CRAY
RESEARCH, INC.

CRAY X-MP AND CRAY-1®
COMPUTER SYSTEMS

LIBRARY
REFERENCE MANUAL

SR-0014

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SR-0014

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	April, 1977 - First edition.
A	August, 1977 - Reprint with revision. This printing obsoletes the first edition.
B	June, 1978 - Reprint with revision. The addition of Section 4, Performance Statistics comprises the major change to this manual.
C	December, 1979 - This printing represents an expansion of the Mathematics Subroutines Reference Manual to include all CRAY-1 subprograms available to the user. To reflect this expansion, the title of the manual has been changed to the CRAY-1 Library Reference Manual. The subprograms listed and described in this manual reside in the $FTLIB, $SYSLIB, and $SCILIB libraries. This printing obsoletes the previous title of the manual and all previous printings.
C-01	January, 1980 - Technical corrections to pages 4-17 and 4-18.
D	April, 1980 - This printing represents a reprint with revision, bringing the manual into agreement with the released system, version 1.08. It obsoletes all previous printings. Major additions include unblocked I/O, directive processing, SKOL support, and library error processing. Replacing the text of the previous summary section is a subprogram list with subprograms grouped by UPDATE deck name within each library. An index section has also been added. Because of the extensive reorganization and repagination of this manual, editing changes are not noted. Technical changes are noted by change bars. The publication number has been changed from 2240014 to SR-0014.

D-01 April, 1980 - This change packet clarifies the formula for the OPFILT routine and corrects the formulas and one parameter for the Fourier transform routines.

E October, 1980 - This reprint with revision brings the manual into agreement with version 1.09 of the released system. Major features include open, close, and inquire routines; SMACH, which returns machine constants, and FOLR and SOLR, which solve first- and second-order linear occurrences. Other revisions include $PAUSE, a special-purpose routine that eliminates the need to substitute $STOP; and an update to the performance statistics for single-precision, single-argument routines. The index now includes all routines listed in the manual. All technical changes except changes to the index are noted by change bars.

All previous versions of this manual are obsolete.

E-01 June, 1981 - This change packet brings the manual into agreement with version 1.10 of the released system. Major features include FORTRAN 77 trigonometric, hyperbolic, and character routines; linear algebra routines, ISEARCH, ISAMIN, FOLRN, SOLRN, and SOLRN3; logical record I/O and dataset control routines, READIBM and WRITIBM; definition and control, SDACCESS; directive processing routine, CEXPR; job control routines, RERUN and NORERUN; and miscellaneous special purpose routines, ECHO, JNAME, LOGECHO, WFBUFFER.

F June, 1982 - This rewrite brings the manual into agreement with version 1.11 of the released COS system and version 1.10 of CFT and $FTLIB. Sections 4 and 5 have been reorganized. Most of the routines in section 3 are now in table format and the calling sequence format has changed. Also, section 2 now contains a subprogram summary with page references. The major new features are tapes and ORDERS, an in-memory sort routine.

All previous versions of this manual are obsolete.

G July, 1983 - This reprint with revision brings the manual into agreement with version 1.12 of the COS system and version 1.11 of CFT. Major changes include the new calling sequence and the split of $FTLIB. Also included are random access dataset I/O routines, dataset skip routines, search routines, and exchange package routines.

Because of extensive repagination of this manual, editing changes are not noted. Technical changes are noted by change bars. All previous versions of this manual are obsolete.

G-01 September, 1983 - This change packet adds Pascal subprograms to the manual.

H	February, 1984 - This reprint with revision brings the manual into agreement with version 1.13 of COS and CFT. Major changes include the addition of the heap and stack routines, array search routines, and bidirectional memory transfer routines; additions to the Linpack, Eispack, and random access routines; and the multitasking capability. Numerous minor changes have also been made. All previous versions of this manual are obsolete.
I	December, 1984 - This reprint with revision brings the manual into agreement with version 1.14 of COS and CFT. Major additions include explicit data conversion routines, byte and bit manipulation routines, tape positioning and synchronizing routines, new Pascal routines, and a number of miscellaneous special purpose routines. Reorganizational changes include moving routines MVC, PACK, UNPACK, and PUTBYT from the miscellaneous special purpose subsection to the byte and bit manipulation subsection. Appendix B contains new statistics for single-precision, single-argument subprograms. Appendix C, formerly containing processing times for the routine ORDERS, now contains sort entry points. The tables containing processing times have been moved to the location of the description of the routine ORDERS. Numerous minor changes have also been made. All previous versions of this manual are obsolete.

PREFACE

This publication describes the subprograms available to users of the CRAY-1 and CRAY X-MP Computer Systems. It also contains subprograms that allow the translation of a subset of International Business Machines (IBM) or Control Data Corporation (CDC) file types online to a Cray Computer System.

The user of this manual is assumed to be familiar with the Cray Operating System (COS), and either the Cray FORTRAN Compiler (CFT), or the Cray Assembly Language (CAL). The following Cray Research publications might be helpful:

CRAY-OS Version 1 Reference Manual, publication SR-0011

Macros and Opdefs Reference Manual, publication SR-0012

FORTRAN (CFT) Reference Manual, publication SR-0009

CAL Assembler Version 1 Reference Manual, publication SR-0000

CRAY-OS Message Manual, publication SR-0039

SKOL Reference Manual, publication SR-0033

COS Table Descriptions Internal Reference Manual, publication SM-0045†

Pascal Reference Manual, publication SR-0060

---

† This manual is available only on tape. See your CRI site analyst for information.

CONTENTS

7. SPECIAL PURPOSE SUBPROGRAMS 7-1

8. PASCAL SUBPROGRAMS 8-1

8. PASCAL SUBPROGRAMS (continued)

APPENDIX SECTION

FIGURES

TABLES

TABLES (continued)

INDEX

INTRODUCTION 1

This manual describes the subprograms provided in the standard libraries, $ARLIB, $FTLIB, $IOLIB, $SCILIB, $SYSLIB, and $UTLIB. Routines generated by CFT in the form of inline code are not included in this manual but are described in the FORTRAN (CFT) Reference Manual, CRI publication SR-0009.

Section 2 provides a subprogram summary grouped alphabetically by primary reference name within each library. Sections 3 through 7 provide detailed descriptions and calling sequences of the subprograms listed in section 2. Sections 3 through 6 cover specific subprogram applications and common mathematics, scientific mathematics, input/output, and dataset management. Section 7 is a collection of special purpose subprograms, grouped by application. These applications are debugging aids, job control, floating-point interrupt control, time and date requests, and control statement processing. The last subsection in section 7 consists of routines whose applications do not lend themselves to any particular grouping and are therefore listed separately. Section 8 presents procedures and functions that reside in the Pascal runtime library, $PSCLIB. Section 9 describes multitasking subprograms.

SUBPROGRAM CLASSIFICATION

The subprograms described in this manual are either subroutines, scalar functions, or vector functions. Some subprograms can be used as either subroutines or scalar functions. Scalar functions produce a single result while vector functions produce a vector of results. Some vector functions (called *pseudo vector functions*) call the corresponding scalar functions. Such a scalar function call can occur when the vector function performs infrequently used calculations or when the calculation is not suited to vectorization.

In general, arguments to vector call-by-value functions are passed in V registers; scalar arguments are broadcast if necessary. However, some routines implicitly called by CFT, such as exponentiation, have mixed scalar and vector arguments.

CFT LINKAGE METHODS

The CFT-callable library routines are accessed by one of two methods: call-by-address or call-by-value. Subroutines are always accessed by the call-by-address method. Library functions intrinsic to CFT, or user functions named in a VFUNCTION directive, are accessed by CFT in either call-by-address or call-by-value mode, depending on context.

In call-by-value mode, arguments are loaded into either S or V registers, and the function returns its result in S1 or V1 (S2 or V2 are also used for complex or double-precision functions). Vector functions must also have the vector length present in the VL register.

In call-by-address mode, addresses of arguments are stored sequentially in memory. Under the CFT 1.10 version of the calling sequence, the address of the first argument is stored at entry 1, the second at entry 2, etc. Under the CFT 1.11 version, the list of addresses is stored in a block of memory allocated for that purpose. Functions return their results in registers. Subroutines return results through their argument lists. (For information on the new calling sequence, see the Macros and Opdefs Reference Manual, CRI publication SR-0012.)

By convention a call-by-value has a % suffix (for example, SIN%), and a vector call-by-value has a % prefix and suffix as shown below.

Type	Call by Address	Call by Value
Scalar	RTE	RTE%
Vector	%RTE	%RTE%

Routines that are accessible from CAL programs only also can be prefixed with a **$**.

CFT LINKAGE MACROS

CFT linkage macros generate code to handle subprogram linkage between CFT-compiled routines and CAL-assembled routines. These linkage macros and their uses follow.

CALL	Provides linkage to call-by-address routines
CALLV	Provides linkage to call-by-value routines

ENTER	Reserves space for parameter addresses, saves B and T registers, and sets up traceback linkage
EXIT	Initiates a return from a routine to its caller; restores any B or T registers not considered scratch by CFT.

CFT linkage macros should be used whenever possible to maintain compatibility across versions of CFT. See the Macros and Opdefs Reference Manual, CRI publication SR-0012, for detailed descriptions of CFT linkage macros and linkage conventions.

All $ARLIB, $FTLIB, $IOLIB, $SCILIB, $UTLIB, and $SYSLIB subroutines can use any of the A, S, V, VL, and VM registers as scratch registers; therefore, the calling routine should not depend on any of these registers being preserved. However, these routines preserve the contents of registers B01 through $B65_8$ and T00 through $T67_8$. Registers $B70_8$ through $B77_8$ and $T70_8$ through $T77_8$ also can be used as scratch registers.

NOTE

Cray Research, Inc., reserves the right to make future use of any of the A, S, V, VL, VM, B66-B77, and T70-T77 registers in any library subroutine. Users cannot depend on the contents of these registers being preserved by any library routine.

CONVENTIONS

The following conventions are used in this manual.

Convention	Description
Italics	Define generic terms representing words or symbols to be supplied by the user and identify new terms
[] Brackets	Enclose optional portions of a command format
(S1), (S2), etc.	Content of register S1, S2, etc., respectively

Arguments are used on entry unless exit or return conditions are specified.

SUBPROGRAM SUMMARY 2

INTRODUCTION

This section summarizes the subprograms in this manual. These subprograms are callable from CAL or Cray FORTRAN programs and reside in the $ARLIB, $FTLIB, $IOLIB, $UTLIB, $SYSLIB, and $SCILIB libraries.

$ARLIB contains routines primarily concerned with returning some numeric result. Mathematical routines intrinsic to FORTRAN such as SIN reside here.

$FTLIB contains CFT-specific routines such as ICHAR, LEN, and LOC.

$IOLIB contains routines that move data from external devices to main memory or control that movement.

$UTLIB contains routines more infrequently used and of a utilitarian nature.

$SCILIB routines perform operations such as matrix multiply or Fast Fourier transform and must be explicitly called. Such processes are not intrinsic properties of the Cray FORTRAN language and are independent of specific Cray Operating System (COS) features.

$SYSLIB routines usually link directly to the operating system through a normal exit. These routines are not usually accessible from a Cray FORTRAN program, but are called by $IOLIB and $UTLIB routines for specific tasks. In general, $SYSLIB serves as a link between the general purpose $IOLIB and $UTLIB routines and the details of COS. $SYSLIB routines depend on specific COS features.

Subprograms implicitly called by a CFT routine (for example, routines used for exponentiation or I/O) have a $ or % character in their names. They are not directly callable by a Cray FORTRAN program.

TABLE DESCRIPTION

Table 2-1 contains the subprogram summary that includes the following items.

- Primary reference name
- Page number
- UPDATE deck name
- Entry type
- CFT call type
- Library
- OS dependency
- Purpose

PRIMARY REFERENCE NAME

Primary reference name is a general group name identifying a subprogram and is generally similar to the subprogram name. For example, BACKSP is the primary reference name for the backspace subprograms BACK, $BACK, BKFILE, BKSP, $BKSP, and BKSPF. The subprograms are alphabetized by primary reference name.

PAGE NUMBER

The page numbers of subprogram locations are listed under each primary reference name and reference detailed descriptions of the subprogram.

UPDATE DECK NAME

UPDATE deck name is the listed name of the subprogram in the UPDATE program library.

ENTRY TYPE

Entry type indicates the source of the subprogram call, either CFT or CAL. Entries callable from CFT are further divided into (1) call-by-address and (2) call-by-value. See the description of these linkage methods in the introduction to this manual.

CFT CALL TYPE

CFT call type indicates three classifications for CFT callable subprograms:

S	Subroutine
SF	Scalar function
VF	Vector function

See the introduction to this manual for a description of these classifications.

LIBRARY

The library column indicates the library residence of the subprogram.

OS DEPENDENCY

Each subprogram is labeled either OS dependent (Dep.) or OS independent (Ind.). This classification is a guideline for use of the routine under operating systems other than the current version of COS. Independent routines can be executed under other operating systems with minor changes such as macro redefinition or substitution of external routines. Dependent routines rely heavily on COS features.

PURPOSE

The purpose is a 1- or 2-line description of the subprogram.

PASCAL SUBPROGRAMS

Table 2-2 summarizes Pascal subprograms with a format similar to that of table 2-1. Exceptions are (1) the primary reference name and UPDATE deck name are always the same; (2) those subprograms callable from CAL only are indicated with an X; and (3) the library is $PSCLIB for all Pascal subprograms.

Table 2-1. Subprogram summary

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ABORT	ABORT	ABORT			S	$UTLIB	Ind.	Aborts job with traceback
ABS	REAL	ABS			SF	$ARLIB	Ind.	Computes absolute value
ACCESS	ACS	ACCESS			S	$SYSLIB	Dep.	Accesses a permanent dataset
ACOS	ACOS ACOSV	ACOS	ACOS% %ACOS%	 %ACOS	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes arccosine Computes vectorized arccosine
ACPTBAD	RCVRBAD	ACPTBAD			S	$SYSLIB	Dep.	Transfers bad data to a specified buffer for the caller

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ACQUIRE	AQR	ACQUIRE			S	$SYSLIB	Dep.	Accesses a permanent dataset or acquires a front-end resident dataset and stages it to the Cray mainframe
ACTTABLE	ACTTABLE	ACTTABLE			S	$SYSLIB	Dep.	Returns job accounting table
ADJUST	ADJ	ADJUST			S	$SYSLIB	Dep.	Expands or contracts a permanent dataset
AIMAG	COMPLX	AIMAG			SF	$ARLIB	Ind.	Returns imaginary part of a complex number
AINT	REAL	AINT			SF	$ARLIB	Ind.	Truncates to integral value

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ALF	ALF	ADDLFT		$ALF	S	$SYSLIB	Dep.	Adds name to Logical File Table (LFT)
AMOD	REAL	AMOD			SF	$ARLIB	Ind.	Computes division remainder
AMU	TM			$AMU		$SYSLIB	Dep.	Returns total allotted table space
AND	BOOLEAN	AND			SF	$ARLIB	Ind.	Forms logical product
ANINT	REAL	ANINT			SA	$ARLIB	Ind.	Calculates nearest whole number
ARERP	ARERP			ARERP%		$ARLIB	Dep.	Processes $ARLIB errors

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ARGPLIMQ	FLOW	ARGPLIMQ			S	$UTLIB	Ind.	Controls listing of argument values for every call and return
ASCDC	ASCDC	ASCDC			S	$UTLIB	Ind.	Converts ASCII character to display code character
ASIN	ACOS ACOSV	ASIN	ASIN% %ASIN%	 %ASIN	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes arcsine Computes vectorized arcsine
ASSIGN	ASS	ASSIGN			S	$SYSLIB	Dep.	Opens dataset and assigns characteristics to it
ATAN	ATAN ATANV	ATAN	ATAN% %ATAN%	 %ATAN	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes arctangent Computes vectorized arctangent

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ATAN2								
	ATAN2	ATAN2	ATAN2%		SF	$ARLIB	Ind.	Computes 2-argument arctangent
	ATAN2V		%ATAN2%	%ATAN2	VF	$ARLIB	Ind.	Computes vectorized 2-argument arctangent
ATS								
	TM			$ATS		$SYSLIB	Dep.	Allocates table space
AXPY								
	CAXPY	CAXPY			S	$SCILIB	Ind.	Computes $y=ax+y$ on complex arrays x and y
	SAXPY	SAXPY			S	$SCILIB	Ind.	Computes $y=ax+y$ on real arrays x and y
	SPAXPY	SPAXPY			S	$SCILIB	Ind.	Computes $y=ax+y$ on real arrays x and y when y is referenced indirectly
BACKSPACE								
	BACK			$BACK		$IOLIB	Dep.	Backspaces one record
	BKFILE	BACKFILE			S	$SYSLIB	Dep.	Backspaces one file
	BKSP			$BKSP		$SYSLIB	Dep.	Backspaces one record
				$BKSPF		$SYSLIB	Dep.	Backspaces one file

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
BICON								
	BICONV	BICONV			S	$UTLIB	Ind.	Converts integer to ASCII character
	BICONZ	BICONZ			S	$UTLIB	Ind.	
BTD								
	BTD	BTD	BTD%	$BTD	SF	$UTLIB	Ind.	Converts binary number to ASCII decimal
		BTD		BTD%	SF	$UTLIB	Ind.	Converts binary to decimal ASCII right justified, blank filled.
		BTDR		BTDR%	SF	$UTLIB	Ind.	Converts binary to decimal ASCII right justified, zero filled.
		BTDL		BTDL%	SF	$UTLIB	Ind.	Converts binary to decimal ASCII left justified, zero filled.
BTO								
	BTO	BTO	BTO%	$BTO	SF	$UTLIB	Ind.	Converts binary number to ASCII octal
		BTO		BTO%	SF	$UTLIB	Ind.	Converts binary to octal ASCII right justified, blank filled.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
BTO continued		BTOR		BTOR%	SF	$UTLIB	Ind.	Converts binary to octal ASCII right justified, zero filled.
		BTOL		BTOL%	SF	$UTLIB	Ind.	Converts binary to octal ASCII left justified, zero filled.
CABS	CABS	CABS	CABS%		SF	$ARLIB	Ind.	Calculates complex absolute value
	CABSV		%CABS%	%CABS	VF	$ARLIB	Ind.	Calculates vectorized complex absolute value
CCOS	CCOS	CCOS	CCOS%		SF	$ARLIB	Ind.	Computes complex cosine
	CCOSV		%CCOS%	%CCOS	VF	$ARLIB	Ind.	Computes vectorized complex cosine

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CDC	CDCI			$CDCI		$IOLIB	Dep.	Translates CDC formatted input data
	CDCO			$CDCO		$IOLIB	Dep.	Translates CDC formatted output data
CEXPR	CEF	CEXPR			S	$SYSLIB	Ind.	Transforms an expression character string to a Reverse Polish Table
CHAR	CARCON			$MOVE		$FTLIB	Ind.	Transfers one character item to result
				$PAD		$FTLIB	Ind.	Terminates transfer
	OCARCON			$CCI		$FTLIB	Ind.	Initializes concatenation for store
				$CCT		$FTLIB	Ind.	Transfers one character item to result
				$CCF		$FTLIB	Ind.	Terminates transfer
	CHARCPR	LGE		$GE	SF	$FTLIB	Ind.	Compares ASCII arguments for greater than or equal to
		LGT		$GT	SF	$FTLIB	Ind.	Compares ASCII arguments for greater than

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CHAR continued		LLT		$LT	SF	$FTLIB	Ind.	Compares ASCII arguments for less than
		LLE		$LE	SF	$FTLIB	Ind.	Compares ASCII arguments for less than or equal to
				$EQ		$FTLIB	Ind.	Compares ASCII arguments for equality
				$NE		$FTLIB	Ind.	Compares ASCII arguments for nonequality
	INDEX	INDEX			SF	$FTLIB	Ind.	Finds position of second argument as substring of first argument
	LEN	LEN			SF	$FTLIB	Ind.	Finds length of argument
CHCONV	CHCONV	CHCONV			S	$UTLIB	Ind.	Converts ASCII character to integer
CLEARBT	BTMODE	CLEARBT			S	$UTLIB	Dep.	Temporarily disables bidirectional memory transfers

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CLEARBTS	BTMODE	CLEARBTS			S	$UTLIB	Dep.	Permanently disables bidirectional memory transfers
CLEARFI	FIMODE	CLEARFI			S	$UTLIB	Dep.	Temporarily prohibits floating-point interrupts
CLEARFIS	FIMODE	CLEARFIS			S	$UTLIB	Dep.	Permanently prohibits floating-point interrupts
CLOCK	CLOCK	CLOCK			S/SF	$UTLIB	Dep.	Supplies current system clock in *hh*:*mm*:*ss* format
CLOSE	CLOSE	CLOSE		$CLS	SA	$IOLIB	Dep.	Terminates the connection of a dataset to a unit
	SYMDBC	CLOSE			S	$SYSLIB	Dep.	Closes a random, unblocked dataset

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CMPLX	REAL	CMPLX			SF	$ARLIB	Ind.	Converts two reals to complex
COMPL	BOOLEAN	COMPL			SF	$ARLIB	Ind.	Forms logical complement
CONJG	COMPLX	CONJG			SF	$ARLIB	Ind.	Computes complex conjugate
COPY	CCOPY	CCOPY			S	$SCILIB	Ind.	Copies complex array into complex array
	SCOPY	SCOPY			S	$SCILIB	Ind.	Copies real array into real array
COPYD	COPY	COPYD			S	$IOLIB	Dep.	Copies one dataset to another; EOD not copied

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
COPYF	COPY	COPYF			S	$IOLIB	Dep.	Copies files from one dataset to another; EOD not written
COPYR	COPY	COPYR			S	$IOLIB	Dep.	Copies records from one dataset to another; EOF not written
COPYU	COPYU	COPYU			S	$IOLIB	Dep.	Copies data to EOD in unblocked format
COS	COS COSV	COS	COS% %COS%	%COS	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes cosine Computes vectorized cosine
COSH	COSH COSHV	COSH	COSH% %COSH%	%COSH	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes hyperbolic cosine Computes vectorized hyperbolic cosine

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
COSS	COSS	COSS		COSS%	SF	$ARLIB	Ind.	Computes cosine and sine
	COSSV			%COSS	VF	$ARLIB	Ind.	Computes vectorized cosine and sine
				%COSS%	VF	$ARLIB	Ind.	Same as %COSS
COSSH	COSH	COSSH		COSSH%	SF	$ARLIB	Ind.	Computes hyperbolic cosine and sine
	COSHV			%COSSH	VF	$ARLIB	Ind.	Computes vectorized hyperbolic cosine and sine
	COSHV			%COSSH%	VF	$ARLIB	Ind.	Same as %COSSH
COT	COT	COT	COT%		SF	$ARLIB	Ind.	Computes cotangent
	COTV		%COT%	%COT	VF	$ARLIB	Ind.	Computes vectorized cotangent
CRACK	CRACK	CRACK			S	$SYSLIB	Dep.	Reformats a user-supplied string into verb, separators, keywords and values

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CS								
	CCS	CCS		$CS $CCS	S S	$SYSLIB $SYSLIB	Dep. Dep.	Cracks control statement Cracks control statement
CSIN								
	CSIN	CSIN	CSIN%		SF	$ARLIB	Ind.	Computes complex sine
	CSINV		%CSIN%	%CSIN	VF	$ARLIB	Ind.	Computes vectorized complex sine
CTOC								
	CTOCSS		CTOC%		SF	$ARLIB	Ind.	Raises complex scalar to complex scalar power
	CTOCSV		CTO%C%		VF	$ARLIB	Ind.	Raises complex scalar to complex vector power
	CTOCVS		%CTOC%		VF	$ARLIB	Ind.	Raises complex vector to complex scalar power
	CTOCVV		%CTO%C%		VF	$ARLIB	Ind.	Raises complex vector to complex vector power
CTOI								
	CTOISS		CTOI%		SF	$ARLIB	Ind.	Raises complex scalar to integer scalar power
	CTOISV		CTO%I%		VF	$ARLIB	Ind.	Raises complex scalar to integer vector power

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
CTOI continued	CTOIVS		%CTOI%		VF	$ARLIB	Ind.	Raises complex vector to integer scalar power
	CTOIVV		%CTO%I%		VF	$ARLIB	Ind.	Raises complex vector to integer vector power
CTOR								
	CTORSS		CTOR%		SF	$ARLIB	Ind.	Raises complex scalar to real scalar power
	CTORSV		CTO%R%		VF	$ARLIB	Ind.	Raises complex scalar to real vector power
	CTORVS		%CTOR%		VF	$ARLIB	Ind.	Raises complex vector to real scalar power
	CTORVV		%CTO%R%		VF	$ARLIB	Ind.	Raises complex vector to real vector power
DABS								
	DBLE	DABS			SA	$ARLIB	Ind.	Determines double-precision absolute value
DATAN								
	DATAN	DATAN	DATAN%		SF	$ARLIB	Ind.	Computes double-precision arctangent
	DATANV		%DATAN%	%DATAN	VF	$ARLIB	Ind.	Computes vectorized double-precision arctangent

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DATAN2								
	DATAN	DATAN2	DATAN2%		SF	$ARLIB	Ind.	Computes double-precision 2-argument arctangent
	DATANV		%DATAN2%	%DATAN2	VF	$ARLIB	Ind.	Computes vectorized double-precision 2-argument arctangent
DATETIME								
	DATE	DATE			S/SF	$UTLIB	Dep.	Returns current date in *mm*:*dd*:*yy* format
	JDATE	JDATE			S/SF	$UTLIB	Dep.	Returns current Julian date in *yyddd* format
	TIMEF	TIMEF			SF	$UTLIB	Dep.	Returns time since initial TIMEF call in milliseconds
DBLE								
	REAL	DBLE			SF	$ARLIB	Ind.	Converts type real to double precision
DCOS								
	DCOS	DCOS	DCOS%		SF	$ARLIB	Ind.	Computes double-precision cosine
	DCOSV		%DCOS%	%DCOS	VF	$ARLIB	Ind.	Computes vectorized double-precision cosine

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DDIM	DDIM DDIMV	DDIM	DDIM% %DDIM%	 %DDIM	SF VF	$ARLIB $ARLIB	Ind. Ind.	Double-precision positive real difference
DECODE	RFD			$DFI $DFV $DFA $DFV% $DFF		$IOLIB $IOLIB $IOLIB	Dep. Dep. Dep.	Initializes for decode Cracks format; decodes input. Terminates decode
DELETE	DELETE	DELETE			S	$SYSLIB	Dep.	Removes a saved dataset from the Dataset Catalog
DELTSK	DELTSK			$DELTSK%		$UTLIB	Ind.	Deletes calling task
DIM	REAL	DIM			SF	$ARLIB	Ind.	Computes positive real difference

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DINT	DINT	DINT			SF	$ARLIB	Ind.	Truncates double-precision numbers to integers
	DINTV				VF	$ARLIB	Ind.	
DISPOSE	DISPOSE	DISPOSE			S	$SYSLIB	Dep.	Directs a dataset to the specified queue
DMOD	DMOD	DMOD	DMOD%		SF	$ARLIB	Ind.	Computes division remainder
			%DMOD	%DMOD	VF	$ARLIB	Ind.	
DNINT	DNINT	DNINT	DNINT%		SF	$ARLIB	Ind.	Calculates nearest integer
	DNINTV		%DNINT%	%DNINT	VF	$ARLIB	Ind.	
DOT	CDOT	CDOTC			SF	$SCILIB	Ind.	Finds the conjugated dot product of two complex arrays
		CDOTU			SF	$SCILIB	Ind.	Finds the unconjugated dot product of two complex arrays

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DOT continued	SDOT	SDOT			SF	$SCILIB	Ind.	Finds the dot product of two real arrays
	SPDOT	SPDOT			SF	$SCILIB	Ind.	Finds the dot product of two real arrays; one array is referenced indirectly.
DPREC								
	DASS		DASS%		SF	$ARLIB	Ind.	Performs double-precision addition
	DDSS		DDSS%		SF	$ARLIB	Ind.	Performs double-precision division
	DMSS		DMSS%		SF	$ARLIB	Ind.	Performs double-precision multiplication
	DSSS		DSSS%		SF	$ARLIB	Ind.	Performs double-precision subtraction
	DAVV		DASV%		VF	$ARLIB	Ind.	Performs double-precision addition, scalar+vector
			DAVS%		VF	$ARLIB	Ind.	Performs double-precision addition, vector+scalar
			DAVV%		VF	$ARLIB	Ind.	Performs double-precision addition, vector+vector
			DDSV%		VF	$ARLIB	Ind.	Performs double-precision division, scalar/vector
	DDVV		DDVS%		VF	$ARLIB	Ind.	Performs double-precision division, vector/scalar
			DDVV%		VF	$ARLIB	Ind.	Performs double-precision division, vector/vector

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DPREC continued	DVVV		DMSV%		VF	$ARLIB	Ind.	Performs double-precision multiplication, scalar x vector
			DMVS%		VF	$ARLIB	Ind.	Performs double-precision multiplication, vector x scalar
			DMVV%		VF	$ARLIB	Ind.	Performs double-precision multiplication, vector x vector
	DSVV		DSSV%		VF	$ARLIB	Ind.	Performs double-precision subtraction, scalar-vector
			DSVS%		VF	$ARLIB	Ind.	Performs double-precision subtraction, vector-scalar
			DSVV%		VF	$ARLIB	Ind.	Performs double-precision subtraction, vector-vector
DPROD	DBLE	DPROD			SA	$ARLIB	Ind.	Performs double-precision product of two real arguments
DRIVER	DRIVER	DRIVER		S	$SYSLIB		Dep.	Allows a user to directly program a CRAY channel on an IOS

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type: CFT Standard: Address	Entry Type: CFT Standard: Value	Entry Type: Call From CAL Only	CFT Call Type	Library	OS Dep	Purpose
DSASC	DSASC	DSASC			S	$UTLIB	Ind.	Converts display code character to ASCII character
DSIGN	DBLE		DSIGN		SA	$ARLIB	Ind.	Transfers sign from one double-precision number to another
DSIN	DCOS	DSIN	DSIN%		SF	$ARLIB	Ind.	Computes double-precision sine
	DCOSV		%DSIN%	%DSIN	VF	$ARLIB	Ind.	Computes vectorized double-precision sine
DSNDSP	DSNDSP			$DSNDSP		$SYSLIB	Dep.	Searches Logical File Table (LFT) in user's I/O area for dataset name
DTB	DTB	DTB	DTB%	$DTB	SF	$UTLIB	Ind.	Converts ASCII decimal integer number to binary

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DTOD								
	DTODSS		DTOD%		SF	$ARLIB	Ind.	Raises double-precision scalar to double-precision scalar power
	DTODSV		DTO%D%		VF	$ARLIB	Ind.	Raises double-precision scalar to double-precision vector power
	DTODVS		%DTOD%		VF	$ARLIB	Ind.	Raises double-precision vector to double-precision scalar power
	DTODVV		%DTO%D%		VF	$ARLIB	Ind.	Raises double-precision vector to double-precision vector power
DTOI								
	DTOISS		DTOI%		SF	$ARLIB	Ind.	Raises double-precision scalar to integer scalar power
	DTOISV		DTO%I%		VF	$ARLIB	Ind.	Raises double-precision scalar to integer vector power

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DTOI continued	DTOIVS		%DTOI%		VF	$ARLIB	Ind.	Raises double-precision vector to integer scalar power
	DTOIVV		%DTO%I%		VF	$ARLIB	Ind.	Raises double-precision vector to integer vector power
DTOR								
	DTODSS		DTOR%		SF	$ARLIB	Ind.	Raises double-precision scalar to real scalar power
	DTODSV		DTO%R%		VF	$ARLIB	Ind.	Raises double-precision scalar to real vector power
	DTODVS		%DTOR%		VF	$ARLIB	Ind.	Raises double-precision vector to real scalar power
	DTODVV		%DTO%R%		VF	$ARLIB	Ind.	Raises double-precision vector to real vector power
DUMP								
	CRAYDUMP	CRAYDUMP			S	$UTLIB	Ind.	Prints a memory dump to a specified dataset
	PDUMP	DUMP		$DUMP	S	$UTLIB	Dep.	Dumps memory and aborts job
	PDUMP	PDUMP		$PDUMP	S	$UTLIB	Dep.	Dumps memory

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
DUMPJOB	DUMPJOB	DUMPJOB				$SYSLIB	Dep.	Creates an unblocked dataset containing the user job area image
ECHO	ECHO	ECHO			S	$SYSLIB	Dep.	Allows user to turn on and off classes of messages to user logfile
EISPACK					S	$SCILIB	Ind.	See table 4-4.
ENCODE	WFD			$EFI		$IOLIB	Dep.	Initializes for encode
	WFD			$EFV $EFV% $EFA		$IOLIB	Dep.	Cracks format; encodes output.
	WFD			$EFF		$IOLIB	Dep.	Finalizes encode
END	END		$END END$		S	$UTLIB	Ind.	Terminates current job step and advances job to next job step

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ENDFILE	EODW	EODW		$EODW	S	$SYSLIB	Dep.	Writes EOD, also EOF and EOR if necessary; clears UEOF flag in DSP.
	EOFR			$EOFR		$SYSLIB	Dep.	Detects EOF on last I/O operation; clears UEOF flag in DSP.
	EOFW			$EOFW		$IOLIB	Dep.	Writes EOF, also EOR if necessary; clears UEOF flag in DSP.
ENDRPV	RPV	ENDRPV		$ENDRPV	S	$SYSLIB	Dep.	Continues normal exit processing after a reprievable request has been processed
EOADF	EOADF			$EOATEST		$SYSLIB	Dep.	Checks for a read/write past the allocated area condition

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
EOF	IEOF	EOF			SF			Returns EOF status; clears UEOF in DSP.
	IEOF	IEOF		$IEOF	SF	$IOLIB	Dep.	
EQV	BOOLEAN	EQV			SF	$ARLIB	Ind.	Computes logical equivalence
ERREXIT	ABORT	ERREXIT			S	$UTLIB	Ind.	Aborts current job step
EVASGN	EVASGN	EVASGN			S	$UTLIB	Ind.	Identifies event
EVCLEAR	EVCLEAR	EVCLEAR			S	$UTLIB	Ind.	Clears event
EVPOST	EVPOST	EVPOST			S	$UTLIB	Ind.	Posts an event

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
EVREL	EVREL	EVREL			S	$UTLIB	Ind.	Releases event identifier
EVTEST	EVTEST	EVTEST			S	$UTLIB	Ind.	Tests event for posted state
EVWAIT	EVWAIT	EVWAIT			S	$UTLIB	Ind.	Delays calling task until event is posted
EXCHANGE	XPFMT FXPF XPFMT	XPFMT FXP B20CT		 $FXP	S S S	$SYSLIB $SYSLIB $SYSLIB	Ind. Dep. Dep.	Format Exchange Package Print Exchange Package Format octal representation of Exchange Package
EXIT	EXIT	EXIT			S	$UTLIB	Ind.	Terminates current job step

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
EXP								
	CEXP	CEXP	CEXP%		SF	$ARLIB	Ind.	Computes complex exponential
	CEXPV		%CEXP%	%CEXP	VF	$ARLIB	Ind.	Computes vector complex exponential
	DEXP	DEXP	DEXP%		SF	$ARLIB	Ind.	Computes double-precision exponential
	DEXPV		%DEXP%	%DEXP	VF	$ARLIB	Ind.	Computes vector double-precision exponential
	EXP	EXP	EXP%		SF	$ARLIB	Ind.	Calculates exponential
	EXPV		%EXP%	%EXP	VF	$ARLIB	Ind.	Calculates vector exponential
FETCH								
	FETCH	FETCH			S	$SYSLIB	Dep.	Obtains a front-end resident dataset and makes it local to a job on a Cray computer
FFT								
	CRFFT2	CRFFT2			S	$SCILIB	Ind.	Computes a Fast Fourier Transform
	CFFT2	CFFT2			S	$SCILIB	Ind.	
	RCFFT2	RCFFT2			S	$SCILIB	Ind.	

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
FP6064	FP6064	FP6064			S	$UTLIB	Ind.	Converts CDC single-precision to Cray single-precision
FP6460	FP6460	FP6460			S	$UTLIB	Ind.	Converts Cray single-precision to CDC single-precision
FILTER	FILTERG	FILTERG			S	$SCILIB	Ind.	Calculates general filter coefficients
	FILTERS	FILTERS			S	$SCILIB	Ind.	Calculates symmetric filter coefficients
	OPFILT	OPFILT			S	$SCILIB	Ind.	Solves equations by the Weiner-Levinson method
FINDCH	FINDCH	FINDCH			S	$UTLIB	Ind.	Searches for the occurrence of a specified character string.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
FLOAT	INTEGER	FLOAT			SF	$ARLIB	Ind.	Converts type integer to real
FTERP	FTERP		FTERP			$FTLIB	Dep.	Processes $FTLIB errors
FLOWTRACE	FLOW	ARGPLIMQ			S	$UTLIB	Ind.	Controls argument printing for flowtrace
	FLOW	FLOWENTR			S	$UTLIB	Ind.	Processes CFT flowtrace option for calls
	FLOW	FLOWEXIT			S	$UTLIB	Ind.	Processes CFT flowtrace option for RETURN statements
	FLOW	FLOWLIM			S	$UTLIB	Ind.	Sets limit on the number of subroutine calls to be traced
	FLOW	FLOWSTOP			S	$UTLIB	Ind.	Processes CFT flowtrace option
	FLOW	SETPLIMQ			S	$UTLIB	Ind.	Controls CALL printing for flowtrace

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
FOLR	FOLR	FOLR			S	$SCILIB	Ind.	Solves first-order linear recurrences
FOLRN	FOLRN	FOLRN			S	$SCILIB	Ind.	Solves for the last term of a first-order linear recurrence
FOLRP	FOLRP	FOLRP			S	$SCILIB	Ind.	Solves first-order linear recurrences
FOLR2	FOLR2	FOLR2			S	$SCILIB	Ind.	Solves first-order linear recurrences
FOLR2P	FOLR2P	FOLR2P			S	$SCILIB	Ind.	Combination of FOLRP & FOLR2

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
GATHER	GATHER	GATHER			S	$SCILIB	Ind.	Gathers a vector from a source vector
GETB1	SYMDBC	GETB1			SF	$SYSLIB	Dep.	Returns address of subroutine name
GETDSP	GTDSP	GETDSP		$GTDSP $GTDSP%	SF	$SYSLIB	Dep.	Locates a Dataset Parameter Table
GETLPP	GLPP	GETLPP			SF	$SYSLIB	Dep.	Returns lines from JCLPP
GETNAMEQ	GNAMEQ	GETNAMEQ			S	$UTLIB	Dep.	Returns ASCII left-justified, space-filled name of routine that called FLOWENTR or FLOWEXIT

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
GETPOS								
	GETPOS	GETPOS			SF	$IOLIB	Dep.	Gets dataset position
	GPOS		$GPOS			$SYSLIB	Dep.	
	GTPOS		GTPOS%			$SYSLIB	Dep.	Gets tape dataset position
GETREGS								
	GNAMEQ	GETREGS			S	$UTLIB	Ind.	Returns register usage statistics for FLOWENTR
GPARAM								
	GPARAM	GETPARAM		$GP	S	$SYSLIB	Dep.	All three routines transfer control statement parameter values to an array provided by the calling routine
				$GPARAM		$SYSLIB	Dep.	
				$PAL		$SYSLIB	Dep.	

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type: CFT Standard: Address	Entry Type: CFT Standard: Value	Entry Type: Call From CAL Only	CFT Call Type	Library	OS Dep	Purpose
HEAP MANAGER	HPALLOC	HPALLOC		ALLOC%	S	$UTLIB	Dep.	Allocates a block of memory from the heap
	HPCHECK	HPCHECK		HCHECK%	S	$UTLIB	Dep.	Checks integrity of the heap
	HPCLMOVE	HPCLMOVE		CLMOVE%	S	$UTLIB	Dep.	Changes length of a heap block, and moves block if it cannot be extended in place
	HPDEALLC	HPDEALLC		DEALLC%	S	$UTLIB	Dep.	Returns a block of memory to the heap
	HPDUMP	HPDUMP		HPDUMP%	S	$UTLIB	Dep.	Writes information about the heap to a dataset
	HPGROW			HPGROW%		$UTLIB	Dep.	Heap expansion routine
	HPLEN	IHPLEN		HPLEN%	SF	$UTLIB	Dep.	Returns length of a heap block
	HPMEM			HPMEM%		$UTLIB	Dep.	Memory request routine
	HPMERGE			HMERGE%		$UTLIB	Dep.	Heap merge routine
	HPNEWLEN	HPNEWLEN		NEWLEN%	S	$UTLIB	Dep.	Changes length of a heap block
	HPSHRINK	HPSHRINK		SHRINK%	S	$UTLIB	Dep.	Returns memory from heap to operating system
	HPSTAT	IHPSTAT		HPSTAT%	SF	$UTLIB	Dep.	Returns information about the heap

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
I00DEL	I00DEL	I00DEL			SF	$UTLIB	Ind.	Deletes SKOL string or substring and returns the length of resulting string
I00ERR	I00ERR	I00ERR			S	$UTLIB	Ind.	Handles run time errors in SKOL programs
I00MVC	I00MVC	I00MVC			SF	$UTLIB	Ind.	Replaces SKOL string or substring with simple character and returns the length of resulting string
I00MVM	I00MVM	I00MVM			SF	$UTLIB	Ind.	Replaces SKOL string or substring with another SKOL string or substring and returns the length of the resulting string

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
I00ORD	I00ORD	I00ORD			SF	$UTLIB	Ind.	Returns internal SKOL code for a given ASCII character
I00READ	I00READ	I00READ			S	$UTLIB	Ind.	Reads a logical record in Al format and converts each word containing an ASCII character, left-justified, space-filled, to its internal SKOL code
I00SETUP	I00SETUP	I00SETUP			S	$UTLIB	Ind.	Initializes a SKOL program's table for direct translation of ASCII character codes to internal ordinal numbers

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type: CFT Standard: Address	Entry Type: CFT Standard: Value	Entry Type: Call From CAL Only	CFT Call Type	Library	OS Dep	Purpose
I0OWRITE	I0OWRITE	I0OWRITE			S	$UTLIB	Ind.	Writes characters defined by TYPE CHAR statement; converts character ordinals to ASCII characters in (Al) format, left-justified, space-filled, and then writes them as a logical record.
IABS	INTEGER	IABS			SF	$ARLIB	Ind.	Computes integer absolute value
IBM	IBMI			$IBMI		$IOLIB	Dep.	Translates input IBM formatted data
	IBMO			$IBMO		$IOLIB	Dep.	Translates output IBM formatted data
ICAMAX	ICAMAX	ICAMAX			SF	$SCILIB	Ind.	Finds the first index of the maximum absolute value of a complex array

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ICEIL	ICEIL	ICEIL			SF	$UTLIB	Ind.	Returns integer ceiling of a rational number represented as two integer parameters
IDIM	INTEGER	IDIM			SF	$ARLIB	Ind.	Computes positive integer difference
IDNINT	IDNINT	IDNINT	IDNINT% %IDNINT	 %IDNINT%	SF VF	$ARLIB $ARLIB	Ind. Ind.	Finds nearest integer to double-precision number
IFDNT	IFDNT	IFDNT			SF	$SYSLIB	Dep.	Determines if a dataset is local to the job
IGTBYT	IGTBYT	IGTBYT			SF	$UTLIB	Ind.	Fetches bytes

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
IILZ	IILZ	IILZ				$SCILIB	Ind.	Returns the number of zero values before the first nonzero value
IIN	NCON			$IIN		$UTLIB	Ind.	Contains integer powers of 10 in range of 10^0 to 10^{18}
ILLZ	ILLZ	ILLZ			SF	$SCILIB	Ind.	Returns the number of values that do not have the first bit set before the first value that does have the first bit set
ILSUM	ILSUM	ILSUM			SF	$SCILIB	Ind.	Returns total number of true values in array declared LOGICAL

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
IMX	NCON			$IMX		$UTLIB	Ind.	Contains double-precision floating-point representation of negative powers of 10 in range of 10^0 to 10^{-4096}
INQUIRE	INQUIRE	INQUIRE		$INQ	SA	$IOLIB	Dep.	Returns the status of a unit or a dataset
INSASCI	INSASCI		INSASCI%		S	$SYSLIB	Ind.	Inserts ASCII parameters into a message
INT	REAL	INT			SF	$ARLIB	Ind.	Truncates to integer value
INT	TM			$INT		$SYSLIB	Dep.	Initializes table pointers

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
INT6064	INT6064	INT6064			S	$UTLIB	Ind.	Converts CDC integer to Cray integer
INT6460	INT6460	INT6460			S	$UTLIB	Ind.	Converts Cray integer to CDC integer
IOERP	IOERP			IOERP% NLERP%		$IOLIB $IOLIB	Dep. Dep.	Processes I/O errors Processes NAMELIST errors
IOSTAT	IOSTAT	IOSTAT			SF	$IOLIB	Dep.	Returns EOF status; clears UEOF in DSP.
IPX	NCON			$IPX		$UTLIB	Ind.	Contains double-precision floating-point representation of positive powers of 10 in range of 10^0 to 10^{-4096}

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ISAMAX	ISAMAX	ISAMAX			SF	$SCILIB	Ind.	Finds the first index of the largest absolute value in a real array
ISAMIN	ISAMIN	ISAMIN			SF	$SCILIB	Ind.	Finds first index of the smallest absolute value in a real vector
ISIGN	INTEGER	ISIGN			SF	$ARLIB	Ind.	Transfers sign from one integer to another
ISRCH		ISEARCH			SF	$SCILIB	Ind.	Returns the first location of a true condition
	ISRCHEQ	ISRCHEQ						
	ISRCHNE	ISRCHNE						
	ISRCHFLT	ISRCHFLT						
	ISRCHFLE	ISRCHFLE						
	ISRCHFGT	ISRCHFGT						
	ISRCHFGE	ISRCHFGE						
	ISRCHILT	ISRCHILT						

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
ISRCH continued	ISRCHILE	ISRCHILE						
	ISRCHIGT	ISRCHIGT						
	ISRCHIGE	ISRCHIGE						
ISMAX	ISMAX	ISMAX			SF	$SCILIB	Ind.	Finds the first index of the maximum of an array
ISMIN	ISMIN	ISMIN			SF	$SCILIB	Ind.	Finds the first index of the minimum of an array
ITOI	ITOISS		ITOI%		SF	$ARLIB	Ind.	Raises integer scalar to integer scalar power
	ITOIVS		%ITOI%		VF	$ARLIB	Ind.	Raises integer vector to integer scalar power
	ITOIVV		ITO%I%		VF	$ARLIB	Ind.	Raises integer scalar to integer vector power
			%ITO%I%		VF	$ARLIB	Ind.	Raises integer vector to integer vector power

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
JNAME	JNAME	JNAME			SF	$SYSLIB	Dep.	Returns job name
KOMSTR	KOMSTR	KOMSTR			SF	$UTLIB	Ind.	Compares bytes
LEADZERO	BOOLEAN	LEADZ			SF	$ARLIB	Ind.	Tallies number of leading zero bits
LENGTH	LENGTH	LENGTH			SF	$IOLIB	Dep.	Returns number of words processed in last BUFFER I/O operation
LINPACK					S	$SCILIB	Ind.	See table 4-3.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type: CFT Standard: Address	Entry Type: CFT Standard: Value	Entry Type: Call From CAL Only	CFT Call Type	Library	OS Dep	Purpose
LGO	LGO	LGO		$LGO	S	$SYSLIB	Dep.	Loads an absolute program from a local dataset containing the binary image as the first record
LOADF	LOADF			$STOREF $LOADI $LOADR $LOADL $LOADD $LOADC		$UTLIB	Ind.	Performs run time array bounds checking (Performed by all LOADF routines)
LOC	LOC	LOC			SF	$FTLIB	Ind.	Returns first word address of argument
LOCKASGN	LOCKASGN	LOCKASGN			S	$UTLIB	Ind.	Identifies lock

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
LOCKOFF	LOCKOFF	LOCKOFF			S	$UTLIB	Ind.	Clears lock
LOCKON	LOCKON	LOCKON			S	$UTLIB	Ind.	Sets lock
LOCKREL	LOCKREL	LOCKREL			S	$UTLIB	Ind.	Releases lock
LOCKTEST	LOCKTEST	LOCKTEST			S	$UTLIB	Ind.	Tests lock
LOG	ALOG	ALOG	ALOG%		SF	$ARLIB	Ind.	Computes natural logarithm
	ALOGV		%ALOG%	%ALOG	VF	$ARLIB	Ind.	Computes vector natural logarithm
	ALOG10	ALOG10	ALOG10%		SF	$ARLIB	Ind.	Computes common logarithm
	ALOG10V		%ALOG10%	%ALOG10	VF	$ARLIB	Ind.	Computes vector common logarithm
	CLOG	CLOG	CLOG%		SF	$ARLIB	Ind.	Computes complex logarithm
	CLOGV		%CLOG%	%CLOG	VF	$ARLIB	Ind.	Computes vector complex logarithm

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
LOG continued	DLOG	DLOG	DLOG%		SF	$ARLIB	Ind.	Computes double-precision natural logarithm
	DLOGV		%DLOG%	%DLOG	VF	$ARLIB	Ind.	Computes vector double-precision natural logarithm
	DLOG	DLOG10	DLOG10%		SF	$ARLIB	Ind.	Computes double-precision common logarithm
	DLOGV		%DLOG10%	%DLOG10	VF	$ARLIB	Ind.	Computes vector double-precision common logarithm
LOGECHO	LOGECHO	LOGECHO			S	$UTLIB	Dep.	Writes last line formatted by $WFD as a message to $LOG file
MASK	BOOLEAN	MASK			SF	$ARLIB	Ind.	Forms ones mask from left of argument bits if $0 \le arg \le 63$; forms ones mask from right of (128-argument) bits if $64 \le arg \le 128$.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
MEM	TM			$MEM		$SYSLIB	Dep.	Requests memory
MEMORY	MEM	MEMORY			S	$SYSLIB	Dep.	Determines or changes the amount of memory assigned to a job
MINV	MINV	MINV			S	$SCILIB	Ind.	Solves linear equations, using a partial pivot search and Gauss-Jordan reduction
MOD	DMOD	DMOD	DMOD%		SF	$ARLIB	Ind.	Performs double-precision modulo arithmetic
	DMODV		%DMOD%	%DMOD	VF	$ARLIB	Ind.	Performs vectorized double-precision modulo arithmetic

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
MOD continued	LDIVV		LDSV%		VF	$ARLIB	Ind.	Performs 64-bit integer division, scalar/vector
			LDVS%		VF	$ARLIB	Ind.	Performs 64-bit integer division, vector/scalar
			LDVV%		VF	$ARLIB	Ind.	Performs 64-bit integer division, vector/vector
	LDMOD		LDSS%		SF	$ARLIB	Ind.	Performs 64-bit integer division, scalar/scalar
		MOD	MOD%		SF	$ARLIB	Ind.	Performs 64-bit modulo arithmetic on two integer scalars
			MODSS%		SF	$ARLIB	Ind.	Same as MOD
	MODVV		%MOD%		VF	$ARLIB	Ind.	Performs 64-bit modulo arithmetic on two integer vectors
			MODVV%		VF	$ARLIB	Ind.	Same as %MOD%
			MODSV%		VF	$ARLIB	Ind.	Performs 64-bit modulo arithmetic on integer scalar and integer vector
			MODVS%		VF	$ARLIB	Ind.	Performs 64-bit modulo arithmetic on integer vector and integer scalar

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
MODIFY	MFY	MODIFY			S	$SYSLIB	Dep.	Changes permanent dataset characteristics
MOVBIT	MOVBIT	MOVBIT			S	$UTLIB	Ind.	Moves bits
MSC	TM			$MSC		$SYSLIB	Dep.	Searches table with mask
MSIO	RAIO	OPENMS WRITMS READMS CLOSMS STINDX FINDMS			S S S S S S	$IOLIB $IOLIB $IOLIB $IOLIB $IOLIB $IOLIB	Dep. Dep. Dep. Dep. Dep. Dep.	Allows user to specify how records of a dataset are to be changed without limitations of sequential access
MVC	MVC	MVC			S	$UTLIB	Ind.	Moves characters

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
MVE	TM			$MVE		$SYSLIB	Dep.	Moves memory words to table
MXM	MXM	MXM			S	$SCILIB	Ind.	Performs matrix multiply with fixed row and column spacing
MXMA	MXMA	MXMA			S	$SCILIB	Ind.	Performs matrix multiply with arbitrary row and column spacing
MXV	MXV	MXV			S	$SCILIB	Ind.	Performs matrix-vector multiply with fixed row and column spacing
MXVA	MXVA	MXVA			S	$SCILIB	Ind.	Performs matrix-vector multiply with arbrtrary row and column spacing

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
NACSED	PDDED	NACSED			SF	$SYSLIB	Dep.	Returns edition number of last dataset that was accessed, acquired, or saved
NAMELIST	RNL			$RNL		$IOLIB	Ind.	Reads NAMELIST input
		RNLSKIP			S	$IOLIB	Ind.	Determines action for wrong NAMELIST group
		RNLTYPE			S	$IOLIB	Ind.	Determines action for NAMELIST type mismatch across equal sign
		RNLECHO			S	$IOLIB	Ind.	Specifies unit for NAMELIST error messages and input echo
		RNLFLAG			S	$IOLIB	Ind.	Adds or removes NAMELIST echo-initiating characters
		RNLDELM			S	$IOLIB	Ind.	Adds or removes NAMELIST group name delimiting character
		RNLSEP			S	$IOLIB	Ind.	Adds or removes NAMELIST separator character
		RNLREP			S	$IOLIB	Ind.	Adds or removes NAMELIST replacement character
		RNLCOMM			S	$IOLIB	Ind.	Adds or removes NAMELIST trailing comment characters

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type				Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only	CFT Call Type			
		Address	Value					
NAMELIST continued	WNL			$WNL		$IOLIB	Ind.	Writes NAMELIST output
		WNLLONG			S	$IOLIB	Ind.	Specifies NAMELIST output line length
		WNLDELM			S	$IOLIB	Ind.	Specifies NAMELIST delimiting character
		WNLSEP			S	$IOLIB	Ind.	Specifies NAMELIST separator character
		WNLREP			S	$IOLIB	Ind.	Specifies NAMELIST replacement character
		WNLFLAG			S	$IOLIB	Ind.	Specifies first character of first line
NEQV	BOOLEAN	NEQV			SF	$ARLIB	Ind.	Computes logical difference
NICV	NICV			NICV% $NICV		$UTLIB $UTLIB	Ind. Ind.	Converts numeric input
	NICONV	NICONV			S	$UTLIB	Ind.	

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
NINT	INTEGER	NINT			SA	$ARLIB	Ind.	Calculates nearest integer
NOCV	NOCV			NOCV%		$UTLIB	Ind.	Converts numeric output
				$NOCV		$UTLIB	Ind.	
	NOCONV	NOCONV			S	$UTLIB	Ind.	
NORERUN	RERUN	NORERUN			S	$SYSLIB	Dep.	Controls monitoring of conditions causing job to be flagged as not rerunnable
NUMBLKS	NUMBLKS	NUMBLKS			S	$SYSLIB	Dep.	Returns current size of dataset in 512-word blocks

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
OPEN	SYMDBC	OPEN			S	$SYSLIB	Dep.	Opens a random, unblocked dataset
	OPEN			$OPN	SA	$IOLIB	Dep.	Connects existing dataset to unit, creates preconnected dataset, creates dataset and connects it to unit, or changes certain specifiers of connection between dataset and unit. (Implements FORTRAN OPEN statement.)
OR	BOOLEAN	OR			SF	$ARLIB	Ind.	Forms logical sum
ORDERS	ORDERS	ORDERS			S	$SCILIB	Ind.	Internally sorts fixed-length records

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
OTB	OTB	OTB		$OTB	SF	$UTLIB	Ind.	Converts octal ASCII number to binary
OVERLAY	OVERLAY	OVERLAY			S	$UTLIB	Dep.	Processes overlays
P32	P32	P32			S	$UTLIB	Ind.	Packs 32-bit words
P6460	P6460	P6460			S	$UTLIB	Ind.	Packs 60-bit words
PACK	PACK	PACK			S	$SYSLIB	Ind.	Packs a specified number of words into a packed list
PAUSE	STOP			$PAUSE		$UTLIB	Ind.	Suspends program execution or terminates job step; installation dependent.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
PBN	PBN			$APBN $PBN		$SYSLIB $SYSLIB	Dep. Dep.	Asynchronously postions dataset Synchronously positions dataset
PERF	PERF	PERF			S	$SYSLIB	Dep.	Hardware performance monitor
PERMIT	PER	PERMIT			S	$SYSLIB	Dep.	Allows the owner of a permanent dataset to control the manner in which other users can use the dataset
PPL	PPL	PPL			S	$SYSLIB	Dep.	Processes keywords for a given directive

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
POPCNT	BOOLEAN	POPCNT			SF	$ARLIB	Ind.	Tallies number of bits set in a word
POPPAR	BOOLEAN	POPPAR			SF	$ARLIB	Ind.	Parity of number of bits set in a word
PRCW	PRCW			$PRCW		$SYSLIB	Dep.	Positions dataset after an RCW
PTS	TM			$PTS%		$SYSLIB	Dep.	Presets table space
PUTBYT	PUTBYT	PUTBYT			S	$UTLIB	Ind.	Stores byte

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RANDOM								
	RANF	RANF	RANF%		SF	$ARLIB	Ind.	Returns a random number
	RANFV		%RANF%	%RANF	VF	$ARLIB	Ind.	Returns a vector of random numbers
	RANGET	RANGET	RANGET%		SF	$ARLIB	Ind.	Returns current seed of the random number generator
	RANSET	RANSET	RANSET%		S	$ARLIB	Ind.	Sets random seed
RANFI	RANSED			RANFI		$ARLIB	Ind.	Contains current index to seed buffer for random number generator
RANFS	RANSED			RANFS		$ARLIB	Ind.	Contains 128 random number seeds in a buffer
RB	RB			$RB		$IOLIB	Dep.	Initiates buffered input

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RBN	RBN	RBN		$RBN	SF	$SYSLIB	Ind.	Replaces trailing blanks with nulls
RCW	RCW			$RCHP $RCHR $RCWP $RCWR		$SYSLIB $SYSLIB $SYSLIB $SYSLIB	Dep. Dep. Dep. Dep.	Reads characters in partial record mode Reads characters in record mode Reads words in partial record mode Reads words in record mode
RDIN	SYMDBC	RDIN			S	$SYSLIB	Dep.	Reads a buffer of data from a random, unblocked dataset
RDYQUE	RDYQUE			$RDYQUE%		$UTLIB	Ind.	Readies a queue of tasks for execution
RDYTSK	RDYTSK			$RDYTSK%		$UTLIB	Ind.	Readies a task for execution

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
READ	READ	READ			S	$IOLIB	Ind.	Reads words, full record mode
READC	READC	READC			S	$IOLIB	Ind.	Reads characters, full record mode
READCP	READCP	READCP			S	$IOLIB	Ind.	Reads characters, partial record mode
READIBM	READIBM	READIBM			S	$IOLIB	Ind.	Reads two IBM 32-bit floating-point words from each Cray 64-bit word
READP	READP	READP			S	$IOLIB	Ind.	Reads words, partial record mode

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
REAL	COMPLX	REAL			SF	$ARLIB	Ind.	Returns real part of a complex number
RELEASE	RLS	RELEASE			S	$SYSLIB	Dep.	Releases a dataset
REMARK	REMARK	REMARK			S	$UTLIB	Dep.	Enters message in logfile
		REMARK2			S	$UTLIB	Dep.	Enters message and message ID in logfile
		REMARKF						
REPRIEVE	RPV	SETRPV		$SETRPV	S	$SYSLIB	Dep.	Transfers control to a specified routine upon encountering a user-selected reprievable error condition
	ENDRPV	ENDRPV		$ENDRPV	S	$SYSLIB	Dep.	Continues job step termination processing or clears an existing reprieve environment

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RERUN	RERUN	RERUN			S	$SYSLIB	Dep.	Allows user to set job rerunnability status
REWD	WCW			$REWD		$SYSLIB	Dep.	Rewinds dataset
RFD	RFD			$RFI		$IOLIB	Dep.	Initializes for formatted read
				$RFV $RFV%		$IOLIB	Dep.	Cracks format; reads input.
				$RFA		$IOLIB	Dep.	Vectorizes formatted read
				$RFF		$IOLIB	Dep.	Terminates formatted read
				$RCHK		$IOLIB	Dep.	Read formatted, check;
				$RNOCHK		$IOLIB	Dep.	Read formatted, no check
RLB	RLB			$RLB		$SYSLIB	Dep.	Reads data directly from user area

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RLD	RLD			$RLI $RLA $RLF		$IOLIB $IOLIB $IOLIB	Dep. Dep. Dep.	Reads list-directed input
RNB	RNB	RNB		$RNB	SF	$SYSLIB	Ind.	Replaces trailing nulls with blanks
ROT	CROT	CROT			S	$SCILIB	Ind.	Applies complex Givens plane rotation
	CROTG	CROTG			S	$SCILIB	Ind.	Constructs complex Givens plane rotation
	SROT	SROT			S	$SCILIB	Ind.	Performs Givens plane rotation
	SROTG	SROTG			S	$SCILIB	Ind.	Constructs Givens plane rotation
	SROTM	SROTM			S	$SCILIB	Ind.	Performs modified Givens plane rotation
	SROTMG	SROTMG			S	$SCILIB	Ind.	Calculates a rotation matrix as parameters for modified Givens plane rotation SROTM

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RTOI								
	RTOISS		RTOI%		SF	$ARLIB	Ind.	Raises real scalar to integer scalar power
	RTOIVS		%RTOI%		VF	$ARLIB	Ind.	Raises real vector to integer scalar power
	RTOIVV		RTO%I%		VF	$ARLIB	Ind.	Raises real scalar to integer vector power
			%RTO%I%		VF	$ARLIB	Ind.	Raises real vector to integer vector power
RTOR								
	RTORSS		RTOR%		SF	$ARLIB	Ind.	Raises real scalar to real scalar power
	RTORSV		RTO%R%		VF	$ARLIB	Ind.	Raises real scalar to real vector power
	RTORVS		%RTOR%		VF	$ARLIB	Ind.	Raises real vector to real scalar power
	RTORVV		%RRO%R%		VF	$ARLIB	Ind.	Raises real vector to real vector power

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
RU								
	RU			$RUI		$IOLIB	Dep.	Initializes for unformatted read
				$RUV $RUV%		$IOLIB	Dep.	Reads unformatted input
				$RUA %$RUV%		$IOLIB	Dep.	Reads unformatted input into a vector
				$RUF		$IOLIB	Dep.	Terminates unformatted read
RUT								
	RUT			RUTI%		$IOLIB	Dep.	Initializes foreign dataset read
				RUTD%		$IOLIB	Dep.	Reads foreign dataset, full record mode
				RUTDP%		$IOLIB	Dep.	Reads foreign dataset, partial record mode
				RUTF%		$IOLIB	Dep.	Terminates foreign dataset read
				FDGPOS%		$IOLIB	Dep.	Gets foreign dataset position
				FDSPOS%		$IOLIB	Dep.	Sets foreign dataset position
				FDBKSP%		$IOLIB	Dep.	Backspaces foreign dataset

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SAVE	SVS	SAVE			S	$SYSLIB	Dep.	Makes a dataset permanent
SCAL	CSCAL	CSCAL			S	$SCILIB	Ind.	Scales a complex array by a complex factor
	CSSCAL	CSSCAL			S	$SCILIB	Ind.	Scales a complex array by a real factor
	SSCAL	SSCAL			S	$SCILIB	Ind.	Scales a real array by a real factor
SCATTER	SCATTER	SCATTER			S	$SCILIB	Ind.	Scatters a vector into another vector
SCERP	SCERP	SCERP		$SCERP	S	$SCILIB	Ind.	Processes $SCILIB errors; issues logfile error message, then aborts with traceback.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SCHED	SCHED			$SCHED%		$UTLIB	Dep.	Schedules logical CPUs for user tasks
SCNRM2	SCNRM2	SCNRM2			SF	$SCILIB	Ind.	Calculates the Euclidean norm (l_2) of a complex array
SDACCESS	SDACC	SDACCESS			S	$SYSLIB	Dep.	Allows a FORTRAN program to access system datasets
SDSP	SDSP			$SDSP		$SYSLIB	Dep.	Searches Dataset Parameter Tables for a dataset name and returns DSP address
SEARCH	OSRCHI OSRCHF	OSRCHI OSRCHF			S S	$SCILIB $SCILIB	Ind. Ind.	Searches an ordered array for a target

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SECOND	SECOND	SECOND			S/SF	$UTLIB	Ind.	Returns time since start of job in floating-point seconds
SEGRES	SEGRES			$SEGRES $SEGCALL		$UTLIB	Dep.	Initializes execution of a segmented program and services intersegment subroutine calls
SENSEBT	BTMODE	SENSEBT			S	$UTLIB	Dep.	Returns mode indicating bidirectional memory transfers are enabled or disabled
SENSEFI	FIMODE	SENSEFI			S	$UTLIB	Dep.	Returns mode indicating floating-point interrupts permitted or prohibited

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SETBT	BTMODE	SETBT			S	$UTLIB	Dep.	Temporarily enables bidirectional memory transfers.
SETBTS	BTMODE	SETBTS			S	$UTLIB	Dep.	Permanently enables bidirectional memory transfers
SETFI	FIMODE	SETFI			S	$UTLIB	Dep.	Temporarily permits floating-point interrupts
SETFIS	FIMODE	SETFIS			S	$UTLIB	Ind.	Enables floating-point interrupts until explicitly disabled
SETPLIMQ	FLOW	SETPLIMQ			S	$UTLIB	Ind.	Processes CFT flowtrace option

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SETPOS								
	SETPOS	SETPOS			S	$IOLIB	Dep.	Set dataset position
	SPOS					$SYSLIB	Dep.	
	STPOS					$SYSLIB	Dep.	Set tape dataset position
	POS			$ASPOS		$SYSLIB	Dep.	Asynchronously positions current dataset
				$FSPOS		$SYSLIB	Dep.	Finishes the asynchronous dataset positioning request
				$GPOS		$SYSLIB	Dep.	Returns current dataset position
				$SPOS		$SYSLIB	Dep.	Positions dataset to a word address
	STPOS			STPOS%		$SYSLIB	Dep.	Positions tape dataset
	SETPOS	GETPOS			SF	$IOLIB	Dep.	Returns current dataset position
		SETPOS			S	$IOLIB	Dep.	Positions dataset
SFN								
	RNB	RNB		$RNB SFN	SF	$SYSLIB	Ind.	Replaces trailing nulls with blanks
SHIFT								
	BOOLEAN	SHIFT			SF	$ARLIB	Ind.	Shifts left circularly

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SHIFTL	BOOLEAN	SHIFTL			SF	$ARLIB	Ind.	Shifts left; zero fill.
SHIFTR	BOOLEAN	SHIFTR			SF	$ARLIB	Ind.	Shifts right; zero fill.
SIGN	REAL	SIGN			SF	$ARLIB	Ind.	Transfers sign from one real number to another
SIN	COS COSV	SIN	SIN% %SIN%	 %SIN	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes sine Computes vectorized sine
SINH	COSH COSHV	SINH	SINH% %SINH%	 %SINH	SF VF	$ARLIB $ARLIB	Ind. Ind.	Computes hyperbolic sine Computes vectorized hyperbolic sine

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SKIP	SKIP	SKIPR			S	$IOLIB	Dep.	Bypasses a specified number of records in a dataset
		SKIPF			S	$IOLIB	Dep.	Bypasses a specified number of files in a dataset
		SKIPD			S	$IOLIB	Dep.	Positions a dataset at end of data
SKIPBAD	RCVRBAD	SKIPBAD			S	$SYSLIB	Dep.	Skips to the first good data encountered
SKIPU	SKIPU	SKIPU			S	$IOLIB	Dep.	Skips sectors on unblocked dataset
SLERP	SLERP			SLERP%		$SYSLIB	Ind.	Processes $SYSLIB errors; aborts with traceback.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SLFT	SDSP			$SLFT		$SYSLIB	Dep.	Searches Logical File Table for dataset name and returns LFT address
SMACH	SMACH	SMACH			SF	$SCILIB	Ind.	Real function of an integer argument that returns Cray machine constants
	CMACH	CMACH			SF	$SCLIB	Ind.	Computes complex Cray constants
SNAP	SNAP	SNAP			S	$SYSLIB	Dep.	Prints current register contents on $OUT
SNRM2	SNRM2	SNRM2			SF	$SCILIB	Ind.	Calculates Euclidean norm (l_2) of a real array
SOLR	SOLR	SOLR			S	$SCILIB	Ind.	Solves second-order linear recurrences

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SOLRN	SOLRN	SOLRN			S	$SCILIB	Ind.	Solves for only the last term of a second-order linear recurrence
SOLR3	SOLR3	SOLR3			S	$SCILIB	Ind.	Computes second-order linear recurrence of three terms
SQRT	CSQRT	CSQRT	CSQRT%		SF	$ARLIB	Ind.	Computes complex square root
	CSQRTV		%CSQRT%	%CSQRT	VF	$ARLIB	Ind.	Computes vectorized complex square root
	DSQRT	DSQRT	DSQRT%		SF	$ARLIB	Ind.	Computes double-precision square root
	DSQRTV		%DSQRT%	%DSQRT	VF	$ARLIB	Ind.	Computes vectorized double-precision square root
	SQRT	SQRT	SQRT%		SF	$ARLIB	Ind.	Calculates square root
	SQRTV		%SQRT%	%SQRT	VF	$ARLIB	Ind.	Calculates vectorized square root

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SRC	TM			$SRC		$SYSLIB	Dep.	Searches table for a specific value
SSWITCH	SSWITCH	SSWITCH			S/SF	$UTLIB	Dep.	Tests pseudo sense switch
STACK	STACKAL			$STKOFEN		$UTLIB	Dep.	Handles overflow of a stack during a subprogram entry
				$STKCR		$UTLIB	Dep.	Creates a stack
				$STKUFEX		$UTLIB	Dep.	Handles stack segment underflow in a subprogram exit
	STACKDE			$STKDE		$UTLIB	Dep.	Deletes a stack
				$STKUFCK		$UTLIB	Dep.	Checks for stack segment underflow in a subprogram exit
STOP	STOP			$STOP		$UTLIB	Ind.	Terminates current job step and advances job to next job step

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
STRMOV	STRMOV	STRMOV			S	$UTLIB	Ind.	Moves bytes
SUBMIT	SUBMIT	SUBMIT		$SUBMIT	S	$SYSLIB	Dep.	Places a job dataset into the COS input queue
SUM	CSUM	CSUM			SF	$SCILIB	Ind.	Sums the elements of a complex array
	SASUM	SASUM			SF	$SCILIB	Ind.	Sums the absolute values of a real array
	SCASUM	SCASUM			SF	$SCILIB	Ind.	Sums the absolute values of real and imaginary parts of a complex array
	SSUM	SSUM			SF	$SCILIB	Ind.	Sums the elements of a real array
SUSTSK	SUSTSK			$SUSTSK%		$UTLIB	Ind.	Suspends execution of the calling task

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
SWAP	CSWAP	CSWAP			S	$SCILIB	Ind.	Exchanges specified elements of complex arrays
	SSWAP	SSWAP			S	$SCILIB	Ind.	Exchanges specified elements of two real arrays
SYMDEBUG	SYMDEBUG	SYMDEBUG DEADBUG			S	$UTLIB	Dep.	Produces a symbolic dump
SYNCH	SYNCH		SYNCH%			$SYSLIB	Dep.	Synchronizes tape dataset
TABLE MANAGER	TM	TMADW			SF	$UTLIB	Dep.	Adds a word to a table
	TM	TMAMU			S	$UTLIB	Dep.	Reports TMGR statistics
	TM	TMATS			SF	$UTLIB	Dep.	Allocates table space
	TM	TMINIT			S	$UTLIB	Dep.	Initializes managed tables

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
TABLE MANAGER continued	TM	TMMEM			S	$UTLIB	Dep.	Requests memory
	TM	TMMSC			SF	$UTLIB	Dep.	Searches table with mask
	TM	TMMVE			S	$UTLIB	Dep.	Moves words
	TM	TMPTS			S	$UTLIB	Dep.	Presets table space
	TM	TMSRC			SF	$UTLIB	Dep.	Searches table
	TM	TMVSC			SF	$UTLIB	Dep.	Searches vector table
TABLES	NCON			$IIN		$UTLIB	Ind.	Table of integer powers of ten
				$IMX		$UTLIB	Ind.	Table of negative real powers of ten
				$IPX		$UTLIB	Ind.	Table of positive real powers of ten
	PDD			PDD		$SYSLIB	Dep.	Table of permanent dataset definitions
	RANSED			RANFI RANFS		$ARLIB	Ind.	Table for random number generator
TADD	TPREC	TADD		TASS $TADD TASS%		$ARLIB	Ind.	Performs triple-precision addition

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
TAN								
	COT	TAN	TAN%		SF	$ARLIB	Ind.	Computes tangent
	COTV		%TAN%	%TAN	VF	$ARLIB	Ind.	Computes vectorized tangent
TANH								
	TANH	TANH	TANH%		SF	$ARLIB	Ind.	Calculates hyperbolic tangent
	TANHV		%TANH%	%TANH	VF	$ARLIB	Ind.	Calculates vectorized hyperbolic tangent
TDIV								
	TDSS	TDIV		TDSS TDSS% $TDIV		$ARLIB	Ind.	Performs triple-precision division
TIBCR								
	TIBCR			$TIBCR%		$UTLIB	Ind.	Builds task information block
TIBDE								
	TIBDE			$TIBDE%		$UTLIB	Ind.	Deletes task information block

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
TIME								
	DTTS	DTTS			S	$SYSLIB	Dep.	Converts from date and time to timestamp
	MTTS	MTTS			SF	$SYSLIB	Dep.	Converts from a real-time value to the corresponding timestamp value
	TSDT	TSDT			S	$SYSLIB	Dep.	Converts between timestamps and the date and time as ASCII strings
	TSMT	TSMT			SF	$SYSLIB	Dep.	Converts from a timestamp to the corresponding real-time clock value
	UNITTS	UNITTS			SF	$SYSLIB	Dep.	Returns the number of timestamp units in a specified number of standard time units
TMLT								
	TMLT	TMLT		TMSS TMSS% $TMLT		$ARLIB	Ind.	Performs triple-precision multiplication
TR								
	TR	TR			S	$UTLIB	Ind.	Translates characters

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
TRBK	TRBK	TRBK		$TRBK	S	$SYSLIB	Dep.	Prints a list of subroutines showing the path from the main program to the current subprogram
TRBKLVL	TRBKLV TRBKLV	TRBKLVL		 TRBKLVL%	S	$UTLIB $SYSLIB	Ind. Ind.	Aids the traceback mechanism by returning information for the current level
TREMAIN	TREMAIN	TREMAIN			S	$SYSLIB	Dep.	Returns time remaining for job execution in floating-point seconds
TSKSTART	TSKSTART	TSKSTART			S	$UTLIB	Ind.	Initiates a task
TSKTEST	TSKTEST	TSKTEST			S	$UTLIB	Ind.	Returns a value indicating whether the indicated task exists

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
TSKTUNE	TSKTUNE	TSKTUNE			S	$UTLIB	Ind.	Modifies scheduling parameters used by a multitasking subroutine
TSKVALUE	TSKVALUE	TSKVALUE			S	$UTLIB	Ind.	Retrieves user id specified in the task control array used to create the executing task
TSKWAIT	TSKWAIT	TSKWAIT			S	$UTLIB	Ind.	Waits for the indicated task to complete execution
TSUB	TSUB	TSUB		TSSS% TSSS $TSUB		$ARLIB	Ind.	Performs triple-precision subtraction
U32	U32	U32			S	$UTLIB	Ind.	Unpacks 32-bit words

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
U6064	U6064	U6064			S	$UTLIB	Ind.	Unpacks 60-bit words
UEOF	UEOFC UEOFK UEOFS			$UEOFCL $UEOFKIL $UEOFSET		$SYSLIB $SYSLIB $SYSLIB	Dep. Dep. Dep.	Sets the uncleared end-of-file flag in DSP
UNIT	UNIT	UNIT		UNITLB	SF	$IOLIB	Dep.	Waits for buffer I/O completion and returns status
UNPACK	UNPACK	UNPACK			S	$SYSLIB	Ind.	Expands full words of data into a larger number of partial words
USCCTC	USCCTC	USCCTC			S	$UTLIB	Ind.	Converts EBCDIC character to ASCII character

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
USCCTI	USCCTI	USCCTI			S	$UTLIB	Ind.	Converts ASCII character to EBCDIC character
USICTP	USICTP	USICTP			S	$UTLIB	Ind.	Converts integer to IBM packed decimal field
USDCTC	USDCTC	USDCTC			S	$UTLIB	Ind.	Converts IBM double-precision to Cray double-precision
USDCTI	USDCTI	USDCTI			S	$UTLIB	Ind.	Converts Cray single-precision to IBM double-precision
USICTC	USICTC	USICTC			S	$UTLIB	Ind.	Converts IBM integer to Cray integer

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
USICTI	USICTI	USICTI			S	$UTLIB	Ind.	Converts Cray integer to IBM integer
USLCTC	USLCTC	USLCTC			S	$UTLIB	Ind.	Converts IBM Logical to Cray Logical
USLCTI	USLCTI	USLCTI			S	$UTLIB	Ind.	Converts Cray logical to IBM logical
USPCTC	USPCTC	USPCTC			S	$UTLIB	Ind.	Converts IBM packed decimal field to integer
USSCTC	USSCTC	USSCTC			S	$UTLIB	Ind.	Converts IBM single-precision to Cray single-precision

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
USSCTI	USSCTI	USSCTI			S	$UTLIB	Ind.	Converts Cray single-precision to IBM single-precision
UTERP	UTERP			UTERP%		$UTLIB	Dep.	Processes $UTLIB errors
WAIO	WAIO	WOPEN			S	$IOLIB	Dep.	Opens a dataset on a disk and specifies dataset as word-addressable, random access
		PUTWA			S	$IOLIB	Dep.	Writes a number of words from memory to a word-addressable, random access dataset
		GETWA			S	$IOLIB	Dep.	Reads a number of words from a word-addressable, random access dataset
		WCLOSE			S	$IOLIB	Dep.	Closes a word-addressable, random access dataset
		SEEK			S	$IOLIB	Dep.	Allows user to asynchronously read data into specified dataset buffers

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
WB	WB	$WB			S	$IOLIB	Dep.	Initiates buffered output
WC	WCH			$WCHP		$SYSLIB	Dep.	Writes characters in partial record mode
	WCH			$WCHR		$SYSLIB	Dep.	Writes characters in record mode
WE	WWD			$WEOF		$SYSLIB	Dep.	Writes EOF and/or EOR, if necessary
	WWD			$WEOD		$SYSLIB	Dep.	Writes EOD; writes EOF and/or EOR, if necessary.
WFBUFFER	WFBUFFER			WFBUFFER		$IOLIB	Ind.	Loads a character from $WFD's buffer
WF	WFD			$WFI		$IOLIB	Dep.	Initializes for formatted output
				$WFV $WFV% %$WFV%		$IOLIB	Dep.	Cracks format; writes output.

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
WF continued				$WFA		$IOLIB	Dep.	Cracks format; writes a vector to output.
				$WFF		$IOLIB	Dep.	Finalizes write
				$WCHK		$IOLIB	Dep.	Write formatted, check;
				$WNOCHK		$IOLIB	Dep.	Write formatted, no check
WHEN								
	WHENEQ	WHENEQ			S	$SCILIB	Ind.	Returns all locations in an array that have a true relational value to the target.
	WHENNE	WHENNE						
	WHENFLT	WHENFLT						
	WHENFLE	WHENFLE						
	WHENFGT	WHENFGT						
	WHENFGE	WHENFGE						
	WHENEQ	WHENEQ						
	WHENNE	WHENNE						
	WHENILT	WHENILT						
	WHENILE	WHENILE						
	WHENIGT	WHENIGT						
	WHENIGE	WHENIGE						
WLB								
	WLB			$WLB		$SYSLIB	Dep.	Writes data directly into user area

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
WLD	WLD			$WLI $WLA $WLF		$IOLIB	Dep.	Writes list-directed data
WRITE	WRITE	WRITE			S	$IOLIB	Ind.	Writes words, full record mode
WRITEC	WRITEC	WRITEC			S	$IOLIB	Ind.	Writes characters, full record mode
WRITECP	WRITECP	WRITECP			S	$IOLIB	Ind.	Writes characters, partial record mode

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
WRITIBM	WRITIBM	WRITIBM			S	$IOLIB	Ind.	Writes two IBM 32-bit floating-point words from each Cray 64-bit word
WU	WU			$WUI		$IOLIB	Dep.	Initializes for unformatted write
				$WUV $WUV%		$IOLIB	Dep.	Writes unformatted output
				%$WUV%		$IOLIB	Dep.	Writes a vector to an output unit
				$WUA				
				$WUF		$IOLIB	Dep.	Finalizes write
WUT	WUT			WUTI%		$IOLIB	Dep.	Initializes foreign dataset write
				WUTD%		$IOLIB	Dep.	Writes foreign dataset, full record mode
				WUTDP%		$IOLIB	Dep.	Writes foreign dataset, partial record mode
				WUTF%		$IOLIB	Dep.	Terminates foreign dataset write
				FDWEOF%		$IOLIB	Dep.	Writes foreign dataset, end-of-file

Table 2-1. Subprogram summary (continued)

Primary Reference Name	UPDATE Deck Name	Entry Type			CFT Call Type	Library	OS Dep	Purpose
		CFT Standard		Call From CAL Only				
		Address	Value					
WW	WCW			$WWDP		$SYSLIB	Dep.	Writes words in partial mode
				$WWPU		$SYSLIB	Dep.	Writes words in partial record mode with unused bit count
				$WWDR		$SYSLIB	Dep.	Writes words in record mode
				$WWDS		$SYSLIB	Dep.	Writes words in record mode with unused bit count
XOR	BOOLEAN	XOR			SF	$ARLIB	Ind.	Forms logical difference
ZTS	TM			$ZTS%		$SYSLIB	Dep.	Clears table space

Table 2-2. Pascal subprogram summary

Primary Ref. and UPDATE Deck Name	Pascal Call Standard		Call from CAL only (X)	Dep.	Call Type	Purpose
	Address	Value				
P$$$HPAD	P$$$HPAD			Ind.	SF	Get address of heap control block
P$ABORT	P$ABORT			Ind.	S	Abort job step
P$CALLR	P$CALLR			Ind.	SF	Get name of calling routine
P$CBV	P$CBV			Ind.	S	Call a call-by-value routine
P$CONNEC	P$CONNEC			Ind.	S	Set file name
P$DATE	P$DATE			Ind.	S	Get date
P$DBP	P$BREAK	P$DBP	X	Ind.	S	Breakpoint checking
P$DISP	P$DISP			Ind.	S	Dispose heap area
P$DIVMOD		P$DIVMOD	X	Ind.	SF	Integer division
P$ENDP	P$ENDP			Ind.	S	End program
P$EOF	P$EOF			Ind.	SF	End of file check
P$EOLN	P$EOLN			Ind.	SF	End of line check
P$GET	P$GET			Ind.	S	Read record
P$HALT	P$HALT			Ind.	S	Terminate program
P$JTIME	P$JTIME			Ind.	SF	Get job CPU time
P$LOGMSG	P$LOGMSG			Ind.	S	Write $LOG message
P$LSTREW	P$LSTREW			Ind.	S	Rewrite file without rewind
P$MEMRY	P$MEMRY			Ind.	SF	Memory management
P$MOD		P$MOD	X	Ind.	SF	Integer modulus
P$NEW	P$NEW			Ind.	S	Allocate heap area
P$OSDBS	P$OSDBS			Dep.	S	Rewind dataset
P$OSDDT	P$OSDDT			Dep.	S	Get date
P$OSDEP	P$OSDEP			Dep.	S	Open dataset
P$OSDJT	P$OSDJT			Dep.	S	Get job CPU time
P$OSDLM	P$OSDLM			Dep.	S	Write $LOG message
P$OSDPR	P$OSDPR			Dep.	S	Set prompt string
P$OSDQI	P$OSDQI			Dep.	S	Query if dataset is interactive
P$OSDRC	P$OSDRC			Dep.	S	Read characters
P$OSDRP	P$OSDRP			Dep.	S	Enable reprieve
P$OSDRW	P$OSDRW			Dep.	S	Read words
P$OSDTM	P$OSDTM			Dep.	S	Get time of day
P$OSDWC	P$OSDWC			Dep.	S	Write characters
P$OSDWF	P$OSDWF			Dep.	S	Write EOF
P$OSDWR	P$OSDWR			Dep.	S	Write record
P$OSDXP	P$OSDXP			Dep.	S	Exit program

Table 2-2. Pascal subprogram summary (continued)

Primary Ref. and UPDATE Deck Name	Pascal Call Standard		Call from CAL only			
	Address	Value	(X)	Dep.	Call Type	Purpose
P$PAGE	P$PAGE			Ind.	S	Start new page
P$PUT	P$PUT			Ind.	S	Write record
P$RB	P$RB			Ind.	S	Read Boolean
P$RCH	P$RCH			Ind.	S	Read character
P$READ	P$READ			Ind.	S	Read record
P$READLN	P$READLN			Ind.	S	Read new line
P$REPRV	P$REPRV			Ind.	S	Reprieve processing
P$RESET	P$RESET			Ind.	S	Reset file
P$REWRIT	P$REWRIT			Ind.	S	Rewrite file
P$RF	P$RF			Ind.	S	Read floating point
P$RI	P$RI			Ind.	S	Read integer
P$ROUND		P$ROUND	X	Ind.	SF	ROUND function
P$RSTR	P$RSTR			Ind.	S	Read string
P$RTIME	P$RTIME			Ind.	S	Runtime timing
P$RTMSG	P$RTMSG	P$DEBUG		Ind.	S	Runtime messages
P$RUNTIM		P$RUNTIM	X	Ind.	S	Runtime initialization routine
P$SFRAME	P$SFRAME			Ind.	SF	Returns pointer to caller's stackframe
P$TIME	P$TIME			Ind.	S	Get time of day
P$TIMER	P$RTIME	P$TIMER	X	Ind.	S	Runtime timing
P$TRACE	P$TRACE			Ind.	S	Stack walkback
P$TRUNC		P$TRUNC	X	Ind.	SF	TRUNC function
P$WB	P$WB			Ind.	S	Write Boolean
P$WCH	P$WCH			Ind.	S	Write character
P$WEOF	P$WEOF			Ind.	S	Write end of file
P$WI	P$WI			Ind.	S	Write integer
P$WO	P$WO			Ind.	S	Write octal integer
P$WR	P$WR			Ind.	S	Write real number
P$WRITE	P$WRITE			Ind.	S	Write record
P$WRITLN	P$WRITLN			Ind.	S	Write end of line
P$WSTR	P$WSTR			Ind.	S	Write string

COMMON MATHEMATICAL SUBPROGRAMS

3

INTRODUCTION

This section lists the following categories of mathematical subprograms. (Algorithms and performance statistics are listed in Appendixes A and B.)

- Logarithmic
- Exponential
- Square root
- Trigonometric
- Hyperbolic
- Boolean
- Base value raised to a power
- Double- and triple-precision arithmetic
- Sixty-four bit integer division
- Character
- ASCII conversion
- Miscellaneous math
- Random number processing
- Math tables

The routines, whether presented in table form or in text form, list definition, argument and register information, and result type for each subprogram or subprogram group. In the routine definition, x and y indicate the first and second real arguments, respectively. Complex arguments are represented by z, which is x+iy. Argument and result types are represented by the following abbreviations.

R	Real
C	Complex
D	Double precision
CH	Character
I	Integer
B	Boolean
L	Logical
H	Hollerith

Each subprogram is listed in the tables with a code (CALL. SEQ. *code*) corresponding to one of the following calling sequences.

NAME - Scalar, call-by-address (SA)

Entry:

arg_1	Address of first argument
arg_2	Address of second argument (if present)

Exit:

(S1)	Result
(S2)	Second word of result; present if complex or double-precision result.

CAL usage:

```
CALL SQRT,(arg1)
```

FORTRAN usage: The CFT compiler does not use scalar call-by-address for these subprograms.

NAME% - Scalar, call-by-value (SV)

(a) One word per argument (SVa)

Entry:

(S1)	First argument
(S2)	Second argument (if present)

Exit:

(S1)	Result

(b) Two words per argument (SVb)

Entry:

(S1),(S2)	First argument
(S3),(S4)	Second argument (if present)

Exit:

(S1),(S2)	Result

(c) Three words per argument (SVc)

Entry:

(S1),(S2),(S3)	First argument
(S4),(S5),(S6)	Second argument

Exit:

(S1),(S2),(S3)	Result

CAL usage:

```
CALLV SQRT%
```

FORTRAN usage: B=SQRT(A)
The CFT compiler generates a scalar call-by-value call to SQRT%.

%NAME - Vector, call-by-address (VA)

Entry:

arg_1	First word address of first argument
arg_2	Address of first increment
arg_3	First word address of second argument (if present)
arg_4	Address of second increment (if present)
(VL)	Vector length

Exit:

(V1)	Result
(V2)	Second word of result; present if double-precision or complex result.

NOTE

For the vector call-by-address calling sequence, the arguments are taken from FWA, FWA+INCREMENT, FWA+2*INCREMENT,..., FWA+((VL-1)*INCREMENT.

CAL usage:

```
CALL %SQRT,(arg1,arg2)
```

FORTRAN usage: The CFT compiler does not use vector call-by-address for these functions.

%NAME% - Vector call-by-value (VV)

(a) One word per argument (VVa)

Entry:

(V1)	First argument
(V2)	Second argument (if present)
(VL)	Vector length

Exit:
  (V1)   Result

(b) Two words per argument (VVb)
Entry:
  (V1),(V2) First argument
  (V3),(V4) Second argument (if present)
Exit:
  (V1),(V2) Result

CAL usage:

```
CALLV %SQRT%
```

FORTRAN usage:

```
   DO 10 I=1,10
10 B(I)=SQRT(A(I))
```

The CFT compiler generates a vector call-by-value call to %SQRT%.

NOTE

The range of many functions is given as $|x|<\infty$. This range is interpreted as x representable on the Cray computer as a floating-point number; that is, $|x|<2^{8192}$ or approximately $|x|<10^{2466}$.

LOGARITHMIC ROUTINES

Table 3-1. Logarithmic routines

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Natural log.	ALOG ALOG% %ALOG %ALOG%	SA SVa VA VVa	$\log_e(x)$ or $\ln(x)$	1	R	$0<x<\infty$	R
Common log.	ALOG10 ALOG10% %ALOG10 %ALOG10%	SA SVa VA VVa	$\log_{10}(x)$	1	R	$0<x<\infty$	R
Complex log.	CLOG CLOG% %CLOG %CLOG%	SA SVb VA VVb	$\ln\|z\|+$ $i \arctan(y/x)$	1	C	$0<x<\infty$	C
Double-prec. Natural log.	DLOG DLOG% %DLOG %DLOG%	SA SVb VA VVb	$\log_e(x)$ or $\ln(x)$	1	D	$0<x<\infty$	D
Double-prec. Common log.	DLOG10 DLOG10% %DLOG10 %DLOG10%	SA SVb VA VVb	$\log_{10}(x)$	1	D	$0<x<\infty$	D

EXPONENTIAL ROUTINES

Table 3-2. Exponential routines

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Complex exponentiation	CEXP CEXP% %CEXP %CEXP%	SA SVb VA VVb	$e^x\cos(y)+ie^x\sin(y)$	1	C	$\lvert x\rvert<2^{13}\ln2$ $\lvert y\rvert<2^{24}$	C
Double-prec. exponentiation	DEXP DEXP% %DEXP %DEXP%	SA SVb VA VVb	e^x	1	D	$\lvert x\rvert<2^{13}\ln2$	D
Exponentiation	EXP EXP% %EXP %EXP%	SA SVa VA VVa	e^x	1	R	$\lvert x\rvert<2^{13}\ln2$	R

SQUARE ROOT ROUTINES

Table 3-3. Square root routines

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Complex square root	CSQRT CSQRT% %CSQRT %CSQRT%	SA SVb VA VVb	$\sqrt{1/2(\lvert z\rvert+x)}+$ $i\sqrt{1/2(\lvert z\rvert-x)}$	1	C	$0\le x,y<\infty$	C
Double-prec. square root	DSQRT DSQRT% %DSQRT %DSQRT%	SA SVb VA VVb	$\sqrt{x}$ or $x^{1/2}$	1	D	$0\le x<\infty$	D
Square root	SQRT SQRT% %SQRT %SQRT%	SA SVa VA VVa	$\sqrt{x}$ or $x^{1/2}$	1	R	$0\le x<\infty$	R

TRIGONOMETRIC ROUTINES

Table 3-4. Trigonometric routines

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Arccosine	ACOS ACOS% %ACOS %ACOS%	SA SVa VA VVa	arccos(x)	1	R	$\|x\| \leq 1$	R
Arcsine	ASIN ASIN% %ASIN %ASIN%	SA SVa VA VVa	arcsin(x)	1	R	$\|x\| \leq 1$	R
Arctangent	ATAN ATAN% %ATAN %ATAN%	SA SVa VA VVa	arctan(x)	1	R	$\|x\| < \infty$	R
Two-arg. arctangent	ATAN2 ATAN2% %ATAN2 %ATAN2%	SA SVa VAb VVa	arctan(x/y)	2	R	$\|x\|, \|y\| < \infty$ (x and y must not both be zero.)	R
Double-prec. arccosine	DACOS DACOS% %DACOS %DACOS%	SA SVb VA VVb	arccos(x)	1	D	$\|x\| \leq 1$	D
Double-prec. arcsine	DASIN DASIN% %DASIN %DASIN%	SA SVb VA VVb	arcsine(x)	1	D	$\|x\| \leq 1$	D

Table 3-4. Trigonometric routines (continued)

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Double-prec. arctangent	DATAN DATAN% %DATAN %DATAN%	SA SVb VA VVb	arctan(x)	1	D	$\|x\| < \infty$	D
Double-prec. two-arg. arctan.	DATAN2 DATAN2% %DATAN2 %DATAN2%	SA SVb VA VVb	arctan(x/y)	2	D	$\|x\|,\|y\| < \infty$ (x and y must not both be 0.)	D
Cosine	COS COS% %COS %COS%	SA SVa VA VVa	cos(x)	1	R	$\|x\|<2^{24}$	R
Complex cosine	CCOS CCOS% %CCOS %CCOS%	SA SVb VA VVb	cos(x)cosh(y)+ i sin(x)sinh(y)	1	C	$\|x\|<2^{24}$ $\|y\|<2^{13}\ln 2$	C
Double-prec. cosine	DCOS DCOS% %DCOS %DCOS	SA SVb VA VVb	cos(x)	1	D	$\|x\|<2^{48}$	D
Sine	SIN SIN %SIN %SIN%	SA SVa VA VVa	sin(x)	1	R	$\|x\|<2^{24}$	R
Complex sine	CSIN CSIN% %CSIN %CSIN%	SA SVb VA VVb	sin(x)cosh(y)+ i cos(x)sinh(y)	1	C	$\|x\|<2^{24}$ $\|y\|<2^{13}\ln 2$	C

Table 3-4. Trigonometric routines (continued)

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Double-prec. sine	DSIN DSIN% %DSIN %DSIN%	SA SVb VA VVb	sin(x)	1	D	$\|x\|<2^{48}$	D
Cosine and sine†	COSS COSS% %COSS %COSS%	SA SVb VA VVb	y_1=cos(x) y_2=sin(x)	1	R	$\|x\|<2^{24}$	R
Tangent	TAN TAN% %TAN %TAN%	SA SVa VA VVa	tan(x)	1	R	$\|x\|<2^{24}$	R
Double-prec. tangent	DTAN DTAN% %DTAN %DTAN%	SA SVb VA VVb	tan(x)	1	D	$\|x\|<2^{46}$ $\|x-n\|>0$, $\|n\|=1,3,5,...$	D
Cotangent	COT COT% %COT %COT%	SA SVa VA VVa	cot(x)	1	R	$\|x\|<2^{24}$	R
Double-prec. cotangent	DCOT DCOT% %DCOT %DCOT%	SA SVb VA VVb	cot(x)	1	D	$\|x\|<2^{46}$ $\|x-n\|>0$, $\|n\|=1,3,5,...$	D

† Not FORTRAN callable

HYPERBOLIC ROUTINES

Table 3-5. Hyperbolic routines

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Hyperbolic cosine	COSH COSH% %COSH %COSH%	SA SVa VA VVa	$(e^x+e^{-x})/2$	1	R	$\lvert x\rvert<2^{13}\ln 2$	R
Hyperbolic sine	SINH SINH% %SINH %SINH%	SA SVa VA VVa	$(e^x-e^{-x})/2$	1	R	$\lvert x\rvert<2^{13}\ln 2$	R
Hyperbolic cosine and sine†	COSSH COSSH% %COSSH %COSSH%	SA SVb VA VVb	$y_1=(e^x+e^{-x})/2$ $y_2=(e^x-e^{-x})/2$	1	R	$\lvert x\rvert<2^{13}\ln 2$	R
Hyperbolic tangent	TANH TANH% %TANH %TANH%	SA SVa VA VVa	$(e^x-e^{-x})/$ (e^x+e^{-x})	1	R	$\lvert x\rvert<2^{13}\ln 2$	R
Double-prec. hyperbolic cosine	DCOSH DCOSH% %DCOSH %DCOSH%	SA SVb VA VVb	cosh(x)	1	D	$\lvert x\rvert<2^{13}\ln 2$	D
Double-prec. hyperbolic sine	DSINH DSINH% %DSINH %DSINH%	SA SVb VA VVb	sinh(x)	1	D	$\lvert x\rvert<2^{13}\ln 2$	D

† Not FORTRAN callable

Table 3-5. Hyperbolic routines (continued)

General Purpose	Entry Names	Call Seq. Code	Definition	Arguments			Func. Value Type
				No.	Type	Range	
Double-prec. hyperbolic tangent	DTANH DTANH %DTANH %DTANH%	SA SVb VA VVb	tanh(x)	1	D	$\|x\|<2^{13}\ln 2$	D

BOOLEAN ARITHMETIC ROUTINES

These scalar subprograms in table 3-6 are external versions of CFT in-line functions. These functions can be passed as arguments to user-defined functions. They are all called by address and results are returned in register S1.

Table 3-6. Boolean arithmetic routines

Function	Definition	Arguments		Function Value Type
		No.	Type	
AND	Computes logical product $\begin{array}{r}0011\\ \underline{1010}\\ 0010\end{array}$	2	I,R,L,B	B
COMPL	Computes logical complement $\begin{array}{r}\underline{01}\\ 10\end{array}$	1	I,R,L,B	B
EQV	Computes logical equivalence $\begin{array}{r}0011\\ \underline{1010}\\ 0110\end{array}$	2	I,R,L,B	B
LEADZ	Counts the number of leading zero bits	1	I,R,L,B	I
MASK	Returns a bit mask of ones. If $0 \le arg \le 63$, the mask is left-justified. If $64 \le arg \le 128$, a right-justified mask of (128-argument) bits is returned.	1	I	B
NEQV	Computes logical difference (same as XOR)	2	I,R,L,B	B
OR	Computes logical sum $\begin{array}{r}0011\\ \underline{1010}\\ 1011\end{array}$	2	I,R,L,B	B

Table 3-6. Boolean arithmetic routines (continued)

Function	Definition	Arguments		Function Value Type
		No.	Type	
POPCNT	Counts the number of bits set to 1	1	I,R,L,B	I
POPPAR	Returns 0 if even number of bits set; returns 1 if odd number of bits set	1	I,R,L,B	I
SHIFT	Performs circular shift of (arg_1) to the left by (arg_2) bits	2	I,R,L,B I	B
SHIFTL	Performs left shift of (arg_1) by (arg_2) bits with zero fill	2	I,R,L,B	B
SHIFTR	Performs right shift of (arg_1) by (arg_2) bits with zero fill	2	I,R,L,B	B
XOR	Computes logical difference $\begin{array}{r}0011\\\underline{1010}\\1001\end{array}$	2	I,R,L,B	B

BASE VALUE RAISED TO A POWER ROUTINES

FORTRAN routines implicitly call the following routines to raise a value to a power. When the call is from CAL, the VL register must be set for vector functions. The following routines are called by value. In table 3-7, a plus sign before the TYPE (as in +D) indicates the value must be positive.

Table 3-7. Values raised to a power

Definition	Function Name	Arguments				Result	
		Base		Power			
		Type	Reg	Type	Reg	Type	Reg
Complex base raised to a complex power (C**C)	CTOC% CTO%C% %CTOC% %CTO%C%	C C C C	S1,S2 S1,S2 V1,V2 V1,V2	C C C C	S3,S4 V3,V4 S3,S4 V3,V4	C C C C	S1,S2 V1,V2 V1,V2 V1,V2
Complex base raised to an integer power (C**I)	CTOI% CTO%I% %CTOI% %CTO%I%	C C C C	S1,S2 S1,S2 V1,V2 V1,V2	I I I I	S3 V3 S3 V3	C C C C	S1,S2 V1,V2 V1,V2 V1,V2
Complex base raised to a real power (C**R)	CTOR% CTO%R% %CTOR% %CTO%R%	C C C C	S1,S2 S1,S2 V1,V2 V1,V2	R R R R	S3 V3 S3 V3	C C C C	S1,S2 V1,V2 V1,V2 V1,V2
Double-precision base raised to a double-precision power (D**D)	DTOD% DTO%D% %DTOD% %DTO%D%	+D +D +D +D	S1,S2 S1,S2 V1,V2 V1,V2	D D D D	S3,S4 V3,V4 S3,S4 V3,V4	D D D D	S1,S2 V1,V2 V1,V2 V1,V2

Table 3-7. Values raised to a power (continued)

Definition	Function Name	Arguments				Result	
		Base		Power			
		Type	Reg	Type	Reg	Type	Reg
Double-precision base raised to an integer power (D**I)	DTOI%	D	S1,S2	I	S3	D	S1,S2
	DTO%I%	D	S1,S2	I	V3	D	V1,V2
	%DTOI%	D	V1,V2	I	S3	D	V1,V2
	%DTO%I%	D	V1,V2	I	V3	D	V1,V2
Double-precision base raised to a real power (D**R)	DTOR%	+D	S1,S2	R	S3	D	S1,S2
	DTO%R%	+D	S1,S2	R	V3	D	V1,V2
	%DTOR%	+D	V1,V2	R	S3	D	V1,V2
	%DTO%R%	+D	V1,V2	R	V3	D	V1,V2
Integer base raised to an integer power (I**I)	ITOI%	I	S1	I	S2	I	S1
	ITO%I%	I	S1	I	V2	I	V1
	%ITOI%	I	V1	I	S2	I	V1
	%ITO%I%	I	V1	I	V2	I	V1
Real base raised to an integer power (R**I)	RTOI%	R	S1	I	S2	R	S1
	RTO%I%	R	S1	I	V2	R	V1
	%RTOI%	R	V1	I	S2	R	V1
	%RTO%I%	R	V1	I	V2	R	V1
Real base raised to a real power (R**R)	RTOR%	+R	S1	R	S2	R	S1
	RTO%R%	+R	S1	R	V2	R	V1
	%RTOR%	+R	V2	R	S2	R	V1
	%RTO%R%	+R	V2	R	V2	R	V1

DOUBLE-PRECISION ARITHMETIC ROUTINES

These routines are implicitly called by FORTRAN to do double-precision arithmetic. Double-precision arithmetic results are stored in two 64-bit computer words. In the first word, the high-order 16 bits contain the exponent and the low-order 48 bits contain the most significant part of the value. In the second word the low-order 48 bits contain the least significant part of the value. The first 16 bits of the second word must be 0. Double-precision arithmetic routines are called by value. Where two function names are given, use of the first one is preferred. Double-precision arithmetic routines are in table 3-8.

Table 3-8. Double-precision arithmetic routines

Definition	Function Name	Arguments				Result	
		Operand 1		Operand 2			
		Type	Reg	Type	Reg	Type	Reg
Double-precision addition (D+D)	DASS%	D	S1,S2	D	S3,S4	D	S1,S2
	DASV%	D	S1,S2	D	V3,V4	D	V1,V2
	DAVS%	D	V1,V2	D	S3,S4	D	V1,V2
	DAVV%	D	V1,V2	D	V3,V4	D	V1,V2
Double-precision division (D/D)	DDSS%	D	S1,S2	D	S3,S4	D	S1,S2
	DDSV%	D	S1,S2	D	V3,V4	D	V1,V2
	DDVS%	D	V1,V2	D	S3,S4	D	V1,V2
	DDVV%	D	V1,V2	D	V3,V4	D	V1,V2
Double-precision multiplication (D*D)	DMSS%	D	S1,S2	D	S3,S4	D	S1,S2
	DMSV%	D	S1,S2	D	V3,V4	D	V1,V2
	DMVS%	D	V1,V2	D	S3,S4	D	V1,V2
	DMVV%	D	V1,V2	D	V3,V4	D	V1,V2
Double-precision subtraction (D-D)	DSSS%	D	S1,S2	D	S3,S4	D	S1,S2
	DSSV%	D	S1,S2	D	V3,V4	D	V1,V2
	DSVS%	D	V3,V4	D	S3,S4	D	V1,V2
	DSVV%	D	V3,V4	D	V3,V4	D	V1,V2

TRIPLE-PRECISION ARITHMETIC ROUTINES

Triple-precision arithmetic results are stored in three contiguous 64-bit computer words. In the first word, the high-order 16 bits contain the exponent and the low-order 48 bits contain the first part of the value. The rest of the value is contained in the low-order 48 bits of the second and third words. The high-order 16 bits of the second and third words must be 0. If these routines are called from FORTRAN, the arguments must be passed in 3-word arrays. Triple-precision arithmetic routines are in table 3-9.

Table 3-9. Triple-precision arithmetic routines

Definition	Name	Call Type	Entry Conditions	Exit Conditions
Triple-precision addition	$TADD	Value	(S1)=address of addend (S2)=address of augend (S3)=address of result	Result to address in S3
	TADD†	Address	*arg*$_1$=address of addend *arg*$_2$=address of augend *arg*$_3$=address of result	Result to address in *arg*$_3$
	TASS	Address	*arg*$_1$=address of addend *arg*$_2$=address of augend	(S1),(S2),(S3)= result
	TASS%	Value	(S1),(S2),(S3)= addend (S4),(S5),(S6)= augend	(S1),(S2),(S3)= result

† FORTRAN entry point

Table 3-9. Triple-precision arithmetic routines (continued)

Definition	Name	Call Type	Entry Conditions	Exit Conditions
Triple-precision division	$TDIV	Value	(S1)=address of dividend (S2)=address of divisor (S3)=address of result	Result to address in S3
	TDIV[†]	Address	arg_1=address of dividend arg_2=address of divisor arg_3=address of result	Result to address in arg_3
	TDSS	Address	arg_1=address of dividend arg_2=address of divisor	(S1),(S2),(S3)= result
	TDSS%	Value	(S1),(S2),(S3)= dividend (S4),(S5),(S6)= divisor	(S1),(S2),(S3)= result
Triple-precision multiplication	$TMLT	Value	(S1)=address of multiplier (S2)=address of multiplicand (S3)=address of result	Result to address in S3
	TMLT[†]	Address	arg_1=address of multiplier arg_2=address of multiplicand arg_3=address of result	Result to address in arg_3
	TMSS	Address	arg_1=address of multiplier arg_2=address of multiplicand	(S1),(S2),(S3)= result
	TMSS%	Value	(S1),(S2),(S3)= multiplier (S4),(S5),(S6)= multiplicand	(S1),(S2),(S3)= result

[†] FORTRAN entry point

Table 3-9. Triple-precision arithmetic routines (continued)

Definition	Name	Call Type	Entry Conditions	Exit Conditions
Triple-precision subtraction	$TSUB	Value	(S1)=address of minuend (S2)=address of subtrahend (S3)=address of result	Result to address in S3
	TSUB[†]	Address	arg_1=address of minuend arg_2=address of subtrahend arg_3=address of result	Result to address in arg_3
	TSSS	Address	arg_1=address of minuend arg_2=address of subtrahend	(S1),(S2),(S3)= result
	TSSS%	Value	(S1),(S2),(S3)= minuend (S4),(S5),(S6)= subtrahend	(S1),(S2),(S3)= result

[†] FORTRAN entry point

Example of FORTRAN use:

```
REAL A(3),B(3),RSLT(3)
CALL TADD(A,B,RSLT)
```

Example 1 of CAL use:

Location	Result	Operand
	CALL	TASS,(ARG1,ARG2)

Example 2 of CAL use:

```
S1        1.
S2        0.
S3        0.
S4        1.
S5        0.
S6        0.
CALLV     TASS%
```

SIXTY-FOUR-BIT INTEGER DIVISION

The 64-bit integer routines in table 3-10 are implicitly called by FORTRAN. They divide two 64-bit integers to produce a 64-bit integer result. The integer division routines are called by value.

Table 3-10. 64-bit integer division

Definition	Name	Registers			Result Type
		Entry	Exit		
			Quo.	Rem.	
Scalar/scalar	LDSS%	S1,S2	S1	S2	I
Scalar/vector	LDSV%	S1,V2,VL	V1	V2	I
Vector/scalar	LDVS%	V1,S2,VL	V1	V2	I
Vector/vector	LDVV%	V1,V2,VL	V1	V2	I

CHARACTER FUNCTIONS

The character functions in table 3-11 are called by address. A character address is 64 bits. These routines are implicitly called by FORTRAN for the character comparisons: GE, GT, LE, and LT.

Table 3-11. Character comparison functions called from FORTRAN

Definition	Function Name	Arguments		Result	
		Operand 1	Operand 2	Type	Reg
ASCII compare for GE	LGE	Character	Character	L	S1
ASCII compare for GT	LGT	Character	Character	L	S1
ASCII compare for LE	LLE	Character	Character	L	S1
ASCII compare for LT	LLT	Character	Character	L	S1
Find position of second argument as substring of first argument	INDEX	Character	Character	I	S1
Find length of argument	LEN	Character		I	S1

Example:

Call from CAL:

```
CALL LGE,(arg1,arg2)
```

Entry:

arg_1	Address of character operand 1
arg_2	Address of character operand 2

Exit:

(S1)	Logical result of comparison

Call from FORTRAN:

```
logical=LGE(chr1,chr2)
```

logical	Logical result of comparison
chr_1	Character operand 1
chr_2	Character operand 2

The character functions in table 3-12 are called with the character address of the first operand in register S1 and the address of the second operand in register S2. These routines are called only from CAL.

Table 3-12. Character comparison functions called from CAL

Definition	Function Name	Arguments		Result	
		Operand 1	Operand 2	Type	Reg
ASCII compare for GE	$GE	Character	Character	L	S1
ASCII compare for GT	$GT	Character	Character	L	S1
ASCII compare for LE	$LE	Character	Character	L	S1
ASCII compare for LT	$LT	Character	Character	L	S1
ASCII compare for EQ	$EQ	Character	Character	L	S1
ASCII compare for NE	$NE	Character	Character	L	S1

Example:

Call from CAL:

```
CALLV $GE
```

Entry:

(S1)	Address of first character operand
(S2)	Address of second character operand

Exit:

(S1)	Logical result of comparison

CHARACTER CONCATENATION AND STORE ROUTINES

FORTRAN routines implicitly call the following routines to perform character concatenation. They are called in a manner similar to the I/O routines (see section 5 of this publication for a detailed description of I/O routines).

INITIALIZATION

$CCI initializes concatenation for store.

Call from CAL:

```
CALLV $CCI
```

Entry:

(S1)	Address of concatenated result

TRANSFER

$CCT transfers one character item to result.

Call from CAL:

```
CALLV $CCT
```

Entry:
  (S1)   Address of item

TERMINATION

$CCF terminates transfer; blank-filled.

Call from CAL:

```
CALLV $CCF
```

ASCII CONVERSION FUNCTIONS

The functions in table 3-13 convert binary integers to or from 1-word ASCII strings (not CFT character variable). The FORTRAN callable entry, *xxx*, uses the call-by-address sequence.

NOTE

These routines are not intrinsic to CFT. Their default type is real even though their results are generally used as integers.

Table 3-13. ASCII conversion

Purpose	Ent. Name	Call Seq. Code	Argument 1		Argument 2		Result
			Type	Range	Type	Range	
Converts binary to decimal ASCII (right-justified, blank-filled)	BTD BTD%	SA SVa	I	0≤x≤D'99999999			One word ASCII string (right-justified, blank-filled, decimal conversion)
Converts binary to decimal ASCII (left-justified, zero-filled)	BTDL BTDL%	SA SVa	I	0≤x≤D'99999999			One word ASCII string (left-justified, zero-filled, decimal conversion)
Converts binary to decimal ASCII (right-justified, zero-filled)	BTDR BTDR%	SA SVa	I	0≤x≤D'99999999			One word ASCII string (right-justified, zero-filled, decimal conversion)
Converts binary to octal ASCII (right-justified, blank-filled)	BTO BTO%	SA SVa	I	0≤x≤O'77777777			One word ASCII string (right-justified, blank-filled, octal conversion)
Converts binary to octal ASCII (left-justified, zero-filled)	BTOL BTOL%	SA SVa	I	0≤x≤O'77777777			One word ASCII string (left-justified, zero-filled, octal conversion)
Converts binary to octal ASCII (right-justified, zero-filled)	BTOR BTOR%	SA SVa	I	0≤x≤O'77777777			One word ASCII string (right-justified, zero-filled, octal conversion)
Converts decimal ASCII to binary	DTB DTB%	SA SVa	I	Decimal ASCII (left-justified, zero-filled)	I	Opt. (error code)	One word containing decimal equivalent of ASCII string. Error code: 0 if no error; -1 if error. Returned in second argument for CFT calls and in S0 for CAL calls. If no error code argument is included for CFT calls, routine aborts on error.
Converts octal ASCII to binary	OTB OTB%	SA SVa	I	Octal ASCII (left-justified, zero-filled)	I	Opt. (error code)	One word containing octal equivalent of ASCII string. Error code: 0 if no error; -1 if error. Returned in second argument for CFT calls and in S0 for CAL calls. If no error code argument is included for CFT calls, routine aborts on error.

Example:

Call from FORTRAN:

```
result=BTD(arg)
```

result	Decimal ASCII result (right-justified, blank-filled)
arg	Integer argument

Call from CAL:

```
CALLV BTD%,S1
```

Entry:

(S1)	Integer value

Exit:

(S1)	Decimal ASCII result (right-justified, blank-filled)

Example:

Call from FORTRAN:

```
result=DTB(arg,errcode)
```

result	Integer value
arg	Decimal ASCII (left-justified, zero-filled)
errcode	0 if conversion successful, -1 if error

Call from CAL:

```
CALLV DTB%,S1
```

Entry:

(S1)	Decimal ASCII (left-justified, zero-filled)

Exit:

(S1)	Integer value
(S0)	Error code 0 if conversion successful -1 if error

PSEUDO VECTORIZATION ROUTINES

Pseudo vectorization simulates vectorized math routines. See the PVEC macro in the Macros and Opdefs Reference Manual, CRI publication SR-0012.

MISCELLANEOUS MATH ROUTINES

The math routines can be divided into the following types.

- Absolute value
  ABS
  CABS
  DABS
  IABS

- Complex conjugate
  CONJG

- Double-precision product of real arguments
  DPROD

- Imaginary portion of complex number
  AIMAG

- Modulo arithmetic
  AMOD
  DMOD
  MOD

- Nearest integer
  NINT
  IDNINT

- Nearest whole number
  ANINT
  DNINT

- Positive difference
  DDIM
  DIM
  IDIM

- Sign transfer
  DSIGN
  ISIGN
  SIGN

- Truncation
  - AINT
  - DINT

- Type conversion
  - CHAR
  - CMPLX
  - DBLE
  - FLOAT
  - INT
  - ICHAR
  - REAL

Table 3-14 contains the miscellaneous math routines.

Table 3-14. Miscellaneous math routines

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Real absolute value	ABS	SA	R		R	
Find the imaginary portion of a complex number	AIMAG	SA	C		R	$\lvert x\rvert, \lvert y\rvert < \infty$
Truncate to integral value	AINT	SA	R		R	$\lvert x\rvert < 2^{46}$
Real modulo arithmetic $y=x_1-x_2[x_1/x_2]$	AMOD	SA x_1	R x_2	R y	R	$\lvert x_1\rvert < 2^{47}$ $0 < \lvert x_2\rvert < 2^{47}$
Calculates nearest whole number $y=\lfloor x+.5\rfloor$ if $x\geq 0$ $y=\lfloor x-.5\rfloor$ if $x<0$	ANINT	SA	R		R	$\lvert x\rvert < 2^{46}$

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Complex absolute value	CABS CABS% %CABS %CABS%	SA SVb VA VVb	C		R	$\lvert x\rvert, \lvert y\rvert < \infty$ $x^2+y^2 < \infty$ where complex argument $z=x+iy$
Integer to character conversion	CHAR		I		CH	
Convert two reals to a complex	CMPLX	SA	R	R	C	
Complex conjugate	CONJG	SA	C		C	$\lvert x\rvert, \lvert y\rvert < \infty$ where complex argument $z=x+iy$
Determine double-precision absolute value	DABS	SA	D		D	$\lvert x\rvert < \infty$
Convert real to double-precision	DBLE	SA	R		D	
Double-precision positive real difference MAX(0,x-y)	DDIM DDIM% %DDIM %DDIM%	SA SVb VA VVb	D	D	D	$\lvert x\rvert, \lvert y\rvert < \infty$

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type 1	Argument Type 2	Result Type	Restrictions
Positive real difference MAX(0,x-y)	DIM	SA	R	R	R	$\|x\|,\|y\|<\infty$
Truncate double-precision numbers, y=[x] fraction last no rounding	DINT DINT% %DINT %DINT%	SA SVb VA VVb	D		D	$\|x\|<2^{95}$
Double-precision modulo arithmetic $y=x_1-x_2[x_1/x_2]$	DMOD DMOD% %DMOD %DMOD%	SA SVb VA VVb	D x_1	D x_2	D y	$\|x_1\|<2^{95}$ $0<\|x_2\|<2^{95}$
Calculates nearest integer; defined as $\lfloor x+.5\rfloor$ if $x\geq 0$ $\lfloor x-.5\rfloor$ if $x<0$	DNINT DNINT% %DNINT %DNINT%	SA SVb VA VVb	D		D	$\|x\|<2^{95}$
Double-precision product of two real arguments	DPROD	SA	R	R	D	$\|x\|,\|y\|<\infty$

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Transfers sign from one double-precision number to another defined as $y=\|x_1\|$ if $x_2 \geq 0$ $y=\|x_1\|$ if $x_2<0$	DSIGN	SA	D (to which sign is transferred)	D (from which sign is transferred)	D	$\|x_1\|,\|x_2\|<\infty$
Convert integer to real	FLOAT	SA	I		R	$\|x\|<2^{46}$
Integer absolute value	IABS	SA	I		I	$\|x\|<\infty$
Character to integer conversion	ICHAR		CH		I	
Positive integer difference MAX(0,x-y)	IDIM	SA	I	I	I	$\|x\|,\|y\|<\infty$

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Nearest integer to a double-precision number; defined as $\lfloor x+.5 \rfloor$ if $x \geq 0$ $\lfloor x-.5 \rfloor$ if $x<0$	IDNINT IDNINT% %IDNINT %IDNINT%	SA SVb VA VVb	D		I	$\lvert x \rvert < 2^{46}$
Truncate to integral value	INT	SA	R		I	$\lvert x \rvert < 2^{46}$
Transfer sign from one integer to another defined as $y=\lvert x_1 \rvert$ if $x_2 \geq 0$ $y=-\lvert x_1 \rvert$ if $x_2<0$	ISIGN	SA	I (to which sign is transferred)	I (from which sign is transferred)	I	$\lvert x_1 \rvert , \lvert x_2 \rvert < \infty$
Perform 64-bit modulo arithmetic on two integer scalars $y=x_1-x_2[x_1/x_2]$	MOD MOD% MODSS%	SA SVb SVb	I x_1	I x_2	I y (S1-remainder S2 quotient)	$\lvert x_1 \rvert < 2^{63}$ $0 < \lvert x_2 \rvert < 2^{63}$

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Perform 64-bit modulo arithmetic on two integer vectors $y=\|x_1-x_2(x_1/x_2)\|$	%MOD% MODVV%	VVb VVb	I x_1 ient)	I x_2	I y (V1-remainder V2-quot-	$\|x_1\|<2^{63}$ $0<\|x_2\|<2^{63}$
Perform 64-bit modulo arithmetic on integer scalar and integer vector	MODSV%		I x_1 (dividend) (S1)	I x_2i- (divisor) (V2)	I y (V1-remainder V2-quotient)	$\|x_1\|<2^{63}$ $0<\|x_2\|<2^{63}$
Perform 64-bit modulo arithmetic on integer vector and integer scalar	MODVS%		I (dividend) (V1)	I (divisor) (S2)	I (V1-remainder S2-quotient)	$\|x_1\|<2^{63}$ $0<\|x_2\|<2^{63}$
Calculate nearest integer $y=\lfloor x+.5\rfloor$ if $x\geq 0$ $y=\lfloor x-.5\rfloor$ if $x<0$	NINT	SA			I	$\|x\|<2^{46}$
Return real portion of a complex number	REAL	SA	C		R	

Table 3-14. Miscellaneous math routines (continued)

General Purpose	Entry Name	Call Seq. Code	Argument Type		Result Type	Restrictions
			1	2		
Transfer sign from one real number to another; defined as $y=\|x_1\|$ if $x_2 \geq 0$ $y=\|x_1\|$ if $x_2 < 0$	SIGN	SA	R (to which sign is transferred)	R (from which sign is transferred)	R	$\|x_1\|,\|x_2\| < \infty$

Examples:

Call from CAL:

```
CALL FLOAT,(arg)
```

Entry:

arg_1 Address of integer argument

Exit:

(S1) Real result

Call from FORTRAN:

```
real=FLOAT(integer)
```

real	Real result
integer	Integer result

RANDOM NUMBER ROUTINES

Table 3-15. Random number routines

Purpose	Entry Name	Call Seq. Code	Argument Type	Result Type
Generates random numbers	RANF RANF% %RANF %RANF%	SA SVa VA VVa	No arguments required (VL) assumed correct for vectorized versions	R
Returns current seed of random number generator	RANGET RANGET%	SA SVa	I (optional)	I
Sets random seed	RANSET RANSET%	SA SVa	I (optional)	I

NOTE

When the seed of the random number generator is reset, RANSET does not store the supplied argument as the first value in the buffer of the random number seeds.

Examples:

RANF

Call from CAL:

```
CALL RANF
CALLV RANF%
CALL %RANF
CALLV %RANF%
```

Call from FORTRAN:

```
random=RANF( )                (Scalar version)
```

```
   DO 10 I=1,10
10 RANDOM (I)=RANF( )         (Vector version)
```

RANGET

Call from CAL:

```
CALL RANGET
CALLV RANGET%
```

Call from FORTRAN:

```
CALL RANGET(iseed)
iseed=RANGET( )
```

iseed	Contains the current seed

RANSET

Call from CAL:

```
CALL RANSET
CALLV RANSET%
```

Call from FORTRAN:

```
CALL RANSET(ivalue)
dummy=RANSET(ivalue)
```

MATH TABLES

The tables in the following list contain no executable instructions but are referenced by other library routines.

$IIN	Contains integer powers of 10 in the range of 10^0 to 10^{18}
$IMX	Contains double-precision floating-point representation of negative powers of 10 in the range of 10^0 to 10^{-4096}
$IPX	Contains double-precision floating-point representation of positive powers of 10 in the range of 10^0 to 10^{4096}
RANFI	Contains the current index to the seed buffer for the random number generator
RANFS	Contains 128 random number seeds in a buffer

SCIENTIFIC APPLICATIONS SUBPROGRAMS

4

INTRODUCTION

The scientific applications subprograms are written to run optimally on the Cray computer. These subprograms use the call-by-address convention when called by a FORTRAN or CAL program. See the introduction to this manual for details of the call-by-address convention.

The subprograms are grouped as follows:

- Basic linear algebra subprograms
- Other linear algebra subprograms
- Functions and linear recurrence routines
- Linpack routines
- Eispack routines
- Matrix inverse and multiplication routines
- Fast Fourier transform routines
- Filter routines
- Gather, scatter routines
- Search routines
- Sort routine

BASIC LINEAR ALGEBRA SUBPROGRAMS

The Cray computer user has access to a subset of the Basic Linear Algebra Subprograms (BLAS), a package of 22 CAL-coded routines. Only the single-precision and complex versions of the BLAS are included in the package. The following operations are available.

- Dot products
- Vector scaling
- Vector copy and swap
- Givens transformations
- Pivot search (maximum element)
- Euclidean norm
- A constant times a vector plus another vector
- Sum of absolute values

Each BLAS routine has a real version and a complex version. Type and dimension declarations for variables occurring in the subprograms must appear in the following manner.

```
REAL      SX(mx), SY(my), SA
REAL      C, S, A, B, PARM(4), D1, D2, B1, B2
COMPLEX   CX(mx), CY(my), CA
```

where dimensions *mx*=max(1,N*|INCX|), *my*=max(1,N*|INCY|), and N is the array length of the input vectors. In all routines, if N≤0, inputs and outputs return unchanged.

Type declarations for function names follow:

```
INTEGER   ISAMAX, ICAMAX
REAL      SASUM, SCASUM, SDOT, SNRM2, SCNRM2
COMPLEX   CDOTC, CDOTU
```

The declaration for complex functions is especially important to avoid type conversion to zero imaginary parts.

Arrays can have non-unit spacing between elements. The parameters *incx* and *incy* specify skip distances, allowing vector operands to be noncontiguous elements of memory. A value of 1 indicates contiguous elements. When a negative skip distance is specified, the operands are used in reverse order. Since FORTRAN dimension statements allow only positive integers for array lengths, references must be confined to array elements having only positive indexes. Therefore, if spacing between elements is negative, reversing the orientation of number is required. Thus if array SX contains elements x1, x2,..., xn, the contents of memory spaces SX(1), SX(2),..., SX(*n*) are

x1,x2,...,xn if *incx*=1, or xn,xn-1,...,x1 if *incx*=-1.

That is, element x*i* is in location SX(1+(*i*-1)**incx*) if *incx*>0 or in location SX(1+(*n*-*i*)**incx*) if *incx*<0. Location SX(1) is passed regardless of the sign of *incx*.

Example:

Let X(1)=1.0, X(2)=2.0, X(3)=3.0,..., X(10)=10. The real function, SASUM, which sums the absolute value of elements of a vector, is evaluated as follows.

```
SASUM(5,X,2)=((((1.0+3.0)+5.0)+7.0)+9.0), and
SASUM(5,X,-2)=((((9.0+7.0)+5.0)+3.0)+1.0)
```

Table 4-1 contains the purpose, name, and type of each BLAS.

Table 4-1. Basic linear algebra subprograms (BLAS)

Purpose	Name(*parameter list*)	Type
Index of element with maximum absolute value	ISAMAX(*n*,*sx*,*incx*) ICAMAX(*n*,*cx*,*incx*)	Integer function
Sum of the absolute values	SASUM(*n*,*sx*,*incx*) SCASUM(*n*,*cx*,*incx*)	Real function
Constant times a vector plus another vector	SAXPY(*n*,*sa*,*sx*,*incx*,*sy*,*incy*) CAXPY(*n*,*ca*,*cx*,*incx*,*cy*,*incy*)	Routine
Copy one array into another	SCOPY(*n*,*sx*,*incx*,*sy*,*incy*) CCOPY(*n*,*cx*,*incx*,*cy*,*incy*)	Routine
Euclidean norm of array	SNRM2(*n*,*sx*,*incx*) SCNRM2(*n*,*cx*,*incx*)	Real function

Table 4-1. Basic linear algebra subprograms (BLAS) (continued)

Purpose	Name(*parameter list*)	Type
Dot product	SDOT(*n,sx,incx,sy,incy*) CDOTC(*n,cx,incx,cy,incy*) CDOTU(*n,cx,incx,cy,incy*)	Real function Complex function
Construct Givens plane rotation	SROTG(*a,b,c,s*)	Routine
Apply Givens plane rotation	SROT(*n,sx,incx,sy,incy,c,s*)	Routine
Construct Givens modified plane rotation	SROTMG (*d1,d2,b1,b2,param*)	Routine
Apply Givens modified plane rotation	SROTM(*n,sx,incx,sy,incy, param*)	Routine
Scale array	SSCAL(*n,sa,sx,incx*) CSSCAL(*n,sa,cx,incx*) CSCAL(*n,ca,cx,incx*)	Routine
Swap two arrays	SSWAP(*n,sx,incx,sy,incy*) CSWAP(*n,cx,incx,cy,incy*)	Routine

INDEX OF ELEMENT HAVING MAXIMUM ABSOLUTE VALUE

These integer functions find the first index of the largest absolute value of the elements of a vector.

ISAMAX returns the first index i such that

$$|x_i|=\max\ |x_j|:j=1,\ldots,n\ .$$

where x_j is an element of a real vector.

Call from FORTRAN:

```
imax=ISAMAX(n,sx,incx)
```

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; etc.)
sx	Real vector to be searched
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

ICAMAX determines the first index i such that

$$|\mathrm{Real}(x_i)|+|\mathrm{Imag}(x_i)|=\max\left\{|\mathrm{Real}(x_j)|+|\mathrm{Imag}(x_j)|:j=1,\ldots,n\right\}.$$

where x_j is an element of a complex vector.

Call from FORTRAN:

```
imax=ICAMAX(n,cx,incx)
```

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; etc.)
cx	Complex vector to be searched
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

SUM OF THE ABSOLUTE VALUES

These real functions sum the absolute values of a vector.

SASUM computes

$$sum=\sum_{i=1}^{n} |x_i|$$

where x_i is an element of a real vector.

Call from FORTRAN:

```
sum=SASUM(n,sx,incx)
```

n	Number of elements in the vector to be summed
sx	Real vector to be summed
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

SCASUM computes

$$sum=\sum_{i=1}^{n}\left\{|\mathrm{Real}(x_i)|+|\mathrm{Imag}(x_i)|\right\}$$

where x_i is an element of a complex vector.

Call from FORTRAN:

```
sum=SCASUM(n,cx,incx)
```

n	Number of elements in the vector to be summed
cx	Complex vector to be summed
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

CONSTANT TIMES A VECTOR PLUS ANOTHER VECTOR

These subroutines add a scalar multiple of one vector to another.

SAXPY computes

$$Y=aX+Y$$

where a is a real scalar multiplier and X and Y are real vectors.

Call from FORTRAN:

```
CALL SAXPY(n,sa,sx,incx,sy,incy)
```

n	Number of elements in the vectors
sa	Real scalar multiplier
sx	Real scaled vector
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.
sy	Real result vector
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.

CAXPY computes

Y=aX+Y

where a is a complex scalar multiplier and X and Y are complex vectors.

Call from FORTRAN:

```
CALL CAXPY(n,ca,cx,incx,cy,incy)
```

n	Number of elements in the vectors
ca	Complex scalar multiplier
cx	Complex scaled vector
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Complex result vector
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.

COPY ONE ARRAY INTO ANOTHER

These subroutines copy a vector.

SCOPY copies a real vector

$$y_i=x_i:i=1,...,n$$

where x_i and y_i are elements of real vectors.

Call from FORTRAN:

```
CALL SCOPY(n,sx,incx,sy,incy)
```

n	Number of elements in the vector to be copied
sx	Real vector to be copied
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.
sy	Real result vector
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.

CCOPY copies a complex vector

$$y_i=x_i:i=1,...,n$$

where x_i and y_i are elements of complex vectors.

Call from FORTRAN:

```
CALL CCOPY(n,cx,incx,cy,incy)
```

n	Number of elements in the vector to be copied
cx	Complex vector to be copied
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Complex result vector
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.

COMPUTE AN INNER PRODUCT OF TWO VECTORS

These real and complex functions compute an inner product of two vectors.

SDOT computes

$$dot = \sum_{i=1}^{n} x_i y_i$$

where x_i and y_i are elements of real vectors.

Call from FORTRAN:

```
dot=SDOT(n,sx,incx,sy,incy)
```

n	Number of elements in the vectors
sx	Real vector operand
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.
sy	Real vector operand
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.

CDOTC computes

$$cdot = \sum_{i=1}^{n} \bar{x}_i y_i$$

where x_i and y_i are elements of complex vectors and $\bar{x}_i$ is the complex conjugate of x_i.

Call from FORTRAN:

```
cdot=CDOTC(n,cx,incx,cy,incy)
```

n	Number of elements in vector
cx	Complex vector operand
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Complex vector operand
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.

CDOTU computes

$$cdot=\sum_{i=1}^{n} x_i y_i$$

where x_i and y_i are elements of complex vectors.

Call from FORTRAN:

```
cdot=CDOTU(n,cx,incx,cy,incy)
```

n	Number of elements in vector
cx	Complex vector operand
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Complex vector operand
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.

EUCLIDEAN NORM OF AN ARRAY (l_2 NORM)

These real functions compute the Euclidean or l_2 norm of a vector.

SNRM2 computes

$$eucnorm=\left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}$$

where x_i is an element of a real vector.

Call from FORTRAN:

```
eucnorm=SNRM2(n,sx,incx)
```

n	Number of elements in vector
sx	Real vector operand
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

SCNRM2 computes

$$eucnorm= \left(\sum_{i=1}^{n} x_i\overline{x}_i\right)^{1/2}$$

where x_i is a complex vector and $\overline{x}_i$ is the complex conjugate of x_i.

Call from FORTRAN:

```
eucnorm=SCNRM2(n,cx,incx)
```

n	Number of elements in vector
cx	Complex vector operand
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

CONSTRUCT GIVENS PLANE ROTATION

SROTG computes the elements of a Givens rotation matrix. The following call calculates the parameters r, z, c, s, from input coordinates a,b as in equation 1.

Call from FORTRAN:

```
CALL SROTG(a,b,c,s)
```

a	Scalar a of equation 1
b	Scalar b of equation 1
c	Scalar cosine of equation 1
s	Scalar sine of equation 1

Equation 1:

$$\begin{bmatrix} r \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

z must contain enough information to reconstruct c,s; that is, from plane coordinates a,b, SROTG calculates

$$\begin{aligned} r &= \mathrm{sgn}(a) * \sqrt{a^2+b^2} \quad \text{if } |a|>|b| \\ &= \mathrm{sgn}(b) * \sqrt{a^2+b^2} \quad \text{if } |a|\leq|b| \end{aligned}$$

and

$$\begin{aligned} c&=a/r && \text{if } r\neq 0 \\ &=1 && \text{if } r=0 \\ s&=b/r && \text{if } r\neq 0 \\ &=0 && \text{if } r=0. \end{aligned}$$

Parameter z is

$$\begin{aligned} z&=s && \text{if } |a|>|b| \quad \text{or } a=b=0 \\ &=1/c && \text{if } 0<|a|\leq|b| \\ &=1 && \text{if } |b|>|a|=0 \end{aligned}$$

Note that if $|z|\leq 1$, then

$$s=z$$

$$c=\sqrt{1-z^2}$$

while if $|z|>1$, then

$$c=1/z$$

$$s=\sqrt{1-(1/z)^2}.$$

The subroutine uses parameters *a* and *b* and returns r,z,c,s, where r overwrites *a* and z overwrites *b*.

APPLY GIVENS PLANE ROTATION

This subroutine performs a matrix multiplication. If the coefficients *c* and *s* satisfy *c*c+s*s*=1.0, the transformation is a Givens rotation. The coefficients *c* and *s* can be calculated from *sx* and *sy* using SROTG.

SROT computes equation 2 on each pair of elements x_i, y_i of real arrays.

Call from FORTRAN:

```
CALL SROT(n,sx,incx,sy,incy,c,s)
```

n	Number of elements in vector
sx	Real vector to be modified
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.
sy	Real vector to be modified
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.
c	Real cosine of equation 2. Normally calculated using SROTG.
s	Real sine of equation 2. Normally calculated using SROTG.

Equation 2:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} :i = 1,\ldots,n$$

SROT returns without modification to any input parameters if *c*=1 and *s*=0.

CONSTRUCT MODIFIED GIVENS PLANE ROTATION

SROTMG computes the elements of a modified Givens plane rotation matrix.

```
CALL SROTMG(d1,d2,b1,b2,param)
```

SROTMG sets up parameters *param* from inputs d_1, d_2, b_1, b_2. The following is a brief description.

An application of the Givens plane rotation

$$\begin{bmatrix} x' \\ o \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = G \begin{bmatrix} x \\ y \end{bmatrix}$$

can be written in a form such that repeated applications require matrix multiplications by matrices containing only two non-unit elements. Row transformations require only 2N multiplications, rather than 4N. Scale factors d_1, d_2 are defined such that

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ 0 & \sqrt{d_2} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = D^{1/2} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where the scaling upon each application of the G's is updated. Let H be a matrix

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

such that

$$G \begin{bmatrix} x \\ y \end{bmatrix} = D'^{1/2} \; H \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

where $D'^{1/2} = \text{diag} \left\{ \sqrt{d'_1}, \sqrt{d'_2} \right\}$ contains the updated scale factors; therefore, H is chosen according to equation 3 or 4.

Equation 3:

$$\begin{pmatrix} x' \\ o \end{pmatrix} = D'^{1/2} \; H \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Equation 4:

$$\begin{pmatrix} \sqrt{d_1'} & h_{11} & \sqrt{d_1'} & h_{12} \\ \sqrt{d_2'} & h_{21} & \sqrt{d_2'} & h_{22} \end{pmatrix} = \begin{pmatrix} \sqrt{d_1}\ \mathrm{c} & d_2\ \mathrm{s} \\ -\sqrt{d_1}\ \mathrm{s} & d_2\ \mathrm{c} \end{pmatrix}$$

Coefficients c and s are determined by equations 5 and 6.

Equation 5:

$$\mathrm{c} = \frac{x}{\sqrt{x^2+y^2}} = \frac{d_1 b_1}{\sqrt{d_1 b_1^2 + d_2 b_2^2}},$$

Equation 6:

$$\mathrm{s} = \frac{y}{\sqrt{x^2+y^2}} = \frac{d_2 b_2}{\sqrt{d_1 b_1^2 + d_2 b_2^2}}$$

Equation 4 shows that the d's are going to be scaled by c or s if two of the h's are to be unity. Two cases, $|c|>|s|$ and $|s|\geq|c|$, are considered so that the d's are scaled down the least upon repeated applications.

Case 1:

If $|c|>|s|$ (which from equations 5 and 6 is the same as $|d_1 b_1^2|>|d_2 b_2^2|$), the solutions for equation 4 are determined by equation 7.

Equation 7:

$$h_{11}=h_{22}=1$$

Case 2:

If $|s|\geq|c|$ (which is $|d_2 b_2^2|\geq|d_1 b_1|$), equation 8 is chosen.

Equation 8:

$$h_{12}=-h_{21} = 1.$$

Distinguishing the two cases, $|c|>\frac{1}{\sqrt{2}}$ or $|s|\geq \frac{1}{\sqrt{2}}$ is the updating factor. Then the complete solutions for $D'^{1/2}$ and H are as follows.

Case 1:

In case 1, where $|c| > |s|$ or $|d_1 b_1^2|>|d_2 b_2^2|$, the following solutions for H are chosen:

$$h_{11}=1 \quad h_{12} = \frac{d_2 b_2}{d_1 b_1}$$

$$h_{21} = \frac{-b_2}{b_1} \quad h_{22}=1$$

and scale factors d_1, d_2 are updated to

$$d_1' = d_1/ u = c^2 \ d_1$$

$$d_2' = d_2 /u = c^2 \ d_2$$

where

$$u = det \ (H) = 1 + \frac{d_2 b_2^2}{d_1 b_1^2}$$

and since $x' = r$, $y' = 0$, and $b_1' = x' / \sqrt{d_1'}$ then $b_1' = b_1 . u$ is updated.

Case 2:

In case 2, where $|s|\geq|c|$ or $|d_1 b_1^2|\leq|d_2 b_2^2|$, the following solutions for H are chosen.

$$h_{11} = \frac{d_1 b_1}{d_2 b_2} \quad h_{12} = 1$$

$$h_{21} = -1 \qquad h = {}^{b_1}/b_2$$

and scale factors d_i are updated to

$$d_1' = d_{2/u}$$
$$d_2' = d_{1/u}$$

with

$$u = det(\mathrm{H}) = 1 + \frac{d_1 b_1^2}{d_2 b_2^2}$$

and the x' factor becomes

$$b_1' = b_2.u.$$

Case 3:

Let $m = 4096$. Whenever the parameters d_i are updated to be outside the window

$$(m)^{-2} \leq |d_i'| \leq (m)^2$$

which preserves about 36 = 48 - 12 bits or 10 decimal digits of precision, all parameters are rescaled such that the d_i's are within that window. However, if either of the d_i's is 0, no rescaling action is taken.

Underflow:

If $|d_i'| < (m)^{-2}$, then the following is set.

$$d_i' := d_i'.(m)^2, \quad h_{i1}' := h_{i1}' .(m)^{-1},$$
$$b_1' := b_1' .(m)^{-1}, \quad h_{i2}' := h_{i2}'.(m)^{-1}.$$

Overflow:

If $|d_i'| > (m)^2$, then we set

$$d_i' := d_i'.(m)^{-2}, \quad h_{i1}' := h_{i1}'.(m),$$
$$b_1' := b_1' .(m), \quad h_{i2}' := h_{i2}' .(m).$$

SROTMG modifies the input parameters D1, D2, and B1 and returns the array PARAM according to the following schedule:

Case 4:

If ABS(D1*B1*B1).GT.ABS(D2*B2*B2), then

```
    PARAM(1)=0

    PARAM(3)=-B2/B1

    PARAM(4)=D2*B2/D1*B1
```

and parameters D1, D2, and B1 are written over by

```
    D1=D1/U

    D2=D2/U

    B1=B1*U
```

where

```
    U=1.+(D2*B2*B2)/(D1*B1*B1).
```

Case 5:

If ABS(D2*B2*B2).GE.ABS(D1*B1*B1), then

```
    PARAM(1)=1.

    PARAM(2)=D1*B1/D2*B2

    PARAM(5)=B1/B2
```

and parameters D1, D2, and B1 are written over according to the following sequence.

```
    TEMP=D1/U

    D1=D2/U

    D2=TEMP

    B1=B2*U
```

U=1.+(D1*B1*B1)/(D2*B2*B2)

Case 6:

If, in either case 4 or 5, the updated parameters D1 and D2 have been rescaled below/above the window,

`(m)**(-2).LE.ABS(D1).LE.(m)**2`

`(m)**(-2).LE.ABS(D2).LE.(m)**2`

then the parameters D1, H11, H12, B1 and D2, H21, H22, respectively, are rescaled up/down by factors of *m*. Rescaling occurs as many times as necessary to bring D1 or D2 within the above window. If D1 and D2 are within the window on entry, rescaling occurs only once.

Output parameters are

PARAM(1)=-1.

PARAM(2)=H11

PARAM(3)=H21

PARAM(4)=H12

PARAM(5)=H22

and D1, D2, and B1 are written over by correctly scaled versions of case 5 or 6.

If D1<0, the matrix H=0 is generated (that is, $h_{11}=h_{12}=h_{21}=h_{22}=0$) PARAM(1)=-1. and the rest of the elements of PARAM contain zero.

Case 7:

If D2*B2=0 on entry, then H=1. Output is

PARAM(1)=-2.0 only.

APPLY MODIFIED GIVENS PLANE ROTATION

SROTM applies the modified Givens plane rotation constructed by SROTMG.

CALL SROTM(*n*,*sx*,*incx*,*sy*,*incy*,*param*) computes

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} : i=1,\ldots,n$$

where the parameters H11, H21, H12, and H22 are passed in the array PARAM according to the following schedule: PARAM(1) is the key parameter having values 1.0, 0.0, -1.0, or -2.0.

Case for which PARAM(1)=1.0:

H11=PARAM(2)

H21=-1.0

H12=1.0

H22=PARAM(5)

and PARAM(3) and PARAM(4) are ignored.

Case for which PARAM(1)=0.0:

H11=1.0

H21=PARAM(3)

H12=PARAM(4)

H22=1.0

and PARAM(2) and PARAM(5) are ignored.

Case for which PARAM(1)=-1.0 is rescaling case:

H11=PARAM(2)

H21=PARAM(3)

H12=PARAM(4)

H22=PARAM(5)

is a full matrix multiplication.

Case for which PARAM(1)=2.0 is H=1, namely:

H11=1.0

H21=0.0

```
H12=0.0

H22=1.0
```

and PARAM(2), PARAM(3), PARAM(4), and PARAM(5) are ignored. If H=1, SROTM returns with no operation on input arrays *sx*, *sy*.

If any other value for PARAM(1) is read (other than 1., 0, -1., -2.), SROTM aborts the job with the message:

```
SROTM CALLED WITH INCORRECT PARAMETER KEY
```

appearing in the logfile.

The array PARAM must be declared in a dimension statement:

```
DIMENSION PARAM(5)
```

in the calling program. See the description of SROTMG for further details about the modified Givens transformation and the array PARAM.

SCALE ARRAY

These subroutines scale a vector.

SSCAL computes

X=aX

where a is a real number and X is a real vector.

Call from FORTRAN:

```
CALL SSCAL(n,sa,sx,incx)
```

n	Number of elements in vector
sa	Real scaling factor
sx	Real vector to be scaled
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

CSSCAL computes

X=aX

where a is a real number and X is a complex vector.

Call from FORTRAN:

```
CALL CSSCAL(n,sa,cx,incx)
```

n	Number of elements in vector
sa	Real scaling factor
cx	Complex vector to be scaled
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

CSCAL computes

Y=aY

where a is a complex number and Y is a complex vector.

Call from FORTRAN:

```
CALL CSCAL(n,ca,cx,incx)
```

n	Number of elements in vector
ca	Complex scaling factor
cx	Complex vector to be scaled
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

SWAP TWO ARRAYS

These subroutines interchange two arrays.

SSWAP exchanges two real vectors.

Call from FORTRAN:

```
CALL SSWAP(n,sx,incx,sy,incy)
```

n	Number of elements in vector
sx	One real vector
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.
sy	Another real vector
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.

CSWAP exchanges two complex vectors.

Call from FORTRAN:

```
CALL CSWAP(n,cx,incx,cy,incy)
```

n	Number of elements in vector
cx	One complex vector
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Another complex vector
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.

OTHER LINEAR ALGEBRA SUBPROGRAMS

These linear algebra subprograms are extensions of the BLAS and conform to the same calling sequence. Table 4-2 contains the purpose, name, and type of each linear algebra subprogram.

SPARSE MATRIX PRIMITIVES

This subroutine and function are useful primitives for the lower upper factorization and solution of sparse linear systems.

SPAXPY is defined in FORTRAN in the following way.

```
      DO 10 I=1,N
 10   SY(INDEX(I))=SA*SX(I)+SY(INDEX(I))
```

Call from FORTRAN:

```
CALL SPAXPY(n,sa,sx,sy,index)
```

n	Number of elements in the vectors
sa	Real scalar multiplier
sx	Real vector operand
sy	Real vector operand
index	Vector of indexes

SPODT is defined in FORTRAN in the following way.

```
      DO 10 I=1,N
 10   PDOT=PDOT+SY(INDEX(I))*SX(I)
```

Call from FORTRAN:

```
pdot=SPDOT(n,sy,index,sx)
```

n	Number of elements in the vectors
sy	Real vector operand
sx	Real vector operand
index	Vector of indexes

Table 4-2. Other linear algebra subprograms

Purpose	Name(*parameter list*)	Type
Primitives for the LU factorization of sparse linear systems	SPAXPY(*n,sa,sx,sy,index*) SPDOT(*n,sy,index,sx*)	Routine Real function
Index of element with maximum or minimum value	ISMAX(*n,sx,incx*) ISMIN(*n,sx,incx*)	Integer functions
Index of element with minimum absolute value	ISAMIN(*n,sx,incx*)	Integer function
Sum the elements of a vector	SSUM(*n,sx,incx*) CSUM(*n,cx,incx*)	Real function Complex function
Construct complex Givens plane rotation	CROTG(*ca,cb,cc,cs*)	Routine
Apply complex Givens plane rotation	CROT(*n,cx,incx,cy,incy, cc,cs*)	Routine
Cray machine constants	SMACH(*mach*)	Real function

INDEX OF ELEMENT WITH MAXIMUM OR MINIMUM VALUE

These integer functions find the first index of the largest or smallest element of a real vector.

ISMAX returns the first index i such that

$$|x_i|=\max\left\{x_j:j = 1,\ldots,n\right\}.$$

where x_j is an element of a real vector.

Call from FORTRAN:

```
imax=ISMAX(n,sx,incx)
```

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; etc.)
sx	Real vector to be searched
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

ISMIN returns the first index i such that

$$|x_i|=\min\{x_j:j=1,...n\}.$$

where x_j is an element of a real vector.

Call from FORTRAN:

```
imin=ISMIN(n,sx,incx)
```

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; etc.)
sx	Real vector to be searched
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

INDEX OF ELEMENT HAVING MINIMUM ABSOLUTE VALUE

This integer function finds the first index of the smallest absolute value of the vector elements of a real vector.

ISAMIN returns the first index i such that

$$|x_i|=\min\{|x_j|:j=1,...n\}.$$

where x_j is an element of a real vector.

Call from FORTRAN:

```
imin=ISAMIN(n,sx,incx)
```

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; etc.)
sx	Real vector to be searched
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

SUM OF THE VALUES

These functions sum the elements of a real or complex vector.

SSUM sums the elements of a real vector.

Call from FORTRAN:

```
sum=SSUM(n,sx,incx)
```

n	Number of elements in vector
sx	Real vector to be summed
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

CSUM sums the elements of a complex vector.

Call from FORTRAN:

```
sum=CSUM(n,cx,incx)
```

n	Number of elements in vector
cx	Complex vector to be summed
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.

COMPUTE COMPLEX GIVENS PLANE ROTATION

CROTG computes the elements of a complex Givens plane rotation matrix as in equation 9.

Equation 9:

$$\begin{pmatrix} a' \\ 0 \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

The 2 x 2 matrix is unitary and A and B are overwritten.

Call from FORTRAN:

```
CALL CROTG(ca,cb,cc,cs)
```

ca	Complex a of equation 9
cb	Complex b of equation 9
cc	Complex sine of equation 9
cs	Complex cosine of equation 9

CONSTRUCT COMPLEX GIVENS PLANE ROTATION

CROT applies the complex Givens plane rotation computed by the subroutine CROTG. It performs equation 10.

Equation 10:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

where x and y are complex row vectors.

Call from FORTRAN:

```
CALL CROT(n,cx,incx,cy,incy,cc,cs)
```

n	Number of elements in vector
cx	Complex vector to be modified
incx	Skip distance between elements of *cx*. For contiguous elements, *incx*=1.
cy	Complex vector to be modified
incy	Skip distance between elements of *cy*. For contiguous elements, *incy*=1.
cc	Complex cosine of equation 10
cs	Complex sine of equation 10

CRAY MACHINE CONSTANTS

In SMACH or CMACH, *job* is an integer argument and returns Cray machine constants as calculated by the FORTRAN version of SMACH or CMACH. (See the Basic Linear Algebra Subprograms for FORTRAN Usage by Chuck L. Lawson, Richard J. Hanson, Davis R. Kincaid, and Fred T. Crow, published by Sandia Laboratories, Albuquerque, 1977, publication number SAND77-0898.)

Call from FORTRAN:

```
x=SMACH(job)
```

SMACH returns the following information.

for *job*=1	0.7105E-14	The machine epsilon (the smallest number ∈ such that 1.± ∈ ≠1.)
=2	.1290E-2449	A number close to smallest normalized, representable number
=3	.7750E+2450	A number close to largest normalized, representable number

Otherwise, an error message is returned to the user's logfile.

Call from FORTRAN:

```
x=CMACH(job)
```

CMACH returns the following information.

for *job*=1	0.7105E-14	The machine epsilon (the smallest number ϵ such that $1.\pm\epsilon\neq 1.$)
=2	.1348E1216	A number close to the square root of the smallest normalized, representable number
=3	.7421E+1217	A number close to the square root of the largest normalized, representable number

Otherwise, an error message is returned to the user's logfile. CMACH (2) and CMACH (3) were chosen to prevent overflow and underflow during complex division.

FUNCTIONS AND LINEAR RECURRENCE SUBROUTINES

These subroutines solve first-order and some second-order linear recurrences, respectively. A linear recurrence uses the result of a previous pass through the loop as an operand for subsequent passes through the loop. Such use prevents vectorization. These subroutines can be used to optimize FORTRAN loops containing linear recurrences.

FOLR solves first-order linear recurrences as in equation 11.

Equation 11:

$$c_1=b_1$$
$$c_i=-a_ic_{i-1}+b_i \text{ for } i=2,3...,n$$

or in FORTRAN,

```
      EQUIVALENCE(B,C)
      C(1)=B(1)
      DO 10 I=2,N
 10   C(I)=-A(I)*C(I-1)+B(I)
```

Call from FORTRAN:

```
CALL FOLR(n,a,inca,b,incb)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 11. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand A. For contiguous elements, *inca*=1.
b	Vector b of equation 11 on input and vector c of equation 11 on output. (The output overwrites the input.)
incb	Skip distance between elements of the vector operand *b* and result C. For contiguous elements, *incb*=1.

FOLRP solves first-order linear recurrences as in equation 12.

Equation 12:

$$c_1=b_1$$
$$c_i=a_ic_{i-1}+b_i \text{ for } i=2,3...,n$$

or in FORTRAN:

```
      EQUIVALENCE(B,C)
      C(1)=B(1)
      DO 10 I=2,N
10    C(I)=A(I)*C(I-1)+B(I)
```

Call from FORTRAN:

```
CALL FOLRP(n,a,inca,b,incb)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 12. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand *a*. For contiguous elements, *inca*=1.
b	Vector b of equation 12 on input and vector c of equation 12 on output. (The output overwrites the input.)
incb	Skip distance between elements of the vector operand *b* and result *c*. For contiguous elements, *incb*=1.

FOLR2 solves first-order linear recurrences as in equation 11. The solution, however, is written to a vector *c*, which is different from vector B in subroutine FOLR.

Call from FORTRAN:

```
CALL FOLR2(n,a,inca,b,incb,c,incc)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 11. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand *a*. For contiguous elements, *inca*=1.
b	Vector b of equation 11
incb	Skip distance between elements of the vector operand *b* and result C. For contiguous elements, *incb*=1.
c	Vector *c* of equation 11
incc	Skip distance between elements of the vector result *c*. For contiguous elements, *incc*=1.

FOLR2P is a combination of FOLRP and FOLR2.

Call from FORTRAN:

```
CALL FOLR2P(n,a,inca,b,incb,c,incc)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 12. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand *a*. For contiguous elements, *inca*=1.
b	Vector *b* of equation 12 on input
incb	Skip distance between elements of the vector operand *b*. For contiguous elements, *incb*=1.

c	Vector *c* of equation 12
incc	Skip distance between elements of the vector result *c*. For contiguous elements, *incc*=1.

FOLRN solves for the last term of a first-order linear recurrence. That is r_n of

$$r_1=b_1$$
$$r_i=-a_i r_{i-1}+b_i \quad i=2,3,\ldots,n$$

Call from FORTRAN:

```
result=FOLRN(n,a,inca,b,incb)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 11. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand A. For contiguous elements, *inca*=1.
b	Vector *b* of length *n* of equation 11. (The output overwrites the input.)
incb	Skip distance between elements of the vector operand and result *b*. For contiguous elements, *incb*=1.

Example:

This routine allows for efficient evaluation of polynomials using Horner's method.

Let $p(x)=\sum_{i=0}^{n} b_i x^{n-i}$

then $p(a)=(\ldots((b_0x+b_1)x+b_2)x+\ldots b_n)$ Horner's rule.

In FORTRAN:

```
      PA=B(0)
      DO 10 I=1,N
         PA=PA*X+B(I)
 10   CONTINUE
```

or equivalently

```
      PA=FOLRN(N+1,-X,0,B(0),1).
```

SOLR solves second-order linear recurrences as in equation 12.

Equation 12:

$$c_{i+2}=a_i c_{i+1}+b_i c_i \text{ for } i=1,2...$$

or in FORTRAN,

```
      DO 10 I=1,N
 10   C(I + 2)=A(I)*C(I+1)+B(I)*C(I)
```

Call from FORTRAN:

```
CALL SOLR(n,a,inca,b,incb,c,incc)
```

n	Length of linear recurrence
a	Vector *a* of length *n* of equation 12
inca	Skip distance between elements of the vector operand A. For contiguous elements, *inca*=1.
b	Vector *b* of length *n* of equation 12
incb	Skip distance between elements of the vector operand B. For contiguous elements, *incb*=1.
c	Vector result C of length N+2 of equation 12
incc	Skip distance between elements of the vector result C. For contiguous elements, *incc*=1. C(1) and C(2) are input to this routine; C(3),C(4),...,C(N+2) are output from this routine.

SOLRN solves for only the last term of a second-order linear recurrence, that is *c*(*n*) of SOLR(*n*,*a*,*inca*,*b*,*incb*,*c*,*incc*). SOLRN is a real function.

Call from FORTRAN:

```
result=SOLRN(n,a,inca,b,incb,c,incc)
```

n	Length of linear recurrence
a	Vector A of length N of equation 12
inca	Skip distance between elements of the vector operand A. For contiguous elements, *inca*=1.
b	Vector B of length N of equation 12
incb	Skip distance between elements of the vector operand B. For contiguous elements, *incb*=1.
c	Vector result C of length N+2 of equation 12
incc	Skip distance between elements of the vector result C. For contiguous elements, *incc*=1. C(1) and C(2) are input to this routine; C(3),C(4),...,C(N+2) are output from this routine.

The FORTRAN loop

```
      R1=C(1)
      R2=C(2)
      DO 10 I=1,N-2
         TEMP=R2
         R2=A(I)*R2+B(I)*R1
         R1=TEMP
   10 CONTINUE
      RESULT=R2
```

could be solved as follows.

```
result=SOLRN(n,a,1,b,1,c)
```

Example:

SOLRN might be used to find r_n of the calculation

$$\overset{n-2}{\underset{i=1}{}} \begin{pmatrix} a_i & b_i \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} r_n \\ r_{n-1} \end{pmatrix}$$

with the following call.

```
rn=SOLRN(n,a,1,b,1,c,1)
```

The equivalent FORTRAN follows.

```
      R1=C(1)
      R2=C(2)
      DO 10 I=1,N
         TEMP=R2
         R2=A(I)*R2+B(I)*R1
         R1=TEMP
 10   CONTINUE
      RN=R2
```

SOLR3 computes a second-order linear recurrence of three terms, that is

$c_1=c_1$
$c_2=c_2$
$c_i=c_i+a_{i-2}c_{i-1}+b_{i-2}c_{i-2} \quad i=3,...,n$

Call from FORTRAN:

```
CALL SOLR3(n,a,inca,b,incb,c,incc)
```

n	Length of linear recurrence
a	Vector A of length N of equation 12
inca	Skip distance between elements of the vector operand A. For contiguous elements, *inca*=1.
b	Vector *b* of length *n* of equation 12

incb	Skip distance between elements of the vector operand *b*. For contiguous elements, *incb*=1.
c	Vector result *c* of length *n*+2 of equation 12
incc	Skip distance between elements of the vector result *c*. For contiguous elements, *incc*=1. *c*(1) and *c*(2) are input to this routine; *c*(3),*c*(4),..., *c*(*n*+2) are output from this routine.

Example:

SOLR3 solves a system of lower bidiagonal linear equations Lx=b.

$$Lx=\begin{pmatrix} 1 & 0 & 0 & 0 & . & . & . & . & . & . & . & 0 \\ e_1 & 1 & 0 & 0. & . & . & . & . & . & . & . & 0 \\ f_1 & e_2 & 1 & 0 & . & . & . & . & . & . & . & 0 \\ 0 & f_2 & e_3 & 1 & 0 & . & . & . & . & . & . & 0 \\ . & 0 & f_3 & e_4 & 1 & 0 & . & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & & & & & f_{n-2} & e_{n-1} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ . \\ . \\ . \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ . \\ . \\ . \\ \\ b_n \end{pmatrix} = b$$

then there is

$$x_1=b_1$$
$$x_2=b_2-e_1x_1$$
$$x_i=b_i-e_{i-1}x_{i-1}-f_{i-2}x_{i-2} \qquad i=3,\dots,n$$

Given this problem, it can be solved with the following FORTRAN.

```
      DO 10 I=1,N-1
 10      E(I)=-E(I)
      DO 20 I=1,N-2
 20      F(I)=-F(I)

      B(1)=B(1)
      B(2)=B(2)+E(1)*B(1)
      CALL SOLR3(N,E(2),1,F(1),1,B(1),1)
```

SINGLE-PRECISION REAL AND COMPLEX LINPACK ROUTINES

LINPACK is a package of FORTRAN routines that solve systems of linear equations and compute the QR, Cholesky, and singular value decompositions. The original FORTRAN programs are documented in the LINPACK User's Guide by J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, published by the Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1979, Library of Congress catalog card number 78-78206.

Each single-precision $SCILIB version of the LINPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following:

- Replacement of calls to the BLAS routines SSCAL, SCOPY, SSWAP, SAXPY, and SROT with in-line FORTRAN code that the CFT compiler vectorizes

- Removal of FORTRAN IF statements if the result of either branch is the same

- Replacement of SDOT to solve triangular systems of linear equations in SGESL, SPOFA, SPOSL, STRSL, and SCHDD with more vectorizable code

These optimizations affect only the execution order of floating-point operations in modified DO-loops. Refer to the LINPACK User's Guide for further descriptions. The complex routines have been added without much optimization.

Table 4-3 contains the name, matrix, and purpose of each LINPACK routine in $SCILIB.

SINGLE-PRECISION EISPACK ROUTINES

EISPACK is a package of FORTRAN routines for solving the eigenvalue problem, computing and using the singular value decomposition, and solving banded symmetric systems of linear equations.

The original FORTRAN versions are documented in the Matrix Eigensystem Routines - EISPACK Guide, second edition by B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, published by Springer-Verlag, New York, 1976, Library of Congress catalog card number 76-2662; and in the Matrix Eigensystem Routines - EISPACK Guide Extension by B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, published by Springer-Verlag, New York, 1977, Library of Congress catalog card number 77-2802.

Table 4-3. Single-precision LINPACK routines

Name	Matrix or Decomposition	Purpose
SGECO	Real general	Factor and estimate condition
SGEFA		Factor
SGESL		Solve
SGEDI		Compute determinant and inverse
CGECO	Complex general	Factor and estimate condition
CGEFA		Factor
CGESL		Solve
CGEDI		Compute determinant and inverse
SGBCO	Real general banded	Factor and estimate condition
SGBFA		Factor
SGBSL		Solve
SGBDI		Compute determinant
CGBCO	Complex general banded	Factor and estimate condition
CGBFA		Factor
CGBSL		Solve
CGBDI		Compute determinant
SPOCO	Real positive definite	Factor and estimate condition
SPOFA		Factor
SPOSL		Solve
SPODI		Compute determinant and inverse
CPOCO	Complex positive definite	Factor and estimate condition
CPOFA		Factor
CPOSL		Solve
CPODI		Compute determinant and inverse
SPPCO	Real positive definite packed	Factor and estimate condition
SPPFA		Factor
SPPSL		Solve
SPPDI		Compute determinant and inverse
CPPCO	Complex positive definite packed	Factor and estimate condition
CPPFA		Factor
CPPSL		Solve
CPPDI		Compute determinant and inverse

Table 4-3. Single-precision LINPACK routines (continued)

Name	Matrix or Decomposition	Purpose
SPBCO SPBFA SPBSL SPBDI	Real positive definite banded	Factor and estimate condition Factor Solve Compute determinant
CPBCO CPBFA CPBSL CPBDI	Complex positive definite banded	Factor and estimate condition Factor Solve Compute determinant
SSICO SSIFA SSISL SSIDI	Symmetric indefinite	Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse
CHICO CHIFA CHISL CHIDI	Hermitian indefinite	Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse
SSPCO SSPFA SSPSL SSPDI	Symmetric indefinite packed	Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse
CHPCO CHPFA CHPSL CHPDI	Hermitian indefinite packed	Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse
STRCO STRSL STRDI	Real triangular	Factor and estimate condition Solve Compute determinant and inverse
CTRCO CTRSL CTRDI	Complex triangular	Factor and estimate condition Solve Compute determinant and inverse

Table 4-3. Single-precision LINPACK routines (continued)

Name	Matrix or Decomposition	Purpose
SGTSL	Real tridiagonal	Solve
CGTSL	Complex tridiagonal	Solve
SPTSL	Real positive definite tridiagonal	Solve
CPTSL	Complex	Solve
SCHDC	Real Cholesky decomposition	Decompose
SCHDD		Downdate
SCHUD		Update
SCHEX		Exchange
CCHDC	Complex Cholesky decomposition	Decompose
CCHDD		Downdate
CCHUD		Update
CCHEX		Exchange
SQRDC	Real	Orthogonal factorization
SQRSL		Solve
CQRDC	Complex	Orthogonal factorization
CQRSL		Solve
SSVDC	Real	Singular value decomposition
CSVDC	Complex	

Each $SCILIB version of the EISPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following.

- Use of the BLAS routines SDOT, SASUM, SNRM2, ISAMAX, and ISMIN when applicable
- Removal of FORTRAN IF statements if the result of either branch is the same

- Unrolling complicated FORTRAN DO-loops to improve vectorization
- Use of the CFT compiler directive CDIR$ IVDEP when no dependencies exist that prevent vectorization

These modifications increase vectorization and, therefore, reduce execution time. Only the order of computations within a loop is changed; the modified versions produce the same answers as the original versions unless the problem is sensitive to small changes in the data.

Table 4-4 contains the name and purpose of each EISPACK routine in $SCILIB.

Table 4-4. Single-precision EISPACK routines

Name	Matrix or Decomposition	Purpose
CG	Complex general	Find eigenvalues and eigenvectors (as desired)
CH	Complex symmetric	
RG	Real general	
RGG	Real general generalize $Ax= \lambda Bx$	
RS	Real symmetric	
RSB	Real symmetric band	
RSG	Real symmetric generalize $Ax= \lambda Bx$	
RSGAB	Real symmetric generalize $ABx= \lambda x$	
RSGBA	Real symmetric generalize $BAx= \lambda x$	
RSP	Real symmetric packed	
RST	Real symmetric tridiagonal	
RT	Special real tridiagonal	
BALANC	Real general	Balances matrix and isolates eigenvalues whenever possible
CBAL	Complex general	

Table 4-4. Single-precision EISPACK routines (continued)

Name	Matrix or Decomposition	Purpose
ELMHES ORTHES COMHES COMTH	Real general Complex general	Reduce matrix to upper Hessenberg form
ELTRAN ORTRAN	Real general	Accumulate transformations used in the reduction to upper Hessenberg form done by ELMHES, ORTHES
BALBAK ELMBAK ORTBAK COMBAK CORTB CBABK2 REBAK REBAKB	Real general Complex general	Form eigenvectors by back transforming those of the corresponding matrices determined by BALANC, ELMHES, ORTHES, COMMES, CORTH, and CBAL
TRED1 TRED2 TRED3	Real symmetric	Reduce to symmetric tridiagonal
TRBAK1 TRBAK3	Real symmetric	Form eigenvectors by back transforming those of the corresponding matrices determined by TRED1 or TRED3
IMTQLV IMTQL1 IMTQL2	Symmetric tridiagonal	Find eigenvalues and/or eigenvectors by implicit QL method
RATQR	Symmetric tridiagonal	Find the smallest or largest eigenvalues by rational QR method with Newton corrections

Table 4-4. Single-precision EISPACK routines (continued)

Name	Matrix or Decomposition	Purpose
TQLRAT TQL1 TQL2	Symmetric tridiagonal	Find the eigenvalues by rational QL method Find the eigenvalues and/or eigenvectors by the rational QL or QL method
BISECT TRIDIB TSTURM TINVIT	Symmetric tridiagonal	Find eigenvalues and/or eigenvectors which lie in a specified interval using bisection and/or inverse iteration
FIGI FIGI2	Nonsymmetric tridiagonal	Reduce to symmetric tridiagonal with the same eigenvalues
BAKVEC	Nonsymmetric	Form eigenvectors by back transforming corresponding matrix determined by FIGI
HQR HQR2 COMQR COMQR2	Real upper Hessenberg Complex upper Hessenberg	Find eigenvalues and/or eigenvectors by QR method
INVIT CINVIT	Upper Hessenberg Complex upper Hessenberg	Find eigenvectors corresponding to specified eigenvalues
BANDR	Real symmetric banded	Reduce to a symmetric tridiagonal matrix
BANDV	Real symmetric banded	Find those eigenvectors corresponding to specified eigenvalues using inverse iteration

Table 4-4. Single-precision EISPACK routines (continued)

Name	Matrix or Decomposition	Purpose
BQR	Real symmetric banded	Find eigenvalues using QR algorithm with shifts of origin
MINFIT	Real rectangular	Determine the singular value decomposition $A=USV^T$, forming U^TB rather than U. Householder bidiagonalization and a variant of the QR algorithm are used.
SVD	Real rectangular	Determine the singular value decomposition $A=USV^T$. Householder bidiagonalization and a variant of the QR algorithm are used.
HTRIBK HTRIB3 HTRIDI HTRID3	Complex Hermitian	All eigenvalues and eigenvectors
QZHES QZIT QZVAL QZVEC	Real generalize eigenproblem $Ax = \lambda Bx$	All eigenvalues and eigenvectors
COMLR COMLR2	Complex general	Reduce matrix to upper Hessenberg
REDUC	Real symmetric generalize $Ax= \lambda Bx$	Transforms generalize symmetric eigenproblems to standard symmetric eigenproblems
REDUC2	Real symmetric generalize $ABx= \lambda Bx$ or $BAx= \lambda Bx$	

MATRIX INVERSE AND MULTIPLICATION ROUTINES

The matrix inverse subroutine, MINV, computes the matrix inverse and solves systems of linear equations using the Gauss-Jordan elimination. MXM and MXMA are two optimal matrix multiplication routines, one more general than the other. MXV and MXVA are similar to MXM and MXMA, respectively; however, MXV and MXVA handle the special case of matrix times vector.

MINV computes the determinant and inverse of a square matrix. MINV can also solve several systems of linear equations described by one square matrix and several right-hand sides.

Call from FORTRAN:

```
CALL MINV(ab,n,nd,scratch,det,eps,m,mode)
```

ab	Augmented matrix of the square matrix a and the n x m matrix b of the m right-hand sides for each system of equations to solve. The solution overwrites the corresponding right-hand side. In the calling routine, *ab* must be dimensioned $a(nd,n+m)$.
n	Order of matrix a
nd	Leading dimension of *ab*
scratch	User-defined working storage array of length at least 2**n*
det	Determinant of matrix a
eps	User-defined tolerance for the product of pivot elements
m	>0 Number of systems of linear equations to solve =0 Determinant of a is computed, depending on the value of MODE.
mode	+1 a is overwritten with a^{-1}. =0 a is not saved and a^{-1} is not computed.

MXM computes a matrix times matrix product ($c=ab$) and assumes a skip distance between elements of the matrices to be 1.

Call from FORTRAN:

```
CALL MXM(a,nar,b,nac,c,nbc)
```

a	First matrix of product
nar	Number of rows of matrices *a* and *c*
b	Second matrix of product
nac	Number of columns of matrix *a* and the number of rows of matrix *b*
c	Result matrix
nbc	Number of columns of matrices *b* and *c*

MXV computes a matrix times a vector and assumes a skip distance between elements of the matrix to be 1.

In FORTRAN, MXV would perform the following calculations.

```
      DO 10 I=1, NAR
 10   C(I)=A(I,1)*B(1)+A(I,2)*B(2)+...+A(I,NBR)*B(NBR)
```

Call from FORTRAN:

```
CALL MXV(a,nar,b,nbr)
```

a	Matrix of product
nar	Number of rows of matrices *a* and *c*
b	Vector of product
nbr	Number of elements of vector *b* and the number of columns of matrix *a*
c	Resulting vector

MXMA computes a matrix times matrix product (*c*=*ab*) and allows for arbitrary spacing of matrix elements.

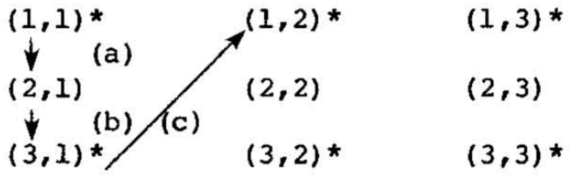
Call from FORTRAN:

```
CALL MXMA(a,na,iad,b,nb,ibd,c,nc,icd,nar,nac,nbc)
```

a	First matrix of product
na	Spacing between column elements of *a*
iad	Spacing between row elements of *a*
b	Second matrix of product
nb	Spacing between column elements of *b*
ibd	Spacing between row elements of *b*
c	Output matrix
nc	Spacing between column elements of *c*
icd	Spacing between row elements of *c*
nar	Number of rows in first operand and result
nac	Number of columns in first operand and number of rows in second operand
nbc	Number of columns in second operand and result

Example 1:

The dimensions of matrix A below are 3x3. Consider the 2x3 submatrix A' marked by asterisks.

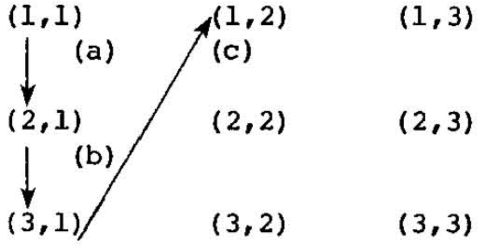


The row spacing of A' (*iad*) is defined as the length of the path through A between two consecutive row elements of A'. In this example, the path is (a) through (c) (*iad*=3).

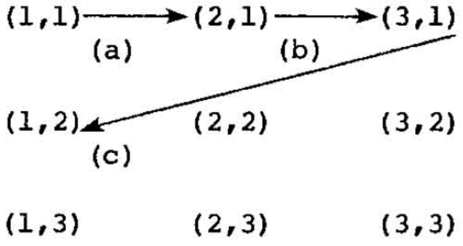
The column spacing of A' (*na*) is defined as the length of the path through A between two consecutive column elements of A'. In this example, the path is (a) through (b) (*na*=2); the number of rows of A' is 2 (*nar*=2); and the number of columns of A' is 3 (*nac*=3).

Example 2:

Consider the matrices below. Let A^T, the transpose of A equal the first operand of a matrix multiply operand. The transpose of a matrix has as its *i*th row the *i*th column of the original matrix.



Matrix A



Matrix A^T

The length of the path between two consecutive column elements of A^T is the same as the length of the path between two consecutive row elements of A. Refer to paths (a) through (c) of both matrices (*na*=3). The length of the path between two consecutive row elements of A^T is the length of the path between two consecutive column elements of A. This path consists of just (a) (*iad*=1). In this example *nar*=3 and *nac*=3.

Therefore, if A is the first operand of a call to MXMA, the following subroutine call is used.

```
CALL MXMA(A,1,3,...)
```

If A^T is the first operand of a call to MXMA, the following subroutine call is used.

```
CALL MXMA(A,3,1,...)
```

MXVA computes a matrix times a vector and allows for arbitrary spacing of matrix elements.

Call from FORTRAN:

```
CALL MXVA(a,na,iad,b,nb,c,nc,nar,nbr)
```

a	First matrix of product
na	Spacing between column elements of *a*
iad	Spacing between row elements of *a*

b	Vector of product
nb	Spacing between elements of *b*
c	Result vector
nc	Spacing between elements of *c*
nar	Number of rows in first operand and number of elements in the result
nbr	Number of columns in first operand and number of elements in the second operand

FAST FOURIER TRANSFORM ROUTINES

These routines apply a Fast Fourier transform. Each routine can compute either a Fourier analysis or a Fourier synthesis. Detailed descriptions, algorithms, performance statistics, and examples of these routines appear in the Complex Fast Fourier Transform Binary Radix Subroutine (CFFT2), CRI publication SN-0203; Real to Complex Fast Fourier Transform Binary Radix Subroutine (RCFFT2), CRI publication SN-0204; and Complex to Real Fast Fourier Transform Binary Radix Subroutine (CRFFT2), CRI publication SN-0206.

Each routine has the same argument list: (*init*, *ix*, *n*, *x*, *work*, *y*).

init	Initialization flag
ix	Analysis/synthesis flag
n	Size of transform
x	Input vector
work	Working storage vector
y	Result vector

The routines are called the first time with $init \neq 0$ and *n* as a power of two in order to initialize the needed sine and cosine tables in the working storage area *work*. Then for each input vector of length *n* (length $(n/2)+1$ for CRFFT2), each routine is called with $init=0$. The sign of IX determines whether a Fourier synthesis or a Fourier analysis is computed. If the sign of *ix* is negative, a synthesis is computed; if positive, an analysis is computed. Table 4-5 shows the size and formats of *x*, *y*, and *work* for each routine.

Table 4-5. Arguments for Fourier transform routines

Argument	CFFT2	RCFFT2	CRFFT2
x	Complex *n*	Real *n*	Complex (*n*/2)+1
work	Complex (5/2)*n*	Complex (3/2)*n*+2	Complex (3/2)*n*+2
y	Complex *n*	Complex (*n*/2)+1	Real *n*

CFFT2 calculates equation 13.

Equation 13:

$$y_k = \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2\pi i}{n} jk)$$

for k=0,1,...,n-1

where x_i i=0,1,...,n-1 are stored in X(I),I=1,N

y_i i=0,1,...,n-1 are stored in Y(I),I=1,N

and the sign of the exponent is determined by SIGN(IX).

Call from FORTRAN:

```
CALL CFFT2(init,ix,n,x,work,y)
```

init	≠0	Generates sine and cosine tables in *work*
	=0	Calculates Fourier transforms using sine and cosine tables of previous call
ix	>0	Calculates Fourier analysis
	<0	Calculates Fourier synthesis
n		Size of Fourier transform; 2^m where $3 \leq m$ for the CRAY X-MP and $2 \leq m$ for the CRAY-1.

x	Input vector. Vector of *n* complex values. Range: $10^{2466}/n \geq x(i) \geq n*(10^{-2466})$ for $i=1,n$.
work	Working storage. Vector of (5/2)*n* complex values.
y	Result vector. Vector of *n* complex values.

NOTE

The input vector *x* can be equivalenced to either *y* or *work*; then the input sequence is overwritten.

RCFFT2 calculates

$$y_k=2 \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2\pi i}{n} jk)$$

for k = 0,1,...,(n/2)

where x_i i=0,1,...,n-1 are stored in X(I),I=1,N
y_i i=0,1,...,n/2 are stored in Y(I),I=1,(N/2)+1

and the sign of the exponent is determined by SIGN(IX).

Call from FORTRAN:

```
CALL RCFFT2(init,ix,n,x,work,y)
```

init	≠0 Generates sine and cosine tables in *work* =0 Calculates Fourier transforms using sine and cosine tables of previous call
ix	>0 Calculates Fourier analysis ≤0 Calculates Fourier synthesis
n	Size of Fourier transform; 2^m where $3 \leq m$.
x	Input vector. Vector of *n* real values Range: $10^{2466}/2*n \geq x(i) \geq 2*n*10^{-2466}$; $i=1,n$.
work	Working storage. Vector, (3/2)*n*+2 complex value.
y	Result vector. Vector of (*n*/2)+1 complex values.

CRFFT2 calculates equation 13 where the x_i elements are complex and $x_j=x_{n-j}$ for j=0,1,...(n/2). Only the first (n/2)+1 elements are stored in X.

Equation 13:

$$y_k = \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2\pi i}{n} jk)$$

for k=0,1,...,n-1

where the x_j elements are complex and are related by $x_j=\bar{x}_{n-j}$

for j=1,2,3,...,(n/2)

Call from FORTRAN:

```
CALL CRFFT2(init,ix,n,x,work,y)
```

init	≠0	Generates sine and cosine tables in *work*
	=0	Calculates Fourier transforms using sine and cosine tables of previous call
ix	>0	Calculates Fourier analysis
	≤0	Calculates Fourier synthesis
n		Size of Fourier transform; 2^m where $3 \leq m$.
x		Input vector. Vector of (*n*/2)+1 complex values Range: $10^{2466}/n \geq x(i) \geq n*10^{-2466}$; *i*=1,*n*.
work		Working storage. Vector, (3/2)*n*+2 complex values.
y		Result vector. Vector of *n* real values.

FILTER SUBROUTINES

These subroutines are intended for filter analysis and design. They also solve more general problems. For detailed descriptions, algorithms, performance statistics, and examples, see Linear Digital Filters for CFT Usage, CRI publication 2240210.

FILTERG computes a convolution of two vectors.

Given:

(a_i)	$i=1,\ldots.,m$	Filter coefficients
(d_j)	$j=1,\ldots.,n$	Data

FILTERG computes the following.

$$o_i = \sum_{j=1}^{m} a_j d_{i+j-1} \qquad i=1,\ldots,n-m+1$$

Call from FORTRAN:

```
CALL FILTERG(a,m,d,n,o)
```

a	Vector of filter coefficients
m	Number of filter coefficients
d	Input data vector
n	Number of data points
o	Output vector

FILTERS computes the same convolution as FILTERG except that it assumes the filter coefficient vector is symmetric.

Given:

(c_i) $i=1,\ldots.,\lceil m/2\rceil$ $(\lceil m/2\rceil=m/2$ for m even and
(d_i) $j=1,\ldots.,n$ $(m+1)/2$ for m odd, called the
ceiling function.)

FILTERS computes the following.

$$\text{m odd:} \quad o_i = \sum_{j=1}^{(m-1)/2} a_j(d_{i+j-1}+d_{i+m-j})$$

$$+a_{\lceil \frac{m}{2} \rceil} \cdot d_{i+\lceil \frac{m}{2} \rceil} \qquad i=1,\ldots,n-m+1$$

$$\text{m even:} \quad o_i = \sum_{j-1}^{m/2} a_j(d_{j+i-1} + d_{i+m-j}) \qquad i=1,\ldots,n-m+1$$

Call from FORTRAN:

```
CALL FILTERS(a,m,d,n,o)
```

a	Symmetric filter coefficient vector
m	M is formally the length of vector A but because A is symmetric ($a_i=a_{m-i+1}$; i=1,...,m), only m/2 elements of A are ever referenced.
d	Input data vector
n	Number of data points
o	Output vector

OPFILT computes the solution to the Weiner-Levinson system of linear equations Ta=b where T is a Toeplitz matrix in which elements are described by the following.

t_{ij}=R(k) for |j-i|+1=k
and k=1,...,n.

Call from FORTRAN:

```
CALL OPFILT(m,a,b,c,r)
```

m	Order of system of equations
a	Output vector of filter coefficients
b	Information auto-correlation vector
c	Scratch vector of length 2*m*
r	Signal auto-correlation vector

NOTE

Although OPFILT solves this matrix equation faster than Gaussian elimination, OPFILT does no pivoting. Therefore, it is less numerically stable than Gaussian elimination unless the matrix T is positive, definite, or diagonally dominant.

Example:

The following system of linear equations can be solved with the call OPFILT (3,A,B,C,R). The array C is one dimension with a length of at least six. (The t_{ij} elements show how the numbers for R are obtained.)

$$\begin{pmatrix} R(1) & R(2) & R(3) \\ R(2) & R(1) & R(2) \\ R(3) & R(2) & R(1) \end{pmatrix} \begin{pmatrix} A(1) \\ A(2) \\ A(3) \end{pmatrix} = \begin{pmatrix} B(1) \\ B(2) \\ B(3) \end{pmatrix}$$

$$\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

GATHER, SCATTER ROUTINES

These subroutines allow the user to gather a vector from a source vector or to scatter a vector into another vector. A third vector of indexes determines which elements are accessed or changed.

GATHER is defined in the following way.

$a_i = b_{j_i}$ where $i=1,\ldots,n$

In FORTRAN:

```
A(I)=B(INDEX(I))
```

where I=1,N

Call from FORTRAN:

```
CALL GATHER(n,a,b,index)
```

n	Number of elements in each vector
a	Output vector
b	Source vector
index	Vector of indexes

SCATTER is defined in the following way.

$a_{j_i}=b_i$ where i=1,...,n

In FORTRAN:

```
A(INDEX(I))=B(I)
```

where I=1,N

Call from FORTRAN:

```
CALL SCATTER(n,a,index,b)
```

n	Number of elements in each vector
a	Output vector
b	Source vector
index	Vector of indexes

SEARCH ROUTINES

Several search routines in $SCILIB have been optimized in CAL. These routines use the vector units and vector mask to quickly find the number or positions of true occurrences in a vector of a given relation. The routines ILLZ and IILZ find the first occurrences. ILSUM counts the number of such occurrences.

Several routines find the positions of a searched for object in a vector. These include: ISRCHEQ, ISRCHNE, ISRCHFLT, ISRCHFLE, ISRCHFGT, ISRCHFGE, ISRCHILT, ISRCHILE, ISRCHIGT, and ISRCHIGE.

The following routines return an indexed array of all positions of an object within a vector. These include: WHENEQ, WHENNE, WHENFLT, WHENFLE, WHENFGT, WHENFGE, WHENILT, WHENILE, WHENIGT, and WHENIGE.

The OSRCHI and OSRCHF routines search ordered arrays for targets.

NUMBER OR SUM OF VALUES WITHIN A VECTOR OR BEFORE AN ELEMENT

IILZ returns the number of zero values before the first nonzero value in an array. When scanning backward (*incl* < 0), this routine starts at the end and moves backward (L(N),L(N + INCL),L(N + 2*INCL),...).

Call from FORTRAN:

kount=IILZ(*n*,*l*,*incl*)

n	Number of elements to process in the vector (N=vector length, if *incl*=1; *n*=vector length/2, if *incl*=2; etc.)
l	Vector operand
incl	Skip distance between elements of the vector operand. For contiguous elements, *incl*=1.

ILLZ returns the number of false values preceding the first true value in a logical vector. When used with an integer or real vector, ILLZ returns the number of positive or zero values preceding the first negative value. When scanning backward (*incl*<0), this routine starts at the end and moves backward (L(N),L(N+INCL),L(N+2*INCL),...).

Call from FORTRAN:

kount=ILLZ(*n*,*l*,*incl*)

n	Number of elements to process in the vector (*n*=vector length, if *incl*=1; *n*=vector length/2, if *incl*=2; etc.)
l	Vector operand
incl	Skip distance between elements of the vector operand. For contiguous elements, *incl*=1.

ILSUM counts the total number of true values in a vector declared LOGICAL. It counts the total number of negative values in a vector declared REAL or INTEGER.

Call from FORTRAN:

kount=ILSUM(*n*,*l*,*incl*)

n	Number of elements to process in the vector (*n*=vector length, if *incl*=1; *n*=vector length/2, if *incl*=2; etc.)
l	Vector operand
incl	Skip distance between elements of the vector operand. For contiguous elements, *incl*=1.

SEARCHING FOR AN OBJECT IN A VECTOR

These functions return the first location in an array that has a true relational value to the target. See table 4-6 for a summary.

Table 4-6. ISRCH routines

Name(*parameter list*)	Description
ISRCHEQ(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is equal to the real target
ISRCHNE(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is not equal to the real target
ISRCHFLT(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is less than the real target
ISRCHFLE(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is less than or equal to the real target
ISRCHFGT(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is greater than the real target
ISRCHFGE(*n*,*array*,*inc*,*target*)	Returns the first location in a real array that is greater than or equal to the real target

Table 4-6. ISRCH routines (continued)

Name(*parameter list*)	Description
ISRCHEQ(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is equal to the integer target
ISRCHNE(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is not equal to the integer target
ISRCHILT(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is less than the integer target
ISRCHILE(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is less than or equal to an integer target
ISRCHIGT(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is not equal to an integer target
ISRCHIGE(*n*,*iarray*,*inc*,*itarget*)	Returns the first location in an integer array that is not equal to an integer target

ISRCHEQ is used when a real array element is equal to a real target, or an integer array is equal to an integer target. ISRCHEQ replaces the ISEARCH routine but has an entry point of ISEARCH as well as ISRCHEQ.

Call from FORTRAN:

```
location=ISRCHEQ(n,array,inc,target)
location=ISRCHEQ(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \leq 0$, then 0 is returned.
array	First element of real array to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array

target	Real value searched for in array. If *target* is not found, then the returned value is n+1.
itarget	Integer value searched for in array

The FORTRAN equivalent follows.

```
      FUNCTION ISRCHEQ(N,ARRAY,INC,TARGET)
      DIMENSION ARRAY(N)
      J=1
      If(INC.LT.0) J=N*(-INC)
      DO 100 I=1.N
        IF(ARRAY(J).EQ.TARGET) GO TO 200
        J=J+INC
 100  CONTINUE
 200  ISRCHEQ=I
      RETURN
      END
```

ISRCHNE is used when the array element (real or integer) is not equal to the target (real or integer).

Call from FORTRAN:

```
location=ISRCHNE(n,array,inc,target)
location=ISRCHNE(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \leq 0$, then 0 is returned.
array	First element of real array to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array. If *target* is not found, then the returned value is n+1.
itarget	Integer value searched for in array.

ISRCHFLT is used when the real array element is less than the real target.

Call from FORTRAN:

```
location=ISRCHFLT(n,array,inc,target)
```

n	Number of elements to be searched. If $n \le 0$, then 0 is returned.
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array. If *target* is not found, then the returned value is *n*+1.

ISRCHFLE is used when the real array element is less than or equal to the real target.

Call from FORTRAN:

location=ISRCHFLE(*n*,*array*,*inc*,*target*)

n	Number of elements to be searched. If $n \le 0$, then 0 is returned.
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array. If *target* is not found, then the returned value is *n*+1.

ISRCHFGT is used when the real array element is greater than the real target.

Call from FORTRAN:

location=ISRCHFGT(*n*,*array*,*inc*,*target*)

n	Number of elements to be searched. If $n \le 0$, then 0 is returned.
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array. If *target* is not found, then the returned value is *n*+1.

ISRCHFGE is used when the real array element is greater than or equal to the real target.

Call from FORTRAN:

```
location=ISRCHFGE(n,array,inc,target)
```

n	Number of elements to be searched. If $n \leq 0$, then 0 is returned.
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array. If *target* is not found, then the returned value is *n*+1.

ISRCHILT is used when the integer array element is less than the integer target.

Call from FORTRAN:

```
location=ISRCHILT(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \leq 0$, then 0 is returned.
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array. If *itarget* is not found, then the returned value is *n*+1.

ISRCHILE is used when the integer array element is less than or equal to the integer target.

Call from FORTRAN:

```
location=ISRCHILE(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \leq 0$, then 0 is returned.

iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array. If *itarget* is not found, then the returned value is $n+1$.

ISRCHIGT is used when the integer array element is greater than the integer target.

Call from FORTRAN:

```
location=ISRCHIGT(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \le 0$, then 0 is returned.
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array. If *itarget* is not found, then the returned value is $n+1$.

ISRCHIGE is used when the integer array element is greater than or equal to the integer target.

Call from FORTRAN:

```
location=ISRCHIGE(n,iarray,inc,itarget)
```

n	Number of elements to be searched. If $n \le 0$, then 0 is returned.
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array. If target is not found, then the returned value is $n+1$.

INDEXED ARRAY OF ALL POSITIONS OF AN OBJECT IN A VECTOR

These routines return all locations in an array that have a true relational value to the target. Table 4-7 summarizes these routines.

Table 4-7. WHEN routines

Name(*parameter list*)	Description
WHENEQ(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are equal to the real target
WHENNE(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are not equal to the real target
WHENFLT(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are less than the real target
WHENFLE(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are less than or equal to the real target
WHENFGT(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are greater than the real target
WHENFGE(*n*,*array*,*inc*,*target*, *index*,*nval*)	Returns all locations in a real array that are greater than or equal to the real target
WHENEQ(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are equal to the integer target
WHENNE(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are equal to the integer target
WHENILT(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are less than the integer target
WHENILE(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are less than or equal to the integer target
WHENIGT(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are greater than the integer target
WHENIGE(*n*,*iarray*,*inc*,*itarget*, *index*,*nval*)	Returns all locations in an integer array that are greater than or equal to the integer target

WHENEQ is used when the real array element is equal to the real target or the integer array is equal to the integer target.

Call from FORTRAN:

```
CALL WHENEQ(n,array,inc,target,index,nval)
CALL WHENEQ(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
itarget	Integer value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

The FORTRAN equivalent follows.

```
      INA=1
      NVAL=0

      IF(INC.LT.0) INA=(-INC)*(N-1)+1

      DO 100 I=1,N

      IF(ARRAY(INA).EQ.TARGET) THEN
        NVAL=NVAL+1
        INDEX(NVAL)=I
      END IF

      INA=INA+INC

  100 CONTINUE
```

WHENNE is used when the real array element is not equal to the real target or the integer array is not equal to the integer target.

Call from FORTRAN:

```
CALL WHENNE(n,array,inc,target,index,nval)
CALL WHENNE(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
itarget	Integer value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENFLT is used when the real array element is less than the real target.

Call from FORTRAN:

```
CALL WHENFLT(n,array,inc,target,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENFLE is used when the real array element is less than or equal to the real target.

Call from FORTRAN:

```
CALL WHENFLE(n,array,inc,target,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENFGT is used when the real array element is greater than the real target.

Call from FORTRAN:

```
CALL WHENFGT(n,array,inc,target,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENFGE is used when the real array element is greater than or equal to the real target.

Call from FORTRAN:

```
CALL WHENFGE(n,array,inc,target,index,nval)
```

n	Number of elements to be searched
array	First element of real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENILT is used when the integer array element is less than the integer target.

Call from FORTRAN:

```
CALL WHENILT(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENILE is used when the integer array element is less than or equal to the integer target.

Call from FORTRAN:

```
CALL WHENILE(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array

itarget	Integer value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENIGT is used when the integer array element is greater than the integer target.

Call from FORTRAN:

```
CALL WHENIGT(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENIGE is used when the integer array element is greater than or equal to the integer target.

Call from FORTRAN:

```
CALL WHENIGE(n,iarray,inc,itarget,index,nval)
```

n	Number of elements to be searched
iarray	First element of integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in array

index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

SEARCH ORDERED ARRAY FOR TARGET

These subroutines search integer arrays for integer targets and search real arrays for real targets.

OSRCHI searches an ordered integer array and returns the index of the first location that contains the target (type integer). Searching always begins at the lowest value in the ordered array. Even if the target is not found, OSRCHI returns the index of the location that would contain the target. The total number of occurrences of the target in the array can also be returned.

Call from FORTRAN:

```
CALL OSRCHI (n,iarray,inc,itarget,index,iwhere,inum)
```

n	Number of elements of the array to be searched
iarray	Beginning address of the integer array to be searched
inc	A positive skip increment indicates an ascending array and returns the index of the first element encountered, starting at the beginning of the array. A negative skip increment indicates a descending array and returns the index of the last element encountered, starting at the beginning of the array.
itarget	Integer target of the search
index	Index of the first location in the searched array that contains the target Exceptional cases: (1) If $n < 1$, *index*=0 (2) If no equal array elements, index=n+1

iwhere Index of the first location in the searched array that would contain the target if it were found in the array. (If the target is found, *index*=*iwhere*.)

Exceptional case: if *n* is less than 1, *iwhere*=0

inum Number of target elements found in the array. For the total number of occurrences of the target in the array, this parameter must be specified nonzero.

OSRCHF searches an ordered real array and returns the index of the first location that contains the target (type real). Searching always begins at the lowest value in the ordered array. Even if the target is not found, OSRCHI returns the index of the location that would contain the target. As an option, the total number of occurrences of the target in the array can also be returned.

Call from FORTRAN:

```
CALL OSRCHF (n,array,inc,target,index,iwhere,inum)
```

n Number of elements of the array to be searched

array Beginning address of the real array to be searched

inc A positive skip increment indicates an ascending array and returns the index of the first element encountered, starting at the beginning of the array.

A negative skip increment indicates a descending array and returns the index of the last element encountered, starting at the beginning of the array.

target Real target of the search

index Index of the first location in the searched array that contains the target

Exceptional cases:
(1) If $n < 1$, *index*=0
(2) If no equal array elements, index=n+1

iwhere Index of the first location in the searched array that would contain the target if it were found in the array. (If the target is found, *index*=*iwhere*.)

Exceptional case: if *n* is less than 1, *iwhere*=0

inum	Number of target elements found in the array. For the total number of occurrences of the target in the array, this parameter must be specified nonzero.

SORT ROUTINE

ORDERS is an internal fixed-length record sort optimized for the Cray computer. It assumes that the *n* records to be sorted are of length *irecLth* and have been stored in an array *data* that has been dimensioned. The ORDERS method and processing are described later.

```
DIMENSION DATA(irecLth,n)
```

ORDERS does not move records within *data* but returns a vector *index* containing pointers to each of the records in ascending order. For example, DATA(1,INDEX(1)) is the first word of the record with smallest key.

Call from FORTRAN:

```
CALL ORDERS(mode,iwork,data,index,n,irecLth,ikeyLth,iradsiz)
```

Although the number of arguments and their interconnections are complicated, careful use can save significant execution time.

mode Integer flag; describes the type of key and indicates an initial ordering of the records.

Upon completion of a call, ORDERS returns an error flag in *mode*. A value equal to the input *mode* value indicates no errors. A value less than 0 indicates an error.

-1	Too few arguments; must be greater than 4.
-2	Too many arguments; must be less than 9.
-3	Number of words per record less than 1 or greater than 2**24
-4	Length of key greater than the record
-5	Radix not equal to 1 or 2
-6	Key less than one byte long

-7 Number of records less than 1 or greater than 2**24

-8 Invalid *mode* input values: must be 0, 1, 2, 10, 11, or 12.

-9 Key length must be eight bytes for real or integer sort.

- 0 The key is binary numbers of length 8**ikeylth*. These numbers are considered positive integers in the range of 0 to $2^{(8*irec
lth)-1}$. (The ordering of ASCII characters is the same as their ordering as positive integers.)
- 1 The key is 64-bit Cray integers. These are twos complement signed integers in the range of -2^{63} to $+2^{63}$. (The key length, if specified, must be eight bytes.)
- 2 The key is 64-bit Cray floating-point numbers. (The key length, if specified, must be eight bytes.)
- 10 The key is the same as *mode*=0, but the array INDEX has an initial ordering of the records (see multipass sorting, later in this section).
- 11 The key is the same as *mode*=1, but the array INDEX has an initial ordering of the records.
- 12 The key is the same as *mode*=2, but the array INDEX has an initial ordering of the records.

iwork User-supplied working storage array of length K where K=257 if *iradsiz*=1, or K=65537 if *iradsiz*=2

data Array dimensioned *ireclth* by N containing the N records of length *ireclth* each. The key in each record starts at the left of the first word of the record and continues *ikeylth* bytes into successive words as necessary. (By offsetting this address, any word within the record can be used as a key. See sort examples at the end of this section.)

index Integer array of length N containing pointers to the records. In *mode*=10, 11, or 12, *index* contains an initial ordering of the records (see multipass sorting, later in this section). On output, *index* contains the ordering of the records; that is, DATA(1,INDEX(I)) is the first word of the record with smallest key and DATA(1,INDEX(N)) is the first word of the record with the largest key.

n Number of records to be sorted. Must be ≥1

irecl th	Length of each record as a number of 64-bit words. Default is 1. *irecl th* is used as a skip for vector loads and stores. Therefore, *irecl th* should be chosen to avoid bank conflicts.
ikeyl th	Length of each key as a number of 8-bit bytes. Default is eight bytes (1 word).
iradsiz	Radix of the sort. *iradsiz* is the number of bytes processed per pass over the records. Default is 1. See section on large radix sorting for *iradsiz*=2.

METHOD

ORDERS uses the radix sort, more commonly known as a bucket or pocket sort. For this sort, the length of the key in bytes determines the number of passes made through all of the records. The method has a linear work factor and is stable in that the original order of records with equal keys is preserved.

ORDERS has the option of processing one or two bytes of the key per pass through the records. This process halves the number of passes through the record but at the expense of increased working storage and overhead per pass (see table 4-8). ORDERS can sort on several keys within a record by using its multipass capability. The first eight bytes of the keys use a radix sort. If the key length is greater than eight bytes and any records have the first eight bytes equal, these records are sorted using a simple bubble sort. Using the bubble sort with many records is time-consuming. Therefore, the multipass option should be used instead.

ORDERS has been optimized in CAL to make efficient use of the vector registers and functional units at each step of a pass through the data. Keys are read into vector registers with a skip through memory of *irecl th*. Therefore, *irecl th* should be chosen to avoid bank conflicts.

Large radix sorting

The number of times the key of each record is read from memory is proportional to *ikeyl th*/*iradsiz*. Using ORDERS with *iradsiz*=2 halves this ratio because two bytes instead of one are processed each time the key is read. One disadvantage of halving the number of passes is the user-supplied working storage array goes from 257 words to 65537 words. Also, a 2-byte pass requires that each pass now use a greater overhead for setup. These two factors favor a 1-byte pass for sorting up to about 5000 records. For more than 5000 records, however, a 2-byte pass is faster.

Multipass sorting

Because the array INDEX can define an ordering of the records, several calls can be made to ORDERS where the order of the records is that of the previous call. *mode*=10, 11, or 12 specifies that the array INDEX contains an ordering from a previous call to ORDERS. This specification allows sorting of text keys that extend over more than one word or keys involving double-precision numbers. (See examples at the end of this section.) Although the length of the key is limited only by the length of the record, up to eight bytes are sorted with the radix sort. The remaining key is sorted using a bubble sort, but only in those records whose keys are equal for the first eight bytes. Therefore, a uniformly distributed key over the first eight bytes of length greater than eight bytes might be sorted faster with a single call with a large IKEYLTH rather than with a multipass call (see table 4-9). Note also that when using the multipass capability, the least significant word must be sorted first.

Tables 4-8 and 4-9 show the processing of one or two bytes of a key per pass through the records for a 16-bank CRAY 1-S Computer System.

Table 4-8. Sort times in seconds for ORDERS

Length of key in bytes				
	N	7	8	15
One byte per pass	2	.00057	.00065	.00065
	5	.00059	.00067	.00067
	10	.00060	.00069	.00070
	50	.00080	.00092	.00095
	100	.00105	.00120	.00125
	500	.00299	.00342	.00370
	1,000	.00543	.00616	.00678
	5,000	.02480	.02824	.03118
	10,000	.04906	.05585	.06173
Two bytes per pass	2	.01515	.01520	.01513
	5	.01510	.01511	.01519
	10	.01507	.01511	.01512
	50	.01522	.01522	.01525
	100	.01540	.01542	.01547
	500	.01651	.01642	.01679
	1,000	.01781	.01786	.01838
	5,000	.02486	.02861	.03156
	10,000	.04216	.04216	.04808

Table 4-9. Sort times in seconds with ASCII key

N	Two passes 8-byte Key	One pass 16-byte Key
2	.00131	.00065
5	.00135	.00068
10	.00140	.00072
50	.00191	.00104
100	.00258	.00145
500	.00796	.00485
1,000	.01470	.00907
5,000	.06874	.04307
10,000	.13602	.08531

Example 1:

```
                 PROGRAM SORT1

C
C     Sort on a 2-word (16-byte) key that is at the beginning of
C     a 5-word (inclusive) record
C
      DIMENSION DATA(5,N)
      DIMENSION INDEX(N)
      DIMENSION WORK(65537)
      N=10000
C
      MODE=0
C
      CALL ORDERS(MODE,WORK,DATA,INDEX,N,5,16,2)
C
C     Print out the keys in increasing alphabetic order
C
      DO 100 I=1,N
         WRITE(6,200) DATA(1,INDEX (I)), DATA(2, INDEX (I))
 200  FORMAT(1X,2A8)
 100  CONTINUE
C
      END
```

Example 2:

```
      PROGRAM SORT2

C
```

```
C     This program uses two calls to ORDERS to completely sort an array
C     of double-precision numbers.  The sign bit of the first
C     word is used to change the second word into a text key that
C     preserves the ordering.  A sort is done on these six bytes of the
C     second word.  (The changes made to the second word are reversed
C     after the call.)  Last, a sort is done on the first word as a
C     real key using the initial ordering from the previous call.
C
      DOUBLE PRECISION DATA(100)
      INTEGER IATA(200)
      EQUIVALENCE(IATA, DATA)
      INTEGER INDEX(100), WORK(257)
      N=12
      DO 5 I=1, N
         DATA(I)=(-1, DO)**10.DO**(-20)*DBLE(RANF())
 5    CONTINUE
C
C     First the second word key is changed
C
      DO 10 I=2, 2*N, 2
         IF(DATA(I/2).LE.0.DO) THEN
           IATA(I)=COMPL(IATA(I))
         ELSE
           IATA(I)=IATA(I)
         ENDIF
 10   CONTINUE
C
C     Sort on second word
C
      CALL ORDERS(0,WORK,IATA(2),INDEX,N, 2, 6, 1)
C
C     Restore second word to original form
C
      DO 20 I=2, 2*N, 2
         IF(DATA(I/2).LE.0.DO) THEN
           IATA(I)=COMPL(IATA(I))
         ELSE
           IATA(I)=IATA(I)
         ENDIF
 20   CONTINUE
C
C     Sort on the first word using the initial ordering
C
      CALL ORDERS(12,SORT,DATA,INDEX,N,2,8,1)
      DO 50 I=1,N
         WRITE(6, 900)I, INDEX(I), DATA(INDEX(I))
 50   CONTINUE
 900  FORMAT(1x, 2I5, 2x, D40.30)
      END
```

INPUT/OUTPUT SUBPROGRAMS

5

INTRODUCTION

The following types of subprograms perform input or output operations on COS datasets.

- FORTRAN I/O routines
- Tape translation routines
- Explicit data conversion routines
- Dataset control routines
- Logical record I/O routines
- Numeric conversion routines
- Random access dataset I/O routines
- Word-addressable I/O routines

FORTRAN I/O ROUTINES

The FORTRAN I/O routines described in this section provide the highest level of user interface with COS datasets. They include formatted and unformatted routines, call-by-address and call-by-value routines, vector and scalar routines, namelist routines, and buffered input and output routines.

NOTE

All formatted I/O is restricted to an input/output list and format specification of not more than 152 characters. To change this default, COMDECK $COMMB in $IOLIB can be modified and an alternate library built.

These routines fall into one of the following categories.

- Initialization
- Data transfer
- Finalization

The routines are named and their functions summarized in table 5-1.

Table 5-1. FORTRAN I/O routines

Operation Sequence	Read Form.	Write Form.	Read Unf.	Write Unf.	Decode	Encode	Read L-d.	Write L-d.
Initialization routines	$RFI	$WFI	$RUI	$WUI	$DFI	$EFI	$RLI	$WLI
Transfer routines call-by-address	$RFA	$WFA	$RUA	$WUA	$DFA	$EFA	$RLA	$WLA
Transfer routines call-by-value	$RFV	$WFV	$RUV	$WUV	$DFV	$EFV		
Termination routines	$RFF	$WFF	$RUF	$WUF	$DFF	$EFF	$RLF	$WLF

Each FORTRAN read/write statement not associated with namelist processing generates a call to an I/O initialization routine and a call to an I/O finalization routine. Between these two calls, I/O list items are processed using transfer routines. These transfer routines are classified as either call-by-value or call-by-address. Each list item generates a corresponding call to one of the two types of routines.

Transfer-by-value is selected for simple variables, constants, expressions, or implied DO lists. Transfer-by-value transfers a single value or a vector of values. Transfer-by-address is selected for an array name as a list item.

INITIALIZATION ROUTINES

Initialization routines process control information lists and set up parameters for the processing of the corresponding transfer routines. These routines, identified by the I suffix, are executed before the transfer routines. No arguments are returned. No initialization is required for namelist I/O or buffered I/O. All initialization routines are called by address.

Input initialization routines

$RFI initializes FORTRAN formatted read.

Call from CAL:

```
CALL $RFI,(arg1,arg2[,arg3][,arg4][,arg5][,arg6])
```

arg_1	Address of unit name or number, or internal file
arg_2	Address of FORMAT specification
arg_3	Address of error exit address, ERR=
arg_4	Address of end exit address, END=
arg_5	Address of IOSTAT parameter
arg_6	Address of record number (direct access)

$RUI initializes FORTRAN unformatted read.

Call from CAL:

```
CALL $RUI,(arg1[,,arg3][,arg4][,arg5][,arg6])
```

arg_1	Address of unit name or number
arg_2	Unused
arg_3	Address of error exit address, ERR=
arg_4	Address of end exit address, END=
arg_5	Address of IOSTAT parameter
arg_6	Address of record number (direct access)

$DFI initializes FORTRAN formatted decode.

Call from CAL:

```
CALL $DFI,(arg1,arg2,arg3)
```

arg_1	Address of record length in Cray characters
arg_2	Address of FORMAT specification
arg_3	Address of input character string

$RLI initializes list-directed reads.

Call from CAL:

```
CALL $RLI,(arg1[,,arg3][,arg4][,arg5])
```

arg_1	Address of unit name or number
arg_2	Unused
arg_3	Address of error exit for ERR=
arg_4	Address of exit address for END=
arg_5	Address of IOSTAT parameter

Output initialization routines

$WFI initializes FORTRAN formatted write.

Call from CAL:

```
CALL $WFI,(arg1,arg2[,arg3][,,arg5][,arg6]
```

arg_1	Address of unit name or number, or internal file
arg_2	Address of FORMAT specification
arg_3	Address of error exit address
arg_4	Unused
arg_5	Address of IOSTAT parameter
arg_6	Address of record number (direct access)

$WUI initializes FORTRAN unformatted write.

Call from CAL:

```
CALL $WUI,(arg1[,,arg3][,,arg5][,arg6])
```

arg_1	Address of unit name or number
arg_2	Unused
arg_3	Address of error exit address
arg_4	Unused
arg_5	Address of IOSTAT parameter
arg_6	Address of record number (direct access)

$EFI initializes FORTRAN formatted encode.

Call from CAL:

CALL $EFI,($arg_1$,$arg_2$,$arg_3$)

arg_1	Address of record length in Cray characters
arg_2	Address of FORMAT specification
arg_3	Address of output character string

$WLI initializes list-directed writes.

Call from CAL:

CALL $WLI,($arg_1$[,,$arg_3$][,,$arg_5$])

arg_1	Address of unit name or number
arg_2	Unused
arg_3	Address of error exit for ERR=
arg_4	Unused
arg_5	Address of IOSTAT parameter

TRANSFER ROUTINES

Read and write transfer routines move data between user locations and the system I/O buffer area allocated to a dataset and associated with a particular I/O unit. Encode and decode transfer routines transfer data between user locations and a user-supplied buffer. The user-supplied buffer contains eight characters per word and has no I/O unit association. All dataset processing by these routines is sequential. Transfer routine names are identified by the suffix A for call-by-address routines and by the suffix V for call-by-value routines.

Each formatted, unformatted, and buffered transfer routine has eight entry points. Each entry point corresponds to the type of data specified in the I/O list and is the name of the routine (*xnam*) plus an increment parcel value. Below is a list of the entry points showing the FORTRAN data type accommodated:

Entry point	Type of data
xnam+0	Typeless (Boolean) or no I/O list or type checking present

Entry point	Type of data
xnam+3	Integer
xnam+6	Real
xnam+9	Double-precision
xnam+12	Complex
xnam+15	Logical
xnam+18	Character
xnam+21	Short integer

If an increment parcel value is omitted, typing is determined from the format specification edit descriptor. The increment value can be omitted for all data types except double-precision. For complex values, however, if the increment value is omitted (or if *xnam*+6 is specified), two calls must be made, one for the real portion and one for the imaginary portion. In these cases, the complex number is treated as two real numbers.

Format specifications identified for initialization routines and used by transfer routines are described in the FORTRAN (CFT) Reference Manual, CRI publication SR-0009.

Restrictions on the format specifications for integer, logical and real variables can be relaxed by using SEGLDR and its EQUIV options. See the FORTRAN (CFT) Reference Manual, CRI publication SR-0009 for details.

Acknowledgement of the reading of an end-of-file (EOF) must occur before initiating another read operation on the same unit. Acknowledgement can be made by providing an EOF exit address, or by writing, rewinding, or backspacing the dataset.

Buffered I/O is a form of data transfer allowing the execution of other statements to proceed simultaneously with the actual transfer. The number of words for one data transfer is represented by (*lwa*-*fwa*+1). If the remaining words in the record are to be skipped, full record mode must be specified. Full record mode resumes transferring at the beginning of the next record. If the rest of the record is to be transferred, partial record mode must be specified.

Formatted and unformatted input transfer routines

The following transfer routines are called by address.

$RFA reads FORTRAN formatted data.
$RUA reads FORTRAN unformatted data.
$DFA decodes FORTRAN formatted data.

Call from CAL:

```
CALL $RFA+offset,(arg1,arg2,arg3)
CALL $RUA+offset,(arg1,arg2,arg3)
CALL $DFA+offset,(arg1,arg2,arg3)
```

arg_1	First word address destination
arg_2	Address of word count
arg_3	Address of increment between destination addresses

$RLA reads list-directed data by address.

Call from CAL:

```
CALL $RLA+offset,(arg1,arg2,arg3)
```

arg_1	First word address of input
arg_2	First word address of item count
arg_3	First word address of increment between items

The following transfer routines are called by value.

$RFV, $RFV% read FORTRAN formatted data.
$RUV, $RUV% read FORTRAN unformatted data.
$DFV, $DFV% decode FORTRAN formatted data.

Call from CAL:

```
CALLV $RFV+offset
CALLV $RUV+offset
CALLV $DFV+offset
```

Exit:

(S1)	Requested value
(S2)	Second value if transfer is for double-precision or complex values

%$RFV% reads FORTRAN formatted, vectorized data.
%$RUV% reads FORTRAN unformatted, vectorized data.

Call from CAL:

```
CALLV %RFV%+offset
CALLV %RUV%+offset
```

Entry:

(VL)	Vector length

Exit:

(V1)	(VL) requested values
(V2)	(VL) second values if transfer is for double-precision or complex values

Buffered input transfer routines

$RB performs FORTRAN read buffered operation.

Call from CAL:

```
CALL $RB+offset,(arg1,arg2,arg3,arg4)
```

Entry:

arg_1	Address of unit name or number
arg_2	Address of mode specifier (mode<0 indicates partial record transfer; mode≥0 indicates full record transfer).
arg_3	First word address of destination
arg_4	Last word address of destination

Exit:

Transfer from unit to first word of destination is initiated. Control is returned to the calling program. To test for completion of transfer, the user should issue a call to the routine UNIT or LENGTH.

Namelist input transfer routines

$RNL reads FORTRAN namelist.

Call from CAL:

```
CALL $RNL,(arg1,arg2[,arg3,][,arg4])
```

arg_1	Address of unit name or number
arg_2	Address of NAMELIST group entries
arg_3	Address of ERR=address
arg_4	Address of END=address

Namelist group entries for variables consist of a group name (one word) plus a 2-word entry. Namelist group entries for arrays consist of a group name (one word) plus (n+2) words, where n is the number of dimensions in the array. The first word of the list is always the group name. The order of the variable and array entries in the list is the same as specified on the NAMELIST statement by the user. Figure 5-1 shows the group name, figure 5-2 shows the variable entry, and figure 5-3 shows the array entry.

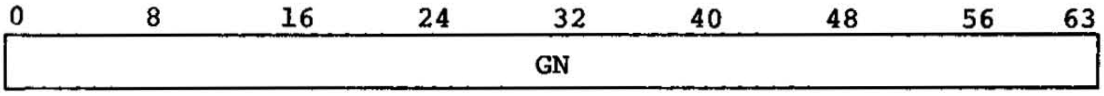


Figure 5-1. Group name

Field	Bits	Description
GN	0-63	8-bit ASCII name, left-justified, zero-filled

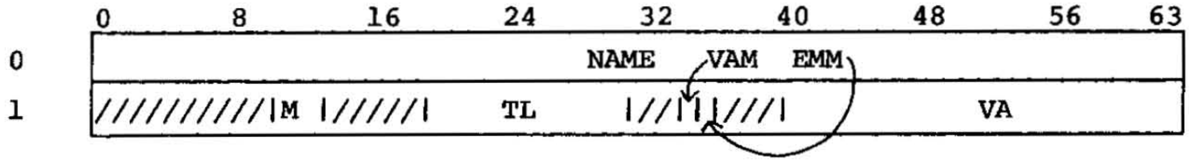


Figure 5-2. Variable entry

Field	Word	Bits	Description
NAME	0	0-63	8-bit ASCII character name, left-justified, zero-filled
Unused	1	0-10	Zero-filled
M	1	11-13	Mode. Number of dimensions in the array (1-7)
Unused	1	14-19	Zero-filled
TL	1	20-31	Internal CFT type and length description:

Field	Word	Bits	Description
			TL **Description** 1077 Logical 4027 24-bit integer 4077 64-bit integer 6077 Real 6177 Complex 7177 Double-precision
Unused	1	32-33	Zero-filled
VAM	1	34	=0 if variable address is actual address =1 if variable address is stack offset from B03
EMM	1	35	Value is 1 if variable address is 24 bits
Unused	1	36-39	Zero-filled
VA	1	40-63	Variable address

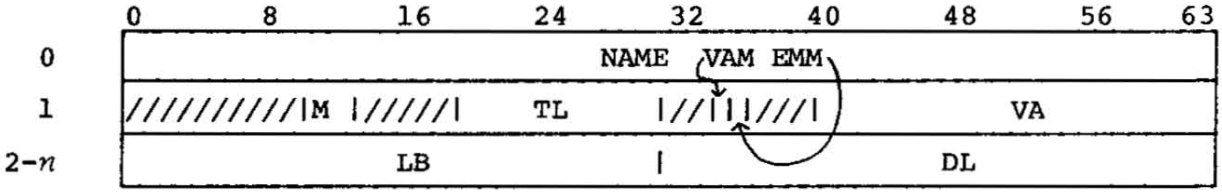


Figure 5-3. Array entry

Field	Word	Bits	Description
NAME	0	0-63	8-bit ASCII character name, left-justified, zero-filled
Unused	1	0-10	Zero-filled
M	1	11-13	Mode. Number of dimensions in the array (1-7)
Unused	1	14-19	Zero-filled
TL	1	20-31	Internal CFT type and length description:

Field	Word	Bits	Description
			TL Description 1077 Logical 4027 24-bit integer 4077 64-bit integer 6077 Real 6177 Complex 7177 Double-precision
Unused	1	32-33	Zero-filled
VAM	1	34	=0 if first word address of array is actual address =1 if first word address of array is stack offset from B03
EMM	1	35	=0 if first word address of array is 22 bits long. Top 2 bits are assumed zero. =1 if first word address of array is 24 bits long
Unused	1	35-39	Zero-filled
FWA	1	40-63	First word address of array
LB	2-*n*	1-31	Lower bound of array dimension
DL	2-*n*	33-63	Dimension length

Formatted and unformatted output transfer routines

The following transfer routines are called by address.

$WFA writes FORTRAN formatted data.
$WUA writes FORTRAN unformatted data.
$EFA encodes FORTRAN formatted data.

Call from CAL:

```
CALL $WFA+offset,(arg1,arg2,arg3)
CALL $WUA+offset,(arg1,arg2,arg3)
CALL $EFA+offset,(arg1,arg2,arg3)
```

arg_1	First word address destination
arg_2	Address of word count
arg_3	Address of increment between destination addresses

$WLA writes list-directed data by address.

Call from CAL:

```
CALL $WLA+offset,(arg1,arg2,arg3)
```

arg_1	First word address of input
arg_2	First word address of item count
arg_3	First word address of increment between items

The following transfer routines are called by value.

$WFV, $WFV% write FORTRAN formatted data.
$WUV, $WUV% write FORTRAN unformatted data.
$EFV, $EFV% encode FORTRAN formatted data.

Call from CAL:

```
CALLV $WFV+offset
CALLV $WUV+offset
CALLV $EFV+offset
```

(S1)	Word to be written or encoded
(S2)	Second word if double-precision or complex values

%$WFV% writes FORTRAN formatted, vectorized data.
%$WUV% writes FORTRAN unformatted, vectorized data.

Call from CAL:

```
CALLV %$WFV%+offset
CALLV %$WUV%+offset
```

(VL)	Vector length
(V1)	(VL) requested values
(V2)	Second values if transfer is for double-precision or complex values

$WLV writes list-directed data by value.

Call from CAL:

```
CALLV $WLV+offset
```

(S1)	Word to be written
(S2)	Second word if double-precision or complex

Buffered output transfer routines

$WB performs FORTRAN write buffered operation.

Call from CAL:

```
CALL $WB+offset,(arg1,arg2,arg3,arg4)
```

Entry:

arg_1	Address of unit name or number
arg_2	Address of mode specifier (mode<0 indicates partial record transfer; mode≥0 indicates full record transfer).
arg_3	First word address of destination
arg_4	Last word address of destination

Exit:

Transfer from unit to first word of destination is initiated. Control is returned to the calling program. To test for completion of transfer, issue a call to the routine UNIT or LENGTH.

Namelist output transfer routines

$WNL writes FORTRAN namelist.

Call from CAL:

```
CALL $WNL,(arg1,arg2[,arg3])
```

arg_1	Address of unit name or number
arg_2	Address of NAMELIST group entries
arg_3	Address of ERR=address

FINALIZATION ROUTINES

Finalization routines, suffixed by F, terminate a record and clear the control information list parameters set up by the corresponding initialization routines. No arguments are required for entry; no arguments are returned. All linkage is call-by-address. Finalization routines are unnecessary for namelist I/O and buffered I/O.

Input finalization routines

$RFF finalizes FORTRAN formatted read.

Call from CAL:

```
CALL $RFF
```

$RUF finalizes FORTRAN unformatted read.

Call from CAL:

```
CALL $RUF
```

$DFF finalizes FORTRAN formatted decode.

Call from CAL:

```
CALL $DFF
```

$RLF finalizes list-directed read.

Call from CAL:

```
CALL $RLF
```

Output finalization routines

$WFF finalizes FORTRAN formatted write.

Call from CAL:

```
CALL $WFF
```

$WUF finalizes FORTRAN unformatted write.

Call from CAL:

```
CALL $WUF
```

$WLF finalizes list-directed write.

Call from CAL:

```
CALL $WLF
```

$EFF finalizes FORTRAN formatted encode.

Call from CAL:

```
CALL $EFF
```

TAPE TRANSLATION ROUTINES

The tape translation routines provide for reading and writing tapes that have been written or are going to be read on computers with different character sets or data formats from those of the Cray computer. Through the ACCESS or ASSIGN control statement (refer to the CRAY-OS Version 1 Reference Manual, publication SR-0011) FORTRAN users can provide foreign tape file characteristics. These parameters are used by the run-time library to correctly translate the dataset. Support is supplied through FORTRAN's formatted, unformatted, and buffered I/O.

A formatted or unformatted transfer routine is called once for each variable in the I/O list. (See transfer routines, this section.) The transfer routine then calls a buffer management routine. The buffer management routine makes a system request for physical I/O when appropriate and processes the COS block and record control words (RCW) if the dataset is in COS blocked format.

Buffer management routines call the record format management routines one or more times for each variable. Record format management routines keep track of the current logical record within the I/O buffer and determine the location of the requested variable within the logical record. They process a partial record and return a status indicating whether the end of the logical record has been reached. The record format management routines call the data format routines once for each element of the requested variable. The data format routines translate between the internal COS representation of the variable and the representation of the variable in the logical record.

NOTE

Some tape translation routines are included in all load modules performing I/O since the code must be included at load time but the dataset characteristics are not known until run time.

If a user or site does not use foreign data types, comdeck $COMFD in $IOLIB can be modified and an alternate library built. Either IBM, CDC, or both forms of conversions can be disabled; the corresponding code is never loaded. Use of a disabled format is not detected; no conversion takes place.

BUFFER MANAGEMENT ROUTINES

The buffer management routines fill and empty the tape buffer, maintain information in the Dataset Parameter Table (DSP) to be saved between READ or WRITE statements, and determine which record translation routine is to be called.

Input buffer management routines

RUTI initializes buffer management.

Call from CAL:

```
CALL RUTI,(arg1)
```

arg_1 DSP address

RUTD% reads data.

Call from CAL:

```
CALLV RUTD%
```

(S1)	First word address of data
(S2)	Number of data items
(S3)	Increment between data items
(S4)	Data type as defined by common deck $COMDT in $IOLIB
(S5)	Length of item in bytes (type character only)

RUTF finalizes buffer management.

Call from CAL:

```
CALL RUTF
```

<u>Output buffer management routines</u>

WUTI initializes buffer management.

Call from CAL:

```
CALL WUTI,(arg)
```

arg	DSP address

WUTD% writes data.

Call from CAL:

```
CALLV WUTD%
```

(S1)	First word address of data
(S2)	Number of data items
(S3)	Increment between data items
(S4)	Data type as defined by common deck $COMDT in $IOLIB
(S5)	Length of item in bytes (type character only)

WUTF finalizes buffer management.

Call from CAL:

```
CALL WUTF
```

RECORD FORMAT MANAGEMENT ROUTINES

Record format management routines keep track of the particular vendor's record and block formats. These routines move the translated bits to or from the tape buffer and format them according to the record and block definitions. These routines are called by address.

Input record format management routines

$IBMI reads IBM file format.
$CDCI reads CDC file format.

Output record format management routines

$IBMO writes IBM file format.
$CDCO writes CDC file format.

DATA FORMAT MANAGEMENT ROUTINES

Data format management routines translate the internal format of the particular variable type. They are accessed by call-by-value subprogram linkage.

The naming convention for data format management routine is $*STD*.

S	System; codes are I (IBM), C (CDC).
T	Variable type and size; for example, I16 is a 16-bit integer.
D	Processing direction is I (input) or O (output).

Input data format management routines

%II16I translates 16-bit IBM integer to 24-bit Cray integer on input.

%II32I translates 32-bit IBM integer to 64-bit Cray integer on input.

%IL8I translates 8-bit IBM logical to 64-bit Cray logical on input.

%IF32I translates 32-bit IBM floating-point to 64-bit Cray floating-point on input.

%ID64I translates 64-bit IBM double-precision floating-point to 128-bit Cray double-precision floating-point on input.

%ICHRI translates 8-bit IBM EBCDIC character to 8-bit ASCII character on input.

%IC64I translates 64-bit IBM complex floating-point to 128-bit Cray complex floating-point on input.

CI60I% translates CDC 60-bit integer to 64-bit Cray integer on input.

CF60I% translates CDC 60-bit floating-point to 64-bit Cray floating-point on input.

CD120I% translates CDC 120-bit double-precision to 128-bit Cray double-precision floating-point on input.

CC120I% translates CDC 120-bit complex number to 128-bit Cray complex floating-point on input.

CCHRI% translates CDC 6-bit display code to 8-bit ASCII character on input.

CL60I% translates CDC 60-bit logical to 64-bit Cray logical on input.

<u>Output data management format routines</u>

%II16O translates 16-bit IBM integer to 24-bit Cray integer on output.

%II32O translates 32-bit IBM integer to 64-bit Cray integer on output.

%IL8O translates 8-bit IBM logical to 64-bit Cray logical on output.

%IF32O translates 32-bit IBM floating-point to 64-bit Cray floating-point on output.

%ID64O translates 64-bit IBM double-precision floating-point to 128-bit Cray double-precision floating-point on output.

%ICHRO translates 8-bit IBM EBCDIC character to 8-bit ASCII character on output.

%IC64O translates 64-bit IBM complex floating-point to 128-bit Cray complex floating-point on output.

CI60O% translates 64-bit Cray integer to CDC 60-bit integer on output.

CF60O% translates 64-bit Cray floating-point to CDC 60-bit floating-point on output.

CD120O% translates 128-bit Cray double-precision floating-point to CDC 120-bit double-precision on output.

CC120O% translates 128-bit Cray complex floating-point to CDC 120-bit complex number on output.

CCHRO% translates 8-bit ASCII character to CDC 6-bit display code on output.

CL60O% translates 64-bit Cray logical to CDC 60-bit logical on output.

EXPLICIT DATA CONVERSION

The explicit data conversion routines described in this subsection are subprograms that allow data translation between Cray internal representations and other vendors' data types.

IBM SINGLE-PRECISION TO CRAY SINGLE-PRECISION ROUTINE

The USSCTC subroutine converts IBM 32-bit floating-point numbers into Cray 64-bit single-precision numbers.

Call from FORTRAN:

CALL USSCTC(*fpn*,*isb*,*dest*,*num*[,*inc*])

fpn	Variable or array of any type or length containing IBM 32-bit floating point numbers to convert
isb	Byte number to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *fpn*.
dest	Variable or array of type real to contain the converted values

num	Number of IBM floating-point numbers to convert. Type integer variable, expression, or constant.
inc	Memory increment for storing the conversion results in *dest*. Optional parameter of type integer variable, expression, or constant. Default value is 1.

IBM DOUBLE-PRECISION TO CRAY SINGLE-PRECISION ROUTINE

The USDCTC subroutine converts IBM 64-bit floating-point numbers into Cray 64-bit single-precision numbers.

Call from FORTRAN:

```
CALL USDCTC(dpn,isb,dest,num[,inc])
```

dpn	Variable or array of any type or length containing IBM 64-bit floating-point numbers to convert
isb	Byte number within *dpn* to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dpn*.
dest	Variable or array of type real to contain the converted values
num	Number of IBM 64-bit floating-point numbers to convert. Type integer variable, expression, or constant.
inc	Memory increment for storing the conversion results in *dest*. Optional parameter of type integer variable, expression, or constant. Default value is 1.

IBM INTEGER TO CRAY INTEGER ROUTINE

The USICTC subroutine converts both IBM INTEGER*2 and INTEGER*4 numbers into Cray 64-bit integer numbers.

Call from FORTRAN:

```
CALL USICTC(in,isb,dest,num,len[,inc])
```

in	Variable or array of any type or length containing IBM INTEGER*2 or INTEGER*4 numbers to convert
isb	Byte number to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *in*.
dest	Variable or array of type integer to contain the converted values
num	Number of IBM numbers to convert. Type integer variable, expression, or constant.
len	Size of the IBM numbers to convert. This value must be 2 or 4. A value of 2 indicates input integers are integer*2 (16-bit). A value of 4 indicates input integers are integer*4 (32-bit). Type integer variable, expression, or constant.
inc	Memory increment for storing the conversion results in *dest*. Optional parameter of type integer variable, expression, or constant. Default value is 1.

EBCDIC TO ASCII ROUTINE

The USCCTC subroutine converts IBM EBCDIC data into ASCII data. The same array can be specified for output as for input only if *isb* = 1 and *npw* = 8.

Call from FORTRAN:

```
CALL USCCTC(src,isb,dest,num,npw[,iuc])
```

src	Variable or array of any type or length containing IBM EBCDIC data to convert
isb	Byte number to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
dest	Variable or array of any type or length to contain the ASCII data.
num	Number of IBM EBCDIC characters to convert. Type integer variable, expression, or constant.

npw	Number of characters per word generated in *dest*. The *npw* characters are left-justified and blank-filled in each word of *dest*. Type integer variable, expression, or constant. Value must be from 1 to 8.
iuc	A value of nonzero specifies lowercase characters (a-z) to be translated to uppercase (A-Z). A value of 0 results in no case translation. Optional parameter of type integer variable, expression, or constant. Default is no case translation.

IBM PACKED DECIMAL FIELD TO INTEGER ROUTINE

The USPCTC subroutine converts a specified number of bytes of an IBM packed decimal field to a 64-bit integer field. The input field must be a valid packed decimal number less than 16 bytes long, of which only the rightmost 15 digits are converted.

Call from FORTRAN:

CALL USPCTC(*src*,*isb*,*num*,*ian*)

src	Variable or array of any type or length containing a valid IBM packed decimal field
isb	Byte number to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
num	Number of bytes to convert. Type integer variable, expression, or constant.
ian	Returned integer result

IBM LOGICAL TO CRAY LOGICAL ROUTINE

The USLCTC subroutine converts both IBM LOGICAL*1 and LOGICAL*4 values into Cray 64-bit logical values.

Call from FORTRAN:

CALL USLCTC(*src*,*isb*,*dest*,*num*,*len*[,*inc*])

src	Variable or array of any type or length containing IBM LOGICAL*1 or LOGICAL*4 values to convert.
isb	Byte number to begin the conversion. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
dest	Variable or array of type logical to contain the converted values
num	Number of IBM logical values to be converted. Type integer variable, expression, or constant.
len	Size of the IBM logical values to convert. This value must be 1 or 4. A value of 1 indicates input logical values are LOGICAL*1 (8-bit). A value of 4 indicates input logical values are LOGICAL*4 (32-bit). Type integer variable, expression, or constant.
inc	Memory increment for storing the conversion results in *dest*. Optional parameter of type integer variable, expression, or constant. Default value is 1.

CRAY SINGLE-PRECISION TO IBM SINGLE-PRECISION ROUTINE

The USSCTI subroutine converts Cray 64-bit single-precision floating-point numbers to IBM 32-bit single-precision floating-point numbers. Numbers that produce an underflow when converted to IBM format are converted to 32 binary zeros. Numbers that produce an overflow when converted to IBM format are converted to the largest IBM floating-point representation with the sign bit set if negative. An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. No such indication is given for underflow.

Call from FORTRAN:

```
CALL USSCTI(fpn,dest,isb,num,ier[,inc])
```

fpn	Variable or array of any length and type real, containing Cray 64-bit, single-precision, floating-point numbers to convert
dest	Variable or array of type real to contain the converted values

isb	Byte number at which to begin storing the converted results. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of Cray floating-point numbers to convert. Type integer variable, expression, or constant.
ier	Overflow indicator of type integer. Value is 0 if all Cray values convert to IBM values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.
inc	Memory increment for fetching the number to be converted. Optional parameter of type integer variable, expression, or constant. Default value is 1.

CRAY SINGLE-PRECISION TO IBM DOUBLE-PRECISION ROUTINE

The USDCTI subroutine converts Cray 64-bit single-precision floating-point number to IBM 64-bit double-precision floating-point numbers. Precision is extended by introducing 8 more bits into the rightmost byte of the fraction from the Cray number being converted. Numbers that produce an underflow when converted to IBM format are converted to 64 binary zeros. Numbers that produce an overflow when converted to IBM format are converted to the largest IBM floating-point representation with the sign bit set, if negative. An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. No such indication is given for underflow.

Call from FORTRAN:

```
CALL USDCTI(fpn,dest,isb,num,ier[,inc])
```

fpn	Variable or array of any length and of type real, containing Cray single-precision floating-point numbers to convert
dest	Variable or array of type real to contain the converted values
isb	Byte number at which to begin storage of the converted results. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of Cray floating-point numbers to convert. Type integer variable, expression, or constant.

ier	Overflow indicator of type integer. Value is 0 if all Cray values are converted to IBM values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.
inc	Memory increment for fetching the number to be converted. Optional parameter of type integer variable, expression, or constant. Default value is 1.

CRAY INTEGER TO IBM INTEGER ROUTINE

The USICTI subroutine converts Cray 64-bit integer numbers into either IBM INTEGER*2 or INTEGER*4 numbers. Numbers that produce an overflow when converted to IBM format are converted to the largest IBM integer representation with the sign bit set if negative. An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow.

Call from FORTRAN:

```
CALL USICTI(in,dest,isb,num,len,ier[,inc])
```

in	Variable or array of any length and type integer, containing Cray integer numbers to convert
dest	Variable or array of type integer to contain the converted values
isb	Byte number at which to begin storing the converted results. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of Cray integers to convert. Type integer variable, expression, or constant.
len	Size of the IBM result numbers. This value must be 2 or 4. A value of 2 indicates output integers are INTEGER*2 (16-bit). A value of 4 indicates output integers are INTEGER*4 (32-bit). Type integer variable, expression, or constant.

ier	Overflow indicator of type integer. Value is 0 if all Cray values are converted to IBM values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.
inc	Memory increment for fetching the number to be converted. Optional parameter of type integer variable, expression, or constant. Default value is 1.

ASCII TO EBCDIC ROUTINE

The USCCTI subroutine converts ASCII data to IBM EBCDIC data. All unprintable characters are converted to blanks. The same array can be specified for output as for input only if *isb* = 1 and *npw* = 8.

Call from FORTRAN:

```
CALL USCCTI(src,dest,isb,num,npw[,inc])
```

src	Variable or array of any type or length containing ASCII data, left justified, in Cray words to convert.
dest	Variable or array of any type or length to contain the IBM EBCDIC data
isb	Byte number at which to begin generating EBCDIC characrters in *dest*. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of ASCII characters to convert. Type integer variable, expression, or constant.
npw	Number of characters per word selected from *src*. Value must be from 1 to 8. Type integer variable, expression, or constant.
inc	A value of nonzero specifies lowercase characters (a-z) to be translated to uppercase (A-Z). A value of 0 results in no case translation. Optional parameter of type integer variable, expression, or constant. Default is no case translation.

INTEGER TO IBM PACKED DECIMAL FIELD ROUTINE

The USICTP subroutine converts a Cray 64-bit integer value to an IBM-packed decimal field. If the input value contains more digits than can be stored in *num* bytes, the leftmost digits are not converted.

Call from FORTRAN:

```
CALL USICTP(ian,dest,isb,num)
```

ian	Cray integer number to be converted to an IBM-packed decimal field. Type integer variable, expression, or constant.
dest	Variable or array of any type or length to contain the generated packed field
isb	Byte number within *dest* specifying beginning location for storage. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of bytes to be stored. Type integer variable, expression, or constant.

CRAY LOGICAL TO IBM LOGICAL ROUTINE

The USLCTI subroutine converts Cray logical values into either IBM LOGICAL*1 or LOGICAL*4 values.

Call from FORTRAN:

```
CALL USLCTI(src,dest,isb,num,len[,inc])
```

src	Variable or array of any length and of type logical, containing Cray logical values to convert
dest	Variable or array of any type or length to contain the converted values
isb	Byte number within *src* specifying beginning location for storage. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.

num	Number of Cray logical values to convert. Type integer variable, expression, or constant.
len	Size of the IBM logical result value. This value must be 1 or 4. A value of 1 indicates output logicals are LOGICAL*1 (8-bit). A value of 4 indicates output logicals are LOGICAL*4 (32-bit). Type integer variable, expression, or constant.
inc	Memory increment for fetching the number to be converted. Optional parameter of type integer variable, expression, or constant. Default value is 1.

UNPACK 60-BIT WORDS ROUTINE

The U6064 subroutine unpacks 60-bit words from Cray 64-bit words.

Call from FORTRAN:

```
CALL U6064(src,isb,dest,num)
```

src	Variable or array of any type or length containing 60-bit words as a continuous stream of data
isb	Bit location that is the leftmost storage location for the 60-bit words. Bit position is counted from the left to right with the leftmost bit 0. Type integer variable, expression, or constant.
dest	Destination array of any type to contain the unpacked 60-bit words, left-justified and zero-filled, in a Cray 64-bit word
num	Number of 60-bit words to unpack. Generates this many elements of *dest*. Type integer variable, expression, or constant.

PACK 60-BIT WORDS ROUTINE

The P6460 subroutine packs 60-bit words into Cray 64-bit words.

Call from FORTRAN:

```
CALL P6460(src,dest,isb,num)
```

src	Variable or array of any type or length containing 60-bit words, left-justified in a Cray 64-bit word
dest	Destination array of any type to contain the packed 60-bit words as a continuous stream of data
isb	Bit location that is the leftmost storage location for the 60-bit words. Bit position is counted from the left to right with the leftmost bit 0. Type integer variable, expression, or constant.
num	Number of 60-bit words to pack. Reads this many elements of *src*. Type integer variable, expression, or constant.

PACK 32-BIT WORDS ROUTINE

The P32 subroutine packs 32-bit words into Cray 64-bit words.

Call from FORTRAN:

```
CALL P32(src,dest,num)
```

src	Variable of any type or length containing 32-bit words, left-justified in a Cray 64-bit word
dest	Destination array of any type to contain the packed 32-bit words as a continuous stream of data
num	Number of 32-bit words to pack. Reads this many elements of *src*. Type integer variable, expression, or constant.

UNPACK 32-BIT WORDS ROUTINE

The U32 subroutine unpacks 32-bit words from Cray 64-bit words.

Call from FORTRAN:

```
CALL U32(src,dest,num)
```

src	Variable or array of any type or length containing 32-bit words as a continuous stream of data. Unpacking always starts with the leftmost bit of *src*.

dest	Destination array of any type to contain the unpacked 32-bit words, left-justified and zero-filled, in a Cray 64-bit word.
num	Number of 32-bit words to unpack. Generates this many elements of *dest*. Type integer variable, expression, or constant.

CDC INTEGER TO CRAY INTEGER ROUTINE

The INT6064 subroutine converts CDC 60-bit integer numbers to Cray integer numbers.

Call from FORTRAN:

```
CALL INT6064(src,idest,num)
```

src	Variable or array of any type or length containing CDC 60-bit integers, left-justified in a Cray 64-bit word
idest	Variable or array of type integer to contain the converted values
num	Number of CDC integers to convert. Type integer variable, expression, or constant.

CDC SINGLE-PRECISION TO CRAY SINGLE-PRECISION ROUTINE

The FP6064 subroutine converts CDC 60-bit single-precision numbers to Cray 64-bit single-precision numbers.

Call from FORTRAN:

```
CALL FP6064(fpn,dest,num)
```

fpn	Variable or array of any type or length containing CDC 60-bit, single-precison numbers, left-justified in a Cray 64-bit word
dest	Variable or array of type real to contain the converted Cray 64-bit, single-precision numbers
num	Number of CDC single-precision numbers to convert. Type integer variable, expression, or constant.

CDC DISPLAY CODE CHARACTER TO ASCII CHARACTER ROUTINE

The DSASC subroutine converts CDC display code characters to ASCII data.

Call from FORTRAN:

```
CALL DSASC(src,sc,dest,num)
```

src	Variable or array of any type or length containing CDC display code characters (64 character set), left-justified in Cray 64-bit word. Contains a maximum of 10 display code characters per word.
sc	Display code character position to begin the conversion. Leftmost position is 1.
dest	Variable or array of any type or length to contain the converted ASCII data. Results are packed 8 characters per word.
num	Number of CDC display code characters to convert. Type integer variable, expression, or constant.

CRAY INTEGER TO CDC INTEGER ROUTINE

The INT6460 subroutine converts Cray 64-bit integer numbers to CDC 60-bit integer numbers.

Call from FORTRAN:

```
CALL INT6460(in,idest,num)
```

in	Variable or array of any length and of type integer containing Cray integer numbers
idest	Variable or array of type integer to contain the converted CDC integer numbers. Each integer is left-justified and zero-filled.
num	Number of Cray integers to convert. Type integer variable, expression, or constant.

CRAY SINGLE-PRECISION TO CDC SINGLE-PRECISION ROUTINE

The FP6460 subroutine converts Cray 64-bit single-precision numbers to CDC 60-bit single-precision numbers.

Call from FORTRAN:

```
CALL FP6460(fpn,dest,num)
```

fpn	Variable or array of any length and of type real containing Cray single-precision numbers
dest	Variable or array of type real to contain the converted CDC 60-bit single-precision numbers. Each floating-point number is left-justified.
num	Number of Cray single-precision numbers to convert. Type integer variable, expression, or constant.

ASCII CHARACTER TO CDC DISPLAY CODE CHARACTER ROUTINE

The ASCDC subroutine converts ASCII data to CDC display code characters.

Call from FORTRAN:

```
CALL ASCDC(src,sc,dest,num)
```

src	Variable or array of any type or length containing ASCII data
sc	ASCII character position to begin the conversion. Leftmost position is 1.
dest	Variable or array of any type or length to contain the converted CDC display code characters (64 character set). Results are packed into continuous strings without regard to word boundries.
num	Number of ASCII characters to convert. Type integer variable, expression, or constant.

DATASET CONTROL ROUTINES

The dataset control routines described in this subsection are utilities that perform the following dataset processes.

- Opening
- Closing
- Inquiry
- Copying
- Skipping
- Positioning
- Termination
- I/O status reporting
- Auxiliary namelist access

OPEN DATASET ROUTINE

The OPEN dataset routine connects an existing dataset to a unit, creates a dataset that is preconnected, creates a dataset and connects it to a unit, or changes certain specifiers of a connection between a dataset and a unit. (For more information on the OPEN dataset routine, see FORTRAN (CFT) Reference Manual, CRI publication SR-0009.)

Call from FORTRAN:

```
CALL OPEN(iolist)
```

iolist	An external unit specifier and, at most, one of each of the other specifiers described in table 5-2.

CLOSE DATASET ROUTINE

The CLOSE dataset routine terminates the connection of a dataset to a unit. (For more information on the CLOSE dataset routine, see FORTRAN (CFT) Reference Manual, CRI publication SR-0009.)

Call from FORTRAN:

```
CALL CLOSE(cllist)
```

cllist	An external unit specifier and, at most, one of each of the other specifiers described in table 5-3.

INQUIRE ROUTINE

The INQUIRE routine returns the status of a unit or a dataset. (For more information on the INQUIRE routine, see FORTRAN (CFT) Reference Manual, CRI publication SR-0009.)

Call from FORTRAN:

```
CALL INQUIRE(ilist)
```

ilist	A list of specifiers described in table 5-4.

DATASET COPYING ROUTINES

Dataset copying routines copy records and files from one dataset to another.

Copy records

COPYR copies a specified number of records from one dataset to another starting at the current dataset position. Following the copy, the datasets are positioned after the end-of-record for the last record copied.

Call from FORTRAN:

```
CALL COPYR(idn,odn,record[,istat])
```

idn	Dataset name or unit number of dataset to be copied
odn	Dataset name or unit number of dataset to receive the copy
record	Number of records to be copied
istat	A 2-element integer array that returns the number of records copied in the first element and number of files copied (always 0) in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

Table 5-2. OPEN specifiers and their meanings

Specifier	Data Type	Meaning	Input (I) or Return value (RV)
UNIT=*u*†	Integer	External unit specifier	(I) Unit number
IOSTAT=*ios*	Integer variable or array element	Error status specifier	(RV) 0 if no error condition exists. If error condition exists, error message number that corresponds to error (see CRAY-OS Message Manual, publication SR-0039 for error message descriptions.)
ERR=*s*	Statement label	Statement label where control is transferred if error condition exists	(I) FORTRAN statement label
FILE=*fin*	Character expression	File specifier	(I) Name of dataset to be connected
STATUS=*sta*	Character expression	Disposition specifier (Default, 'UNKNOWN')	(I) 'OLD', dataset must exist and FILE= must be specified. 'NEW', dataset is created, status becomes 'OLD', FILE= must be specified. 'SCRATCH', dataset is deleted when CLOSE statement is executed or when program is terminated. Dataset must not be named. 'UNKNOWN', the status is 'SCRATCH' if no file specifier is supplied and the unit is not connected; otherwise, the status becomes 'OLD'.
ACCESS=*acc*	Character expression	Access specifier (Default, 'SEQUENTIAL')	(I) 'SEQUENTIAL' is access method; 'DIRECT' is access method.
FORM=*fm*††	Character expression	Form specifier (Default, 'UNFORMATTED' if access is direct; 'FORMATTED' if access is sequential.)	(I) 'FORMATTED', formatted I/O; 'UNFORMATTED', unformatted I/O.
RECL=*rl*	Positive integer expression	Record length for direct access method (omitted for sequential access)	(I) For formatted I/O, number of characters per record; For unformatted I/O, 8 times the number of words.
BLANK=*blnk*	Character expression	Blank specifier (Default, 'NULL')	(I) 'NULL' if numeric input blanks are ignored; 'ZERO' if all nonleading blanks are treated as zeros. This specifier permitted on datasets opened for formatted I/O only.

† UNIT= does not need to be included in the unit specification if *u* is the first item in *olist*.

†† CFT allows formatted and unformatted records in the same dataset (non-ANSI).

Table 5-3. CLOSE specifiers and their meanings

Specifier	Data Type	Meaning	Input (I) or Return value (RV)
UNIT=*u*[†]	Integer	External unit specifier	(I) Unit number
IOSTAT=*ios*	Integer variable or array element	Error status specifier	(RV) 0 if no error condition exists; If error condition exists, error message number that corresponds to error (see CRAY-OS Message Manual, publication SR-0039 for error message descriptions).
ERR=*s*	Statement label	Statement label where control is transferred if error condition exists	(I) FORTRAN statement label
STATUS=*sta*	Character expression	Disposition specifier (Default, 'KEEP' if OPEN status is 'OLD', 'NEW', or 'UNKNOWN'. Default, 'DELETE; if OPEN status is 'SCRATCH' or dataset is memory resident.)	(I) 'KEEP', the dataset continues to exist after CLOSE statement execution. Do not specify 'KEEP' for a dataset with 'SCRATCH' status on an OPEN statement. 'DELETE', the dataset does not exist after execution of the CLOSE statement.

[†] UNIT= does not need to be included in the unit specification if *u* is the first item in *cllist*.

Copy files

COPYF copies a specified number of files from one dataset to another, starting at the current dataset position. Following the copy, the datasets are positioned after the end of file for the last file copied.

Call from FORTRAN:

```
CALL COPYF(idn,odn,file[,istat])
```

idn	Dataset name or unit number of dataset to be copied
odn	Dataset name or unit number of dataset to receive the copy
file	Number of files to be copied
istat	A 2-element integer array that returns the number of records copied in the first element and number of files copied in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

Table 5-4. INQUIRE specifiers and their meanings

Specifier	Data Type	Meaning	Input (I) or Return value (RV)
IOSTAT=*ios*	Integer variable or array element	Error status specifier	(RV) 0 if no error condition exists. If error condition exists, error message number that corresponds to error (see CRAY-OS Message Manual, publication SR-0039 for error message descriptions.)
ERR=*s*	Statement label	Statement label where control is transferred if error condition exists	(I) FORTRAN statement label
EXIST=*ex*	Logical variable or array element	Existence specifier	(RV) .TRUE. if unit or file exists; else, .FALSE.
OPENED=*od*	Logical variable or array element	Connection specifier	(RV) .TRUE. if unit and dataset are connected; else, .FALSE.
NUMBER=*num*	Integer variable or array element	External unit specifier	(RV) Unit currently connected; if no unit, *num* is undefined
NAMED=*nmd*	Logical variable or array element	Unit name specifier	(RV) .TRUE. if unit has a name; else, .FALSE.
RECL=*rcl*	Integer variable or array element	Record length of unit or file connected for direct access	(RV) Record length in characters. (For unformatted I/O, the record length is a positive integer multiple of eight.) If not connected for direct access, *rcl* is undefined.
NEXTREC=*nr*	Integer variable or array element	Next record	(RV) The record number that follows the last record read or written for direct access. If none have been written, *nr*=1. If access is not direct, *nr* is undefined.
NAME=*fn*	Character variable or array element	File name	(RV) File name if file has a name; else, *fn* is undefined.
ACCESS=*acc*	Character variable or array element	Access specifier	(RV) 'SEQUENTIAL' is access method; 'DIRECT' is access method.
SEQUENTIAL=*seq*	Character variable or array element	Sequential as possible access method	(RV) 'YES' if sequential is allowed; 'NO' if sequential is not allowed; 'UNKNOWN' if unable to determine.
DIRECT=*dir*	Character variable or array element	Direct as possible access method	(RV) 'YES' if direct is allowed; 'NO' if direct is not allowed; 'UNKNOWN if unable to determine.
FORM=*fm*†	Character variable or array element	Format specifier	(RV) 'FORMATTED' if file is connected for formatted I/O; 'UNFORMATTED' if file is connected for unformatted I/O.
FORMATTED=*fmt*†	Character variable or array element	Formatted as a possible allowed form	(RV) 'YES' if formatted is allowed; 'NO' if formatted is not allowed; 'UNKNOWN' if unable to determine.
UNFORMATTED=*unf*†	Character variable or array element	Unformatted as a possible allowed form	(RV) 'YES' if unformatted is allowed; 'NO' if unformatted is not allowed; 'UNKNOWN' if unable to determine.
BLANK=*blnk*†	Character variable or array element	Blank control specifier	(RV) 'NULL' if null blank control is in effect; 'ZERO' if zero blank control is in effect. Blank control applies only to formatted records.

† CFT allows formatted and unformatted records in the same dataset (non-ANSI).

Copy dataset

COPYD copies one dataset to another, starting at their current positions. Following the copy, both datasets are positioned after the end of file of the last file copied. The end of data is not written to the output dataset.

Call from FORTRAN:

```
CALL COPYD(idn,odn[,istat])
```

idn	Dataset name or unit number of dataset to be copied
odn	Dataset name or unit number of dataset to receive the copy
istat	A 2-element integer array that returns the number of records copied in the first element and number of files copied in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

Copy sectors (unblocked)

COPYU copies either a specified number of sectors or all data to end-of-data (EOD). Copying begins at the current position on both datasets. Following the copy, the datasets are positioned after the last sector copied.

Call from FORTRAN:

```
CALL COPYU(idn,odn,ns[,istat])
```

idn	Name of unblocked dataset to be copied
odn	Name of unblocked dataset to receive the copy
ns	Decimal number of sectors to copy. If the unblocked dataset contains fewer than *ns* sectors, the copy terminates at EOD. If the keyword *ns* is specified without a value, the copy terminates at EOD. The default is 1.
istat	An integer array or variable that returns the number of sectors copied. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

DATASET SKIP ROUTINES

The dataset skip routines allow the user to skip a specified number of records or files, or a single dataset.

Skip records

SKIPR directs the system to bypass a specified number of records from the current position of the named blocked dataset.

Call from FORTRAN:

```
CALL SKIPR(dn,record[,istat])
```

dn	Dataset name or unit number that contains the record to be skipped. Must be a character constant, integer variable, or an array element containing Hollerith data of not more than seven characters.
record	Decimal number of records to be skipped. The default is 1. If *record* is negative, SKIPR skips backward on *dn*.
istat	A 2-element integer array that returns the number of records skipped in the first element and number of files skipped (always 0) in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

SKIPR does not bypass end-of-file (EOF) or beginning of data (BOD). If an EOF or BOD is encountered before *record* records have been bypassed when skipping backwards, the dataset is positioned after the EOF or BOD. When skipping forward, the dataset is positioned after the last EOR of the current file.

Skip files

SKIPF directs the system to skip a specified number of files from the current position of the named blocked dataset.

Call from FORTRAN:

```
CALL SKIPF(dn,file[,istat])
```

dn	Dataset name or unit number that contains the file to be skipped. Must be a character constant, integer variable, or an array element containing Hollerith data of not more than seven characters.
file	Decimal number of files to be skipped. The default is 1. If *file* is negative, SKIPR skips backward on *dn*. If *dn* is positioned midfile, the partial file skipped counts as one file.
istat	A 2-element integer array that returns the number of records skipped in the first element and number of files skipped in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

SKIPF does not skip end-of-data (EOD) or beginning-of-data (BOD). If a BOD is encountered before *file* files have been skipped when skipping backward, the dataset is positioned after the BOD. When skipping forward, the dataset is positioned before the EOD of the current file.

Example:

If the dataset connected to unit FT07 is positioned just after an end-of-file, the following FORTRAN call positions the dataset after the previous end-of-file. If the dataset is positioned midfile, it is positioned at the beginning of that file.

```
CALL SKIPF('FT07',-1)
```

Skip dataset

SKIPD directs the system to position a blocked dataset at end-of-data (EOD), that is, after the last end-of-file of the dataset. If the specified dataset is empty or already at EOD, the call has no effect.

Call from FORTRAN:

```
CALL SKIPD(dn[,istat])
```

dn	Dataset name or unit number to be skipped. Must be a character constant, integer variable, or an array element containing Hollerith data of not more than seven characters.
istat	A 2-element integer array that returns the number of records skipped in the first element and number of files skipped in the second element. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

Skip sectors (unblocked)

SKIPU directs the system to bypass a specified number of sectors or all data from the current position of the named unblocked dataset.

Call from FORTRAN:

```
CALL SKIPU(dn,ns[,istat])
```

dn	Dataset name or unit number of unblocked dataset to be bypassed. Must be an integer variable or an array element containing ASCII data of not more than seven characters.
ns	Decimal number of sectors to bypass. The default value is 1. If *ns* is negative, SKIPU skips backward on *dn*.
istat	An integer array or variable that returns the number of sectors skipped. *istat* is an optional parameter. If present, only fatal messages are written to the logfile.

DATASET POSITIONING ROUTINES

Dataset positioning routines change or reflect the position of the current dataset. The following positioning routines, except for $GPOS and GETPOS, set the current processing direction to input (read). If the previous processing direction is output (write), end-of-data is written on a sequential dataset and the buffer is flushed. On a random dataset, the buffer is flushed.

Get position of mass storage dataset

The $GPOS routine returns the position of the specified mass storage dataset. It determines the current word address of the dataset and can return flags indicating that the dataset is positioned at a record, file, or dataset boundary. The dataset's position is not altered.

The GTPOS%, FDGPOS%, and GETPOS routines return the position of the specified interchange tape or mass storage dataset. The dataset's position is not altered.

Call from CAL:

```
CALLV $GPOS
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

Exit:

(A1)	DSP address
(S2)	Dataset position
(S1)	Dataset position with flags. (S1)=0 if the dataset is at the beginning-of-data (BOD); otherwise, the following flags are set:

Bit	Description
0	End-of-record (EOR) flag
1	Dataset is positioned at a record boundary, that is, following a record control word (RCW). This bit is also set if end-of-file (EOF) or end-of-data (EOD) bits are set.
0	Dataset is either at BOD or in the middle of a record.
1	Unused
2	End-of-file flag. One indicates the dataset is at a file boundary, that is, following the end-of-file RCW. This bit is also set if the EOD bit is set.
3	End-of-data flag. One indicates the dataset is currently at EOD.
31-63	Word address. This word address is the current physical word address within the dataset, including RCWs but not including block control words (BCW).

GTPOS% obtains, for the CAL user, position information about an opened interchange tape dataset. The information returned by GTPOS% refers to the last block processed if the dataset is an input dataset. For output datasets, the information returned by GTPOS% is meaningless unless the tape dataset has been synchronized before the GTPOS% request is made. GTPOS% uses call-by-value linkage.

Call from CAL:

```
CALLV GTPOS%
```

Entry:

(A1)	Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)
(A2)	Address of tape position information storage area. This area must contain LE@TPI words to hold the tape information.

Exit:

(A1)	Absolute DSP address
(A2)	Address of tape position information storage area. The tape position information is returned in fields as defined in the Macros and Opdefs Reference Manual, CRI publication SR-0012.
(S1)	Return conditions. On exit this register returns errors and warnings from the tape get position routine. =0 Tape position information successfully returned ≠0 Error or warning encountered during request +2 Dataset is not a tape dataset.

FDGPOS% obtains, for the CAL user, position information about a foreign interchange tape or mass storage dataset being processed with the library data conversion support (FD parameter on the ACCESS and ASSIGN control statements). The information returned by FDGPOS% is internal information to be retained and passed on to a FDSPOS% request. FDGPOS% uses call-by-value linkage.

Call from CAL:

```
CALLV FDGPOS%
```

Entry:

(A1)	Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)

Exit:

(A1)	Absolute DSP address
(S1)	Return conditions. On exit this register returns errors and warnings from the tape get position routine. =0 Tape position information successfully returned ≠0 Error or warning encountered during request; error message number; see coded $IOLIB messages in CRAY-OS Message Manual, publication SR-0039.
(S2)	Dataset position information for finding the beginning of the current block (Cray physical record). For disk datasets, this register is set to the Cray block number for the sector containing the beginning of the current block. For tape datasets, this register is set to the volume block count as returned from a TAPEPOS macro request.
(S3)	Dataset position information for finding the beginning of the current block (Cray physical record). For disk datasets, this register is set to the dataset position, including BCWs, of the beginning of the current block. For tape datasets, this register is set to the volume serial number as returned from a TAPEPOS macro request.
(S4)	Dataset position information for finding the beginning of the current logical record. For disk and tape datasets, this register is set to the current foreign dataset block bit length.

GETPOS returns to the FORTRAN user the current position of the specified interchange tape or mass storage dataset. This is a generic routine independent of the device medium. The routine does not alter the dataset's position, but captures information that later can be used to recover the current position.

If foreign dataset conversion has not been requested, the physical tape block and volume position is determined. For disk datasets, a non-tape GETPOS request is made.

Call from FORTRAN:

```
CALL GETPOS (dn,len,pa[,stat])
```

dn	Dataset name or unit number
len	Length in Cray words of the position array. This parameter determines the maximum number of position values to return. This parameter should be set equal to 3.

pa	Position array. On exit, *pa* contains the current position information. This information should not be modified by the user. It should be retained to be passed on to the SETPOS routine.
stat	Return conditions. This optional parameter returns errors and warnings from the position information routine. =0 Position information successfully returned ≠0 Error or warning encountered during request; Error message number; see coded $IOLIB messages in the CRAY-OS Message Manual, publication SR-0039.

Set position of dataset

To set the position of a mass storage dataset, the position must be at a record boundary, that is, at BOD or following an EOR or EOF, or before an EOD. A dataset cannot be positioned beyond the current EOD.

The $SPOS%, $ASPOS, and $FSPOS routines set the position of the specified interchange tape or mass storage dataset. The STPOS%, FDSPOS%, and SETPOS routines set the position of the specified interchange tape or mass storage dataset.

$SPOS synchronously sets the dataset position, and $ASPOS asynchronously sets the dataset position. Library routines finish the asynchronous positioning ($ASPOS) if necessary. A bit in the Dataset Parameter Table (DPTPOS) can be tested to determine if a $FSPOS call is required.

Call from CAL:

```
CALLV $SPOS
CALLV $ASPOS
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN).
(S1)	Dataset position

Bit	Description
0-30	Unused
31-63	Word address. The desired physical word address within the dataset aligned on a record boundary. Record control words are included.

Exit:

(A1)	DSP address
(S1)	Dataset position
(S6)	RCW after which the dataset is positioned; (S6)=0 if at BOD.

$FSPOS finishes the asynchronous dataset positioning request.

Call from CAL:

```
CALLV $FSPOS
```

STPOS% positions, for the CAL user, a tape dataset at a particular tape block of the dataset. Data blocks on the tape are numbered so that block number 1 is the first data block on a tape. Before a tape dataset is positioned with STPOS%, the tape must be synchronized with a call to SYNCH%. STPOS% uses call-by-value linkage.

Call from CAL:

```
CALLV STPOS%
```

Entry:

(A1)	Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)
(S1)	Block number request sign. This register must be set to '+'L, '-'L, or ' 'L. Refer to register S2, the block number value register for usage details.
(S2)	Block number or number of blocks to forward space or backspace from the current position. The direction of the positioning is specified by the block number request sign register, S1. For forward block positioning, set register S1 to '+'L. The plus sign is invalid if either the volume number (S4) is unsigned or the volume identifier (S5) has been specified. For backward block positioning, set register S1 to '-'L. The minus sign is invalid if either the volume number (S4) is unsigned or the volume identifier (S5) has been specified. For absolute block positioning, set register S1 to ' 'L.

(S3) Volume number request sign. This register must be set to '+'L, '-'L, or ' 'L. Refer to volume number value register, S4, for usage details.

(S4) Volume number or number of volumes to forward space or backspace from the current position. This parameter should be set equal to a binary volume number or number of volumes to forward space or backspace. The direction of the positioning is specified by the volume number request sign register, S3. This parameter is invalid if the volume identifier (S5) has also been specified.

For forward volume positioning, set register S3 to '+'L. A block request (register S2) must not be specified with plus or minus signs.

For backward volume positioning, set register S3 to '-'L. A block request (register S2) must not be specified with plus or minus signs.

For absolute volume positioning, set register S3 to ' 'L.

(S5) Volume identifier to be mounted. This parameter is invalid if number of volumes (register S4) has also been requested. Also a block request (register S2) must not be specified without plus or minus signs. The volume identifier must be of the form 'VOL'L.

Exit:

(A1) Absolute DSP address

(S1) Return conditions. On exit this register is used to return errors and warnings from the tape get position routine.

- =0 Tape successfully positioned
- ≠0 Error or warning encountered during request
- -1 Set tape position parameter error
- +2 Dataset is not a tape dataset.
- +3 Positioning request not fully satisfied

FDSPOS% positions, for the CAL user, a foreign interchange tape or mass storage dataset being processed with the library data conversion support (FD parameter on the ACCESS or ASSIGN control statements). FDSPOS% returns to the positions retained from a FDGPOS% request. FDSPOS% uses call-by-value linkage.

Call from CAL:

```
CALLV FDSPOS%
```

Entry:

(A1)	Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)
(S2)	Dataset position information for finding the beginning of the current block (Cray physical record). For disk datasets, this register is set to the Cray block number for the sector containing the beginning of the current block. For tape datasets, this register is set to the volume block count as returned from a TAPEPOS macro request.
(S3)	Dataset position information for finding the beginning of the current block (Cray physical record). For disk datasets, this register is set to the dataset position, including BCWs, of the beginning of the current block. For tape datasets, this register is set to the volume serial number as returned from a TAPEPOS macro request.
(S4)	Dataset position information for finding the beginning of the current logical record. For disk and tape datasets, this register is set to the current block bit length.

Exit:

(A1)	Absolute DSP address
(S1)	Return conditions. On exit this register returns errors and warnings from the tape get position routine. =0 Tape position information successfully returned ≠0 Error or warning encountered during request; error message number; see coded $IOLIB messages in CRAY-OS Message Manual, publication SR-0039.

SETPOS allows the FORTRAN user to return to the position retained from the GETPOS request. This is a generic routine independent of the device medium. SETPOS can be used on interchange tape or mass storage datasets.

SETPOS positions to a logical record when processing a foreign file with the library data conversion support (FD parameter on the ACCESS and ASSIGN control statements). This same capability also exists for mass storage files that have been assigned foreign dataset characteristics.

If foreign dataset conversion has not been requested, the physical tape block and volume position is determined. A non-tape SETPOS request is made for disk datasets if foreign conversion has not been specified.

For interchange tape datasets, SETPOS must synchronize before the dataset can be positioned. Thus, for input datasets, the dataset must be positioned at a Cray end-of-record. An end-of-record is added to the end of data before the synchronization if the dataset is an output dataset and the end of the tape block was not already written.

Call from FORTRAN:

```
CALL SETPOS (dn,len,pa[,stat])
```

dn	Dataset name or unit number
len	Length in Cray words of the position array. This parameter determines the maximum number of position values to process and allows for the addition of more information fields while ensuring that existing codes continue to run. This parameter should be set equal to 3.
pa	Position array. On entry, *pa* contains the desired position information from the GETPOS call.
stat	Return conditions. This optional parameter returns errors and warnings from the position routine. =0 Dataset successfully positioned ≠0 Error or warning encountered during request; Error message number; see coded $IOLIB messages in the CRAY-OS Message Manual, publication SR-0039.

Backspace one record

The $BKSP and BACKSPACE routines position the dataset after the previous end-of-record (EOR). The function is nonoperational if the dataset is at beginning-of-data (BOD). If the dataset is at the first record of a file, these routines position the dataset before the end-of-file (EOF).

Call from CAL:

```
CALLV $BKSP
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JSDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.

Exit:

(A1)	Address of DSP
(S6)	Contains record control word (RCW) after which dataset is positioned (equals 0 if at BOD)

Call from FORTRAN (also clears UEOF flag in the DSP):

$$\text{BACKSPACE}\left(\begin{Bmatrix} u \\ fin \\ alist \end{Bmatrix}\right)$$

u	Unit identifier
fin	File identifier whose value specifies the name of the file
alist	The following set of identifiers: [UNIT]=*u* IOSTAT=*ios* ERR=*s* Must contain a single external unit specifier and can contain, at most, one of each of the other specifiers. See the UNIT, IOSTAT, and ERR specifiers described for the OPEN or CLOSE statement, table 5-2 or table 5-3.

The FDBKSP% routine allows a CAL user to backspace a logical record on a foreign interchange tape or mass storage dataset being processed with the library data conversion support.

Call from CAL:

```
CALLV FDBKSP%
```

Entry:
- (A1) Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)

Exit:
- (A1) Absolute DSP address
- (S1) Return conditions. On exit this register returns errors and warnings from the tape get position routine.
  - =0 Tape position information successfully returned
  - ≠0 Error or warning encountered during request; error message number; see coded $IOLIB messages in CRAY-OS Message Manual, publication SR-0039.

Backspace one file

The $BKSPF and BACKFILE routines position a dataset after the previous end-of-file (EOF). The function is nonoperational if the dataset is at beginning-of-data (BOD).

Call from CAL:

```
CALLV $BKSPF
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.

Exit:

(A1)	Address of DSP
(S6)	Contains record control word (RCW) after which dataset is positioned (equals 0 if at BOD)

Call from FORTRAN (also clears UEOF flag in DSP):

```
CALL BACKFILE(dn)
```

dn	Dataset name or unit number

Rewind dataset

Rewind routines rewind a sequential access dataset.

Call from CAL:

```
CALLV $REWD
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.

Exit:

(A1)	Address of DSP

Call from FORTRAN:

$$\text{REWIND(}\left\{\begin{array}{l} u \\ fin \\ alist \end{array}\right\}\text{)}$$

u	Unit identifier. If the dataset is $IN, the file is positioned after the first EOF. Otherwise, it is positioned at BOD.
fin	File identifier whose value specifies the name of a file
alist	The following set of specifiers must contain a single external unit specifier and can contain, at most, one of each of the other specifiers. [UNIT]=*u* IOSTAT=*ios* ERR=*s* See the UNIT, IOSTAT, and ERR specifiers described for the OPEN or CLOSE statement, table 5-2, or table 5-3.

Position dataset

$PBN and $APBN position a dataset to a specific block, update DPOUT of DSP, clear DPRCW of DSP, and update DPIN of DSP, if a read is from disk. $PBN synchronously positions the dataset; $APBN asynchronously positions the dataset.

Call from CAL:

```
CALL $PBN
CALL $APBN
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP)
(A4)	Block number

NOTE

A dataset must be in read mode or positioned after a $WEOD call. If a disk read was done without recall (asynchronous), the user is responsibile for checking on completion of the read and any outstanding errors. The only valid exit value is the DSP address found in A1.

Exit:

(A1)	Address of DSP
(A4)	New block number
(A5)	DPFRST from DSP:
(S0)	0 (asynchronous). Disk read required and recall not requested 1 (synchronous). Disk read not required, or disk read required and recall requested
(S3)	DPOUT from DSP

$PRCW positions dataset after a record control word (RCW). The dataset is positioned after end-of-record in record mode and after end-of-file in file mode. The dataset must already be positioned at the block control word (BCW) for the block containing the RCW.

Call from CAL:

```
CALLV $PRCW
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP)
(A4)	DPOBN from DSP
(S3)	DPOUT from DSP
(S5)	Open Dataset Name Table (ODN) (word address including block control words)
(S6)	Mode<0: Record mode >0: File mode

Exit:

(A1)	Address of DSP
(S6)	RCW after which dataset is positioned (0 if at BOD)

Synchronize tape dataset

SYNCH% synchronizes a CAL program and an interchange tape dataset. If the dataset is synchronized for input, the tape must be positioned at the end of a block (Cray end-of-record). However, if the dataset is an output dataset and a Cray end-of-record has not been written, the SYNCH% routine writes an EOR before synchronizing. For an output tape, control is not returned to the user until all of the data in the circular I/O buffer has been written to the tape. SYNCH% uses call-by-value linkage.

Call from CAL:

```
CALLV SYNCH%
```

Entry:

(A1)	Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)
(S1)	Processing direction: =0 Input dataset =1 Output dataset

Exit:

(A1)	Absolute DSP address
(S1)	Return conditions. On exit this register returns errors and warnings from the synchronization routine. =0 Tape successfully synchronized ≠0 Error or warning encountered during request +1 Execution error; the error code is in DPERR field of DSP table. +2 Dataset is not a tape dataset.

DATASET TERMINATION ROUTINES

The $WEOF, $WEOD, EODW, ENDFILE, and $EOFR routines terminate datasets by writing end-of-record (EOR) and end-of-file (EOF), or EOR, EOF, and end-of-data (EOD).

$WEOF writes an EOF preceded by an EOR, if necessary, as the next words in the I/O buffer.

$WEOD writes an EOD preceded by an EOR and an EOF, if necessary, as the next words in the I/O buffer. The $WEOD forces the final block of data to be written on the disk; that is, it flushes the I/O buffer. The dataset is left positioned before the EOD.

Call from CAL:

```
CALLV $WEOF
CALLV $WEOD
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.

Exit:

(A1)	Address of DSP

EODW writes EOD, EOF, and EOR, if necessary, and clears the UEOF flag in the DSP.

Call from FORTRAN:

```
CALL EODW(dn)
```

dn	Dataset name or unit number

ENDFILE writes EOF and EOR, if necessary, on sequential access file and clears UEOF flag in DSP.

Call from FORTRAN:

```
ENDFILE(dn[,iostat,err])
```

dn	Dataset name or unit number
iostat	Address of IOSTAT parameter
err	ERR parameter

$EOFR determines if UEOF flag in DSP is set and clears UEOF flag in DSP.

Call from CAL:

```
CALL $EOFR,(arg)
```

Entry:

arg	Address of dataset name or unit number

Exit:

(S1)	0 if UEOF was not set 1 if UEOF was set

I/O STATUS ROUTINES

UNIT returns I/O status upon completion of an I/O operation.

Call from FORTRAN:

exit=UNIT(*dn*)

exit	-2.0	Operation complete, no errors, partial read, did not read EOR (blocked I/O only)
	-1.0	Operation complete, no errors
	0.0	EOF or EOD on last read on blocked datasets, or end-of-information (EOI) on unblocked datasets
	+1.0	Parity error
	+2.0	Unrecovered hardware error
dn		Dataset name or unit number

LENGTH returns number of words processed in last buffer operation.

Call from FORTRAN:

exit=LENGTH(*dn*)

exit	On exit, word count from last buffer operation or 0 if EOF or EOD is encountered
dn	Dataset name or unit number

IEOF returns integer EOF status and clears UEOF flag in DSP.

Call from FORTRAN:

iexit=IEOF(*dn*)

iexit	-1	EOD on last operation
	0	Neither EOD nor EOF on last operation
	+1	EOF on last operation
dn	Dataset name or unit number	

EOF returns a real value EOF status and clears UEOF flag in DSP.

Call from FORTRAN:

rexit=EOF(*dn*)

rexit	-1.0	EOD on last operation
	0.0	Neither EOD nor EOF on last operation
	+1.0	EOF on last operation
dn	Dataset name or unit number	

IOSTAT returns end-of-file status.

Call from FORTRAN:

iexit=IOSTAT(*dn*)

iexit	0	No error
	1	UEOF cleared (EOF)
	2	UEOD cleared (EOD)
dn	Dataset name or unit number	

$EOATEST checks for a read/write past the allocated area condition. If such a condition exists, an error message is issued and the job aborted.

Call from CAL:

```
CALLV $EOATEST
```

Entry:
  (A1)   DSP address
Exit (if job not aborted):
  (A1)   DSP address

$UEOFSET sets the uncleared End-of-file flag in the DSP.

Call from CAL:

```
CALLV $UEOFSET
```

Entry:
(A1) DSP address
Exit:
(A1) DSP address

$UEOFTCL clears the uncleared End-of-file flag in the DSP and indicates whether it had been set or not.

Call from CAL:

```
CALLV $UEOFTCL
```

Entry:
(A1) DSP address
Exit:
(A1) DSP address
(S1) 0, if UEOF was not set
Nonzero if UEOF was set

$UEOFKIL aborts job if uncleared End-of-file flag is set in the DSP.

Call from CAL:

```
CALLV $UEOFKIL
```

Entry:
(A1) DSP address
Exit (if job not aborted):
(A1) DSP address

AUXILIARY NAMELIST ROUTINES

NAMELIST routines allow user control of input and output defaults and are accessed by call-by-address subprogram linkage. No arguments are returned. For a complete description of the NAMELIST feature, see the FORTRAN (CFT) Reference Manual, CRI publication SR-0009.

RNLSKIP determines action if NAMELIST group encountered is not the desired group.

Call from FORTRAN:

```
CALL RNLSKIP(mode)
```

mode	>0	Skip past the group and issue an information logfile message (default).
	=0	Skip past group.
	<0	Abort job or go to ERR=address.

RNLTYPE determines the action if a type mismatch occurs across the equal sign on an input card.

Call from FORTRAN:

```
CALL RNLTYPE(mode)
```

mode	=0	Abort job or go to ERR=address.
	≠0	Convert constant type to variable type if possible; otherwise, abort or go to ERR=address (default).

RNLECHO specifies unit for error messages and input echo.

Call from FORTRAN:

```
CALL RNLECHO(unit)
```

unit	=0	Error messages and echoed input go to $OUT (default).
	≠0	Error messages and echoed input go to specified unit. All input is echoed.

The following four routines (RNLFLAG, RNLDELM, RNLSEP, and RNLCOMM) add or delete characters from the set of characters recognized by the namelist input routine in various positions. *char* is a single Hollerith character specified in H, C, or R format; it is not a character variable. No checks are made to ensure that alternate character selections are consistent.

RNLFLAG deletes or adds echo character. If an echo character appears in column 1 of an input record, that record and all subsequent records processed by the current READ, are copied to the echo output unit.

Call from FORTRAN:

```
CALL RNLFLAG(char,mode)
```

char	Echo character. Default is E.
mode	=0 Delete character. ≠0 Add character.

RNLDELM deletes or adds NAMELIST group delimiting character. The group character is the first character of the group name and the END terminator.

Call from FORTRAN:

```
CALL RNLDELM(char,mode)
```

char	Delimiting character. Default is $ and &.
mode	=0 Delete character. ≠0 Add character.

RNLSEP deletes or adds separator character. The separator character separates data items in the input records.

Call from FORTRAN:

```
CALL RNLSEP(char,mode)
```

char	Separator character. Default is ,.
mode	=0 Delete character. ≠0 Add character.

RNLREP deletes or adds replacement character.

Call from FORTRAN:

```
CALL RNLREP(char,mode)
```

char	Replacement character. Default is =.
mode	=0 Delete character. ≠0 Add character.

RNLCOMM deletes or adds trailing comment indicator.

Call from FORTRAN:

```
CALL RNLCOMM(char,mode)
```

char	Trailing comment indicator. Default is : and ;.
mode	=0 Delete character. ≠0 Add character.

WNLLONG indicates output line length.

Call from FORTRAN:

```
CALL WNLLONG(length)
```

length	Output line length; 8<length<161 or length=-1 (-1 specifies default of 133 unless the unit is 102 or $PUNCH, in which case the default is 80).

WNLDELM defines ASCII NAMELIST delimiter.

Call from FORTRAN:

```
CALL WNLDELM(char)
```

char	NAMELIST delimiter. Default is &.

WNLSEP defines ASCII NAMELIST separator.

Call from FORTRAN:

CALL WNLSEP(*char*)

char NAMELIST separator. Default is ,.

WNLREP defines ASCII NAMELIST replacement character.

Call from FORTRAN:

CALL WNLREP(*char*)

char NAMELIST replacement character. Default is =.

WNLFLAG indicates the first ASCII character of the first line.

Call from FORTRAN:

CALL WNLFLAG (*char*)

char First ASCII character of the first line. Default is blank.

LOGICAL RECORD I/O ROUTINES

Logical record I/O routines are normally called by FORTRAN I/O routines and communicate with the system through Exchange Processor requests.

These routines are divided into read, write, CAL I/O interface, character, and bad data error recovery routines.

READ ROUTINES

Read routines transfer partial or full records of data from the I/O buffer to the user data area. Depending on the read request issued, the data is placed in the user data area one character per word or in full words. (Blank decompression occurs only when data is being read one character per word.) In partial mode the dataset maintains its position after the read is executed. In record mode the dataset position is maintained after the EOR that terminates the current record. Figure 5-4 provides an overview of the logical read operation.

Read words

Routines transferring full words transmit the words from the I/O buffer to the area beginning at the first word address (FWA). This process continues until either the word count in A3 is satisfied or an EOR is encountered.

Unrecovered data errors do not cause the job to abort. Control is returned to the user to use the good data that was read, (A2) through (A4)-1, and to decide whether to abort or to skip or accept the bad data. If the user does nothing, the job is aborted on the next read request.

$RWDP reads words, partial record mode. $RWDR reads words, full record mode.

Call from CAL:

```
CALLV $RWDP
CALLV $RWDR
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.
(A2)	FWA of user data area
(A3)	Word count. If count=0, no data is transferred.

Exit:

(A1)	Address of DSP
(A2)	FWA of user data area
(A3)	Word count
(A4)	Actual LWA+1 (equals FWA if null record)
(S0)	Termination mode: <0 Read terminated by EOR =0 Null record, EOF, EOD, or unrecovered data error encountered >0 Read terminated by count. If count is exhausted simultaneously with reaching EOR, the EOR takes precedence.
(S1)	Error status: 0 No errors encountered 1 Unrecovered data error encountered
(S6)	Record control word (RCW) if (S0)≤0 and (S1)=0

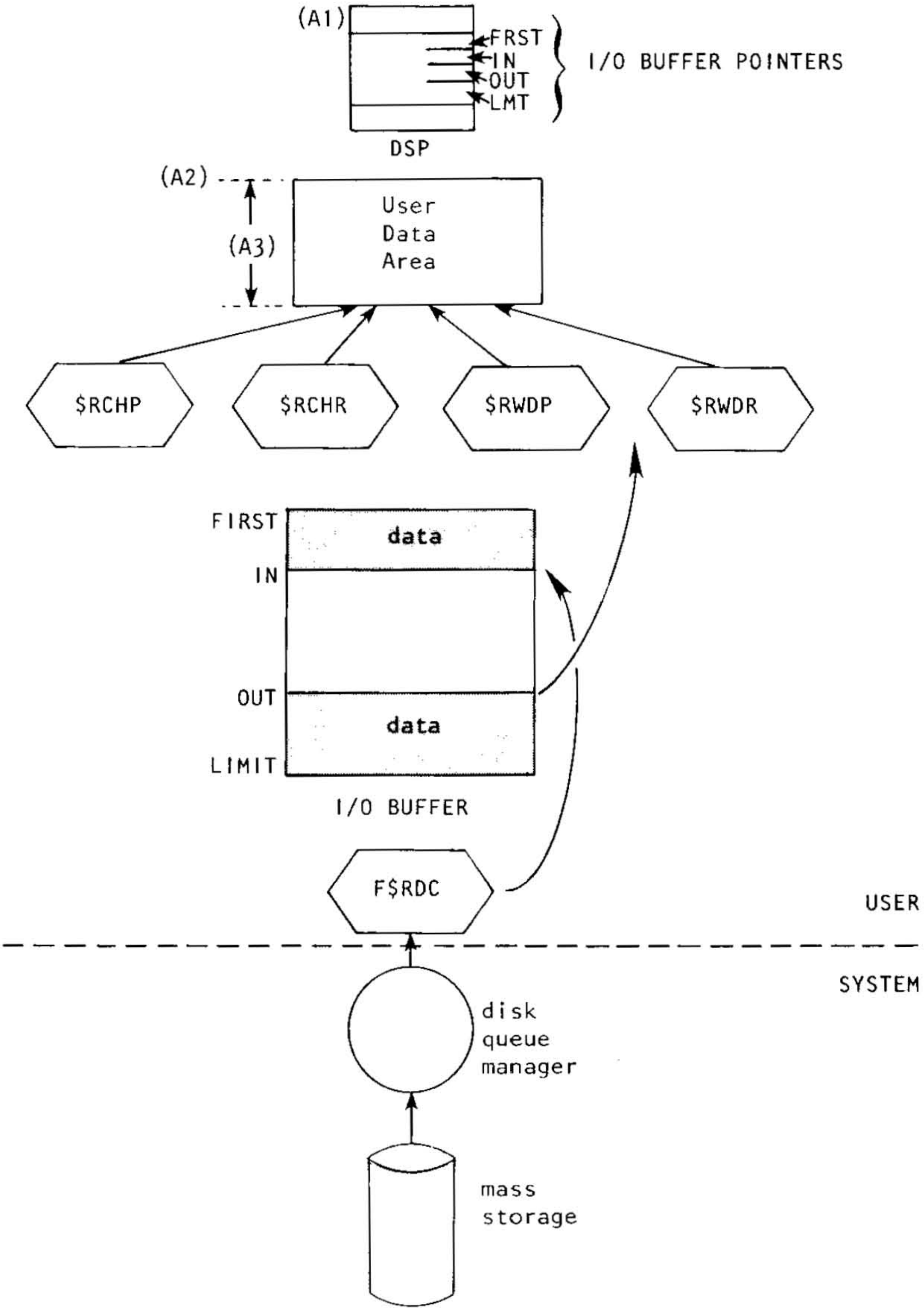
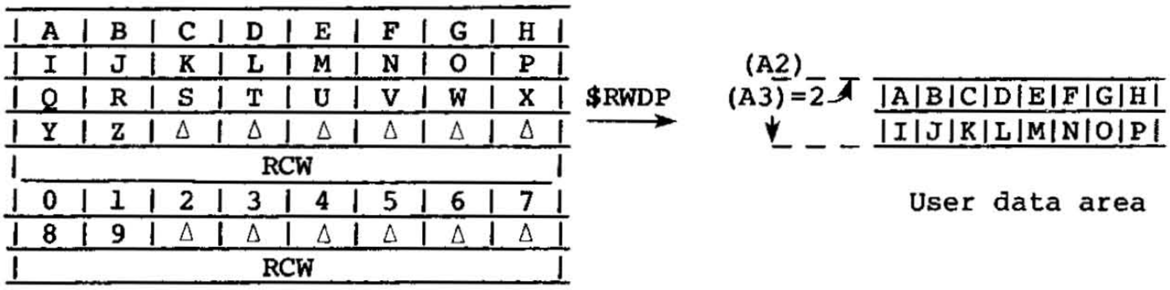


Figure 5-4. Logical read

Example of $RWDP:



Data in I/O buffer

READ reads words, full record mode. READP reads words, partial record mode.

Call from FORTRAN:

```
CALL READ(dn,word,count,status[,ubc])
CALL READP(dn,word,count,status[,ubc])
```

dn	Dataset name or unit number
word	Word-receiving data area
count	On entry: number of words requested On exit: number of words actually transferred
status	On exit: -1 Words remain in record. 0 EOR 1 Null record 2 EOF 3 EOD 4 Unrecovered data error encountered
ubc	Optional unused bit count. On exit, if end of record is reached, *ubc* contains the unused bit count in the last word. The unused bits are zeroed in the user's data area.

Read characters

Read character routines unpack characters from the I/O buffer and insert them into the user data area beginning at the first word address (FWA). This process continues until either the count is satisfied or an EOR is encountered. If an EOR is encountered first, the remainder of the field specified by the character count is filled with blanks.

Unrecovered data errors do not cause the job to abort. Control is returned to the user to use the good data that was read, (A2) through (A4)-1, and to decide whether to abort, or to skip or accept the bad data. If the user does nothing, the job is aborted on the next read request.

$RCHP reads characters, partial record mode. $RCHR reads characters, full record mode.

Call from CAL:

```
CALLV $RCHP
CALLV $RCHR
```

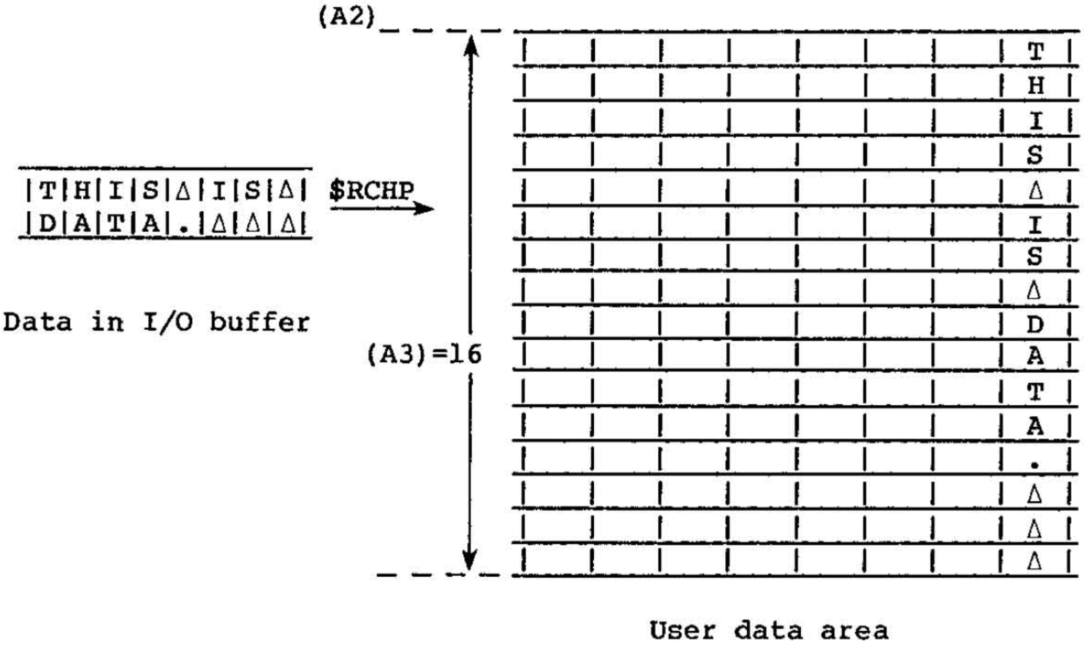
Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.
(A2)	First word address (FWA) of user data area
(A3)	Character count. If count=0, no data is transferred.

Exit:

(A1)	Address of DSP
(A2)	FWA of user data area
(A3)	Character count
(A4)	Actual LWA+1 (equals FWA if null record)
(S0)	Termination mode: <0 Read terminated by EOR =0 Null record, end-of-file, end-of-data, or unrecovered data error encountered >0 Read terminated by count. If count is exhausted simultaneously with reaching EOR, the EOR takes precedence.
(S1)	Error status: =0 No errors encountered =1 Unrecovered data error encountered
(S6)	RCW if (S0)≤0 and (S1)=0

Example of $RWDP:



READC reads characters, full record mode. READCP reads characters, partial record mode.

Call from FORTRAN:

```
CALL READC(dn,char,count,status)
CALL READCP(dn,char,count,status)
```

dn	Dataset name or unit number
char	Character-receiving data area
count	On entry: Number of characters requested On exit: Number of characters actually transferred
status	On exit: -1 Characters remain in record. 0 EOR 1 Null record 2 EOF 3 EOD 4 Unrecovered data error

Read IBM words

READIBM reads two IBM 32-bit floating-point words from each Cray 64-bit word.

Call from FORTRAN:

```
CALL READIBM(dn,fwa,word,increment)
```

dn	Dataset name or unit number
fwa	First word address (FWA) of user data area
word	Number of words needed
increment	Increment of IBM words read

On exit, the IBM 32-bit format is converted to the equivalent Cray 64-bit value. The Cray 64-bit words are stored in user data area.

Read unblocked data

$RLB reads data directly into the user area without the use of system I/O buffers, RCWs, or BCWs.

Call from CAL:

```
CALLV $RLB
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.
(A2)	First word area (FWA) of user data area
(A3)	Word count. If count=0, no data is transferred.
(S1)	Recall indicator; specifies whether read is done synchronously or asynchronously. 0 No I/O recall requested (asynchronous) 1 I/O recall requested (synchronous)

Exit:

(A1-A3)	Unchanged	
If recall requested:		
(S0)	-1.0	Operation complete, no errors
	0.0	End-of-information (EOI) on last read
	+1.0	Parity error
(A4)	Actual last word address+1	

WRITE ROUTINES

Write routines transfer partial or full records of data from the user data area to the I/O buffer. Depending on the write operation requested, data is taken from the user data area one character per word and packed eight characters per word or is transferred in full words. (Blank compression occurs only when data is being written one character per word). In partial mode, no EOR is inserted in the I/O buffer to terminate the record. In record mode an EOR is inserted in the I/O buffer in the next word following the data that terminates the record. Figure 5-5 provides an overview of the logical write operation.

Write words

In routines where words are written, the number of words specified by the count is transmitted from the area beginning at the first word address (FWA) and is written in the I/O buffer.

$WWDP writes words, partial record mode.

$WWPU writes words, partial record mode with unused bit count. The user can specify the unused bit count in the last word of a partial record as an entry condition.

$WWDR writes words, full record mode.

$WWDS writes words, full record mode with unused bit count. The user can specify the unused bit count in the last word of the record as an entry condition.

Call from CAL:

```
CALLV $WWDP
CALLV $WWPU
CALLV $WWDR
CALLV $WWDS
```

Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.
(A2)	First word address (FWA) of user data area
(A3)	Word count. If count=0, no data is transferred; EOR is written.
(A4)	Unused bit count, value 0 to 63 ($WWDS and $WWPU only)

Exit:

(A1)	Address of DSP
(A2)	FWA of user data area
(A3)	Word count

Example of **$WWDP**:

```
 _______________                   _______________
|A|B|C|D|E|F|G|H|                 |A|B|C|D|E|F|G|H|
|I|J|K|L|M|N|O|P|      $WWDP      |I|J|K|L|M|N|O|P|
|Q|R|S|T|U|V|W|X|    -------->    |Q|R|S|T|U|V|W|X|
|Y|Z| | | | | | |                 |Y|Z| | | | | | |
 ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾                   ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾

   User data area                     I/O buffer
```

WRITE writes words, full record mode.

Call from FORTRAN:

```
CALL WRITE(dn,word,count[,ubc])
```

dn	Dataset name or unit number
word	Data area containing words
count	Word count. A value of 0 causes an end-of-record record control word to be written.
ubc	Optional unused bit count. Number of unused bits contained in the last word of the record.

WRITEP writes words, partial record mode.

Call from FORTRAN:

```
CALL WRITEP(dn,word,count[,ubc])
```

dn	Dataset name or unit number
word	Data area containing words
count	Word count
ubc	Optional unused bit count. Number of unused bits contained in the last word of the record.

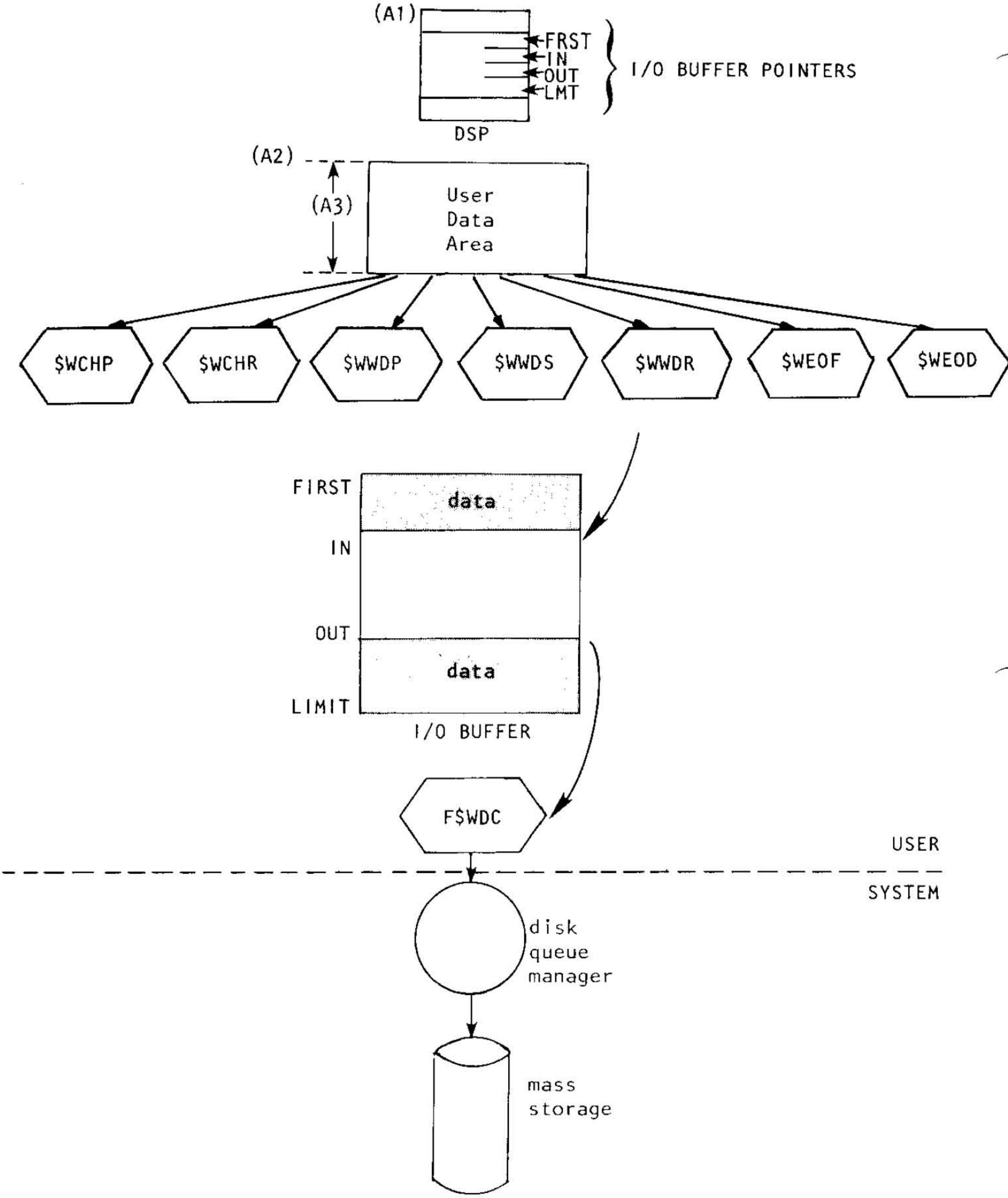


Figure 5-5. Logical write

Write characters

Write character routines pack characters into the I/O buffer for the dataset. The number of characters packed is specified by the count. These characters originate from the user area defined at first word address (FWA).

$WCHP writes characters, partial record mode.

$WCHR writes characters, full record mode. The unused bit count in the record control word (RCW) specifies the EOD in the previous word.

Call from CAL:

```
CALLV $WCHP
CALLV $WCHR
```

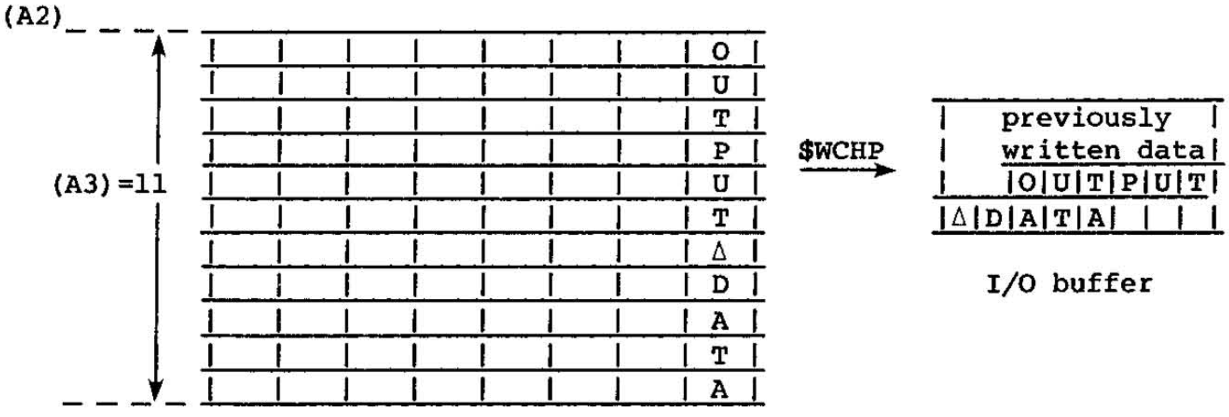
Entry:

(A1)	Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.
(A2)	FWA of user data area
(A3)	Character count. If count=0, no data is transferred; EOR is written.

Exit:

(A1)	Address of DSP
(A2)	FWA of user data area
(A3)	Word count

Example of $WCHP:



User data area

WRITEC writes characters, full record mode. WRITECP writes characters, partial record mode.

Call from FORTRAN:

```
CALL WRITEC(dn,char,count)
CALL WRITECP(dn,char,count)
```

dn	Dataset name or unit number
char	Data area containing characters
count	Character count

Write IBM words

WRITIBM writes two IBM 32-bit floating-point words from each Cray 64-bit word.

Call from FORTRAN:

```
CALL WRITIBM(dn,fwa,value,increment)
```

dn	Dataset name or unit number
fwa	FWA of user data area
value	Number of values to be written
increment	Increment of source (Cray) words written

On exit, IBM 32-bit words written to unit

Write unblocked data

$WLB writes data directly from user area without the use of system I/O buffers, RCWs, or BCWs.

Call from CAL:

```
CALLV $WLB
```

Entry:

(A1) Address of Dataset Parameter Table (DSP) or, if negative, DSP offset relative to DSP base (JCDSP). The second word of Open Dataset Name Table (ODN) also contains this negative value.

(A2) First word address (FWA) of user data area

(A3) Word count. If count=0, no data is transferred.

(S1) Recall indicator. Specifies whether write is done synchronously or asynchronously.
- 0 No I/O recall requested (asynchronous)
- 1 I/O recall requested (synchronous)

Exit:

(A1-A3) Unchanged

If recall requested:

(S0)
- -1.0 Operation complete, no errors
- +1.0 Parity error
- +2.0 Unrecovered hardware error

CAL I/O INTERFACE ROUTINE

$CBIO provides buffered I/O interface for CAL requests. The user is responsible for ensuring that requested data transfers are complete and error free by examining the Dataset Parameter Table (DSP) before attempting to process input data or requesting additional writes.

Call from CAL:

```
CALLV $CBIO
```

Entry:

(S1) Address of DSP

BAD DATA ERROR RECOVERY ROUTINES

Bad data error recovery routines enable a user program to continue processing a dataset when bad data is encountered. Bad data means an unrecovered error was encountered while the dataset was being read. Skipping the data forces the dataset to a position past the bad data so no data is transferred to a user-specified buffer. Accepting the data causes the bad data to be transferred to a user-specified buffer and the dataset is positioned immediately following the bad data.

When an unrecovered data error is encountered, the user continues processing by calling either SKIPBAD or ACPTBAD.

SKIPBAD allows the user to skip the bad data so no bad data is sent to the user-specified buffer.

Call from FORTRAN:

```
CALL SKIPBAD(dn,blocks,termcnd)
```

dn	Dataset name or unit number
blocks	On exit, contains the number of blocks skipped. The dataset format determines the meaning of block (see the CRAY-OS Version 1 Reference Manual, publication SR-0011, for definitions of the tape block formats). The format can be: ● Interchange. Number of physical tape blocks skipped. ● Blocked transparent. Number of sectors or 512-word blocks skipped. ● Unblocked. Number of sectors or 512-word blocks skipped.
termcnd	On exit, address of termination condition. For an unblocked dataset, only a negative value (not at EOD) or a 2 (at EOD) is returned. <0 Not positioned at a record control word (RCW) =0 Positioned at EOR >0 If 1, positioned at EOF If 2, positioned at EOD

ACPTBAD makes bad data available to the user by transferring it to the user-specified buffer.

- Interchange. The portion of bad data between the current position and the next control word is transferred (no control words are transferred).

- Blocked transparent. The entire sector or 512-word block following the current position is transferred (control words are transferred). The possibility of having a partial block (<512 words) exists.

- Unblocked. The entire sector or 512-word block is read into the user data area and then transferred to a user-specified buffer. The bad data is placed after the good data from the read. The user can specify the word address of the user-supplied buffer to be the same as the address immediately following the good data. This specification prevents transfer to a user-specified buffer.

Call from FORTRAN:

```
CALL ACPTBAD(dn,uda,wrdcnt,termcnd,ubcnt)
```

dn	Dataset name or unit number
uda	User data area to receive the bad data; length must be 512 words.
wrdcnt	On exit, number of words transferred (≤512)
termcnd	On exit, address of termination condition. This is defined for datasets in interchange and blocked transparent format. <0 Not positioned at a record control word (RCW) =0 Positioned at EOR >0 If 1, positioned at EOF If 2, positioned at EOD
ubcnt	On exit, address of unused bit count. Only defined if termination condition is 0, and *wrdcnt* is nonzero.

CHARACTER ROUTINES

Character routines load and store character items, find the beginning of a format, and increment character addresses. These routines are used by I/O routines. $LCI and $SCI are used by character routines and I/O routines, and $FFS and $UIO are used by I/O routines only.

$LCI loads a character item.

Call from CAL:

```
CALLV $LCI
```

Entry:

(S1)	Character address pointer

Exit:

(A1)	Length of item in words
(VL)	Length of item in words
(V1)	Character item adjusted to word boundary
(S1)	Size in characters

$SCI stores a character item.

Call from CAL:

```
CALLV $SCI
```

Entry:	
(S1)	Address of character item
(V1)	Place to store, adjusted to word boundary; blank fill

$FFS finds the start of a format.

Call from CAL:

```
CALLV $FFS
```

Entry:	
(S1)	Address of format
Exit:	
(A0)	0 if starting (is found Nonzero if non-numeric character is found before (
(A1)	Word address of starting (
(A2)	Character position of starting (

$UIO increments character address.

Call from CAL:

```
CALLV $UIO
```

Entry:	
(S1)	Address of character address pointers
(S2)	Increment between items

NUMERIC CONVERSION ROUTINES

Numeric conversion routines convert a character to a numeric format or a number to a character format.

****

CAUTION

$NICV and $NOCV are obsolete. NICV% and NOCV% should be used in their place.

****

NICV% and NICONV perform numeric integer input conversion to character format.

Call from CAL:

```
CALLV NICV%
```

Entry:

(A1)	First input character address
(S3)	Field width (optional)
(S4)	Decimal places (optional)
(S5)	P factor (optional)
(S6)	Mode in bits 49 to 63 (symbols defined in $IOLIB). See table 5-5 for description of bits.

Exit:

(A1)	Last input character address+1
(S1,S2)	Binary result
(S3)	Next character
(S4)	Return conditions. See table 5-6 for descriptions.

Call from FORTRAN:

CALL NICONV(*fca*,*fw*,*dp*,*pf*,*mode*,*br*,*stat*)

fca	First input character address; on exit, last input character address +1
fw	Field width in characters
dp	Decimal places
pf	P factor
mode	Conversion mode. See table 5-5 for description of bits.

br	High-order binary result followed by low-order binary result. This parameter must be two Cray 64-bit words long.
stat	Errors and warnings returned from the conversion routines. See table 5-6 for descriptions.

Table 5-5. Conversion mode descriptions

Bit	Symbol	Description
49	SVDPART	Set if decimal places field present
50	SVCFT	Set if called from library; 0 if called from CFT.
51	SVPLS	Set if requested + sign output
52	SVEXPS	Set if exponent size defined
53	SVSEXP	Sign of exponent
54	SVSMAN	Sign of fraction or number
55	SVDFLD	D conversion
56	SVGFLD	G conversion
57	SVEFLD	E conversion
58	SVFFLD	F conversion
59	SVIFLD	I conversion
60	SVZFLD	Hex conversion
61	SVOFLD	Octal conversion
62, 63	SVBZR,SVBNL	Blank conversion indicators: 00 Blanks treated as delimiters 01 Blanks ignored (as in FORTRAN source) 10 Blanks treated as zeros (as in FORTRAN run-time input)

NOCV% and NOCONV perform numeric output conversion.

Call from CAL:

```
CALLV NOCV%
```

Entry:

(A1)	First output character address
(A2)	Size of exponent (if SVEXPS bit set)
(S1),(S2)	Binary number to be output
(S3)	Field width
(S4)	Decimal places

(S5)	P factor
(S6)	Mode in bits 49 to 63 (symbols defined in $IOLIB). See table 5-5 for description of bits.

Exit:

(A1)	Last output character address

Table 5-6. Conversion return conditions

Mode	Description
O'0077	Typeless
O'4027	24-bit integer
O'4077	64-bit integer
O'6077	64-bit real
O'6177	128-bit real
-1	Illegal character
-2	Overflow
-3	Exponent underflow
-4	Exponent overflow
-5	Null field

Call from FORTRAN:

```
CALL NOCONV(fca,fw,dp,pf,mode,sexp,br,status
```

fca	First output character address; on exit, last output character address.
fw	Field width in characters
dp	Decimal places (cannot apply to conversion)
pf	P factor
mode	Conversion mode. See table 5-5 for bit descriptions.
sexp	Size of exponent if Mode flag is set
br	Binary number to be converted. High-order binary value followed by the low-order value. This parameter must be two Cray 64-bit words long.

stat	Errors and warnings returned from the conversion routines

≥0	No error in conversion
<0	Error in conversion. (Current version of numeric output conversion has no error conditions to return.)

$NICV performs numeric input conversion. This routine is obsolete.

Call from CAL:

```
R $NICV
```

Entry:

(A3)	Field width (optional)
(A4)	Decimal places (optional)
(A5)	P factor (optional)
(A6)	First character address
(S6)	Mode in bits 49 to 63 (symbols defined in $IOLIB). See table 5-5 for bit descriptions.

Exit:

(A1)	Next character
(A2)	Mode (in octal). See table 5-6 for descriptions.
(A6)	Last input character address+1
(S1)	High-order result
(S2)	Low-order result

$NOCV performs numeric output conversion. This routine is obsolete.

Call from CAL:

```
R $NOCV
```

Entry:

(A3)	Field width
(A4)	Decimal places
(A5)	P factor
(A7)	First output character address
(S1),(S2)	Binary number to be output
(S6)	Conversion mode in bits 49 to 61 (symbols defined in $IOLIB). See table 5-5 for bit descriptions.

Exit:

(A7)	Last output character address

RANDOM ACCESS DATASET I/O ROUTINES

Sequentially accessed datasets are used for applications that read input to a job once at the start of the process and write output to a job once at the end of the process. However, when large numbers of intermediate results are used randomly as input in later stages of jobs, a random access dataset capability is more efficient to use than sequential access. A random access dataset consists of records that are accessed and changed in the same manner. Random access of data removes the slow processing and inconvenience of sequential access, particularly when the order of reading and writing records differs in various applications or when I/O speed is important.

Random access dataset I/O routines allow the user to specify how records of a dataset are to be changed without the usual limitations of sequential access. Only those I/O routines meant for each type of dataset can be used with predictable results.

Random access datasets can be created and accessed by the record-addressable dataset routines (READMS/WRITMS, READDR/WRITDR) or the word-addressable dataset routines (GETWA/PUTWA).

NOTE

Generally, random access dataset I/O routines used in a program with overlays or segments should reside in the root segment or first overlay. However, if all I/O is done within one overlay, the routines can reside in that overlay. If all I/O is done in that overlay's successor, the routines can reside in the successor overlay.

RECORD-ADDRESSABLE, RANDOM ACCESS DATASET I/O ROUTINES

Record-addressable, random access dataset I/O routines allow the user to generate datasets containing variable-length, individually addressable records. These records can be read and rewritten at the user's discretion. The library routines update indexes and pointers.

The random access dataset information is stored in two places: in an array in user memory and at the end of the random access dataset.

When a random access dataset is opened, an array in user memory contains the master index to the records of the dataset. This master index contains the pointers, and optionally the names of the records within the dataset. Although this storage area is provided by the user, it must be modified only by the random access dataset I/O routines.

When a random access dataset is closed and optionally saved, the storage area containing the master index is mapped to the end of the random access dataset, thus recording changes to the contents of the dataset.

The following FORTRAN-callable routines can change or access a record-addressable, random access dataset: OPENMS, WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, WAITMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, WAITDR, ASYNCDR, SYNCDR and STINDX.

The READDR/WRITDR random access I/O routines are direct-to-disk versions of READMS/WRITMS. All input or output goes directly to or from the user's data area from or to the mass storage dataset without passing through a system maintained buffer in high memory. Since mass storage can only be addressed in even 512 word blocks, all record lengths are rounded up to the next multiple of 512 words.

Users can intermix both READMS/WRITMS and READDR/WRITDR datasets in the same program. Do not use the same file in both packages at the same time.

OPENMS/OPENDR opens a local dataset and specifies the dataset as a random access dataset that can be accessed or changed by the record-addressable, random access dataset I/O routines. If the dataset does not exist, the master index contains zeros; if the dataset does exist, the master index is read from the dataset. The master index contains the current index to the dataset. The current index is updated when the dataset is closed using CLOSMS/CLOSDR.

A single job can use up to 40 active READMS/WRITMS files and 20 READDR/WRITDR files.

Call from FORTRAN:

```
CALL OPENMS(dn,index,length,it[,ierr])
CALL OPENDR(dn,index,length,it[,ierr])
```

dn	Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset FT07).
index	Type INTEGER array. The name of the array in the user's program that is going to contain the master index to the records of the dataset. This array must be changed only by the random access dataset I/O routines.

length Type INTEGER variable, expression, or constant. The length of the index array. The length of *index* depends upon the number of records on or to be written on the dataset using the master index and the type of master index. The *length* must be at least 2**nrec* if *it*=1 or *nrec* if *it*=0. *nrec* is the number of records in or to be written to the dataset using the master index.

it Type INTEGER variable, expression, or constant flag indicating the type of master index.

- *it*=0 Records synchronously referenced with a number between 1 and *length*.
- *it*=1 Records synchronously referenced with an alphanumeric name of eight or fewer characters.
- *it*=2 Records asynchronously referenced with a number between 1 and *length*.
- *it*=3 Records asynchronously referenced with an alphanumeric name of eight or fewer characters.

For a named index, odd numbered elements of the index array contain the record name, and even numbered elements of the index array contain the pointers to the location of the record within the dataset. For a numbered index, a given index array element contains the pointers to the location of the corresponding record within the dataset.

ierr Type INTEGER variable. Error control and code. If *ierr* is supplied on the call to OPENMS/OPENDR, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error aborts the job.

If the user sets *ierr*>0 on input to OPENMS/OPENDR, error messages are not placed in the logfile. Otherwise, an error code is returned, and the error message is added to the job's logfile. OPENMS/OPENDR writes an open message to the logfile whether the value of *ierr* selects log messages or not.

On output from OPENMS/OPENDR:

- *ierr*=0 No errors detected
- <0 Error detected. *ierr* contains one of the error codes in table 5-7.

Table 5-7. Error codes for record-addressable, random access dataset I/O routines

Code Number	Routines Affected	Description
-1	OPENDR OPENMS WRITDR WRITMS READDR READMS STINDR STINDX CLOSDR CLOSMS CHECKDR CHECKMS WAITDR WAITMS ASYNCDR ASYNCMS SYNCDR SYNCMS	The dataset name or unit number is illegal.
-2	OPENDR OPENMS	The user-supplied index length is less than or equal to 0.
-3	OPENDR OPENMS	The number of datasets has exceeded memory or size availability.
-4	OPENDR OPENMS	The dataset index length read from the dataset is greater than the user-supplied index length (nonfatal message).
-5	OPENDR OPENMS	The user-supplied index length is greater than the index length read from dataset (nonfatal message).

Table 5-7. Error codes for record-addressable, random access dataset I/O routines (continued)

Code Number	Routines Affected	Description
-6	WRITMS READMS FINDMS	The user-supplied named index is illegal.
-7	WRITMS READMS	The named record index array is full.
-8	WRITMS READMS FINDMS	The index number is greater than the maximum on the dataset.
-9	WRITMS READMS	Rewrite record exceeds the original.
-10	READMS FINDMS	The named record was not found in the index array.
-11	OPENMS	The index word address read from the dataset is less than or equal to 0.
-12	OPENMS	The index length read from the dataset is less than 0.
-13	OPENMS	The dataset has a checksum error.
-14	OPENMS	OPENMS already has opened the dataset.
-15	WRITMS WRITDR READMS CLOSMS STINDR STINDX	OPENMS/OPENDR was not called on this dataset.

Table 5-7. Error codes for record-addressable, random access dataset I/O routines (continued)

Code Number	Routines Affected	Description
-15	FINDMS CHECKMS CHECKDR WAITMS WAITDR ASYNCMS ASYNCDR SYNCMS SYNCDR WAITDR WAITMS	OPENMS/OPENDR was not called on this dataset.
-16	STINDR STINDX	A STINDX/STINDR call cannot change the index type.
-17	WRITMS WRITDR READMS FINDMS	The index entry is less than or equal to 0 in the user's index array.
-18	WRITMS WRITDR READMS FINDMS	The user-supplied word count is less than or equal to 0.
-19	WRITMS WRITDR READMS FINDMS	The user-supplied index number is less than or equal to 0.
-20	OPENMS OPENDR	Dataset created by WRITDR/WRITMS

WRITMS/WRITDR writes data from user memory to a record in a random access dataset on disk and updates the current index.

Call from FORTRAN:

```
CALL WRITMS(dn,ubuff,n,irec,rrflag,s[,ierr])
CALL WRITDR(dn,ubuff,n,irec,rrflag,s[,ierr])
```

dn	Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (that is, *dn*=7 corresponds to dataset FT07).
ubuff	Type determined by user. The location of the first word in the user's program to be written to the record.
n	Type INTEGER variable, expression, or constant. The number of words to be written to the record. *n* contiguous words from memory, beginning at *ubuff*, are written to the dataset record. Since COS unblocked dataset I/O is in multiples of 512 words, it is recommended that *n* be a multiple of 512 words when speed is of importance. However, the random access dataset I/O routines support a record length other than multiples of 512 words. WRITDR rounds *n* up to the next multiple of 512 words, if necessary.
irec	Type INTEGER variable, expression, or constant. The record number or record name of the record to be written. A record name is limited to a maximum of eight characters. For a numbered index, *irec* must be between 1 and the length of the index declared in the OPENMS/OPENDR call. For a named index, *irec* is any 64-bit entity the user specifies.
rrflag	Type INTEGER variable, expression, or constant. A flag indicating record rewrite control. *rrflag* can be one of the following codes. 0 Write the record at EOD. 1 If the record already exists and the new record length is less than or equal to the old record length, rewrite the record over the old record. If the new record length is greater than the old, abort the job step or return the error code in *ierr*. If the record does not exist, the job aborts or the error code is returned in *ierr*.

-1 If the record exists and its new length does not exceed the old length, write the record over the old record. Otherwise, write the record at EOD.

s Type INTEGER variable, expression, or constant. A sub-index flag.[†]

ierr Type INTEGER variable. Error control and code. If *ierr* is supplied on the call to WRITMS/WRITDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile.

On output from WRITMS/WRITDR:

ierr=0	No errors detected
<0	Error detected. *ierr* contains one of the error codes in table 5-7.

READMS/READDR reads a record from a random access dataset to a contiguous memory area in the user's program.

//

WARNING

If you are using READDR in asynchronous mode and the record size is not a multiple of 512 words, user data can be overwritten and not restored. With the SYNCDR routine, the dataset can be switched to read synchronously, causing data to be copied out and restored after the read has completed.

//

Call from FORTRAN:

```
CALL READMS(dn,ubuff,n,irec[,ierr])
CALL READDR(dn,ubuff,n,irec[,ierr])
```

[†] Deferred implementation

dn	Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *unit*=7 corresponds to dataset FT07).
ubuff	Type specified by user. The location in the user's program where the first word of the record is placed.
n	Type INTEGER variable, expression, or constant. The number of words to be read. *n* words are read from the random access record *irec* and placed contiguously in memory, beginning at *ubuff*. If necessary, READDR rounds *n* up to the next multiple of 512 words. If the file is in synchronous mode, the data is saved and restored after the read.
irec	Type INTEGER variable, expression, or constant. The record number or record name of the record to be read. A record name is limited to a maximum of eight characters. For a numbered index, *irec* must be between 1 and the length of the index declared in the OPENMS/OPENDR call. For a named index, *irec* is any 64-bit entity the user specifies.
ierr	Type INTEGER variable. Error control and code. If *ierr* is supplied on the call to READMS/READDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from READMS/READDR: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-7.

CLOSMS/CLOSDR writes the master index specified in OPENMS/OPENDR from the user's program area to the random access dataset and then closes the dataset. Statistics are collected on the activity of the random access dataset and written in a readable format to dataset $STATS. (See table 5-8). The statistics can be written to $OUT by using the following control statements or their equivalent after the random access dataset has been closed by CLOSMS/CLOSDR.

```
REWIND,DN=$STATS.
COPYF,I=$STATS,O=$OUT.
```

Call from FORTRAN:

```
CALL CLOSMS(dn[,ierr])
CALL CLOSDR(dn[,ierr])
```

dn	Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (that is, *dn*=7 corresponds to dataset FT07).
ierr	Type INTEGER variable. Error control and code. If *ierr* is supplied on the call to CLOSMS/CLOSDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile.

CLOSMS/CLOSDR writes a message to $LOG upon closing the dataset whether or not the user has requested error messages to be written to the logfile.

On output from CLOSMS/CLOSDR:

ierr=0	No errors detected
<0	Error detected. *ierr* contains one of the error codes in table 5-7.

****

CAUTION

If a job step terminates without closing the random access dataset with CLOSMS/CLOSDR, the dataset integrity is questionable.

****

Table 5-8. CLOSMS statistics

Message	Description
TOTAL ACCESSES =	Number of accesses
READS =	Number of reads
WRITES =	Number of writes

Table 5-8. CLOSMS statistics (continued)

Message	Description
SEQUENTIAL READS =	Number of sequential reads
SEQUENTIAL WRITES =	Number of sequential writes
REWRITES IN PLACE =	Number of rewrites in place
WRITES TO EOI =	Number of writes to EOI
TOTAL WORDS MOVED =	Number of words moved
MINIMUM RECORD =	Minimum record size
MAXIMUM RECORD =	Maximum record size
TOTAL ACCESS TIME =	Total access time
AVERAGE ACCESS TIME =	Average access time

STINDX/STINDR allows an index other than the master index to be used as the current index by creating a sub-index. STINDX/STINDR reduces the amount of memory needed by a dataset containing a large number of records. It also maintains a dataset containing records logically related to each other. Records in the dataset, rather than records in the master index area, hold secondary pointers to records in the dataset.

STINDX/STINDR allows more than one index to manipulate the dataset. Generally, STINDX/STINDR toggles the index between the master index (maintained by OPENMS/OPENDR and CLOSMS/CLOSDR) and a sub-index (supplied and maintained by the user).

The user is responsible for maintaining and updating sub-index records stored in the dataset. Records in the dataset can be accessed and changed only by the current index.

After a STINDX/STINDR call, subsequent calls to READMS/READDR and WRITMS/WRITDR use and alter the current index array specified in the STINDX/STINDR call. The user saves the sub-index by calling STINDX/STINDR with the master index array, then writes the sub-index array to the dataset using WRITMS/WRITDR. Retrieving the sub-index is performed by calling READMS/READDR on the record containing the sub-index information. STINDX/STINDR thus allows logically infinite index trees into the dataset and reduces the amount of memory needed for a random access dataset containing many records.

****

CAUTION

When generating a new sub-index (for example, building a database), the array or memory area used for the sub-index must be set to 0. If the sub-index storage is not set to 0, unpredictable results occur.

****

Call from FORTRAN:

```
CALL STINDX(dn,index,length,it[,ierr])
CALL STINDR(dn,index,length,it[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset corresponding to COS conventions as a Hollerith constant or the unit number of the file (that is, *unit*=7 corresponds to dataset FT07).
index	Type integer array. The user-supplied array used for the sub-index or new current index. If *index* is a sub-index, it must be a storage area that does not overlap the area used in OPENMS/OPENDR to store the master index.
length	Type integer variable, expression, or constant. The length of the index array. The length of *index* depends upon the number of records on or to be written on the dataset using the master index and the type of master index. If *it*=1, *length* must be at least twice the number of records on or to be written to the dataset using *index*. If *it*=0, *length* must be at least the number of records on or to be written to the dataset using *index*.
it	Type integer variable, expression, or constant. A flag to indicate the type of index. When *it*=0, the records are referenced with a number between 1 and *length*. When *it*=1, the records are referenced with an alphanumeric name of eight or fewer characters. For a named index, odd-numbered elements of the index array contain the record name, and even-numbered elements of the index array contain the pointers to the location of the record within the dataset. For a numbered index, a given index array element contains the pointers to the location of the corresponding record within the dataset. The index type defined by STINDX/STINDR must be the same as that used by OPENMS/OPENDR.

ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to STINDX/STINDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from STINDX/STINDR: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-7.

FINDMS asynchronously reads the desired record into the data buffers used by the random access dataset routines for the specified dataset. The next READMS or WRITMS call waits for the read to complete and transfers data appropriately.

Call from FORTRAN:

```
CALL FINDMS(dn,n,irec[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (that is, *dn*=7 corresponds to dataset FT07).
n	Type integer variable, expression, or constant. The number of words to be read as in READMS or WRITMS.
irec	Type integer variable, expression, or constant. The record name or number as in READMS or WRITMS to be read into the data buffers.
ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to FINDMS, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from FINDMS: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-7.

ASYNCMS/ASYNCDR sets the I/O mode for the random access routines to be asynchronous. Therefore, input/output operations can be initiated and subsequent execution can proceed simultaneously with the actual data transfer. With READMS, asynchronous reads should be done with the FINDMS routine.

Call from FORTRAN:

```
Call ASYNCMS(dn[,ierr])
Call ASYNCDR(dn[,ierr])
```

dn Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset FT07).

ierr Type integer variable. Error control and code. If *ierr* is supplied on the call to ASYNCMS/ASYNCDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile.

On output from ASYNCMS/ASYNCDR:

ierr=0	No errors detected
<0	Error detected. *ierr* contains one of the error codes in table 5-7.

SYNCMS/SYNCDR sets the I/O mode for the random access routines to be synchronous. All input/output operations wait for completion.

Call from FORTRAN:

```
CALL SYNCMS(dn[,ierr])
CALL SYNCDR(dn[,ierr])
```

dn Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset FT07).

ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to SYNCMS/SYNCDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from SYNCMS/SYNCDR: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-7.

CHECKMS/CHECKDR checks the status of an asynchronous random access input or output operation. A status flag is returned to the user, indicating whether the specified dataset is active.

Call from FORTRAN:

```
CALL CHECKMS(dn,istat[,ierr])
CALL CHECKDR(dn,istat[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset. (For example, *dn*=7 corresponds to dataset FT07.)
istat	Type integer variable. Dataset I/O Activity flag. *istat*=0 No I/O activity on the specified dataset. *istat*=1 I/O activity on the specified dataset
ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to CHECKMS/CHECKDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from CHECKMS/CHECKDR: *ierr*=0 No error detected <0 Error detected. *ierr* contains on of the error codes in table 5-7.

WAITMS/WAITDR waits for the completion of an active asynchronous input or output operation. A status flag is returned to the user, indicating whether the I/O on the specified dataset was completed without error.

Call from FORTRAN:

```
CALL WAITMS(dn,istat[,ierr])
CALL WAITDR(dn,istat[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset. (For example, *dn*=7 corresponds to dataset FT07.)
istat	Type integer variable. Dataset error flag. *istat*=0 No error occurred during the asynchronous I/O operation. *istat*=1 Error occurred during the asynchronous I/O operation.
ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to WAITMS/WAITDR, *ierr* returns any error codes to the user. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned and the message is added to the job's logfile. On output from WAITMS/WAITDR: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-7.

The following examples show some of the features and uses of random access dataset routines.

Example 1:

In the program SORT, a sequence of records is read in and then printed out as a sorted sequence of records.

```
  1        PROGRAM SORT
  2        INTEGER IARRAY (512)
  3        INTEGER INDEX (512), KEYS (100)
  4        CALL OPENMS ('SORT',INDEX,255,1)
  5        N=50
   C READ IN RANDOM ACCESS RECORDS FROM UNIT "SORT"
  6        DO 21 I=1,N
  7        READ(5,1000) (IARRAY(J),J=1,512)
  8        NAME=IARRAY(1)
  9        KEYS(I)=IARRAY(1)
 10        CALL WRITMS ('SORT',IARRAY,512,NAME,0)
 11     21 CONTINUE
   C SORT KEYS ALPHABETICALLY IN ASCENDING ORDER USING EXCHANGE SORT
 12        DO 23 I=1,N-1
 13        MIN=I
 14        J=I+1
 15        DO 22 K=J,N
 16           IF (KEY(K).LT.KEYS(MIN)) MIN=K
 17     22 CONTINUE
 18        IB=KEYS(I)
 19        KEYS(I)=KEYS(MIN)
 20        KEYS(MIN)=IB
 21     23 CONTINUE
   C WRITE OUT RANDOM ACCESS RECORDS IN ASCENDING
   C ALPHABETICAL ORDER
 22        DO 24 I=1,N
 23        NAME=KEYS(I)
 24        CALL READMS ('SORT',IARRAY,512,NAME)
 25        WRITE(6,5120) (IARRAY(J),J=1,512)
 26     24 CONTINUE
 27 1000 FORMAT ("......")
 28 5120 FORMAT (1X,"......")
 29        CALL CLOSMS ('SORT')
 30        STOP
 31        END
```

In this example, the random access dataset is initialized as shown in line 4. Lines 6 through 11 show that from unit 5 a record is read into array IARRAY and then written as a record to the random access dataset SORT. The first word of each record is assumed to contain an 8-character name to be used as the name of the record.

Lines 12 through 21 show that the names of the records are sorted in the array KEYS. Lines 22 through 26 show that the records are read in and then printed out in alphabetical order.

Example 2:

The programs INITIAL and UPDATE show how the random access dataset might be updated without the usual search and positioning of a sequential access dataset.

Program INITIAL:

```
     PROGRAM INITIAL
     INTEGER IARRAY(512)
     INTEGER INDEX (512)
C
C OPEN RANDOM ACCESS DATASET
C THIS INITIALIZES THE RECORD KEY "INDEX"
C
     CALL OPENMS ('MASTER',INDEX,101,1)
C
C READ IN RECORDS FROM UNIT 6 AND
C WRITE THEM TO THE DATASET "MASTER"
C
     DO 10 I=1,50
     READ(6,600) (IARRAY(J),J=1,512)
     NAME=IARRAY(1)
     CALL WRITMS ('MASTER',IARRAY,512,NAME,0,0)
 10  CONTINUE
C
C CLOSE "MASTER" AND SAVE RECORDS FOR UPDATING
C
     CALL CLOSMS ('MASTER')
 600 FORMAT (1X,'.....')
     STOP
     END
```

Program UPDATE:

```
     PROGRAM UPDATE
     INTEGER INEWRCD(512)
     INTEGER INDX (512)
C
C OPEN RANDOM ACCESS DATASET CREATED IN THE
C PREVIOUS PROGRAM "INITIAL"
C
C INDX WILL BE WRITTEN OVER THE OLD RECORD KEY
C
     CALL OPENMS ('MASTER',INDX,101,1)
C
C READ IN NUMBER OF RECORDS TO BE UPDATED
C
     READ (6,610) N
```

```
    C
    C READ IN NEW RECORDS FROM UNIT 6 AND
    C WRITE THEM IN PLACE OF THE OLD RECORD THAT HAS
    C THAT NAME
    C
   6     DO 10 I=1,N
   7     READ(6,600) (INEWRCD(J),J=1,512)
   8     NAME=INEWRCD(1)
   9     CALL WRITMS ('MASTER',INEWRCD,512,NAME,1,0)
  10 10 CONTINUE
    C
    C CLOSE "MASTER" AND SAVE NEWLY UPDATED RECORDS
    C FOR FURTHER UPDATING
    C
  11     CALL CLOSMS ("MASTER")
  12 600 FORMAT (1X,"......")
  13 610 FORMAT (1X,"......")
  14     STOP
  15     END
```

In this example, program INITIAL creates a random access dataset on unit MASTER; then program UPDATE replaces particular records of this dataset without changing the remainder of the records.

Line 10 shows that the call to CLOSMS at the end of INITIAL caused the contents of INDEX to be written to the random access dataset.

Line 4 shows that the call to OPENMS at the beginning of UPDATE has caused the record key of the random access dataset to be written to INDX. The random access dataset and INDX are the same now as the random access dataset and INDEX at the end of INITIAL.

Lines 6 through 10 show that certain records are replaced.

Example 3:

The program SNDYMS is an example of the use of the secondary index capability, using STINDX. In this example, dummy information is written to the random access dataset.

```
   1     PROGRAM SNDYMS
   2     IMPLICIT INTEGER (A-Y)
   3     DIMENSION PINDEX(20),SINDEX(30),ZBUFFR(50)
   4     DATA PLEN,SLEN,RLEN /20,30,50/
    C OPEN THE DATASET.
   5     CALL OPENMS (1,PINDEX,PLEN,0,ERR)
```

```
           IF (ERR.NE.0) THEN
            PRINT*,' Error on OPENMS, err=',ERR
            STOP 1
           ENDIF
C LOOP OVER THE 20 PRIMARY INDICES.  EACH TIME
C A SECONDARY INDEX IS FULL, WRITE THE
C SECONDARY INDEX ARRAY TO THE DATASET.
           DO 40 K=1,PLEN
C ZERO OUT THE SECONDARY INDEX ARRAY.
           DO 10 I=1,SLEN
        10 SINDEX(I)=0
C CALL STINDX TO CHANGE INDEX TO SINDEX.
           CALL STINDX (1,SINDEX,SLEN,0,ERR)
           IF (ERR.NE.0) THEN
            PRINT*,' Error on STINDX, err=',ERR
            STOP 2
           ENDIF
C WRITE SLEN RECORDS.
           DO 30 J=1,SLEN
C GENERATE A RECORD LENGTH BETWEEN 1 AND RLEN.
           TRLEN=MAX0(IFIX(RANF(0)*FLOAT(RLEN)),1)
C FILL THE "DATA" ARRAY WITH RANDOM FLOATING POINT NUMBERS.
           DO 20 I=1,TRLEN
        20 ZBUFFR(I)=(J+SIN(FLOAT(I)))**(1.+RANF(0))
           CALL WRITMS (1,ZBUFFR,TRLEN,J,-1,DUMMY,ERR)
           IF (ERR.NE.0) THEN
            PRINT*,' Error on WRITMS, err=',ERR
            STOP 3
           ENDIF
        30 CONTINUE
C "TOGGLE" THE INDEX BACK TO THE MASTER AND
C WRITE THE SECONDARY INDEX TO THE DATASET.
           CALL STINDX (1,PINDEX,PLEN,0)
C NOTE THE ABOVE STINDX CALL DOES NOT USE THE
C OPTIONAL ERROR PARAMETER, AND WILL ABORT
C IF STINDX DETECTS AN ERROR.
           CALL WRITMS (1,SINDEX,SLEN,K,-1,DUMMY,ERR)
           IF (ERR.NE.0) THEN
            PRINT*,' Error on STINDX, err=',ERR
            STOP 4
           ENDIF
 40 CONTINUE
C CLOSE THE DATASET.
           CALL CLOSMS (1,ERR)
           IF (ERR.NE.0) THEN
            PRINT*,' Error on CLOSMS, err=',ERR
            STOP 5
           ENDIF
           STOP 'Normal'
           END
```

WORD-ADDRESSABLE, RANDOM ACCESS DATASET I/O ROUTINES

A word-addressable, random access dataset consists of an adjustable number of contiguous words. Any word or contiguous sequence of words is accessible from a word-addressable, random access dataset using the associated word-addressable, random access I/O routines. These datasets and their I/O routines are similar to the record-addressable, random access datasets and their I/O routines. The FORTRAN-callable word-addressable, random access I/O routines are: WOPEN, WCLOSE, PUTWA, APUTWA, GETWA and SEEK. WOPEN opens a dataset and specifies it as a word-addressable, random access dataset that can be accessed or changed with the word-addressable I/O routines. The WOPEN call is optional. If a user call to GETWA or PUTWA is executed first, the dataset is opened for the user with the default number of blocks (16) and *istats* turned on.

Call from FORTRAN:

CALL WOPEN(*dn*,*blocks*,*istats*[,*ierr*])

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to FT07).
blocks	Type integer variable, expression, or constant. The maximum number of 512-word blocks that the word-addressable package can use for a buffer.
istats	Type integer variable, expression, or constant. If *istats* is nonzero, then statistics about the changes and accesses to the dataset *dn* are collected. (See table 5-9 for information about the statistics that are collected.) These statistics are written to dataset $STATS and can be written to $OUT by using the following control statements or their equivalent after the dataset has been closed by WCLOSE.

```
REWIND,DN=$STATS.
COPYD,I=$STATS,O=$OUT.
```

ierr Type integer variable. Error control and code. If *ierr* is supplied on the call to WOPEN, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error aborts the job.

On output from WOPEN:
ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the error codes in table 5-10.

Table 5-9. WOPEN statistics

Message	Description
BUFFERS USED =	Number of 512-word buffers used by this dataset
TOTAL ACCESSES =	Number of accesses. This is the sum of the GETWA and PUTWA calls.
GETS =	Number of times the user calls GETWA
PUTS =	Number of times the user calls PUTWA
FINDS =	Number of times the user calls SEEK
HITS =	Number of times word addresses desired were resident in memory
MISSES =	Number of times no word addresses desired were resident in memory
PARTIAL HITS =	Number of times that some but not all of the word addresses desired were in memory
DISK READS =	Number of physical disk reads done
DISK WRITES =	Number of times a physical disk was written to
BUFFER FLUSHES =	Number of times buffers were flushed
WORDS READ =	Number of words moved from buffers to user
WORDS WRITTEN =	Number of words moved from user to buffers
TOTAL WORDS =	TOTAL WORDS. Sum of WORDS READ and WORDS WRITTEN
TOTAL ACCESS TIME =	Real time spent in disk transfers
AVER ACCESS TIME =	TOTAL ACCESS TIME divided by the sum of DISK READS and DISK WRITES
EOD BLOCK NUMBER =	Number of the last block of the dataset
DISK WORDS READ =	Count of number of words moved from disk to buffers
DISK WDS WRITTEN =	Count of number of words moved from buffers to disk

Table 5-9. WOPEN statistics (continued)

Message	Description
TOTAL DISK XFERS =	Sum of DISK WORDS READ and DISK WDS WRITTEN
BUFFER BONUS % =	TOTAL WORDS divided by value TOTAL DISK XFERS multiplied by 100

PUTWA writes a number of words from memory to a word-addressable, random access dataset.

APUTWA asynchronously writes a number of words from memory to a word-addressable, random-access dataset.

Call from FORTRAN:

```
CALL PUTWA(dn,source,addr,count[,ierr])
CALL APUTWA(dn, source, addr, count[, ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to FT07).
source	Variable or array of any type. The location of the first word in the user's program to be written to the dataset.
addr	Type integer variable, expression, or constant. The word location of the dataset that is to receive the first word from the user's program. *addr*=1 indicates beginning of file.
count	Type integer variable, expression, or constant. The number of words from *source* to be written.
ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to PUTWA, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error causes the job to abort. On output from PUTWA/APUTWA: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-10.

GETWA synchronously reads a number of words from the word-addressable, random access dataset into the user's memory. The SEEK routine performs asynchronous word-addressable input.

Call from FORTRAN:

```
CALL GETWA(dn,result,addr,count[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to FT07).
result	Variable or array of any type. The location in the user's program where the first word is placed.
addr	Type integer variable, expression, or constant. The word location of the dataset from which the first word is transferred.
count	Type integer variable, expression, or constant. The number of words from *result* written into the user's memory from the dataset.
ierr	Type integer variable. Error control and code. If *ierr* is supplied on the call to GETWA, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error causes the job to abort. On output from GETWA: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-10.

Table 5-10. Error codes for word-addressable, random access dataset I/O routines

Code Number	Routines Affected	Description
-1	WOPEN APUTWA PUTWA GETWA WCLOSE SEEK	Illegal unit number

Table 5-10. Error codes for word-addressable, random access dataset I/O routines (continued)

Code Number	Routines Affected	Description
-2	WOPEN APUTWA PUTWA GETWA SEEK	Number of datasets has exceeded memory or size availability.
-3	GETWA SEEK	User attempt to read past end of data
-4	APUTWA PUTWA GETWA SEEK	User-supplied word address less than or equal to 0
-5	APUTWA PUTWA GETWA SEEK	User-requested word count greater than maximum
-6	WOPEN APUTWA PUTWA GETWA WCLOSE SEEK	Illegal dataset name
-7	APUTWA PUTWA GETWA SEEK	User word count less than or equal to 0

WCLOSE finalizes the additions and changes to the word-addressable dataset and closes the dataset.

Call from FORTRAN:

```
CALL WCLOSE(dn[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to FT07).
ierr	Type integer variable, expression, or constant. Error control and code. If *ierr* is supplied on the call to WCLOSE, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error aborts the job.

On output from WCLOSE:

ierr=0	No errors detected
<0	Error detected. *ierr* contains one of the error codes in table 5-10.

SEEK asynchronously reads data into specified dataset buffers. The SEEK and GETWA calls are used together. The SEEK call reads the data asynchronously; the GETWA call waits for I/O to complete and then transfers the data. The SEEK call moves the last write operation pages from memory to disk, loading the user-requested word addresses to the front of the I/O buffers. The user can load as much data as fits into the dataset buffers. Subsequent GETWA and PUTWA calls that reference word addresses in the same range do not cause any disk I/O.

Call from FORTRAN:

```
CALL SEEK(dn,addr,count[,ierr])
```

dn	Type integer variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to FT07).
addr	Type integer variable, expression, or constant. The word address of the next read.
count	Type integer variable, expression, or constant. The number of words of the next read.

ierr	Type integer variable, expression, or constant. Error control and code. Error control and code. If *ierr* is supplied on the call to SEEK, *ierr* returns any error codes to the user. If *ierr* is not supplied, an error aborts the job. On output from SEEK: *ierr*=0 No errors detected <0 Error detected. *ierr* contains one of the error codes in table 5-10.

Example:

Assume a user wants to use a routine that reads word addresses 1,000,000 to 1,051,200. A dataset could be opened with 101 blocks of buffer space, and CALL SEEK(*dn*,1000000,51200,*ierr*) can be used before calling the routine. Subsequent GETWA or PUTWA calls with word addresses in the range of 1,000,000 to 1,051,200 do not trigger any disk I/O.

WORD-ADDRESSABLE I/O AND DATASET CONTROL ROUTINES

Word-addressable and I/O dataset control routines are normally called by FORTRAN I/O routines and communicate with the system through Exchange Processor requests.

OPENWA opens a word-addressable dataset.

Call from FORTRAN:

```
CALL OPENWA(dn,index,ieoi,iaddr,blocks,idsp)
```

dn	Dataset name or unit number
index	On exit, contains relative index into the Dataset Parameter Table (DSP) area where this dataset name goes
ieoi	On exit, contains the number of disk blocks
iaddr	On exit, contains the address of the buffer assigned to this dataset
blocks	Number of blocks of memory to request from the system
idsp	Address of DSP assigned

NOTE

Subsequent OPEN or CLOSE statements, field length changes, etc., can cause the DSP address to change.

CLOSEWA closes a word-addressable dataset.

Call from FORTRAN:

```
CALL CLOSEWA(index)
```

index	Offset in Dataset Parameter Table (DSP) area

READWA reads words from a word-addressable dataset.

Call from FORTRAN:

```
CALL READWA(index,buffer,address,cnt,rcl)
```

index	Offset in Dataset Parameter Table (DSP) area
buffer	Buffer to receive data
address	Word address on disk to read
cnt	Number of words to read
rcl	Recall flag. If 0, waits for completion.

WRITEWA writes words from a word-addressable dataset.

Call from FORTRAN:

```
CALL WRITEWA(index,buffer,address,cnt,rcl)
```

index	Offset into Dataset Parameter Table (DSP) area
buffer	Buffer to receive data

address	Word address on disk to write
cnt	Number of words to write
rcl	Recall flag. If 0, waits for completion.

WDSETB sets buffer address to current value.

Call from FORTRAN:

```
CALL WDSETB(dn,index,iaddr)
```

dn	Dataset name or unit number
index	Offset into Dataset Parameter Table (DSP) area
iaddr	Value to get buffer address

WDSET sets dataset names for word-addressable datasets.

Call from FORTRAN:

```
CALL WDSET(dn,idn)
```

dn	Dataset name or unit number
idn	On exit, returns dataset name or 0 if error

DATASET MANAGEMENT SUBPROGRAMS

6

INTRODUCTION

Dataset management subprograms provide the user with the means of managing permanent datasets, staging datasets, creating datasets, changing dataset attributes, and releasing datasets. These routines are grouped into control statement type, dataset search type, and I/O type.

CONTROL STATEMENT TYPE SUBPROGRAMS

The control statement type subprogram resembles job control language (JCL) control statements in name and purpose. However, a subprogram can be called from within a FORTRAN or CAL program while JCL control statements cannot. See the CRAY-OS Version 1 Reference Manual, publication SR-0011, for a description of the parameters.

Following is an example of a FORTRAN call to a control statement type subprogram.

```
EXAMPL='EXAMPL'L
IDC='PR'L
CALL ASSIGN(irtc,'DN'L,EXAMPL,'U'L,'MR'L,'DC'L,IDC)
```

irtc is an integer variable that contains a status code upon return. The status code is 0 if the call was successful.

This type of subprogram requires call-by-address subroutine linkage with the following calling sequence.

```
CALL SUBROUTINE NAME(stat,key)
```

stat	Returned status code
key	Keyword/value combinations in one of the following formats:

`'keyword'L,'value'L or 'keyword'L`

NOTE

See the CRAY-OS Version 1 Reference Manual, publication SR-0011 for control statements and their keywords.

PERMANENT DATASET MANAGEMENT (PDM) ROUTINES

Permanent dataset management routines access the COS Permanent Dataset Manager and return the status of the operation in parameter 1. The value is equal to 0 if no error condition exists, and not equal to 0 if an error condition does exist.

ACCESS associates a permanent dataset with the job.

ADJUST expands or contracts a permanent dataset.

DELETE removes a saved dataset. The dataset remains available for the life of the job.

MODIFY changes permanent dataset characteristics.

SAVE makes a dataset permanent and enters the dataset's identification and location into the Dataset Catalog.

PERMIT specifies user access mode to a user permanent dataset by other users.

DATASET STAGING ROUTINES

Dataset staging routines stage files to or from a front-end processor or to the Cray input queue. The job aborts if an error occurs.

ACQUIRE obtains a front-end resident dataset, stages it to the Cray mainframe, and makes it permanent and accessible to the job making the request.

DISPOSE directs a dataset to the specified front-end processor.

FETCH brings a front-end resident dataset to the Cray mainframe and makes the dataset local.

SUBMIT places a job dataset into the Cray input queue.

DEFINITION AND CONTROL ROUTINES

Definition and control routines allow changing of dataset attributes and creation and release of datasets. They return the status of the operation in parameter 1. The value of parameter 1 is 0 if no error condition exists and is not equal to 0 if an error condition exists.

ASSIGN opens dataset for reading and writing and assigns characteristics to it.

RELEASE closes a dataset, releases I/O buffer space, and renders it unavailable to the job.

SDACCESS allows a program to access datasets in the System Directory. This function has no control statement.

Call from FORTRAN:

```
CALL SDACCESS(istat,dn)
```

istat	An integer variable to receive the completion status (0 or 1). If 0, the dataset is a system dataset and has been accessed. If 1, the dataset is not a system dataset and has not been accessed.
dn	Name of the system dataset to be accessed

Example:

This example is in a general format.

```
    IF NOT (IFDNT(NAME)) THEN
       BEGIN
       CALL SDACCESS (STATUS, NAME);
       IF STATUS < > 0 THEN
          BEGIN
          OUTPUT ('***DATASET NOT AVAILABLE';
          BLANKFILL (NAME): A8);
          CALL ABORT;
          END
       END
```

DATASET SEARCH TYPE SUBPROGRAMS

$SDSP searches Dataset Parameter Table (DSP) for a dataset name and returns DSP address.

Call from CAL:

```
CALLV $SDSP
```

Entry:

(S1)	Dataset name or alias (ASCII, left-justified, zero-filled)

Exit:

(S1)	Dataset name or alias
(A0)	Return code, positive if found; negative if not found.
(A1)	Dataset Parameter Table (DSP) address

$SLFT searches Logical File Table (LFT) for dataset name and returns LFT address.

Call from CAL:

```
CALLV $SLFT
```

Entry:
  (S1)   Dataset name (ASCII, left-justified, zero-filled)
Exit:
  (S1)   Dataset name
  (A0)   Return code, positive if found; negative if not found.
  (A2)   LFT address

ADDLFT and $ALF add a name to the Logical File Table (LFT).

Call from FORTRAN:

```
CALL ADDLFT(dn,dsp)
```

dn	Name to add to Logical File Table (LFT)
dsp	Dataset Parameter Table (DSP) address for name

Call from CAL:

```
CALLV $ALF
```

Entry:
  (S1)   Name to add to Logical File Table (LFT)
  (A1)   Dataset Parameter Table (DSP) address for name
Exit:
  No arguments returned

$DSNDSP searches Logical File Table in the user's I/O area for dataset name and returns Dataset Parameter Table (DSP) address.

Call from CAL:

```
CALLV $DSNDSP
```

Entry:
  (S1)   Dataset name (ASCII, left-justified, zero- or blank-filled)
Exit:
  (S1)   Dataset name
  (A1)   Dataset Parameter Table (DSP) address (0 if not found)

GETDSP and $GTDSP% search for a Dataset Parameter Table (DSP) address. If none is found, a DSP is created.

Call from CAL:

```
CALLV $GTDSP%
```

Entry:

(A1)	Address of dataset name or unit number

Exit:

(A1)	Address of Dataset Parameter Table (DSP)
(S1)	Negative DSP offset relative to base of DSPs if system area DSP; DSP address if user area DSP.
(S2)	Dataset name (ASCII, left-justified, blank-filled)

Call from FORTRAN:

```
CALL GETDSP(unit,dsp,ndsp,dn)
```

unit	Dataset name or unit number
dsp	Dataset Parameter Table (DSP)
ndsp	Negative DSP offset relative to base address of DSPs
dn	Dataset name (ASCII, left-justified, blank-filled)

IFDNT determines if a dataset has been accessed.

Call from FORTRAN:

```
stat=IFDNT(dn)
```

stat	.TRUE. if dataset was accessed or opened; otherwise, .FALSE.
dn	Dataset name (ASCII, left-justified, zero-filled)

NOTE

IFDNT must be declared LOGICAL in the calling program.

NUMBLKS returns current size of dataset in 512-word blocks.

Call from FORTRAN:

```
val=NUMBLKS(dn)
```

val	Number of blocks returned as integer value. If number of blocks cannot be determined, a negative function value is returned.
dn	Dataset name or unit number

DATASET INPUT/OUTPUT SUBPROGRAMS

Before the 1.09 release, these subprograms were used for input/output control. Although they are still usable, use of the FORTRAN unblocked I/O subprograms (see section 5) is recommended.

OPEN opens a random, unblocked dataset.

Call from FORTRAN:

```
CALL OPEN(dst,dn,ds,dstat)
```

dst	Dataset Parameter Table (DSP), Open Dataset Name Table (ODN), and Dataset Definition List (DDL) array (dataset tables normally residing in the high-address end of the user field)
dn	Dataset name or unit number
ds	Dataset size
dstat	Dataset status

On exit, dataset size and status returned to locations specified in entry. See RDIN note below for description of status.

CLOSE terminates processing of a random, unblocked dataset.

Call from FORTRAN:

```
CALL CLOSE(dst)
```

dst	Dataset Parameter Table (DSP), Open Dataset Name Table (ODN), and Dataset Definition List (DDL) array (dataset tables normally residing in the high-address end of the user field)

On exit, no arguments returned

RDIN reads one buffer of data from a random, unblocked dataset.

Call from FORTRAN:

```
CALL RDIN(dst,abuf,sbuf,bnum,stat)
```

dst	Dataset Parameter Table (DSP), Open Dataset Name Table (ODN), and Dataset Definition List (DDL) array (dataset tables normally residing in the high-address end of the user field)
abuf	Buffer area
sbuf	Buffer size
bnum	Block number
stat	Return status is stored

On exit, dataset status returned in locations specified in entry

NOTE

Status consists of Dataset Parameter Table (DSP) error flags, right-justified. The return status is 0 if no error occurred.

TABLES

PDD is the table of permanent dataset definitions.

PDD is created and managed by some of the dataset management subprograms, so the user generally has no need to be concerned with it. For a detailed description of the table, see the CRAY-OS Version 1 Reference Manual, publication SR-0011.

SPECIAL PURPOSE SUBPROGRAMS 7

INTRODUCTION

Special purpose subprograms are grouped into the following categories:

- Debug aid routines
- Table management routines
- Stack management routines
- Heap management routines
- Job control routines
- Floating-point interrupt routines
- Bidirectional memory transfer routines
- Time and date routines
- Timestamp routines
- Control statement processing routines
- Job control language symbol routines
- SKOL run-time support routines
- Error processing routines
- Byte and bit manipulation routines
- Miscellaneous special-purpose routines

DEBUG AID ROUTINES

Debug aid routines consist of

- Flow trace routines
- Traceback routines

- Dump routines
- Exchange Package processing routines
- Array bounds checking routines

FLOW TRACE ROUTINES

Flow trace routines process the CFT flow trace option (ON=F). (See the FORTRAN (CFT) Reference Manual, CRI publication SR-0009, for details on flow tracing.) Calls to these routines are automatically inserted into code by the CFT compiler. A CAL call to a flow trace routine must be preceded by an ENTER macro or its equivalent. Flow trace routines are called by address.

FLOWENTR processes entry to a subroutine.

Call from CAL and FORTRAN:

```
CALL FLOWENTR
```

FLOWEXIT processes RETURN execution.

Call from CAL and FORTRAN:

```
CALL FLOWEXIT
```

FLOWSTOP processes a STOP statement.

Call from CAL and FORTRAN:

```
CALL FLOWSTOP
```

GETNAMEQ returns the ASCII, left-justified, space-filled name of the routine that called FLOWENTR or FLOWEXIT.

Call from FORTRAN:

```
CALL GETNAMEQ(arg)
```

Entry:
arg Address of output
Exit:
Caller name stored in address pointed to by *arg*

GETREGS returns register usage statistics for FLOWENTR.

Call from FORTRAN:

```
CALL GETREGS(arg)
```

Entry:
arg Address of output array
Exit:
Statistics stored in array

SETPLIMQ initiates detailed tracing of every call and return.

Call from FORTRAN:

```
CALL SETPLIMQ(lines)
```

lines Number of trace lines printed

ARGPLIMQ initiates listing of argument values for every call and return. This subprogram can be called only once in the user program.

Call from FORTRAN:

```
CALL ARGPLIMQ(list)
```

list List of argument values for every call and return

FLOWLIM sets a limit on the number of subroutine calls that can be traced. A summary is printed when the limit is reached.

Call from FORTRAN:

```
CALL FLOWIM(limit)
```

limit	Limit of the number of subroutine calls that can be traced

TRACEBACK ROUTINES

$TRBK and TRBK print a list of all subroutines active in the current calling sequence from the currently active subprogram. It also identifies the address of the reference. The user can specify a unit to receive the list. If no unit is specified, the list is printed to the user logfile.

Call from FORTRAN:

```
CALL TRBK[(arg)]
```

arg	Address of dataset name or unit number

TRBKLVL aids the traceback mechanism by returning information for the current level of the calling sequence.

Call from CAL:

```
CALLV TRBKLVL%
```

Entry:	
(A2)	Traceback table address of the current level (the B and T register save area)
(A3)	Argument list address of the current level's caller
Exit:	
(A2)	Traceback table address of the current level's caller; 0 if current level is a main level routine.
(A3)	Argument list address of the current level's caller; 0 if current level is a main level routine.

(S1)	Status: <0 if error =0 if no error >0 if no error and current level is main level
(S2)	Name of current level (ASCII, left-justified, blank-filled)
(S3)	Parcel address from where call to current level was made
(S4)	Parcel address of current level's entry point
(S5)	Line sequence number corresponding to call address; 0 if none.
(S6)	Number of arguments and registers passed to current level.

Call from FORTRAN:

```
CALL TRBKLVL(trbktab,arglist,status,name,calladr,entpnt,seqnum,numarg)
```

trbktab	Current level's traceback table address. On exit, current level's caller's traceback table address. Zero if current level is a main level routine.
arglist	Current level's argument list address. On exit, current level's caller's argument list address. Zero if current level is a main level routine.
status	<0 if error =0 if no error >0 if no error and current level is main level
name	Current level's name (ASCII, left-justified, blank-filled)
calladr	Parcel address from where call to current level was made
entpnt	Parcel address of current level's entry point
seqnum	Line sequence number corresponding to call address (a zero indicates none)
numarg	Number of arguments or registers passed to current level

DUMP ROUTINES

Dump routines produce a memory image and are called by address.

$PDUMP and PDUMP dump memory to $OUT and return control to calling program.

Call from FORTRAN:

```
CALL PDUMP(fwd,lwd,type)
```

fwd	First word to be dumped
lwd	Last word to be dumped
type	Dump type code: 0 or 3 Octal dump 1 Floating-point dump 2 Integer dump

$DUMP and DUMP dump memory to $OUT and abort the job.

Call from FORTRAN:

```
CALL DUMP(fwd,lwd,type)
```

fwd	First word to be dumped
lwd	Last word to be dumped
type	Dump type code: 0 or 3 Octal dump 1 Floating-point dump 2 Integer dump

NOTE

- If 4 is added to the dump type code, first word and last word addresses specified above are then addresses of addresses (indirect addressing).
- First word/last word/dump type address sets can be repeated up to 19 times.

DUMPJOB creates an unblocked dataset containing the user job area image (including register states). This data is suitable for input to the DUMP or DEBUG programs.

Call from FORTRAN:

```
CALL DUMPJOB(dn)
```

dn	FORTRAN unit number or Hollerith unit name. If no parameter is supplied, $DUMP is used by default.

SNAP copies current register contents to $OUT.

Call from FORTRAN:

```
CALL SNAP(regs,control,form)
```

regs	Code indicating registers to be copied: 1 B registers 2 T registers 3 B and T registers 4 V registers 5 B and V registers 6 T and V registers 7 B, T, and V registers
control	Control word (not currently used)
form	Code indicating format of dump. Dumps from registers S, T, and V are controlled by the following type codes: 0 Octal 1 Floating-point 2 Decimal 3 Hexadecimal Dumps from registers A and B are in octal format.

SYMDEBUG and DEADBUG produce a symbolic dump.

Call from FORTRAN:

```
CALL SYMDEBUG(char)
CALL DEADBUG
```

char	Character string (integer)

NOTE

The character string consists of the keyword/parameter pairs listed with the DEBUG utility in the CRAY-OS Version 1 Reference Manual, publication SR-0011. The string must terminate with a period.

CRAYDUMP prints a memory dump to a specified dataset.

Call from FORTRAN:

```
CALL CRAYDUMP(fwa,lwa,dn)
```

fwa	First word to be dumped
lwa	Last word to be dumped
dn	Name or unit number of the dataset to receive the dump output

EXCHANGE PACKAGE PROCESSING ROUTINES

Exchange processing switches execution from one program to another. An Exchange Package is a 16-word block of memory associated with a particular program. The Exchange Package processing routines include XPFMT, $FXP, FXP, and B20CT.

XPFMT produces a printable image of an Exchange Package in a user-supplied buffer. A and S registers appear in the buffer in both octal and character form; in the character form, the contents of the register are copied unchanged to the printable buffer. The calling program is responsible for proper translation of non-printable characters. Parcel addresses have a lowercase a, b, c, or d suffixed to the memory address.

The user can specify that the Exchange Package be formatted as a CRAY-1 or CRAY X-MP Exchange Package, or can allow XPFMT to determine which format to use based on the values in the Exchange Package. Values within the Exchange Package determine the Exchange Package format. XPFMT assumes the Exchange Package was produced by or for a CRAY X-MP computer if either the data base address or the data limit address is nonzero. Otherwise, it assumes the Exchange Package was produced by or for a CRAY-1 computer.

Call from FORTRAN:

```
CALL XPFMT(address,in,out,mode)
```

address	The nominal location of the Exchange Package to be printed as the starting Exchange Package address. This is not the address of the 16-word buffer containing the Exchange Package to be formatted.
in	A 16-word integer array containing the binary representation of the Exchange Package
out	An integer array, dimensioned (8,0:23) into which the character representation of the Exchange Package is stored. Line 0 is a ruler for debugging and is not usually printed (see figure 7-1). The first word of each line is an address and need not always be printed.
mode	An integer word indicating the mode in which the Exchange Package is to be printed. 'S'L forces the Exchange Package to be formatted as a CRAY-1 Exchange Package; 'X'L forces the Exchange Package to be formatted as a CRAY X-MP Exchange Package; 0 means that the subprogram is to use the Exchange Package value to deduce the machine type. Figure 7-1 is an example of a printout when the *mode* selected is 'X'L.

FXP and $FXP format and write to the output dataset the contents of the Exchange Package, the contents of the vector mask (VM), and the contents of the B0 register. These routines complement the user reprieve processing code by formatting the supplied Exchange Package to an output dataset.

Call from CAL:

```
CALLV $FXP
```

Entry:

(A1)	Address of output Dataset Parameter Area (DSP)
(A7)	Address of Exchange Package
(S2)	Vector mask (VM) to be formatted
(A3)	Contents of B00 to be formatted

Exit:

(A1)	Output DSP address
(A7)	Exchange Package address

Call from FORTRAN:

CALL FXP(*dsp*,*xp*,*vm*,*ret*)

dsp	Output Dataset Parameter Area (DSP) address
xp	Exchange Package address
vm	Vector mask (VM) to be formatted
ret	Contents of B0 to be formatted

B2OCT places the ASCII representation of the low-order *n* bits of a full Cray word into a specified character area.

Call from FORTRAN:

CALL B2OCT(*s*,*j*,*k*,*v*,*n*)

s	First word of an array where the ASCII representation is to be placed
j	Byte offset within array *s* where the first character of the octal representation is to be placed. A value of 1 indicates the destination begins with the first (leftmost) byte of the first word of *s*. *j* must be greater than 0.
k	Number of characters used in the ASCII representation. *k* must be greater than 0. *k* is the size of the total area to be filled and it is blank-filled if necessary.
v	Value to be converted. The low-order *n* bits of word *v* are used to form the ASCII representation. *v* must be less than or equal to $2^{63}-1$.

n Number of low-order bits of v to convert to ASCII character representation ($1 \le n \le 64$). If insufficient character space is available ($3k<n$), the character region is filled with asterisks (*).

The k characters pointed to by j in array s are first set to blanks. The low-order n bits of v are then converted to octal ASCII, using leading zeros if necessary. The converted value ($n/3$ characters, rounded up) is right-justified into the blanked-out destination character region.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....
oooooooo P   ooooooooa        A0 oooooooo aaa  MODES    FLAGS
oooooooo IBA oooooooo         A1 oooooooo aaa OFF  ON  OFF  ON
oooooooo ILA oooooooo         A2 oooooooo aaa MM   MM  PCI  PCI
oooooooo XA       oooo VL ooo A3 oooooooo aaa ICM ICM  MCU  MCU
oooooooo DBA oooooooo         A4 oooooooo aaa IFP IFP  FPE  FPE
oooooooo DLA oooooooo         A5 oooooooo aaa IUM IUM  ORE  ORE
oooooooo                      A6 oooooooo aaa IMM IMM  PRE  PRE
oooooooo                      A7 oooooooo aaa SEI SEI  ME   ME
                                              BDM BDM  IOI  IOI
oooooooo S0 oooooooooooooooooooooo aaaaaaaa FPS FPS  EEX  EEX
oooooooo S1 oooooooooooooooooooooo aaaaaaaa WS  WS   NEX  NEX
oooooooo S2 oooooooooooooooooooooo aaaaaaaa IOR IOR  DL   DL
oooooooo S3 oooooooooooooooooooooo aaaaaaaa          ICP  ICP
oooooooo S4 oooooooooooooooooooooo aaaaaaaa
oooooooo S5 oooooooooooooooooooooo aaaaaaaa
oooooooo S6 oooooooooooooooooooooo aaaaaaaa
oooooooo S7 oooooooooooooooooooooo aaaaaaaa
```

```
          PROCESSOR  = o    CLUSTER = o PS = o
          ERROR TYPE = aaaaaaaaaaaaaaaa VU = o
          CHIP SLCT  = oooooo BANK = oo
          READ MODE  = aaaaaa SYNDROME = ooo
```

---

a means alpha
o means octal

① Derived from *address* parameter

② Character representation of A registers

③ Character representation of S registers

④ A mode or flag mnemonic appears in the ON or OFF column depending on the state of the mode or flag bit in the Exchange Package.

Figure 7-1. Exchange Package printout

ARRAY BOUNDS CHECKING ROUTINES

$STOREF and $LOADF are generated by the compiler to perform run-time array bound checking. See the FORTRAN (CFT) Reference Manual, CRI publication SR-0009, for details on array bound checking.

TABLE MANAGEMENT ROUTINES

Table management routines perform the following functions.

FORTRAN	CAL	Function
TMINIT	$INT	Initializes managed tables
TMATS	$ATS	Allocates space to a table
TMADW	$ADW	Adds a word to a table
TMSRC	$MSC, $MSCO, $SRC	Searches a table with or without masking
TMPTS	$PTS%, $ZTS%	Presets memory with any given value (default is 0)
TMMVE	$MVE	Moves memory
TMMEM	$MEM	Requests additional memory
TMAMU	$AMV	Returns table management operation statistics

The routines in the FORTRAN column are FORTRAN callable and those in the CAL column are CAL callable. The Job Communication Block (JCB) field JCHLM defines the beginning address of the table area.

The user must provide two control information tables with corresponding CAL ENTRY pseudo-ops:

- Table Base Table (BTAB)
- Table Length Table (LTAB)

Their formats follow.

BTAB 2/-,14/ISP$_i$,24/AL$_i$,24/BASE$_i$

ISP$_i$	Normal interspace between table$_i$ and table$_{i+1}$,
AL$_i$	Current allocated table length, and
BASE$_i$	First word address of table$_i$.

```
LTAB    40/-,24/LEN_i

        LEN_i          Current length of table_i
```

The last entry in each control information table (TEND) must be a dummy entry. The TEND entry has zeros in the AL and LEN fields. The ISP field in the TEND entry contains the minimum field length increment to be made for table space. If the Table Manager needs to expand the job's field length, it does so by a minimum of ISP words. ISP is ignored for $MEM calls.

The number of entries in a control information table must not exceed 64, TEND included. (The FORTRAN callable versions of these routines use a default BTAB and LTAB definition from a common area in $SYSLIB.)

TMINIT initializes the table descriptor vector, BTAB, and zeros all elements of the table length vector, LTAB. The user must preset each element of BTAB to contain the desired interspace value for the corresponding table; for instance, S1 in the example below determines the interspace value for table 1. Interspace values determine how many words are added to a table when more room is needed for that table or for any table with a lower number.

TMINIT accepts a single parameter, *n*, in the prototype statements below, which determines the number of tables that can exist for the life of the calling program.

After the call to TMINIT, BTAB should not be changed. The interspace values have been shifted 48 bits to the left, bits 16 through 39 contain the current size of each table, and the rightmost 24 bits contain the absolute address of each table's first word. LTAB is used only to pass new table lengths from the user to the Table Manager.

```
INTEGER, BTAB(n), LTAB(n)
DATA BTAB /s1,s2,s3...,sn/
.
.
.
CALL TMINIT
```

The FORTRAN programmer can use statements like the following to access each table. In this example, table i is accessed.

```
EQUIVALENCE (BTAB(i), PTRi)
INTEGER PTRi, TABLEi (0:0)
POINTER (PTRi, TABLEi)
.
.
.
TABLEi (subscript) = ...
```

TM COMMON BLOCK

The common block labeled TM is reserved for use by the Table Manager and must always contain 64 BTAB words and 64 LTAB words.

```
    COMMON /TM/ BTAB(64), LTAB(64)
```

Blank common can be used in the customary way, but the last entry in it should be for a 1-dimensional array declared to contain just one word. The name of this array is then used to access the tables, beginning immediately after the end of blank common.

```
    COMMON // TABLES(1)
```

The following statement function extracts the rightmost 24 bits from a BTAB word and changes that value from an absolute address to a relative address or offset within the table area. Thus the result of BASE(N) is an index into TABLES(1) pointing to the first word currently allocated to table N.

```
    BASE(N) = (BTAB(N) .AND. 77777777B) - LOC(TABLES(1))
```

```
      WRITE (6,101) TABN
 101  FORMAT ('0 Dump of table ',I2,/)

      OFFSET = 0
 102  CONTINUE
        DO 103 I = 1, 4
           INTABLE = OFFSET .LT. LTAB(TABN)
           IF (INTABLE) THEN
              OCTAL(I) = TABLES(1 + BASE(TABN) + OFFSET)
              ALPHA(I) = TABLES(1 + BASE(TABN) + OFFSET)
           ELSE
              OCTAL(I) = 0
              ALPHA(I) = '        '
           END IF
           OFFSET = OFFSET + 1
 103    CONTINUE
        WRITE (6,104) OFFSET-4, OCTAL, ALPHA
104     FORMAT (I6,2X,4(O22,1X),4A8)

      INTABLE = OFFSET .LT. LTAB(TABN)
      IF (INTABLE) GO TO 102

      WRITE (6,105)
 105  FORMAT (/)

      RETURN
      END
```

$INT initializes table pointers. Upon entry, the user must provide all table interspace values. The remaining BTAB and LTAB fields are set by $INT. LTAB array is zeroed. Memory to be used for the managed tables is zeroed.

Call from CAL:

```
CALLV $INT
```

TMINIT initializes managed tables. Upon entry, the BTAB array contains the desired table expansion increments. Upon exit, the BTAB array is initialized, the LTAB array is zeroed, and the memory to be used for the managed tables is zeroed.

$MEM requests memory.

Call from CAL:

```
CALLV $MEM
```

Entry:

(A1)	Length of memory field requested

Exit:

No arguments returned. Memory is extended by requested amount.

TMMEM requests memory. Upon exit, memory is extended by the requested amount. No value is returned.

Call from FORTRAN:

```
CALL TMMEM(mem)
```

mem	Length of memory requested

$ATS allocates table space.

Call from CAL:

```
CALLV $ATS
```

Entry:

(A1)	Table pointer
(S1)	Increment

Exit:

(A2)	Pointer to expanded portion of table
(A3)	Address of expanded portion of table

TMATS allocates table space.

Call from FORTRAN:

```
index=TMATS(number,incre)
```

index	Index of change
number	Table number
incre	Table increment

$ADW adds a word to a table.

Call from CAL:

```
CALLV $ADW
```

Entry:

(A1)	Table pointer
(S1)	Entry for table

Exit:

(A2)	Index of word
(A3)	Address of word

TMADW adds a word to a table.

Call from FORTRAN:

```
index=TMADW(number,entry)
```

index	Index of word

number	Table number
entry	Entry for table

$MSC searches table with mask to locate a specific field within an entry.

Call from CAL:

```
CALLV $MSC
```

Entry:

(S1)	Search word
(S2)	Field being searched for within entry
(A1)	Table number
(A2)	Number of words per table entry

Exit:

(A0)	Address of match, if found; 0 if match not found.
(A3)	Address of match, if found

TMMSC searches table with mask to locate a specific field within an entry.

Call from FORTRAN:

```
index=TMMSC(tabnum,mask,sword,nword)
```

index	Table index of match, if found; -1 if not found.
tabnum	Table number
mask	Field being searched for within entry
sword	Search word
nword	Number of words per entry group

$MSCO searches table with mask to locate a specific field within an entry and an offset.

Call from CAL:

```
CALLV $MSCO
```

Entry:

(S1)	Search word
(S2)	Field being searched for within entry
(A1)	Table number
(A2)	Number of words per table entry
(A4)	Word offset within entry to be searched

Exit:

(A0)	Address of match, if found; 0 if match not found.
(A3)	Address of match, if found

$SRC searches table for a specific value.

Call from CAL:

```
CALLV $SRC
```

Entry:

(S1)	Search word
(A1)	Table number
(A2)	Number of words per table entry

Exit:

(A0)	Address of match, if found; 0 if match not found.
(A2)	Table index of match, if found
(A3)	Address of match, if found
(A4)	Ordinal of entry, or next ordinal if no match. First entry ordinal is 0.

TMSRC searches table with optional mask to locate a specific field within an entry and an offset.

Call from FORTRAN:

```
index=TMSRC(tabnum,arg,nword,offset,mask)
```

index	Table index of match, if a match is found; -1 if not found.
tabnum	Table number to search
arg	Search argument or key
nword	Number of words per entry
offset	Offset into entry group
mask	Field being searched for within entry

TMVSC searches vector table for the search argument.

Call from FORTRAN:

```
index=TMVSC(tabnum,arg,nword)
```

index	Table index match, if found; -1 if not found.
tabnum	Table number
arg	Search argument
nword	Number of words per entry group

$AMU returns total allotted table space.

Call from CAL:

```
CALLV $AMU
```

```
Entry:
  No arguments required
Exit:
  (A2)    Allocated length of tables
  (A3)    Number of table entries
```

TMAMU reports TMGR statistics.

Call from FORTRAN:

```
CALL TMAMU(len,tabnum,tabmov,tabmar,nword)
```

len	Allocated length of table
tabnum	Number of tables used
tabmov	Number of table moves
tabmar	Maximum amount of memory used throughout Table Manager
nword	Number of words moved

$PTS% presets table space.

Call from CAL:

```
CALLV $PTS%
```

Entry:

(A1)	Base address of space
(A2)	Length to be preset or zeroed
(S1)	Preset

Exit:

(V0)	Preset vector

$ZTS% zeros table space.

Call from CAL:

```
CALLV $ZTS%
```

Entry:

(A1)	Base address of space
(A2)	Length to be preset or zeroed

Exit:

(V0)	Zero vector

TMPTS presets table space.

Call from FORTRAN:

```
CALL TMPTS(start,len,preset)
```

start	Starting address
len	Length to preset
preset	Preset value

$MVE moves memory words to table.

Call from CAL:

```
CALLV $MVE
```

Entry:

(A1)	Address from where words are to be moved
(A2)	Address where words are to be moved
(A3)	Number of words to be moved

TMMVE moves words.

Call from FORTRAN:

```
CALL TMMVE(from,to,count)
```

from	Address from where words are to be moved
to	Address where words are to be moved
count	Number of words to be moved

STACK MANAGEMENT ROUTINES

Stack management routines are called automatically by CFT compiler code or, if in stack mode, by the CAL EXIT and ENTER macros.

A stack consists of a stack header plus one or more discontiguous segments. A stack segment includes memory for use as a stack followed by control words. Segment control words provide an overflow area and linkage to a discontiguous segment. Each discontiguous portion of the available space is linked through a control word to the next block of available space. The control word also contains the size of a given block.

A stack header control word exists at the base of each stack.

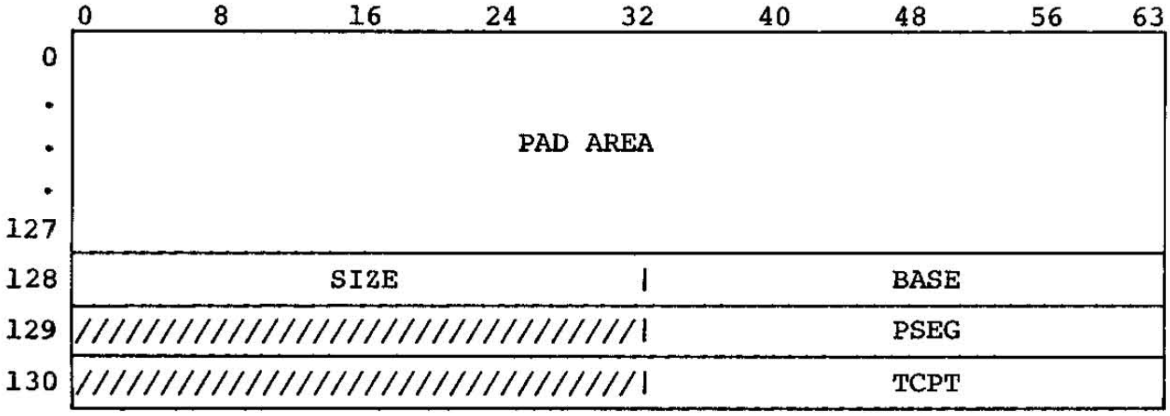
Format:

	0	8	16	24	32	40	48	56	63
0	GROW				ASEG				
1	HWM				SIZE				

Field	Word	Bits	Description
SHGROW	0	0-31	The number of times the stack has grown
SHASEG	0	31-63	Size of additional increments to the stack. Zero means overflow is a fatal error.
SHHWM	1	0-31	The largest size of the stack so far.
SHSIZE	1	32-63	Current size of the stack; includes usable stack space for all stack segments

Stack segment control words exist at the top of each stack segment.

Format:



Field	Word	Bits	Description
PAD AREA	0-127	0-63	Area indicating that B/T register save can always occur before a stack overflow check is made. In a segment that has overflowed to a discontiguous segment, this area contains the traceback packet used in returning from the underflow routine.
SSSIZE	128	0-31	Size of stack segment. Size includes the length of the stack header but does not include the length of stack segment control words.

Field	Word	Bits	Description
SSBASE	129	31-63	Offset to stack base relative to the absolute top of stack.
SSPSEG	130	32-63	Offset to previous top of stack. If PSEG=0, this is an initial stack segment.
SSTCPT	131	32-63	Pointer to task common address block ($TASKCOM).

Stack management routines perform the following functions.

Routine	Function
$STKOFEN	Manages stack overflow
$STKCR	Creates initial stack segment
$STKUFCK	Manages stack underflow
$STKUFEX	Releases a topmost stack segment
$STKDE%	Releases all stack segments

$STKOFEN is called if stack overflow is detected during allocation of contiguous static space onto a stack at subprogram entry. Information from the JCB, the stack header, and the B register stack pointers determines how much additional space, if any, to allocate to the stack.

The current stack segment is enlarged if possible. Otherwise, a discontiguous segment is created. In the latter case, the discontiguous segment is released from the stack on exit from the routine that caused the overflow condition.

This routine creates a stack for the root task the first time overflow occurs. In all cases, it updates the B register stack pointers.

Call from CAL:

```
CALLV $STKOFEN
```

No arguments required.

$STKCR creates an initial stack segment and the stack header.

Call from CAL:

```
CALLV $STKCR
```

Entry:
  (S1)   Initial size of stack
  (S2)   Size of increments to this stack; zero implies stack overflow is a fatal error.
Exit:
  (A1)   Address of first available word
          (First word address + length of header)
  (A2)   Size of stack (does not include the length of control words)

$STKUFCK determines whether an exit sequence has produced stack underflow. This routine releases the topmost segment if underflow has occurred.

Call From CAL:

```
CALLV $STKUFCK
```

No arguments required.

$STKUFEX releases the topmost stack segment. The call to this routine occurs on exit from the routines that caused the stack to overflow to a discontiguous segment.

Call from CAL:

```
CALLV $STKUFEX
```

No arguments required.

$STKDE% releases all segments of the indicated stack to the available space.

Call from CAL:

```
CALLV $STKDE%
```

No arguments required.

HEAP MANAGER ROUTINES

Heap manager routines provide dynamic storage allocation by managing a block of memory within a user's job area, the heap. Each job has its own heap. The functions of the heap manager routines are allocating a block of memory, returning a block of memory to the heap's list of available space, and changing the length of a block of memory. Heap manager routines also move a heap block to a new location if there is no room to extend it, return part of the heap to the operating system, check the integrity of the heap, and report information about the heap. See the CRAY-OS Version 1 Reference Manual, publication SR-0011, for the location of the heap and parameters on the LDR control statement that affect the heap.

The heap manager consists of the following routines.

FORTRAN	CAL	Function
HPALLOC	ALLOC%	Allocate a block of memory from the heap
HPDEALLC	DEALLC%	Return a block of memory to the heap
HPNEWLEN	NEWLEN%	Change the length of a heap block
HPCLMOVE	CLMOVE%	Change the length of a heap block and move the block if it cannot be extended in place
IHPLEN	HPLEN%	Return the length of a heap block
HPSHRINK	SHRINK%	Return memory from the heap to the operating system
HPCHECK	HCHECK%	Check the integrity of the heap
IHPSTAT	HPSTAT%	Return information about the heap
HPDUMP	HPDUMP%	Write information about the heap to a dataset

The heap manager routines keep various statistics on the use of the heap. These include values used to tune heap parameters specified on the LDR control statement and information used in debugging.

ALLOCATE ROUTINES

Allocate routines search the linked list of available space for a block greater than or equal to the size requested.

The length of an allocated block can be greater than the requested length because blocks smaller than the managed memory *epsilon* specified on the LDR control statement are never left on the free space list.

Error conditions checked in allocate routines:

- Length is not an integer greater than zero (-1)
- No more memory is available from the system (-2) (checked if the request cannot be satisfied from the available blocks on the heap)

Call from CAL:

```
CALLV ALLOC%
```

Entry:

(S1)	Size (number of words) of requested block
(S2)	Abort flag: nonzero requests abort on error; zero requests an error code.

Exit:

(S1)	First word address of allocated block
(S2)	Number of words requested (unchanged)
(S3)	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error.

Call from FORTRAN:

```
CALL HPALLOC (addr,length,errcode,abort)
```

addr	First word address of the allocated block (output)
length	Number of words of memory requested (input)
errcode	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error (output).
abort	Abort code: nonzero requests abort on error; zero requests an error code (input)

DEALLOCATE ROUTINES

Deallocate routines return a block to the list of available space.

Error conditions checked in deallocate routines:

- Address outside bounds of the heap (-3)
- Block already free (-4)
- Address not at beginning of block (-5)
- Control word for next block overwritten (-7)

Call from CAL:

```
CALLV DEALLC%
```

Entry:

(S1)	First word address of block being deallocated
(S2)	Nonzero requests abort on error; zero requests an error code.

Exit:

(S1)	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error.

Call from FORTRAN:

```
CALL HPDEALLC (addr,errcode,abort)
```

addr	First word address of the block to deallocate (input)
errcode	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error (output).
abort	Abort code: nonzero requests abort on error; zero requests an error code (input).

SET NEW LENGTH ROUTINES

Set new length routines change the size of an allocated block. If the new length is less than the allocated length, the portion starting at ADDR+LENGTH is returned to the heap. If the new length is greater than the allocated length, the block is extended if it is followed by a free block. A status is returned, telling whether the change was successful.

The new length of the block can be greater than the requested length because blocks smaller than the managed memory *epsilon* specified on the LDR control statement are never left on the free space list.

Error conditions checked in set new length routines:

- Length not an integer greater than zero (-1)
- Addresss outside the bounds of the heap (-3)
- Block free (-4)
- Address not at beginning of block (-5)
- Control word for next block has been overwritten (-7)

Call from CAL:

```
CALLV NEWLEN%
```

Entry:

(S1)	Address of block to change
(S2)	Requested new total length of block
(S3)	Nonzero requests abort on error; zero requests an error code.

Exit:

(S1)	Address of block (unchanged)
(S2)	Requested new total length (unchanged)
(S3)	Status: zero if the change in length was successful; one if the block could not be extended in place; a negative integer for the type of error detected.

Call from FORTRAN:

CALL HPNEWLEN (*addr*,*length*,*status*,*abort*)

addr	First word address of the block to change (input)
length	Requested new total length of the block (input)
status	Status: zero if the change in length was successful; one if the block could not be extended in place; a negative integer for the type of error detected (output).
abort	Abort code: nonzero requests abort on error; zero requests an error code (input).

CHANGE LENGTH AND MOVE ROUTINES

Change length and move routines extend a block if it is followed by a large enough free block or copy the contents of the existing block to a larger block and return a status code that the block has been moved. They can also reduce the size of a block if the new length is less than the old length. In this case, they have the same effect as the length change routines.

The new length of the block can be greater than the requested length because blocks smaller than the managed memory *epsilon* specified on the LDR control statement are never left on the free space list.

Error conditions checked in change length and move routines:

- Length not an integer greater than zero (-1)
- No more memory available from the system (-2) (checked if the block cannot be extended and the free space list does not include a large enough block)
- Address outside the bounds of the heap (-3)
- Block free (-4)
- Address not at beginning of block (-5)
- Control word for next block has been overwritten (-7)

Call from CAL:

```
CALLV CLMOVE%
```

Entry:

(S1)	Address of block to change
(S2)	Requested new total length
(S3)	Abort code: nonzero requests abort on error; zero requests an error code.

Exit:

(S1)	Address of changed block; this value can be different from the entry value.
(S2)	Requested new total length (unchanged)
(S3)	Status: zero if the block was extended in place; one if it was moved; a negative integer for the type of error detected.

Call from FORTRAN:

```
CALL HPCLMOVE (addr,length,status,abort)
```

addr	On entry, first word address of the block to change; on exit, the new address of the block if it was moved
length	Requested new total length (input)
status	Status: zero if the block was extended in place; one if it was moved; a negative integer for the type of error detected (output).
abort	Abort code: nonzero requests abort on error; zero requests an error code (input).

HEAP BLOCK LENGTH ROUTINES

Heap block length routines return the length of a heap block. The length of the block can be greater than the amount requested because of the managed memory *epsilon*.

Error conditions checked in heap block length routines:

- Address outside the bounds of the heap (-3)
- Block free (-4)
- Address not at beginning of block (-5)
- Control word for next block has been overwritten (-7)

Call from CAL:

```
CALLV HPLEN%
```

Entry:

(S1)	First word address of block
(S2)	Abort code: nonzero requests abort on error; zero requests an error code

Exit:

(S1)	Number of words in the block
(S2)	First word address of block (copied from S1)
(S3)	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error.

Call from FORTRAN:

```
length=IHPLEN (addr,errcode,abort)
```

length	Length of the block starting at *addr* (output)
addr	First-word-address of the block (input)
errcode	Error code: zero if no error was detected; otherwise, a negative integer code for the type of error (output).
abort	Abort code: nonzero requests abort on error; zero requests an error code (input).

HEAP SHRINK ROUTINES

Heap shrink routines return an unused portion of the heap to the operating system. This is done only if the blocks closest to HLM are free; no allocated blocks are moved. The minimum amount of memory to be returned is the managed memory increment specified on the LDR control statement. These routines are called only from the user program.

Call from CAL:

```
CALLV SHRINK%
```

No arguments

Call from FORTRAN:

```
CALL HPSHRINK
```

No arguments

HEAP INTEGRITY CHECK ROUTINES

Heap integrity check routines check the integrity of the heap. Each control word is examined to ensure that it has not been overwritten. Error conditions checked in heap integrity check routines:

- Bad control word for allocated block (-5)
- Bad control word for free block (-6)

Call from CAL:

```
CALLV HCHECK%
```

Exit:

(S1) Error code: zero if no error was detected; otherwise, a negative integer code for the type of error.

Call from FORTRAN:

```
CALL HPCHECK (errcode)
```

errcode Error code: zero if no error was detected; otherwise, a negative integer code for the type of error (output).

HEAP STATISTICS ROUTINES

Heap statistics routines return statistics about the heap.

Call from CAL:

```
CALLV HPSTAT%
```

Entry:

(S1) Code for the information requested:

1 Current heap length
2 Largest size of the heap so far
3 Smallest size of the heap so far
4 Number of allocated blocks
5 Number of times heap has grown
6 Number of times heap has shrunk
7 Last routine that changed the heap
8 Caller of last routine that changed the heap
9 First word address of heap area changed last
10 Size of the largest free block
11 Amount by which the heap can shrink
12 Amount by which the heap can grow
13 First word address of the heap
14 Last word address of the heap

Exit:

(S1) Requested value

Call from FORTRAN:

```
value=IHPSTAT (code)
```

code	Code for the type of information requested (see CAL entry point)
value	Requested information

DUMP HEAP CONTROL WORD ROUTINES

Dump heap control word routines dump the address and size of each block in the heap. Three types of dump are available: a dump of all heap blocks, a dump of free blocks that traces the links to the next block on the free list, and a dump of free blocks that traces the links to the previous block on the free list. The dump stops if an invalid value is found in a field needed to continue the dump.

Call from CAL:

```
CALLV HPDUMP%
```

Entry:

(S1) Code for the type of dump requested:
- 0 Print heap statistics
- 1 Dump all heap blocks in storage order
- 2 Dump free blocks; follow NEXT links.
- 3 Dump free blocks; follow PREV links.

(S2) Dataset name; name of the dataset to which the dump is to be written.

Call from FORTRAN:

```
CALL HPDUMP (code,dsname)
```

code	Code for the type of dump requested (see CAL entry point)
dsname	Name of the dataset to which the dump is to be written. *dsname* must be in left-justified, Hollerith form.

HEAP EXPANSION ROUTINE

The heap expansion subroutine is used by the allocate and new length routines when there is not enough space in the heap to meet a request. The subroutine requests additional memory from the operating system, adds the new memory to the free space list in the heap, and adjusts the control words at the end of the heap.

NOTE

The heap expansion routine should not be called directly by a user program.

Call from CAL:

```
CALLV HPGROW%
```

Entry:

(S1)	Number of words in pending allocate request

Exit:

(S1)	Success flag; one if more memory was added to the heap, zero if the heap could not be expanded.

HEAP MEMORY REQUEST ROUTINE

The heap memory request routine requests more memory from the operating system to be added to the heap.

NOTE

The heap memory request routine should not be called directly by a user program.

Call from CAL:

```
CALLV HPMEM%
```

Entry:

(S1)	Number of additional words needed for the heap

Exit:

(S1)	Number of additional words that can be added to the heap; zero if the heap could not be expanded.

HEAP MERGE ROUTINE

The heap merge routine is used by the heap shrink routine to coalesce free blocks before finding out how much the heap can shrink. It is also used by HPSTAT to determine how much the heap can shrink and the size of the largest free block in the heap.

NOTE

The heap merge routine should not be called directly by a user program.

Call from CAL:

```
CALLV HMERGE%
```

Exit:

(S1)	Last free block; zero if the last free block is allocated, or the address of the last heap block if it is free.
(S2)	The size of the largest free block

JOB CONTROL ROUTINES

Job control routines perform functions relating to job step termination, either causing a termination or instructing the system how to handle a termination. Unless otherwise specified, these routines are called by address. No arguments are returned.

ABORT requests abort with traceback and provides optional logfile message. The optional user-supplied logfile message is written to both user and system logfiles. The message is written in the same format in which it is sent.

Call from FORTRAN:

```
CALL ABORT[(log)]
```

log Logfile message

END$ and $END terminate the job step and advance the job to the next job step.

Call from FORTRAN:

```
END
```

$ENDRPV and ENDRPV continue normal exit processing after a reprievable request has been processed. This exit processing can be the result of normal termination or abort processing.

Call from CAL:

```
CALLV $ENDRPV
```

Call from FORTRAN:

```
CALL ENDRPV
```

ERREXIT requests abort.

Call from FORTRAN:

```
CALL ERREXIT
```

EXIT provides exit for FORTRAN programs, writes the following message to the logfile, and advances the job to the next step.

UT003 _ EXIT CALLED BY *routine name*

Call from FORTRAN:

```
CALL EXIT
```

NORERUN controls the monitoring of conditions causing the job to be flagged as not rerunnable.

Call from FORTRAN:

```
CALL NORERUN(param)
```

param	One argument is required. If argument is 0, the system monitors for conditions causing the job to be flagged as not rerunnable. If nonzero, such conditions are not monitored.

RERUN allows the user to declare the job rerunnable or not rerunnable.

Call from FORTRAN:

```
CALL RERUN(param)
```

param	One argument is required. If the argument is 0, the job can be rerun. If the argument is nonzero, the job cannot be rerun.

$SETRPV and SETRPV transfer control to the specified routine when a user-selected reprievable condition occurs. $SETRPV is called by value; SETRPV is called by address. See the Macros and Opdefs Reference Manual, CRI publication SR-0012, for details of the SETRPV parameter formats.

Call from CAL:

```
CALLV $SETRPV
```

Entry:

(A1)	Reprieve code entry address
(A2)	Reprieve table address
(S1)	Mask

Call from FORTRAN:

```
CALL SETRPV(rpvcode,rpvtab,mask)
```

rpvcode	Routine where control is transferred
rpvtab	A 40-word array reserved for system use
mask	User mask specifying reprievable conditions

$STOP terminates the job step, advances the job to its next job step, and prints an optional user-supplied message to the logfile. The message is written in the same format in which it is sent.

Call from CAL:

```
CALL $STOP[,(log)]
```

log	Address of the logfile message

$PAUSE suspends program execution. An installation parameter, I@PAUSE, determines whether $PAUSE or $STOP is to be executed. The default is program suspension. $PAUSE prints an optional user-supplied message to the logfile. The message is written in the same format in which it is sent.

Call from CAL:

```
CALL $PAUSE[,(log)]
```

log	Address of optional logfile message

FLOATING-POINT INTERRUPT ROUTINES

Floating-point interrupt routines allow the user to test, set, and/or clear the Floating-point Interrupt Mode flag. Subroutine linkage is call-by-address.

FLOATING-POINT INTERRUPT TEST

The floating-point interrupt test routine determines whether interrupts are permitted or prohibited.

SENSEFI determines the current interrupt mode.

Call from FORTRAN:

```
CALL SENSEFI(mode)
```

mode Interrupt mode:
If *mode*=1, permit interrupts.
If *mode*=0, prohibit interrupts.

TEMPORARY FLOATING-POINT INTERRUPT CONTROL

These routines are local to the current job step. The system restores the most recent mode setting at the start of the next job step. No arguments are required or returned.

CLEARFI temporarily prohibits floating-point interrupts.

Call from FORTRAN:

```
CALL CLEARFI
```

SETFI temporarily permits floating-point interrupts.

Call from FORTRAN:

```
CALL SETFI
```

JOB FLOATING-POINT INTERRUPT CONTROL

The results of routines are propagated through job steps. The system does not alter the mode setting unless another floating-point interrupt control subroutine is called or a MODE control statement is executed. No arguments are required or returned.

CLEARFIS prohibits floating-point interrupts for a job until they are enabled or the job terminates.

Call from FORTRAN:

```
CALL CLEARFIS
```

SETFIS enables floating-point interrupts until they are explicitly disabled or the job terminates.

Call from FORTRAN:

```
CALL SETFIS
```

BIDIRECTIONAL MEMORY TRANSFER ROUTINES

Bidirectional memory transfer routines test, set, and/or clear the bidirectional Memory Transfer Mode flag. Subroutine linkage is call-by-address.

NOTE

These routines are only effective on the CRAY X-MP, which has hardware support for bidirectional memory transfer.

BIDIRECTIONAL MEMORY TRANSFER TEST

The bidirectional memory transfer test routine determines whether bidirectional memory transfer is enabled or disabled.

SENSEBT determines the current memory transfer mode.

Call from FORTRAN:

```
CALL SENSEBT(mode)
```

mode Transfer mode:
  If *mode*=1, bidirectional memory transfer enabled
  If *mode*=0, bidirectional memory transfer disabled

TEMPORARY BIDIRECTIONAL MEMORY TRANSFER CONTROL

These routines are local to the current job step. The system restores the most recent mode setting at the start of the next job step. No arguments are required or returned.

CLEARBT temporarily disables bidirectional memory transfers.

Call from FORTRAN:

```
CALL CLEARBT
```

SETBT temporarily enables bidirectional memory transfers.

Call from FORTRAN:

```
CALL SETBT
```

PERMANENT BIDIRECTIONAL MEMORY TRANSFER CONTROL

The results of these routines are permanent and are propagated through job steps. The system does not alter the mode setting unless another bidirectional memory transfer control subroutine is called or a MODE control statement is executed. No arguments are required or returned.

CLEARBTS permanently disables bidirectional memory transfers.

Call from FORTRAN:

```
CALL CLEARBTS
```

SETBTS permanently enables bidirectional memory transfers.

Call from FORTRAN:

```
CALL SETBTS
```

TIME AND DATE ROUTINES

Time and date routines produce the time and/or date in specified forms. These routines can be called as FORTRAN functions or routines. All of the routines are called by address.

CLOCK returns current system clock time in ASCII *hh:mm:ss* format.

Call from FORTRAN:

```
time=CLOCK()
CALL CLOCK(time)
```

time	Time in *hh:mm:ss* format (type integer)

DATE returns today's date in *mm/dd/yy* format.

Call from FORTRAN:

```
date=DATE()
CALL DATE(date)
```

date	Today's date in *mm/dd/yy* format (type integer)

JDATE returns today's Julian (ordinal) date in *yyddd* format, left-justified, blank-filled.

Call from FORTRAN:

```
date=JDATE()
CALL JDATE(date)
```

date	Today's Julian date in *yyddd* format

SECOND returns CPU time since start of job in floating-point seconds.

Call from FORTRAN:

```
second=SECOND([result])
CALL SECOND(second)
```

second	Result (CPU time used by job since start of job in floating-point seconds). Contents of S1 stored at address of argument.
result	Same as above (optional for function call)

TIMEF returns a value, in floating-point milliseconds, that is, the amount of wall-clock time passed since the initial call to TIMEF in the program.

Call from FORTRAN:

```
timef=TIMEF([result])
CALL TIMEF(timef)
```

timef	Wall-clock time passed since the initial call to TIMEF, in floating-point milliseconds. The initial call to TIMEF returns 0.
result	Same as *timef*

TREMAIN returns CPU time remaining for job execution in floating-point seconds.

Call from FORTRAN:

```
CALL TREMAIN(result)
```

result	Calculated CPU time remaining; stored in result.

TIMESTAMP ROUTINES

These routines are used by system accounting programs to convert between various representations of time.

TSDT converts between timestamps and the date and time as ASCII strings.

Call from FORTRAN:

```
CALL TSDT(ts,date,hhmmss,ssss)
```

ts	Timestamp on entry (type integer)
date	Word to receive ASCII date *mm/dd/yy*
hhmmss	Word to receive ASCII time *hh:mm:ss*
ssss	Word to receive ASCII fractional seconds, *ssss*

DTTS converts from date and time to timestamp.

Call from FORTRAN:

```
ts=DTTS(date,time)
```

ts	Timestamp corresponding to *date* and *time* (type integer)
date	ASCII date on entry in the form of *mm/dd/yy*
time	ASCII time on entry in the form of *hh:mm:ss*

On return, if *ts*=0, an incorrect parameter was passed to DTTS.

TSMT converts from a timestamp to the corresponding real-time clock value.

Call from FORTRAN:

```
irtc=TSMT(ts)
```

irtc	Real-time clock value corresponding to specified timestamp
ts	Timestamp to be converted (type integer)

MTTS converts from a real-time clock value to the corresponding timestamp value.

Call from FORTRAN:

```
ts=MTTS(irtc)
```

ts	Timestamp corresponding to real-time clock value (type integer)
irtc	Real-time clock value

UNITTS returns the number of timestamp units in a specified number of standard time units. UNITTS must be declared type integer.

Call from FORTRAN:

```
ts=UNITTS(periods,units)
```

ts	Number of timestamp units in *periods* and *units* (type integer)
periods	Number of timestamp units wanted in standard time units (that is, number of seconds, minutes, etc.); type integer.
units	Specification for the units in which *periods* is expressed. The following values are accepted: 'DAYS'H, 'HOURS'H, 'MINUTES'H, 'SECONDS'H, 'MSEC'H (milliseconds), 'USEC'H (microseconds), 'USEC100'H (100s of microseconds). Left-justified, blank-filled, Hollerith.

Example:

```
ts=UNITTS(2,'DAYS'H)
```

ts Number of timestamp units in two days

CONTROL STATEMENT PROCESSING ROUTINES

Control statement processing routines place control statement elements in appropriate memory locations to perform the specified operations. These routines, CRACK, PPL, and CEXPR, also can process directives obtained from some source other then the control statement file ($CS).

CONTROL STATEMENT CRACKING ROUTINES

Control statement cracking routines take the uncracked image from the JCCCI field and crack it into the JCCPR field. The Job Communication Block (JCB) contains the control image in JCCCI. JCDLIT is a flag indicating whether or not literal delimiters are to be retained in the string.

$CS, $CCS, and CCS are different entry points in the same routine.

$CS does not abort the job if errors are encountered.

Call from CAL:

```
CALLV $CS
```

```
Exit:
  (S0)      =0  No errors
            ≠0  Errors
```

$CCS,CCS aborts the job if errors are encountered.

Call from CAL:

```
CALLV $CCS
```

Call from FORTRAN:

```
CALL CCS
```

CRACKED PARAMETER LIST

Control statement parameters are available to the user in the form of a cracked parameter list starting at location W@JCCPR in the JCB.

Example:

The following control statement appears in the cracked parameter list as described below (shaded area is binary zero).

```
VERB,KEYWORD='THIS IS A LITERAL VALUE...'.
```

	0	8	16	24	32	40	48	56 63
W@JCCPR+0	V	E	R	B	////////	////////	////////	////////
1	////////	////////	////////	////////	////////	////////	////////	017
2	K	E	Y	W	O	R	D	////////
3	////////	////////	////////	////////	////////	////////	////////	007
4	T	H	I	S		I	S	
5	A		L	I	T	E	R	A
6	L		V	A	L	U	E	.
7	.	.	////////	////////	////////	////////	////////	037

An internal code for each control statement separator or terminator is positioned in the last byte of odd-numbered words following keyword or parameter values in the cracked parameter list.

Control statement scanning stops when a continuation separator or terminator is encountered. Thus, for continued control statements, the caller can call F$GNS again to scan the remainder of the control statement.

Example:

This is an example of a continued control statement.

```
ACQUIRE,DN=dn,TEXT='ABC' ∧
'DEF'.
```

The first crack control statement (CCS) call results in the following cracked parameter list:

	0	8	16	24	32	40	48	56 63
W@JCCPR+0	A	C	Q	U	I	R	E	////////
1	////////	////////	////////	////////	////////	////////	////////	017
2	D	N	////////	////////	////////	////////	////////	////////
3	////////	////////	////////	////////	////////	////////	////////	007
4	D	N	////////	////////	////////	////////	////////	////////
5	////////	////////	////////	////////	////////	////////	////////	017
6	T	E	X	T	////////	////////	////////	////////
7	////////	////////	////////	////////	////////	////////	////////	007
8	A	B	C	////////	////////	////////	////////	////////
9	////////	////////	////////	////////	////////	////////	////////	002

The calling program can then interpret this portion of the control statement. The second CCS call replaces the cracked parameter list with:

	0	8	16	24	32	40	48	56 63
W@JCCPR+0	D	E	F	////////	////////	////////	////////	////////
1	////////	////////	////////	////////	////////	////////	////////	037

The current control statement image remains in unmodified form at location W@JCCCI in the JCB.

GET PARAMETER ROUTINE

The get parameter routine processes control statement parameter values from an already cracked control statement. If the statement has been continued across card images, GETPARAM automatically requests the next control statement and calls $CCS to crack it. Processing is determined by the rules set up by the Parameter Control Table (PCT).

The PCT indicates default values for unspecified parameters. Through the PCT, the caller also indicates the following.

- If a parameter must be specified on the statement
- If a parameter is positional or keyword
- If a keyword parameter can have an equated value

- If a keyword parameter must have an equated value
- If any parameters are allowed

Call from FORTRAN:

```
CALL GETPARAM(table,number,param)
```

table The Parameter Control Table (PCT), dimensioned (5,*number*) and containing the following in each 5-element row.

1. A left-justified, zero-filled keyword
2. A default value for use if the keyword is missing
3. A keyed value for use if the keyword is unqualified
4. An initial subscript for use in output to *param*
5. A limiting subscript for use in output to *param*

If item 2 is negative, GETPARAM requires the keyword to be on the control statement.

If item 3 is negative, GETPARAM does not allow the use of the keyword alone (as in "...,keyword,...").

Either item 2 or 3 can be 0; GETPARAM does not distinguish between zeros and any other positive values such as character strings, but the caller can test them after GETPARAM returns.

If items 2 and 3 are 0 and 1, or 1 and 0 respectively, GETPARAM does not allow the keyword to be followed by an =. The keyword must be simply absent or present.

If item 1 is a 64-bit mask (that is, 177777 7777 7777 7777 7777B), then the value given as the keyword is returned in the control table. When an entry of this type has been specified in the control table, the number of parameters is limited to one.

If item 1 is given a value of 0, then the entry describes a positional parameter. Entries of this nature must be described in positional order.

If bit 2 in item 4 (that is, 020000 0000 0000 0000 0000B) is set, the parameters following the keyword are defined to be secure and are edited out before the statement is echoed to the user's logfile.

number	The number of parameters described in the control table. If given a value of 0, GETPARAM does not allow any parameters on the control statement.
param	An array sufficiently large to receive all the parameter values

Call from CAL:

```
CALLV $GPARAM
```

(A1)	*table* (Address)
(A2)	*number* (Actual number)
(A3)	*params* (Address)

NOTE

$GP is an alternate entry for CAL callers and provides a "no-abort" exit. Upon exit, S0=0 if no errors are detected.

Example of control table definition in FORTRAN:

```
      INTEGER PERMFILE(2) PARAMS(15), TABLE(5,4), INPUT, LIBRARY(10),
      LIST
      EQUIVALENCE(PARAMS(1),INPUT),
     *            (PARAMS(2),PERMFILE),
     *            (PARAMS(4),LIBRARY(1)),
     *            (PARAMS(14),LIST)
      DATA PARAMS/15*0/
      DATA (TABLE(I,1),I=1,5)/'I'L,'$IN'L,'$IN'L,1,1/,
     -     (TABLE(I,2),I=1,5)/'P'L,0,-1,2,3/,
     -     (TABLE(I,3),I=1,5)/'LIB'L,-1,'$FTLIB'L,4,13/,
     -     (TABLE(I,4),I=1,5)/'LIST'L,0,1,14,14/
      CALL GETPARAM (TABLE,4,PARAMS)
```

This table (for a hypothetical program) tells GETPARAM that the only keywords to be accepted are I, P, LIB, and LIST. The -1 value means that P cannot appear alone (without an equal sign) and that LIB (with or without an equal sign) must appear in the control statement.

In this table, only one word is provided for the I parameter; therefore, if I=*xxx* appears in the control statement, the option *xxx* must not exceed eight characters. The two words provided for the P parameter allow for the maximum of 16 characters or for two subparameters (up to eight characters each) separated by a colon in the control statement. Ten words are provided for the LIB parameter so that up to ten subparameters (or five 2-word parameters) are allowed in the control statement. GETPARAM requires the keyword LIST to appear alone or not at all. If LIST is specified, the value returned in the Parameter Value Table is 1. LIST cannot be followed by an equal sign.

NOTE

The following two subparameters cannot be distinguished from one another in the PARAMS table:

A=A1234567:B1234567 (Two 8-character parameters)

A=A1234567B1234567 (One 16-character parameter)

Thus, the caller is responsible for restricting such cases.

The output array PARAMS must be as large as the largest subscript. If PARAMS is initialized to zeros, the programmer can determine how many words are returned by GETPARAM for multiword parameters such as P and LIB.

Since FORTRAN array numbering starts with 1, the array's base address is reduced by 1 in GETPARAM. Therefore, the CAL user must supply the table address + 1[†] in order to use labels directly in lieu of the FORTRAN subscripts.

GETPARAM aborts if the control statement violates either the standard control statement syntax rules or the additional rules imposed by the Parameter Control Table. If there are no errors, the array is filled with values from the control statement and/or with default values. The Parameter Control Table is not altered by GETPARAM.

DIRECTIVE CRACKING ROUTINE

The directive cracking routine reformats (cracks) a user-supplied string into verb, separators, keywords, and values. The cracked directive is placed in a user-supplied buffer and returns the status of the crack to the caller. CRACK can be called repeatedly to process a control statement across several records.

[†] This is not true for $GP.

Call from FORTRAN:

```
CALL CRACK(ibuf,ilen,cbuf,clen,flag[,dflag])
```

ibuf	Image of the statement to be cracked
ilen	Integer length (in words) of the image of the statement to be cracked. Maximum value is 10 words.
cbuf	Array to receive the cracked image
clen	Integer length in words of the array *cbuf*

NOTE

Each keyword or positional parameter should be assigned a separate word. Keywords or positional parameters of more than eight characters must be assigned one word for each eight characters plus one for any remaining characters if length is not a multiple of eight characters. Each separator must also be assigned a separate word.

---

flag Integer variable to receive completion status. The Return Value flag has the following meanings.

0	Normal termination
1	No error; continuation character encountered.
2	Invalid character encountered
3	Premature end of input line
4	CRACK buffer overflow
5	Unbalanced parentheses
6	Input buffer too large

dflag Integer flag indicating that literal string delimiters are to be preserved in the cracked image. If set to 0 or omitted, quotes are not in the cracked string. If set to 1, all quotes are included in the string.

NOTE

flag should be set to 0 before the first call to CRACK and not changed (except by CRACK) until after the last call to CRACK.

PROCESS PARAMETER LIST ROUTINE

The process parameter list routine processes the keywords for a given directive. Processing is governed by the Parameter Description Table. This table has the same format as the table GETPARAM uses, except that the length of the table used by PPL is seven words with the extra two words unused.

Call from FORTRAN:

```
CALL PPL(cbuf,ctable,ltable,outarray,stattbl)
```

cbuf	Array to receive the cracked image
ctable	PPL control table
ltable	Number of 7-word entries in PPL control table
outarray	Array to receive parameter values
stattbl	3-word completion status array. On the first-time call, the Return Status Table is initialized to 0. If PPL returns a non-normal status, and PPL is called again with the non-normal values left in, it attempts recovery.

Array element	Meaning
1	Return status code: 0 Normal termination 1 Required keyword not found 2 Output keyword overflow 3 Syntax error 4 Unknown or duplicate keyword 5 Unexpected separator encountered 6 Keyword cannot be equated 7 Keyword must have value 8 Maximum of 64 keywords exceeded 9 Invalid return status; cannot recover.
2	Keyword in error
3	Ordinal keyword value

CRACK EXPRESSION ROUTINE

The crack expression routine transforms an expression character string (one right-justified character per word) to a Reverse Polish Table.

Call from FORTRAN:

```
CALL CEXPR(char,out,lmt,size)
```

char	Expression character string array (terminated by a zero byte)
out	Reverse Polish Table array for output
lmt	Upper limit to the size of the Reverse Polish Table
size	Actual size of the Reverse Polish Table on return

An expression can contain a mixture of symbols, literals, numeric values and operators. Expressions handled by this routine are FORTRAN-like in syntax. The legal operators are listed later in this section.

Operator hierarchy follows FORTRAN rules and does parenthesis nesting. Symbols are defined as 1- to 8-character strings having unknown value to CEXPR. CEXPR simply flags the strings for the caller. The first character cannot be numeric. Literals are 1 through 15 character strings enclosed by double quotes (").

A character string consisting of numeric digits is taken as a 64-bit integer. A trailing B signifies a Boolean number.

The following is the format of the Reverse Polish Table.

```
         0       8       16      24      32      40      48      56   63
         ________________________________________________________________
    0   |                          ITEM                         |T | OC  |
        |_______________________________________________________|__|_____|
    .
    .
    .
    n
```

Field	Bits	Description
ITEM$_i$	0-55	15-character data item from the character string
T	56-57	2-bit type field describing ITEM 01 1- through 8-character symbol 10 1-word binary value 11 1- through 15-character literal string (delimiting double quotes not included) 00 Null, OC is valid

Field	Bits	Description
OC	58-63	6-bit operator code.

Operator code (octal)	Operator
01	.XOR.
10	.OR.
20	.AND.
30	.NOT.
40	.EQ.
41	.NE.
42	.LT.
43	.GT.
44	.LE.
45	.GE.
50	+
51	-
52	+ (unary)
53	- (unary)
60	*
61	/

CEXPR references the following internal routines: CEXERR, ITBLSTO, ITBLSYM, ITBLOC, LITERAL, ERR1, ERR2, PUTLINE.

JOB CONTROL LANGUAGE SYMBOL ROUTINES

The job control language symbol routines manipulate job control language (JCL) symbols for conditional JCL statements.

JSYMSET allows the user to change a value for a JCL symbol. The value is the actual value given to the symbol, with no evaluation being performed.

Call from CAL:

```
CALLV $JSYMSET
```

Entry:

(S1)	Symbol name
(A1)	Points to value
(A2)	Length of value

Call from FORTRAN:

```
CALL JSYMSET(sym,val[,len])
```

sym	Valid JCL symbol name
val	Actual value assigned to the symbol
len	Length of *val* in words (elements)

JSYMGET allows user programs to retrieve JCL symbols. The JSYMGET routine also allows the creation of JCL symbols if they do not exist.

Call from CAL:

```
CALLV $JSYMGET
```

Entry:

(S1)	Symbol name; left-justified, zero-filled.
(A1)	Where value is returned
(A2)	Number of words in the area that is to receive the value

Exit:

(A2)	Actual length of the value

Call from FORTRAN:

```
CALL JSYMGET(sym,val[,len])
```

sym	Valid JCL symbol name
val	Receives the actual value of the symbol if the value buffer is large enough and the symbol currently has a value
len	Length of the value buffer in words (elements). *len* is changed to the actual length of the symbol's value (less than or equal to the value buffer).

SKOL RUN-TIME SUPPORT ROUTINES

SKOL run-time support routines include character-string manipulation, character-code translation, and error handling applications.

CHARACTER-STRING MANIPULATION ROUTINES

I00DEL deletes a SKOL string or substring and returns the length of the resulting string.

```
len=I00DEL(string,max,clen,fchar,lchar)
```

len	New length of the string addressed by *string*
string	A SKOL string
max	Maximum size of the string
clen	Current length of the string
fchar	Index of the first character to be deleted
lchar	Index of the last character to be deleted

I00MVC replaces a SKOL string or substring with a single character and returns the length of the resulting string.

```
len=I00MVC(string,max,clen,fchar,lchar,ichar,lnum)
```

len	New length of string addressed by *string*
string	A SKOL string
max	Maximum size of string
clen	Current length of string
fchar	Index of first character to be replaced
lchar	Index of last character to be replaced
ichar	Character to be inserted in place of deleted substring
lnum	SKOL source line number of call to I00MVC

I00MVM replaces a SKOL string or substring with another SKOL string or substring and returns the length of the resulting string.

```
len=I00MVM(string,max,clen,fchar,lchar,sstri,sfchar,slchar,lnum)
```

len	New length of string addressed by *string*
string	A SKOL string
max	Maximum size of string
clen	Current length of string
fchar	Index of first character to be replaced
lchar	Index of last character to be replaced
sstri	Second SKOL string
sfchar	Index of first character in second string to be inserted
slchar	Index of last character in second string to be inserted
lnum	SKOL source line number of call to I00MVM

CHARACTER-CODE TRANSLATION ROUTINES

I00ORD returns the internal SKOL code for a given ASCII character.

NOTE

I00SETUP must be called before any call to I00ORD.
I00SETUP is described later in this section.

---

ord=I00ORD(*char*,*lnum*)

ord	Ordinal number of the given character
char	ASCII character; left-justified, blank-filled.
lnum	SKOL source line number of the call to I00ORD

I00READ reads a logical record in A1 format and converts each word containing an ASCII character, left-justified, blank-filled, to its internal SKOL code (character ordinal). Internal codes for characters are defined by a TYPE CHAR statement.

NOTE

I0OSETUP must be called before any call to I0OREAD. I0OSETUP is described later in this section.

```
CALL I0OREAD(dn,array,size,fchar,lchar,lnum)
```

dn	Name or unit number of the dataset to be read
array	Array to receive the character ordinals
size	Size of array
fchar	Index of first character position to be filled
lchar	Index of last character position to be filled
lnum	SKOL source line number of call to I0OREAD

I0OWRITE writes characters defined by the TYPE CHAR statement and converts character ordinals (internal SKOL codes) to ASCII characters in A1 format, left-justified, blank-filled, and then writes them as a logical record.

```
CALL I0OWRITE(dn,array,size,ford,lord,lnum)
```

dn	Name or unit number of the dataset to be written
array	Array containing the character ordinals
size	Size of array
ford	Index of first ordinal to be output
lord	Index of last ordinal to be output
lnum	SKOL source line number of call to I0OWRITE

I00SETUP initializes a SKOL program's table (128 words long) for direct translation of ASCII character codes to internal ordinal numbers.

```
CALL I00SETUP(ord,lnum)
```

ord	Highest ordinal number to be used in the calling program
lnum	SKOL source line number of the call to I00SETUP

ERROR-HANDLING ROUTINE

I00ERR handles run-time errors in SKOL programs, writes an error message to $OUT and $LOG, and then terminates the job step.

```
CALL I00ERR(ord,lnum)
```

ord	Ordinal number of a SKOL error
lnum	SKOL source line number where the call to I00ERR was inserted by the macro translator

ERROR PROCESSING ROUTINES

Error processing routines issue logfile messages and abort the job. See the CRAY-OS Message Manual, publication SR-0039, for the messages issued by these routines.

ARERP% processes $ARLIB errors.

Call from CAL:

```
CALLV ARERP%
```

Entry:
(S1) Error message ID defined in $SYSTXT, AR*nnn*

FTERP% processes $FTLIB errors.

Call from CAL:

```
CALLV FTERP%
```

Entry:

(S1)	Error message ID defined in $SYSTXT, FT*nnn*

IOERP% processes $IOLIB errors.

Call from CAL:

```
CALLV IOERP%
```

Entry:

(S1)	Error message ID defined in $SYSTXT, FT*nnn*
(S2)	Unit identification, ASCII, left-adjusted
(S3)	If nonzero, format FWA
(S4)	Character position in format
(S5)	If nonzero, string buffer FWA
(S6)	Character position in string buffer
(S7)	Mode ('INPUT', 'OUTPUT', etc.)

NLERP% processes NAMELIST I/O errors.

Call from CAL:

```
CALLV NLERP%
```

Entry:

(S1)	Error message ID defined in $SYSTXT, FT*nnn*
(S2)	Unit identification, Hollerith, left-justified, zero-filled
(S3)	Record name
(S4)	If nonzero, item name
(S5)	If nonzero, address of string buffer
(S6)	If nonzero, character position in buffer
(S7)	Mode ('INPUT', 'OUTPUT', etc.) Hollerith, left-justified, zero-filled

---

NOTE

If a format or string buffer address is provided, the format or buffer is listed on $OUT and, if possible, the error position is marked.

$SCERP processes $SCILIB errors. Aborts with traceback.

Call from CAL:

```
CALLV $SCERP
```

Entry:
  (S1)   Error message ID defined in $SYSTXT, SC*nnn*

SLERP% processes $SYSLIB errors and abort with traceback.

Call from CAL:

```
CALLV SLERP%
```

Entry:
  (S1)   Error message ID defined in $SYSTXT, SL*nnn*
    If error message has two arguments:
  (S2)   Name of unit, ASCII format
  (S3)   ASCII descriptive word ('READ','WRITE', or program name of calling routine); blank-filled
    If error message has one argument:
  (S2)   Dataset name or unit number

UTERP% processes $UTLIB errors.

Call from CAL:

```
CALLV UTERP%
```

Entry:
  (S1)   Error message ID defined in $SYSTXT, UT*nnn*

BYTE AND BIT MANIPULATION ROUTINES

Byte and bit manipulation routines move bytes and bits between variables and arrays, compare bytes, perform searches with byte count as a search argument, perform conversion on bytes, and pack and unpack bits.

MOVE BYTES ROUTINE

The STRMOV routine moves a specific number of bytes from one variable or array to another. The arguments *num*, *isb*, and *idb* must be greater than 0 for any data to be moved. The argument *dest* must be declared long enough to hold *num* bytes or spill will occur and data will be destroyed.

Call from FORTRAN:

```
CALL STRMOV(src,isb,num,dest,idb)
```

src	Variable or array of any type or length that contains the bytes to be moved. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
isb	Starting byte in the *src* string. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
num	Type integer variable, expression, or constant that contains the number of bytes to be moved
dest	Variable or array of any type or length that contains the string to receive the bytes
idb	Type integer variable, expression, or constant that contains the starting byte to receive the data. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

MOVE BITS

The MOVBIT routine moves a specified number of bits from one variable or array to another. The arguments *num*, *isb*, and *idb* must be greater than 0 for any data to be moved. The argument *dest* must be declared long enough to hold *num* bits or spill will occur and data will be destroyed.

Call from FORTRAN

```
CALL MOVBIT(src,isb,num,dest,idb)
```

src	Variable or array of any type or length that contains the string of bits to be moved.
isb	Starting bit in the *src* string. Type integer variable, expression, or constant. Bits are numbered from 1, beginning at the leftmost bit position of *src*.
num	Type integer variable, expression, or constant that contains the number of bits to be moved
dest	Variable or array of any type or length that contains the string to receive the bits
idb	Type integer variable, expression, or constant that contains the starting bit to receive the data. Bits are numbered from 1, beginning at the leftmost bit position of *dest*.

MOVE CHARACTERS ROUTINE

MVC moves characters from one memory area to another. The source and destination strings can occur on any byte boundary. The move is performed one character at a time from left to right. The destination string can overlap the source string.

Call from FORTRAN:

CALL MVC(s_1,j_1,s_2,j_2,k)

s_1	Word address of the Hollerith source string
j_1	Byte offset from the word address of the source string of the first byte of the source string (the high-order byte of the first word of the source string is byte number 1)
s_2	Word address of the Hollerith destination string

j_2	Byte offset from the word address of the destination string of the first byte of the destination string (the high-order byte of the first word of the destination string is byte number 1)
k	Number of bytes to be moved

For example, the first byte of an array can be copied throughout the array by the following call (where K is the length of the array in bytes).

```
CALL MVC(ARRAY,1,ARRAY,2,K-1)
```

REPLACE BYTE ROUTINE

PUTBYT replaces a specified byte in a variable with a specified value. The high-order 8 bits of the first word of the variable is called byte 1.

Call from FORTRAN:

```
CALL PUTBYT(string,position,value)
```

string	The address of a variable. The variable may be of any type except CHARACTER.
position	The number of the byte to be replaced. The parameter must be an integer greater than or equal to 1.
value	The new value to be stored into the byte. The parameter must be an integer with value between 0 and 255.

If PUTBYT is called as an integer function (having been properly declared in the user program), the value of the function is the value of the byte stored.

If *position* is less than or equal to 0, no change to the destination string is made; the function value is -1.

COMPARE BYTES FUNCTION

The KOMSTR function performs an unsigned, twos complement compare of a specified number of bytes from one variable or array with a specified number of bytes from another variable or array. The arguments *num*, *isb*, and *idb* have no size limits.

Call from FORTRAN:

```
ik = KOMSTR(src,isb,num,dest,idb)
```

ik	Type integer result. Contains 0 if the strings *src* and *dest* are equal. Contains 1 if *src* > *dest* and contains -1 if *src* < *dest*.
src	Variable or array of any type or length containing the first string to compare
isb	Starting byte in the *src* string. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
num	Type integer variable, expression, or constant that contains the number of bytes to compare.
dest	Variable or array of any type or length containing the second string to compare
idb	Type integer variable, expression, or constant that contains the starting byte. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

SEARCH BYTES ROUTINE

The FINDCH subroutine searches a variable or array for the occurrence of a specified character string. The result is equal to the byte count in the variable or array where the string was found, or equal to 0 if no string was found.

Call from FORTRAN:

```
CALL FINDCH(chrs,len,str,ls,nb,ifnd)
```

chrs	Variable or array of any type or length containing the search string

len	Length of the search string in bytes (must be from 1 to 256). Type integer variable, expression, or constant.
str	Variable or array of any type or length that is searched for a match with *chrs*
ls	Starting byte in the *str* string. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *str*.
nb	Number of bytes to be searched. Type integer variable, expression, or constant.
ifnd	Type integer result

ASCII TO INTEGER ROUTINE

The CHCONV subroutine converts a specified number of ASCII characters to an integer value. Blanks in the input field are treated as zeros. A minus sign encountered anywhere in the input field produces a negative result. Input characters other than blank, digits 0 through 9, minus sign, or more than one minus sign, produce a fatal error.

Call from FORTRAN

```
CALL CHCONV(src,isb,num,ir)
```

src	Variable or array of any type or length containing ASCII data or blanks
isb	Starting character in the *src* string. Type integer variable, expression, or constant. Characters are numbered from 1, beginning at the leftmost character position of *src*.
num	Number of ASCII characters to convert. Type integer variable, expression, or constant.
ir	Type integer result

INTEGER TO ASCII ROUTINES

The BICONV subroutine converts a specified integer to an ASCII string. The string generated by BICONV is blank-filled, right-justified, and has a maximum width of 256 bytes. If the specified field width is not long enough to hold the converted integer number, left digits are truncated and no indication of overflow is given. If the number to be converted is negative, a minus sign is positioned in the output field to the left of the first significant digit.

The BICONZ subroutine is the same as BICONV except that the output string generated is zero-filled, right-justified.

Call from FORTRAN:

```
CALL BICONV(int,dest,isb,len)
CALL BICONZ(int,dest,isb,len)
```

int	Type integer variable, expression, or constant containing the integer value to be converted
dest	Variable or array of any type or length to contain the ASCII result
isb	Starting byte count to generate the output string. Type integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
len	Desired length of the output string. Type integer variable, expression, or constant.

PACK, UNPACK

The PACK and UNPACK routines compress and expand stored data.

PACK takes the 1, 2, 4, 8, 16, or 32 rightmost bits of several partial words and concatenates them into full 64-bit words. The formula for the number of full words is shown in equation 1.

Equation 1:

$$n=(nw \mathrm{~x~} nbits)/64$$

nw	Number of partial words
nbits	Number of rightmost bits of each partial word that is useful data
n	Number of resulting full words

Equation 1 restricts *nw* x *nbits* to a multiple of 64.

Call from FORTRAN:

```
CALL PACK(p,nbits,u,nw)
```

p	Vector of packed data
nbits	Number of rightmost bits of data in each partial word. Must be 1, 2, 4, 8, 16, or 32.
u	Vector of partial words to be compressed
nw	Number of partial words to be compressed

UNPACK reverses the action of PACK and expands full words of data into a larger number of right-justified partial words. This routine assumes *nw* x *nbits* to be a multiple of 64.

Call from FORTRAN:

```
CALL UNPACK(p,nbits,u,nw)
```

p	Vector of full 64-bit words to be expanded
nbits	Number of rightmost bits of data in each partial word. Must be 1, 2, 4, 8, 16, or 32.
u	Vector of unpacked data
nw	Number of resulting partial words

MISCELLANEOUS SPECIAL PURPOSE ROUTINES

Each miscellaneous routine described below has a separate purpose; therefore, the routines are not grouped.

ACTTABLE returns Job Accounting Table (JAT).

Call from FORTRAN:

```
CALL ACTTABLE(array,count)
```

array	An array
count	Count; the first *count* words of the JAT are returned in the array. If *count* is greater than the size of the JAT, the array is padded with zeros.

DRIVER allows a user to directly program a CRAY channel on an IOS[†]. This is a privileged function available to all single-tasked jobs. It is prohibited to multitasking jobs.

Call from FORTRAN:

```
CALL DRIVER(array,lentry,status)
```

array First element of the integer parameter block array. The array is *lentry* words long. In all cases, FUNC, PLEN and LN are required in the parameter block and COSS is returned in the parameter block ERPB. DP is always sent to the driver and returned to the user.

For the FORTRAN user, FUNC, DIR and COSS are literal strings (i.e. set FUNC to 'CFN$OPE' and DIR to 'DIR$INP' to open an input channel. 'DRS$RSV' in COSS means the channel is reserved for another job).

'CFN$OPE' subfunction opens a channel; a job cannot access a channel until it opens the channel. DRNM, DRTO, DIR, and OPD are required.

'CFN$CLS' subfunction closes a channel. Any open channels are closed during termination. DIR is required.

[†] This capability is available only on the MIOP.

'CFN$RD', 'CFN$RDH', 'CFN$RDD' subfunctions read data. BAD and DLEN are required; TLEN is returned. For read, either the channel is read to central memory or data is moved from IOS buffer memory to central memory (if a read/hold was done prior to this read). For read/hold, a second read is performed and the data is held in buffer memory for a subsequent read. For read/read, a second read to central memory is done.

'CFN$WT', 'CFN$WTH', 'CFN$WTD' subfunctions write data. BAD and TLEN are required; TLEN is returned. For write, data is written to the channel from central memory or buffer memory (if a write/hold was done prior to this request). For write/hold, a second buffer of data is moved to and held in buffer memory for a subsequent write. For write/write a second write is performed from central memory.

'CFN$DMIN'-'CFN$DMAX' subfunctions are defined by the driver. DFP and DIR are required.

lentry Length of the parameter block entry in *array*; integer variable set by the user.

status Status; integer variable set by the system. On return, *status* is 0 if no errors occurred and the job must poll COMS for nonzero. When COMS is nonzero, the driver has completed the request and the driver status is in DRS. See the individual driver specifications for driver status. If *status* is nonzero on return, COSS contains the error code and the request is not sent to the driver.

If no errors occurred and if *status* is nonzero on return, COSS contains the error code.

ECHO allows the user to turn on and off the classes of messages to the user logfile.

Call from FORTRAN:

```
CALL ECHO('ON'L[,param-array],'OFF'L[,param-array])
```

param-array

Array of message class names or 'ALL'. Message class names are defined in the CRAY-OS Version 1 Reference Manual, publication SR-0011.

ERECALL allows a job to suspend itself until one or more selected events occur. This routine is available to all single-tasking jobs; it is prohibited to multitasking jobs.

When event monitoring is enabled, the system monitors selected events for a job, keeping track of which ones have occurred. Monitoring is disabled at the beginning of each job step and can be enabled by making a system request, specifying the events to monitor. Once monitoring is enabled, a job can make a system request to change the events that are to be monitored, get a map indicating which of the monitored events occurred, go into event recall until one of the selected events occurs, or disable monitoring.

When monitoring is enabled, a map of occurred events is returned to the user and discarded by the system. If monitoring was disabled when the enable occurred, the map is 0.

When the events to be monitored are changed, a map of occurred events is returned to the user and discarded by the system.

When a map of occurred events is requested, the map is returned to the user and discarded by the system.

When recall is requested and the map of occurred events is 0, the job is suspended for an event until one of the events occurs. If the map is nonzero, the map is returned to the user immediately and discarded by the system.

When recall is disabled, the map of occurred events is discarded by the system.

Call from FORTRAN:

```
CALL ERECALL(func,status,sevents,to,oevents,levents)
```

func Integer variable set by the user to define what information or action is requested.

'DISABLE' Disables event monitoring. All other words ignored.

'ENABLE' Enables event monitoring or changes the events to be monitored. *levents* and *sevents* are required. If *levents* is 0, timeout is the only enabled event. Timeout is enabled in order to prevent a job remaining indefinitely in recall. *levents* and *oevents* are returned by the system. *to* is ignored.

'RECALL'	Places the job in recall. An error is returned in *status* if monitoring is disabled. *to* is required, *sevents* is ignored. *levents* and *oevents* are set by the system. If *to* is 0, an installation-defined default, I@TODEF, is used. If *to* is specified, but less than the installation defined minimum, I@TOMIN, the installation minimum is used with no notification. If *levents* is 0 on return, timeout is the only event that occurred.
'RETURN'	Requests *levents* and *oevents* be set by the system; all other words are ignored. An error is returned in *status* if monitoring is disabled.

status Status; integer variable set by the system. *Status* is 0 if no errors occurred; otherwise, see the parameter block ERPB definition for error codes. The codes are returned as blank-filled literal strings (for example, ERER$BFN will be returned as 'ERER$BFN').

sevents Integer array set by the user containing the events to be monitored. *levents* is the number of events specified in *sevents*. The events can be selected from the following.

'IJ'	Inter-job message received
'UO'	Unsolicited operator message received†
'OR'	Operator reply received†

The following are privileged:

'CH'	Channel driver done
'IQ'	SDT placed in Input queue†
'OQ'	SDT placed in Output queue†

to Timeout duration in milleseconds (rightmost 24 bits); integer variable set by the user.

oevents Integer array set by the system to the occurred events. *levents* is the number of event words that have been placed in *oevents* by the system. See *sevents* for possible values.

---

† Deferred implementation

levents	Integer value specifying the number of events in either *sevents* or *oevents*. For ENABLE, the user sets *levents* to the number of event words that the user has placed in *sevents*. On return from ENABLE, RECALL, and RECALL *levents* is the number of event words that the system has placed in *oevents*.

GETB1 returns the contents of register B1 (address of calling subroutine name) in register S1.

Call from FORTRAN:

```
b1=GETB1
```

b1	Calling subroutine name address (B1)

GETLPP returns the lines per page from field JCLPP of the JCB in register S1.

Call from FORTRAN:

```
lpp=GETLPP( )
```

lpp	Lines per page (type integer)

ICEIL is an integer function which returns the integer ceiling of a rational number represented as two integer parameters.

Call from FORTRAN:

```
i=ICEIL(j,k)
```

j	The numerator of a rational number
k	The denominator of a rational number

The value of the function *i* is the smallest integer larger than or equal to *j*/*k*.

IGTBYT extracts a specified byte from a variable. The high-order 8 bits of the first word of the array are called byte 1. The value of the byte is returned as an integer value between 0 and 255.

Call from FORTRAN:

```
byte=IGTBYT(string,position)
```

string	Address of an array; the array can be of any type except CHARACTER.
position	Number of the byte to be extracted. Must be an integer larger than or equal to 1. If *position* is less than or equal to 0, the function value returned is -1. If *position* is greater than 0, the function value is an integer between 0 and 255.

IJCOM allows a job to communicate with another job. This feature is available to all single-tasking jobs. Inter-job communication is prohibited to multitasking jobs.

Call from FORTRAN:

```
CALL IJCOM(array,larray,lentry,nentry,status)
```

array First element of the integer parameter block array. An installation-defined maximum number of parameter blocks (I@MPBS) can be specified in array. The array is *larray* words long and each of the *nentry* parameter blocks in it is *lentry* words long. See the ERPB table definition for a description of a parameter block. The FORTRAN user ignores LINK; the system links the entries together for the user. In all cases, FUNC, RID, and PLEN are required in each parameter block and STAT is set in each parameter block by the system.

For the FORTRAN user, FUNC and STAT are literal strings (for example, set FUNC to 'IJM$OPEN' to open a path. 'IJMS$BP' in STAT means the path was busy).

'IJM$NOP'	Subfunction is a no op.
'IJM$REC'	Subfunction marks the job as receptive. RCB is required; all other words are ignored.
'IJM$OPEN'	Subfunction initiates an attempt to open a communication path with another job. HLEN, TID, and NCB are required; all other words are ignored.

'IJM$ACCE'	Subfunction accepts a request from another job to open communication. TID, HLEN, and NCB are required; all other words are ignored.
'IJM$REJE'	Subfunction rejects a request from another job to open communication. TID is required; all other words are ignored.
'IJM$SNDM'	Subfunction sends a message to another job. NCB, TID, BADD and BLEN are required; all other words are ignored.
'IJM$SNDL'	Subfunction sends a message to an attached job's logfile. This is a privileged function. TID, OVR, FCS, FCU, CLS and BADD are required; all other words are ignored.
'IJM$CLOS'	Closes a communication path. Either NCB and TID or neither are required; all other words are ignored. If NCB and TID are specified only the path determined by RID and TID is closed; otherwise all communication paths with RID are closed.
'IJM$END'	Subfunction marks the job as not receptive. All other words are ignored. Existing communication paths are not affected.

larray	Length of array; integer variable set by the user.
lentry	Length of each parameter block entry in array; integer variable set by the user.
nentry	Number of parameter blocks in array; integer variable set by the user. Default is 1.
status	Status; integer variable set to 0 if no errors occurred. If status is nonzero, STAT contains the error code. If multiple parameter blocks are used, all STAT fields must be examined if status is nonzero.

INSASCI% inserts ASCII parameters into a message. Insertion is controlled by the Message Control Table (MCT).

Call from CAL:

```
CALLV INSASCI%
```

Entry:

(S1) Message number; used as an offset into the MCT to select the entry (message) to be processed.

(S2) Base address of parameters to be inserted. Parameters must be in the order defined by the parameter control entry (PCE). Parameters can span words, and each parameter is assumed to be right-justified. See figures 7-2 and 7-3.

(S3) Base address of the MCT. The format of the MCT is shown in figure 7-4.

Exit:

The message assembled in the destination buffer, unless the message number is out of range or the MCT is improperly defined, in which case, error message SL000 is issued and the routine aborted

Entry (one per variation in message format):

```
0       8        16       24       32       40       48       56     63
| NP    |////////////////////////////////////|         PCA              |
```

Figure 7-2. Parameter Control Table

Field	Bits	Description
PCNP	0-7	Number of parameters to be inserted
PCPCA	40-63	Address of parameter control entry (PCE)

Entry (one per parameter):

```
0       8        16       24       32       40       48       56     63
|/////////////////////////////////////////////| SW  | SB  |    LEN      |
```

Figure 7-3. Parameter control entry

Field	Bits	Description
PCSW	40-45	Starting word in text (0, 1,...)
PCSB	46-51	Starting bit in starting word (0...63)
PCLEN	52-63	Length in bits (maximum is 64 bits)

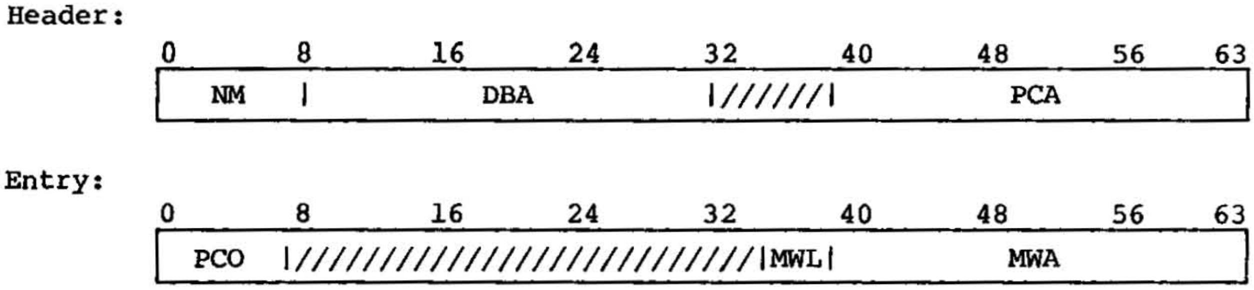


Figure 7-4. Message Control Table

Field	Bits	Description
MCNM	0-7	Number of messages defined in MCT
MCDBA	8-31	Address of buffer in which to assemble message
MCPCA	40-63	Address of Parameter Control Table (PCT)
MCPCO	0-7	Offset into PCT; selects desired entry.
MCMWL	36-39	Message length in words
MCMWA	40-63	Message address in words

JNAME returns job name.

Call from FORTRAN:

```
name=JNAME(result)
```

name	Job name; left-justified with trailing blanks.
result	Returned job name

LGO loads an absolute program from a local dataset containing the binary image as the first record. The loaded program is then executed. Control does not return to LGO.

Call from CAL:

```
CALLV $LGO
```

Entry:
(S1) Dataset name containing the absolute load module

Call from FORTRAN:

```
CALL LGO('dn'L)
```

dn	Dataset name containing the absolute load module

LOC returns memory address of specified variable or array.

Call from FORTRAN:

```
address=LOC(arg)
```

address	Argument address
arg	Argument whose address is to be returned

LOGECHO writes the last line formatted by $WFD as a message to the $LOG file.

Call from FORTRAN:

```
CALL LOGECHO
```

Entry:
A WRITE or PRINT statement has previously been executed in a FORTRAN program, or a WRITE or OUTPUT statement has been executed previously in a SKOL program, so that the characters written still remain in $WFD's local buffer. The character string should end with a period. The first period encountered ends the message.

Exit:
The message has been written to $LOG (using REMARK2) unless $WFD's buffer contains nothing but blanks after the print control character in the buffer.

MEMORY determines or changes a job's memory allocation and/or mode of field length reduction.

Call from FORTRAN:

```
CALL MEMORY(code,value)
```

code	Determines what information or action is requested (blank-filled).
'UC'	*value* specifies the number of words to be added to (if value is positive) or subtracted from (if value is negative) the end of the user code/data area.
'FL'	*value* specifies the number of words of field length to be allocated to the job. If FL is specified and *value* is not, the new field length is set to the maximum allowed the job.
'USER'	The job is put in user-managed field length reduction mode. *value* is ignored.
'AUTO'	The job is put in automatic field length reduction mode. *value* is ignored.
'MAXFL'	The maximum field length allowed the job is returned in *value*.
'CURFL'	The current field length is returned in *value*.
'TOTAL'	The total amount of unused space in the job is returned in *value*.
value	An integer value or variable when *code* is 'UC' or 'FL'. An integer variable that is to contain a returned value if code is 'CURFL', 'MAXFL', or 'TOTAL'.

Memory can be added to or deleted from the end of the user code/data area by using the 'UC' code. If the user code/data area is expanded, the new memory is initialized to an installation-defined value.

The job's field length can be changed by using the 'FL' code. The field length is set to the larger of the requested amount rounded up to the nearest multiple of 512 decimal words or the smallest multiple of 512 decimal words large enough to contain the user code/data, LFT, DSP and buffer areas. The job is placed in user-managed field length reduction mode for the duration of the job step.

The job's mode of field length reduction can be changed by using either the 'USER' or 'AUTO' code. When 'USER' code is specified, the job is placed in user mode until a subsequent request is made to return it to automatic mode. When AUTO code is specified, the job is placed in automatic mode and the field length is reduced to the smallest multiple of 512 decimal words that can contain the user code/data, LFT, DSP, and buffer areas.

The job's maximum or current field length can be determined by the 'MAXFL' or 'CURFL' code. The total amount of unused space in the job can be determined by the 'TOTAL' code.

The job is aborted if filling the request would result in a field length greater than the maximum allowed the job. The maximum is the smaller of the total number of words available to user jobs minus the job's JTA or the amount determined by the MFL parameter on the JOB statement.

Examples:

```
CALL MEMORY('FL')
```

The job's field length is set to the maximum allowed the job and the job is placed in user mode for the duration of the job step.

```
CALL MEMORY('AUTO')
```

The job's field length is reduced to a minimum and the job is placed in automatic mode.

```
CALL MEMORY('UC',-5)
CALL MEMORY('UC',IVAL)
```

where IVAL is -5

The job's user code/data area is reduced by five words.

NACSED returns edition number for permanent dataset just accessed by CALL ACCESS(DN=*dn*).

Call from FORTRAN:

```
ed=NACSED( )
```

ed Edition number in binary form

OPTION changes the user-specified options for a job.

Call from FORTRAN:

```
CALL OPTION (['LPP'L, '45'L][,'STAT', {'ON' }])
                                       {'OFF'}
```

OVERLAY loads overlays and transfers to the overlay address (see CRAY-OS Version 1 Reference Manual, publication SR-0011, for details of the OVERLAY routine).

Call from FORTRAN:

CALL OVERLAY (dn,lev_1,lev_2[,*recall*])

dn	Dataset in which overlay resides
lev_1	Overlay level 1 (LEV1)
lev_2	Overlay level 2 (LEV2)
recall	Optional message: RECALL-DON'T RELOAD IF IN MEMORY. If the overlay is already in memory, use the overlay that is already there.

PERF provides an interface to the hardware performance monitor feature on the CRAY X-MP mainframe. Thirty-two counters are available, arranged into 4 groups of 8 counters each. (See table 7-1.) Only one counter can be accessed at a time.

Call from FORTRAN:

CALL PERF(*func*,*group*,*buffer*,*bufl*)

func Performance monitor function. Either an integer function number (PM$*xxx*, where *xxx* is the function name), or one of the following ASCII strings, left-justified, and zero-filled.

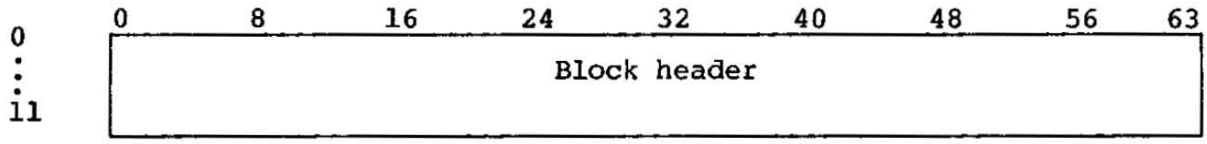
'ON'L	Enable performance monitoring.
'OFF'L	Disable performance monitoring.
'REPORT'L	Report current performance monitor statistics.
'RESET'L	Report current statistics, then clear performance monitor tables.

group	Performance monitor group number (integer). See table 7-1 for group numbers and their corresponding counters and counter contents.
buffer	First-word address of a performance monitor request buffer.
bufl	Number of words in the buffer array

The PERF request block format contains a fixed header and a variable number of subblocks following the header. The first three words of the header are set in subroutine PERF before calling the system, while the remaining words in the header are returned by the system.

The words in the block header allow one to analyze the information returned in the subblocks without the use of constants, allowing programs to continue to execute correctly when the contents of the header or subblocks change.

Block header format:



Field	Word	Description
HMRSF	0	Subfunction (PM$ON,PM$OFF,PM$REP,PM$RST)
HMRGN	1	Group number (0 through 3) for PM$ON
HMRNW	2	Length of request block
HMRNU	3	Number of words used
HMRBH	4	Number of words in block header
HMRTS	5	Set nonzero if block too small
HMRCT	6	Offset to first group counter in subblock
HMRCP	7	Offset to first group accounted cycles
HMRGE	8	Length of counter group entry in subblock
HMRNC	9	Number of counters in each group entry
HMRNG	10	Number of groups in each subblock
HMRLE	11	Length of subblock entries

Timing subblocks are returned for every REPORT and RESET call. Each subblock contains hardware performance monitor data from a single COS user task.

The address of the first timing subblock is at (BLOCK FWA)+(contents of block header field HMRBH), with the next following (contents of block header field HMRLE) word after the first. Subblocks end when the offset to the next block would start after (contents of block header field HMRNU) words.

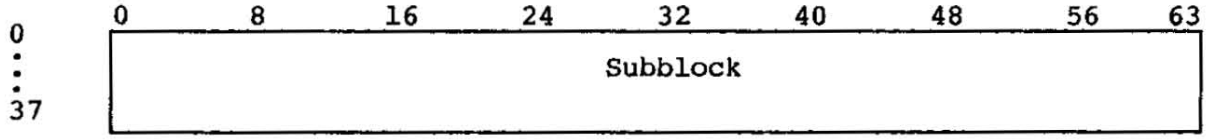
Each subblock contains a 2-word header, with fields HMTN and HMGRP. HMTN is the COS user task number associated with the subblock.

HMGRP is the last hardware performance monitor group number active for the subblock.

Within the subblock, there are (contents of block header field HMRNG) performance monitor groups reported. Each group report consists of two fields: counters associated with the group, and the number of CPU cycles that were accounted for while the specified monitor was active. The offset to the first group counter is (contents of block header field HMRCT) words into the subblock; there are (contents of block header field HMRNC) counters for each performance monitor group. The offset to the first group's accounted CPU cycle is at (contents of block header field HMRCP).

Timing groups within a subblock follow each other by (contents of block header field HMRGE) words.

Subblock format:



Field	Word	Description
HMTN	0	User task number
HMGRP	1	Latest performance monitor group number
HMCNT0	2-9	Group 0, counter 0 through 7
HMCCY0	10	Group 0 accounted CP cycles
HMCNT1	11-18	Group 1, counter 0 through 7

Field	Word	Description
HMCCY1	19	Group 1 accounted CP cycles
HMCNT2	20-27	Group 2, counter 0 through 7
HMCCY1	28	Group 1 accounted CP cycles
HMCNT3	29-36	Group 3, counter 0 through 7
HMCCY1	37	Group 1 accounted CP cycles

$RBN and RBN convert trailing blanks to nulls.

Call from CAL:

```
CALLV $RBN
```

Entry:

(S1)	Argument to be converted

Exit:

(S1)	Result of conversion

Call from FORTRAN:

```
noblanks=RBN(blanks)
```

blanks	Argument to be converted
noblanks	Argument after conversion

$RNB and RNB convert trailing nulls to blanks.

NOTE

FORTRAN programs using RBN or RNB must specify the function to be type integer.

Table 7-1. Performance counter group descriptions

group	Performance Counter	Description
0		Number of:
	0	Instructions issued
	1	CPs holding issue
	2	Fetches
	3	I/O references
	4	CPU references
	5	Floating-point add operations
	6	Floating-point multiply operations
	7	Floating-point reciprocal operations
1		Hold issue conditions:
	0	Semaphores
	1	Shared registers
	2	A registers and functional units
	3	S registers and functional units
	4	V registers
	5	V functional units
	6	Scalar memory
	7	Block memory
2		Number of:
	0	Fetches
	1	Scalar references
	2	Scalar conflicts
	3	I/O references
	4	I/O conflicts
	5	Block references
	6	Block conflicts
	7	Vector memory references
3		Number of:
	0	000 - 017 instuctions
	1	020 - 137 instructions
	2	140 - 157, 175 instructions
	3	160 - 174 instructions
	4	176, 177 instructions
	5	Vector integer operations
	6	Vector floating-point operations
	7	Vector memory references

Call from CAL:

```
CALLV $RNB
```

Entry:
(S1) Argument to be converted
Exit:
(S1) Result

Call from FORTRAN:

```
blanks=RNB(noblanks)
```

blanks	Argument after conversion
noblanks	Argument to be converted

REMARK enters a message in the user and system logfiles preceded by a message prefix, 'UT008'. REMARK allows a maximum of 71 characters.

Call from FORTRAN:

```
CALL REMARK(message)
```

message	Message terminated by a zero byte or a 71-character message

REMARK2 enters a message in the user and system logfiles. REMARK2 allows a maximum of 79 characters including message ID.

Call from FORTRAN:

```
CALL REMARK2(message)
```

message	Message terminated by a zero byte or maximum of 79 characters

REMARKF enters a message in the user logfile. Up to 12 variables can be passed in arguments 2 through 13. The variables must be of type integer, real, or logical so that they occupy only one word each. The message is prefixed by 'UT009' unless the caller supplies a prefix. The caller is judged to have supplied a prefix if the characters '*b*-*b*' where *b*=blank, appear in columms 6 through 8 of the formatted message.

Sample FORTRAN calling sequences:

```
  10030 FORMAT ('CA001 - ', I4, ' errors')
        ASSIGN 10030 TO LABEL
        CALL REMARKF (LABEL, IERRCNT)

  10770 FORMAT ('PD001 - ACCESS ', A8,A7,' ED=', I4, ';')
        ASSIGN 10770 TO LABEL
        CALL REMARKF (LABEL, DN(1), DN(2), ED)
```

Call from FORTRAN:

```
CALL REMARKF(var,fvar,[,fvar2,...fvar12])
```

var	Variable containing the address of a format statement for ENCODE
fvar	Address of variable

$SEGRES initializes execution of a segmented program.
$SEGCALL services intersegment subroutine calls in a segmented program.

These routines require the creation of a Segment Description Table (SDT) whose address is stored in /$SEGRES/. Calls to these routines are made through code modifications performed by SEGLDR. These routines should not be called directly by a user program.

SSWITCH tests sense switch.

Call from FORTRAN:

```
CALL SSWITCH(swnum,result)
```

swnum	Switch number (integer)
result	Value is 1 if switch value ranges from 1 to 6 and switch is on. Value is 2 if switch value is less than 1 or greater than 6 or if switch is off (integer).

SYSTEM makes requests of the operating system.

Call from FORTRAN:

```
STATUS=SYSTEM(function,arg1,arg2)
```

function	Executive request number
arg_1	Optional argument (required by some requests)
arg_2	Optional argument (required by some requests)
STATUS	Status returned in S0 (function dependent)

TR translates a string in place from one character code to another using a user-supplied translation table. The same size characters are assumed.

Call from FORTRAN:

```
CALL TR(s,j,k,table)
```

s	First word of an array containing characters to be translated
j	Byte offset within array *s* where the first character to be translated occurs
k	Number of characters to be translated
table	Translation table

The translation table must be considered a 256-byte vector. As each character to be translated is fetched, it is used as an index into the translation table. The new value of the character is the contents of the translation table byte addressed by the old value. (The first byte of the translation table is considered to be byte 0.)

WFBUFFER returns a character from the output buffer belonging to the $WFD module. (This buffer contains the last line formatted by ENCODE, WRITE, or PRINT. Each word in the buffer contains a character in FORTRAN R1 format.)

Call from CAL:

```
CALL WFBUFFER,(arg)
```

Entry:

arg	Index of the desired character

Exit:

(S1)	Requested character right-justified, zero-filled

Users can call procedures from a CFT or CAL program, using the F$CSP system call. FORTRAN users should use the SYSTEM routine with the appropriate parameters. CAL users should set up the call parameters in the proper registers and EX.

PASCAL SUBPROGRAMS 8

INTRODUCTION

Pascal subprograms are procedures and functions that reside in the Pascal runtime library, $PSCLIB. Some of these subprograms are implicitly called from Pascal programs. For example, the Pascal source statement

```
WRITE (10)
```

is translated into a call on P$WI. To explicitly call such a subprogram, declare it at the beginning of the Pascal program with an external directive. Each subprogram that can be explicitly called is listed with its accompanying external directive.

Some of the Pascal subprograms are called from CAL only; their parameter linkages are such that they cannot be called directly from Pascal programs. For details of Pascal program organization, refer to the Pascal Reference Manual, CRI publication SR-0060.

P$$$HPAD

P$$$HPAD returns the address of the program's heap control block, which is the data area used by P$MEMRY, P$NEW and P$DISP to control use of the dynamic variable storage area. P$$$HPAD is used only within these routines. No parameters are required.

```
FUNCTION P$$$HPAD: INTEGER; EXTERNAL;
     .
     .
     .
Integer variable := P$$$HPAD
```

P$ABORT

P$ABORT calls P$OSDXP to abort the current job step. No parameters are required.

```
PROCEDURE P$ABORT; EXTERNAL;
     .
     .
     .
P$ABORT
```

P$BREAK

P$BREAK is used with P$DBP to perform data breakpoint checking for Pascal programs. P$BREAK informs P$DBP of the address to be monitored. See the Pascal Reference Manual, CRI publication SR-0060 for a detailed description and examples of the use of data breakpoint checking.

```
PROCEDURE P$BREAK (VAR V: INTEGER; ISVAR: BOOLEAN); EXTERNAL;
     .
     .
     .
Integer variable := address;
```

```
P$BREAK (integer variable, FALSE)
```

P$CALLR

P$CALLR returns the name of the current procedure's caller. No parameters are required.

```
PROCEDURE P$CALLR (VAR CALLER:  ALFA); EXTERNAL;
            .
            .
            .
P$CALLR (alfa variable)
```

P$CBV

P$CBV provides access to call-by-value routines from Pascal programs. It takes a procedural parameter describing the call-by-value routine to be called and an array parameter describing register contents to be in effect when the routine is called. P$CBV returns the register contents at the end of the call-by-value routine.

```
TYPE REGISTER = (AREG, SREG);
REGISTERS = ARRAY REGISTER, 0..7 OF INTEGER;
VAR REGS: REGISTERS;
PROCEDURE call-by-value-routine; EXTERNAL;
PROCEDURE P$CBV (PROCEDURE P; VAR R: REGISTERS); EXTERNAL;
     .
     .
     .
REGS := desired contents of registers;
```

P$CBV (call-by-value-routine, REGS)

P$CONNEC

P$CONNEC changes the COS filename (7-character maximum) to the alfa type variable specified by STR. The change becomes effective only after a call to P$RESET or P$REWRITE.

F is the file variable; STR is the new name for the file.

```
PROCEDURE P$CONNEC (VAR F:TEXT;STR:ALFA);EXTERNAL;
     .
     .
     .
P$CONNEC (file name, alfa string)
```

P$CONNEC implements the Pascal procedure CONNECT.

P$DATE

P$DATE calls P$OSDDT to obtain the current date in the form DD/MM/YY. An ALFA variable to receive the current date should be passed as a parameter.

```
PROCEDURE P$DATE (VAR CURRDATE: ALFA); EXTERNAL;
      .
      .
      .
P$DATE (alfa variable)
```

P$DBP

Calls to P$DBP are generated by the Pascal compiler for modules compiled with the BP+ option. P$DBP checks the contents of the word registered with the P$BREAK routine. If they have changed since the last call to P$DBP, a breakpoint message is written to $OUT. P$CBV is callable from CAL only. See the Pascal Reference Manual, CRI publication SR-0060 for more information on data breakpoint checking.

P$DEBUG

P$DEBUG (call by value only) produces and prints the runtime post-mortem dump (pmd) and, depending on the value of S1, takes one of the following actions.

S1 < 0	Continues execution
S1 in (0..1000)	Performs an ENDP macro (same as halt)
S1 > 1000	Aborts

The Pascal-callable entry name is P$RTMSG.

P$DISP

P$DISP deallocates the storage area at the address indicated by the pointer variable. Since P$DISP performs the inverse operation of P$NEW, the word at PTR-1 must contain the same linkage information put there previously by procedure P$NEW. The pointer PTR must conform to the limits of the heap.

P$DISP attempts to meld the newly freed area with an already existing free area. Should no meld be possible, the routine inserts the area into the free chain in ascending address order. Storage acquired from the system by exceeding the minimum heap size is not returned to the system.

```
PROCEDURE P$DISP (PTR:POINTER);EXTERNAL;
      .
      .
      .
P$DISP (pointer variable)
```

P$DISP implements the Pascal procedure DISPOSE.

P$DIVMOD

P$DIVMOD (call by value from CAL only) performs a single-precision integer divide and modulus, returning the quotient in register S1 and the remainder in register S2.

P$DIVMOD implements the Pascal procedure DIV.

P$ENDP

P$ENDP executes a normal program exit. Processing resumes with the next job control statement if reprieve processing is not enabled for normal job step termination. No parameters are required.

```
PROCEDURE P$ENDP;EXTERNAL;
      .
      .
      .
P$ENDP
```

P$EOF

P$EOF returns the end-of-file status for a Pascal text/record file.

```
FUNCTION P$EOF (VAR F:FILE OF type):BOOLEAN;EXTERNAL;
      .
      .
      .
Boolean variable:=P$EOF (file name)
```

P$EOF implements the Pascal procedure EOF.

P$EOLN

P$EOLN returns the end-of-line status for a Pascal text file.

```
FUNCTION P$EOLN (VAR F:TEXT):BOOLEAN;EXTERNAL;
      .
      .
      .
Boolean variable:=P$EOLN (file name)
```

P$EOLN implements the Pascal procedure EOLN.

P$GET

P$GET reads a logical record from file F and then sets the corresponding status bits in the file variable to reflect the current file status before returning to the caller.

For files randomly accessed, the file is pre-positioned. For TEXT files, P$GET calls P$OSDRC to fill the buffer.

For the FILE OF *type* clause, P$GET calls P$OSDRW. The user's variable serves as the I/O buffer by the setting of the buffer address and length with the statement:

```
USER VARIABLE := FILE VARIABLE ;
```

P$GET implements the Pascal procedure GET.

```
PROCEDURE P$GET (VAR F:FILE OF type);EXTERNAL;
      .
      .
      .
P$GET (file variable)
```

P$HALT

P$HALT stops program execution and invokes the runtime routine P$DEBUG, which provides a post-mortem dump.

```
PROCEDURE P$HALT;EXTERNAL;
      .
      .
      .
P$HALT
```

P$HALT implements the Pascal procedure HALT.

P$JTIME

P$JTIME calls P$OSDJT to obtain the number of CPU seconds used by the job. No parameters are required.

```
TYPE STRING = PACKED ARAAY (1.80) OF CHAR;
FUNCTION P$JTIME: REAL; EXTERNAL;
      .
      .
      .
Real variable := P$JTIME
```

P$LOGMSG

P$LOGMSG writes a message of up to 80 characters to the logfile. The character string must be truncated by a 0 byte or be 80 characters long.

```
TYPE STRING = PACKED ARRAY [1.80] OF CHAR;
PROCEDURE P$LOGMSG (STR:STRING);EXTERNAL;
      .
      .
      .
P$LOGMSG ('character string')
```

P$LSTREW

P$LSTREW is an alternate entry point into P$REWRITE which rewrites a file without positioning it.

```
PROCEDURE P$LSTREW (VAR F: TEXT); EXTERNAL;
      .
      .
      .
P$LSTREW (textfile)
```

P$MEMRY

P$MEMRY obtains information about the heap in which dynamic variables are allocated. It takes an integer parameter describing the aspect of the heap to be returned.

```
FUNCTION P$MEMRY (REQUEST: INTEGER): INTEGER; EXTERNAL;
      .
      .
      .
Integer variable := P$MEMRY (request code)
```

Request codes for P$MEMRY:

Request	Action
1	QUERY SIZE OF HEAP
2	QUERY HIGH WATER MARK
3	QUERY LOW WATER MARK
4	QUERY NUMBER OF ALLOCATED HEAP AREAS
5	QUERY NUMBER OF TIMES HEAP HAS GROWN
6	QUERY NUMBER OF TIMES HEAP HAS SHRUNK
7	QUERY NAME OF LAST HEAP ROUTINE USED
8	QUERY NAME OF LAST ROUTINE TO CALL A HEAP ROUTINE
9	QUERY LAST HEAP ITEM ALLOCATED/DEALLOCATED/ETC.
10	QUERY SIZE OF LARGEST FREE BLOCK
11	QUERY AMOUNT HEAP CAN SHRINK
12	QUERY AMOUNT HEAP CAN GROW
13	QUERY HEAP F.W.A.
14	QUERY HEAP L.W.A.
15	QUERY NUMBER OF TIMES HPALLOC CALLED
16	QUERY NUMBER OF TIMES HPDEALLC CALLED
17	DO A HEAP SHRINK
18	DO A HEAP INTEGRITY CHECK
19	DUMP HEAP STATISTICS TO $OUT
20	DUMP HEAP CONTROL WORDS TO $OUT(ALL BLKS)
21	DUMP HEAP CONTROL WORDS TO $OUT(FREE BLKS VIA NEXT)
22	DUMP HEAP CONTROL WORDS TO $OUT(FREE BLKS VIA LAST)

P$MOD

P$MOD (call by value from CAL only) returns the modulus in register S1. P$MOD implements the Pascal procedure MOD.

P$NEW

P$NEW allocates the storage area at the address indicated by the pointer variable. SIZE contains the number of words to be allocated. The contents of the word preceding the user area contains linkage information for use by the P$NEW and P$DISP routines. Upon completion of P$NEW, PTR contains the address of the allocated user area. This area is zero-filled by the P$NEW routine.

```
PROCEDURE P$NEW (VAR PTR:POINTER;SIZE:INTEGER);EXTERNAL;
     .
     .
     .
P$NEW (pointer variable,integer variable)
```

P$NEW implements the Pascal procedure NEW.

P$OSDBS

P$OSDBS is an operating system dependent routine that rewinds a dataset. It takes the DSP offset of the dataset to be rewound as a parameter.

```
PROCEDURE P$OSDBS (DSP: INTEGER); EXTERNAL;
     .
     .
     .
P$OSDBS (DSP offset of dataset)
```

P$OSDDT

P$OSDDT is an operating system dependent routine that returns the current date in the form DD/MM/YY. An ALFA variable to receive the current date should be passed as a parameter.

```
PROCEDURE P$OSDDT (VAR CURRDATE: ALFA); EXTERNAL;
     .
     .
     .
P$OSDDT (alfa variable)
```

P$OSDEP

P$OSDEP is an operating system dependent routine that opens a dataset for processing. It takes the dataset name as a parameter and returns a DSP offset (which is used to describe the dataset to the other P$OSDxx routines that perform I/O).

```
FUNCTION P$OSDEP (DSN: ALFA): INTEGER; EXTERNAL;
      .
      .
      .
Integer variable := P$OSDEP ('dataset name')
```

P$OSDJT

P$OSDJT is an operating system dependent routine that returns the number of CPU seconds used by the job. No parameters are required.

```
FUNCTION P$OSDJT: REAL; EXTERNAL;
      .
      .
      .
Real variable := P$OSDJT
```

P$OSDLM

P$OSDLM is an operating system dependent routine that writes a message to the logfile ($LOG). The logfile message is terminated by a 0 byte (ASCII NUL) or the 80th character, whichever comes first.

```
TYPE STRING = PACKED ARRAY 1..80 OF CHAR;
PROCEDURE P$OSDLM (STR: STRING); EXTERNAL;
      .
      .
      .
P$OSDLM ('character string')
```

P$OSDPR

P$OSDPR is an operating system dependent routine that sets the prompt string for the interactive dataset $IN.

```
PROCEDURE P$OSDPR (PROMPT: ALFA); EXTERNAL;
      .
      .
      .
P$OSDPR (DSP offset of dataset, 'prompt string')
```

P$OSDQI

P$OSDQI is an operating system dependent routine that returns TRUE if the dataset whose DSP offset is passed as a parameter is an interactive dataset.

```
FUNCTION P$OSDQI (DSP: INTEGER): BOOLEAN; EXTERNAL;
      .
      .
      .
Boolean variable := P$OSDQI (DSP offset of dataset)
```

P$OSDRC

P$OSDRC is an operating system dependent routine that reads a character record from a dataset into a buffer. The DSP offset of the dataset to be read, the address and length of the buffer, and an integer variable to receive the return code are passed as parameters. After the call to P$OSDRC, the return code variable indicates the dataset's status as follows:

Returned value	0	Implies normal return; length of record is (-returned value)
	= 0	EOF, EOD, or null record
	= 1	Insufficient space in buffer
	= 2	Unrecoverable hardware error

```
CONST MAXBUFFER = buffer length;
TYPE BUFFER = ARRAY 1..MAXBUFFER OF CHAR;
VAR BUF: BUFFER;
PROCEDURE P$OSDRC (DSP: INTEGER; VAR BUF: BUFFER;
BUFLENGTH: INTEGER; VAR RETURNINFO: INTEGER); EXTERNAL;
      .
      .
      .
P$OSDRC (DSP offset of dataset, BUF, MAXBUFFER, integer variable)
```

P$OSDRP

P$OSDRP is an operating system dependent routine that enables the job to perform reprieve processing. It takes a procedural parameter describing the routine to receive control in the event of a reprieve, a buffer area to hold the exchange package, and a mask value selecting the types of errors to be reprieved as parameters. See the Macros and Opdefs Reference Manual, CRI publication SR-0012 for a description of allowable mask values.

```
TYPE EXCHANGEPACKAGE = ARRAY 1..40 OF INTEGER;
VAR EP: EXCHANGEPACKAGE;
PROCEDURE reprieve routine;
PROCEDURE P$OSDRP (PROCEDURE P; EP: EXCHANGEPACKAGE;
MASK: INTEGER);
      .
      .
      .
P$OSDRP (reprieve routine, EP, mask)
```

P$OSDRW

P$OSDRW is an operating system dependent routine that reads a record from a dataset into a buffer. The DSP offset of the dataset to be read, the address and length of the buffer, and an integer variable to receive the dataset's status are passed as parameters. The status codes returned by P$OSDRW are identical to those returned by P$OSDRC above.

```
CONST MAXBUFFER = buffer length;
TYPE BUFFER = ARRAY 1..MAXBUFFER OF INTEGER;
VAR BUF: BUFFER;
PROCEDURE P$OSDRW (DSP: INTEGER; VAR BUF: BUFFER;
BUFLENGTH: INTEGER; VAR RETURNINFO: INTEGER); EXTERNAL;
      .
      .
      .
P$OSDRW (DSP offset of dataset, BUF, MAXBUFFER, integer
variable)
```

P$OSDTM

P$OSDTM is an operating system dependent routine that returns the current date in the form HH:MM:SS. An ALFA variable to receive the current time should be passed as a parameter.

```
PROCEDURE P$OSDTM (VAR CURRTIME: ALFA); EXTERNAL;
      .
      .
      .
P$OSDTM (alfa variable)
```

P$OSDWC

P$OSDWC is an operating system dependent routine that writes a character record to a dataset. It takes the DSP offset of the dataset and a buffer's address and length as parameters.

```
CONST MAXBUFFER = maximum buffer length;
TYPE BUFFER = ARRAY 1..MAXBUFFER OF CHAR;
VAR BUF: BUFFER;
PROCEDURE P$OSDWC (DSP: INTEGER;
     BUF: BUFFER;
     BUFLENGTH: INTEGER); EXTERNAL;
     .
     .
     .
P$OSDWC (DSP offset of dataset, BUF, number of characters in buffer)
```

P$OSDWF

P$OSDWF is an operating system dependent routine that writes an end of file mark to the dataset whose DSP offset is passed as a parameter.

```
PROCEDURE P$OSDWF (DSP: INTEGER); EXTERNAL;
     .
     .
     .
P$OSDWF (DSP offset of dataset)
```

P$OSDWR

P$OSDWR is an operating system dependent routine that writes a record to a dataset. The DSP offset of the dataset and the buffer's address and length are passed as parameters.

```
CONST MAXBUFFER = length of buffer;
TYPE BUFFER = ARRAY 1..MAXBUFFER OF INTEGER;
VAR BUF: BUFFER;
PROCEDURE P$OSDWR (DSP: INTEGER;
     BUF: BUFFER;
     BUFLENGTH: INTEGER); EXTERNAL;
     .
     .
     .
P$OSDWR (DSP offset of dataset, BUF, length of buffer)
```

P$OSDXP

P$OSDXP is an operating system dependent routine that either terminates the program normally or aborts the current job step, depending on whether its Boolean parameter is FALSE or TRUE, respectively.

```
PROCEDURE P$OSDXP (DOABORT: BOOLEAN); EXTERNAL;
     .
     .
     .
P$OSDXP (TRUE or FALSE)
```

P$PAGE

P$PAGE writes a page-eject function to the file specified by F.

```
PROCEDURE P$PAGE (VAR F:TEXTFILE);EXTERNAL;
     .
     .
     .
P$PAGE (file name)
```

P$PAGE implements the Pascal procedure PAGE.

P$PUT

P$PUT writes a logical record from the file specified by F and then sets the corresponding status bits in the file variable to reflect the current file status before returning to the caller.

For files randomly accessed, the file is pre-positioned. For TEXT files, P$PUT pads the line to the end of the current word with blanks and then calls P$OSDWC.

For the FILE OF *type* clause, P$PUT calls P$OSDWR.

```
PROCEDURE P$PUT (VAR F:FILE OF type);EXTERNAL;
      .
      .
      .
P$PUT (file variable)
```

P$PUT implements the Pascal procedure PUT.

P$RB

P$RB reads a 1- to 5-character Boolean argument from the text file F. Leading blanks are ignored. Arguments can be upper or lower case.

```
PROCEDURE P$RB (VAR F:TEXT;VAR BOOL:BOOLEAN);EXTERNAL;
      .
      .
      .
P$RB (file name,Boolean argument)
```

P$RCH

P$RCH returns, right-justified, the next character in the file. File status functions P$EOF and P$EOLN are set where applicable. If P$EOLN is set, the next record is read before return.

```
PROCEDURE P$RCH (VAR F:TEXT;VAR CH:CHAR);EXTERNAL;
     .
     .
     .
P$RCH (file variable,character variable)
```

P$READ

P$READ performs a call to the procedure P$GET and then moves the file buffer to a user variable.

```
PROCEDURE P$READ (VAR F:TEXT;VAR X:USERVARIABLE);EXTERNAL;
     .
     .
     .
P$READ(file variable,variable)
```

P$READLN

P$READLN fills the user text file buffer with the next record in the file.

```
PROCEDURE P$READLN (VAR F:TEXT);EXTERNAL;
     .
     .
     .
P$READLN (file variable)
```

P$READLN implements the Pascal procedure READLN.

P$REPRV

P$REPRV allows a program to trap certain runtime errors. The errors to be trapped are selected by the bit mask that is passed as a parameter. See the Macros and Opdefs Reference Manual, CRI publication SR-0012 for a description of allowable mask values.

```
PROCEDURE P$REPRV (MASK: INTEGER); EXTERNAL;
      .
      .
      .
P$REPRV (mask selecting errors to be reprieved)
```

P$RESET

P$RESET resets the specified file. If the file is not open, P$OPEN is called to open the file for reading.

Upon exit from P$RESET, the file pointer is positioned at the beginning of the file. P$RESET implements the Pascal procedure RESET.

```
PROCEDURE P$RESET (VAR F:FILE OF type);EXTERNAL;
      .
      .
      .
P$RESET (file variable)
```

P$REWRIT

P$REWRIT resets the specified file. If the file is not open, P$OPEN is called to open the file for writing.

Upon exit from P$REWRIT, the file pointer is positioned at the beginning of the file.

```
PROCEDURE P$REWRIT (VAR F:FILE OF type);EXTERNAL;
     .
     .
     .
P$REWRIT (file variable)
```

P$REWRIT implements the Pascal procedure REWRITE.

P$RF

P$RF performs a character-string read and returns the next signed real number in the file. Preceding spaces and end-of-lines are skipped.

```
PROCEDURE P$RF (VAR F:TEXT;VAR R:REAL);EXTERNAL;
     .
     .
     .
P$RF (file variable,real variable)
```

P$RI

P$RI returns the next signed integer in the file. The range of the integer read is limited to -MAXINT≤integer≤MAXINT.

Preceding spaces and end-of-lines are skipped. The character sequence read must conform to the proper signed-integer syntax.

```
PROCEDURE P$RI (VAR F:TEXT;VAR INT:INTEGER);EXTERNAL;
     .
     .
     .
P$RI (file variable,integer variable)
```

P$ROUND

P$ROUND (call by value from CAL only) returns in register S1 the rounded value of the argument as an integer. P$ROUND implements the Pascal procedure ROUND.

P$RSTR

P$RSTR reads the number of characters specified by WIDTH and places them in the packed character string specified by STR. The first character is right-justified in the first word. If the length of the current line exceeds the character string, the file pointer is left in mid-record. If the current line is exhausted by the read, left-over positions in the character string are blank-filled on the right and the function P$EOLN is set.

```
PROCEDURE P$RSTR (VAR F:TEXT;STR:STRING;WIDTH:INTEGER);EXTERNAL;
      .
      .
      .
P$RSTR (file variable,character string,integer variable)
```

P$RTIME

A call to P$RTIME is generated by the Pascal program at the end of a main program compiled with the BT+ option. It takes the statistics collected by P$TIMER and prints a timing report to $OUT. No parameters are required.

```
PROCEDURE P$RTIME; EXTERNAL;
      .
      .
      .
P$RTIME
```

P$RTMSG

P$RTMSG is the Pascal-callable entry name for the subprogram P$DEBUG. P$RTMSG produces and prints the runtime post-mortem dump (pmd) and, depending on the value of X, takes one of the following actions.

X < 0	Continues execution
X in (0..1000)	Performs an ENDP macro (same as halt)
X > 1000	Aborts

```
PROCEDURE P$RTMSG (X:INTEGER);EXTERNAL;
     .
     .
     .
P$RTMSG
```

P$RUNTIM

P$RUNTIM (call by value from CAL only) initializes the Pascal environment with the following procedures:

1. Creates a heap of size A1 + A2 words
2. Allocates space for the stack from the heap
3. Initializes P$$$HEAP structure
4. Opens the input file and initializes the variable
5. Opens the output file and initializes the variable

The input and output files are not initialized if the address given for their variables is zero. P$RUNTIM initializes a fixed-extent stack. However, the heap is extendable to the memory limit. See procedures P$RESET and P$REWRIT for information on opening Pascal textfiles. See procedures P$NEW and P$DISP for information on using the Pascal heap.

At the time of the call, the registers specified contain the following information.

A1	Heap size in words
A2	Stack size in words
A3	Address of input file variable
A4	Address of output file variable

P$SFRAME

P$SFRAME returns a pointer to the base of the stack frame of the calling procedure. No parameters are required.

```
FUNCTION P$SFRAME: INTEGER; EXTERNAL;
      .
      .
      .
Integer variable := P$SFRAME
```

P$TIME

P$TIME calls P$OSDTM to obtain the current time in the form HH:MM:SS. An ALFA variable to receive the current time should be passed as a parameter.

```
PROCEDURE P$TIME (VAR CURRTIME: ALFA); EXTERNAL;
      .
      .
      .
P$TIME (alfa variable)
```

P$TIMER

Calls to P$TIMER are generated by the Pascal compiler for modules compiled with the BT+ option. P$TIMER collects runtime statistics to be printed later by P$RTIME. P$TIMER is callable only from CAL.

P$TRACE

P$TRACE is called by P$RTMSG to produce a stack walkback. No parameters are necessary.

```
PROCEDURE P$TRACE; EXTERNAL;
      .
      .
      .
P$TRACE
```

P$TRUNC

P$TRUNC (call by value from CAL only) truncates the value of the argument and returns the truncated value in register S1. P$TRUNC implements the Pascal procedure TRUNC.

P$WB

P$WB writes a Boolean argument to a text file.

```
PROCEDURE P$WB (VAR F:TEXT;BOOL:BOOLEAN;W:INTEGER);EXTERNAL;
      .
      .
      .
P$WB (file variable,Boolean variable,integer width)
```

P$WCH

P$WCH writes the character that is right-justified in CH as the next character in the file.

```
PROCEDURE P$WCH (VAR F:TEXT;CH:CHAR);EXTERNAL;
      .
      .
      .
P$WCH (file variable,character variable)
```

P$WEOF

P$WEOF calls P$OSDWF to write an end of file mark on the textfile passed as a parameter.

```
PROCEDURE P$WEOF (VAR T: TEXT); EXTERNAL;
     .
     .
     .
P$WEOF (textfile)
```

P$WI

P$WI writes the integer with the width specified by FIELDW, to the file.

```
PROCEDURE P$WI (VAR F:TEXT;INT:INTEGER;FIELDW:INTEGER);EXTERNAL;
     .
     .
     .
P$WI(file variable,integer variable,integer width)
```

P$WO

P$WO writes the octal representation of an integer value to a textfile. It takes a textfile, an integer value, and the desired field width as parameters.

```
PROCEDURE P$WO (VAR T: TEXT; IVAL, WIDTH: INTEGER); EXTERNAL;
     .
     .
     .
P$WO (textfile, integer value, field width)
```

P$WRFIX

P$WRFIX writes the real number to a Pascal text file according to the rules for fixed-point representations.

```
PROCEDURE P$WRFIX (VAR F:TEXT;E:REAL;W,FR:INTEGER);EXTERNAL;
      .
      .
      .
P$WRFIX (file variable,real variable,integer width,
              integer fraction digits)
```

P$WRFIX implements the Pascal fixed-point write procedure.

P$WRFLT

P$WRFLT writes the real number to a Pascal text file according to the rules for floating-point representations.

```
PROCEDURE P$WRFLT (VAR F:TEXT;E:REAL;W:INTEGER);EXTERNAL;
      .
      .
      .
P$WRFLT (file variable,real variable,integer width)
```

P$WRITE

P$WRITE moves the user variable to the file buffer and performs a call to the procedure P$PUT.

```
PROCEDURE P$WRITE (VAR F:TEXT;X:USERVARIABLE);EXTERNAL;
      .
      .
      .
P$WRITE (file variable,variable)
```

P$WRITE implements the Pascal procedure WRITE.

P$WRITLN

P$WRITLN causes a line to be written to the textfile passed as a parameter. P$WRITLN implements the standard Pascal procedure WRITELN.

```
PROCEDURE P$WRITLN (VAR T: TEXT); EXTERNAL;
      .
      .
      .
P$WRITLN (textfile)
```

P$WSTR

P$WSTR writes the packed character string STRING to the file F. The first character is right-justified in the first word. The length and width of the character string are specified by LEN and FIELDW, respectively.

```
PROCEDURE P$WSTR (VAR F, TEXT; STRING:  CHARSTRING; LEN, WIDTH:
INTEGER)' EXTERNAL;
      .
      .
      .
P$WSTR(F, STRING, LEN, FIELDW)
```

P$WSTR implements the Pascal character string write procedure.

MULTITASKING SUBPROGRAMS 9

Multitasking subprograms create and synchronize parallel tasks within programs. They are grouped in the following categories:

- Task routines
- Lock routines
- Event routines
- Utility routines

For further information on using these subprograms in a multitasking environment, see the Multitasking User's Guide, CRI publication SN-0222.

TASK ROUTINES

Task routines handle tasks and task-related information.

TSKSTART initiates a task.

Call from FORTRAN:

```
CALL TSKSTART (task-array,name[,list])
```

task-array
Task control array (described under subtitle task control array) used for this task. Word 1 must be set. Word 3, if used, must also be set. On return, word 2 is set to a unique task identifier that must not be changed by the program.

name External entry point at which task execution begins. This name must be declared EXTERNAL in the program or subroutine making the call to TSKSTART.

NOTE

CFT does not allow a program unit to use its own name in this parameter.

---

list (optional parameter)
List of arguments being passed to the new task when it is entered. This list can be of any length. See the Multitasking User's Guide, CRI publication SN-0222, for restrictions on arguments included in *list*.

Call from CAL:

```
CALL TSKSTART,(task-array name,A register,list),USE=A7
```

task-array name
Control array (described under subtitle task control array)

A register
An A register containing the parcel address of the routine to multitask

list List of arguments passed to the new task. No limitations on the length of the list.

Example:

```
      PROGRAM MULTI
      INTEGER TASK1ARY(3),TASK2ARY(3)
      EXTERNAL PLLEL
      REAL DATA(40000)
C
C LOAD DATA ARRAY FROM SOME OUTSIDE SOURCE
C ...
C
C CREATE TASK TO EXECUTE FIRST HALF OF THE DATA
      TASK1ARY(1)=3
      TASK1ARY(3)='TASK 1'
C
      CALL TSKSTART(TASK1ARY,PLLEL,DATA(1),20000)
C
C CREATE TASK TO EXECUTE SECOND HALF OF THE DATA
      TASK2ARY(1)=3
      TASK2ARY(3)='TASK 2'
```

```
C
      CALL TSKSTART(TASK2ARY,PLLEL,DATA(20001),20000)
C ...
      END
```

TSKWAIT waits for the indicated task to complete execution.

Call from FORTRAN:

CALL TSKWAIT (*task-array*)

task-array

Task control array (described under subtitle task control array)

Example:

```
      PROGRAM MULTI
      INTEGER TASK1ARY(3),TASK2ARY(3)
      EXTERNAL PLLEL
      REAL DATA(40000)
C
C LOAD DATA ARRAY FROM SOME OUTSIDE SOURCE
C ...
C
C CREATE TASK TO EXECUTE FIRST HALF OF THE DATA
      TASK1ARY(1)=3
      TASK1ARY(3)='TASK 1'
C
      CALL TSKSTART(TASK1ARY,PLLEL,DATA(1),20000)
C
C CREATE TASK TO EXECUTE SECOND HALF OF THE DATA
      TASK2ARY(1)=3
      TASK2ARY(3)='TASK 2'
C
      CALL TSKSTART(TASK2ARY,PLLEL,DATA(20001),20000)
C ...
C NOW WAIT FOR BOTH TO FINISH
      CALL TSKWAIT(TASK1ARY)
      CALL TSKWAIT(TASK2ARY)
C
C AND PERFORM SOME POST-EXECUTION CLEANUP
C ...
      END
```

In the above example, TSKSTART is called once for each of two tasks. As an alternative, the second TSKSTART could be replaced by a call to PLLEL and the TSKWAIT removed. This alternate approach reduces the overhead of the additional task but can make understanding the program structure more difficult. The two approaches produce the same results.

TSKVALUE retrieves the user identifier (if any) specified in the task control array used to create the executing task.

Call from FORTRAN:

```
CALL TSKVALUE (return)
```

return	Integer value that was in word 3 of the task control array (described under subtitle task control array) when the calling task was created. A 0 is returned if the task control array length is less than 3 or if the task is the initial task.

Example:

```
      SUBROUTINE PLLEL(DATA,SIZE)
      REAL DATA(SIZE)
C
C DETERMINE WHICH OUTPUT FILE TO USE
      CALL TSKVALUE(IVALUE)
      IF(IVALUE .EQ. 'TASK 1')THEN
        IUNITNO=3
      ELSEIF(IVALUE .EQ. 'TASK 2')THEN
        IUNITNO=4
      ELSE
        STOP               Error condition; do not continue.
      ENDIF
C ...
      END
```

TSKTEST returns a value indicating whether the indicated task exists.

Call from FORTRAN:

```
return=TSKTEST (task-array)
```

return	A logical .TRUE. if the indicated task exists. A logical .FALSE. if the task was never created or has completed execution.

task-array

Task control array (described under subtitle tasl control array)

NOTE

TSKTEST must be declared LOGICAL in the calling module.

TSKTUNE modifies tuning parameters within the library scheduler. Each parameter has a default setting within the library and can be modified at any time to another valid setting.

NOTE

This routine should not be used when multitasking on a CRAY-1 Computer System.

Because of variability between and during runs, the effects of this routine are not measurable in a batch environment.

Call from FORTRAN:

CALL TSKTUNE(*keyword*$_1$,*value*$_1$,*keyword*$_2$,*value*$_2$,...)

Each keyword is an ASCII character string. Each value is an integer. The parameters must be specified in pairs but the pairs can occur in any order. The following lists the legal keywords. For more information about using this routine, see the Multitasking User's Guide, CRI publication SN-0222.

MAXCPU	Maximum number of COS logical CPUs allowed for the job
DBRELEAS	Deadband for release of logical CPUs
DBACTIVE	Deadband for activation or acquisition of logical CPU
HOLDTIME	Number of clock periods to hold a CPU, waiting for tasks to become ready, before releasing it to the operating system
SAMPLE	Number of clock periods between checks of the ready queue

Example:

```
    CALL TSKTUNE('DBACTIVE',1,'MAXCPU',2)
```

TASK CONTROL ARRAY

Each user-created task is represented by an integer task control array, constructed by the user program. At a minimum, the array must be two Cray words. A third word can be included. Following is the array structure:

<table>
<tr><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56</th><th>63</th></tr>
<tr><td colspan="9">LENGTH</td></tr>
<tr><td colspan="9">TASK ID</td></tr>
<tr><td colspan="9">TASK VALUE</td></tr>
</table>

LENGTH	Length of the array in Cray words. The length must be set to a value of 2 or 3, depending on the optional presence of the task value field. The user sets the length field before creating the task.
TASK ID	A task identifier assigned by the multitasking library when a task is created. This identifier is unique among active tasks within the job step. The multitasking library uses this field for task identification, but the task identifier is of limited use to user programs.
TASK VALUE (optional field)	Field that the user can set to any value before creating the task. If TASK VALUE is used, LENGTH must be set to a value of 3. The task value can be used for any purpose. Suggested values include a programmer generated task name or identifier or a pointer to a task local storage area. During execution, a task can retrieve this value with the TSKVALUE subroutine.

Example:

```
      PROGRAM MULTI
      INTEGER TASKARY(3)
C
C     SET TASKARY PARAMETERS
      TASKARY(1)=3
      TASKARY(3)='TASK 1'
C     ...
      END
```

TSKLIST lists the status of each existing task, telling whether each task is running, ready to run, or waiting. If the task is waiting, the address of the lock or event or the identifier of the task waited upon is reported.

Call from FORTRAN:

```
CALL TSKLIST (dn)
```

dn Optional name or unit number of the dataset to receive the task status list. The default is $OUT.

LOCK ROUTINES

Lock routines protect critical regions of code and shared memory.

LOCKASGN identifies an integer variable that the program intends to use as a lock. This subroutine must be called for each lock variable before its use with any of the other lock subroutines.

Call from FORTRAN:

```
CALL LOCKASGN (name [,value])
```

name	Name of an integer variable to be used as a lock. The library stores an identifier into this variable. The variable should not be modified by the user.
value	The initial integer value of the lock variable. An identifier should be stored into the variable only if it contains the value. If *value* is not specified, an identifier is stored into the variable unconditionally.

LOCKON sets a lock and returns control to the calling task. If the lock is already set, the task is suspended until the lock is cleared by another task and can be set by this one. In either case, the lock will have been set by the task when it next resumes execution of user code.

Call from FORTRAN:

```
CALL LOCKON (name)
```

name	Name of an integer variable used as a lock.

LOCKOFF clears a lock and returns control to the calling task. Clearing the lock allows one of the waiting tasks to resume execution, but this is transparent to the task calling LOCKOFF.

Call from FORTRAN:

```
CALL LOCKOFF (name)
```

name	Name of an integer variable used as a lock.

LOCKREL releases the identifier assigned to the lock. If the lock is set when LOCKREL is called, an error results. This subroutine detects some errors that arise when a task is waiting for a lock that will never be cleared. The lock variable can be reused following another call to LOCKASGN.

Call from FORTRAN:

```
CALL LOCKREL (name)
```

name	Name of an integer variable used as a lock.

LOCKTEST tests a lock to determine its state (locked or unlocked). Unlike LOCKON, the task does not wait. A task using LOCKTEST must always test the return value before continuing.

Call from FORTRAN:

```
return=LOCKTEST (name)
```

return	A logical .TRUE. if the lock was originally in the locked state. A logical .FALSE. if the lock was originally in the unlocked state, but has now been set.
name	Name of an integer variable used as a lock.

NOTE

LOCKTEST must be declared LOGICAL in the calling module.

EVENT ROUTINES

Event routines signal and synchronize between tasks.

EVASGN identifies an integer variable that the program intends to use as an event. Before this routine can be used with any of the other event routines, it must be called for each event variable.

Call from FORTRAN:

```
CALL EVASGN (name [,value])
```

name	Name of an integer variable to be used as an event. The library stores an identifier into this variable. The variable should not be modified by the user.
value	The initial integer value of the event variable. An identifier should be stored into the variable only if it contains the value. If *value* is not specified, an identifier is stored into the variable unconditionally.

EVWAIT delays the calling task until the specified event is posted. If the event is already posted, the task resumes execution without waiting.

Call from FORTRAN:

```
CALL EVWAIT (name)
```

name	Name of an integer variable used as an event.

EVPOST posts an event and returns control to the calling task. Posting the event allows any other tasks waiting on that event to resume execution, but this is transparent to the task calling EVPOST.

Call from FORTRAN:

```
CALL EVPOST (name)
```

name	Name of an integer variable used as an event.

EVCLEAR clears an event and returns control to the calling task. When the posting of a single event is required (a simple signal), EVCLEAR should be called immediately after EVWAIT to note that the posting of the event has been detected.

Call from FORTRAN:

```
CALL EVCLEAR (name)
```

name	Name of an integer variable used as an event.

EVREL releases the identifier assigned to the event. If tasks are currently waiting for this event to be posted, an error results. This subroutine detects erroneous uses of the event beyond the specified region. The event variable can be reused following another call to EVASGN.

Call from FORTRAN:

```
CALL EVREL (name)
```

name	Name of an integer variable used as an event.

EVTEST tests an event to determine its posted state.

Call from FORTRAN:

```
return=EVTEST (name)
```

return	A logical .TRUE. if the event is posted. A logical .FALSE. if the event is not posted.
name	Name of an integer variable used as an event.

NOTE

EVTEST must be declared logical in the calling module.

UTILITY SUBPROGRAMS

Utility subprograms are used by the user-callable multitasking subprograms to perform queue manipulation and task scheduling functions.

$TIBCR% builds a task information block, and returns the address in S1.

Call from CAL:

```
CALLV $TIBCR%
```

Entry:
  (S1)   Address of the task information block for the parent task.
Exit:
  (S1)   Task identifier assigned to the new task.

$TIBDE% deletes a task information block. No explicit arguments are required by this routine. However, it does use the current value of the top of stack pointer (B67). The user should be careful not to destroy this value when calling this routine.

Call from CAL:

```
CALLV $TIBDE%
```

$RDYTSK% readies a task for execution.

Call from CAL:

```
CALLV $RDYTSK%
```

Entry:
(S1)   Address of task information block of task to ready.

NOTE

On the CRAY X-MP system, this routine should be called with the TSKLK hardware semaphore set. This semaphore is cleared before the routine exits.

$RDYQUE% readies a queue of tasks for execution.

Call from CAL:

```
CALLV $RDYQUE%
```

Entry:
(S1)   Address of task information block of first task to ready
(S2)   Address of task information block of last task to ready

---

NOTE

On the CRAY X-MP system, this routine should be called with the TSKLK hardware semaphore set. This semaphore is cleared before the routine exits.

$SUSTSK% suspends execution of the calling task.

Call from CAL:

```
CALLV $SUSTSK%
```

Entry:
(S1)   Address of queue where task is to be placed.

NOTE

On the CRAY X-MP system, this routine should be called with the TSKLK hardware semaphore set. This semaphore is cleared before the routine exits.

$DELTSK% deletes the calling task and activates any tasks waiting for its completion.

Call from CAL:

```
CALLV $DELTSK%
```

$SCHED% schedules logical CPUs for user tasks. This routine is executed when a change in task status occurs. $SCHED% does not necessarily return to the calling routine. It resumes execution at the address specified in the task information block of the first task in the ready queue.

Call from CAL:

```
J $SCHED%
```

Entry:

(S1) Current logical CPU running the calling task.

APPENDIX SECTION

ALGORITHMS

The following algorithms describe the routines in section 3, Common Mathematical Routines. Descriptions are arranged alphabetically by routine name.

ACOS AND ACOSV

Arccosine and arcsine are related by:

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

therefore, if arcsine is calculated, a final subtraction from $\pi/2$ furnishes the arccosine if desired. Moreover, only positive values of x need be considered since

$$\arcsin(-x) = -\arcsin(x).$$

1. Calculate a reduced argument, r, a multiplier, m, and an addend, a, depending on x:

x	r	m	a
$0 \le x \le \sin(\pi/6)$	x	1	0
$\sin(\pi/6) < x \le \sin(\pi/3)$	$T_2(x) = 2x^2 - 1$	1/2	$\pi/4$
$\sin(\pi/3) < x \le \sin(5\pi/12)$	$T_4(x) = 8x^4 - 8x^2 + 1$	1/4	$3\pi/8$
$\sin(5\pi/12) < x \le 1$	$\sqrt{1-x^2}$	-1	$\pi/2$

2. Compute $\arcsin(r)$ using a rational function of the form

$$rP(r^2)/Q(r^2)$$

where P and Q are polynomials of degree 4 in r^2.

3. Multiply the result by m and add a.

4. Insert the result sign, which is the sign of x.

ALOG AND ALOGV

The common and natural logarithms are related by

$$\log(x) = \log e \cdot \ln x$$

therefore, if the natural logarithm is calculated, a final multiplication by $\log e$ furnishes the common logarithm, if desired.

1. Express the argument, x, as $2^n f$ where $1/\sqrt{2} \leq f < \sqrt{2}$. Then

   $$\ln(x) = \ln(2^n f) = \log_2 2^n \cdot \ln 2 + \ln(f) = n \ln 2 + \ln(f)$$

   To determine n and f, observe that x is represented in floating-point format as $x = 2^b c$ where $1/2 \leq c < 1$. When $1/2 \leq c < 1\sqrt{2}$, then $1 \leq 2c < \sqrt{2}$. In this case, $\ln(x) = \ln(2^b c) = \ln[2^{b-1}(2c)] = (b-1)\ln 2 + \ln(2c)$ so that $n = b-1$ and $f = 2c$. When $1/\sqrt{2} \leq c < 1$, we have $n = b$ and $f = c$.

2. Compute $t = (f - 1)/(f + 1)$.

3. Compute $\ln(f)$ using a rational function of the form

   $$2t - tP(t^2)/Q(t^2)$$

   where P and Q are polynomials of degree 3 in t^2.

4. Multiply the exponent n by $\ln 2$ and add to the result.

5. If the common logarithm is desired, multiply by $\log e$.

ATAN AND ATANV

1. Let $z = |x|$. If $z > 1$, replace z with $1/z$

2. Compute the integer $n = \lfloor 16z \rfloor$. Let $t = n/16$.

3. Compute $w = (z - t)/(1 + tz)$

4. Compute arctan(w) using a polynomial $P(w)$ of degree 9.

5. Let $a = 0$ if $z \leq 1$ and $a = \pi/4$ if $z > 1$.

6. Let $c = \arctan(t)$.

7. Form $\arctan(z) = (a - c) + c - P(w)$.

8. Insert the result sign depending on the magnitude of z, the sign of x and the sign of arctan(z).

ATAN2 AND ATAN2V

Use the same algorithm as $ATAN and $ATANV.

CABS AND CABSV

The complex absolute value of $z = x + iy$ is defined by the following equation.

$$|z| = \sqrt{x^2 + y^2}$$

1. Compute $u = \min(|x|, |y|)$ and $v = \max(|x|, |y|)$.

2. If $v = 0$, return with $|z| = 0$.

3. If $v \neq 0$, express $|z|$ as $v\sqrt{1 + (u/v)^2} = v\sqrt{w}$ where w lies in the interval [1,2]. This expression avoids possible overflow in the computation $x^2 + y^2$.

4. Compute an initial approximation to $\sqrt{w}$ as

$$a_0 = 33/32 + 3/8\ (w - 33/32).$$

5. Apply the Newton-Raphson iteration three times to compute $\sqrt{w}$.

$$a_{n+1} = 1/2\ (a_n + w/a_n)$$

three times to compute $\sqrt{w}$.

6. Multiply by v to obtain the final result.

CCOS AND CCOSV

The complex cosine of $z = x + iy$ is defined by

$$\cos(z) = \cos(x) \cosh(y) + \sin(x) \sinh(y)$$

1. Call COSS% to compute $\cos(x)$ and $\sin(x)$.
2. Call COSSH% to compute $\cosh(y)$ and $\sinh(y)$.
3. Multiply $\cos(x)$ by $\cosh(y)$ to obtain the real part of $\cos(z)$.
4. Multiply $\sin(x)$ by $\sinh(y)$ to obtain the imaginary part of $\cos(z)$.

CEXP AND CEXPV

The complex exponential of $z = x + iy$ is defined by

$$e^{z} = e^{x} \cos(y) + ie^{x} \sin(y).$$

1. Call EXP% to compute e^{x}.
2. Call COSS% to compute $\cos(y)$ and $\sin(y)$.
3. Multiply e^{x} by $\cos(y)$ to obtain the real part of e^{z}.
4. Multiply e^{x} by $\sin(y)$ to obtain the imaginary part of e^{z}.

CLOG AND CLOGV

The complex logarithm of $z = x + iy$ is defined by

$$\log(z) = \ln|z| + i \arctan(y/x).$$

1. Call ATAN2% to compute $\arctan(y/x)$ which is the imaginary part of $\log(z)$.
2. Call CABS% to compute $|z|$.
3. Call ALOG% to compute $\ln|z|$ which is the real part of $\log(z)$.

COS AND COSV; COSS AND COSSV

1. Compute $n = \lfloor 4/\pi \ |x|\rfloor$ which is the number of multiples of $\pi/4$ in $|x|$.

2. Define $m = n$ mod 8.

3. Define $a = 1$ if computing cosine; else $a = 0$.

   b = Low-order bit of m

   c = Middle bit of m

   d = High-order bit of m

   e = Argument sign

4. Compute the reduced argument $f = |x| - \pi/4(n + 6)$ in double-precision.

5. Define $t = a \backslash b \backslash c$.†

6. Compute $\sin(f)$ if $t = 0$ and $\cos(f)$ if $t = 1$ using approximations of degree 6 in f^2.

7. Insert the result sign which is $[\#a\&(d\backslash e)]$! $[a\&(d\backslash c)]$.†

COSH AND COSHV

1. Compute the integer n nearest to $|x|/\ln 2$. $x = n\ln 2 + f$ where $|f| \leq \frac{\ln 2}{2}$.

2. Approximate $\cosh(f) - 1$ using a polynomial $P(f^2)$ of degree 5 in f^2.

3. Approximate $\sinh(f)$ using a rational function of the form

   $$P(f) = f^2(1/2 + f^2(P_4 + f^2(P_6 + f^2)(P_8 + f^2)\ P_{10})))$$

4. If $\cosh(x)$ is desired, use the following formula

   $$\cosh(x) = [1 + (P - Q)]2^{-n-1} + [1 + (P + Q)]2^{n-1}$$

   where $P = \cosh(f) - 1$ and $Q = \sinh(f)$. Observe that since

   $$e^x = e^{n\ln 2 + f} = (e^{\ln 2})^n e^f = 2^n e^f$$

---

† \ ≡ Logical difference; # ≡ complement; & ≡ AND; ! ≡ OR.

we have

$$\cosh(x) = \frac{e^x + e^{-x}}{2} = \frac{2^n e^f + 2^{-n} e^{-f}}{2} = 2^{n-1} e^f + 2^{-n-1} e^{-f}$$

$$= \left(\frac{e^f + e^{-f}}{2} - \frac{e^f - e^{-f}}{2} \right) 2^{-n-1} + \left(\frac{e^f + e^{-f}}{2} + \frac{e^f - e^{-f}}{2} \right) 2^{n-1}$$

$$= (\cosh(f) - \sinh(f))\, 2^{-n-1} + (\cosh(f) + \sinh(f))\, 2^{n-1}$$

$$= [1 + (\cosh(f) - 1) - \sinh(f)]\, 2^{-n-1} + [1 + (\cosh(f) - 1) + \sinh(f)]\, 2^{n-1}$$

$$= [1 + (P - Q)]\, 2^{-n-1} + [1 + (P + Q)]\, 2^{n-1}.$$

5. If sinh(x) is desired, return with sinh(x) = sinh(f) if n=0. When $n \neq 0$, use the formula

$$\sinh(x) = 2^{n-3} + (2^{n-3} - [1 + (P - Q)]\; 2^{-n-1}) + [2^{-1} + (P + Q)]\, 2^{n-1}$$

where $P = \cosh(f) - 1$ and $Q = \sinh(f)$. Observe that since

$$e^x = e^{n \ln 2 + f} = e^{\ln 2\ n} e^f = 2^n e^f$$

we have

$$\sinh(x) = \frac{e^x - e^{-x}}{2} = \frac{2^n e^f - 2^{-n-f}}{2} = 2^{n-1} e^f - 2^{-n-1} e^{-f}$$

$$= \left(\frac{e^f + e^{-f}}{2} + \frac{e^f - e^{-f}}{2} \right) 2^{n-1} - \left(\frac{e^f + e^{-f}}{2} - \frac{e^f - e^{-f}}{2} \right) 2^{-n-1}$$

$$= (\cosh(f) + \sinh(f))\, 2^{n-1} - (\cosh(f) - \sinh(f))\, 2^{-n-1}$$

$$= [1 + (\cosh(f) - 1) + \sinh(f)]\, 2^{n-1} - [1 + (\cosh(f) - 1) - \sinh(f)]\, 2^{-n-1}$$

$$= [1 + (P + Q)]\, 2^{n-1} - [1 + (P - Q)]\, 2^{-n-1}$$

$$= [2^{-1} + (P + Q)]\, 2^{n-1} - [1 + (P - Q)]\, 2^{-n-1} + 2^{n-3} + 2^{n-3}$$

$$= 2^{n-3} + (2^{n-3} - [1 + (P - Q)]\, 2^{-n-1}) + [2^{-1} + (P + Q)]\, 2^{n-1}$$

COT AND COTV

1. Compute $n = \lfloor 4/\pi |x| \rfloor$, which is the number of multiples of $\pi/4$ in $|x|$.

2. Define $m = n$ mod 8.

3. Define $a = 1$ if computing cotangent; else, $a=0$.

   b = Low-order bit of m

   c = High-order bit of m

   d = Argument sign

4. Compute the reduced argument $f = |x| - \pi/4(n + 6)$ in double-precision.

5. Define $t = a \setminus b \setminus c$.†

6. Compute tan(f) using a rational function of the form

   $$fP(f^2)/Q(f^2)$$

   where P and Q are polynomials of degree 3 in f^2.

7. If $t = 1$, replace the result with its reciprocal.

8. Insert the result sign, which is $c \setminus d$.†

CSIN AND CSINV

The complex sine of $z = x + iy$ is defined by

$$\sin(z) = \sin(x) \cosh(y) + i\cos(x) \sinh(y).$$

1. Call COSS% to compute cos(x) and sin(x).

2. Call COSSH% to compute cosh(y) and sinh(x).

3. Multiply sin(x) by cosh(y) to obtain the real part of sin(z).

4. Multiply cos(x) by sinh(y) to obtain the imaginary part of sin(z).

---

† $\setminus \equiv$ Logical difference

CSQRT AND CSQRTV

The complex square root of $z = x + iy$ is defined by

$$\sqrt{z} = \sqrt{1/2(|z| + x)} \pm i\sqrt{1/2(|z| - x)} = u + iv$$

where $2uv = y$ and where the ambiguous sign is taken to be the same as the sign of y.

1. If $z = 0$, return with 0 as the answer.
2. Call CABS% to compute $|z|$.
3. Compute $c = \frac{|z| + |x|}{2}$
4. Call SQRT% to compute $u = \sqrt{c}$.
5. Compute $v = \frac{|y|}{2u}$
6. If $x < 0$, swap u and v.
7. Insert the imaginary result sign, which is the sign of y.

CTOCSS, CTOCSV, CTOCVS, AND CTOCVV

Exponentiation of a complex number $z = x + iy$ is defined by

$$\begin{aligned} z^a = e^{a\ln(z)} &= \exp(a\ln|z| + i\arctan(z)) \\ &= \exp[\underbrace{(u\ln|z| - v\arctan\ y/x\ )}_{s} + \underbrace{i(u\arctan\ y/x\ + v\ln|z|)}_{t}] \\ &= e^{s}(\cos(t) + i\sin(t)) \end{aligned}$$

where $a = u + iv$.

Other library routines are called according to the above definition to compute the result.

CTOISS, CTOISV, CTOIVS, AND CTOIVV

1. Return 0 if the base is 0 and the exponent is greater then 0.

2. Return (1,0) if the base is nonzero and the exponent is 0.

3. Compute the absolute value of the exponent e. Write $|e|$ as

   $$e_n 2^n + e_{n-1} 2^{n-1} + \ldots + e_1 2 + e_0$$

   where e_i is either 0 or 1.

4. Form $\underset{e_i = 1}{\pi} z^i$

5. Take the reciprocal if the exponent is negative and return.

CTORSS, CTORSV, CTORVS, AND CTORVV

Exponentiation of a complex number $z = x + iy$ is defined by

$$z^n = e^{n\ln|z|} [\cos(n\arctan(y/x)) + i\sin(n\arctan(y/x))]$$

1. Return 0 if the base is 0 and the exponent is nonzero.

2. Return (1,0) if the base is nonzero and the exponent is 0.

3. Call other library routines according to the above definition to compute the result.

DACOS AND DACOSV

Compute using the identity

$$\arccos(x) = \arctan(\frac{1-x^2}{x})$$

DASIN AND DASINV

Compute using the identity

$$\arcsin(x) = \arctan(x/\sqrt{1 - x^2})$$

DATAN AND DATANV

The subroutine computes the double-precision arctangent of y/x. An alternate 1-argument entry point is provided with x assigned the value 1.

1. If $x = 0$, return with $\sin(y) \cdot \pi/2$ as the result.
2. If $y/x \leq 2^{-17}$, compute the arctangent using the first two terms of its Taylor series.
3. Define $q = |y/x|$, $a = \tan(\pi/16)$ and $b = \tan(3\pi/16)$.
4. Select r and c depending on q.

q	r	c
$0 \leq q < \tan(\pi/8)$	$(q - a)/(1 + qa)$	$\pi/16$
$\tan(\pi/8) \leq q < 1)$	$(q - b)/(1 + qb)$	$3\pi/16$
$1 \leq q < \sqrt{2}$	$(1 - qb)/(q + b)$	$5\pi/16$
$\sqrt{2} \leq q < \infty$	$(1 - qa)/(q + a)$	$7\pi/16$

5. Compute $\arctan(r)$ using the polynomial of degree 27 obtained by economizing the power series of degree 39.
6. Add c to the result to obtain $\arctan(q)$.
7. Insert the result sign, which is the logical difference of the signs of y and x.

DCOS AND DCOSV

1. Compute the integer n nearest to $x \cdot 2/\pi$.

2. Compute the reduced argument $y = x - n \cdot \pi/2$ which lies in the interval $(-\pi/4, \pi/4)$.

3. Define $a = 1$ if computing cosine; else $a = 0$.

4. Compute $\cos(y)$ if $n + a$ is odd and $\sin(y)$ if $n + a$ is even. Use the polynomial obtained by economizing the first 13 terms of the Taylor series to 11 terms.

5. Insert the result sign, which is the sign of $n + a - 1$.

DCOSH AND DCOSHV

Compute using the identity

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2}$$

DCOT AND DCOTV

Compute using the identity

$$\cot(x) = \cos(x)/\sin(x)$$

DEXP AND DEXPV

1. Compute the integer n nearest to $x\log_2 e$.

2. Compute the reduced argument $w = x - n\ln 2$.

3. Compute e^{x} using the polynomial of degree 17 obtained by economizing the power series of degree 20.

4. Add n to the exponent to obtain the final result.

DLOG AND DLOGV

1. Express x as $2^n f$ where $1/\sqrt{2} \leq f < \sqrt{2}$. Then

$$\ln(x) = \ln(2^n f) = \log_2 2^n \cdot \ln 2 + \ln(f) = n\ln 2 + \ln(f).$$

2. For a first approximation a_0 to $\ln(f)$, use a polynomial of degree 7 in $z = f - 1/f + 1$.

$$a_0 = c_1 z + c_3 z^3 + c_5 z^5 + c_7 z^7.$$

3. Apply the Newton-Raphson iteration

$$a_{n+1} = a_n - \frac{q(a_n)}{q'(a_n)} = a_n - \frac{e^{a_n} - f}{e^{a_n}} = a_n - (1 - fe^{-a_n})$$

two times to compute $\ln(f)$. a_2 is computed directly from a_0 using the formula

$$a_2 = [(a_0 - t_u) - t_l] - t_u{}^2 (\frac{1}{2} + \frac{t_u}{3}).$$

The formula is derived by neglecting terms that are insignificant.

Let $r = fe^{-a_0}$ and $t = 1 - r = t_u + t_l$ where t_u is the most significant part of t and t_l is the least significant. Then

$$a_2 = a_1 - (1 - fe^{-a_1}) = a_1 - [1 - fe^{-(a_0 - t_u)}]$$
$$= a_1 - (1 - fe^{-a_0} e^{t_u}) = a_1 - (1 - re^{t_u}) = a_0 - t_u - (1 - re^{t_u})$$

ignoring the least significant part of t.

Expanding e^{t_u} as a power series truncated to four terms, we have

$$a_0 - t_u - (1 - re^{t_u}) \approx a_0 - t_u - [1 - r(1 + t_u + \frac{t_u{}^2}{2!} + \frac{t_u{}^3}{3!})]$$
$$\approx a_0 - t_u - (1 - r - rt_u - \frac{rt_u{}^2}{2} - \frac{rt_u{}^3}{6})$$
$$\approx a_0 - t_u - [t_u + t_l - (t_u - t_u{}^2) - (\frac{t_u{}^2}{2} - \frac{t_u{}^3}{2}) - \frac{t_u{}^3}{6}]$$
$$\approx a_0 - t_u - t_l - \frac{t_u{}^2}{2} - \frac{t_u{}^3}{3}$$
$$\approx (a_0 - t) - t_u{}^2 (\frac{1}{2} + \frac{t_u}{3})$$

Note that in the derivation above, some terms have been ignored.

$$1 - r = t_u + t_l == \quad r = 1 - t_u - t_l$$

$$rt_u = (1 - t_u - t_l)t_u = t_u - t_u^2 - \cancel{t_l t_u} \quad t_u - t_u^2$$

$$\frac{rt_u^2}{2} = \frac{1 - t_u - t_l}{2} t_u^2 = \frac{t_u^2}{2} - \frac{t_u^3}{2} - \cancel{\frac{t_u^2 t_l}{2}} \quad \frac{t_u^2}{2} - \frac{t_u^3}{2}$$

$$\frac{rt_u^3}{6} = \frac{1 - t_u - t_1}{6} t_u^3 = \frac{t_u^3}{6} - \cancel{\frac{t_u^4}{6}} - \cancel{\frac{t_u^3 t_l}{6}} \quad \frac{t_u^3}{6}$$

4. Multiply the exponent n by ln 2 and add to the result.

5. If the common logarithm is desired, multiply by log e.

DMOD AND DMODV

1. Divide x by y and extract the integer part n.

2. Return with $x - ny$ as the result.

DSINH AND DSINHV

Compute using the identity

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

DSQRT AND DSQRTV

1. Call SQRT% to compute an initial approximation, y_0.

2. Perform one Newton-Raphson iteration to yield the double-precision result.

$$y_1 = \frac{1}{2}(y_0 + \frac{x}{y_0})$$

DTAN AND DTANV

Compute using the identity

$$\tan(x) = \sin(x)/\cos(x)$$

DTANH AND DTANHV

Compute using the identity

$$\tanh(x) = \sinh(x)/\cosh(x)$$

For $x = 0$, return $(\tanh(0) = 0)$
For $x < 0$, $\tanh(x) = (e^{2x} - 1.)/(e^{2x} + 1.)$
For $x > 0$, $\tanh(x) = (1.-e^{-2x})/(e^{-2x} + 1.)$

DTODSS, DTODSV, DTODVS, AND DTODVV

1. Return 0 if base = 0 and exponent > 0.
2. Call DLOG% to compute logarithm of base.
3. Call DMSS% to multiply logarithm of base by exponent.
4. Call DEXP% to compute final result.

DTOISS, DTOISV, DTOIVS, AND DTOIVV

1. See the algorithm for $RTOISS, $RTOISV, $RTOIVS, and $RTOIVV.
2. The series of multiplies is done by successive calls to DMSS%.
3. If base = 0, exponent must be > 0.

EXP AND EXPV

1. Multiply x by 2/ln 2 and write

$$y = x \cdot 2/\ln 2 = 2n_1 + n_2 + f$$

where n_1 and n_2 are integers, $0 \leq n_2 < 2$ and $0 \leq f < 1$. Observe that

$$e^x = e^{(x/\ln 2)(\ln 2)} = (e^{\ln 2})^{x/\ln 2} = 2^{x/\ln 2}$$

$$= 2^{(n_1 + n_2/2 + f/2)} = 2^{n_1}2^{n_2/2}2^{f/2}.$$

2. Extract the integer and fractional parts of y. The least significant bit of the integer part is n_1 and the remaining bits are n_2. The fractional part is f.

3. Compute $2^{f/2}$ using a rational function of the form

$$\frac{Q(x^2) + xP(x^2)}{Q(x^2) - xP(x^2)}$$

where P and Q are polynomials of degree 2 in x^2. Let $x = f/2$.

4. Multiply $2^{f/2}$ by $\sqrt{2}$ if n_2 is 1.

5. Add n_1 to the exponent of the result.

ITOISS

1. If base $i = 2$ and exponent $j \geq 0$, calculate result $t = 2^j$ and return. Otherwise, calculate ABS(i) as the initial product.

2. Using ABS(j) as an index, read the section of the multiplication tree that gives the path of the minimum number of multiplications necessary for this exponent.

3. Calculate intermediate products until the path is exhausted.

4. If $i < 0$ and j(modulo 2) $\neq 0$, negate final product to obtain the result. Otherwise, return the final product as the result.

ITOIVS, ITOIVV

1. See the algorithm for RTOISS, RTOIVS, RTOIVV.

2. The series of multiplies are done as 64-bit integer multiplication.

3. If base $i = 0$, exponent j must be greater than 0.

RANF

Let $<S(n)>$ be the linear congruential sequence produced by setting

$$S_{n+1} = mS_n \bmod 2^{48},\ n \geq 0.$$

Select a suitable multiplier m and starting seed S_0. Use S_n, $n > 0$, as the coefficient for the n^{th} random number. Use 40000 as the biased octal exponent and normalize the result to obtain random numbers in the range (0,1). Since $(i \bmod N)(j \bmod N) = ij \bmod N$ for integers i,j,N, the result is

$$S_{n+64} = m^{64}S_n \bmod 2^{48}$$

enabling computation of 64 seeds at a time using vector instructions and 64 initial values S_i, $1 \leq i \leq 64$. Care must be taken to allow intermixing of scalar and vector RANF calls. The algorithm follows.

1. If only 64 random seeds remain in the 128-word buffer, go to step 4. Otherwise, read the next seed and increment the buffer index.

2. Use the seed as the coefficient and 40000_8 as the biased exponent to create the unnormalized random number.

3. Normalize the random number and return.

4. Move the 64 remaining seeds from the second half of the buffer to the first half.

5. Apply the multiplicative congruential method to generate the next 64 seeds and store them in the second half of the buffer.

6. Reset the buffer index to point to the second seed in the buffer.

7. Use the first seed in the buffer as the coefficient and use 40000_8 as the biased exponent to create the unnormalized random number.

8. Normalize the random number and return.

RANFV

Refer to the description for RANF for additional detail.

1. Read the next VL seeds from the buffer.

2. Use the seeds as the coefficients and use 40000_8 as the unbiased exponent to create the unnormalized random numbers.

3. Normalize the random numbers.

4. Compute VL and increment the buffer index by that amount.

5. Return if 64 or more seeds remain in the buffer.

6. Move the 64 newest seeds from the second half of the buffer to the first half and decrement the buffer index by 64.

7. Apply the multiplicative congruential method to generate the next 64 seeds and store them in the second half of the buffer.

8. Restore VL to its original value and return.

RANGET

Return the current seed as a 64-bit integer. The random number generator can be reset at some later time to the current state by calling $RANSET with the 64-bit integer as an argument.

RANSET

1. Use the default seed if the supplied seed is 0. Otherwise, take the seed modulo 2^{48} and make it odd to prevent zero propagation.

2. Apply the multiplicative congruential method to generate the first 128 seeds to fill the buffer.

3. Clear the buffer index and return.

RTOISS, RTOIVS, AND RTOIVV

1. If exponent $e < 0$, compute base $z = 1/z$.

2. Compute the absolute value of the exponent e. Write $|e|$ as

   $$e_n 2^n + e_{n-1} 2^{n-1} + \ldots e_1 2 + e_0$$

   where e_i is either 0 or 1.

3. Form $\underset{e_i=1}{\pi} \quad z^i$

RTORSS

1. Return 0 if the base is 0 and the exponent is greater than 0.
2. Return 1 if the base is nonzero but the exponent is 0.
3. Call ALOG% to compute the logarithm of the base.
4. Multiply by the exponent.
5. Call EXP% to compute the final result and return.

RTORVS

1. Return 0 if the base is 0 and the exponent is greater than 0.
2. Return 1 if the base is nonzero but the exponent is 0.
3. Call %ALOG% to compute the logarithm of the base.
4. Multiply by the exponent.
5. Call %EXP% to compute the final result.

RTORVV

1. Call %ALOG% to compute the logarithm of the base.
2. Multiply by the exponent.
3. Call %EXP% to compute the final result. Return 0 for any zero base if the exponent is greater than 0.

SQRT AND SQRTV

This algorithm was designed and implemented by Harry K. Nelson, Lawrence Livermore Laboratories.

1. If $x = 0$, return $\sqrt{x} = 0$.

2. Compute an initial approximation $y_0/16$ accurate to 17% as a function of $x/2048$.

3. Using a half-precision divide approximation, compute three Newton-Raphson iterations of the form

$$\frac{y_n}{2^{4-n}} = \left(\frac{y_{n} - 1}{2^{4-(n-1)}} + \left(\frac{2^{4-(n-1)}}{y_{n-1}}\right) \frac{x}{2^{4-(n-1)} \cdot 2^{4-(n-1)}}\right)$$

4. Using a full-precision divide approximation, compute a final Newton-Raphson iteration yielding

$$\sqrt{x} \approx y_4 = \frac{y_3}{2} + \left(\frac{2}{y_3}\right) \frac{x}{4}$$

TANH AND TANHV

1. If $x \geq 17.3287$, return with $\tanh(x) = \pm 1$ where the sign of the result is the sign of x.

2. If $|x| < 0.12$, approximate $\tanh(x)$ using the first six terms of the power series.

3. For all other values of x, call EXP% to compute e^{2x}.

4. Return with $\tanh(x) = 1 - 2/1 + e^{2x}$.

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of Mathematical Functions*. National Bureau of Standards, Washington, D.C. Library of Congress catalog card number 64-60036.

Control Data Corporation. (1975). *FORTRAN Common Library Mathematical Routines*. Control Data Corp., Bloomington, Mn. Publication number 60387400.

Dongarra, J. J., et al. (1979). *LINPACK User's Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia. Library of Congress catalog card number 78-78206.

Garbow, B. S., et al. (1977). *Matrix Eigensystem Routines - EISPACK Guide Extension*. Springer-Verlog, New York, N.Y. Library of Congress catalog card number 7702802.

Gentalman, W. M. (December 1973). "Least Squares Computations by Givens Transformations Without Square Roots." *J. Inst. Math. Appl.*

Hart, John F., et al. (1968). *Computer Approximations*. John Wiley & Sons, New York, N.Y. Library of Congress catalog card number 67-23326.

IBM Corporation. (1968). *IBM System/360 FORTRAN IV Library Subprograms*. IBM Corp., New York, N.Y. File number S360-25 GC28-6596-4.

Knuth, D. E. (1969). *The Art of Computer Programming, Volume 2 Seminumerical Algorithms*. Addison-Wesley, Reading, Mass. Library of Congress catalog card number 67-26020.

Lawson, C. L., et al. (1979). "Basic Linear Algebra Subprograms for FORTRAN Usage." *Transactions on Mathematical Software*. Sandia Laboratory Report SAND77-0898.

Ralston, Anthony. (1965). *A First Course in Numerical Analysis*. McGraw-Hill, New York, N.Y. Library of Congress catalog card number 64-7940.

Smith, B. T., et al. (1976). *Matrix Eigensystem Routines - Eispack Guide,* second edition. Springer-Verlog, New York, N.Y. Library of Congress catalog card number 76-2662.

Stockmal, F. (January, 1964). "Calculations with Pseudo-Random Numbers." *Journal of the ACM, Volume 11, No. 1,* pp 41-52.

PERFORMANCE STATISTICS B

This section contains accuracy and timing statistics for the single-precision, single-argument routines listed in section 3, Common Mathematical Subprograms, and for $SCILIB routines listed in section 4, Scientific Applications Subprograms.

The figures listed in table B-1 are for a CRAY-1 S Series Computer System with 8-bank memory. The CRAY-1 S computer with 16-bank memory does not typically show a significant difference in performance; however, for these routines, the performance is three to seven percent faster.

COMMON MATHEMATICAL SUBPROGRAMS

These statistics are arranged alphabetically according to entry names of the subprograms. Listed in table B-1 are the following column headings:

Entry name - the entry name that must be used to call the subprogram

Domain - the domain used to obtain the accuracy figures. For each function, accuracy figures are given for one or more representative segments within the valid domain. In each case, the figures given are the most meaningful to the function and domain under consideration.

Ten thousand arguments are selected for each domain. The arguments are uniformly distributed unless otherwise noted.

Accuracy - accuracy figures (maximum relative error and standard deviation) for one or more representative segments within the valid domain. The accuracy figures supplied are based on the assumption that the arguments are perfect (that is, without error and, therefore, having no error propagation effect upon the answers). The only error in the answers are those introduced by the subroutines.

Average result rate (usec/result) - timing statistics. Values represent the average result rate in microseconds for scalar computations and for vector computations using vector lengths of 1, 2, 4, 10, 32, and 64 elements. All timing information was gathered with interrupts disabled.

Table B-1. Statistics for single-precision, single-argument subprograms

		Accuracy		Average Result Rate (π Secs/Result)						
		Max. Error ($x10^{-14}$)	Stdrd Dev ($x10^{-15}$)	Scalar	Vector Length					
Entry	Domain				1	2	4	10	32	64
ACOS% %ACOS%	(-1, 1)	2.910	4.233	2.2	3.8	2.0	1.0	0.5	0.3	0.2
	(0, 1)	2.134	3.388	2.2	3.9	2.0	1.0	0.5	0.3	0.2
	(0, 1/2)	0.944	2.695	2.0	3.5	1.8	0.9	0.4	0.2	0.2
	(1/2, sin π/3)	1.064	3.150	2.2	3.9	2.0	1.0	0.5	0.3	0.2
	(sin π/3, sin5 π/12)	2.469	5.635	2.5	4.5	2.3	1.2	0.6	0.3	0.3
	(sin5 π/12, 1)	2.134	4.564	4.2	7.3	3.7	2.0	0.9	0.5	0.4
ALOG%	(1/2, 2)	1.639	3.886	2.0	3.0	1.5	0.8	0.4	0.3	0.2
%ALOG%	(0, ∞)†	1.562	4.282	2.0	2.9	1.5	0.8	0.4	0.3	0.2
ALOG10%	(1/2, 2)	1.500	4.077	1.9	3.2	1.7	0.9	0.4	0.3	0.2
%ALOG10%	(0, ∞)†	1.786	3.924	1.9	3.2	1.6	0.9	0.4	0.3	0.2
ASIN% %ASIN%	(-1, 1)	1.765	3.888	2.2	3.7	1.9	1.0	0.5	0.3	0.2
	(0, 1)	1.655	3.874	2.0	3.7	1.9	1.0	0.5	0.3	0.2
	(0, 1/2)	1.985	4.008	1.8	3.3	1.7	0.9	0.4	0.2	0.2
	(1/2, sin π/3)	1.189	3.093	2.0	3.8	1.9	1.0	0.5	0.3	0.2
	(sin π/3, sin5 π/12)	1.349	5.594	2.2	4.3	2.2	1.2	0.6	0.3	0.3
	(sin5 π/12, 1)	0.578	2.152	3.9	6.9	3.5	1.9	0.9	0.5	0.4
ATAN% %ATAN%	(- π/2, π/2)	1.057	3.549	1.7	4.1	2.1	1.1	0.6	0.3	0.2
COS% %COS%	(- π/4, π/4)	0.474	1.656	1.9	4.0	2.1	1.1	0.6	0.3	0.3
	(- π, π)	1.010	2.312	2.1	4.0	2.1	1.1	0.6	0.3	0.3
	(-100, 100)	50.043	16.824	2.1	4.0	2.1	1.1	0.6	0.3	0.3
COSH%	(-0.301, 0.301)	0.712	3.177	2.1	3.6	1.8	1.0	0.5	0.3	0.3
%COSH%	(-5677, 5677)	1.319	3.980	2.1	3.6	1.8	1.0	0.5	0.3	0.3
COT%	(- π/4, π/4)	1.835	5.058	2.3	3.4	1.7	0.9	0.5	0.3	0.2
%COT%	(-100, 100)	60.847	24.833	2.3	3.4	1.7	0.9	0.5	0.3	0.2

† Samples are exponentially distributed.

Table B-1. Statistics for single-precision, single-argument subprograms (continued)

		Accuracy		Average Result Rate (π Secs/Result)						
		Max.	Stdrd		Vector Length					
Entry	Domain	Error ($\times10^{-14}$)	Dev ($\times10^{-15}$)	Scalar	1	2	4	10	32	64
EXP%	(-1, 1)	2.170	7.482	1.6	2.5	1.3	0.7	0.3	0.2	0.2
%EXP%	(-5677, 5677)	2.045	7.408	1.6	2.5	1.3	0.7	0.3	0.2	0.2
SIN%	$(-\pi/4, \pi/4)$	0.698	2.244	1.8	3.9	2.0	1.1	0.5	0.3	0.3
%SIN%	$(-\pi, \pi)$	0.927	2.108	1.9	3.8	2.0	1.1	0.5	0.3	0.3
	(-100, 100)	20.809	8.512	2.0	3.8	2.0	1.1	0.5	0.3	0.3
SINH%	(-0.301, 0.301)	0.686	2.273	2.3	4.0	2.1	1.1	0.6	0.4	0.3
%SINH%	(-5677, 5677)	1.032	3.974	2.2	4.0	2.1	1.1	0.6	0.4	0.3
SQRT%	(0, 1)	0.767	1.931	1.5	2.3	1.2	0.6	0.3	0.1	0.1
%SQRT%	$(0, \infty)$†	0.724	2.099	1.5	2.3	1.2	0.6	0.3	0.1	0.1
TAN%	$(-\pi/4, \pi/4)$	1.444	4.274	1.8	3.4	1.8	0.9	0.5	0.3	0.2
%TAN%	(-100, 100)	60.456	24.830	2.1	3.4	1.8	0.9	0.5	0.3	0.2
TANH%	(-20, 20)	5.671	5.822	2.8	4.6	2.4	1.2	0.6	0.3	0.3
%TANH%	(-1/2, 1/2)	8.127	13.481	2.7	4.6	2.3	1.2	0.6	0.3	0.3

† Samples are exponentially distributed.

SCIENTIFIC APPLICATIONS SUBPROGRAMS

This section presents timings of selected **$SCILIB** routines and some comparisions between $SCILIB routines and FORTRAN DO-loops performing the same functions. Table B-2 gives timings and statistics for ten popular DO-loops and their $SCILIB equivalent subroutines. The FORTRAN DO-loops are compiled with CFT version 1.10. The $SCILIB version is 1.11. The timings in the tables are generated on a CRAY-1 S Series 16-bank machine.

The timings for these tables are generated by a routine in SCILBPL called TIMINGS. The following job produces timings for various machines with the default compiler and $SCILIB.

```
    JOB,JN=TIMINGS.
    ACCOUNT, AC=myown.
    CFT.
    LDR.
    /EOF
              CALL TIMINGS
              STOP
              END
```

The output appears in $OUT.

The following job produces timings for various versions of CFT or $SCILIB.

```
    JOB,JN=TIMINGS.
    ACCOUNT,AC=myown.
    ACCESS,DN=$SCILIB,ID=myown.
    ACCESS,DN=CFT,ID=myown.
    ACCESS,DN=SCILBPL,ID=myown.
    UPDATE,P=SCILIBPL,I=0,Q=TIMINGS.
    CFT,I=$CPL.
    CFT.
    LDR.
    /EOF
              CALL TIMINGS
              STOP
              END
```

Subroutine name - name of the $SCILIB routine and a brief description of its function

Loop length - the number of passes through the FORTRAN DO-loop

CFT time - seconds used computing the results with CFT version 1.10

$SCILIB time - seconds used computing the same result with a call to a $SCILIB subroutine

$SCILIB/CFT - the ration of $SCILIB time to CFT time

Clocks per operation - number of 12.5 nanosecond clocks per floating-point operation or per loop length if no floating-point operations are done

Two points are evident from the column of ratios between CFT DO-loop and $SCILIB subroutines.

1. The overhead of a subroutine call dominates the execution time for short loop lengths.

2. For long loop lengths, $SCILIB versions can be faster than CFT versions by a factor of 2 or 3.

Table B-3 presents timings and MFLOP rates for $SCILIB versions of SGEFA and SGESL of LINPACK and ELMHES and HQR of EISPACK.

Subroutine name and function - name of $SCILIB routine and a brief description

Dimension of matrix - the size of the matrix problem

Execution time - second used to compute the result

MFLOP rate - the approximate number of millions of floating-point operations per second

Table B-2. $SCILIB timings and comparisons

Subroutine Name (Function)	Loop Length	CFT Time ($x10^{-6}$ sec)	$SCILIB Time ($x10^{-6}$ sec)	$SCILIB/ CFT (Ratio)	Clocks Per Operation
FOLR (first order linear recurrence)	1	2.5	3.9	1.58	315.
	2	3.1	4.1	1.32	165.
	3	3.7	4.3	1.18	87.
	4	4.3	4.6	1.08	61.33
	5	4.8	4.8	1.00	48.00
	10	7.6	5.9	.78	26.33
	25	16.	9.3	.58	15.50
	50	30.	15.	.50	12.18
	100	58.	26.	.45	10.59
	250	140.	60.	.42	9.63
GATHER A(I)=B(INDEX(I))	1	3.0	4.6	1.57	371.
	2	3.4	4.8	1.43	193.
	3	3.8	5.0	1.32	133.67
	4	4.2	5.1	1.22	102.75
	5	4.6	4.9	1.07	79.2
	10	6.7	5.6	.84	44.7
	25	13.	7.5	.58	24.
	50	23.	11.	.46	17.10
	100	44.	17.	.39	13.65
	250	110.	36.	.34	11.58
ISAMIN (finds the first position of the minimum absolute value of a vector)	1	2.5	6.2	2.48	493.
	2	3.4	6.2	1.83	494.
	3	4.2	6.2	1.46	247.50
	4	5.1	6.2	1.22	165.33
	5	6.0	6.2	1.04	124.25
	10	10.	7.1	.69	63.11
	25	23.	9.0	.39	30.08
	50	45.	11.0	.25	18.04
	100	88.	51.	.58	41.55
	250	220.	70.	.32	22.33

Table B-2. $SCILIB timings and comparisons (continued)

Subroutine Name (Function)	Loop Length	CFT Time ($x10^{-6}$ sec)	$SCILIB Time ($x10^{-6}$ sec)	$SCILIB/ CFT (Ratio)	Clocks Per Operation
ISRCHEQ	1	4.3	5.5	1.27	438.00
(searches a	2	4.7	5.5	1.55	219.49
vector for	3	5.2	5.5	1.06	146.67
a word	4	5.6	5.5	0.97	110.25
match)	5	6.1	5.5	0.90	88.40
	10	8.3	5.6	.67	44.70
	25	15.	5.8	.38	18.48
	50	28.	6.1	.21	9.74
	100	50.	7.0	.14	5.59
	250	110.	9.1	.07	2.90
MXM M	1	6.6	5.5	.84	443.
(full matrix	2	24.	6.5	.46	57.44
multiply)	3	25.	8.0	.32	18.37
	4	40.	9.2	.23	8.07
	5	59.	12.	.20	5.04
	10	220.	30.	.13	1.38
	25	1700.	260.	.16	.71
	50	8900.	1800.	.20	.59
	100	59000.	14000.	.24	.57
	250	770000.	210000.	.27	.54
MXM V	1	5.8	5.6	.97	447.
(matrix	2	7.6	6.4	.85	102.2
times a vextor)	3	9.3	6.5	.70	40.
	4	11.0	7.0	.63	22.44
	5	13.0	7.3	.56	12.4
	10	23.0	9.8	.42	4.35
	25	66.	24.	.37	1.62
	50	180.	76.	.43	1.24
	100	580.	280.	.49	1.14
	250	3100.	1700.	.55	1.09
SAXPY	1	3.3	5.3	1.60	212.5
(add a scalar	2	3.3	5.3	1.60	106.5
multiple of	3	3.3	5.3	1.59	71.17
one vector	4	3.4	5.3	1.59	53.50
to another)	5	3.4	5.4	1.59	42.90

Table B-2. $SCILIB timings and comparisons (continued)

Subroutine Name (Function)	Loop Length	CFT Time ($x10^{-6}$ sec)	$SCILIB Time ($x10^{-6}$ sec)	$SCILIB/ CFT (Ratio)	Clocks Per Operation
SAXPY (continued)	10	3.5	5.5	1.58	21.95
	25	4.0	6.0	1.50	9.66
	50	5.0	7.0	1.41	5.58
	100	7.3	9.3	1.28	3.72
	250	14.0	16.	1.15	2.52
SCATTER A(INDEX(I))=B(I)	1	2.7	6.0	2.24	482.00
	2	3.0	6.0	2.00	241.5
	3	3.0	6.1	1.81	161.33
	4	3.7	6.1	1.66	121.25
	5	4.0	6.1	1.52	97.20
	10	5.6	6.9	1.23	55.20
	25	10.	8.8	.83	28.00
	50	19.	12.	.63	18.66
	100	35.	19.	.53	14.90
	250	84.	39.	.46	12.42
SDOT (computes the dot product of 2 real vectors)	1	2.9	5.5	1.89	439.
	2	7.1	5.5	.77	146.33
	3	7.1	5.5	.77	87.80
	4	7.2	5.5	.77	62.71
	5	7.2	5.5	.76	48.78
	10	7.3	5.6	.76	23.63
	25	8.2	6.0	.73	9.78
	50	9.8	6.6	.68	5.34
	100	13.	7.8	.59	3.15
	250	23.	12.	.51	1.93
SSUM (sums the elements of a real vector)	1	2.6	3.9	1.52	314.
	2	6.6	4.1	.62	326.
	3	6.6	4.2	.64	169.
	4	6.6	4.4	.66	116.
	5	6.6	4.5	.68	90.5
	10	6.8	4.8	.71	42.89
	25	7.2	5.0	.69	16.71
	50	8.2	5.3	.65	8.69
	100	10.	7.0	.67	5.66
	250	17.	9.1	.55	2.92

Table B-3. $SCILIB timings and MFLOP rates

Subroutine Name (Function)	Dimension of matrix	Execution time (seconds)	MFLOP rate
ELMHES	5	.000057	3.63
(Reduction of	10	.00026	6.32
full matrix	25	.0020	13.30
to upper	50	.010	20.23
Hessenberg form)	100	.064	25.83
	250	.83	31.42
HQR	5	.00053	2.25
(Reduction form	10	.0022	4.70
upper Hessenberg	25	.011	11.41
form to upper	50	.043	21.05
traingular)	100	.18	36.02
	250	1.4	62.62
SGEFA	5	.000062	1.35
(L-U decomposition)	10	.00019	3.48
	25	.0011	9.62
	50	.0048	17.41
	100	.026	25.51
	250	.30	35.00
SGESL	5	.000049	1.73
(Backward and	10	.000056	3.59
forward solving)	25	.00015	8.58
	50	.00033	15.01
	100	.0089	22.01
	250	.039	32.58

SORT ENTRY POINTS

C

SORT consists of library subroutines accessed through FORTRAN calls. The user-callable entry points are as follows:

- SAMSORT
- SAMFILE
- SAMKEY
- SAMGO
- SAMEQU
- SAMOPT
- SAMSEQ
- SAMSIZE
- SAMTUNE

Following are the entry points within the CRI sort package. The code is proprietary and must not be used for any purpose other than as part of the CRI sort package.

Module PSRV8IO entry points

- V8PUT
- V8GET
- V8WAIT
- V8POS
- V8CLOSE
- V8OPEN
- V8STATS
- V8LNTHDS

Module IOXOXOI entry points

- P87DNT
- P87DOD
- P87CLS
- P87DLT
- P87WDC
- P87RDC
- P87LBN
- P87OPN
- P87WAIT
- P87SNSE

For more details, see the SORT Reference Manual, CRI publication SR-0074.

INDEX

INDEX

READERS COMMENT FORM

Library Reference Manual SR-0014 I

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ________________________________

JOB TITLE ________________________________

FIRM ________________________________

ADDRESS ________________________________

CITY ________________ STATE ________ ZIP ________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

READERS COMMENT FORM

Library Reference Manual SR-0014 I

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ______________________________

JOB TITLE ______________________________

FIRM ______________________________

ADDRESS ______________________________

CITY ____________ STATE ______ ZIP ______

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS