File No. S360-25
GC28-2026-4

IBM Systems Reference Library

IBM Time Sharing System
FORTRAN IV Library Subprograms

This publication describes the FORTRAN IV library subprograms provided with IBM Time Sharing System (TSS) and provides the information necessary to use the subprograms in either a FORTRAN IV or an assembler-language program.

Preface

This publication describes the FORTRAN IV mathematical, service, and input/output (I/O) subprograms for both FORTRAN and assembler-language programmers. Included is detailed information on:

- Algorithms within the mathematical subprograms
- Sizes of the subprograms
- Use of the subprograms by FORTRAN programmers
- Use of the mathematical and service subprograms by assembler language programmers
- Techniques for replacing the TSS versions of subprograms with user-written versions.

Prerequisite Publications

FORTRAN users should be familiar with:

IBM Time Sharing System: IBM FORTRAN IV, Form C28-2007.

IBM Time Sharing System: FORTRAN Programmer's Guide, Form C28-2025.

A general discussion of TSS, with descriptions of other facilities related to FORTRAN-supplied subprograms, is given in:

IBM Time Sharing System: Concepts and Facilities, Form C28-2003.

There are also references to:

IBM Time Sharing System: Command System User's Guide, Form C28-2001.

IBM Principles of Operation, Form A22-6821.

FIFTH EDITION (May 1976)

This edition revises, and makes obsolete C28-2026-3 and Technical Newsletter GN28-3182.

This edition applies to Release 2.0 of Time Sharing System/370 (TSS/370) and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing Systems – Dept. 80M, 1133 Westchester Avenue, White Plains, New York 10604.

Contents

Illustrations

Tables

Figures

Introduction

The FORTRAN IV library contains three types of subprograms: mathematical, service, and input/output (I/O). Although these subprograms are written specifically for FORTRAN programmers, they are also available to assembler-language programmers who use the correct linkage and pass the necessary information (see Appendix B). All library subprograms are written in assembler language.

The mathematical subprograms are similar to FUNCTION subprograms, because they are mathematical or computational in nature, and always return one answer (function value) to the calling program. Mathematical subprograms can be categorized by use:

1. *Direct reference,* as in reference to the sine subprogram in the statement

   X = SIN (Y)

2. *Indirect reference,* as in reference to an exponentiation subprogram in the statement

   X = Y * * I

The service subprograms correspond to a subprogram defined with a SUBROUTINE statement in a FORTRAN source program. These subprograms are called with a CALL statement or are implicitly called by the occurrence of certain situations during execution. Service subprograms test program-simulated machine indicators or perform utility functions.

The FORTRAN I/O library consists of twenty routines that link together in various ways, depending upon the function to be performed. I/O routines are not usually thought of as subprograms, because any single I/O function depends on a number of routines. Nevertheless, the FORTRAN I/O library can be thought of as three major subprograms—the Control Initialization, List Item Processor, and List Termination routines—and seventeen supporting subprograms. This categorization is based upon the fact that when control is passed from a FORTRAN program to the FORTRAN I/O library, it is always one of these three routines that receives control.

The FORTRAN I/O routines may also be categorized, by function, into language control routines and data conversion routines. These groups interact, in fulfilling an I/O request, by means of a communication and work region called the I/O Communication Routine.

Any reference by the user program to a FORTRAN IV library subprogram causes a search of SYSLIB for that program at execution time. Normally this search obtains for the user the subprogram provided as part of TSS. The user can, however, provide his own version of the subprogram, as described in Appendix A.

Section 1: Mathematical Subprograms

The two types of mathematical subprograms are directly referenced subprograms and indirectly referenced subprograms. The directly referenced subprograms are called by the object program in response to a statement of the form

X = SIN (Y)

In this statement, direct reference is made to the mathematical sine subprogram, by its entry name: SIN.

An example of indirectly referenced subprogram usage is the call on an exponentiation subprogram, made as the result of a statement of the form

X = Y * * I

In this statement, no direct reference is made to a subprogram by the FORTRAN programmer. The FORTRAN compiler determines that a subprogram is required to perform the exponentiation operation, however, and causes the object program to call the appropriate exponentiation subprogram.

The algorithms describing the method of computation of the mathematical subprograms are given in Appendix F. Other information concerning these subprograms is contained in Tables 1, 2, 3, 4, and 5 of this section and in Appendix A.

Tables 1 and 2 give this information:

Function: A brief description of the type of mathematical operation performed.

Entry Name: The mathematical subprograms contain an entry point corresponding to each name that may be *directly* referenced (such as SIN) and each name that may be *indirectly* referenced (such as CHCBGA, when raising an I * 4 integer to an I * 4 power). This column shows all entry points in the mathematical subprograms.

Definition: This column gives a mathematical equation that represents the computation. (It is not meant to represent the way the subprogram is called.) An alternative equation is given when there is another way of representing the computation in mathematical notation. For example, the square root can be represented as either

$\sqrt{x}$ or $x^{1/2}$

Argument(s): These columns describe the value(s) for which the function value is to be computed.

- Argument Number—The number of arguments (one or two) that the user must supply.
- Argument Type—The type and length of each argument. *Integer, real,* and *complex* represent the type of number; the notations *4, *8, and *16 represent the length, in bytes, of the argument.

NOTE: In FORTRAN IV, a *real* argument corresponds to the REAL*4 argument, and a *double-precision* argument corresponds to the REAL*8 argument. A *single-precision complex* argument corresponds to the COMPLEX*8 argument, and a *double-precision complex* argument corresponds to the COMPLEX*16 argument.

- Argument Range (Table 1 only)—The valid range for each argument. If an argument is not within its valid range, an error message is issued and execution of this load module is terminated. (See the Error Condition and Error Message column descriptions below.)

Function Value Returned: This column describes the function value returned by the subprogram; the notation is the same as that used for the argument type.

Error Condition: This column describes the argument ranges not allowed when using the mathematical subprogram.

Storage Estimates: This column shows the approximate number of bytes required for each mathematical subprogram: the approximate, total size of each subprogram's CSECT and PSECT. (FORTRAN IV mathematical subprograms each contain one public, read-only, reenterable CSECT and one PSECT. The length of each of the control sections is less than 4096 bytes. The subprograms are link edited, and their CSECTS are combined.)

Other Subprograms Required: Many mathematical subprograms require other mathematical subprograms to perform their function. The entry names of the other subprograms are listed in this column. (This column does not include CHCBZA, which is called by all mathematical subprograms where error exit is possible.)

Routine Name: Each mathematical subprogram is assigned a routine name that is normally of no interest to FORTRAN programmers. Appendixes A and B describe use of this name.

Accuracy Figures (Table 1 only): These columns give accuracy figures for one or more representative segments within the valid argument range. The accuracy figures are based upon the assumption that the arguments are perfect (i.e., without error and, therefore, having no error-propagation effect on the answers). The only errors in the answers are those introduced by the subprograms. Information given in the accuracy-figures columns is:

Table 1. Summary of Directly Referenced Mathematical Subprograms

1	2	3	4			5	6	7		8	9	10					
Function	Entry Name	Definition	Argument(s)			Function Value Returned	Error Condition	Storage Estimates		Other Subprograms Required	Routine Name	Accuracy Figures					
			No.	Type	Range			Hex	Dec			Argument Range	Sample E/U	relative M(ε)	relative σ(ε)	absolute M(E)	absolute σ(E)
COMMON AND NATURAL LOGARITHM	CDLOG	Ln (arg) or Log_e (arg) See Note 8	1	COMPLEX * 16	arg ≠ 0 + 0i	COMPLEX * 16	Argument = 0 + 0i	1E8	488	CDABS, DLOG, DATAN2, DSQRT	CHCAP	The full range except (1 + 0i)	Note 1	2.72×10^{-16}	5.38×10^{-17}		
	CLOG	Ln (arg) or Log_e (arg) See Note 8	1	COMPLEX * 8	arg ≠ 0 + 0i	COMPLEX * 8	Argument = 0 + 0i	1D0	464	CABS, ALOG, ATAN2, SQRT	CHCAO	The full range except (1 + 0i)	Note 1	7.15×10^{-7}	1.36×10^{-7}		
	DLOG	Ln (arg) or Log_e(arg)	1	REAL * 8	arg > 0	REAL * 8	Argument ≤ 0	21A	538		CHCAF	0.5 ≤ X ≤ 1.5	U			4.60×10^{-17}	2.09×10^{-17}
												X < 0.5, X > 1.5	E	3.32×10^{-16}	5.52×10^{-17}		
	DLOG10	Log_{10} (arg)	1	REAL * 8	arg > 0	REAL * 8	Argument ≤ 0	21A	538		CHCAF	0.5 ≤ X ≤ 1.5	U			2.73×10^{-17}	1.07×10^{-17}
												X < 0.5, X > 1.5	E	3.02×10^{-16}	6.65×10^{-17}		
	ALOG	Ln (arg) or Log_e (arg)	1	REAL * 4	arg > 0	REAL * 4	Argument ≤ 0	1D0	464		CHCAE	0.5 ≤ X ≤ 1.5	U			6.85×10^{-8}	2.33×10^{-8}
												X < 0.5, X > 1.5	E	8.32×10^{-7}	1.19×10^{-7}		
												0.5 ≤ X ≤ 1.5	U			7.13×10^{-8}	2.26×10^{-8}
	ALOG10	Log_{10} (arg)	1	REAL * 4	arg > 0	REAL * 4	Argument ≤ 0	1D0	464		CHCAE	X < 0.5, X > 1.5	E	1.05×10^{-6}	2.17×10^{-7}		
EXPONENTIAL	CDEXP	e^{arg}	1	COMPLEX * 16	real arg ≤ 174.673 |imag arg| < $2^{50}\pi$	COMPLEX * 16	Real Argument > 174.673 |Imaginary Argument| ≥ $2^{50}\pi$	270	624	DEXP, DSIN, DCOS	CHCAN	$\|X_1\| \leq 1$, $\|X_2\| \leq \frac{\pi}{2}$	U	3.76×10^{-16}	1.10×10^{-16}		
												$\|X_1\| \leq 20$, $\|X_2\| \leq 20$	U	2.74×10^{-15}	9.64×10^{-16}		
	CEXP	e^{arg}	1	COMPLEX * 8	real arg 174.673 |imag arg| < $2^{18}\pi$	COMPLEX * 8	Real Argument > 174.673 |Imaginary Argument| ≥ $2^{18}\pi$	250	592	EXP, SIN, COS	CHCAM	$\|X_1\| \leq 170$, $\|X_2\| \leq \frac{\pi}{2}$	U	9.93×10^{-7}	2.67×10^{-7}		
												$\|X_1\| \leq 170$, $\frac{\pi}{2} < \|X_2\| \leq 20$	U	1.07×10^{-6}	2.73×10^{-7}		
	DEXP	e^{arg}	1	REAL * 8	arg ≤ 174.673	REAL * 8	Argument > 174.673	2C0	704		CHCAD	|X| ≤ 1	U	2.04×10^{-16}	5.43×10^{-17}		
												1 < |X| ≤ 20	U	2.03×10^{-16}	4.87×10^{-17}		
												20 < |X| ≤ 170	U	1.97×10^{-16}	4.98×10^{-17}		
	EXP	e^{arg}	1	REAL * 4	arg ≤ 174.673	REAL * 4	Argument ≥ 174.673	1A8	424	—	CHCAC	|X| ≤ 1	U	4.65×10^{-7}	1.28×10^{-7}		
												1 < |X| ≤ 170	U	4.42×10^{-7}	1.15×10^{-7}		
SQUARE ROOT	CDSQRT	$(arg)^{1/2}$ or $\sqrt{arg}$	1	COMPLEX * 16	Any	COMPLEX * 16	None	148	328	CDABS, DSQRT	CHCAT	The full range	Note 1	1.76×10^{-16}	4.06×10^{-17}		
	CSQRT	$(arg)^{1/2}$ or $\sqrt{arg}$	1	COMPLEX * 8	Any	COMPLEX * 8	None	138	312	CABS, SQRT	CHCAS	The full range	Note 1	7.00×10^{-7}	1.71×10^{-7}		
	DSQRT	$(arg)^{1/2}$ or $\sqrt{arg}$	1	REAL * 8	arg ≠ 0	REAL * 8	Negative Argument	160	352		CHCAB	The full range	E	1.06×10^{-16}	2.16×10^{-17}		
	SQRT	$(arg)^{1/2}$ or $\sqrt{arg}$	1	REAL * 4	arg ≠ 0	REAL * 4	Negative Argument	158	344		CHCAA	The full range	E	4.45×10^{-7}	8.43×10^{-8}		
ARCSINE AND ARCCOSINE	DARSIN	arcsine (arg)	1	REAL * 8	|arg| ≤ 1	REAL * 8	|Argument| > 1	288	648	DSQRT	CHCAX	-1 ≤ X ≤ +1	U	2.04×10^{-16}	5.15×10^{-17}		
	DARCOS	arccosine (arg)	1	REAL * 8	|arg| ≤ 1	REAL * 8	|Argument| > 1	288	648	DSQRT	CHCAX	-1 ≤ X ≤ +1	U	2.07×10^{-16}	7.05×10^{-17}		
	ARSIN	arcsine (arg)	1	REAL * 4	|arg| ≤ 1	REAL * 4	|Argument| > 1	1F0	496	SQRT	CHCAW	-1 ≤ X ≤ +1	U	9.34×10^{-7}	2.06×10^{-7}		
	ARCOS	arccosine (arg)	1	REAL * 4	|arg| ≤ 1	REAL * 4	|Argument| > 1	1F0	496	SQRT	CHCAW	-1 ≤ X ≤ +1	U	8.85×10^{-7}	3.19×10^{-7}		
ARCTANGENT	DATAN	arctan (arg)	1	REAL * 8	Any	REAL * 8	None	288	648		CHCBR	The full range	Note 7	2.18×10^{-16}	7.04×10^{-17}		
	DATAN2	arctan (arg_1/arg_2)	2	REAL * 8	arg ≠ 0	REAL * 8	$X_1 = X_2 = 0$	288	648		CHCBR	The full range	Note 7	2.18×10^{-16}	7.04×10^{-17}		
	ATAN	arctan (arg)	1	REAL * 4	Any	REAL * 4	None	1E8	488		CHCBQ	The full range	Note 7	1.01×10^{-6}	4.68×10^{-7}		
	ATAN2	arctan (arg_1/arg_2)	2	REAL * 4	arg ≠ 0	REAL * 4	$X_1 = X_2 = 0$	1E8	488		CHCBQ	The full range	Note 7	1.01×10^{-6}	4.68×10^{-7}		
TRIGONOMETRIC SINE & COSINE	CDSIN	sin (arg), arg in radians	1	COMPLEX * 16	|real arg| < $2^{50}\pi$ |imag arg| ≤ 174.673	COMPLEX * 16	|Real Argument| ≥ $2^{50}\pi$ |Imaginary Argument| > 174.673	340	832	DSIN, DCOS, DEXP	CHCAR	$\|X_1\| \leq 10$, $\|X_2\| \leq 1$	U	2.35×10^{-15} See Note 4	2.25×10^{-16}		
	CDCOS	cos (arg), arg in radians	1	COMPLEX * 16	|real arg| < $2^{50}\pi$ |imag arg| ≤ 174.673	COMPLEX * 16	|Real Argument| ≥ $2^{50}\pi$ |Imaginary Argument| > 174.673	340	832	DSIN, DCOS, DEXP	CHCAR	$\|X_1\| \leq 10$, $\|X_2\| \leq 1$	U	3.98×10^{-15} See Note 3	2.50×10^{-16}		
	CSIN	sin (arg), arg in radians	1	COMPLEX * 8	|real arg| < $2^{18}\pi$ |imag arg| ≤ 174.673	COMPLEX * 8	|Real Argument| ≥ $2^{18}\pi$ |Imaginary Argument| > 174.673	2F8	760	SIN, COS, EXP	CHCAQ	$\|X_1\| \leq 10$, $\|X_2\| \leq 1$	U	1.92×10^{-6} See Note 6	7.38×10^{-7}		

Table 1. Summary of Directly Referenced Mathematical Subprograms (cont.)

1 Function	2 Entry Name	3 Definition	4 Argument(s) No.	4 Argument(s) Type	4 Argument(s) Range	5 Function Value Returned	6 Error Condition	7 Storage Estimates Hex	7 Storage Estimates Dec	8 Other Subprograms Required	9 Routine Name	10 Accuracy Figures: Argument Range	Sample E/U	Relative M(ε)	Relative σ(ε)	Absolute M(E)	Absolute σ(E)
TRIGONOMETRIC SINE & COSINE (Continued)	CCOS	cos (arg), arg in radians	1	COMPLEX * 8	$\lvert$real arg$\rvert < 2^{18}\pi$ $\lvert$imag arg$\rvert \leq 174.673$	COMPLEX * 8	$\lvert$Real Argument$\rvert \geq 2^{18}\pi$ $\lvert$Imaginary Argument$\rvert > 174.673$	2F8	760	SIN, COS, EXP	CHCAQ	$\lvert X_1\rvert \leq 10, \lvert X_2\rvert \leq 1$	U	2.50×10^{-6} See Note 2	7.66×10^{-7}		
	DSIN	sin (arg), arg in radians	1	REAL * 8	$\lvert$arg$\rvert < 2^{50}\pi$	REAL * 8	$\lvert$Argument$\rvert \geq 2^{50}\pi$	2B8	696		CHCAJ	$\lvert X\rvert \leq \frac{\pi}{2}$	U	3.60×10^{-16}	4.82×10^{-7}	7.74×10^{-17}	1.98×10^{-17}
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U			1.64×10^{-16}	6.49×10^{-17}
												$10 < \lvert X\rvert \leq 100$	U			2.68×10^{-15}	1.03×10^{-15}
	DCOS	cos (arg), arg in radians	1	REAL * 8	$\lvert$arg$\rvert < 2^{50}\pi$	REAL * 8	$\lvert$Argument$\rvert \geq 2^{50}\pi$	2B8	696		CHCAJ	$0 \leq X \leq \pi$	U			1.79×10^{-16}	6.53×10^{-17}
												$-10 \leq X < 0$ $\pi < X \leq 10$	U			1.75×10^{-16}	5.93×10^{-17}
												$10 < \lvert X\rvert \leq 100$	U			2.64×10^{-15}	1.01×10^{-15}
	SIN	sin (arg), arg in radians	1	REAL * 4	$\lvert$arg$\rvert < 2^{18}\pi$	REAL * 4	$\lvert$Argument$\rvert \geq 2^{18}\pi$	1F8	504		CHCAI	$\lvert X\rvert \leq \frac{\pi}{2}$	U	1.32×10^{-6}	1.82×10^{-7}	1.18×10^{-7}	4.55×10^{-8}
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U			1.15×10^{-7}	4.64×10^{-8}
												$10 < \lvert X\rvert \leq 100$	U			1.28×10^{-7}	4.52×10^{-8}
	COS	cos (arg), arg in radians	1	REAL * 4	$\lvert$arg$\rvert < 2^{18}\pi$	REAL * 4	$\lvert$Argument$\rvert \geq 2^{18}\pi$	1F8	504		CHCAI	$0 \leq X \leq \pi$	U			1.19×10^{-7}	4.60×10^{-8}
												$-10 \leq X < 0$ $\pi < X \leq 10$	U			1.28×10^{-7}	4.55×10^{-8}
												$10 < \lvert X\rvert \leq 100$	U			1.14×10^{-7}	4.60×10^{-8}
TRIGONOMETRIC TANGENT	DTAN	tan (arg), arg in radians	1	REAL * 8	$\lvert$arg$\rvert < 2^{50}\pi$	REAL * 8	$\lvert$Argument$\rvert \geq 2^{50}\pi$ Argument too close to a Singularity (i.e., too close to an odd multiple of $\pi/2$)	2F8	760		CHCAZ	$\lvert X\rvert \leq \frac{\pi}{4}$	U	3.41×10^{-16}	6.27×10^{-17}		
												$\frac{\pi}{4} < \lvert X\rvert \leq \frac{\pi}{2}$	U	1.43×10^{-12} See Note 5	2.95×10^{-14}		
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U	2.78×10^{-13} See Note 5	7.23×10^{-15}		
												$10 < \lvert X\rvert \leq 100$	U	3.79×10^{-12} See Note 5	9.50×10^{-14}		
	DCOTAN	cotan (arg), arg in radians	1	REAL * 8	$\lvert$arg$\rvert < 2^{50}\pi$	REAL * 8	$\lvert$Argument$\rvert \geq 2^{50}\pi$ Argument too close to a Singularity (i.e., too close to a multiple of π)	2F8	760		CHCAZ	$\lvert X\rvert \leq \frac{\pi}{4}$	U	2.46×10^{-16} See Note 5	8.79×10^{-17}		
												$\frac{\pi}{4} < \lvert X\rvert \leq \frac{\pi}{2}$	U	2.78×10^{-13} See Note 5	8.61×10^{-15}		
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U	5.40×10^{-13} See Note 5	1.13×10^{-14}		
												$10 < \lvert X\rvert \leq 100$	U	8.61×10^{-13} See Note 5	4.61×10^{-14}		
	TAN	tan (arg), arg in radians	1	REAL * 4	$\lvert$arg$\rvert < 2^{18}\pi$	REAL * 4	$\lvert$Argument$\rvert \geq 2^{18}\pi$ Argument too close to a Singularity (i.e., too close to an odd multiple of $\pi/2$)	288	648		CHCAY	$\lvert X\rvert \leq \frac{\pi}{4}$	U	1.71×10^{-6}	2.64×10^{-7}		
												$\frac{\pi}{4} < \lvert X\rvert \leq \frac{\pi}{2}$	U	1.05×10^{-6} See Note 5	3.59×10^{-7}		
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U	6.49×10^{-6} See Note 5	3.38×10^{-7}		
												$10 < \lvert X\rvert \leq 100$	U	1.57×10^{-6} See Note 5	3.07×10^{-7}		
	COTAN	cotan (arg), arg in radians	1	REAL * 4	$\lvert$arg$\rvert < 2^{18}\pi$	REAL * 4	$\lvert$Argument$\rvert \geq 2^{18}\pi$ Argument too close to a Singularity (i.e., too close to a multiple of π)		648		CHCAY	$\lvert X\rvert \leq \frac{\pi}{4}$	U	1.07×10^{-6}	3.58×10^{-7}		
												$\frac{\pi}{4} < \lvert X\rvert \leq \frac{\pi}{2}$	U	1.40×10^{-6} See Note 5	2.56×10^{-7}		
												$\frac{\pi}{2} < \lvert X\rvert \leq 10$	U	1.30×10^{-6} See Note 5	3.11×10^{-7}		
												$10 < \lvert X\rvert \leq 100$	U	1.49×10^{-6} See Note 5	3.15×10^{-7}		
HYPERBOLIC SINE & COSINE	DSINH	sinh (arg)	1	REAL * 8	$\lvert$arg$\rvert < 175.366$	REAL * 8	$\lvert$Argument$\rvert \geq 174.673$	250	592	DEXP	CHCBB	$\lvert X\rvert \leq 0.88137$	U	2.06×10^{-16}	3.74×10^{-17}		
												$0.88137 < \lvert X\rvert \leq 5$	U	3.80×10^{-16}	9.21×10^{-17}		
	DCOSH	cosh (arg)	1	REAL * 8	$\lvert$arg$\rvert < 175.366$	REAL * 8	$\lvert$Argument$\rvert \geq 174.673$	250	592	DEXP	CHCBB	$-5 \leq X \leq +5$	U	3.63×10^{-16}	9.05×10^{-17}		
	SINH	sinh (arg)	1	REAL * 4	$\lvert$arg$\rvert < 175.366$	REAL * 4	$\lvert$Argument$\rvert \geq 174.673$	1F8	504	EXP	CHCBA	$-5 \leq X \leq +5$	U	1.26×10^{-6}	2.17×10^{-7}		
	COSH	cosh (arg)	1	REAL * 4	$\lvert$arg$\rvert < 175.366$	REAL * 4	$\lvert$Argument$\rvert \geq 174.673$	1F8	504	EXP	CHCBA	$-5 \leq X \leq +5$	U	1.27×10^{-6}	2.63×10^{-7}		

Table 1. Summary of Directly Referenced Mathematical Subprograms (cont.)

1	2	3	4			5	6	7		8	9	10					
Function	Entry Name	Definition	Argument(s)			Function Value Returned	Error Condition	Storage Estimates		Other Subprograms Required	Routine Name	Accuracy Figures					
												Argument Range	Sample E/U	relative		absolute	
			No.	Type	Range			Hex	Dec					M (ε)	σ (ε)	M (E)	σ (E)
HYPERBOLIC TANGENT	DTANH	tanh (arg)	1	REAL * 8	Any	REAL * 8		130	304	DEXP	CHCAL	$\|X\| \leqq 0.54931$	U	1.91×10^{-16}	3.86×10^{-17}		
												$0.54931 < \|X\| \leqq 5$	U	1.54×10^{-16}	1.87×10^{-17}		
	TANH	tanh (arg)	1	REAL * 4	Any	REAL * 4		164	356	EXP	CHCAK	$\|X\| \leqq 0.7$	U	8.48×10^{-7}	1.48×10^{-7}		
												$0.7 < \|X\| \leqq 5$	U	2.44×10^{-7}	4.23×10^{-8}		
ABSOLUTE VALUE	CDABS	|arg|	1	COMPLEX * 16	Any See Note 9	REAL * 8		C8	200	DSQRT	CHCAV	The full range	Note 1	2.03×10^{-16}	4.83×10^{-17}		
	CABS	|arg|	1	COMPLEX * 8	Any See Note 9	REAL * 4		C0	192	SQRT	CHCAU	The full range	Note 1	9.15×10^{-7}	2.00×10^{-7}		
ERROR FUNCTION	ERF	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$	1	REAL * 4	Any	REAL * 4		208	520	EXP	CHCBU	$\|X\| \leqq 1$	U	8.16×10^{-7}	1.10×10^{-7}		
												$1 < \|X\| \leqq 2.04$	U	1.13×10^{-7}	3.70×10^{-8}		
												$2.04 < \|X\| \leqq 3.9192$	U	5.95×10^{-8}	3.41×10^{-8}		
	DERF	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$	1	Real * 8	Any	Real * 8		328	808	DEXP	CHCBW	$\|X\| \leqq 1$	U	1.89×10^{-16}	2.60×10^{-17}		
												$1 < \|X\| \leqq 2.04$	U	2.87×10^{-17}	9.84×10^{-18}		
												$2.04 < \|X\| < 6.092$	U	1.39×10^{-17}	8.02×10^{-18}		
COMPLEMENTED ERROR FUNCTION	ERFC	1 - erf (x) or $\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-u^2}du$	1	REAL * 4	Any	REAL * 4		208	520	EXP	CHCBU	$-3.8 < X < 0$	U	9.10×10^{-7}	2.96×10^{-7}		
												$0 \leqq X \leqq 1$	U	7.42×10^{-7}	1.27×10^{-7}		
												$1 < X \leqq 2.04$	U	1.54×10^{-6}	3.78×10^{-7}		
												$2.04 < X < 4$	U	2.28×10^{-6}	3.70×10^{-7}		
												$4 \leqq X \leqq 13.3$	U	1.55×10^{-5}	8.57×10^{-6}		
	DERFC	1 - erf (x) or $\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-u^2}du$	1	Real * 8	Any	Real * 8		328	808	DEXP	CHCBW	$-6 < X < 0$	U	2.08×10^{-16}	6.52×10^{-17}		
												$0 \leqq X \leqq 1$	U	1.40×10^{-16}	2.59×10^{-17}		
												$1 < X \leqq 2.04$	U	4.11×10^{-16}	8.86×10^{-17}		
												$2.04 < X < 4$	U	3.26×10^{-16}	8.65×10^{-17}		
												$4 \leqq X < 13.3$	U	3.51×10^{-15}	1.96×10^{-15}		
GAMMA (Γ)	GAMMA	$\int_0^\infty u^{x-1}e^{-u}du$	1	REAL * 4	$X > 2^{-252}$ $X < 57.5744$	REAL * 4	Real Argument > 57.5744 Real Argument $< 2^{-252}$	350	848	EXP, ALOG	CHCBT	$0 < X < 1$	U	9.86×10^{-7}	3.66×10^{-7}		
												$1 \leqq X \leqq 2$	U	1.13×10^{-7}	3.22×10^{-8}		
												$2 < X \leqq 4$	U	9.47×10^{-7}	3.79×10^{-7}		
												$4 < X < 8$	U	2.26×10^{-6}	8.32×10^{-7}		
												$8 \leqq X < 16$	U	2.20×10^{-5}	7.61×10^{-6}		
												$16 \leqq X < 57$	U	4.62×10^{-5}	1.51×10^{-5}		
	DGAMMA	$\int_0^\infty u^{x-1}e^{-u}du$	1	REAL * 8	$X > 2^{-252}$ $X < 57.5744$	REAL * 8	Real Argument > 57.5744 Real Argument $< 2^{-252}$	420	1056	DEXP, DLOG	CHCBV	$0 < X < 1$	U	2.14×10^{-16}	7.84×10^{-17}		
												$1 \leqq X \leqq 2$	U	2.52×10^{-17}	6.07×10^{-18}		
												$2 < X < 4$	U	2.21×10^{-16}	8.49×10^{-17}		
												$4 \leq X < 8$	U	5.05×10^{-16}	1.90×10^{-16}		
												$8 \leqq X < 16$	U	6.02×10^{-15}	1.78×10^{-15}		
												$16 \leqq X < 57$	U	1.16×10^{-14}	4.11×10^{-15}		
LOG - GAMMA	ALGAMA	$\log_e\int_0^\infty u^{x-1}e^{-u}du$	1	REAL * 4	$X > 0$ $X < 4.2913 \times 10^{73}$	REAL * 4	Real Argument $> 4.2937 \times 10^{73}$ Real Argument < 0	350	848	EXP, ALOG	CHCBT	$0 < X < 0.5$	U	1.16×10^{-6}	3.54×10^{-7}		
												$0.5 \leq X < 3$	U			9.43×10^{-7}	3.42×10^{-7}
												$3 \leqq X < 8$	U	1.25×10^{-6}	3.04×10^{-7}		
												$8 \leqq X < 16$	U	1.18×10^{-6}	3.80×10^{-7}		
												$16 \leqq X < 500$	U	9.85×10^{-7}	1.90×10^{-7}		
	DLGAMA	$\log_e\int_0^\infty u^{x-1}e^{-u}du$	1	REAL * 8	$X > 0$ $X < 4.2913 \times 10^{73}$	REAL * 8	Real Argument $> 4.2937 \times 10^{73}$ Real Argument < 0	420	1056	DEXP, DLOG	CHCBV	$0 < X \leqq 0.5$	U	2.77×10^{-16}	9.75×10^{-17}		
												$0.5 < X < 3$	U			2.24×10^{-16}	7.77×10^{-17}
												$3 \leqq X < 8$	U	2.89×10^{-16}	8.80×10^{-17}		
												$8 \leqq X < 16$	U	2.86×10^{-16}	8.92×10^{-17}		
												$16 \leqq X < 500$	U	1.99×10^{-16}	3.93×10^{-17}		

Notes

1. The distribution of sample arguments upon which these statistics are based is exponential radially and is uniform around the origin.

2. The maximum relative error cited for the CCOS function is based upon a set of 2000 random arguments within the range. In the immediate proximity of the points $(n + 1/2)\pi + 0i$ (where $n = 0, \pm 1, \pm 2, \ldots,$) the relative error can be quite high, although the absolute error is small.

3. The maximum relative error cited for the CDCOS function is based upon a set of 1500 random arguments within the range. In the immediate proximity of the points $(n + 1/2)\pi + 0i$ (where $n = 0, \pm 1, \pm 2, \ldots,$) the relative error can be quite high although the absolute error is small.

4. The maximum relative error cited for the CDSIN function is based upon a set of 1500 random arguments within the range. In the immediate proximity of the points $n\pi + 0i$ (where $n = \pm 1, \pm 2, \ldots,$) the relative error can be quite high although the absolute error is small.

5. The figures cited as the maximum relative errors are those encountered in a sample of 2500 random arguments within the respective ranges. See the appropriate section in Appendix F for a description of the behavior of errors when the argument is near a singularity or a zero of the function.

6. The maximum relative error cited for the CSIN function is based upon a set of 2000 random arguments within the range. In the immediate proximity of the points $n\pi + 0i$ (where $n = \pm 1, \pm 2, \ldots,$) the relative error can be quite high although the absolute error is small.

7. The sample arguments were tangents of numbers uniformly distributed between $-\pi/2$ and $+\pi/2$.

8. The answer given is the principal value, i.e., the one whose imaginary part lies between $-\pi$ and $+\pi$.

9. Floating-point overflow can occur.

Table 2. Summary of Indirectly Referenced Mathematical Subprograms

	2	3	4		5	6	7		8	10
Function	Entry Name	Definition	Argument(s)		Function Value Returned	Error Condition	Storage Estimates		Other Subprograms Required	Routine Name
			No.	Type			Hex	Dec		
RAISE AN INTEGER BASE TO AN INTEGER POWER	CHCBGA	y = i * * i	2	i = INTEGER * 4	INTEGER * 4	Base is zero	1B4	436	——	CHCBG
	CHCBGB	y = j * * j	2	j = INTEGER * 2	INTEGER * 2	Base is zero and exponent is zero or negative	1B4	436	——	CHCBG
	CHCBGC	y = j * * i	2	j = INTEGER * 2 i = INTEGER * 4	INTEGER * 4	Base is zero and exponent is zero or negative	1B4	436	——	CHCBG
	CHCBGD	y = i * * j	2	i = INTEGER * 4 j = INTEGER * 2	INTEGER * 4	Base is zero and exponent is zero or negative	1B4	436	——	CHCBG
RAISE A REAL BASE TO AN INTEGER POWER	CHCBHA	y = a * * i	2	a = REAL * 4 i = INTEGER * 4	REAL * 4	Base is zero and exponent is zero or negative	144	324	——	CHCBH
	CHCBHB	y = a * * j	2	a = REAL * 4 j = INTEGER * 2	REAL * 4	Base is zero and exponent is zero or negative	144	324	——	CHCBH
RAISE A DOUBLE PRECISION BASE TO AN INTEGER POWER	CHCBIA	y = a * * i	2	a = REAL * 8 = INTEGER * 4	REAL * 8	Base is zero and exponent is zero or negative	14C	332	——	CHCBI
	CHCBIB	y = a * * j	2	a = REAL * 8 j = INTEGER * 2	REAL * 8	Base is zero and exponent is zero or negative	14C	332	——	CHCBI
RAISE A REAL BASE TO A REAL POWER	CHCBJA	y = a * * b	2	a = REAL * 4 b = REAL * 4	REAL * 4	Base is zero and exponent is zero or negative	1C0	448	EXP ALOG	CHCBJ
RAISE AN INTEGER BASE TO A REAL POWER	CHCBJB	y = i * * b	2	b = REAL * 4 i = INTEGER * 2	REAL * 4	Base is zero and exponent is zero or negative	1C0	448	EXP, ALOG	CHCBJ
	CHCBJC	y = i * * b	2	b = REAL * 4 i = INTEGER * 4	REAL * 4	Base is zero and exponent is zero or negative	1C0	448	EXP, ALOG	CHCBJ
RAISE A REAL OR INTEGER BASE TO A REAL POWER; BASE AND/OR EXPONENT DOUBLE PRECISION	CHCBKA	y = a * * b	2	a = REAL * 8 b = REAL * 8	REAL * 8	Base is zero and exponent is zero or negative	230	560	DEXP, DLOG	CHCBK
	CHCBKB	y = j * * b	2	b = REAL * 8 i = INTEGER * 2	REAL * 8	Base is zero and exponent is zero or negative	230	560	DEXD, DLOG	CHCBK
	CHCBKC	y = i * * b	2	b = REAL * 8 i = INTEGER * 4	REAL * 8	Base is zero and exponent is zero or negative	230	560	DEXP, DLOG	CHCBK
	CHCBKD	y = a * * b	2	a = REAL * 4 b = REAL * 8	REAL * 8 See Note.	Base is zero and exponent is zero or negative	230	560	DEXP, DLOG	CHCBK
	CHCBKE	y = a * * b	2	a = REAL * 8 b = REAL * 4	REAL * 8	Base is zero and exponent is zero or negative	230	560	DEXP, DLOG	CHCBK
RAISE A COMPLEX BASE TO AN INTEGER POWER	CHCBMA	y = a * * i	2	a = COMPLEX * 16 i = INTEGER * 4	COMPLEX * 16	Base is zero and exponent is zero or negative	274	628	——	CHCBM
	CHCBMB	y = a * * j	2	a = COMPLEX * 16 i = INTEGER * 2	COMPLEX * 16	Base is zero and exponent is zero or negative	274	628	——	CHCBM
	CHCBCA	y = a * * i	2	a = COMPLEX * 8 i = INTEGER * 4	COMPLEX * 8	Base is zero and exponent is zero or negative	24C	588	——	CHCBC
	CHCBCB	y = a * * j	2	a = COMPLEX * 8 i = INTEGER * 2	COMPLEX * 8	Base is zero and exponent is zero or negative	24C	588	——	CHCBC
PRODUCE ERROR MESSAGE AND TERMINATE EXECUTION	CHCBZA						E8	232	As required by use of the EXIT macro instruction	CHCBZ

NOTE: The REAL*8 function value returned by CHCBKD is not more accurate than the REAL*4 base given as an argument.

- Argument range—This column gives the argument range used to obtain the accuracy figures. For each function, accuracy figures are given more representative segments within the valid argument range. These figures are the most meaningful to the function and range under consideration.

  The maximum relative error and standard deviation of the relative error are generally useful and revealing statistics. However, they are useless for the range of a function where its value becomes 0, because the slightest error in the argument can cause an unpredictable fluctuation in the magnitude of the answer. When a small argument error would have this effect, the maximum absolute error and standard deviation of the absolute error are given for the range. For example, absolute error is given for sin (x) for values of x near π.

- Sample—This column indicates the type of sample used for the accuracy figures; the type depends upon the function and range under consideration. The statistics may be based either upon an exponentially distributed (E) argument sample or a uniformly distributed (U) argument sample.

- Statistical results:

$$M(\epsilon)=\text{Max}\left|\frac{f(x)-g(x)}{f(x)}\right|$$ Maximum relative error produced during testing

$$\sigma(\epsilon)=\sqrt{\frac{1}{N}\sum_i\left|\frac{f(x_i)-g(x_i)}{f(x_i)}\right|^2}$$ Standard deviation (root-mean-square) of the relative error

$$M(E)=\text{Max}\left|f(x)-g(x)\right|$$ Maximum absolute error produced during testing

$$\sigma(E)=\sqrt{\frac{1}{N}\sum_i\left|f(x_i)-g(x_i)\right|^2}$$ Standard deviation (root-mean-square) of the absolute error.

In the formulas for the standard deviation, N represents the total number of arguments in the sample; i is a subscript that varies from 1 to N. Appendix F explains other symbols used above.

Test ranges, where they do not cover the entire legal range of a subroutine, were selected so that users may infer from the accuracy figures presented the trend of errors as an argument moves away from the principal range. The accuracy of the answer deteriorates substantially as the argument approaches the limit of the permitted range in several of the subroutines. This is particularly true for trigonometric functions. However, an error generated by any of these subroutines is, at worst, comparable in order of magnitude to the effect of the inherent rounding error of the argument.

Error Message: CHCBZ100 is issued each time an error occurs. This message gives the error condition, the entry name, and the address of the call to the math routine in the user's program.

Table 3. Exponentiation With Integer Base and Exponent

Base (I)	Exponent (J)		
	$J>0$	$J=0$	$J<0$
$I>1$	Compute the function value	Function value = 1	Function value = 0
$I=I$	Compute the function value	Function value = 1	Function value = 1
$I=0$	Function value = 0	Error message	Error message
$I=-1$	Compute the function value	Function value = 1	If J is an odd number, function value = −1 If J is an even number, function value = 1
$I<-1$	Compute the function value	Function value = 1	Function value = 0

Table 4. Exponentiation With Real or Double-Precision Base and Integer Exponent

Base (A)	Exponent (J)		
	$J>0$	$J=0$	$J<0$
$A>0$	Compute function value	Function value = 1	Compute function value
$A=0$	Function value = 0	Error message	Error message
$A<0$	Compute function value	Function value = 1	Compute function value

Table 5. Exponentiation With Real or Double-Precision Base and Exponent

Base (A)	Exponent (B)		
	$B>0$	$B=0$	$B<0$
$A>0$	Compute function value	Function value = 1	Compute function value
$A=0$	Function value = 0	Error message	Error message
$A<0$	Error message	Function value = 1	Error message

Section 2: Service Subprograms

The service subprograms supplied with FORTRAN IV are:

- Pseudo sense light subprograms (SLITE, SLITET)
- STOP, EXIT, and PAUSE subprograms
- Dump subprograms (DUMP, PDUMP)
- Overflow and underflow subprograms (OVERFL, DVCHK)
- Specification exception subprograms

These subprograms are briefly described below and in Table 6. In most cases the actual entry point name of the subprogram is identical to the command name. However, when the user keys in the EXIT, STOP or PAUSE command, the compiler translates the command name into a separate entry point name to call the subprogram. Both names are shown in Table 6. Further information concerning their usage is given in *IBM FORTRAN IV*.

Table 6. Summary of Service Subprograms Characteristics

1		2	3	4		5
Function		Entry Name	Error Condition	Storage Estimates HEX	Storage Estimates DEC	Module Name
Pseudo sense light subprograms	Turn all sense lights off or one sense light on	SLITE	Argument other than 0, 1, 2, 3, 4	324	804	CHCBE
	Test a sense light or record its status	SLITET	Argument other than 1, 2, 3, 4			CHCBE
Overflow and underflow subprogram	Test and record status of exponent overflow and underflow indicators	OVERFL				CHCBE
Divide check subprogram	Test and record status of divide check indicator	DVCHK				CHCBE
Exception processing subprograms	Process arithmetic exceptions	CHCBE3 (exponent overflow) CHCBE4 (exponent underflow) CHCBE5 (divide check)				CHCBE
	Process specification exceptions	CHCBE2 (specification)				CHCBE
Exit subprogram	Terminate execution	EXIT (CHCIW1) STOP (CHCIW2) PAUSE (CHCIW3)		1AC	428	CHCIW
Dump subprogram	Dump specified storage area with or without termination	DUMP, PDUMP		48	168	CHCIV

Pseudo Sense Light Subprograms

The program-simulated machine indicator subprograms test the status of pseudo indicators, and return a value indicating the result of this test to the calling program. When the indicator is 0, it is off; when the indicator is other than 0, it is on. In the following descriptions of the subprograms, *i* represents an integer expression, and *j* represents an integer variable.

The CALL SLITE statement is used to alter the status of pseudo sense lights; the CALL SLITET statement is used to test, and/or record their status. The particular user reference name used in the CALL statement depends upon the operation to be performed.

SLITE is used if the four sense lights are to be turned off or one sense light is to be turned on. The source-language statement is

CALL SLITE(*i*)

where *i* has a value of 0, 1, 2, 3, or 4.

If the value of *i* is 0, the four sense lights are turned off; if the value of *i* is 1, 2, 3, or 4, the corresponding sense light is turned on. If the value of *i* is not 0, 1, 2, 3, or 4, error message 216 is issued, and execution is terminated.

SLITET is used if a sense light is to be tested and its status recorded. The source-language statement is

CALL SLITET (*i*, *j*)

where *i* has a value of 1, 2, 3, or 4, and indicates which sense light to test; *j* is set to 1 if the sense light is on or to 2 if the sense light is off.

If the value of *i* is not 1, 2, 3, or 4, error message 216 is issued and execution is terminated.

DUMP and PDUMP Subprograms

The CALL DUMP and CALL PDUMP statements allow the user to request that data contained within his program be dumped in one of nine formats. The dumps produced will be added to the user's SYSOUT.

It is also possible to obtain dumps using the facilities of the Program Control System (PCS). For information concerning PCS, see *FORTRAN Programmer's Guide and Command System User's Guide.*

The CALL DUMP statement is used if execution is to be terminated after the dump is taken. The source-language statement is

CALL DUMP $(a_1,b_1,f_1, \ldots, a_n,b_n,f_n)$

where *a* and *b* are variables that indicate the limits of storage to be dumped (either *a* or *b* may represent the upper or lower limits of storage). The dump format is indicated by *f* and may be one of the integers given in Table 7. A sample printout for each format is given in Appendix D.

If execution of the object module is to be resumed after the dump is taken, the CALL PDUMP statement is used. The source-language statement is

CALL PDUMP $(a_1,b_1,f_1, \ldots, a_n,b_n,f_n)$

where *a*, *b*, and *f* have the same meaning as explained previously.

Table 7. DUMP/PDUMP Format Specifications

Integer	Specified Format
0	hexadecimal
1	logical *1
2	logical *4
3	integer *2
4	integer *4
5	real *4
6	real *8
7	complex *8
8	complex *16
9	literal (character)

Programming Considerations

1. If the format control integer *f* is omitted, it is assumed to be equal to 0, and the dump will be hexadecimal.
2. The arguments *a* and *b* should be defined in the program in which the DUMP or PDUMP statement occurs; otherwise, the compiler will assign arbitrary addresses to them.
3. If the program in which DUMP or PDUMP occurs is a subprobram, and if *a* and *b* are argument names, a range of storage from the calling program will be dumped. However, if one is an argument name and the other is not, unpredictable and probably large areas of storage will be dumped; this should be avoided.
4. If one of the limits (*a* or *b*) of storage definition variable names is in COMMON and the other is not or if it is a different (named) COMMON, unpredictable and probably large areas of storage will be dumped; this situation should be avoided.
5. The literal format in Table 7 causes the area that is to be dumped to be treated as a string of alphameric characters.

STOP, EXIT, and PAUSE Subprograms

The STOP, EXIT, and PAUSE subprograms are called by the compiled object programs as a result of the source statements

CALL EXIT
STOP
PAUSE

Statements that cause the user's program to be terminated are

CALL EXIT
STOP

If STOP is issued in a conversational task, a message is written on the user's terminal, and control is returned to the terminal for entry of the next command by the user. If STOP is issued by a nonconversational task, the message is written on the SYSOUT data set, and the next command is taken from the SYSIN data set. The STOP statement has the same effect when used in either a subprogram or main program. The CALL EXIT statement is equivalent to a STOP statement.

A PAUSE statement executed in a program running in a nonconversational task will result in any associated messages being written to SYSOUT; the program then continues execution. In a conversational task the system prints, at the terminal, the word PAUSE followed by 00000 or a 1-to-5-digit integer constant, or a message, depending on how the operand field of the PAUSE statement was written. The system then transfers control to the terminal and awaits the user's input before resuming program execution.

Overflow and Underflow Subprograms

The CALL OVERFL statement allows a test for prior occurrence of an exponent overflow or underflow exception. The value returned by this CALL indicates which of these two conditions occurred last. After testing, the overflow or underflow indication is no longer available. The source language statement is

CALL OVERFL (*j*)

where *j* is set to 1 if a floating point overflow condition (i.e., $\geq 16^{63}$) exists; is set to 2 if no overflow or underflow condition exists; or to 3 if a floating point underflow (i.e., $<16^{-65}$) condition exists. A more detailed description of each exception is given in Appendix E.

Divide Check Subprogram

The CALL DVCHK statement allows a test for prior occurrence of a floating point divide-check exception, and returns a value that indicates the existing condition. (Fixed-point divide checks are ignored by FORTRAN-compiled programs.) After testing, the indication of a prior divide check is no longer available. The source-language statement is

CALL DVCHK (*j*)

where *j* is set to 1 if the divide-check indicator was on, or to 2 if the indicator was off. A more detailed description of the divide-check exception is given in Appendix E.

Section 3: I/O Subprograms

This section discusses the functions, entry requirements, error checks, and data references of the TSS/360 FORTRAN I/O library in executing the FORTRAN I/O statements: READ, WRITE, REWIND, BACKSPACE, END FILE, PRINT, and PUNCH.

This section is written for both FORTRAN and assembler-language programmers. The FORTRAN programmer may be interested in the assumptions that the I/O routines make, the error conditions that they check for, and the actions they take in case of error. The assembler-language programmer may be interested in the advantages of FORTRAN I/O facilities, particularly the data conversion, list-processing, and DCB-maintenance routines. The assembler-language programmer should read this section after reading *IBM Time Sharing System: FORTRAN Programmer's Guide,* Form C28-2025, "Appendix E. Specification of Data Set Characteristics," and *IBM Time Sharing System: IBM FORTRAN IV,* Form C28-2007, the sections titled "Input/Output Statements," and "Elements of the Language." Of the section on elements of the language, he need only read the subsections titled "Constants," "Variables," and "Arrays."

Overview of the FORTRAN I/O Library

There are twenty-one FORTRAN I/O routines. Only three routines, Control Initialization (CHCIA), List Item Processor (CHCIE), or List Termination (CHCIU), can take control from, or return control to, a FORTRAN object program. Thus, the FORTRAN I/O library can be regarded as three subprograms and a number of subroutines of these subprograms.

Since the assembler-language programmer has techniques (described in Appendix B) for linking to any of the FORTRAN I/O routines, he can look upon any one of these routines as a subprogram.

Another way of looking at the FORTRAN I/O Library is as two main categories of routines: I/O language control routines and data conversion routines. The routines of each group interact with one another by means of a common communication and work region in a common PSECT.

I/O Language Control Routines

There are two types of I/O language control routines: I/O operation control and I/O list control. These routines analyze the user's I/O requests to determine information such as: the type of I/O operation to be performed; the number and type of list items present, if any; the type of format control, if any; and the I/O statement relationships with a user-specified DDEF command.

I/O Operation Control Routines

These routines control the I/O request by creating, if necessary, a data control block (DCB), and analyzing FORMAT and NAMELIST control specified by the user. After this information is processed, the I/O operation control routines interface with the TSS data management routines that actually fulfill the I/O request. The interface with data management is accomplished by the routines CHCIB and CHCIC, via the data management macro instruction facilities.

I/O List Control Routines

These routines examine the list items, if any, in each I/O request to determine the type of conversion to be performed. After the type of data conversion is determined, control is given to the I/O operation control routines which in turn call the appropriate data conversion routines for final processing.

Data Conversion Routines

The data conversion routines are subdivided into routines used for input processing and routines used for the preparation of output. These routines can process all the permissible types of FORTRAN-formatted data specified in either a FORMAT or NAMELIST statement.

When converting a user's data, the data conversion routines interact with each other according to the requirements of the user-specified FORMAT or NAMELIST control. For example, for input data that is defined by a G-format conversion code, the General Input Conversion routine (CHCIS) is called. This routine analyzes the data type to determine whether it is integer, real, logical, or alphameric and calls the appropriate data conversion routine.

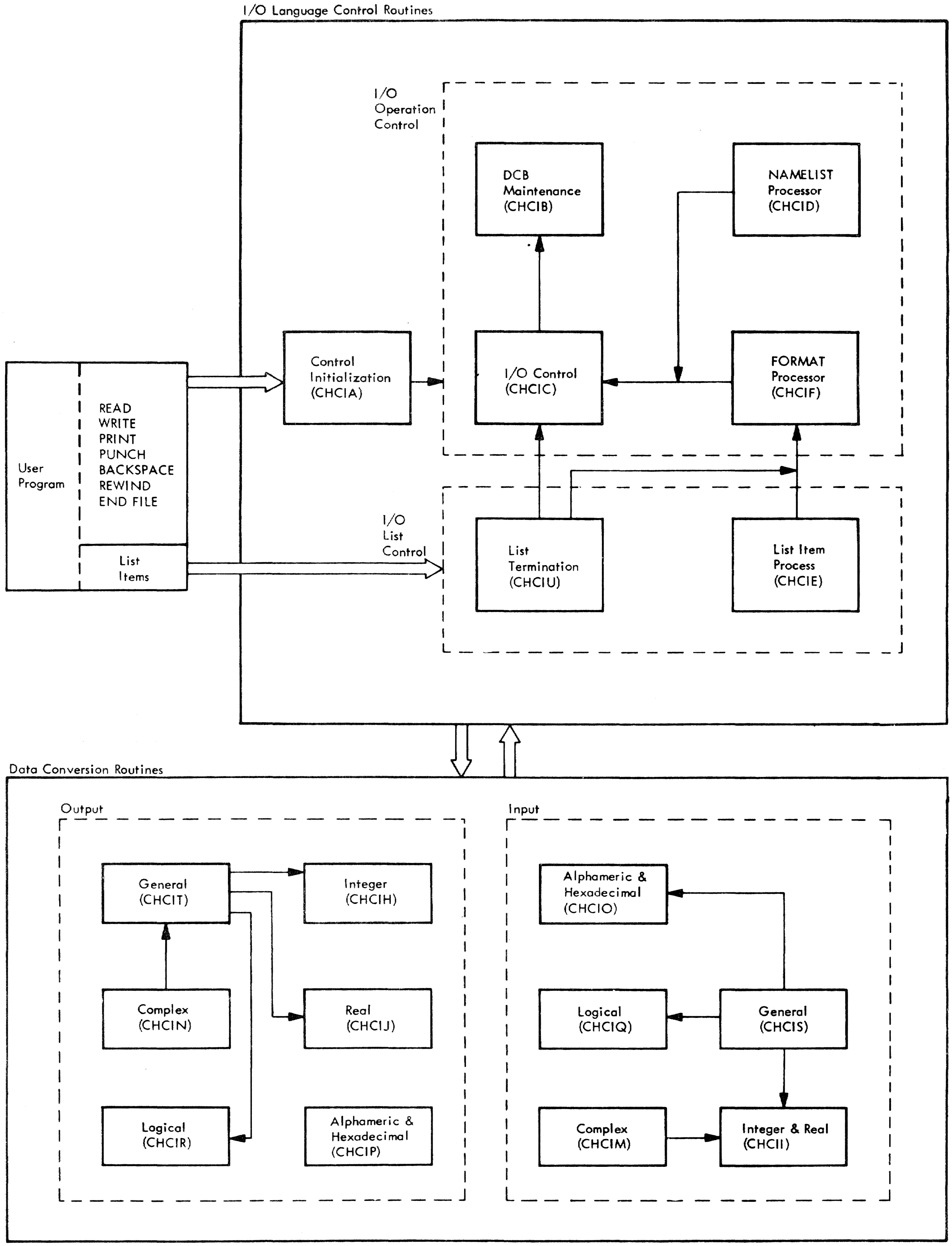

Figure 1. Functional Flow of FORTRAN I/O Routines

Routine Interrelationships

Table 8 presents the calling relationships between the user program, the FORTRAN I/O routines, Data Management, and the Supervisor.

Table 8. Calling Relationships of I/O Routines

Calling Routines \ Routines Called	CHCIA	CHCIB	CHCIC	CHCID	CHCIE	CHCIF	CHCIH	CHCII	CHCIJ	CHCIM	CHCIN	CHCIO	CHCIP	CHCIQ	CHCIR	CHCIS	CHCIT	CHCIU	CHCIV	CHCIW	CHCBD	Data Management	Supervisor
USER PROGRAM	X				X													X	X	X	X		
CHCIA		X	X	X		X														X			
CHCIB																				X		X	X
CHCIC		X																		X		X	X
CHCID			X							X	X					X	X			X			
CHCIE			X			X														X			
CHCIF			X				X	X	X	X	X	X	X	X	X	X	X			X			
CHCIH																							
CHCII																							
CHCIJ																							
CHCIM								X															
CHCIN																	X						
CHCIO																							
CHCIP																							
CHCIQ																							
CHCIR																							
CHCIS								X				X		X									
CHCIT							X		X						X								
CHCIU			X			X																	
CHCIV		X	X				X				X		X		X		X			X			
CHCIW		X	X																				X
CHCBD																							

The following figures describe the relationships between routines when fulfilling a particular I/O operation. Since the relationships vary, depending on the kind of I/O operation being performed, a separate diagram is presented for each of the basic I/O operations. Exceptions to the logical flows presented in this subsection are described in detail under the individual routine descriptions in the following subsection.

The type of I/O operation and its related figure reference are:

TYPE OF OPERATION (FUNCTION)	FIGURE
Formatted READ with List	2
Formatted READ without List	3
READ with NAMELIST	4
Unformatted READ with List	5
Unformatted READ without List	6
Formatted WRITE with List	7
Formatted WRITE without List	8
WRITE with NAMELIST	9
Unformatted WRITE with List	10
Unformatted WRITE without List	11
REWIND, BACKSPACE, END FILE	12

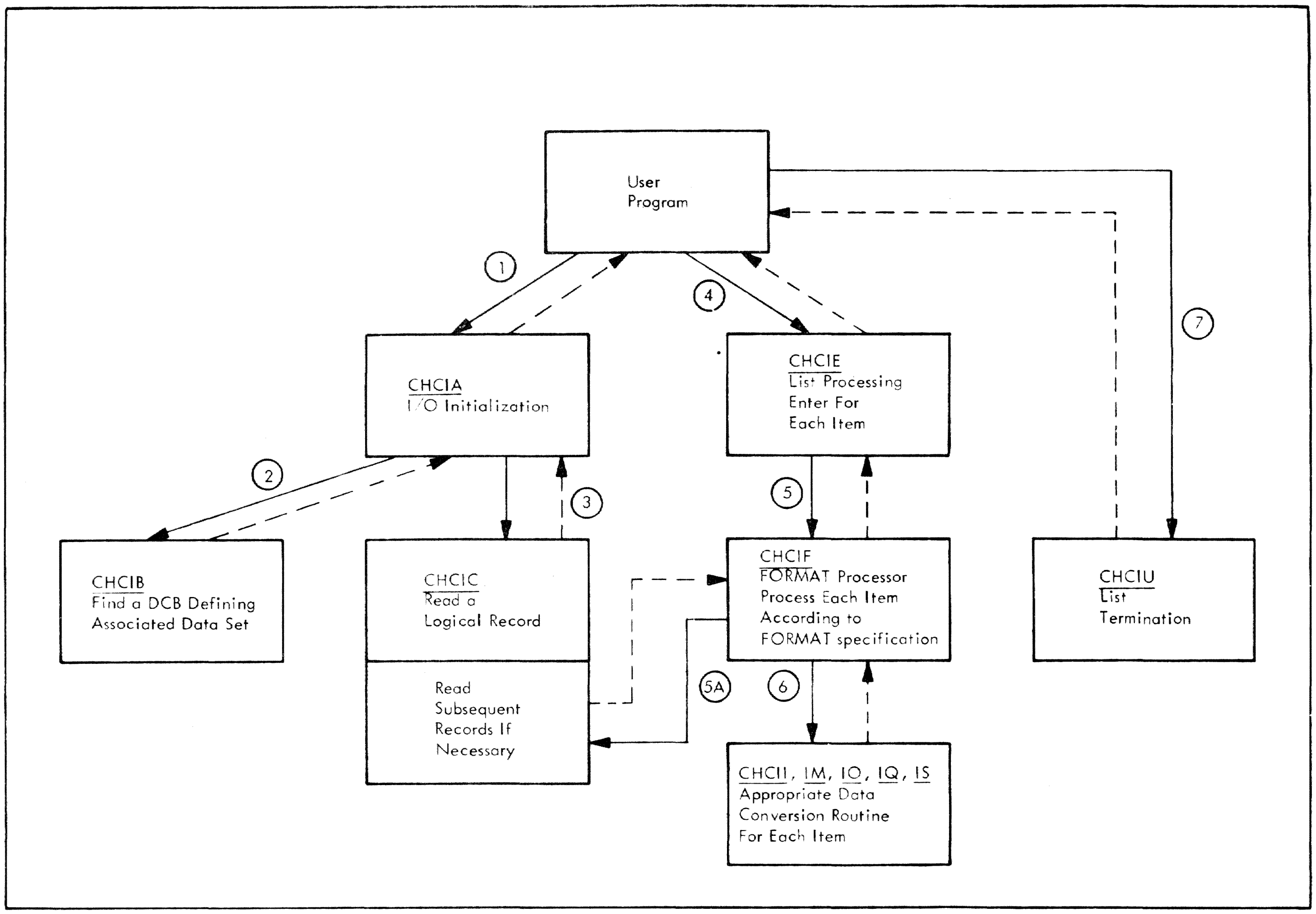

Figure 2. Formatted READ with List

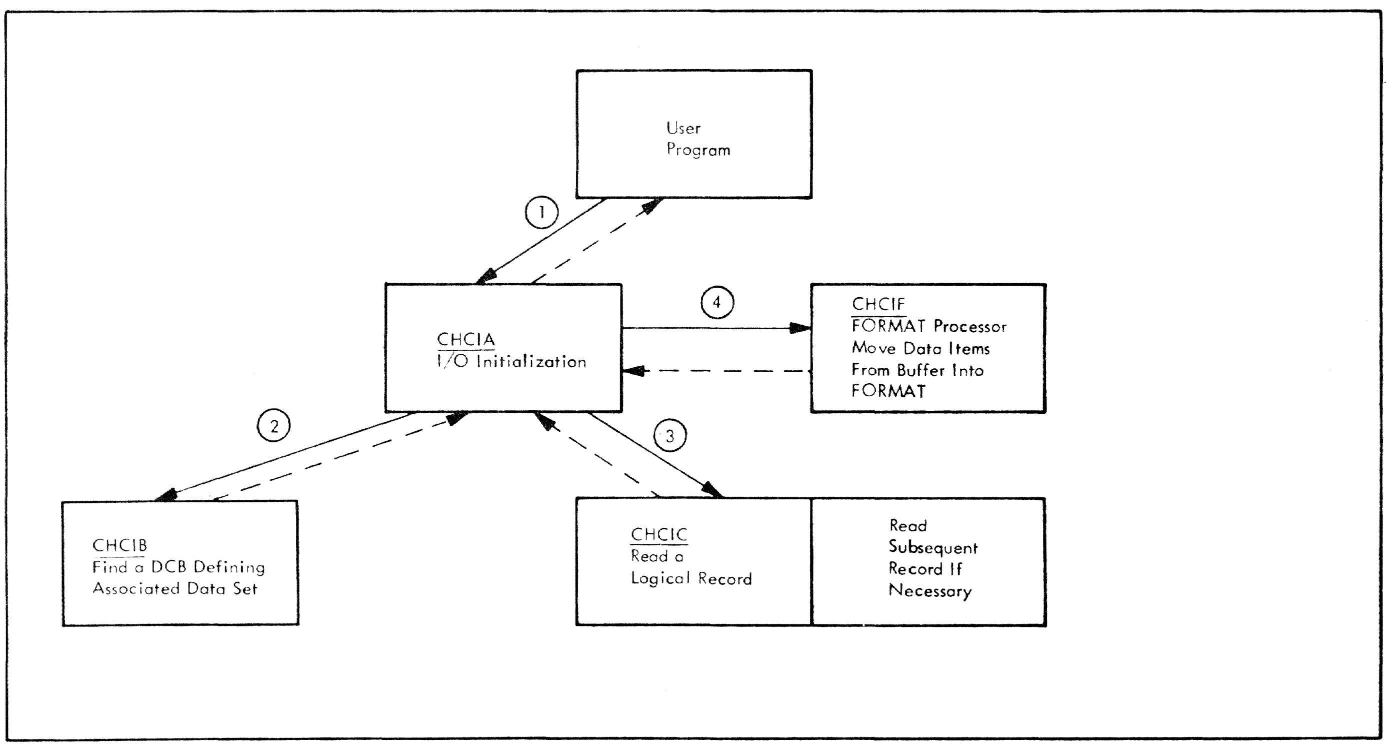

Figure 3. Formatted READ without List

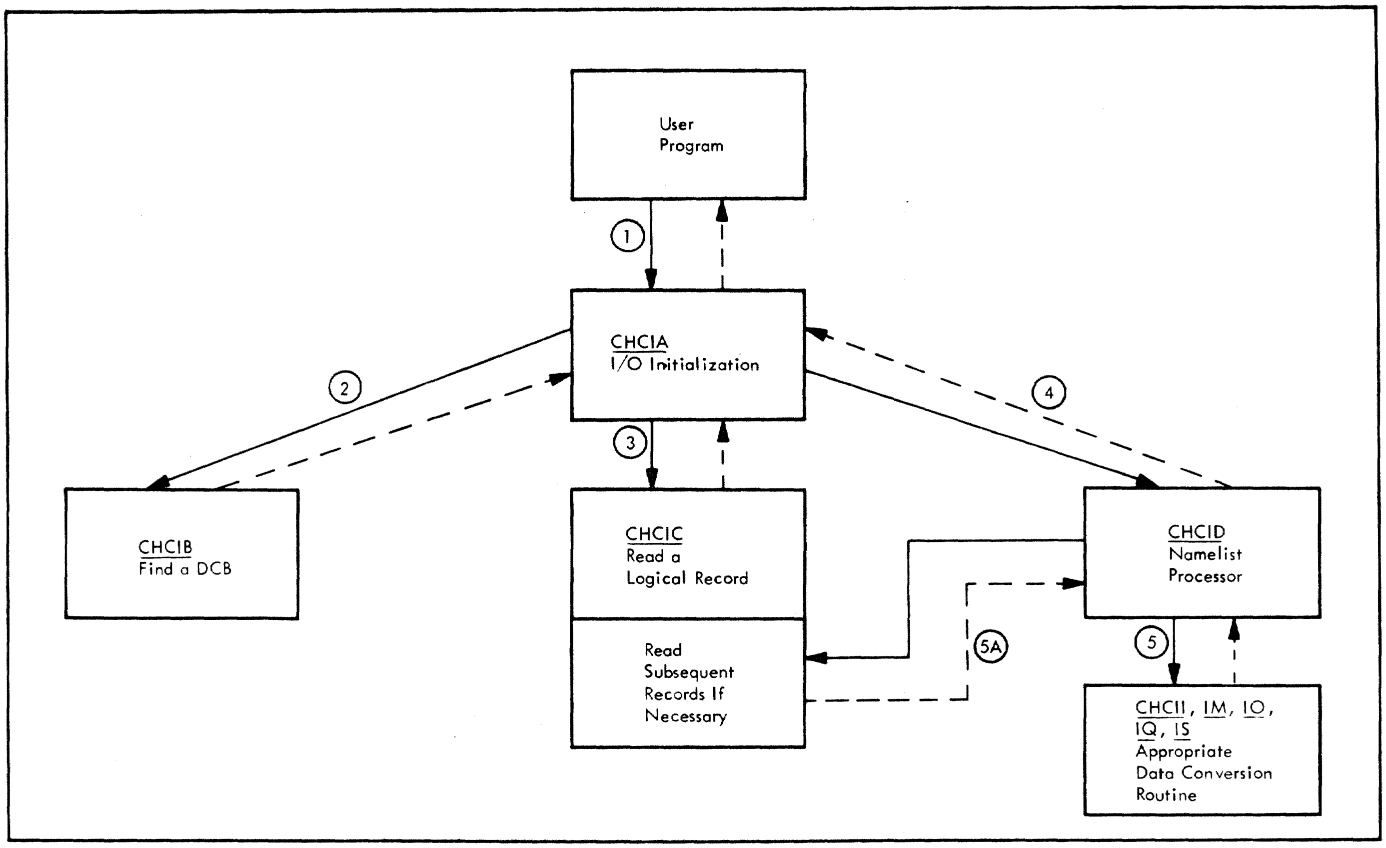

Figure 4. READ with NAMELIST

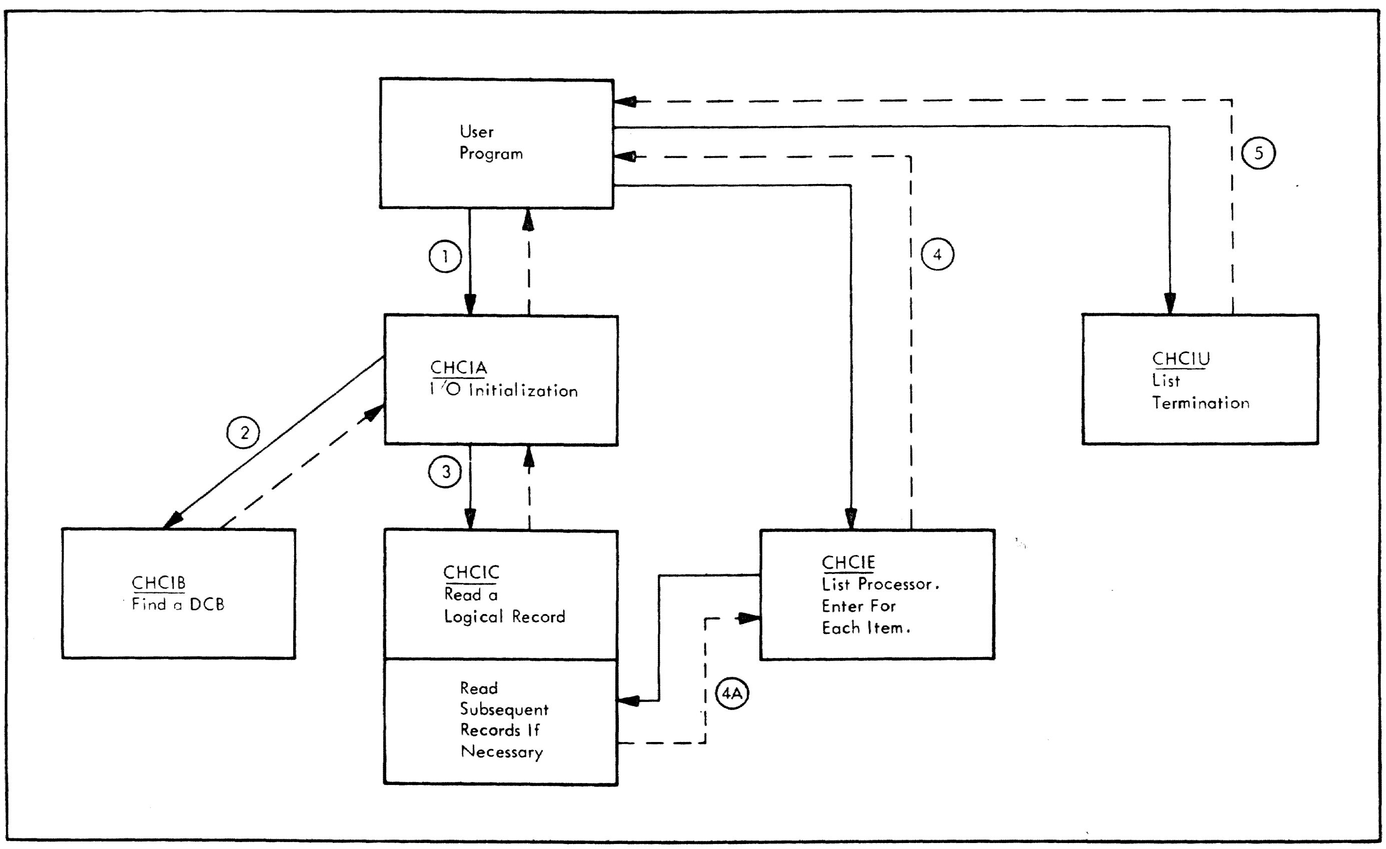

Figure 5. Unformatted READ with List

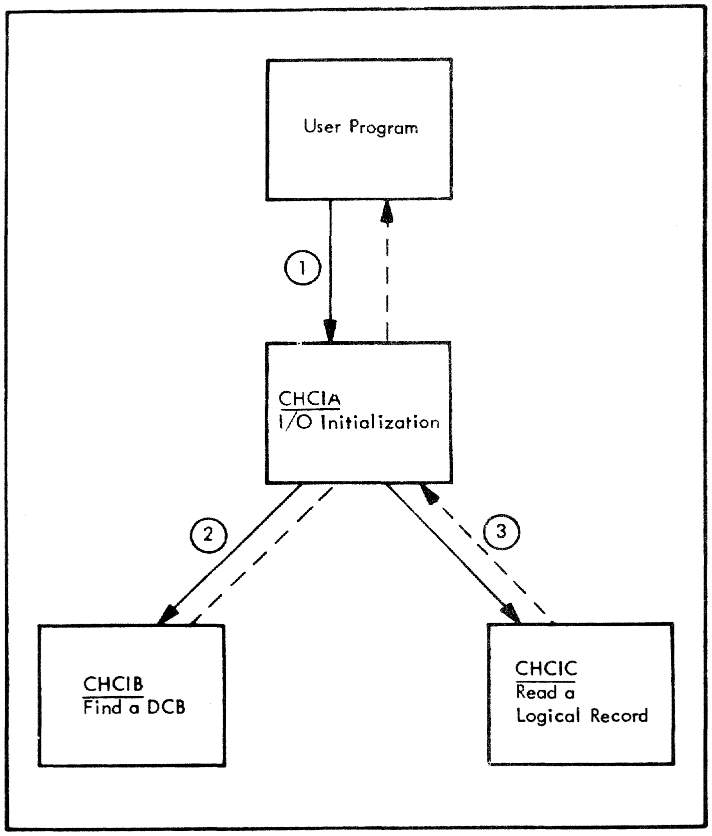

Figure 6. Unformatted READ without List

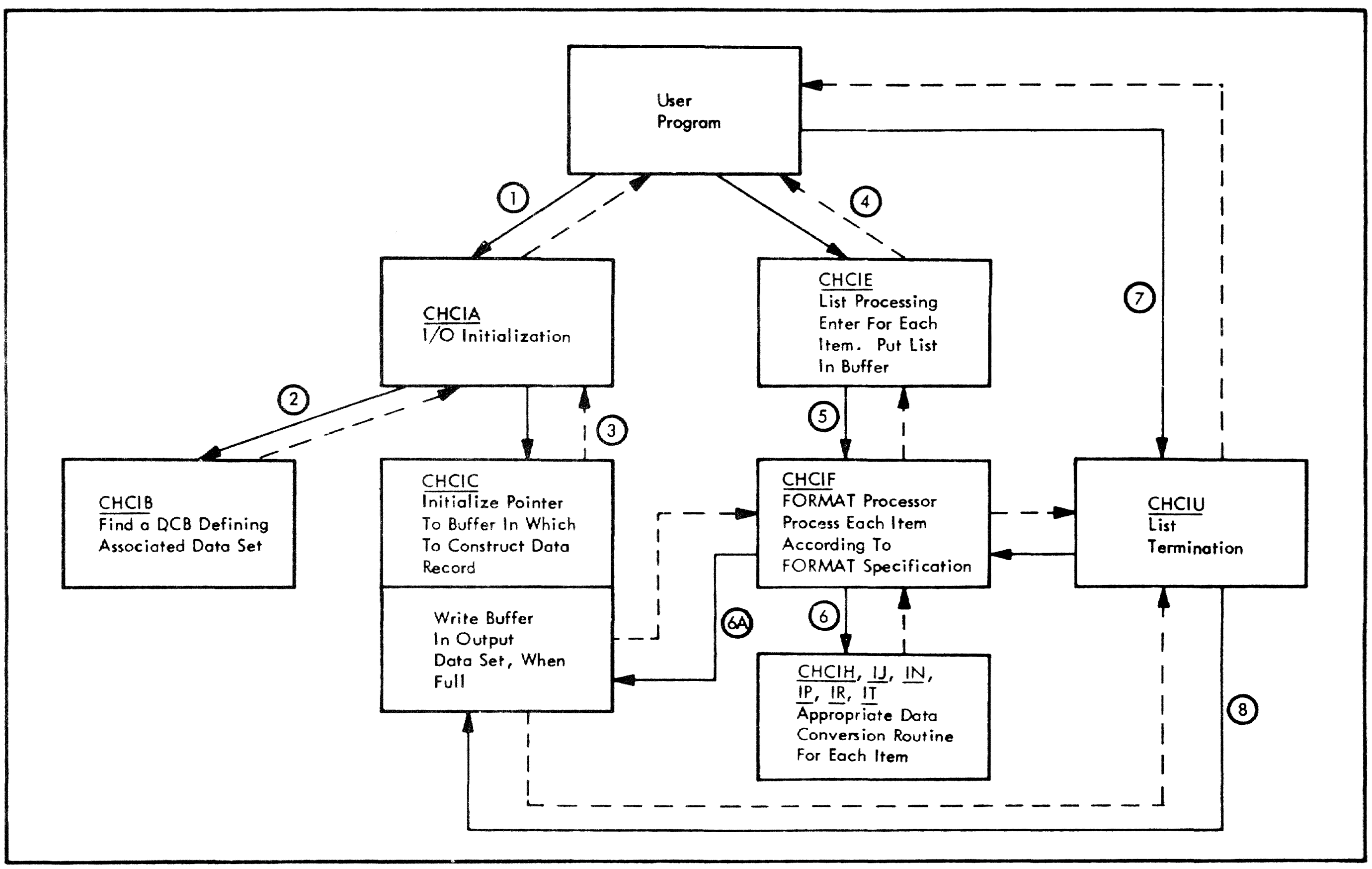

Figure 7. Formatted WRITE with List

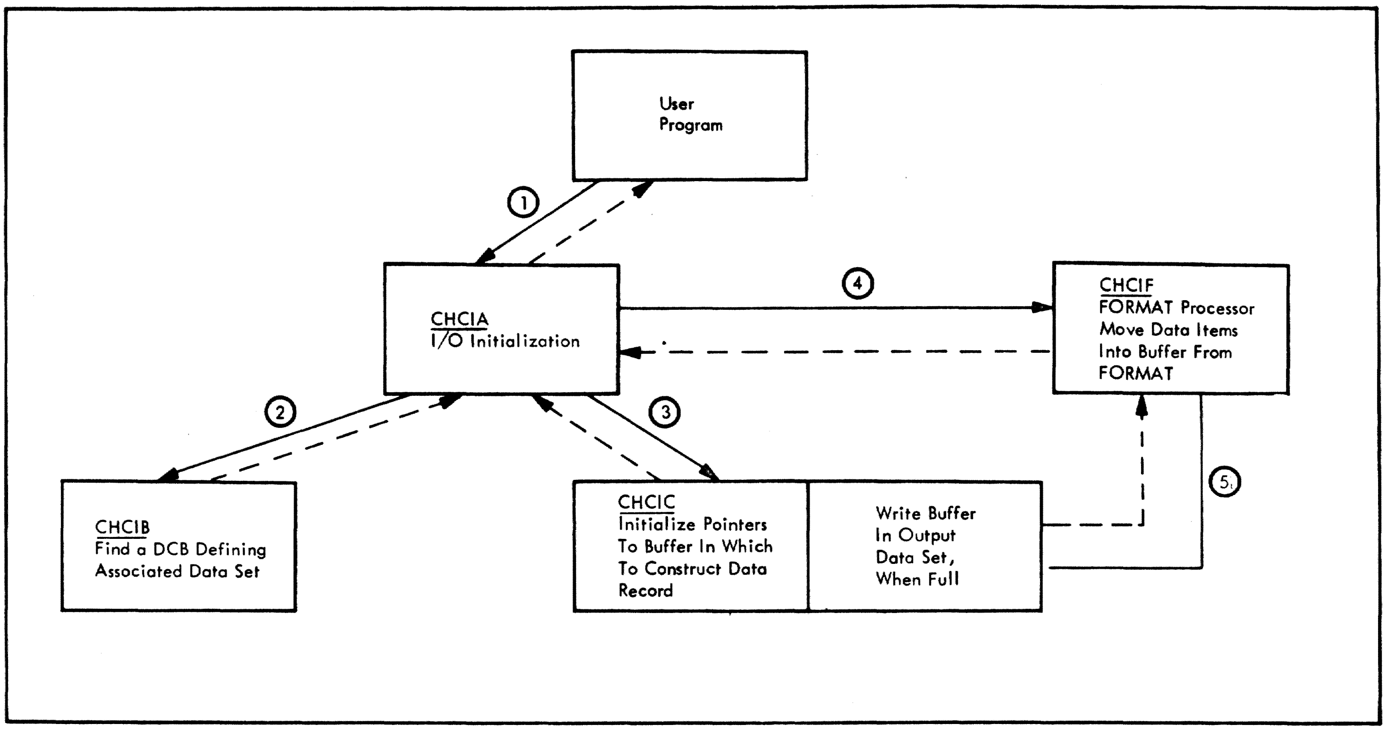

Figure 8. Formatted WRITE without List

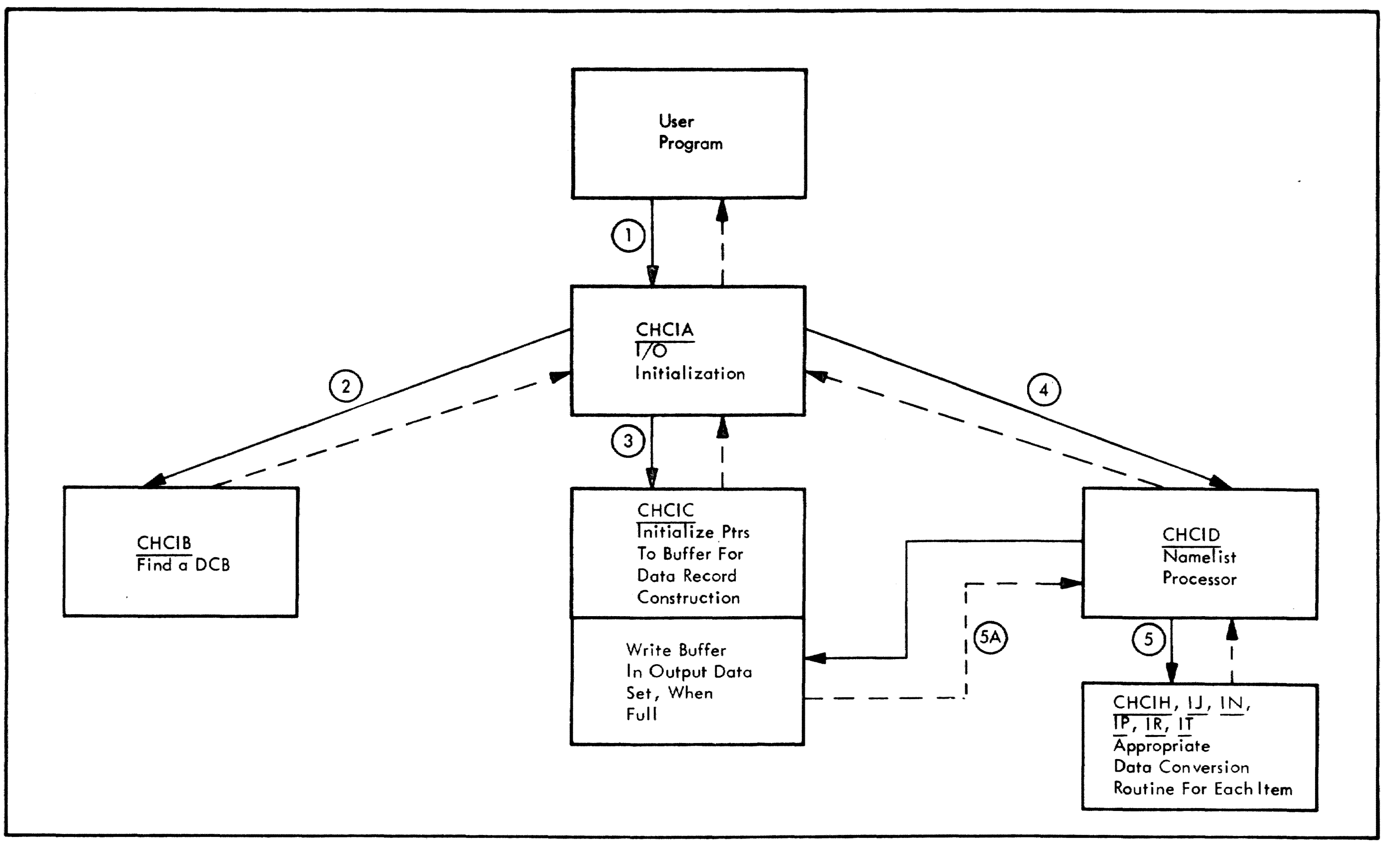

Figure 9. WRITE with NAMELIST

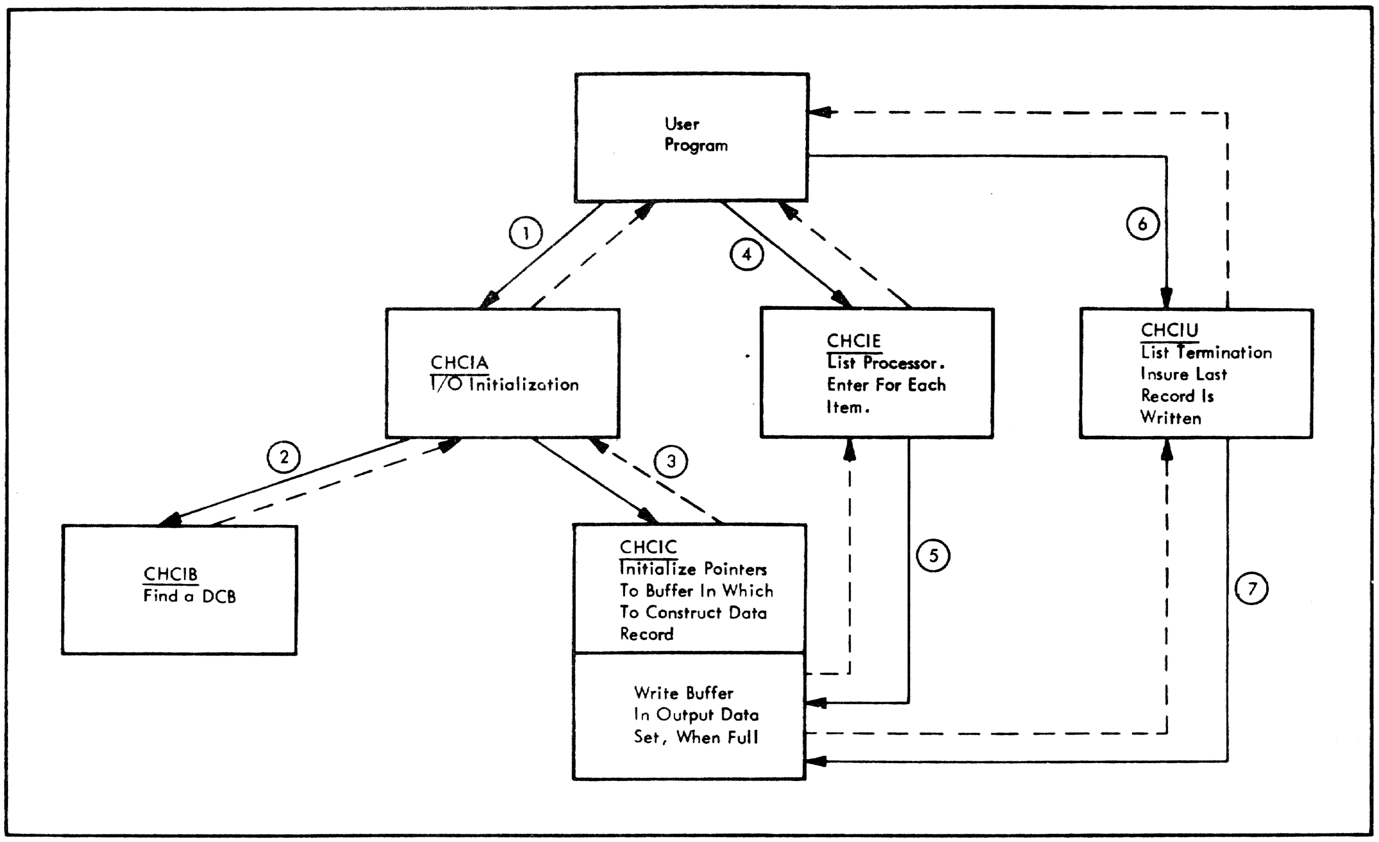

Figure 10. Unformatted WRITE with List

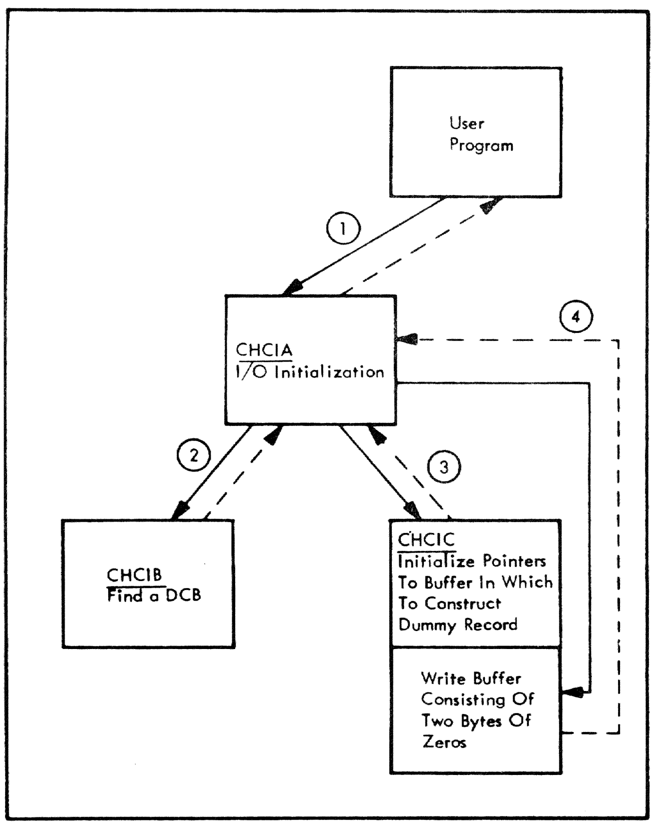

Figure 11. Unformatted WRITE without List

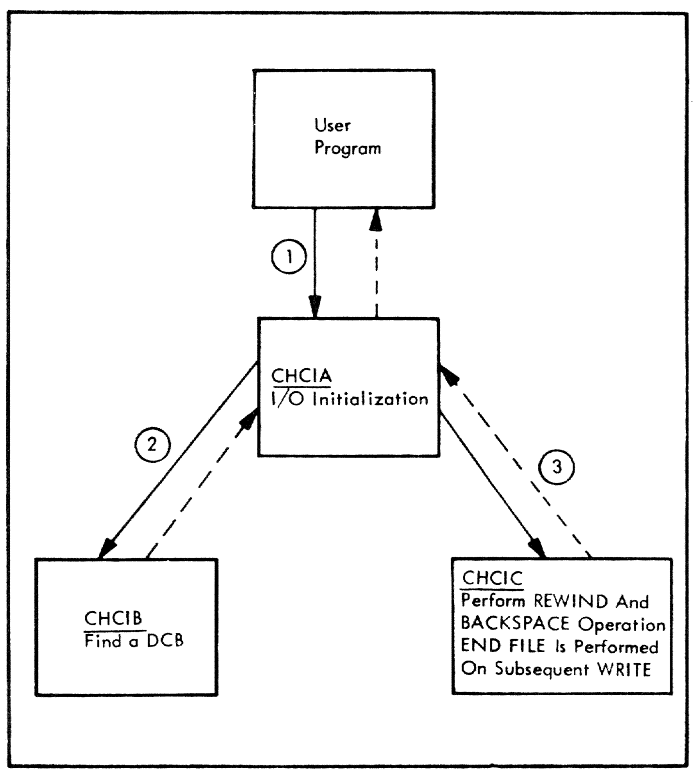

Figure 12. BACKSPACE, REWIND, and END FILE

Routine Descriptions

This subsection identifies the functions, attributes, entries, routines called, error checks and data references of each of the twenty-one FORTRAN I/O routines. The assembler-language user should read this subsection in conjunction with Appendix B.

Certain information is common to most routines; this information includes: a description of the attributes of each routine and parameter-list formats common to data conversion routines.

Attributes

Unless otherwise stated, all FORTRAN I/O routines are nonprivileged, reenterable, closed routines residing in SYSLIB. CHCIA, CHCIE, CHCIU, CHCIW (except at CHCIW4), CHCIV, and CHCBD are entered by standard Type I linkage with the address of a parameter list in register 1, and exit is a return to the calling routine. All of the other I/O routines are entered by restricted Type IV linkage. Unless otherwise stated in the description for a given routine, all routine exits are assumed to be returns to the calling routine.

Data Conversion Routines' Parameter List

All of the data conversion routines have a common parameter list in the I/O common PSECT. Certain data conversion routines do not use all the fields of the parameter list, in which case the fields are set to zero. Table 9 shows the format of the data conversion routines' parameter list and indicates the fields supplied by the appropriate data conversion routine. Note that in some cases the parameters are supplied as part of a common setup but either are not used by the routine itself or are used only to pass on as parameters to other I/O routines.

I/O Initialization—CHCIA

This routine is the initial FORTRAN I/O Library interface with the user. It manages the disposition of each I/O request by setting information switches about formatted and unformatted I/O (for use by other I/O routines), by allocating a buffer area for output requests, and by obtaining a logical record for input requests.

Every FORTRAN source program I/O statement generates a call to this routine. On this call, if there is no list, CHCIA supervises the complete execution of an I/O request. If the I/O request is a READ, WRITE, PRINT, or PUNCH with list, CHCIA simply prepares the I/O library for compiler-generated calls to List Item Processor (CHCIE) and List Termination (CHCIU).

Table 9. Format of Data Conversion Routines' Parameter List

WORD LOCATION	CONTENTS	DATA CONVERSION ROUTINE AFFECTED
Word 1	Address of list item[1]	All
Word 2	Byte 1 Format control character[2]	All
	Byte 2 Scale factor[3]	CHCII and CHCIJ, only
	Byte 3 Scale size[4]	CHCII and CHCIJ, only
	Byte 4 Bits 1 to 4—Byte size of list item, minus one; Bits 5 to 8—Type of list item[5]	All
Word 3	Address of input or output buffer	All
Word 4	Byte size of buffer, minus one	All except CHCIM
Word 5	Decimal fraction width[6]	CHCII, CHCIJ, CHCIN, CHCIS, and CHCIT only

NOTES:

1. A list item is the storage area specified by a list parameter in the READ or WRITE statement.
2. G, E, I, F, D, L, Z, A, H, X, T, or P (N indicates NAMELIST).
3. CHCII or CHCIJ tests for an EBCDIC minus sign, which indicates a negative scale factor. Anything else indicates positive scale factor.
4. The integer preceding the 'P'.
5. Where 01, 02, 03, and 04 represent logical, integer, real, and complex, respectively.
6. The number of decimal places to the right of the decimal point.

Entry: The entry point is CHCIA1. The parameter list is variable-length and has the following format:

Word 1	Address of a fullword containing the user-specified data set reference number.
Word 2	Address of a control byte indicating type of operation.[1]
Word 3	Address of a control byte indicating whether a list was present in the I/O statement and whether any of the following parameters in this list are present.[2]
Word 4 (Optional)	Address of a FORMAT string or NAMELIST table. This address is included in this parameter list only if the user-requested I/O operation had an associated FORMAT or NAMELIST source statement.[3]
Word 5 (Optional)	Address of an error exit. This address is included only if the user-requested I/O operation had the ERR operand specified in his source statement.
Word 6 (Optional)	Address of an end-of-file exit. This address is included only if the user-requested I/O operation has the END operand specified in his source statement.

NOTES:

1. In that control byte, READ = 128 (X'80'), WRITE = 64 (X'40'), PRINT = 32 (X'20'), PUNCH = 16 (X'10'), REWIND = 8 (X'08'), BACKSPACE = 4 (X'04'), END FILE = 2 (X'02').
2. In that control byte, the configuration is: flnrdxxx, where f = FORMAT statement, l = LIST parameter, n = NAMELIST statement, r = ERR operand, d = END operand, and xxx bits are always set to zero. Setting any of the first five bits to one indicates that the corresponding elements are present.
3. The FORMAT string is in user-written form, beginning with the first parentheses, minus the statement number and the word 'FORMAT'. See the CHCID routine description for the details of the NAMELIST table.

If any optional parameter is missing, any parameters following it are moved up in the list and the list is shortened. For example, if there is no FORMAT or NAMELIST address and no error exit address, word 4 of the parameter list would be the end-of-file exit address.

Routines Called:

- DCB Maintenance (CHCIB)
- I/O control (CHCIC)
- FORMAT Processor (CHCIF)
- NAMELIST Processor (CHCID)
- PRMPT (CZATJ1)
- Exit (CHCIW)

Error Checks: If the user-specified data set reference number is negative, an error message is issued by the PRMPT facility, and CHCIW is entered to terminate the user program.

Data References:

- Parameter lists for the modules called by this module.
- A chained list of save areas to accommodate all possible calls to other modules.
- A table of adcons pointing to items in the work areas of other modules that are to be initialized.
- The DCB prefix (generated by DCB maintenance—CHCIB) to be set with the input parameters from this module.

DCB Maintenance—CHCIB

This routine finds or initializes the data control block (DCB) that contains a description of the data to be transmitted by a user-specified I/O operation. If an appropriate DCB is not found, this routine allocates the necessary space in the DCB table and constructs a new DCB, including within it information about the data to be transmitted that the user defined in his DDEF command.

Entry: The entry point is CHCIB1. The parameter list in the common PSECT is fixed-length and has the following format:

Word 1	Address of a fullword containing the user-supplied data set reference number.
Word 2	The data set reference number.

CHCIA stores the address of the user-supplied data set reference number and the data set reference number itself, if present, in the I/O common PSECT.

Routines Called:

- Data management routines used to search for and read JFCB[1] (CZAEB)
- Data management routines used to allocate storage for DCB construction (CZCGA)
- PRMPT (CZATJ1)
- Exit (CHCIW)

Error Checks: If the user-specified data set reference number exceeds 99, an error message is issued by the PRMPT facility (CZATJ1) and CHCIW is entered to terminate the user program.

If a discrepancy exists in the user DDEF command between permissible RECFM, KEYLEN, and DSORG values, an error message is issued by the PRMPT facility (CZATJ1) and CHCIW is called to terminate processing. A description of the assumption FORTRAN I/O makes in initializing associated DCBS contained in Appendix C.

[1] The Job File Control Block (JFCB) is a system control block constructed for each data set at DDEF time. It contains information that must be referred by access method routines or volume mounting routines while the data set is OPEN, and provides a hierarchy of pointers defining JOBLIB, USERLIB, and SYSLIB.

Data References:

- Parameter lists for the routines called by CHCIB.
- Pointers to the DCB table which consists of the DCB Prefix, the DCB itself, and additionally two DECBS if the user has specified Basic Sequential Access Method (BSAM) in his DDEF command.
- A chained list of save areas to accommodate all possible calls to other routines.

Format and Content of the DCB Prefix: The DCB prefix is used by the FORTRAN I/O routines, in conjunction with the DCB, when performing any type of I/O operation. The DCB prefix, created by CHCIB, is eight words long and always immediately precedes the DCB itself.

Table 10. Format and Content of DCB Prefix

Word 1		The address of the starting location in the buffer area for the current logical record.
Word 2		The address of the current location in the buffer area for the current logical record.
Word 3		The address of the end location in the buffer area for the current logical record.
Word 4	Byte 1:	Current operation (READ, WRITE, etc.)[1]
	Byte 2:	Control flags (FORMAT, NAMELIST, List, ERR exit, END exit)[2]
	Byte 3:	Control flags (Span, GATE, recent READ, END or ERR encountered)[3]
	Byte 4:	Previous operation (byte 1 from last call on CHCIC with this DCB)
Word 5		The address of current DECB, if required (BSAM)
Word 6		The user-specified data set reference number, plus one.
Word 7		The address of the next DCB.
Word 8		Save area for the address of the previous DCB for that data set reference number.
Word 9		DCB begins here.

NOTES:

1. See parameter list at entry to CHCIA, Note 1.
2. See parameter list at entry to CHCIA, Note 2.
3. The configuration is: gxdrxxln, where g = GATE I/O, d = end of data set (END), r = error (ERR), l = span from last record, or recent READ, and n = span to next record. The x bits are always set to zero. All bits set to zero signifies that there is no span. (*Spanning* is used in the case of unformatted records, where a physical block size was defined. It is the process of jumping from the end of one record to the beginning of the next.)

I/O Control—CHCIC

This routine fulfills I/O requests made through other I/O library routines by using the data management macro instruction facilities of TSS. The particular data management facilities to be used are determined both by the type of I/O statement issued in the user program, and by any related DDEF commands, if any, defining such things as the type of records being transferred and the manner in which they should be processed.

The following list identifies the more significant macro instructions used by CHCIC for each of the FORTRAN I/O statements.

FORTAN I/O STATEMENT	CHCIC FUNCTION
READ	Obtains a logical record from a user-specified input source by using the READ, GATRD, or GET macro instruction.
WRITE	Initializes the writing of a logical record by establishing pointers to the output buffer area. Subsequent output processing is performed by using the WRITE, GATWR, or PUT macro instruction.
REWIND	Repositions the user-specified volume of one or more data sets to the first record of the first data set by using the POINT or SETL macro instruction.
BACKSPACE	Repositions the user-specified data set to the previous logical record by using the NOTE, POINT, SETL, and BSP macro instructions.
END FILE	Defines the end of the user-specified data set by using the WRITE and STOW macro instructions.

Entry: The entry point is CHCIC1.

Routines Called:

- DCB Maintenance (CHCIB)
- Exit (CHCIW)
- Data management routines to perform I/O functions as determined by the macro instruction issued.
- Error message control (CHCIX)
- PRMPT (CZATJ1)

Error Checks: If the I/O operations performed by data management cause either a SYNAD[1] or EODAD[2] exit, and if the user provided an ERR or END return point, CHCIC locates the adcons for these return points in the work area CHCRWW and locates the register save area for the user's program registers. Return is then made to the ERR or END return point rather than to the calling I/O routine.

If the user did not provide return points (or if the operation was other than a READ statement), an error message is issued and the program is terminated.

If an invalid character is encountered in hexadecimal input from a GATE[3] read operation performed for an unformatted READ statement, an error message is issued and the erroneous character is treated as the termination of the hexadecimal input. Processing then continues.

[1] SYNAD: synchronous error exit address, for automatically transferring control to a user-supplied routine if an uncorrectable I/O error occurs.
[2] EODAD: end of data set address, for automatically transferring control to an end-of-data routine when end of an input data set is detected during processing.
[3] GATE I/O is input from SYSIN or output to SYSOUT.

In addition to the above error checks, error messages are issued (PRMPT macro instruction) and the user program is terminated by CHCIW for any of the following reasons:

- The record is not format-V for unformatted READ statement.
- Error return code received from the use of the FIND or STOW macro instruction for a member in a VPAM data set.
- Invalid sequence of I/O operations for a user-specified data set reference number. The invalid sequences are: READ preceded by END FILE; END FILE preceded by READ; and READ preceded by WRITE (except when using GATE I/O).

Data References:

- References to the standard DCB and its associated DCB prefix.
- A chained list of save areas to accommodate all possible calls to other routines needed.

NAMELIST Processor—CHCID

This routine interacts with CHCIC to control the I/O for each NAMELIST record and interacts with the appropriate data conversion routines to bring about the desired item-by-item conversion.

Entry: The entry point is CHCID1. The parameter list consists of a single word:

Word 1	Address of the NAMELIST table generated by the FORTRAN compiler as part of the user object program.

Routines Called:

- I/O Control (CHCIC)
- Complex Input Conversion (CHCIM)
- Complex Output Conversion (CHCIN)
- General Input Conversion (CHCIS)
- General Output Conversion (CHCIT)
- PRMPT (CZATJ1)
- Exit (CHCIW)

Error Checks: There are no error checks for output. For input, if errors are detected in the NAMELIST table, a message is issued via PRMPT and CHCIW is called to terminate the user program. Other error messages are generated for any of the conditions listed below. In these cases, processing continues with the next entry of the input record.

- Name exceeds six characters
- First character of each input record is not blank
- Subscripts appear on a name that is not an array name
- Incorrect number or range of subscripts
- Subscripting causes array size to be exceeded
- Multiple constants or repeated constants appear with a name that is not a subscripted array name, or exceed the size of an array
- An equal sign or left parenthesis is not preceded by the variable or array name for that item.
- An invalid character appears in a repeat constant
- End of a logical record caused an item to be logically incomplete
- The NAMELIST name is not in the NAMELIST table.

Data References:

- Parameter lists for other I/O library routines called by this routine.
- A chained list of save areas to accommodate all possible calls to other routines needed.

NAMELIST Table: The address of the NAMELIST table generated by the FORTRAN compiler or by the assembler-language programmer is communicated in the call to I/O Initialization (CHCIA) and then passed to this routine. The table is made up of two-word entries, each of which contains an identifier in the first halfword.

NAMELIST NAME ENTRY:
Bytes 0-1: Identifier (X'0100')
2-7: Name (left-justified)

VARIABLE NAME ENTRY:
Bytes 0-1: Identifier (X'0200')
2-7: Name (left-justified)

VARIABLE TYPE AND LOCATION ENTRY:
Bytes 0-1: Identifier (X'0300')
2: Length and Type (4 bits each)
Length: Number of bytes minus 1
Type: X'01' Logical
X'02' Integer
X'03' Real
X'04' Complex
3: Class: Letter A for array; otherwise, an S
4-7: Storage Location

ARRAY SIZE ENTRY:
Bytes 0-1: Identifier (x'0400')
2-3: Not used
4-7: Number of bytes in array

DIMENSION PRODUCT ENTRY:
Bytes 0-1: Identifier (x'0500')
2-3: Not used
4-7: Dimension Product (see explanation below)

TERMINAL ENTRY:
Bytes 0-3: Zero
4-7: Not used

A *dimension* is a level of subdivision, or level of subscripting, within an array. For example, an array could be a string of *seven* thirty-word elements (first dimension), each subdivided into *six* five-word elements (second dimension), each subdivided into *five* one-word elements (third dimension). An array may have as many as seven dimensions.

For each dimension there is a corresponding *dimension product,* which is the product of 1) the byte-size of the array's smallest element, 2) the number of elements within all lower dimensions except the first dimension, and 3) the number of elements within that dimension. In the example just given, the dimension product for the third dimension would be 4 x 6 x 5, or 120. This dimension product would be seven times greater if there were another dimension before the seven-element dimension. The dimension product for the first dimension is always the byte-size of the array's smallest element—this dimension product is never entered. If there is only one level of subdivision, there should be no Dimension Product Entry.

Following is a hexademical representation of the NAMELIST table for a three-dimension array such as that described above, where the array is named 'C' and contains real numbers. The NAMELIST name is LIST.

01 E2	00 E3	D3 40	C9 40	NAMELIST name
02 40	00 40	C3 40	40 40	Array name
03 00	00 0E	33 63	C1 74	Variable type
04 00	00 00	00 03	00 48	Array size
05 00	00 00	00 00	00 18	Dimension product
05 00	00 00	00 00	00 78	Dimension product
00	00	00	00	Terminal entry

ist Item Processor—CHCIE

Every I/O statement in the user's source program generates one or more calls to this routine if there is a list associated with a READ, WRITE, PRINT, or PUNCH. A list item may be a simple variable, an array element (a subscripted variable), or an entire array. If a FORMAT statement is specified, this routine calls on Format Processor (CHCIF) to control any necessary conversion. If there is no FORMAT statement, CHCIE is directly responsible for filling or emptying the output or input buffer area.

Entry: The entry point is CHCIE1. Register 0 contains either zeros, if the list item is a single element, or a number expressing the array length, in bytes, if the list item is an entire array. The parameter list is fixed-length and has the following format:

Word 1	Address of a control byte. The first four bits of the control byte contain the size of the element, minus one. The second four bits contain a flag indicating the type of item as follows: *Flag* — *Type of Item* 01 — logical 02 — integer 03 — real 04 — complex
Word 2	Address of a first (or only) element of the list item.

Routines Called:
- Format Processor (CHCIF)
- I/O Control (CHCIC)
- PRMPT (CZATJI)
- Exit (CHCIW)

Error Check: With unformatted input, if a list item is requested after the logical record is exhausted, an error message is transmitted to the user via PRMPT, and CHCIW is called to terminate the user-program.

Data References:
- Parameter lists for other I/O library routines called by CHCIE.
- A chained list of save areas to accommodate all possible calls to other routines needed.
- A fullword, CHCIB2, which is in the CHCIB work area and contains the address of the DCB prefix.
- The first fifteen bytes of the DCB prefix.

FORMAT Processor—CHCIF

This routine interacts with CHCIC to control the I/O for each FORMAT-referenced record, and interacts with the appropriate data conversion routines to bring about the item-by-item conversion specified by the FORMAT statement.

Entry: Before the first entry to CHCIF to process a reference to a FORMAT statement, CHCIA (or the assembler-language programmer, if he is bypassing CHCIA) does the following:
- Store the address of the FORMAT character string in CHCRWW. The statement number and the word 'FORMAT' are omitted from the string.
- Set to zero the second and third words of CHCIFW.

The entry point is CHCIF1. The parameter list is fixed-length and has the following format:

Word 1	Address of the list item, if any. Zero indicates that no list item was specified.
Word 2	Byte size of list item and type in low order byte of word. (See word 1 of CHCIE parameter list.)
Word 3	Address of the start of the format string.

Routines Called:

- I/O Control (CHCIC)
- Error Message Control (CHCIX)
- Exit (CHCIW)
- One of the eleven data conversion routines (CHCIH through CHCIT)

Error Checks: Since FORMAT statements may be dynamically modified, certain error conditions may arise due to the syntax of the FORMAT string. If there are no syntax errors, errors could arise due to conversion of the data. In such cases the conversion routines issue messages describing the errors before returning. All syntax error checks produce messages describing the error.

Processing is terminated upon encountering invalid control characters in the string, strings that exceed the maximum, or too many levels of parentheses. When it is possible to assume values other than those specified (as in the case of invalid size of *w* or *d* fields after a control character), processing will continue on the current item after the error message is issued. Otherwise, the erroneous FORMAT item is skipped and processing continues with the next control character.

Data References:

- Parameter lists for the routines called by CHCIF.
- A chained list of save areas to accommodate all possible calls to other routines.
- Counters for any repetition and scale factors encountered.

Integer Output Conversion—CHCIH

This routine converts a two-byte or four-byte binary list item to an integer field in the output buffer, according to the format I*n*, where *n* is the integer field size.

Entry: The entry point is CHCIH1. The parameter list is described at the beginning of this subsection, under "Data Conversion Routine Parameter Lists."

Routines Called:

- Error Message Control (CHCIX)

Error Checks: If the output buffer area is too small to contain the integer field, the field is filled with asterisks and a message is issued by CHCIX.

Data References:

- A parameter list for CHCIX.
- A save area to accommodate the call to CHCIX.
- A work area, CHCIHW, to be used by this routine.

Real and Integer Input Conversion—CHCII

This routine converts a data field in an input buffer to the appropriate type list item. An integer field in the input buffer is converted to a binary list item. A real field in the input buffer is converted to a single- or double-precision floating-point list item. The integer field has a format I*n*, where *n* is the field width. The real field has a format F*w.d*, E*w.d*, or D*w.d*, where *w* is the field width and *d* is the width of the decimal fraction.

Entry: There are three entry points: CHCII, CHCIK, and CHCIG. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called:

- Error Message Control (CHCIX)

Error Checks: If the format specification (F, E, D, or I) is improperly specified or the data field is greater than the permissible range, CHCIX is called.

Data References:

- A parameter list for CHCIX.
- Adcons for the table in Real Output Conversion (CHCIJ) that contains powers of ten.
- A work area, CHCIIW, containing: two doubleword areas for calls to CHCIK and CHCIG, and a 32-byte area for temporary storage.

Real Output Conversion—CHCIJ

This routine converts a single- or double-precision floating point list item to a real field in the output buffer. The real field has a format of either E*w.d*, D*w.d*, or F*w.d* where *w* is the field width and *d* is the size of the fractional position, in digit positions.

Entry: There are two entry points: CHCIJ1 and CHCIL1. The parameter list is at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called: There are no calls that can occur besides the final return to the calling routine.

Error Checks: If the output buffer area is too small to contain the real field, the real field is filled with asterisks.

Data References:

- A table of power of ten in double-precision floating-point. It has an external name CHCIL2, so that it can be referred to and used by other I/O library routines. The table structure is:

```
CHCIL2 DC D'1E1,1E2,1E3,1E4,1E5,1E6,1E7,1E8,1E9,1E10'
       DC D'1E11,1E12,1E13,1E14,1E15,1E16,1E17,1E18,
          1E19,1E20'
       .
       .
       .
       DC D'1E71,1E72,1E73,1E74,1E75,1E-76,1E-77,
          1E-78'
```

Complex Input Conversion—CHCIM

This routine converts a complex data field from an input buffer to a complex list item, consisting of two real data fields. Each real field is converted to a single- or double-precision floating-point list item according to the format F*w.d*, E*w.d*, or D*w.d*, where *w* is the real field width and *d* is the width of the decimal fraction.

Entry: The entry point is CHCIM1. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called:
- Real and Integer Input Conversion (CHCII)
- PRMPT (CZATJI)

Error Checks: If only one or if no real fields exist in the complex data field in the input buffer, or if there is a missing parentheses or central comma, CHCIM issues an error message via PRMPT. No further action is taken and the list items remain unchanged. If either or both real fields contain invalid characters or exceed the permissible magnitude range, CHCII assumes the responsibility for producing an error message.

Data References:
- Parameter lists for routines called by CHCIM.
- Adcons for the table produced by CHCIJ, containing powers of ten.
- A chained list of save areas to accommodate all possible calls to other routines.

Complex Output Conversion—CHCIN

This routine converts a complex list item consisting of two, single- or double-precision floating point items to a complex data field in an output buffer. Each floating point list item is converted to a real data field according to the format code F*w.d*, E*w.d*, D*w.d*, or G*w.s*, where *w* is the real field width, *d* is the width of the decimal fraction, and *s* is the number of significant digits.

Entry: The entry point is CHCIN1. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called:
- General Output Conversion (CHCIT)

Error Check: If the FORMAT specifications (F, E ,D, or G) is improperly specified or the real data field is greater than the permissible range, the general output conversion routine (CHCIT) assumes the responsibility for producing an error message.

Data References:
- Parameter list for CHCIT.
- Adcons for the table produced by CHCIJ, containing powers of ten.
- A chained list of save areas to acommodate all possible calls to other routines needed.

Alphameric and Hexadecimal Input Conversion—CHCIO

This routine transfers a specified number of bytes (alphameric or hexadecimal characters) from an input buffer area to a list item. The format is A*w* (alphameric) or Z*w* (hexadecimal), where *w*, field width, is the number of characters being transferred.

Entry: The entry points are CHCIO1 (alphameric data) and CHCIO2 (hexadecimal data). The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called: None.

Error Checks: None.

Data References: None.

Alphameric and Hexadecimal Output Conversion—CHCIP

This routine transfers a specified number of bytes (alphameric or hexadecimal characters) to an output buffer area from a list item. The format is A*w* (alphameric) or Z*w* (hexadecimal), where *w*, field width, is the number of characters being transferred.

Entry: The entry points are CHCIP1 (alphameric data) and CHCIP2 (hexadecimal data). The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called: None.

Error Checks: None.

Data References: None.

Logical Input Conversion—CHCIQ

This routine converts a logical field in the input buffer area. The logical field has the format L*w*, where *w* is the logical field width.

Entry: The entry point is CHCIQ1. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called: None.

Error Checks: None.

Data References: None.

Logical Output Conversion—CHCIR

This routine converts a list item to a logical field in the output buffer area. The logical field has the format L*w*, where *w* is the logical field width.

Entry: The entry point is CHCIR1. The parameter list is described at the beginning of this subsection, under "Data Conversion Routine Parameter Lists."

Routines Called: None.

Error Checks: None.

Data References: None.

General Input Conversion—CHCIS

This routine converts a data field in the input buffer to a list item according to the format G*w.s*, where *w* is the field width and *s* is an optional specification of the number of significant digits.

Entry: The entry point is CHCISI. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called:

- Real and Integer Input Conversion (CHCII)
- Logical Input Conversion (CHCIQ)
- Alphameric Input Conversion (CHCIO)

Error Checks: CHCIS performs no error checking. Error checks, if any, are made by the called data conversion routines.

Data References:

- Parameter lists for the routines called by CHCIS.
- A chained list of save areas to accommodate all possible calls to other routines.

General Output Conversion—CHCIT

The routine converts a list item to a data field in the output buffer, according to the format G*w.s*, where *w* is the field width and *s* is an optional specification of the number of significant digits.

Entry: The entry point is CHCITI. The parameter list is described at the beginning of this subsection under "Data Conversion Routine Parameter Lists."

Routines Called:

- Integer Output Conversion (CHCIH)
- Real Output Conversion (CHCIJ)
- Logical Output Conversion (CHCIR)

Error Checks: CHCIT performs no error checks. Discrepancies between the size and type specification of the list item and the data field are detected by the called conversion routine.

Data References:

- Parameter lists for the routines called by CHCIT.
- A chained list of save areas to accommodate all possible calls to other routines.

List Termination—CHCIU

This routine terminates list processing for a READ, WRITE, PRINT, or PUNCH statement, and completes any I/O operation that is pending.

Entry: The single entry point is CHCIUI. No parameters are passed.

Routines Called:

- Format Processor (CHCIF)
- I/O Control (CHCIC)

The final return is made with registers unchanged, except that register 13 will be set to the address of the calling module's PSECT and register 15 will be set to zero.

Error Check: None

Data References:

- Parameter lists for other I/O library routines called by CHCIU.
- A chained list of save areas to accommodate all possible calls to other routines.
- A control byte within the DCB prefix that describes the current operation. (See "DCB Maintenance—CHCIB" and Table 10, "Format and Content of the DCB Prefix," in this section.)

Exit—CHCIW

The Exit Routine's subprograms, STOP, EXIT, and PAUSE, are described in Section 2.

Error Message Control—CHCIX

This routine receives the text of error messages from other I/O library routines during execution, and delivers those messages as output via the GATE macro instruction, to the user's SYSOUT. In conversational mode, for example, any error message generated is passed to this routine for transmission to the user's terminal.

Entry: The entry point is CHCIXI. The parameter list is fixed-length and has the following format:

Word 1	Address of first part of message.
Word 2	Byte length of first part of message, minus one.
Word 3	Address of second part of message.
Word 4	Byte length of second part of message, minus one.

The first part of each message is a character string that never changes for that message, and is therefore part of the calling routine's CSECT. The second part is some data item that does change (such as the contents of a field containing invalid characters), and which, therefore, is in a PSECT (either of the user's problem program or of the I/O library routines). If only a single part message is to be transmitted to SYSOUT, word 3 of the parameter list is set to zero.

Routines Called:

- GATWR macro instruction

Error Checks: The size of the second part of a message must not be greater than 49 bytes. If this limit is exceeded, only the leftmost 49 bytes of data will be obtained from the invalid field. No error message is generated for this situation.

Data References:

- A 100-byte buffer area used for the error message.
- Parameter lists for the routines called by CHCIX.
- A chained list of save areas to accommodate all possible calls to other routines.

Interruption and Machine Indicator Routine—CHCBD

This routine sets bits in the PSW so that the fixed-point overflow and significance exceptions will be ignored, and directs the system interruption handler where to pass control if any of the following four exceptions occur:

Exception	*Subprogram*
Specification	CHCBE2
Exponent overflow	CHCBE3
Exponent underflow	CHCBE4
Divide check	CHCBE5

In addition, this routine initializes the machine indicator flags and the sense light indicators, and clears any pointers to entries in the DCB table. It then returns control to the calling program.

Entry: The entry point is CHCBD1. There are no entry parameters.

I/O Communication—CHCIY

This table contains space for linking register save areas and an area in which to construct a chain of DCBs.

The format of CHCIY within the I/O PSECT communication region (i.e., save and DCB areas) is:

PSECT Communication Region

0	SAVE 1	6 19-Word Save Areas (Each area has the address of the next save area in word 19)
76	SAVE 2	
152	SAVE 3	
228	SAVE 4	
304	SAVE 5	
380	SAVE 6	
456	460 464 468 472	Pointers to work areas for CHCIA, CHCIE, CHCIU, CHCIV, CHCIW
	CHCIY9	Area for construction of DCBs and DCB prefixes

(THIS PAGE INTENTIONALLY LEFT BLANK)

Appendix A: Replacing FORTRAN IV Library Subprograms

This appendix provides a general description of techniques for replacing a FORTRAN IV library subprogram with a "private" version of the same program. The discussion below does *not* describe a technique for replacing the copy of a subprogram in a manner that will cause *all* users of FORTRAN IV library subprograms to use the new version.

It is recommended that a user-written version be loaded explicitly, with a LOAD command. The FORTRAN IV mathematical subprograms, service subprograms, and I/O subprograms reside in SYSLIB as six link-edited modules, and implicit loading of a user-written version is possible only when the corresponding FORTRAN IV library module is not already loaded.

Many subprograms call other subprograms, as shown in Table 1, Table 8, and Figures 1-11. For example, the CSQRT subprogram, called by a FORTRAN program to find the square root of a COMPLEX*8 number, requires the CABS and SQRT subprograms. If the FORTRAN user loaded his own version of SQRT, the CSQRT subprogram would use this version. Note that if the FORTRAN user wishes the CSQRT subprogram to use his own version of SQRT, he must supply the entire MATHLIB (since it is link-edited). The user may not supply one routine only without performing a new link-edit.

The FORTRAN compiler and the FORTRAN IV library subprograms expect a substituted subprogram to satisfy the same references as the original subprogram. The following table serves as a guide to the external names of each subprogram.

Table 11. External Names of FORTRAN IV Library Subprograms

	MATHEMATICAL SUBPROGRAMS	SERVICE SUBPROGRAMS	I/O SUBPROGRAMS
Entry Name	See Tables 1 and 2.	See Table 6.	See Section 3.
Routine Name	See Tables 1 and 2.	See Table 6.	See Section 3.
CSECT Name	Routine name suffixed by 'W'.	CHCBD and CHCBE: Routine name suffixed by 'W'. CHCIV and CHCIW: Routine name suffixed by 'C'.	Routine name suffixed by 'C'. CHCIB and CHCIC have additional CSECTS with routine name suffixed by 'X'.
PSECT Name	Routine name suffixed by 'R'.	CHCBD and CHCBE: Routine name suffixed by 'R'. CHCIV and CHCIW: Routine name suffixed by 'W'.	Routine name suffixed by 'W'.

Appendix B: Assembler Language Information

The mathematical, service, and I/O subprograms are available to the TSS assembler-language programmer. The following explains the method of calling a library subprogram from an assembler-language program and gives other information for the assembler-language programmer who wants to use these subprograms. Before reading any subdivision of this appendix, the assembler-language programmer should become familiar with the corresponding section of the main text.

NOTE: The examples in this appendix have not been tested on the current system.

The linkage from FORTRAN compiled programs to FORTRAN IV subprograms is a standard, Type I linkage. Assembler-language programmers must link to these subprograms using an identical linkage. The CALL macro instruction provides a number of different means for establishing the correct linkage. (See *Assembler User Macro Instructions.*) A hand-coded linkage may also be used, but such linkages should generally be avoided when macro instructions supply the service required. Regardless of which form of linkage is used, however, the register usages for linkage are:

1. Register 1 must point to whatever parameter list the subprogram requires.
2. Register 13 must point to a 19-word save area in the calling program.
3. Register 14 must contain the address in the calling program to which control will be returned by the called program at the completion of its operation.
4. Register 15 must be loaded with the address of the entry name, and this register is used to transfer control to the called program.

Before returning to the calling program, FORTRAN library subprograms always restore general registers 1 through 14. General register 0 is restored except when the result is returned by a mathematical subprogram and is an integer, in which case the integer is contained in this register. The floating registers are not restored, and should be assumed destroyed. General register 15 is not restored, as future modifications to the FORTRAN library subprograms may make use of this register for a return code (they do not currently do so); this register should be assumed destroyed.

Mathematical Subprograms

The parameter list for a mathematical subprogram must contain the addresses of the arguments in the proper order:

- Directly referenced subprograms. The order is the same as that in the list of operands within the parentheses in the corresponding FORTRAN source statement. For example the source statement

  ANS = SIN (RADIAN)

  in FORTRAN coding corresponds to an assembler-language call containing one address in the parameter list—the address of RADIAN. The FORTRAN statement

  ANS = ATAN2(X,Y)

  produces a linkage with a parameter list containing the addresses of X and Y, in that order. The assembler-language programmer's linkage to ATAN2 must do the same.

- Indirectly referenced subprograms. The order for the exponentiation subprograms is: address of the number to be raised to a power and the address of the power itself.

The arguments pointed to by the parameter list can be either integer values, or normalized floating point real or complex values, as required by the called program. An integer argument occupies four locations of storage. A real argument occupies either four or eight locations of storage. An argument occupying eight locations of storage starts on a doubleword boundary and occupies two adjacent words. The address of the first word is the address of the entire argument.

A complex argument occupies either eight or sixteen locations of storage, starts on a doubleword boundary, and occupies adjacent words. The first half of the argument contains the real part of the complex argument; the second half contains the imaginary part. The address of the real part of the argument is the address of the entire argument.

Each mathematical subprogram returns a single answer—either an integer value, a normalized floating point value, or a complex value. An integer answer is stored in general register 0, a real answer is stored in floating point register 0, and a complex answer is stored in floating point registers 0 and 2. The real and complex parts of a complex number occupying eight storage locations will be in the high-order four storage locations of floating point registers 0 and 2.

Examples of the use of the CALL macro instruction for an assembler-language programmer using the sine program are:

```
         LA       13, SAVE              Point to a 19-
                                        word save area.
         CALL     SIN, (RADIAN)
or
         LA       13, SAVE
         LA       15, VCON
         CALL     (15), MF=(E, PARLIST)
                  .
                  .
                  .
SAVE     DS       19F
PARLIST  DC       A(RADIAN)
VCON     ADCON    IMPLICIT, EP=SIN
```

The above examples produce code equivalent to the following hand-coded linkages. (Several additional instructions are included for greater clarity.) This example assumes that appropriate cover registers have been established, and RADIAN contains the value, in radians, for which the sine is to be obtained.

```
         LA    13, SAVE       Point to a 19-word save area.
         LA    1, PARLIST     Point to the parameter list.
         L     14, RCON       Store the R-con in the 19th
         ST    14, 72 (0, 13)   word of the callers save area.
         L     15, VCON       Obtain the address of the entry
                                point.
         BASR 14, 15          Branch to the entry point, set-
                                ting register 14 to the address
                                of the instruction following
                                the BASR.
         STE   0, ANS         Store the result in ANS.
           .
           .
           .
SAVE     DS    19F            The 19-word save area.
PARLIST  DC    A (RADIAN)     The sine at RADIAN is to be
                                computed.
VCON     DC    V (SIN)        The V-R-con pair for the
RCON     DC    R (SIN)          system entry to the sine
                                program.
RADIAN   DS    F
ANS      DS    F              The result is stored here.
```

Service Subprograms

The calling sequence for DUMP and PDUMP may specify a variable number of parameters. Forms of the CALL macro instruction are available for this purpose. The linkage is identical to that described above, with one exception: immediately preceding the address of the first parameter there must be a word containing, in binary and right adjusted, the number of addresses in the parameter list. Note that this word contains a count, *not* the address of a count.

I/O Subprograms

As with other I/O, data sets used with the FORTRAN I/O library must be defined. Unless the program is using GATE I/O, the programmer must give a DDEF command. For example:

```
DDEF   DDNAME=FT10F001,DSORG=VS,DSNAME=PAY
```

This command is presented in keyword form, for clarity. It could also be written in the shorter, positional form as follows:

```
DDEF   FT10F001,VS,PAY
```

Note that the DDNAME is in FORTRAN format and contains the data set reference number in the two digits following the 'FT.'

Having satisfied DDEF requirements, the programmer is in position to implement the information given in *Section 3: I/O Subprograms.* The following are examples of ways the assembler-language programmer might use FORTRAN I/O facilities.

Formatted READ with List

Assume that the programmer wants to read an eighty-byte record containing three integer numbers in the first half of the record. The first number occupies bytes three through eight, the second occupies bytes fifteen and sixteen, and the third occupies bytes thirty-nine and forty. The rest of the first forty bytes are blank. The second forty bytes are to be ignored.

The numbers are to be converted from character to integer form and placed in storage areas (list items) labeled A, B, and C, respectively.

The programmer chooses not to construct a DCB, since CHCIB (DCB Maintenance) will construct one for him when it finds that there is no DCB for the data set reference number given in the DDEF command.

```
LA     13,SAVE
CALL   CHCIA1, (PARLIST0)        The linkage shown by arrow
                                   4 of Figure 2, to CHCIA
                                   (I/O Initialization).
```

At this point, CHCIA (1) causes CHCIB to create the DCB, (2) causes CHCIC (I/O Control) to perform the I/O, and (3) passes the FORMAT string to CHCIE (List Item Processor).

```
SR          0,0                  Indicate to CHCIE that the
                                   list item is a single
                                   element.
CALL        CHCIE1,(PARLIST1)    The linkage shown by arrow
                                   2 of Figure 2, to CHCIE.
                                   CHCIE will process the
                                   first list item.
SR          0,0
CALL        CHCIE1, (PARLIST2)   The second list item.
SR          0,0
CALL        CHCIE1,(PARLIST3)    The third list item.
CALL        CHCIU1               The linkage shown by arrow
                                   7 of Figure 2, to CHCIU
                                   (List Termination).
                                   There are no parameters.
                 .
                 .
                 .
SAVE        DS         19F
*  PARAMETER LIST FOR CHCIA
PARLIST0    DC     A(DSRN)
            DC     A(CREAD)
            DC     A(COPNDS)
            DC     A(FORMAT)
```

```
         DC      A(LABEL1)    Addresses of the user-
         DC      A(LABEL2)      written error-handling
                                and end-of-file routines.
                                Both parameters are
                                optional.
DSRN     DC      XL4'0A'      The data set reference
                                number ($10_{10}$). The I/O
                                routines expect it to be in
                                fullword, binary form.
CREAD    DC      X'80'        The control byte addressed
                                by the second word of
                                the parameter list.
                                Signfies READ
                                operation.
COPNDS   DC      X'D8'        The control byte addressed
                                by the third word of the
                                parameter list. Specifies
                                that there will be list
                                processing, and that there
                                are entries in the last
                                three words of the
                                parameter list.
         DS      0F           Puts FORMAT string on a
                                fullword boundary.
```

Following is the FORMAT string. Note that the fields are defined in such a way that the numbers are in the *rightmost* portions of the fields. This must always be done with integer conversion, since blanks are treated as zero and would multiply any integer value by ten for every blank on the right.

```
FORMAT    DC     C'(G8,G8,    The FORMAT string.
                 G24)'
*  PARAMETER LISTS FOR CHCIE
PARLIST1  DC     A(ITEM)
          DC     A(A)
PARLIST2  DC     A(ITEM)
          DC     A(B)
PARLIST3  DC     A(ITEM)
          DC     A(C)
ITEM      DC     X'32'        The first four bits of this
                                control byte indicate that
                                the list item (into which
                                an integer will be placed)
                                is four (3+1) bytes long.
                                The second four bits
                                indicate that the
                                characters which the
                                FORMAT statement
                                causes to be read are to
                                be converted into integer
                                form.
```

FORMAT Conversion and List Processing

Assume that the programmer has scanned numbers into HOLD, a 400-byte area. The numbers are in EBCDIC form, with the format xxx.xxx, where 'x' is any digit. They occupy contiguous, two-word elements. The programmer wants to convert them into real form and move the result into a 50-word array. (An array is simply a string of equal-length elements.) The programmer wants to use the FORTRAN I/O library only for its data conversion and list processing facilities, and is not requesting I/O. Thus, the user program will enter the I/O library at the point shown by arrow 4 in Figure 2. Arrows 4-7 show the linkages that will occur.

Note that each doubleword in HOLD contains a blank. It does not matter whether the blank is to the right or to the left, since FORTRAN data conversion will treat it as a zero. (Though if the numbers were whole numbers, it would matter.)

To begin with, the user program stores into the CHCIB work area, at CHCIB2, the address of a parameter list which substitutes for the first four pointers of the DCB prefix.

```
          .
          .
          .
     LA     2,PTRS
     L      3,VCON1
     ST     2,0(0,3)
```

Next, since the user program is bypassing CHCIA, it must store the address of the FORMAT character string into the first word of CHCIFW and zero out the second and third words of CHCIFW.

```
     LA     2,FORMAT
     L      3,VCON2
     ST     2,0(0,3)
     SR     4,4
     SR     5,5
     STM    4,5,4(3)
```

Then comes the usual sequence of code for calling CHCIE.

```
          LA      0,200(0,0)     Indicates that the list item
                                   is an array, and that the
                                   array is 200 bytes in
                                   length.
CALL  CHCIE1,     (CITEM,        Causes the conversion and
                    ARRAY)         movement of data to be
                                   completed.
          .
          .
          .
HOLD      DS      400X
ARRAY     DS      50F            List Item
CITEM     DC      X'33'          The first four bits of this
                                   control byte indicate that
                                   the elements of the array
                                   are four bytes long. The
                                   second four bits indicate
                                   that the data in the buffer
                                   is to be converted to real
                                   form.
PTRS      DC      A(HOLD)        Starting location of raw
                                   data.
          DC      A(HOLD)        Current location. (Same as
                                   starting location.)
          DC      A(HOLD+        End of raw data.
                  400)
          DC      X'80C00000'    First byte indicates a
                                   READ operation. The
                                   second byte indicates a
                                   FORMAT statement
                                   with a list with the
                                   FORMAT statement
                                   not encoded.
VCON1     DC      V(CHCIB2)
VCON2     DC      V(CHCIFW)
FORMAT    DC      C'(50F8.3)'
```

This appendix describes the assumptions that the FORTRAN I/O library makes in initializing DCBs with information concerning record format (RECFM) and data set organization (DSORG). These assumptions are described to help reduce a frequent source of error encountered by the user when performing I/O.

Some introductory material is presented on the DCB describing its general use, contents, and sources of initialization, before discussing the permissible record formats and data set organizations.

DCB Use

The Data Control Block (DCB) is created by DCB Management (CHCIB) and is used by certain data management routines invoked by macro instruction references in I/O Control (CHCIC). The DCB is required for all I/O performed using either BSAM or VAM. However, the DCB is not required for I/O when using the GATE macro instructions.

DCB Content

The DCB contains information such as the DDNAME, type of data set organization, the type and size of records, block size for blocked data sets, number of buffer areas, exits for SYNAD and EODAD, and various control flags used by data management.

DCB Initialization

The FORTRAN I/O routines, when processing an input data set, take advantage of information in the DCB to adapt to the characteristics of the data set and read it correctly. Characteristics are based on the parameters for a DCB that can be supplied from:

- The user program—type of I/O used and associated data format.
- User-supplied DDEF commands—some of the information in the DCB can be changed in this manner; however, the extent of change is limited.
- Input data set labels—these override both of the above sources of information, within limits set by data management.

Combinations of DSORG and RECFM

Table 12 gives the permissible combinations of record formats and data set organizations that may be specified when using the FORTRAN I/O library.

Table 12. Combinations of DSORG and RECFM Values

RECFM	DSORG VALUES				
	VS	PS	VSP	VI	VIP
V	A	A	A	A	A
VB	N	A	N	N	N
VT	N	A	N	N	N
F	A	A	A	A	A
FB	N	A	N	N	N
FS	N	A	N	N	N
FT	N	A	N	N	N
FBS	N	A	N	N	N
FBT	N	A	N	N	N
FBST	N	A	N	N	N
FST	N	A	N	N	N
U	L	A	L	N	N

Codes mean:
A — Acceptable
L — Limited Acceptable
N — Not acceptable

DSORG abbreviations mean:
VS — Virtual sequential (direct-access only)
PS — Physical sequential—BSAM—(any device except terminals)
VSP — Virtual sequential partitioned (like VS)
VI — Virtual index sequential (like VS)
VIP — Virtual index sequential partitioned (like VS)

RECFM abbreviations mean:
V — Variable-length unblocked records
VB — Variable-length blocked records
VT — Variable-length unblocked with track overflow
F — Fixed-length unblocked records
FB — Fixed-length blocked records
FS — Same as F, no truncated blocks or unfilled tracks
FT — Same as F, track overflow
FBS — Same as FB, no truncated blocks or unfilled tracks
FBT — Same as FB, track overflow
FBST — Same as FBS, track overflow
FST — Same as F, no truncated blocks, track overflow
U — Undefined record length

Any of the RECFM codes shown can be followed by the letter A or M. A indicates that the first character of every logical record is an extended ANSI FORTRAN IV carriage or punch control code. M indicates that the first character of every record is a TSS/360 machine control byte. In general, the M option cannot be used by FORTRAN output data sets, since the control codes are unprintable and do not conform to FORTRAN conventions.

Unformatted FORTRAN Logical Records

Under any of the organization techniques used, an unformatted WRITE statement may lead to a logical record that exceeds the length of the maximum record supported by the access method. Furthermore, it is not possible to enter the byte size of the entire FORTRAN logical record into the beginning of the I/O physical record without the possibility of tying up an indefinite amount of virtual storage. Therefore, unformatted FORTRAN logical records may span over data management physical records. In the management of unformatted FORTRAN data, the first two bits of every VS physical record or the third byte of every PS physical record is a control byte defined as follows:

X'00' A FORTRAN logical record does not span into or out of the data management physical record.

X'01' This data management physical record is the first of a span.

X'02' This data management physical record is the last of a span.

X'03' This data management physical record is within the range of a span.

No data management physical record will be written containing more than one unformatted FORTRAN logical record.

Appendix D: DUMP and PDUMP Sample Storage Printouts

This appendix contains a sample printout for each dump format that can be specified for the DUMP and PDUMP subprogram. The printouts are given in the order: hexadecimal, logical *1, logical *4, integer *2, integer *4, real *4, real *8, complex *8, complex *16, and literal.

Table 13. Sample Storage Printouts

```
CONVERSION CODE 0 - HEXADECIMAL
0003E220  C1C2C3C4  C5C6C7C8  C9D1D2D3  D4D5D6D7  D8D9E2E3

CONVERSION CODE 1 - LOGICAL * 1
0003E1B0        T      F      T      F      T      F      F      F      F      F

CONVERSION CODE 2 - LOGICAL * 4
0003E1DC        T      F      T      F      T      F

CONVERSION CODE 3 - INTEGER * 2
0003E1BA                 1             2             3             4             5             6

CONVERSION CODE 4 - INTEGER * 4
0003E1F8                 1             2             3             4             5             6

CONVERSION CODE 5 - REAL * 4
0003E248        1.00000E 00       0.20000E 01       0.30000E 01       0.40000E 01

CONVERSION CODE 6 - REAL * 8
0003E270        1.00000D 00       0.20000D 01       0.30000D 01       0.40000D 01

CONVERSION CODE 7 - COMPLEX * 8
0003E2C0        1.00000E 00       0.20000E 01       0.20000E 01       0.30000E 01

CONVERSION CODE 8 - COMPLEX * 16
0003E310        1.00000D 00       0.20000D 01       0.20000D 01       0.30000D 01

CONVERSION CODE 9 - LITERAL
0003E220  ABCDEFGHIJKLMNOPQRSTUVW
```

Appendix E: Interruption Procedures

This appendix contains descriptions of the procedures followed when the user's program is temporarily interrupted due to certain types of interrupts. *Interrupts* are hardware-originated breaks in the flow of processing. *Program interrupts* result from improper specification or use of instructions and data. The term *exception* is used to refer to these types of interrupts (see *Principles of Operation*). Six such exceptions occur frequently enough during normal FORTRAN programming to warrant special treatment.

1. Fixed point overflow exception
2. Significance exception
3. Exponent overflow exception
4. Exponent underflow exception
5. Floating point-divide exception
6. Specification exception

The procedure for handling these exceptions follows. The compiler generates code at the beginning of all main programs that calls the CHCBDI entry to module CHCBD. At CHCBDI these operations are performed:

1. Bits are set in the PSW such that the fixed point overflow and significance exceptions will be ignored.
2. Initialization is performed such that control will be passed to an entry in module CHCBD or CHCBE if any of the remaining four exceptions occur:

EXCEPTION	ENTRY
Exponent overflow	CHCBD3
Exponent underflow	CHCBD4
Floating point divide	CHCBD5
Specification	CHCBE1

At the first three of these entries, flags are set for later interrogation by routines called as a result of the CALL OVERFL (tests for exponent overflow or underflow exceptions) and CALL DVCHK (tests for floating point divide exception) statements.

A specification exception occurs when a variable is not on a proper word boundary. This condition may exist in a FORTRAN program if an EQUIVALENCE or COMMON statement forces misalignment. The compiler issues a warning diagnostic, but such a misalignment does not prevent the user from executing the program. An installation option specifies that one of two courses of action is to be taken if a specification interrupt occurs: (1) terminate the task, or (2) transfer control to a program that will perform the desired operation, using instructions that will not cause an exception due to incorrect boundary alignment. The routine entered for either of these eventualities is CHCBE, which is entered by the CHCBE1 entry. The installation option is tested, and one of the two above courses of action taken.

An exponent overflow exception is recognized when the result of a floating point addition, subtraction, multiplication, or division is either greater than or equal to 16^{63} (approximately 7.2 x 10^{75}). An exponent underflow exception is recognized when the result of a floating point addition, subtraction, multiplication, or division is less than 16^{-65} (approximately 5.4 x 10^{-79}). A divide exception is recognized when division by zero is attempted.

Appendix F: Algorithms

Information about the computations used in the explicitly called mathematical subprograms is arranged alphabetically in this appendix, according to subprogram module name. The user entry name associated with each subprogram is given in parentheses following the module name.

The information for each subprogram is divided into two parts: a description of the algorithm used and a description of the effect of an argument error upon the accuracy of the answer (function value).

The presentation of each algorithm is divided into its major computational steps; the formulas necessary for each step are supplied. Some formulas are widely known; others are derived from common formulas. In these cases, the process leading from the common formula to the computational formula is sketched in enough detail that the derivation may be reconstructed.[1]

For the sake of brevity, the needed constants are normally given only symbolically. (The actual values can be found in the assembly listing of the subprograms.) Some of the formulas are widely known; those that are not so widely known are derived from more common formulas. The process leading from the common formula to the computational formula is sketched in enough detail so that the derivation can be reconstructed by anyone who has an understanding of college mathematics and access to the common texts on numerical analysis.[1] Many approximations were derived by the so-called "minimax" methods. The approximation sought by these methods can be characterized as follows. Given a function $f(x)$, an interval I, the form of the approximation (such as the rational form with specified degrees), and the type of error to be minimized (such as the relative error), there is normally a unique approximation to $f(x)$ whose maximum error over I is the smallest among all possible approximations of the given form. Details of the theory and the various methods of deriving such approximation are provided in the reference.[1] The accuracy figures cited in the algorithm sections are theoretical, and they do not take round-off errors into account. Minor programming techniques used to minimize round-off errors are not necessarily described here.

The accuracy of an answer produced by these algorithms is influenced by two factors: the performance of the subprogram and the accuracy of the argument. (Performance statistics are given in Table 1.) The effect of an argument error upon the accuracy of an answer depends solely upon the mathematical function involved and not upon the particular coding used in the subprogram.

Because argument errors, whether accumulated prior to use of the subprogram or introduced by newly converted data, always influence the accuracy of answers, a guide to the propagational effect of argument errors is provided. This guide (expressed as a simple formula, where possible) is intended to assist users in assessing the effect of an argument error.

[1] Any of the common numerical analysis texts may be used as a reference. One such text is F. B. Hildebrand's *Introduction to Numerical Analysis* (McGraw-Hill Book Company, Inc., New York, N.Y., 1956). Background information for algorithms that use continued fractions may be found in H. S. Wall's *Analytic Theory of Continued Fractions* (D. VanNostrand Co., Inc., Princeton, N.J., 1948).

These symbols are used in this appendix to describe the effect of an argument error upon the accuracy of the answer:

SYMBOL	EXPLANATION	
g(x)	Result given by subprogram	
f(x)	Correct result	
ϵ	$\frac{f(x)-g(x)}{f(x)}$	Relative error of result given by subprogram
δ	Relative error of argument	
E	f(x)−g(x)	Absolute error of result given by subprogram
Δ	Absolute error of argument	

The notation used for the continued fractions in this appendix complies with the specifications set by the National Bureau of Standards. For more information, see Milton Abramowitz and Irene A. Stegun (editors), *Handbook of Mathematical Functions,* Applied Mathematics Series-55 (National Bureau of Standards, Washington, D.C., 1965).

Although it is not specifically stated below for each subroutine, the algorithms in this chapter were programmed to conform to the following standards governing floating-point overflow/underflow.

1. Intermediate underflow and overflows are not permitted to occur. This prevents the printing of irrelevant messages.
2. Those arguments for which the answer can overflow are excluded from the permitted range of the subroutine. This rule does not apply to CDABS and CABS.
3. When the magnitude of the answer is less than 16^{-65}, zero is given as the answer. If the floating-point underflow exception mask is on at the time, the underflow message will be printed.

Control of Program Exceptions in Mathematical Functions

The FORTRAN mathematical functions have been coded with careful control of error situations. A result is provided whenever the answer is within the range representable in the floating-point form. In order to be consistent with FORTRAN control of exponent overflow/underflow exceptions, the following types of conditions are recognized and handled separately.

When the magnitude of the function value is too large to be represented in the floating-point form, the condition is called a terminal overflow; when the magnitude is too small to be represented, a terminal underflow. On the other hand, if the function value is representable, but if execution of the chosen algorithm causes an overflow or underflow in the process, this condition is called an intermediate overflow or underflow.

Function subroutines in the FORTRAN library have been coded to observe the following rules for these conditions:

1. Algorithms which can cause an intermediate overflow have been avoided. Therefore an intermediate overflow should not occur during the execution of a function subroutine of the library.
2. Intermediate underflows are detected and not allowed to cause an interrupt. In other words, spurious underflow signals are not allowed to be given. Computation of the function value is successfully carried out.
3. Terminal overflow conditions are screened out by the subroutine. The argument is considered out of range for computation and an error diagnostic is given.

4. Terminal underflow conditions are handled by forcing a floating-point underflow exception. This provides for the detection of underflow in the same manner as for an arithmetic statement. Terminal underflows can occur in the following function subroutines: EXP, DEXP, ATAN2, DATAN2, ERFC, and DERFC.

For implicit arithmetic subroutines, these rules do not apply. In this case, both terminal overflows and terminal underflows will cause respective floating-point exceptions. In addition, in case of complex arithmetic (implicit multiply and divide), premature overflow/underflow is possible when the result of arithmetic is very close to an overflow or underflow condition.

Explicitly Called Subprograms

Absolute Value Subprograms

CABS/CDABS

1. Write $|x + iy| = a + ib$.
2. Let $v_1 = \max(|x|, |y|)$, and $v_2 = \min(|x|, |y|)$.
3. If characteristics of v_1 and v_2 differ by 7 (15 for CDABS) or more, or if $v_2 = 0$, then $a = v_1$, $b = 0$.
4. Otherwise,

$$a = 2 \cdot v_1 \cdot \sqrt{1/4 + 1/4 \left(\frac{v_2}{v_1}\right)^2}, \text{ and } b = 0.$$

If the answer is greater than 16^{63}, the floating-point overflow interruption will take place (see Appendix C). The algorithms for both complex absolute value subprograms are identical. Each subprogram uses the appropriate real square root subprogram (SQRT or DSQRT).

Effect of an Argument Error

$$\epsilon \sim \frac{1}{2}\delta.$$

Arcsine and Arccosine Subprograms

ARSIN/ARCOS

Algorithm

1. If $0 \leqq x \leqq 1/2$, then compute arcsin (x) by a continued fraction of the form:

$$\arcsin(x) \cong x + x^3 \cdot F \text{ where}$$

$$F = \cfrac{d_1}{(x^2 + c_1) + \cfrac{d_2}{(x^2 + c_2)}}.$$

The coefficients of this formula were derived by transforming the minimax rational approximation (in relative error, over the range $0 \leqq x^2 \leqq 1/4$) for $\arcsin(x)/x$ of the following form:

$$\frac{\arcsin(x)}{x} \cong a_0 + x^2 \cdot \left[\frac{a_1 + a_2x^2}{b_0 + b_1x^2 + x^4}\right].$$

Minimax was taken under the constraint that $a_0 = 1$ exactly. The relative error of this approximation is less than $2^{-28.3}$.

If $0 \leqq x \leqq 1/2$, arccos(x) is computed as:

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x).$$

2. If $1/2 < x \leqq 1$, then compute arccos(x) essentially as:

$$\arccos(x) = 2 \cdot \arcsin\left(\sqrt{\frac{1-x}{2}}\right).$$

This case is now reduced to the first case because within these limits,

$$0 \leqq \sqrt{\frac{1-x}{2}} \leqq 1/2.$$

This computation uses the real square root subprogram (SQRT)

If $1/2 < x \leqq 1$, arcsin(x) is computed as:

$$\arcsin(x) = \frac{\pi}{2} - \arccos(x).$$

Implementation of the above algorithms (steps 1 and 2) were carried out with care to minimize the round-off errors.

3. If $-1 \leqq x < 0$, then $\arcsin(x) = -\arcsin|x|$

and $\arccos(x) = \pi - \arccos|x|$.

This reduces these cases to one of the two positive cases.

Effect of an Argument Error

$E \sim \frac{\Delta}{\sqrt{1-x^2}}$. For small values of x, $E \sim \Delta$. Toward the limits (± 1) of the range, a small Δ causes a substantial error in the answer. For the arcsine, $\epsilon \sim \delta$ if the value of x is small.

DARSIN/DARCOS

Algorithm

1. If $0 \leqq x \leqq \frac{1}{2}$, then compute $\arcsin(x)$ by a continued fraction of the form:

$$\arcsin(x) \cong x + x^3 \cdot F \text{ where}$$

$$F = c_1 + \frac{d_1}{(x^2+c_2)+} \frac{d_2}{(x^2+c_3)+} \frac{d_3}{(x^2+c_4)+} \frac{d_4}{(x^2+c_5)}.$$

The relative error of this approximation is less than $2^{-57.2}$.

The coefficients of this formula were derived by transforming the minimax rational approximation (in relative error, over the range $0 \leqq x^2 \leqq \frac{1}{4}$) for $\arcsin(x)/x$ of the following form:

$$\frac{\arcsin(x)}{x} \cong a_0 + x^2\left[\frac{a_1 + a_2x^2 + a_3x^4 + a_4x^6 + a_5x^8}{b_0 + b_1x^2 + b_2x^4 + b_3x^6 + x^8}\right].$$

Minimax was taken under the constraint that $a_0 = 1$ exactly.

If $0 \leqq x \leqq \frac{1}{2}$, $\arccos(x)$ is computed as:

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x).$$

2. If $\frac{1}{2} < x \leqq 1$, then compute $\arccos(x)$ essentially as:

$$\arccos(x) = 2 \cdot \arcsin\left(\sqrt{\frac{1-x}{2}}\right).$$

This case is now reduced to the first case because within these limits,

$$0 \leqq \sqrt{\frac{1-x}{2}} \leqq \tfrac{1}{2}.$$

This computation uses the real square root subprogram (DSQRT).

If $\frac{1}{2} < x \leqq 1$, $\arcsin(x)$ is computed as:

$$\arcsin(x) = \frac{\pi}{2} - \arccos(x).$$

Implementation of the above algorithms (steps 1 and 2) were carried out with care to minimize the round-off errors.

3. If $-1 \leqq x < 0$, then $\arcsin(x) = -\arcsin|x|$, and $\arccos(x) = \pi - \arccos|x|$. This reduces these cases to one of the two positive cases.

Effect of an Argument Error

$E \sim \frac{\Delta}{\sqrt{1-x^2}}$. For small values of x, $E \sim \Delta$. Toward the limits (± 1) of the range a small Δ causes a substantial error in the answer. For the arcsine, $\epsilon \sim \delta$ if the value of x is small.

Arctangent Subprograms

ATAN/ATAN2

Algorithm

1. For arctan (x_1, x_2):
   If $x_1 < 0$, use the identity arctan $(x_1, x_2) = -$arctan $(-x_1, x_2)$.
   Hence we may assume that $x_1 \geqq 0$. Then:

   If either $x_2 = 0$ or $\left|\frac{x_1}{x_2}\right| > 2^{24}$, the answer $= \frac{\pi}{2}$.

   If $x_2 < 0$ and $\left|\frac{x_1}{x_2}\right| < 2^{-24}$, the answer $= \pi$.

   For the general case, if $x_2 > 0$, the answer $= \arctan\left(\left|\frac{x_1}{x_2}\right|\right)$, and

   if $x_2 < 0$, the answer $= \pi - \arctan\left(\left|\frac{x_1}{x_2}\right|\right)$.

   The remainder of the computation is identical for either one or two arguments.

2. Reduce the computation of arctan (x) to the case $0 \leqq x \leqq 1$, by using

   $$\arctan(-x) = -\arctan(x), \text{ or}$$

   $$\arctan\left(\frac{1}{|x|}\right) = \frac{\pi}{2} - \arctan|x|.$$

3. If necessary, reduce the computation further to the case $|x| \leqq \tan 15°$ by using

   $$\arctan(x) = 30° + \arctan\left(\frac{\sqrt{3}\cdot x - 1}{x + \sqrt{3}}\right).$$

   The value of $\left|\frac{\sqrt{3}\cdot x - 1}{x + \sqrt{3}}\right| \leqq \tan 15°$ if the value of x is within the range, $\tan 15° < x \leqq 1$. The value of $(\sqrt{3}\cdot x - 1)$ is computed as $(\sqrt{3} - 1)\,x - 1 + x$ to avoid the loss of significant digits.

4. For $|x| \leqq \tan 15°$, use the approximation formula:

   $$\frac{\arctan(x)}{x} \cong 0.60310579 - 0.05160454x^2 + \frac{0.55913709}{x^2 + 1.4087812}.$$

   This formula has a relative error less than $2^{-27.1}$ and can be obtained by transforming the continued fraction

   $$\frac{\arctan(x)}{x} = 1 - \frac{x^2}{3 +}\;\frac{\frac{x^2}{5}}{\left(\frac{5}{7} + x^{-2}\right) - w}$$

   where w has an approximate value of $\left(-\frac{75}{77}x^{-2} + \frac{3375}{77}\right)10^{-4}$, but the true

   value of w is $\dfrac{\frac{4\cdot 5}{7\cdot 7\cdot 9}}{\left(\frac{43}{7\cdot 11} + x^{-2}\right) +}\;\ldots.$

   The original continued fraction can be obtained by transforming the Taylor series into continued fraction form.

Effect of an Argument Error

$E \sim \frac{\Delta}{1 + x^2}$. For small values of x, $\epsilon \sim \delta$; as the value of x increases, the effect of δ upon ϵ diminishes.

DATAN/DATAN2

Algorithm

1. For $\arctan(x_1, x_2)$:
   If $x_1 < 0$, use the identity $\arctan(x_1, x_2) = -\arctan(-x_1, x_2)$.
   Hence we may assume that $x_1 \geqq 0$. Then:

   If either $x_2 = 0$ or $\left|\frac{x_1}{x_2}\right| > 2^{56}$, the answer $= \frac{\pi}{2}$.

   If $x_2 < 0$ and $\left|\frac{x_1}{x_2}\right| < 2^{-56}$, the answer $= \pi$.

   For the general case, if $x_2 > 0$, the answer $= \arctan\left(\left|\frac{x_1}{x_2}\right|\right)$, and

   if $x_2 < 0$, the answer $= \pi - \arctan\left(\left|\frac{x_1}{x_2}\right|\right)$.

   The remainder of the computation is identical for either one or two arguments.

2. Reduce the computation of $\arctan(x)$ to the case $0 \leqq x \leqq 1$ by using
   $\arctan(-x) = -\arctan(x)$ and
   $\arctan\frac{1}{|x|} = \frac{\pi}{2} - \arctan|x|$.

3. If necessary, reduce the computation further to the case $|x| \leqq \tan 15°$ by using
   $$\arctan(x) = 30° + \arctan\left(\frac{\sqrt{3} \cdot x - 1}{x + \sqrt{3}}\right).$$
   The value of $\left|\frac{\sqrt{3} \cdot x - 1}{x + \sqrt{3}}\right| \leqq \tan 15°$, if the value of x is within the range $\tan 15° < x \leqq 1$. The value of $(\sqrt{3} \cdot x - 1)$ is computed as $(\sqrt{3} - 1)x - 1 + x$ to avoid the loss of significant digits.

   The relative error of this approximation is less than $2^{-60.7}$.
   The coefficients of this formula were derived by transforming a minimax rational approximation (in relative error, over the range $0 \leqq x^2 \leqq 0.071797$) for $\arctan(x)/x$ of the following form:
   $$\frac{\arctan(x)}{x} \cong a_0 + x^2\left[\frac{c_0 + c_1x^2 + c_2x^4 + c_3x^6}{d_0 + d_1x^2 + d_2x^4 + x^6}\right].$$
   Minimax was taken under the constraint that $a_0 = 1$ exactly.

4. For $|x| \leqq \tan 15°$, use a continued fraction of the form:
   $$\frac{\arctan(x)}{x} \cong 1 + x^2\left[b_0 - \frac{a_1}{(b_1 + x^2) -} \frac{a_2}{(b_2 + x^2) -} \frac{a_3}{(b_3 + x^2)}\right].$$

Effect of an Argument Error

$E \sim \frac{\Delta}{1 + x^2}$. For small values of x, $\epsilon \sim \delta$, and as the value of x increases, the effect of ϵ upon δ diminishes.

Error Functions Subprograms

ERF/ERFC

Algorithm

1. If $0 \leqq x \leqq 1$, then compute the error function by the following approximation:

$$\text{erf}(x) \cong x(a_0 + a_1x^2 + a_2x^4 + \ldots + a_5x^{10}).$$

   The coefficients were obtained by the minimax approximation (in relative error) of $\text{erf}(x)/x$ as a function of x^2 over the range $0 \leqq x^2 \leqq 1$. The relative error of this approximation is less than $2^{-24.6}$. The value of the complemented error function is computed as $\text{erfc}(x) = 1 - \text{erf}(x)$.

2. If $1 < x < 2.040452$, then compute the complemented error function by the following approximation:

$$\text{erfc}\ (x) \cong b_0 + b_1z + b_2z^2 + \ldots + b_9z^9$$

   where $z = x - T_0$ and $T_0 \cong 1.709472$. The coefficients were obtained by the minimax approximation (in absolute error) of the function $f(z) = \text{erfc}(z + T_0)$ over the range $-0.709472 \leqq z \leqq 0.33098$. The absolute error of this approximation is less than $2^{-31.5}$. The limits of this range and the value of the origin T_0 were chosen to minimize the hexadecimal round-off errors. The value of the complemented error function within this range is between $\frac{1}{256}$ and 0.1573.

   The value of the error function is computed as $\text{erf}(x) = 1 - \text{erfc}(x)$.

3. If $2.040452 \leqq x < 13.306$, then compute the complemented error function by the following approximation:

$$\text{erfc}(x) \cong e^{-z} \cdot F/x \text{ where } z = x^2 \text{ and}$$

$$F = c_0 + \frac{c_1 + c_2z + c_3z^2}{d_1z + d_2z^2 + z^3}.$$

   The coefficients for F were obtained by transforming a minimax rational approximation (in absolute errors, over the range $13.306^{-2} \leqq w \leqq 2.040452^{-2}$) of the function $f(w) = \text{erfc}(x) \cdot x \cdot e^{x^2}$, $w = x^{-2}$, of the following form:

$$f(w) \cong \frac{a_0 + a_1w + a_2w^2 + a_3w^3}{b_0 + b_1w + w^2}.$$

   The absolute error of this approximation is less than $2^{-26.1}$. This computation uses the real exponential subprogram (EXP).

If $2.040452 \leqq x < 3.919206$, then the error function is computed as $\text{erf}(x) = 1 - \text{erfc}(x)$.
If $3.919206 \leqq x$, then the error function is $\cong 1$.

4. If $13.306 \leqq x$, then the error function is $\cong 1$, and the complemented error function is $\cong 0$ (underflow).
5. If $x < 0$, then reduce to a case involving a positive argument by the use of the following formulas:

$$\text{erf}(-x) = -\text{erf}(x), \text{ and } \text{erfc}(-x) = 2 - \text{erfc}(x).$$

Effect of an Argument Error

$E \sim e^{-x^2} \cdot \Delta$. For the error function, as the magnitude of the argument exceeds 1, the effect of an argument error upon the final accuracy diminishes rapidly. For small values of x, $\epsilon \sim \delta$. For the complemented error function, if the value of x is greater than 1, $\text{erfc}(x) \sim \frac{e^{-x^2}}{2x}$. Therefore, $\epsilon \sim 2x^2 \cdot \delta$. If the value of x is negative or less than 1, then $\epsilon \sim e^{-x^2} \cdot \Delta$.

DERF/DERFC

Algorithm

1. If $0 \leqq x < 1$, then compute the error function by the following approximation:

$$\text{erf}(x) \cong x(a_0 + a_1x^2 + a_2x^4 + \ldots + a_{11}x^{22}).$$

The coefficients were obtained by the minimax approximation (in relative error) of $\text{erf}(x)/x$ as a function of x^2 over the range $0 \leqq x^2 \leqq 1$. The relative error of this approximation is less than $2^{-56.9}$. The value of the complemented error function is computed as $\text{erfc}(x) = 1 - \text{erf}(x)$.

2. If $1 \leqq x < 2.040452$, then compute the complemented error function by the following approximation:

$$\text{erfc}(x) \cong b_0 + b_1z + b_2z^2 + \ldots b_{18}z^{18}$$

where $z = x - T_0$ and $T_0 \cong 1.709472$. The coefficients were obtained by the minimax approximation (in absolute error) of the function $f(z) = \text{erfc}(z + T_0)$ over the range $-0.709472 \leqq z \leqq 0.33098$. The absolute error of this approximation is less than $2^{-60.3}$. The limits of this range and the value of the origin T_0 were chosen to minimize the hexadecimal round-off errors. The value of the complemented error function within this range is between $\frac{1}{256}$ and 0.1573. The value of the error function is computed as $\text{erf}(x) = 1 - \text{erfc}(x)$.

3. If $2.040452 \leqq x < 13.306$, then compute the complemented error function by the following approximation:

$$\text{erfc}(x) \cong e^{-z} \cdot F/x \text{ where } z = x^2 \text{ and}$$

$$F = c_0 + \frac{d_1}{(z + c_1) +} \frac{d_2}{(z + c_2) +} \cdots \frac{d_6}{(z + c_6) +} \frac{d_7}{(z + c_7)}.$$

The coefficients for F were derived by transforming a minimax rational approximation (in absolute errors, over the range $13.306^{-2} \leqq w \leqq 2.040452^{-2}$) of the function $f(w) = \text{erfc}(x) \cdot x \cdot e^{x^2}$, $w = x^{-2}$, of the following form:

$$f(w) \cong \frac{a_0 + a_1w + a_2w^2 + \ldots + a_7w^7}{b_0 + b_1w + b_2w^2 + \ldots + b_6w^6 + w^7}.$$

The absolute error of this approximation is less than $2^{-57.9}$. This computation uses the real exponential subprogram (DEXP). If $2.040452 \leqq x < 6.092368$, then the error function is computed as $\text{erf}(x) = 1 - \text{erfc}(x)$.
If $6.092368 \leqq x$, then the error function is $\cong 1$.

4. If $13.306 \leqq x$, then the error function is $\cong 1$, and the complemented error function $\cong 0$ (underflow).
5. If $x < 0$, then reduce to a case involving a positive argument by the use of the following formulas:

$$\text{erf}(-x) = -\text{erf}(x), \text{ and } \text{erfc}(-x) = 2 - \text{erfc}(x).$$

Effect of an Argument Error

$E \sim e^{-x^2} \cdot \Delta$. For the error function, as the magnitude of the argument exceeds 1, the effect of an argument error upon the final accuracy diminishes rapidly. For small values of x, $\epsilon \sim \delta$. For the complemented error function, if the value of x is greater than 1, $\text{erfc}(x) \sim \dfrac{e^{-x^2}}{2x}$. Therefore, $\epsilon \sim 2x^2 \cdot \delta$. If the value of x is negative or less than 1, then $\epsilon \sim e^{-x^2} \cdot \Delta$.

Exponential Subprograms

EXP

Algorithm

1. If $x < -180.218$, then 0 is given as the answer via floating-point underflow.
2. Otherwise, divide x by $\log_e 2$ and write

$$y = \frac{x}{\log_e 2} = 4a - b - d$$

where a and b are integers, $0 \leqq b \leqq 3$ and $0 \leqq d < 1$.
3. Compute 2^{-d} by the following fractional approximation:

$$2^{-d} \cong 1 - \cfrac{2d}{0.034657359\, d^2 + d + 9.9545948 - \cfrac{617.97227}{d^2 + 87.417497}}$$

This formula can be obtained by transforming the Gaussian continued fraction

$$e^{-z} = 1 - \frac{z}{1+}\,\frac{z}{2-}\,\frac{z}{3+}\,\frac{z}{2-}\,\frac{z}{5+}\,\frac{z}{2-}\,\frac{z}{7+}\,\frac{z}{2}.$$

The maximum relative error of this approximation is 2^{-29}.
4. Multiply 2^{-d} by 2^{-b}.
5. Finally, add the hexadecimal exponent a to the characteristic of the answer.

Effect of an Argument Error

$\epsilon \sim \Delta$. If the magnitude of x is large, even the round-off error of the argument causes a substantial relative error in the answer because $\Delta = \delta \cdot x$.

DEXP

Algorithm

1. If $x < -180.2187$, then 0 is given as the answer via floating-point underflow.
2. Divide x by $\log_e 2$ and write

$$x = \left(4a - b - \frac{c}{16}\right) \cdot \log_e 2 - r$$

where a, b, and c are integers, $0 \leqq b \leqq 3$, $0 \leqq c \leqq 15$, and the remainder r is within the range $0 \leqq r < \dfrac{1}{16} \cdot \log_e 2$. This reduction is carried out in an extra precision to ensure accuracy. Then $e^x = 16^a \cdot 2^{-b} \cdot 2^{-c/16} \cdot e^{-r}$.

3. Compute e^{-r} by using a minimax polynomial approximation of degree 6 over the range $0 \leqq r < \frac{1}{16} \cdot \log_e 2$. In obtaining coefficients of this approximation, the minimax of relative errors was taken under the constraint that the constant term a_0 shall be exactly 1. The relative error is less than $2^{-56.87}$.
4. Multiply e^{-r} by $2^{-c/16}$. The 16 values of $2^{-c/16}$ for $0 \leqq c \leqq 15$ are included in the subprogram. Then halve the result b times.
5. Finally, add the hexdecimal exponent of a to the characteristic of the answer.

Effect of an Argument Error

$E \sim \Delta$. If the magnitude of x is large, even the round-off error of the argument causes a substantial relative error in the answer because $\Delta = \delta \cdot x$.

CEXP/CDEXP

Algorithm

The value of e^{x+iy} is computed as $e^x \cdot \cos(y) + i \cdot e^x \cdot \sin(y)$. The algorithms for both complex exponential subprograms are identical. Each subprogram uses the appropriate real exponential subprogram (EXP or DEXP) and the appropriate real sine/cosine subprogram (COS/SIN or DCOS/DSIN).

Effect of an Argument Error

The effect of an argument error depends upon the accuracy of the individual parts of the argument. If $e^{x+iy} = R \cdot e^{iH}$, then $H = y$ and $\epsilon(R) \sim \Delta(x)$.

Gamma and Log Gamma Subprograms

GAMMA/ALGAMA

Algorithm

1. If $0 < x \leqq 2^{-252}$, then compute log-gamma as $\log_e\Gamma(x) \cong -\log_e(x)$. This computation uses the real logarithm subprogram (ALOG).
2. If $2^{-252} < x < 8$, then compute log-gamma by taking the natural logarithm of the value obtained for gamma. The computation of gamma depends upon the range into which the argument falls.
3. If $2^{-252} < x < 1$, then use $\Gamma(x) = \dfrac{\Gamma(x+1)}{x}$ to reduce to the next case.
4. If $1 \leqq x \leqq 2$, then compute gamma by the minimax rational approximation (in absolute error) of the following form:
$$\Gamma(x) \cong c_0 + \frac{z\,[a_0 + a_1 z + a_2 z^2 + a_3 z^3]}{b_0 + b_1 z + b_2 z^2 + z^3}$$
where $z = x - 1.5$. The absolute error of this approximation is less than $2^{-25.9}$.
5. If $2 < x < 8$, then use $\Gamma(x) = (x-1)\,\Gamma(x-1)$ to reduce step by step to the preceding case.
6. If $8 \leqq x$, then compute log-gamma by the use of Stirling's formula:
$$\log_e\Gamma(x) \cong x(\log_e(x) - 1) - \tfrac{1}{2}\log_e(x) + \tfrac{1}{2}\log_e(2\pi) + G(x).$$
The modifier term $G(x)$ is computed as
$$G(x) \cong d_0 x^{-1} + d_1 x^{-2}.$$
These coefficients were obtained by a form of minimax approximation minimizing the ratio of the absolute error to the value of x. The absolute error is less than $x \cdot 2^{-26.2}$. Remembering the fact that $x < \log_e\Gamma(x)$ in this range, the contribution of this error to the relative error of the value for log-gamma is less

than $2^{-26.2}$. This computation uses the real logarithm subprogram (ALOG).

For gamma, compute $\Gamma(x) = e^y$, where y is the value obtained for log-gamma. This computation uses the real exponential subprogram (EXP).

Effect of an Argument Error

$\epsilon \sim \psi(x) \cdot \Delta$ for gamma, and $E \sim \psi(x) \cdot \Delta$ for log-gamma, where ψ is the digamma function.

If $\frac{1}{2} < x < 3$, then $-2 < \psi(x) < 1$. Therefore, $E \sim \Delta$ for log-gamma. However, because $x = 1$ and $x = 2$ are zeros of the log-gamma function, even a small δ can cause a substantial ϵ in this range.

If the value of x is large, then $\psi(x) \sim \log_e(x)$. Therefore, for gamma, $\epsilon \sim \delta x \cdot \log_e(x)$. In this case, even the round-off error of the argument contributes greatly to the relative error of the answer. For log-gamma with large values of x, $\epsilon \sim \delta$.

DGAMMA/DLGAMA

Algorithm

1. If $0 < x \leq 2^{-252}$, then compute log-gamma as $\log_e\Gamma(x) \cong -\log_e(x)$. This computation uses the real logarithm subprogram (DLOG).
2. If $2^{-252} < x < 8$, then compute log-gamma by taking the natural logarithm of the value obtained for gamma. The computation of gamma depends upon the range into which the argument falls.
3. If $2^{-252} < x < 1$, then use $\Gamma(x) = \dfrac{\Gamma(x+1)}{x}$ to reduce to the next case.
4. If $1 \leq x \leq 2$, then compute gamma by the minimax rational approximation (in absolute error) of the following form:

   $$\Gamma(x) \cong c_0 + \frac{z\,[a_0 + a_1 z + \ldots + a_6 z^6]}{b_0 + b_1 z + \ldots + b_6 z^6 + z^7}$$

   where $z = x - 1.5$. The absolute error of this approximation is less than $2^{-59.3}$.
5. If $2 < x < 8$, then use $\Gamma(x) = (x-1)\,\Gamma(x-1)$ to reduce to the preceding case.
6. If $8 \leq x$, then compute log-gamma by the use of Stirling's formula:

   $$\log_e\Gamma(x) \cong x(\log_e(x) - 1) - \tfrac{1}{2}\log_e(x) + \tfrac{1}{2}\log_e(2\pi) + G(x).$$

   The modifier term $G(x)$ is computed as

   $$G(x) \cong d_0 x^{-1} + d_1 x^{-3} + d_2 x^{-5} + d_3 x^{-7} + d_4 x^{-9}.$$

   These coefficients were obtained by a form of minimax approximation minimizing the ratio of the absolute error to the value of x. The absolute error is less than $x \cdot 2^{-56.1}$. Remembering the fact that $x < \log_e\Gamma(x)$ in this range, the contribution of this error to the relative error of the value for log-gamma is less than $2^{-56.1}$. This computation uses the real logarithm subprogram (DLOG). For gamma, compute $\Gamma(x) = e^y$, where y is the value obtained for log-gamma. This computation uses the real exponential subprogram (DEXP).

Effect of an Argument Error

$\epsilon \sim \psi(x) \cdot \Delta$ for gamma, and $E \sim \psi(x) \cdot \Delta$ for log-gamma, where ψ is the digamma function.

If $\frac{1}{2} < x < 3$, then $-2 < \psi(x) < 1$. Therefore, $E \sim \Delta$ for log-gamma. However, because $x = 1$ and $x = 2$ are zeros of the log-gamma function, even a small δ can cause a substantial ϵ in this range.

If the value of x is large, then $\psi(x) \sim \log_e(x)$. Therefore, for gamma, $\epsilon \sim \delta \cdot x \cdot \log_e(x)$. In this case, even the round-off error of the argument contributes greatly to the relative error of the answer. For log-gamma with large values of x, $\epsilon \sim \delta$.

Hyperbolic Sine and Cosine Subprograms

SINH/COSH

Algorithm

1. If $|x| < 1.0$, then compute $\sinh(x)$ as:

   $$\sinh(x) \cong x + c_1x^3 + c_2x^5 + c_3x^7.$$

   The coefficient c_i were obtained by the minimax approximation (in relative error) of $\frac{\sinh(x)}{x}$ as the function of x^2. The maximum relative error of this approximation is $2^{-25.6}$.
2. If $x \geqq 1.0$, then $\sinh(x)$ is computed as:

   $$\sinh(x) = (1 + \delta)\ [e^{x+\log_e v} - v^2/e^{x+\log_e v}].$$

   Here, $1 + \delta = \frac{1}{2v}$, so that this expression is theoretically equivalent to $[e^x - e^{-x}]/2$. The value of v (and consequently those of $\log_e v$ and δ) was so chosen as to satisfy the following conditions:
   a) v is slightly less than ½, so that $\delta > 0$ and small.
   b) $\log_e v$ is an exact multiple of 2^{-16}.
   The condition *b*) insures that the addition $x + \log_e v$ is carried out exactly. This maneuver was designed to reduce the round-off errors and also to enlarge the limits of acceptable arguments. This computation uses the real exponential subprogram (EXP).
3. If $x \leqq -1.0$, use $\sinh(x) = -\sinh(|x|)$ to reduce to case 2 above.
4. If cosh (x) is desired, then for all valid values of arguments use the identity: $\cosh(x) = (1 + \delta)\ [e^{x+\log_e v} + v^2/e^{x+\log_e v}]$. Here the notation and the consideration are identical to case 2 above. This computation uses the real exponential subprogram (EXP).

Effect of an Argument Error

For the hyperbolic sine, $E \sim \Delta \cdot \cosh(x)$ and $\epsilon \sim \Delta \cdot \coth(x)$.
For the hyperbolic cosine, $E \sim \Delta \cdot \sinh(x)$ and $\epsilon \sim \delta \cdot \tanh(x)$.

Specifically, for the cosine, $\epsilon \sim \Delta$ over the entire range; for the sine, $\epsilon \sim \delta$ for small values of x.

DSINH/DCOSH

Algorithm

1. If $|x| < 0.881374$, then compute $\sinh(x)$ as:

   $$\sinh(x) \cong c_0x + c_1x^3 + c_2x^5 + \ldots + c_6x^{13}.$$

   The coefficients c_i were obtained by the minimax approximation (in relative error) of $\frac{\sinh(x)}{x}$ as the function of x^2. Minimax was taken under the constraint that $c_0 = 1$ exactly. The maximum relative error of this approximation is $2^{-55.7}$.
2. If $x \geqq 0.881374$, then $\sinh(x)$ is computed as:

   $$\sinh(x) = (1 + \delta)\ [e^{x+\log_e v} - v^2/e^{x+\log_e v}].$$

Here, $1 + \delta = \frac{1}{2v}$, so that this expression is theoretically equivalent to $[e^x - e^{-x}]/2$. The value of v (and consequently those of $\log_e v$ and δ) was so chosen as to satisfy the following conditions:

a) v is slightly less than ½, so that $\delta > 0$ and small.

b) $\log_e v$ is an exact multiple of 2^{-16}.

The condition *b*) insures that the addition $x + \log_e v$ is carried out exactly. This maneuver was designed to reduce the round-off errors and also to enlarge the limits of acceptable arguments. This computation uses the real exponential subprogram (DEXP).

3. If $x \leqq -0.881374$, then use $\sinh(x) = -\sinh(|x|)$ to reduce to case 2 above.
4. If $\cosh(x)$ is desired, then, for all valid arguments use the identity: $\cosh(x) = (1 + \delta)\,[e^{x+\log_e v} + v^2/e^{x+\log_e v}]$. Here the notation and the consideration are identical to case 2 above. This computation uses the real exponential subprogram (DEXP).

Effect of an Argument Error

For the hyperbolic sine, $E \sim \Delta \cdot \cosh(x)$ and $\epsilon \sim \Delta \cdot \coth(x)$.
For the hyperbolic cosine, $E \sim \Delta \cdot \sinh(x)$ and $\epsilon \sim \Delta \cdot \tanh(x)$.

Specifically, for the cosine, $\epsilon \sim \Delta$ over the entire range; for the sine, $\epsilon \sim \delta$ for the small values of x.

Hyperbolic Tangent Subprograms

TANH

Algorithm

1. If $|x| \leqq 2^{-12}$, then $\tanh(x) \cong x$.
2. If $2^{-12} < |x| \leqq 0.7$, use the following fractional approximation:

$$\frac{\tanh(x)}{x} \cong 1 - x^2\left[0.0037828 + \frac{0.8145651}{x^2 + 2.471749}\right].$$

The coefficients of this approximation were obtained by taking the minimax of relative error, over the range $x^2 < 0.49$, of approximations of this form under the constraint that the first term shall be exactly 1.0. The maximum relative error of this approximation is $2^{-26.4}$.

3. If $0.7 < x < 9.011$, then use the identity $\tanh(x) = 1 - \frac{2}{(e^x)^2 + 1}$.

The computation for this case uses the real exponential subprogram (EXP).

4. If $x \geqq 9.011$, than $\tanh(x) \cong 1$.
5. If $x < -0.7$, then use the identity $\tanh(x) = -\tanh(-x)$.

Effect of an Argument Error

$E \sim (1 - \tanh^2 x)\,\Delta$, and $\epsilon \sim \frac{2\Delta}{\sinh\,(2x)}$. For small values of x, $\epsilon \sim \delta$, and as the value of x increases, the effect of δ upon ϵ diminishes.

DTANH

Algorithm

1. If $|x| \leqq 2^{-28}$, then $\tanh(x) \cong x$.
2. If $2^{-28} < |x| < 0.54931$, use the following fractional approximation:

$$\frac{\tanh(x)}{x} \cong c_0 + \frac{d_1 x^2}{x^2 + c_1 +}\;\frac{d_2}{x^2 + c_2 +}\;\frac{d_3}{x^2 + c_3}.$$

This approximation was obtained by rewriting a minimax approximation of the following form:

$$\frac{\tanh(x)}{x} \cong c_0 + x^2 \cdot \frac{a_0 + a_1x^2 + a_2x^4}{b_0 + b_1x^2 + b_2x^4 + x^6}.$$

Here the minimax of relative error, over the range $x^2 \leqq 0.30174$, was taken under the constraint that c_0 shall be exactly 1.0. The maximum relative error of the above is 2^{-63}.

3. If $0.54931 \leqq x < 20.101$, then use the identity $\tanh(x) = 1 - \frac{2}{e^{2x} + 1}$.

   This computation uses the double precision exponential subprogram (DEXP).
4. If $x \geqq 20.101$, then $\tanh(x) \cong 1$.
5. If $x \leqq -0.54931$, then use the identity $\tanh(x) = -\tanh(-x)$.

Effect of an Argument Error

$E \sim (1 - \tanh^2 x)\,\Delta$, and $\epsilon \sim \frac{2\Delta}{\sinh(2x)}$. For small values of x, $\epsilon \sim \delta$. As the value of x increases, the effect of δ upon ϵ diminishes.

Logarithmic Subprograms (Common and Natural)

ALOG/ALOG10

Algorithm

1. Write $x = 16^p \cdot 2^{-q} \cdot m$ where p is the exponent, q is an integer, $0 \leqq q \leqq 3$, and m is within the range, $\frac{1}{2} \leqq m < 1$.
2. Define two constants, a and b (where a = base point and $2^{-b} = a$), as follows:

   If $\frac{1}{2} \leqq m < \frac{1}{\sqrt{2}}$, then $a = \frac{1}{2}$ and $b = 1$.

   If $\frac{1}{\sqrt{2}} \leqq m < 1$, then $a = 1$ and $b = 0$.
3. Write $z = \frac{m - a}{m + a}$. Then, $m = a \cdot \frac{1 + z}{1 - z}$ and $|z| < 0.1716$.
4. Now, $x = 2^{4p-q-b} \cdot \frac{1 + z}{1 - z}$, and $\log_e(x) = (4p - q - b) \log_e 2 + \log_e\left(\frac{1 + z}{1 - z}\right)$.
5. To obtain $\log_e\left(\frac{1 + z}{1 - z}\right)$, first compute $w = 2z = \frac{m - a}{0.5m + 0.5a}$ (which is represented in our system with slightly more significant digits than z itself), and apply an approximation of the following form:

$$\log_e\left(\frac{1 + z}{1 - z}\right) \cong w\left[c_0 + \frac{c_1w^2}{c_2 - w^2}\right].$$

   These coefficients were obtained by the minimax rational approximation of $\frac{1}{2z}\log_e\left(\frac{1 + z}{1 - z}\right)$ over the range $z^2 \,\epsilon\, (0, 0.02944)$ under the constraint that c_0 shall be exactly 1.0. The maximum relative error of this approximation is less than $2^{-25.33}$.
6. If the common logarithm is desired, then $\log_{10}x = \log_{10}e \cdot \log_e x$.

Effect of an Argument Error

$E \sim \delta$. Specifically, if δ is the round-off error of the argument, e.g., $\delta \sim 6 \cdot 10^{-8}$, then $E \sim 6 \cdot 10^{-8}$. Therefore, if the argument is close to 1, the relative error can be very large because the value of the function is very small.

DLOG/DLOG10

Algorithm

1. Write $x = 16^p \cdot 2^{-q} \cdot m$ where p is the exponent, q is an integer, $0 \leqq q \leqq 3$, and m is within the range $\frac{1}{2} \leqq m < 1$.
2. Define two constants, a and b (where a = base point and $2^{-b} = a$), as follows:

   If $\frac{1}{2} \leqq m < \frac{1}{\sqrt{2}}$, then $a = \frac{1}{2}$ and $b = 1$.

   If $\frac{1}{\sqrt{2}} \leqq m < 1$, then $a = 1$ and $b = 0$.
3. Write $z = \frac{m - a}{m + a}$. Then, $m = a \cdot \frac{1 + z}{1 - z}$ and $|z| < 0.1716$.
4. Now, $x = 2^{4p-q-b} \cdot \frac{1+z}{1-z}$, and $\log_e x = (4p - q - b) \log_e 2 + \log_e \left(\frac{1+z}{1-z}\right)$.
5. To obtain $\log_e \left(\frac{1+z}{1-z}\right)$, first compute $w = 2z = \frac{m - a}{0.5m + 0.5a}$ (which is represented in our system with slightly more significant digits than z itself), and apply an approximation of the following form:

$$\log_e \left(\frac{1+z}{1-z}\right) \cong w \left[c_0 + c_1 w^2 \left(w^2 + c_2 + \cfrac{c_3}{w^2 + c_4 + \cfrac{c_5}{w^2 + c_6}} \right) \right].$$

   These coefficients were obtained by the minimax rational approximation of $\frac{1}{2z} \log_e \left(\frac{1+z}{1-z}\right)$ over the range $z^2 \epsilon\ (0, 0.02944)$ under the constraint that c_0 shall be exactly 1.0. The maximum relative error of this approximation is less than $2^{-60.55}$.
6. If the common logarithm is desired, then $\log_{10} x = \log_{10} e \cdot \log_e x$.

Effect of an Argument Error

$E \sim \delta$. Therefore, if the value of the argument is close to 1, the relative error can be very large because the value of the function is very small.

CLOG/CDLOG

Algorithm

1. Write $\log_e (x + iy) = a + ib$.
2. Then, $a = \log_e |x + iy|$ and b = the principal value of arctan (y, x).
3. $\log_e |x + iy|$ is computed as follows:

   Let $v_1 = \max(|x|, |y|)$, and $v_2 = \min(|x|, |y|)$.

   Let t be the exponent of v_1, i.e., $v_1 = m \cdot 16^t$, $\frac{1}{16} \leqq m < 1$.

   Finally, let $t_1 = \begin{cases} t \text{ if } t \leqq 0 \\ t - 1 \text{ if } t > 0 \end{cases}$,

   and $s = 16^{t_1}$.

   Then, $\log_e |x + iy| = 4t_1 \cdot \log_e(2) + \frac{1}{2} \log_e \left[\left(\frac{v_1}{s}\right)^2 + \left(\frac{v_2}{s}\right)^2 \right]$.

   Computation of v_1/s and v_2/s are carried out by manipulation of the characteristics of v_1 *and* v_2. In particular, if $v_2/s <$ 1, it is taken to be 0. The algorithms for both complex logarithm subprograms are identical. Each subprogram uses the appropriate real natural logarithm subprogram (ALOG or DLOG) and the appropriate arctangent subprogram (ATAN2 or DATAN2).

Effect of an Argument Error

The effect of an argument error depends upon the accuracy of the individual parts of the argument. If $x + iy = r \cdot e^{ih}$ and $\log_e (x + iy) = a + ib$, then $h = b$ and $E(a) = \delta(r)$.

Sine and Cosine Subprograms

SIN/COS

Algorithm

1. Define $z = \frac{4}{\pi} \cdot |x|$ and separate z into its integer part (q) and its fraction part (r). Then $z = q + r$, and $|x| = \left(\frac{\pi}{4} \cdot q\right) + \left(\frac{\pi}{4} \cdot r\right)$.
2. If the cosine is desired, add 2 to q. If the sine is desired and if x is negative, add 4 to q. This adjustment of q reduces the general case to the computation of $\sin(x)$ for $x \geqq 0$ because

   $$\cos(\pm x) = \sin\left(\frac{\pi}{2} + x\right), \text{ and}$$
   $$\sin(-x) = \sin(\pi + x).$$
3. Let $q_0 \equiv q \bmod 8$.

   Then, for $q_0 = 0$, $\sin(x) = \sin\left(\frac{\pi}{4} \cdot r\right)$,

   $q_0 = 1$, $\sin(x) = \cos\left(\frac{\pi}{4}(1 - r)\right)$,

   $q_0 = 2$, $\sin(x) = \cos\left(\frac{\pi}{4} \cdot r\right)$,

   $q_0 = 3$, $\sin(x) = \sin\left(\frac{\pi}{4}(1 - r)\right)$,

   $q_0 = 4$, $\sin(x) = -\sin\left(\frac{\pi}{4} \cdot r\right)$,

   $q_0 = 5$, $\sin(x) = -\cos\left(\frac{\pi}{4}(1 - r)\right)$,

   $q_0 = 6$, $\sin(x) = -\cos\left(\frac{\pi}{4} \cdot r\right)$,

   $q_0 = 7$, $\sin(x) = -\sin\left(\frac{\pi}{4}(1 - r)\right)$.

   These formulas reduce each case to the computation of either $\sin\left(\frac{\pi}{4} \cdot r_1\right)$ or $\cos\left(\frac{\pi}{4} \cdot r_1\right)$ where r_1 is either r or $(1 - r)$ and is within the range, $0 \leqq r_1 \leqq 1$.
4. If $\sin\left(\frac{\pi}{4} \cdot r_1\right)$ is needed, it is computed by a polynomial of the following form:

   $$\sin\left(\frac{\pi}{4} \cdot r_1\right) \cong r_1 (a_0 + a_1 r_1^2 + a_2 r_1^4 + a_3 r_1^6).$$

   The coefficients were obtained by the interpolation at the roots of the Chebyshev polynomial of degree 4. The relative error is less than $2^{-28.1}$ for the range.

5. If $\cos\left(\frac{\pi}{4} \cdot r_1\right)$ is needed, it is computed by a polynomial of the following form:

$$\cos\left(\frac{\pi}{4} \cdot r_1\right) \cong 1 + b_1 r_1^2 + b_2 r_1^4 + b_3 r_1^6.$$

Coefficients were obtained by a variation of the minimax approximation which provides a partial rounding for the short precision computation. The absolute error of this approximation is less than $2^{-24.57}$.

Effect of an Argument Error

$E \sim \Delta$. As the value of x increases, Δ increases. Because the function value diminishes periodically, no consistent relative error control can be maintained outside the principal range, $-\frac{\pi}{2} \leqq x \leqq +\frac{\pi}{2}$.

DSIN/DCOS

Algorithm

1. Divide $|x|$ by $\frac{\pi}{4}$ and separate the quotient (z) into its integer part (q) and its fraction part (r). Then, $z = |x| \cdot \frac{4}{\pi} = q + r$, where q is an integer and r is within the range, $0 \leqq r < 1$.
2. If the cosine is desired, add 2 to q. If the sine is desired and if x is negative, add 4 to q. This adjustment of q reduces the general case to the computation of $\sin(x)$ for $x \geqq 0$, because

$$\cos(\pm x) = \sin\left(|x| + \frac{\pi}{2}\right), \text{ and}$$
$$\sin(-x) = \sin(|x| + \pi).$$

3. Let $q_0 \equiv q \bmod 8$.

Then, for $q_0 = 0$, $\sin(x) = \sin\left(\frac{\pi}{4} \cdot r\right)$,

$$q_0 = 1, \sin(x) = \cos\left(\frac{\pi}{4}(1 - r)\right),$$
$$q_0 = 2, \sin(x) = \cos\left(\frac{\pi}{4} \cdot r\right),$$
$$q_0 = 3, \sin(x) = \sin\left(\frac{\pi}{4}(1 - r)\right),$$
$$q_0 = 4, \sin(x) = -\sin\left(\frac{\pi}{4} \cdot r\right),$$
$$q_0 = 5, \sin(x) = -\cos\left(\frac{\pi}{4}(1 - r)\right),$$
$$q_0 = 6, \sin(x) = -\cos\left(\frac{\pi}{4} \cdot r\right),$$
$$q_0 = 7, \sin(x) = -\sin\left(\frac{\pi}{4}(1 - r)\right).$$

These formulas reduce each case to the computation of either $\sin\left(\frac{\pi}{4} \cdot r_1\right)$ or $\cos\left(\frac{\pi}{4} \cdot r_1\right)$; where r_1 is either r or $(1 - r)$, and is within the range, $0 \leqq r_1 \leqq 1$.

4. Finally, either $\sin\left(\frac{\pi}{4} \cdot r_1\right)$ or $\cos\left(\frac{\pi}{4} \cdot r_1\right)$ is computed, using the polynomial interpolations of degree 6 in r_1^2 for the sine, and of degree 7 in r_1^2 for the cosine. In either case, the interpolation points were the roots of the Chebyshev polynomial of one higher degree. The maximum relative error of the sine polynomial is 2^{-58} and that of the cosine polynomial is $2^{-64.3}$.

Effect of an Argument Error

$E \sim \Delta$. As the value of the argument increases, Δ increases. Because the function value diminishes periodically, no consistent relative error control can be maintained outside of the principal range, $-\frac{\pi}{2} \leqq x \leqq +\frac{\pi}{2}$.

CSIN/CCOS

Algorithm

1. If the sine is desired, then
$$\sin(x + iy) = \sin(x) \cdot \cosh(y) + i \cdot \cos(x) \cdot \sinh(y).$$
If the cosine is desired, then
$$\cos(x + iy) = \cos(x) \cdot \cosh(y) - i \cdot \sin(x) \cdot \sinh(y).$$
2. The value of $\sinh(x)$ is computed within the subprogram as follows. Assume $x \geqq 0$ for this, since $\sinh(-x) = -\sinh(x)$.
3. If $x \geqq 0.346574$, then use $\sinh(x) = \frac{1}{2}\left(e^x - \frac{1}{e^x}\right)$.
4. If $0 \leqq x < 0.346574$, then compute $\sinh(x)$ by use of a polynomial:
$$\frac{\sinh(x)}{x} \cong a_0 + a_1x^2 + a_2x^4.$$
The coefficients were obtained by the minimax approximation (in relative error) of $\sinh(x)/x$ over the range $0 \leqq x^2 \leqq 0.12011$ under the constraint that a_0 shall be exactly 1.0. The relative error of this approximation is less than $2^{-26.18}$.
5. The value of $\cosh(x)$ is computed as $\cosh(x) = \sinh|x| + \frac{1}{e^{|x|}}$.
This computation uses the real expoential subprogram (EXP) and the real sine/cosine subprogram (SIN/COS).

Effect of an Argument Error

To understand the effect of an argument error upon the accuracy of the answer, the programmer must understand the effect of an argument in the SIN/COS, EXP, and SINH/COSH subprograms.

CDSIN/CDCOS

Algorithm

1. If the sine is desired, then
$$\sin(x+iy) = \sin(x) \cdot \cosh(y) + i \cdot \cos(x) \cdot \sinh(y).$$
If the cosine is desired, then
$$\cos(x + iy) = \cos(x) \cdot \cosh(y) - i \cdot \sin(x) \cdot \sinh(y).$$
2. The value of $\sinh(x)$ is computed within the subprogram as follows. Assume $x \geqq 0$ for this, since $\sinh(-x) = -\sinh(x)$.
3. If $x \geqq 0.481212$, then use $\sinh(x) = \frac{1}{2}\left(e^x - \frac{1}{e^x}\right)$.

4. If $0 \leqq x < 0.481212$, then compute $\sinh(x)$ by use of a polynomial:

$$\frac{\sinh(x)}{x} \cong a_0 + a_1x^2 + a_2x^4 + a_3x^6 + a_4x^8 + a_5x^{10}.$$

The coefficients were obtained by the minimax approximation (in relative error) of $\sinh(x)/x$ over the range $0 \leqq x^2 \leqq 0.23156$ under the constraint that a_0 shall be exactly 1.0. The relative error of this approximation is less than $2^{-56.07}$.

5. The value of $\cosh(x)$ is computed as $\cosh(x) = \sinh|x| + \frac{1}{e^{|x|}}$.

This computation uses the real exponential subprogram (DEXP) and the real sine/cosine subprogram (DSIN/DCOS).

Effect of an Argument Error

To understand the effect of an argument error upon the accuracy of the answer, the programmer must understand the effect of an argument error in the DSIN/DCOS, DEXP, and DSINH/DCOSH subprograms.

Square Root Subprograms

SQRT

Algorithm

1. If $x = 0$, then the answer is 0.
2. Write $x = 16^{2p-q} \cdot m$, where $2p - q$ is the exponent and q equals either 0 or 1; m is the mantissa and is within the range $\frac{1}{16} \leqq m < 1$.
3. Then, $\sqrt{x} = 16^p \cdot 4^{-q}\sqrt{m}$.
4. For the first approximation of $\sqrt{x}$, compute the following:

$$y_0 = 16^p \cdot 4^{-q} \cdot \left(1.681595 - \frac{1.288973}{0.8408065 + m}\right).$$

This approximation attains the minimax relative error for hyperbolic fits of $\sqrt{x}$. The maximum relative error is $2^{-5.748}$.

5. Apply the Newton-Raphson iteration

$$y_{n+1} = \tfrac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

twice. The second iteration is performed as

$$y_2 = \tfrac{1}{2}\left(y_1 - \frac{x}{y_1}\right) + \frac{x}{y_1},$$

with a partial rounding. The maximum relative error of y_2 is theoretically $2^{-25.9}$.

Effect of an Argument Error

$$\epsilon \sim \frac{1}{2}\,\delta.$$

DSQRT

Algorithm

1. If $x = 0$, then the answer is 0.
2. Write $x = 16^{2p-q} \cdot m$, where $2p - q$ is the exponent and q equals either 0 or 1; m is the mantissa and is within the range $\frac{1}{16} \leqq m < 1$.

3. Then, $\sqrt{x} = 16^p \cdot 4^{-q} \sqrt{m}$.
4. For the first approximation of $\sqrt{x}$, compute the following:

   $$y_0 = 16^p \cdot 4^{1-q} \cdot 0.2202\ (m + 0.2587).$$

   The extrema of relative errors of this approximation for $q = 0$ are $2^{-3.202}$ at $m = 1$, $2^{-3.265}$ at $m = 0.2587$, and $2^{-2.925}$ at $m = \frac{1}{16}$. This approximation, rather than the minimax approximation, was chosen so that the quantity $\frac{x}{y_3} - y_3$ below becomes less than 16^{p-8} in magnitude. This arrangement allows us to substitute short form counterparts for some of the long form instructions in the final iteration.
5. Apply the Newton Raphson iteration

   $$y_{n+1} = \tfrac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

   four times to y_0, twice in the short form and twice in the long form. The final step is performed as

   $$y_4 = y_3 + \tfrac{1}{2}\left(\frac{x}{y_3} - y_3\right)$$

   with an appropriate truncation maneuver to obtain a virtual rounding. The maximum relative error of the final result is theoretically $2^{-63.23}$.

Effect of an Argument Error

$$\epsilon \sim \frac{1}{2}\delta$$

CSQRT/CDSQRT

Algorithm

1. Write $\sqrt{x + iy} = a + ib$.
2. Compute the value $z = \sqrt{\dfrac{|x| + |x + iy|}{2}}$ as $k \cdot \sqrt{w_1 + w_2}$ where k, w_1 and w_2 are defined in 3, or 4, below. In any case let $v_1 = \max(|x|, |y|)$ and
   $v_2 = \min(|x|, |y|)$.
3. In the special case when either $v_2 = 0$ or $v_1 > v_2$, let $w_1 = v_2$ and $w_2 = v_1$ so that $w_1 + w_2$ is effectively equal to v_1.
   Also let $k = 1$ if $v_1 = |x|$ and
   $k = 1/\sqrt{2}$ if $v_1 = |y|$.
4. In the general case, compute $F = \sqrt{\tfrac{1}{4} + \tfrac{1}{4}\left(\dfrac{v_2}{v_1}\right)^2}$.
   If $|x|$ is near the underflow threshold, then take
   $w_1 = |x|$, $w_2 = v_1 \cdot 2F$, and $k = 1/\sqrt{2}$.
   If $v_1 \cdot F$ is near the overflow threshold, then take
   $w_1 = |x|/4$, $w_2 = v_1 \cdot F/2$, and $k = \sqrt{2}$.
   In all other cases, take $w_1 = |x|/2$, $w_2 = v_1 \cdot F$, and $k = 1$.
5. If $z = 0$, then $a = 0$ and $b = 0$.
   If $z \neq 0$ and $x \geq 0$, then $a = z$, and
   $$b = \frac{y}{2z}.$$
   If $z \neq 0$ and $x < 0$, then $a = \left|\dfrac{y}{2z}\right|$, and
   $$b = (\text{sign } y) \cdot z.$$

The algorithms for both complex square root subprograms are identical. Each subprogram uses the appropriate real square root subprogram (SQRT or DSQRT).

Effect of an Argument Error

The effect of an argument error depends upon the accuracy of the individual parts of the argument. If $x + iy = r \cdot e^{ih}$ and $\sqrt{x + iy} = R \cdot e^{iH}$,

then $\epsilon(R) \sim \frac{1}{2}\,\delta(r)$, and $\epsilon(H) \sim \delta(h)$.

Tangent and Cotangent Subprograms

TAN/COTAN

Algorithm

1. Divide $|x|$ by $\frac{\pi}{4}$ and separate the result into integer part (q) and the fraction part (r). Then $|x| = \frac{\pi}{4}(q + r)$.
2. Obtain the reduced argument (w) as follows:
   if q is even, then $w = r$
   if q is odd, then $w = 1 - r$.
   The range of the reduced argument is $0 \leqq w \leqq 1$.
3. Let $q_0 \equiv q \bmod 4$.
   Then for $q_0 = 0$, $\tan|x| = \tan\left(\frac{\pi}{4} \cdot w\right)$ and $\cot|x| = \cot\left(\frac{\pi}{4} \cdot w\right)$,
   $q_0 = 1$, $\tan|x| = \cot\left(\frac{\pi}{4} \cdot w\right)$ and $\cot|x| = \tan\left(\frac{\pi}{4} \cdot w\right)$,
   $q_0 = 2$, $\tan|x| = -\cot\left(\frac{\pi}{4} \cdot w\right)$ and $\cot|x| = -\tan\left(\frac{\pi}{4} \cdot w\right)$,
   $q_0 = 3$, $\tan|x| = -\tan\left(\frac{\pi}{4} \cdot w\right)$ and $\cot|x| = -\cot\left(\frac{\pi}{4} \cdot w\right)$.
4. The value of $\tan\left(\frac{\pi}{4} \cdot w\right)$ and $\cot\left(\frac{\pi}{4} \cdot w\right)$ are computed as the ratio of two polynomials:
   $$\tan\left(\frac{\pi}{4} \cdot w\right) \cong \frac{w \cdot P(u)}{Q(u)}, \quad \cot\left(\frac{\pi}{4} \cdot w\right) \cong \frac{Q(u)}{w \cdot P(u)}$$
   where $u = \frac{1}{2}w^2$ and
   $P(u) = -3.460901 + u$
   $Q(u) = -10.772754 + 5.703366 \cdot u - 0.159321 \cdot u^2$.
   These coefficients were obtained by the minimax rational approximation (in relative error) of the indicated form. The maximum relative error of this approximation is 2^{-26}. Choice of u rather than w^2 as the variable for P and Q is to improve the round-off quality of the coefficients.
5. If $x < 0$, then $\tan(x) = -\tan|x|$, and $\cot(x) = -\cot|x|$.
6. This program is provided with two kinds of error controls. One is for arguments whose magnitude is greater than $2^{18} \cdot \pi$. The other is for arguments which are very close to a singularity of the function. In either case, the precision of the argument is deemed insufficient for obtaining a reliable result. More specifically, the second control screens out the following arguments:
   a) $|x| \leqq 16^{-63}$ for COTAN (the result would overflow).
   b) x is such that one can find a singularity within eight units of the last digit

value of the floating-point representation of the sum $q + r$. Singularities are cases when the cotangent ratio is to be taken and $w = 0$.

The test threshold of this control can be dynamically modified by assembler code programs.

Effect of an Argument Error

$E \sim \dfrac{\Delta}{\cos^2(x)}$, and $\epsilon \sim \dfrac{2}{\sin(2x)}$ for $\tan(x)$. Therefore, near the singularities $x = \left(k + \dfrac{1}{2}\right)\pi$, where k is an integer, no error control can be maintained. This is also true for $\text{cotan}(x)$ for x near $k\pi$, where k is an integer.

DTAN/DCOTAN

Algorithm

1. Divide $|x|$ by $\dfrac{\pi}{4}$ and separate the result into integer part (q) and the fraction part (r). Then $|x| = \dfrac{\pi}{4}(q + r)$.
2. Obtain the reduced argument (w) as follows:
   if q is even, then $w = r$
   if q is odd, then $w = 1 - r$.

   The range of the reduced argument is $0 \leqq w \leqq 1$.
3. Let $q_0 \equiv q \bmod 4$.

   Then for $q_0 = 0$, $\tan|x| = \tan\left(\dfrac{\pi}{4} \cdot w\right)$ and $\cot|x| = \cot\left(\dfrac{\pi}{4} \cdot w\right)$,

   $q_0 = 1$, $\tan|x| = \cot\left(\dfrac{\pi}{4} \cdot w\right)$ and $\cot|x| = \tan\left(\dfrac{\pi}{4} \cdot w\right)$,

   $q_0 = 2$, $\tan|x| = -\cot\left(\dfrac{\pi}{4} \cdot w\right)$ and $\cot|x| = -\tan\left(\dfrac{\pi}{4} \cdot w\right)$,

   $q_0 = 3$, $\tan|x| = -\tan\left(\dfrac{\pi}{4} \cdot w\right)$ and $\cot|x| = -\cot\left(\dfrac{\pi}{4} \cdot w\right)$.
4. The value of $\tan\left(\dfrac{\pi}{4} \cdot w\right)$ and $\cot\left(\dfrac{\pi}{4} \cdot w\right)$ are computed as the ratio of two polynomials:

   $$\tan\left(\frac{\pi}{4} \cdot w\right) \cong \frac{w \cdot P(w^2)}{Q(w^2)}, \text{ and } \cot\left(\frac{\pi}{4} \cdot w\right) \cong \frac{Q(w^2)}{w \cdot P(w^2)}.$$

   where both P and Q are polynomials of degree 3 in w^2. The coefficients of P and Q were obtained by the minimax rational approximation (in relative error) of $\dfrac{1}{w}\tan\left(\dfrac{\pi}{4}w\right)$ of the indicated form. The maximum relative error of this approximation is $2^{-55.6}$.
5. If $x < 0$, then $\tan(x) = -\tan|x|$, and $\cot(x) = -\cot|x|$.
6. This program is provided with two kinds of error controls. One is for arguments whose magnitude is greater than $2^{50} \cdot \pi$. The other is for arguments which are very close to a singularity of the function. In either case, the precision of the argument is deemed insufficient for obtaining a reliable result. More specifically, the second control screens out the following arguments:
   a) $|x| \leqq 16^{-63}$ for COTAN (the result would overflow).
   b) x is such that one can find a singularity within eight units of the last digit value of the floating-point representation of the sum $q + r$. Singularities are cases when the cotangent ratio is to be taken and $w = 0$.

The test threshold of this control can be dynamically modified by assembler code programs.

Effect of an Argument Error

$E \sim \frac{\Delta}{\cos^2(x)}$, and $\epsilon \sim \frac{2}{\sin(2x)}$ for $\tan(x)$. Therefore, near the singularities of $x = \left(k + \frac{1}{2}\right)\pi$, where k is an integer, no error control can be maintained. This is also true for $\cotan(x)$ for values of x near $k\pi$, where k is an integer.

Implicitly Called Subprograms

The entry point names of the following implicitly called subprograms are generated by the compiler.

Complex Multiply and Divide Subprograms

CDVD#/CMPY# (Divide/Multiply for COMPLEX*8 Arguments)

CDDVD#/CDMPY# (Divide/Multiply for COMPLEX*16 Arguments)

Algorithm

Multiply: $(A + Bi)(C + Di) = (AC - BD) + (AD + BC)i$

Divide: $(A + Bi)/(C + Di)$

1. If $|C| \leq |D|$, set
   $A = B, B = -A, C = D, D = -C$, since
   $\frac{A + Bi}{C + Di} = \frac{B = Ai}{D - Ci}$ before step 2.
2. Set $A' = \frac{A}{C}, B' = \frac{B}{C}, D' = \frac{D}{C}$;
   then compute
   $$\frac{A + Bi}{C + Di} = \frac{A' + B'i}{1 + D'i} = \frac{A' + B'D}{1 + D'D'} + \frac{B' - A'D'}{1 + D'D'} i.$$

Error Conditions

Partial underflows can occur in preparing the answer.

Complex Exponentiation Subprograms

FCDXI# (COMPLEX*16 Arguments)

FCXPI# (COMPLEX*8 Arguments)

Algorithm

The value of $y_1 + y_2 i = (z_1 + z_2 i)^j$ is computed as follows.

Let $|j| = \sum_{k=0}^{K} r_k \cdot 2^k$ where $r_k = 0$ or 1 for $k = 0, 1, \ldots, K$.

Then $z^{|j|} = \pi_{r_k \neq 0}\, z^{2^k}$, and the factors z^{2^k} can be obtained by successive squaring.

More specifically:

1. Initially: $k = 0$, $n^{(0)} = |j|$, $y_1^{(0)} + y_2^{(0)} i = 1 + 0i$,
   $z_1^{(0)} + z_2^{(0)} i = z_1 + z_2 i$.

2. Raise the index k by 1, and let $n^{(k-1)} = 2q + r$, where q is the integer quotient and $r = 0$ or 1.
3. Let $n^{(k)} = q$.
4. If $r = 0$, then $y_1^{(k)} + y_2^{(k)}i = y_1^{(k-1)} + y_2^{(k-1)}i$.
   If $r = 1$, then $y_1^{(k)} + y_2^{(k)}i = (y_1^{(k-1)} + y_2^{(k-1)}i)\ (z_1^{(k-1)} + z_2^{(k-1)}i)$.
5. If $n^{(k)} \neq 0$, then $z_1^{(k)} + z_2^{(k)}i = (z_1^{(k-1)} + z_2^{(k-1)})^2$, and steps 2 through 5 are repeated until $n^{(k)} = 0$.
6. When $n^{(k)} = 0$, and $j \geqq 0$, then $y_1 + y_2i = y_1^{(k)} + y_2^{(k)}i$.
   If $j < 0$, then $y_1 + y_2i = (1+ 0i)\ /\ (y_1^{(k)} + y_2^{(k)}i)$.

Exponentiation of a Real Base to a Real Power Subprograms

FDXPD# (REAL*8 Arguments)

FRXPR# (REAL*4 Arguments)

Algorithm

1. If $a = 0$ and $b \leqq 0$, error return.
   If $a = 0$ and $b > 0$, the answer is 0.
2. If $a \neq 0$ and $b = 0$, the answer is 1.
3. All other cases, compute a^b as $e^{b \cdot \log a}$. In this computation the exponential subroutine and the natural logarithm subroutine are used. If a is negative or if $b \cdot \log a$ is too large, an error return is given by one of these subroutines.

Error estimate

The relative error of the answer can be expressed as $(\epsilon_1 + \epsilon_2)\ b \cdot \log\ (a) + \epsilon_3$ where ϵ_1, ϵ_2, and ϵ_3 are relative errors of the logarithmic routine, machine multiplication, and the exponential routine, respectively.

For FDXPD#, $\epsilon_1 \leqq 3.5x10^{-16}$, $\epsilon_2 \leqq 2.2x10^{-16}$, and $\epsilon_3 \leqq 2.0x10^{-16}$. Hence the relative error $\leqq 5.7x10^{-16}x\ |\ b \cdot \log a\ |\ + 2.0x10^{-16}$. Note that $b \cdot \log a$ is the natural logarithm of the answer.

For FRXPR#, $\epsilon_1 \leqq 8.3x10^{-7}$, $\epsilon_2 \leqq 9.5x10^{-7}$, and $\epsilon_3 \leqq 4.7x10^{-7}$. Hence the relative error $\leqq 1.8x10^{-6}\ x\ |\ b \cdot \log a\ |\ + 4.7x10^{-7}$.

Effect of an Argument Error

$[a(1 + \delta_1)]\ b(1 + \delta_2) \cong a^b(1 + \delta_2 b \cdot \log a + b\delta_1)$. Note that if the answer does not overflow, $|b \cdot \log a| < 175$. On the other hand b can be very large without causing an overflow of a^b if $\log a$ is very small. Thus, if $a \cong 1$ and if b is very large, then the effect of the perturbation δ_1 of a shows very heavily in the relative error of the answer.

Exponentiation of a Real Base to an Integer Power Subprograms

FDXPI# (REAL*8 Arguments)

Algorithm

The value of $y = a^j$ is computed as follows: Let $|j| = \sum_{k=0}^{K} r_k 2^k$ where $r_k = 0$ or 1 for $k = 0, 1, \ldots, K$. Then $a^{|j|} = \pi_{r_k \neq 0}\ a^{2^k}$ and the factors a^{2^k} can be obtained by successive squaring.

More specifically:

1. Initially: $k = 0$, $n^{(0)} = |j|$, $y^{(0)} = 1$, and $z^{(0)} = a$.

2. Raise the index k by 1, and decompose $n^{(k-1)} = 2q + r$, where q is the integer quotient and $r = 0$ or 1.
3. Let $n^{(k)} = q$.
4. If $r = 0$, then $y^{(k)} = y^{(k-1)}$.
   If $r = 1$, then $y^{(k)} = y^{(k-1)}z^{(k-1)}$.
5. If $n^{(k)} \neq 0$, then $z^{(k)} = z^{(k-1)}z^{(k1-)}$, and steps 2 through 5 are repeated until $n^{(k)} = 0$.
6. When $n^{(k)} = 0$, and $j \geqq 0$, then $y = y^{(k)}$. If $j < 0$, then $y = \frac{1}{y^{(k)}}$.

Note: The negative exponent is computed by taking the reciprocal of the positive power. Thus it is not possible to compute 16.0**−64 because there is a lack of symmetry for real floating-point numbers – i.e., 16.0**−64 can be represented, but 16.0**64 cannot. The result is obtained by successive multiplications and is exact only if the answer contains less than 14 significant hexadecimal digits.

FRXPI# (REAL*4 Arguments)

Algorithm

This subprogram has the same algorithm as FIXPI#, which follows.

Exponentiation of Integer Base to Integer Power Subprogram

FIXPI# (INTEGER*4 Arguments)

Algorithm

The value of $L = I^j$ is computed as follows: Let $j = \sum_{k=0}^{K_1} r_k \cdot 2^k$ where $r_k = 0$ or 1 for $k = 0, 1, \ldots, K$. Then $I^j = \pi_{r_k \neq 0} I^{2^k}$, and the factors I^{2^k} can be obtained by successive squaring.

More specifically:

1. Initially: $k = 0$, $n^{(0)} = j$, $y^{(0)} = 1$, and $m^{(0)} = I$.
2. Raise the index k by 1, and decompose $n^{(k-1)} = 2q + r$, where q is the integer quotient and $r = 0$ or 1.
3. Let $n^{(k)} = q$.
4. If $r = 0$, then $y^{(k)} = y^{(k-1)}$.
   If $r = 1$, then $y^{(k)} = y^{(k-1)} \cdot m^{(k-1)}$.
5. If $n^{(k)} \neq 0$, then $m^{(k)} = m^{(k-1)} \cdot m^{(k-1)}$, and steps 2 through 5 are repeated until $n^{(k)} = 0$.
6. When $n^{(k)} = 0$, $L = L^{(k)}$.

Note: The result is obtained by successive multiplications. The result is exact only if it is less than (2**31) − 1. Results are meaningless when this limit is exceeded and may even be of changed sign.

Index

GC28-2026-4

IBM Time Sharing System FORTRAN IV Library Subprograms Printed in U.S.A. GC28-2026-4

IBM

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

IBM World Trade Corporation
360 Hamilton Avenue, White Plains, New York 10601
(International)