-point target data item. However, if the target item is a long floating-point data item, the floating-point intermediate is moved directly into the target item.

Errors Detected: None

Work Area

Library work area is obtained from level zero.

Local Variables

Standard relocatable and nonrelocatable library work areas

Program Interface

Entry Points

IHEVFCA

Input Parameters

WINT = Long Floating-Point Intermediate Number
WRCD = A (Target), A (Target DED)

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called: None

Global Variables

WINT
WRCD

TITLE: FIXED BINARY TO FLOAT INTERMEDIATE (IHEVFD)

Program Definition

Purpose and Usage

Fixed Binary to Float Intermediate assists in arithmetic and/or character string manipulations.

Description

A fixed-point binary source is converted to a floating-point intermediate number.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

WORK	Nonrelocatable library work area

Program Interface

Entry Points

```
    IHEVFDA
       P7 = A (Parameter List)

    where Parameter List:

       A (Source Data)
       A (Source DED)
```

Input Parameters

```
    WBR1 = A (Next Module Entry Point)
```

Exit Conditions

Normal exit. A branch is made to address in WBR1. WINT is set to a long floating-point intermediate number.

Routines Called

None

Global Variables

WBR1
WINT
WRCD
WSCF
WSWC

TITLE: FLOAT SOURCE TO FLOAT INTERMEDIATE (IHEVFE)

Program Definition

Purpose and Usage

The Float Source to Float Intermediate routine moves a short or long floating-point binary number into the library communications area (LCA) to make it available for use as a floating-point intermediate by routines scheduled for subsequent execution by the Arithmetic Conversion Director. (Intermediate arithmetic values are long-precision numbers.)

Description

If a short floating-point number is input, it is expanded to a long floating-point number and placed into the LCA at address WINT; however, if a long floating-point number is input, it is simply moved into the LCA as previously noted.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

Standard relocatable and nonrelocatable library work areas

Program Interface

Entry Points

```
    IHEVFEA
       P7 = A (Parameter List)

    where Parameter List:

               A (Source Data)
               A (Source DED)
```

Input Parameters

```
    WBR1 = A (Next Module Entry Point)
```

Exit Conditions

Normal exit. A branch is made to the address contained in LCA address WBR1. WINT is set to a long floating-point intermediate number.

Routines Called

None

Global Variables

```
    WBR1
    WINT
```

TITLE: PACKED DECIMAL INTERMEDIATE TO FLOAT INTERMEDIATE (IHEVPA)

Program Definition

Purpose and Usage

The Packed Decimal Intermediate to Float Intermediate routine converts a packed decimal intermediate number to a long floating-point intermediate number and stores the result into the library communications area (LCA) to make it available to the routine scheduled next for execution by the Arithmetic Conversion Director. (Intermediate arithmetic values are long-precision numbers.)

Description

The packed decimal number with scale factor currently residing in the LCA addresses WINT and WSCF, respectively, is converted to a long floating-point intermediate number and stored back into LCA address WINT.

Errors Detected

None

Work Area

Library work area is obtained from level one.

Local Variables

WORK	Nonrelocatable library work area

Program Interface

Entry Points

IHEVPAA

Input Parameters

WINT = Seventeen-digit (nine-byte) packed decimal number
WSCF = Fixed-point scale factor for WINT
WBR2 = A (Next module entry point)

Exit Conditions

Normal exit. A branch is made to the address contained in WBR2. WINT is set to a long floating-point intermediate number.

Routines Called

None

Global Variables

WBR2
WINT
WRP
WSCF
WSWC

TITLE: PACKED DECIMAL INTERMEDIATE TO F-FORMAT (IHEVPB)

Program Definition

Purpose and Usage

The Packed Decimal Intermediate to F-Format routine converts a packed decimal intermediate number to an F-format character string and stores it into a target string data item. (Intermediate arithmetic values are long-precision numbers.)

Description

The packed decimal intermediate number with scale factor contained in library communications area (LCA) addresses WINT and WSCF, respectively, is converted to a character string according to an F-format element pointed to by LCA address WFDT. The generated string is stored into the string data item specified in LCA address WRCD.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

WORK	Nonrelocatable library work area

Program Interface

Entry Points

IHEVPBA

Input Parameters

WINT = Seventeen-digit (nine-byte) packed decimal number
WSCF = Fixed-point scale factor of WINT
WFDT = A (FED)
WRCD = A (Target), A (Target DED)

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

None

Global Variables

WFDT
WINT
WRCD
WRP
WSCF
WSWC
WTEMP

TITLE: PACKED DECIMAL INTERMEDIATE TO E-FORMAT (IHEVPC)

Program Definition

Purpose and Usage

The Packed Decimal Intermediate to E-Format routine converts a packed decimal intermediate number to an E-format character string and stores it into a target string data item. (Intermediate arithmetic values are long-precision numbers.)

Description

The packed decimal intermediate number with scale factor contained in library communications area (LCA) addresses WINT and WSCF, respectively, is converted to a character string according to an E-format element pointed to by LCA address WFDT. The generated string is stored into the string data item specified in LCA address WRCD.

Errors Detected

    CONVERSION ERROR IN E FORMAT. (602)

Work Area

Library work area is obtained from level zero.

Local Variables

WORK	Nonrelocatable library work area

Program Interface

Entry Points

    IHEVPCA

Input Parameters

    WINT = Seventeen-digit (nine-byte) packed decimal number
    WSCF = Fixed-point scale factor of WINT
    WFDT = A (FED)
    WRCD = A (Target), A (Target DED)

Exit Conditions

Normal exit. Return to caller via link register.

Abnormal exit. Call IHEERRB to raise FED error condition, and IHEERRCD 53 is given.

Routines Called

None

Global Variables

    WFDT
    WINT
    WRCD
    WRP
    WSCF
    WSWC
    WTEMP

TITLE: STRING WITH FORMAT TO PACKED DECIMAL INTERMEDIATE (IHEVPE)

Program Definition

Purpose and Usage

The String with Format to Packed Decimal Intermediate routine converts a character string paired with an F/E-format element to packed decimal intermediate and stores it into the library communications area (LCA) for use by the next routine scheduled for execution by the Arithmetic Conversion Director. (Intermediate arithmetic values are long-precision numbers.)

Description

The source character string described by a DED is paired with an F/E-format element descriptor and converted accordingly to a packed decimal intermediate number with scale factor. The F/E-format element descriptor is contained in LCA address WFED. The packed decimal intermediate result is stored into LCA addresses WINT and WSCF, respectively, for use by the next routine scheduled for execution by the Arithmetic Conversion Director.

Errors Detected

    CONVERSION. (600)

Work Area: Library work area is obtained from level zero.

Local Variables

PWRK	Relocatable library work area

Program Interface

Entry Points

```
    IHEVPEA
       P7 = A (Parameter List)

    where Parameter List:

            A (Source Field)
            A (Source DED)
```

Input Parameters

    WBR1 = A (Next module entry point)

Exit Conditions

Normal exit. A branch is made to address contained in LCA address WBR1. WINT is set to 17-digit (nine-byte) packed decimal number. WSCF is set to the fixed-point scale factor of WINT.

Abnormal exit. Call IHEERRB to raise error condition if the source field is invalid, and IHEERRCD 51 is given.

Routines Called: None

Global Variables

WBR1	WINT	WRP
WFED	WSWC	WSCF

TITLE: TABLE OF POWERS OF TEN (IHEVTB)

Program Definition

Purpose and Usage

The Table of Powers of Ten is used by the two radix conversion routines, IHEVPA and IHEVFA.

Description

This module is a table of long-precision floating-point numbers representing powers of 10 from 1 to 70.

Errors Detected

None

Work Area

None

Local Variables

None

Program Interface

Base Address

IHEVTBA

Exit Conditions

None

Routines Called

None

Global Variables

None

TITLE: DATA ANALYSIS ROUTINE (IHEVCA)

Program Definition

Purpose and Usage

The Data Analysis Routine creates a DED (flag byte, p-byte, q-byte) to describe the scale, mode, and precision of a character representation of an arithmetic value.

Description

Gets start/stop string delimiters from LCA address WCNP. Scans character string for the following characters:

.	0	5
E	1	6
+	2	7
-	3	8
I	4	9

1. If designated characters or digits are encountered, appropriate construction of the DED block is accomplished.
2. If other characters are encountered, the scan continues, ignoring such characters in the calculation of the DED parameters; however, said characters remain embedded in the string.
3. Return is made to caller.

Errors Detected

None

Work Area

Library work area is obtained from level zero.

Local Variables

CODE	Nonrelocatable library work area

Program Interface

Entry Points

IHEVCAA - Entry
P7 = A (DED)

Input Parameters

WCNP = A (Start/end address pair delimiting character string)

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

None

Global Variables

WCNP

STRING MANIPULATION PACKAGE

The following routines constitute the String Manipulation Package (SIMP). Descriptions of these routines are given on succeeding pages of this manual. The routines are discussed in alphabetic order, according to their mnemonics, as indicated.

Character String Compare (IHECSC)
Character String Assignment (IHECSM)
Character String SUBSTR (IHECSS)

Execution times given in this subsection are based on information in IBM System/360 Instruction Timing Information (A22-6825). They are intended to be simple enough to give a good general guide to performance while averaging out many logical variations, the effects of which are relatively small.

TITLE: CHARACTER STRING COMPARE (IHECSC)

Program Definition

Purpose and Usage

Character String Compare is used to compare two character strings and return the resulting condition code.

Description

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.

The condition code is returned as bits 2 and 3 of a fullword target field as follows:

00	If strings are equal
01	If the 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 target field is used to preserve the rightmost 32 bits in the PSW for the calling routine. The second word of the PSW contains:

Bits	Contents
0 to 1	Instruction length code 01
2 to 3	Condition code as above
4 to 7	Program mask (calling routine)
8 to 32	An address in IHECSC

Implementation:

- Module size: 364 bytes
- Execution times:

  Let

  L1 = the length of the strings compared up to the first inequality (proceeding left to right)

  L2 = the length of the additional part of the longer string compared with blanks if necessary

  Mi = FLOOR((Li-1)/256)

  Ni = MOD(Li-1,256)

  S = SIGN(L2)

  where 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*M1 + c*N1 + S*(d + e*M2 + c*N2)

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

<u>Program Interface</u>

Entry Points

```
IHECSC0
   P7       = A(PLIST)
   PLIST    = A(SDV of first operand)
              A(SDV of second operand)
              A(target)
```

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

None

Global Variables

None

<u>Comments</u>

Called by compiled code.

TITLE: CHARACTER STRING ASSIGNMENT (IHECSM)

Program Definition

Purpose and Usage

Character String Assignment is used to assign a character string to a fixed-length target, with blank padding as required.

Description

Method:

The minimums of the source length and the target length are calculated. If the source and target lengths are equal, the source is moved to the target. If the source length is longer than the target length, the source is truncated to fit and moved to the target; if the source length is shorter than the target length, the source is moved to the target, and the remainder of the target is filled with blanks.

Implementation:

- Module size: 276 bytes
- Execution times:

Let

```
L1 = length for blank filling
L2 = length of the shorter of the source and target fields
M1 = FLOOR((L1-2)/256)
N1 = MOD(L1-2,256)        where L1 ≥ 2 for both M1 and N1
M2 = FLOOR((L2-1)/256)
N2 = MOD(L2-1,256)
 S = SIGN(L1)
```

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

h + g*M2 + c*N2 + S*(i + b*M1 + c*N1)

	IBM System/360 Model Number				
	30	40	50	65	75
b	1171	696	316	107	95.2
c	4	2.5	1.1	0.4	0.3
g	1196	706	319	108	95.8
h	1298	446	190	57.4	40.6
i	400	129	57.6	18.5	13.7

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHECSMF
       P7      = A(PLIST)
       PLIST   = A(SDV of source string)
                 A(SDV of target field)
```

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: CHARACTER STRING SUBSTR (IHECSS)

Program Definition

Purpose and Usage

Character String SUBSTR is used to produce a string dope vector describing the SUBSTR pseudo-variable and function of a character string.

Description

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: 252 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	887	310	127	36.9	26.2

Errors Detected: None

Local Variables: None

Program Interface

Entry Points

```
IHECSS2 - SUBSTR (Character-string, i)
   P7      = A(PLIST)
   PLIST   = A(SDV source string)
             A(I)
             Dummy Argument
             A(Field for target SDV)
```

Exit Conditions

Normal exit. Return to caller via the link register.

Routines Called: None

Global Variables: None

Comments

Called by compiled code.

ARITHMETIC FUNCTION PACKAGE

The following routines constitute the Arithmetic Function Package (AFUNC). As noted in the previous section, these routines support the ABS, MAX, and MIN built-in functions and multiplication, division, integer exponentiation, and general exponentiation operations. The particular routines in each group are listed below and explained on succeeding pages of this manual.

Note that, as far as possible, the general groupings are listed in alphabetic order according to the mnemonics of the routines. Within each grouping, discussion of short floating-point precedes discussion of long floating-point, and discussion of real precedes discussion of complex.

Support of built-in functions:

- Binary Fixed Complex ABS (IHEABU)
- Short Float Complex ABS (IHEABW)
- Long Float Complex ABS (IHEABZ)
- Real Binary Fixed MAX/MIN (IHEMXB)
- Real Short Float MAX/MIN (IHEMXS)
- Real Long Float MAX/MIN (IHEMXL)

Support of multiplication and division:

- Short Float Complex Division (IHEDZW)
- Long Float Complex Division (IHEDZZ)
- Binary Fixed Complex Mult/Div (IHEMZU)
- Short Float Complex Mult (IHEMZW)
- Long Float Complex Mult (IHEMZZ)

Support of integer exponentiation:

- Real Fixed Binary Integer EXP (IHEXIB)
- Real Short Float Integer EXP (IHEXIS)
- Real Long Float Integer EXP (IHEXIL)
- Z**N, Z Fixed Binary Complex (IHEXIU)
- Z**N, Z Short Float Complex (IHEXIW)
- Z**N, Z Long Float Complex (IHEXIZ)

Support of general exponentiation:

- Short Float Real General EXP (IHEXXS)
- Long Float Real General EXP (IHEXXL)
- Short Float Complex General EXP (IHEXXW)
- Long Float Complex General EXP (IHEXXZ)

SPEED

The average execution times given in this subsection are based on information in _IBM System/360 Instruction Timing Information_ (GA22-6285) and include the times required by the modules called.

ACCURACY

Statistics for accuracy of floating-point modules are given where considered meaningful and helpful; an explanation of their use is given under "Accuracy" in the subsection entitled "Mathemtical Function Package" which follows. Precise results are obtained from all fixed-point modules except complex division and complex ABS, where small truncation errors inevitably occur.

ARGUMENTS

Any restrictions on arguments are noted under two headings:

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

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

P - Programmed conditions in the module concerned. Programmed tests are made where not too costly and, if an invalid argument is found, a branch is taken to the entry point IHEERRB of Error Routine. (See "Handling of Interrupts Package." Error Routine is part of EXEP, which is a subpackage of HIP.) An appropriate message is printed and the ERROR condition is 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 IHEERR and are treated in the normal way. The machine is assumed to be enabled for all interruptions except significance, which is masked.

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.

TITLE: BINARY FIXED COMPLEX ABS (IHEABU)

Program Definition

Purpose and Usage

Binary Fixed Complex ABS is used to calculate

ABS(z) = SQRT(x**2 + y**2)

where z = x + yI and x and y are binary fixed-point real expressions.

Description

Method:

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

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

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 ≤ 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.

Implementation:

- Module size: 324 bytes
- Execution times:

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

  a = 2**(15-q)

	IBM System/360 Model Number				
X1	30	40	50	65	75
<a	3809	1218	320	79.4	52.3
≥a	4601	1473	372	93.1	59.8

Errors Detected

Error and Exceptional Conditions:

I: FIXEDOVERFLOW (320)

Local Variables

None

Program Interface

Entry Points

```
IHEABG0
   P7         = A(PLIST)
   PLIST      = A(z)
                A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: SHORT FLOAT COMPLEX ABS (IHEABW)

Program Definition

Purpose and Usage

Short Float Complex ABS is used to calculate

ABS(z) = SQRT(x**2 + y**2)

where z = x + yI and x and y are short floating-point real expressions.

Description

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).

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 ≤ g ≤ 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 seven 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.

Implementation:

- Module size: 298 bytes
- Execution times:

Let (p,q) = Precision of the argument

L	=	CEIL ((p+1)/2), that is, the length of each of the real and imaginary parts of the argument.
D1	=	Maximum number of significant digits in real and imaginary parts of the argument.
D2	=	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 $D2 \le 7$:	a
$L \le 5$, $7 < D1 < 10$ and $D2 > 7$:	b + f*L + g*L**2
$5 < L \le 8$ and $D2 \le 7$:	c
$5 < L \le 8$, $7 < D1 < 10$ and $D2 > 7$:	d + f*L + g*L**2
$5 < L \le 8$ and $10 \le D1 \le 15$:	e + f*L + g*L**1

	IBM System/360 Model Number				
	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

Errors Detected

<u>Error and Exceptional Conditions</u>:

I: FIXEDOVERFLOW (320)

Local Variables

None

<u>Program Interface</u>

Entry Points

```
IHEABTO
   P7      =  A(PLIST)
   PLIST   =  A(z)
              A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

IHESQS   Short Float Real SQRT

Global Variables

None

<u>Comments</u>

Called by compiled code and IHESQW.

TITLE: LONG FLOAT COMPLEX ABS (IHEABZ)

Program Definition

Purpose and Usage

Long Float Complex ABS is used to calculate

ABS(z) = SQRT(x**2 + y**2)

where z = x + yI and x and y are long floating-point real expressions.

Description

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)

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

Implementation:

- Module size: 306 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	14318	3191	695	174	104

Errors Detected

<u>Error</u> <u>and</u> <u>Exceptional</u> <u>Conditions</u>:

    I: OVERFLOW (300)

Local Variables

None

<u>Program</u> <u>Interface</u>

Entry Points

```
    IHEABM0
       P7       =  A(PLIST)
       PLIST    =  A(z)
                   A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

IHESQL	Long Float Real SQRT

Global Variables

None

<u>Comments</u>

Called by compiled code and IHESQZ.

TITLE: REAL BINARY FIXED MAX/MIN (IHEMXB)

Program Definition

Purpose and Usage

Real Binary Fixed MAX/MIN is used to find the maximum or the minimum of a group of real fixed-point binary numbers.

Description

Method:

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. The current value is stored as the result.

Implementation:

- Module size: 272 bytes
- Execution times:

  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:

  IHEMXF0

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

  IHEMNF0

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

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHEMXF0 - MAX(x1,x2,...,xn)
              where x1,x2,...,xn are real fixed-point binary expressions.

       P7      = A(PLIST)
       PLIST   = A(x1)
                 A(x2)
                 .
                 .
                 .
                 A(xn)
                 A(Target)

    IHEMNF0 - MIN(x1,x2,...,xn)
              where x1,x2,...,xn are real fixed-point binary expressions.

    Linkage same as for IHEMXF0
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: REAL SHORT FLOAT MAX/MIN (IHEMXS)

Program Definition

Purpose and Usage

Real Short Float MAX/MIN is used to find the maximum or the minimum of a group of short floating-point real expressions.

Description

Method:

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. The current value is stored as the result.

Implementation:

- Module size: 272 bytes
- Execution times:

  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:

  IHEMXS0

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

  IHEMNS0

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

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHEMXS0 - MAX(x1,x2,...,xn)
              where x1,x2,...,xn are real short floating-point expressions.

       P7       = A(PLIST)
       PLIST    = A(x1)
                  A(x2)
                    .
                    .
                    .
                  A(xn)
                  A(Target)

       (High-order bit of last argument indicates end of parameter list.)

    IHEMNS0 - MIN(x1,x2,...,xn)
              where x1,x2,...,xn are real short floating-point expressions.

    Linkage as for IHEMXS0
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: REAL LONG FLOAT MAX/MIN (IHEMXL)

Program Definition

Purpose and Usage

Real Long Float MAX/MIN is used to find the maximum or minimum of a group of long floating-point real expressions.

Description

Method:

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. The current value is stored as the result.

Implementation:

- Module size: 272 bytes
- Execution times:

  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:

IHEMXLO

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

IHEMNLO

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

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

<u>Program</u> <u>Interface</u>

Entry Points

```
    IHEMXL0 - MAX(x1,x2,...,xn)
              where x1,x2,...,xn are real long floating-point expressions.

       P7      = A(PLIST)
       PLIST   = A(x1)
                 A(x2)
                 .
                 .
                 .
                 A(xn)
                 A(Target)

       (High-order bit of last argument indicates end of parameter list.)

    IHEMNL0 - MIN(x1,x2,...,xn)
              where x1,x2,...,xn are real long floating-point expressions.

    Linkage same as for IHEMXL0
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

<u>Comments</u>

Called by compiled code.

TITLE: SHORT FLOAT COMPLEX DIVISION (IHEDZW)

Program Definition

Purpose and Usage

Short Float Complex Division is used to compute z1/z2 in floating point, when z1 = a + bI and z2 = c + dI, and a, b, c, and d are short floating-point real expressions.

Description

Method:

1. ABS(c) ≥ ABS(d)

   Compute

   q = d/c

   Then

   REAL(z1/z2) = (a + b*q)/(c + d*q)
   IMAG(z1/z2) = (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.

Implementation:

- Module size: 252 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	3546	875	221	60.8	35.7

Errors Detected

Error and Exceptional Conditions:

I: OVERFLOW (300)
ZERODIVIDE (330)
UNDERFLOW (340)

Local Variables

None

Program Interface

Entry Points

```
    IHEDZT0
       P7      = A(PLIST)
       PLIST   = A(z1)
                 A(z2)
                 A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: LONG FLOAT COMPLEX DIVISION (IHEDZZ)

Program Definition

Purpose and Usage

Long Float Complex Division is used to compute z1/z2 in floating-point, when z1 = a + bI and z2 = c + dI, and a, b, c, and d are long floating-point real expressions.

Description

Method:

1. ABS(c) ≥ ABS(d)

   Compute

   $q = d/c$

   Then

   REAL(z1/z2) = (a + b*q)/(c + d*q)
   IMAG(z1/z2) = (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.

Implementation:

- Module size: 252 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	11741	2515	234	92.5	51.1

Errors Detected

Error and Exceptional Conditions:

I: OVERFLOW (300)
ZERODIVIDE (330)
UNDERFLOW (340)

Local Variables

None

Program Interface

Entry Points

```
    IHEDZM0
       P7      = A(PLIST)
       PLIST   = A(z1)
                 A(z2)
                 A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: BINARY FIXED COMPLEX MULT/DIV (IHEMZU)

Program Definition

Purpose and Usage

Binary Fixed Complex Mult/Div is used to calculate z1*z2 or z1/z2, where z1 and z2 are fixed-point binary complex expressions.

Description

Method:

Let

z1 = a + bI
z2 = 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.

Implementation:

- Module size: 488 bytes

- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	2421	689	256	56.6	37.1

  Let

  M = number of significant bits in z2*CONJG(z2)
  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

	IBM System/360 Model Number				
	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 $M \leq 31$
$c = d = 0$ if $M \leq 32$

Errors Detected

Error and Exceptional Conditions:

I: FIXEDOVERFLOW (320) in either routine
ZERODIVIDE (330) in the division routine

Local Variables

None

Program Interface

Entry Points

```
IHEMZG0 - The product z1*z2
   P7      = A(PLIST)
   PLIST   = A(z1)
             A(DED for z1)
             A(z2)
             A(DED for z2)
             A(Target)

IHEDZG0 - The quotient z1/z2

          Linkage as for IHEMZG0

   P7      = A(PLIST)
   PLIST   = A(z1)
             A(DED for z1)
             A(z2)
             A(DED for z2)
             A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called: None

Global Variables: None

Comments

IHEMZG0 and IHEDZG0 called by compiled code. IHEMZG0 called also by IHEXIU.

TITLE: SHORT FLOAT COMPLEX MULT (IHEMZW)

Program Definition

Purpose and Usage

Short Float Complex Mult is used to compute z1*z2 in floating point, when z1 = a + bI and z2 = c + dI, and a, b, c, and d are short floating-point real expressions.

Description

Method:

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

Implementation:

- Module size: 220 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	1979	550	172	41.9	23.3

Errors Detected

Error and Exceptional Conditions:

I: Exponent OVERFLOW (300)
   Exponent UNDERFLOW (340)

Local Variables: None

Program Interface

Entry Points

IHEMZT0 - Computes z1/z2 when z1 = a + bI and z2 = c + dI, and a, b, c, and d are all short floating-point real expressions.

```
P7      = A(PLIST)
PLIST   = A(z1)
          A(z2)
          A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called: None

Global Variables: None

Comments: Called by compiled code and IHEXIW.

TITLE: LONG FLOAT COMPLEX MULT (IHEMZZ)

Program Definition

Purpose and Usage

Long Float Complex Mult is used to compute z1*z2 in floating point, when z1 = a + bI and z2 = c + dI, and a, b, c, and d are long floating-point real expressions.

Description

Method:

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

Implementation:

- Module size: 220 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	5115	1307	251	62.3	31.3

Errors Detected

Error and Exceptional Conditions:

I: Exponent OVERFLOW (300)
   Exponent UNDERFLOW (340)

Local Variables: None

Program Interface

Entry Points

IHEMZM0 - Computes z1/z2 when z1 = a + bI and z2 = c + dI, and a, b, c, and d are long floating-point real expressions.

```
P7      = A(PLIST)
PLIST   = A(z1)
          A(z2)
          A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called: None

Global Variables: None

Comments: Called by compiled code and IHEXIZ.

TITLE: REAL FIXED BINARY INTEGER EXP (IHEXIB)

Program Definition

Purpose and Usage

Real Fixed Binary Integer EXP is used to calculate x to the power n (x**n), where x is a real fixed-point binary number and n is a positive integer.

Description

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 CALL/360-OS PL/I impose a further restriction: 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: 244 bytes
- Execution times:

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

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHEXIFI
       P7      = A(PLIST)
       PLIST   = A(x)
                 A(DED for x)
                 A(n)
                 A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: REAL SHORT FLOAT INTEGER EXP (IHEXIS)

Program Definition

Purpose and Usage

Real Short Float Integer EXP is used to calculate x to the power n (x**n), where n is an integer between -2**31 and 2**31 - 1 inclusive, and x is a short floating-point real expression.

Description

Method:

If the exponent is zero and the argument nonzero, the result 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.

RMS relative error/exponent $*10^6$	Maximum relative error/exponent $*10^6$
0.00871	0.692

Implementation:

- Module size: 294 bytes
- Execution times:

  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

	IBM System/360 Model Number				
	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

Other Information:

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, IHEXXS will not accept a negative first argument. 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.

Errors Detected

Error and Exceptional Conditions:

P: $x = 0$ with $n \leq 0$ (209)
I: OVERFLOW (300)
UNDERFLOW (340)

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.

Local Variables

None

Program Interface

Entry Points

```
IHEXISI
   P7        = A(PLIST)
   PLIST     = A(x)
               A(n)
               A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called: None

Global Variables: None

Comments

Called by compiled code.

TITLE: REAL LONG FLOAT INTEGER EXP (IHEXIL)

Program Definition

Purpose and Usage

Real Long Float Integer EXP is used to calculate x to the power n (x**n), where n is an integer between -2**31 and 2**31 - 1 inclusive, and x is a long floating-point real expression.

Description

Method:

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

RMS relative error/exponent $*10^{15}$	Maximum relative error/exponent $*10^{15}$
0.0995	1.73

Implementation:

- Module size: 302 bytes
- Execution times:

  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

	IBM System/360 Model Number				
	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

Errors Detected

Error and Exceptional Conditions:

P: x = 0 with n ≤ 0 (209)
I: OVERFLOW (300)
UNDERFLOW (340)

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.

Local Variables

None

Program Interface

Entry Points

```
IHEXILI
   P7       = A(PLIST)
   PLIST    = A(x)
              A(n)
              A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

None

Global Variables

None

Comments

Called by compiled code.

TITLE: Z**N, Z FIXED BINARY COMPLEX (IHEXIU)

Program Definition

Purpose and Usage

Z**N, Z Fixed Binary Complex is used to calculate z to the power n, where n is a positive integer less than 2**31, and z is a fixed-point binary number.

Description

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 Binary Fixed Complex Mult/Div (IHEMZU).

Range:

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

The precision rules of CALL/360-OS PL/I impose a further restriction: if z has a precision (p, q), this module may only be called if $n * (p + 1) - 1 \le 31$. This implies that $n \le 32/(p+1) \le 16$ for all such cases.

Implementation:

- Module size: 310 bytes
- Execution times:

  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:

  $-a + b*M + c*N$

	IBM System/360 Model Number				
	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

Errors Detected

None

Local Variables

None

Program Interface

Entry Points

```
    IHEXIGI
       P7          = A(PLIST)
       PLIST       = A(z)
                     A(DED for z)
                     A(n)
                     A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.

Routines Called

IHEMZU	Binary Fixed Complex Mult/Div

Global Variables

None

Comments

Called by compiled code.

TITLE: Z**N, Z SHORT FLOAT COMPLEX (IHEXIW)

Program Definition

Purpose and Usage

Z**N, Z Short Float Complex is used to calculate z to the power n, where n is an integer between -2**31 and 2**31 - 1 inclusive, and z is a short floating-point complex number.

Description

Method:

If the exponent is 0 and the argument nonzero, the answer 1 is returned immediately. If the exponent is nonzero, 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 (IHEMZW). Then, if the exponent was negative, the reciprocal of the result is taken; otherwise it is left unchanged.

Implementation:

- Module size: 424 bytes
- Execution times:

  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

	IBM System/360 Model Number				
	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

Errors Detected

Error and Exceptional Conditions:

P: z = 0 with n ≤ 0 (209)
I: OVERFLOW (300)
UNDERFLOW (340)
H: OVERFLOW (300) in Short Float Complex Mult (IHEMZW)
UNDERFLOW (340) in Short Float Complex Mult (INEMZW)

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.

Local Variables

None

<u>Program</u> <u>Interface</u>

Entry Points

```
    IHEXITI
       P7         = A(PLIST)
       PLIST      = A(z)
                    A(n)
                    A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

IHEMZW	Short Float Complex Mult

Global Variables

None

<u>Comments</u>

Called by compiled code.

TITLE: Z**N, Z LONG FLOAT COMPLEX (IHEXIZ)

Program Definition

Purpose and Usage

Z**N, Z Long Float Complex is used to calculate z to the power n, where n is an integer between -2**31 and 2**31 - 1 inclusive, and z is a long floating-point complex number.

Description

Method:

If the exponent is 0 and the argument nonzero, the answer 1 is returned immediately. If the exponent is nonzero, 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 (IHEMZZ). Then, if the exponent was negative, the reciprocal of the result is taken; otherwise it is left unchanged.

Implementation:

- Module size: 432 bytes
- Execution times:

  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

	IBM System/360 Model Number				
	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

Errors Detected

Error and Exceptional Conditions:

P : z =0 with n ≤ 0 (209)
I : OVERFLOW (300)
    UNDERFLOW (340)
H : OVERFLOW (300) in Long Float Complex Mult (IHEMZZ)
    UNDERFLOW (340) in Long Float Complex Mult (IHEMZZ)

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.

Local Variables

None

Program Interface

Entry Points

```
    IHEXIMI
       P7        = A(PLIST)
       PLIST     = A(z)
                   A(n)
                   A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

IHEMZZ	Long Float Complex Mult

Global Variables

None

Comments

Called by compiled code.

TITLE: SHORT FLOAT REAL GENERAL EXP (IHEXXS)

Program Definition

Purpose and Usage

Short Float Real General EXP is used to calculate x to the power y (x**y), where x and y are short floating-point real expressions.

Description

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.

Implementation:

- Module size: 308 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	9809	2861	902	236	143

Errors Detected

Error and Exceptional Conditions:

```
    P: x = 0 with y ≤ 0
    O: x < 0 with y ≠ 0 in Short Float Real Log (IHELNS)
       y*LOG(x) > 174.673 in Short Float Real EXP (IHEEXS)
```

Local Variables

None

Program Interface

Entry Points

```
    IHEXISF
       P7         = A(PLIST)
       PLIST      = A(y)
                    A(x)
                    A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

IHEEXS	Short Float Real EXP
IHELNS	Short Float Real Log

Global Variables

None

<u>Comments</u>

Called by compiled code.

TITLE: LONG FLOAT REAL GENERAL EXP (IHEXXL)

Program Definition

Purpose and Usage

Long Float Real General EXP is used to calculate x to the power y (x**y), where x and y are long floating-point real expressions.

Description

Method:

When x = 0, the result

$$x**y = 0$$

is given if $y > 0$, and an error message if $y \leq 0$. When $x \neq 0$ 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.

Implementation:

- Module size: 308 bytes
- Execution times:

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

	IBM System/360 Model Number				
	30	40	50	65	75
	30444	7453	1579	358	203

Errors Detected

Error and Exceptional Conditions:

P: $x = 0$ with $y \leq 0$
O: $x < 0$ with $y \neq 0$ in Long Float Real Log (IHELNL)
y*LOG(x) > 174.673 in Long Float Real EXP (IHEEXL)

Local Variables

None

Program Interface

Entry Points

```
IHEXILF
   P7        = A(PLIST)
   PLIST     = A(y)
               A(x)
               A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

IHEEXL	Long Float Real EXP
IHELNL	Long Float Real Log

Global Variables

None

<u>Comments</u>

Called by compiled code.

TITLE: SHORT FLOAT COMPLEX GENERAL EXP (IHEXXW)

Program Definition

Purpose and Usage

Short Float Complex General EXP is used to calculate z1**z2 where z1 and z2 are short floating-point complex expressions.

Description

Method:

When z1 = 0, the result 0 is returned if REAL(z2) > 0 and IMAG(z2) = 0. Otherwise, z1**z2 is computed as EXP(z2*LOG(z1)) with the provision that if IMAG(z1) = 0, then LOG(ABS(z1)) is calculated by a call to the real log routine, not to the complex log routine.

Implementation:

- Module size: 464 bytes
- Execution times:

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

  a = IMAG(z1) c = REAL(z1)
  b = IMAG(z2) d = REAL(z2)

	IBM System/360 Model Number				
	30	40	50	65	75
a = 0 c > 0	20606	5834	1816	480	291
a = 0 d < 0	21750	6171	1929	509	311
a ≠ 0 b = 0	27448	8022	2414	687	417
a ≠ 0 b ≠ 0	28263	8229	2576	711	424

Errors Detected

Error and Exceptional Conditions:

P: z1 = 0 with either REAL(z2) ≤ 0 or IMAG(z2) ≠ 0 (210)
O: REAL(z2*LOG(z1)) > 174.673 in Short Float Real EXP (IHEEXS)
ABS(IMAG(z2*LOG(z1))) ≤ 2**18*pi in Short Float Real Sin/Cos (IHESNS)

Local Variables

None

Program Interface

Entry Points

```
    IHEXITF
       P7         = A(PLIST)
       PLIST      = A(z1)
                    A(z2)
                    A(Target)
```

Exit Conditions

Normal exit. Return to caller via link register.
Abnormal exit. If P type error, a branch is made to IHEERRC.

Routines Called

IHELNS	Short Float Real Log
IHELNW	Short Float Complex Log
IHEEXW	Short Float Complex EXP

Global Variables

None

Comments

Called by compiled code.

TITLE: LONG FLOAT COMPLEX GENERAL EXP (IHEXXZ)

Program Definition

Purpose and Usage

Long Float Complex General EXP is used to calculate z1**z2 where z1 and z2 are long floating-point complex expressions.

Description

Method:

When z1 = 0, the result 0 is returned if REAL(z2) > 0 and IMAG(z2) = 0. Otherwise, z1**z2 is computed as EXP(z2*LOG(z1)) with the provision that if IMAG(z1) = 0, then LOG(ABS(z1)) is calculated by a call to the real log routine, not to the complex log routine.

Implementation:

- Module size: 472 bytes
- Execution times:

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

  a = IMAG(z1) c = REAL(z1)
  b = IMAG(z2) d = REAL(z2)

	IBM System/360 Model Number				
	30	40	50	65	75
a = 0 c > 0	62440	15325	3206	745	426
a = 0 d < 0	65208	16062	3366	781	450
a ≠ 0 b = 0	90418	21611	4524	1056	604
a ≠ 0 b ≠ 0	92809	22197	4623	1077	616

Errors Detected

Error and Exceptional Conditions:

P: z1 = 0 with either REAL(z2) ≤ 0 or IMAG(z2) ≠ 0 (210)
O: REAL(z2*LOG(z1)) > 174.673 in Long Float Real EXP (IHEEXL)
   ABS(IMAG(z2*LOG(z1))) ≤ 2**50*pi in Long Float Real Sin/Cos (IHESNL)

Local Variables

None

Program Interface

Entry Points

```
    IHEXIMF
       P7         = A(PLIST)
       PLIST      = A(z1)
                    A(z2)
                    A(Target)
```

Exit Conditions

Normal. Return to caller via link register.
Abnormal. If P type error, a branch is made to IHEERRC.

Routines Called

IHELNL	Long Float Real Log
IHELNZ	Long Float Complex Log
IHEEXZ	Long Float Complex EXP

Global Variables

None

Comments

Called by compiled code.

READER'S COMMENT FORM

CALL/360-OS

PL/I SM Vol. II

GY20-0568-0

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold

fold

fold

fold

• Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, STAPLE AND MAIL.

GY20-0568-0

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

IBM®

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

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

GY20-0568-0

CALL/360-OS PL/I SM Vol. II Printed in U.S.A. GY20-0568-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)