CRAY X-MP AND CRAY-1®
COMPUTER SYSTEMS

LIBRARY
REFERENCE MANUAL
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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>April, 1977 - First edition.</td>
</tr>
<tr>
<td>B</td>
<td>August, 1977 - Reprint with revision. This printing obsoletes the first edition.</td>
</tr>
<tr>
<td>C</td>
<td>June, 1978 - Reprint with revision. The addition of Section 4, Performance Statistics comprises the major change to this manual.</td>
</tr>
<tr>
<td>C-01</td>
<td>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.</td>
</tr>
<tr>
<td>D</td>
<td>January, 1980 - Technical corrections to pages 4-17 and 4-18.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

The publication number has been changed from 2240014 to SR-0014.
April, 1980 - This change packet clarifies the formula for the OPFILT routine and corrects the formulas and one parameter for the Fourier transform routines.

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.

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, WBUFFER.

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.

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.

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

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.
This publication describes the subprograms available to users of the CRAY-I 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

PREFACE .............................................. V

1. INTRODUCTION ...................................... 1-1
   SUBPROGRAM CLASSIFICATION ........................ 1-1
   CFT LINKAGE METHODS ................................ 1-2
   CFT linkage macros .................................. 1-2
   CONVENTIONS ........................................ 1-3

2. SUBPROGRAM SUMMARY ............................... 2-1
   INTRODUCTION ....................................... 2-1
   TABLE DESCRIPTION .................................. 2-1
     Primary reference name ............................ 2-2
     Page number ...................................... 2-2
     UPDATE deck name ................................ 2-2
     Entry type ........................................ 2-2
     CFT call type .................................... 2-3
     Library ............................................ 2-3
     OS dependency .................................... 2-3
     Purpose ............................................ 2-3
     Pascal subprograms ................................ 2-3

3. COMMON MATHEMATICAL SUBPROGRAMS ................ 3-1
   INTRODUCTION ....................................... 3-1
   LOGARITHMIC ROUTINES .............................. 3-5
   EXPONENTIAL ROUTINES .............................. 3-6
   SQUARE ROOT ROUTINES .............................. 3-7
   TRIGONOMETRIC ROUTINES ............................ 3-8
   HYPERBOLIC ROUTINES ................................ 3-11
   BOOLEAN ARITHMETIC ROUTINES ........................ 3-13
   BASE VALUE RAISED TO A POWER ROUTINES ........... 3-15
   DOUBLE-PRECISION ARITHMETIC ROUTINES ............. 3-17
   TRIPLE-PRECISION ARITHMETIC ROUTINES ............. 3-18
   SIXTY-FOUR-BIT INTEGER DIVISION ................... 3-21
   CHARACTER FUNCTIONS ................................ 3-22
   CHARACTER CONCATENATION AND STORE ROUTINES ....... 3-24
     Initialization .................................... 3-24
     Transfer ......................................... 3-24
     Termination ...................................... 3-25
COMMON MATHEMATICAL SUBPROGRAMS (continued)

ASCII CONVERSION FUNCTIONS ........................................... 3-25
PSEUDO VECTORIZATION ROUTINES ........................................ 3-28
MISCELLANEOUS MATH ROUTINES ........................................... 3-28
RANDOM NUMBER ROUTINES ............................................... 3-36
MATH TABLES ................................................................. 3-38

4. SCIENTIFIC APPLICATIONS SUBPROGRAMS ................................. 4-1

INTRODUCTION ..................................................................... 4-1

BASIC LINEAR ALGEBRA SUBPROGRAMS .................................... 4-1
  Index of element having maximum absolute value .................. 4-4
  Sum of the absolute values ............................................. 4-5
  Constant times a vector plus another vector ....................... 4-6
  Copy one array into another ........................................... 4-8
  Compute an inner product of two vectors .............................. 4-9
  Euclidean norm of an array ($l_2$ norm) ............................. 4-10
  Construct Givens plane rotation ....................................... 4-11
  Apply Givens plane rotation ........................................... 4-13
  Construct modified Givens plane rotation ......................... 4-13
  Apply modified Givens plane rotation .............................. 4-19
  Scale array ...................................................................... 4-21
  Swap two arrays ............................................................. 4-22

OTHER LINEAR ALGEBRA SUBPROGRAMS .................................. 4-23
  Sparse matrix primitives .................................................. 4-24
  Index of element with maximum or minimum value .................. 4-25
  Index of element having minimum absolute value ................... 4-26
  Sum of the values ........................................................... 4-27
  Compute complex Givens plane rotation ............................... 4-28
  Construct complex Givens plane rotation ......................... 4-28
  Cray machine constants .................................................. 4-29

FUNCTIONS AND LINEAR RECURRENCE SUBROUTINES ...................... 4-30

SINGLE-PRECISION REAL AND COMPLEX LINPACK ROUTINES .......... 4-38

SINGLE-PRECISION EISPACK ROUTINES ................................ 4-38

MATRIX INVERSE AND MULTIPLICATION ROUTINES ................... 4-46

FAST FOURIER TRANSFORM ROUTINES ................................ 4-50

FILTER SUBROUTINES ...................................................... 4-53

GATHER, SCATTER ROUTINES .............................................. 4-56

SEARCH ROUTINES .......................................................... 4-57
  Number or sum of values within or before an element .......... 4-58
  Searching for an object in a vector ................................. 4-59
  Indexed array of all positions of an object in a vector ....... 4-64

SEARCH ORDERED ARRAY FOR TARGET .................................... 4-71

SORT ROUTINE ............................................................... 4-73
  Method ......................................................................... 4-75
  Large radix sorting ..................................................... 4-75
  Multipass sorting ........................................................ 4-76
## 5. **INPUT/OUTPUT SUBPROGRAMS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>5-1</td>
</tr>
<tr>
<td><strong>FORTRAN I/O ROUTINES</strong></td>
<td>5-1</td>
</tr>
<tr>
<td>Initialization routines</td>
<td>5-3</td>
</tr>
<tr>
<td>Input initialization routines</td>
<td>5-3</td>
</tr>
<tr>
<td>Output initialization routines</td>
<td>5-4</td>
</tr>
<tr>
<td>Transfer routines</td>
<td>5-5</td>
</tr>
<tr>
<td>Formatted and unformatted input transfer routines</td>
<td>5-6</td>
</tr>
<tr>
<td>Buffered input transfer routines</td>
<td>5-8</td>
</tr>
<tr>
<td>Namelist input transfer routines</td>
<td>5-8</td>
</tr>
<tr>
<td>Formatted and unformatted output transfer routines</td>
<td>5-11</td>
</tr>
<tr>
<td>Buffered output transfer routines</td>
<td>5-13</td>
</tr>
<tr>
<td>Namelist output transfer routines</td>
<td>5-13</td>
</tr>
<tr>
<td>Finalization routines</td>
<td>5-14</td>
</tr>
<tr>
<td>Input finalization routines</td>
<td>5-14</td>
</tr>
<tr>
<td>Output finalization routines</td>
<td>5-14</td>
</tr>
<tr>
<td><strong>TAPE TRANSLATION ROUTINES</strong></td>
<td>5-15</td>
</tr>
<tr>
<td>Buffer management routines</td>
<td>5-16</td>
</tr>
<tr>
<td>Input buffer management routines</td>
<td>5-16</td>
</tr>
<tr>
<td>Output buffer management routines</td>
<td>5-17</td>
</tr>
<tr>
<td>Record format management routines</td>
<td>5-18</td>
</tr>
<tr>
<td>Input record format management routines</td>
<td>5-18</td>
</tr>
<tr>
<td>Output record format management routines</td>
<td>5-18</td>
</tr>
<tr>
<td>Data format management routines</td>
<td>5-18</td>
</tr>
<tr>
<td>Input data format management routines</td>
<td>5-18</td>
</tr>
<tr>
<td>Output data management format routines</td>
<td>5-19</td>
</tr>
<tr>
<td><strong>EXPLICIT DATA CONVERSION</strong></td>
<td>5-20</td>
</tr>
<tr>
<td>IBM single-precision to Cray single-precision routine</td>
<td>5-20</td>
</tr>
<tr>
<td>IBM double-precision to Cray single-precision routine</td>
<td>5-21</td>
</tr>
<tr>
<td>IBM integer to Cray integer routine</td>
<td>5-21</td>
</tr>
<tr>
<td>EBCDIC to ASCII routine</td>
<td>5-22</td>
</tr>
<tr>
<td>IBM packed decimal field to integer routine</td>
<td>5-23</td>
</tr>
<tr>
<td>IBM logical to Cray logical routine</td>
<td>5-23</td>
</tr>
<tr>
<td>Cray single-precision to IBM single-precision routine</td>
<td>5-24</td>
</tr>
<tr>
<td>Cray single-precision to IBM double-precision routine</td>
<td>5-25</td>
</tr>
<tr>
<td>Cray integer to IBM integer routine</td>
<td>5-26</td>
</tr>
<tr>
<td>ASCII to EBCDIC routine</td>
<td>5-27</td>
</tr>
<tr>
<td>Integer to IBM packed decimal field routine</td>
<td>5-28</td>
</tr>
<tr>
<td>Cray logical to IBM logical routine</td>
<td>5-28</td>
</tr>
<tr>
<td>Unpack 60-bit words routine</td>
<td>5-29</td>
</tr>
<tr>
<td>Pack 60-bit words routine</td>
<td>5-29</td>
</tr>
<tr>
<td>Pack 32-bit words routine</td>
<td>5-30</td>
</tr>
<tr>
<td>Unpack 32-bit words routine</td>
<td>5-30</td>
</tr>
<tr>
<td>CDC integer to Cray integer routine</td>
<td>5-31</td>
</tr>
<tr>
<td>CDC single-precision to Cray single-precision routine</td>
<td>5-31</td>
</tr>
<tr>
<td>CDC display code character to ASCII character routine</td>
<td>5-32</td>
</tr>
<tr>
<td>Cray integer to CDC integer routine</td>
<td>5-32</td>
</tr>
<tr>
<td>Cray single-precision to CDC single-precision routine</td>
<td>5-33</td>
</tr>
<tr>
<td>ASCII character to CDC display code character routine</td>
<td>5-33</td>
</tr>
<tr>
<td><strong>DATASET CONTROL ROUTINES</strong></td>
<td>5-34</td>
</tr>
<tr>
<td>Open dataset routine</td>
<td>5-34</td>
</tr>
</tbody>
</table>
DATASET CONTROL ROUTINES (continued)

Close dataset routine .......................................................... 5-34
Inquire routine ........................................................................... 5-35
Dataset copying routines ............................................................ 5-35
  Copy records ............................................................................. 5-35
  Copy files ............................................................................... 5-37
  Copy dataset ........................................................................... 5-39
  Copy sectors (unblocked) ......................................................... 5-39
Dataset skip routines ................................................................. 5-40
  Skip records ............................................................................ 5-40
  Skip files ............................................................................... 5-40
  Skip dataset ........................................................................... 5-41
  Skip sectors (unblocked) ......................................................... 5-42
Dataset positioning routines ....................................................... 5-42
  Get position of mass storage dataset ..................................... 5-42
  Set position of dataset ............................................................ 5-46
  Backspace one record .............................................................. 5-50
  Backspace one file .................................................................... 5-52
  Rewind dataset ........................................................................ 5-52
  Position dataset ....................................................................... 5-53
  Synchronize tape dataset ......................................................... 5-55
Dataset termination routines ....................................................... 5-55
I/O status routines ...................................................................... 5-57
Auxiliary NAMELIST routines ...................................................... 5-59
LOGICAL RECORD I/O ROUTINES ................................................. 5-63
Read routines .............................................................................. 5-63
  Read words ............................................................................. 5-64
  Read characters ...................................................................... 5-66
  Read IBM words ..................................................................... 5-69
  Read unblocked data .............................................................. 5-69
Write routines .............................................................................. 5-70
  Write words .......................................................................... 5-70
  Write characters ..................................................................... 5-73
  Write IBM words .................................................................... 5-74
  Write unblocked data ............................................................. 5-74
CAL I/O interface routine ............................................................ 5-75
Bad data error recovery routines .................................................. 5-75
Character routines ...................................................................... 5-77
NUMERIC CONVERSION ROUTINES ............................................. 5-78
RANDOM ACCESS DATASET I/O ROUTINES .................................. 5-83
  Record-addressable, random access dataset I/O routines ............ 5-83
  Word-addressable, random access dataset I/O routines ............... 5-103
WORD-ADDRESSABLE I/O AND DATASET CONTROL ROUTINES ....... 5-109

6. DATASET MANAGEMENT SUBPROGRAMS ......................................... 6-1

INTRODUCTION ........................................................................... 6-1
CONTROL STATEMENT TYPE SUBPROGRAMS ............................... 6-1
  Permanent dataset management (PDM) routines ......................... 6-2
  Dataset staging routines .......................................................... 6-2
  Definition and control routines ................................................ 6-3
DATASET SEARCH TYPE SUBPROGRAMS ............................................. 6-4
DATASET INPUT/OUTPUT SUBPROGRAMS .......................................... 6-7
TABLES ...................................................................................... 6-9

7. SPECIAL PURPOSE SUBPROGRAMS ............................................... 7-1

INTRODUCTION ............................................................................. 7-1
DEBUG AID ROUTINES ..................................................................... 7-1
   Flow trace routines .................................................................. 7-2
   Traceback routines ................................................................... 7-4
   Dump routines ......................................................................... 7-5
   Exchange Package processing routines ................................. 7-8
   Array bounds checking routines ...................................... 7-12

TABLE MANAGEMENT ROUTINES ................................................. 7-12
   TM common block ................................................................. 7-14

STACK MANAGEMENT ROUTINES .................................................. 7-21

HEAP MANAGER ROUTINES .......................................................... 7-25
   Allocate routines .................................................................... 7-25
   Deallocate routines ............................................................... 7-27
   Set new length routines ....................................................... 7-27
   Change length and move routines ......................................... 7-29
   Heap block length routines ............................................... 7-30
   Heap shrink routines .......................................................... 7-31
   Heap integrity check routines ........................................... 7-31
   Heap statistics routines ...................................................... 7-32
   Dump heap control word routines ............................... 7-33
   Heap expansion routine ..................................................... 7-34
   Heap memory request routine ........................................... 7-34
   Heap merge routine ........................................................... 7-35

JOB CONTROL ROUTINES .............................................................. 7-35

FLOATING-POINT INTERRUPT ROUTINES ..................................... 7-38
   Floating-point interrupt test .............................................. 7-39
   Temporary floating-point interrupt control ....................... 7-39
   Job floating-point interrupt control .................................. 7-39

BIDIRECTIONAL MEMORY TRANSFER ROUTINES ......................... 7-40
   Bidirectional memory transfer test ...................................... 7-40
   Temporary bidirectional memory transfer control ............... 7-41
   Permanent bidirectional memory transfer control .............. 7-41

TIME AND DATE ROUTINES ............................................................ 7-42

TIMESTAMP ROUTINES .................................................................. 7-44

CONTROL STATEMENT PROCESSING ROUTINES .............................. 7-46
   Control statement cracking routines .................................. 7-46
   Cracked parameter list ....................................................... 7-47
   Get parameter routine ....................................................... 7-48
   Directive cracking routine ............................................... 7-51
   Process parameter list routine ....................................... 7-53
   Crack expression routine .................................................. 7-53

JOB CONTROL LANGUAGE SYMBOL ROUTINES ............................... 7-55
SKOL RUN-TIME SUPPORT ROUTINES .......................... 7-56
  Character-string manipulation routines .................. 7-57
  Character-code translation routines ...................... 7-58
  Error-handling routine ................................ 7-60
ERROR-PROCESSING ROUTINES ................................. 7-60
BYTE AND BIT MANIPULATION ROUTINES ........................ 7-61
  Move bytes routine .................................... 7-61
  Move bits .............................................. 7-61
  Move characters routine ................................. 7-62
  Replace byte routine ................................... 7-63
  Compare bytes function .................................. 7-63
  Search bytes routine .................................... 7-63
  ASCII to integer routine ............................... 7-64
  Integer to ASCII routines ............................. 7-65
  Pack, unpack .......................................... 7-65
MISCELLANEOUS SPECIAL PURPOSE ROUTINES .................. 7-70

8. PASCAL SUBPROGRAMS ...................................... 8-1

  INTRODUCTION ............................................ 8-1
  P$HIDPA ............................................... 8-1
  P$ABORT ............................................. 8-2
  P$BREAK ............................................. 8-2
  P$CALLR ............................................. 8-2
  P$CBV .............................................. 8-3
  P$CONNEC ............................................ 8-3
  P$DATE .............................................. 8-4
  P$DBP .............................................. 8-4
  P$DEBUG ............................................ 8-4
  P$DISP .............................................. 8-4
  P$DIVMOD ........................................... 8-5
  P$ENDP ............................................. 8-5
  P$EOF ............................................... 8-6
  P$EOLN ............................................. 8-6
  P$GET ............................................... 8-6
  P$HALT .............................................. 8-7
  P$JTIME ............................................ 8-7
  P$LOGMSG ........................................... 8-8
  P$LSTREW ............................................ 8-8
  P$MEMRY ............................................. 8-8
  P$MOD ............................................... 8-9
  P$NEW ............................................... 8-9
  P$OSDBS ............................................ 8-10
  P$OSDDT ............................................ 8-10
  P$OSDIP ............................................. 8-11
  P$OSDJT ............................................. 8-11
  P$OSDIM ............................................. 8-11
  P$OSDPR ............................................. 8-12
  P$OSDIQ ............................................. 8-12
  P$OSDRC ............................................. 8-12
  P$OSDRP ............................................. 8-13
8. **PASCAL SUBPROGRAMS** (continued)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>P$OSDREW</td>
<td>8-13</td>
</tr>
<tr>
<td>P$OSDTM</td>
<td>8-14</td>
</tr>
<tr>
<td>P$OSDWC</td>
<td>8-14</td>
</tr>
<tr>
<td>P$OSDFWF</td>
<td>8-15</td>
</tr>
<tr>
<td>P$OSDWR</td>
<td>8-15</td>
</tr>
<tr>
<td>P$OSDXP</td>
<td>8-16</td>
</tr>
<tr>
<td>P$PAGE</td>
<td>8-16</td>
</tr>
<tr>
<td>P$PUT</td>
<td>8-17</td>
</tr>
<tr>
<td>P$RB</td>
<td>8-17</td>
</tr>
<tr>
<td>P$RCH</td>
<td>8-17</td>
</tr>
<tr>
<td>P$READ</td>
<td>8-18</td>
</tr>
<tr>
<td>P$READLN</td>
<td>8-18</td>
</tr>
<tr>
<td>P$REPKV</td>
<td>8-19</td>
</tr>
<tr>
<td>P$RESET</td>
<td>8-19</td>
</tr>
<tr>
<td>P$REWRIT</td>
<td>8-19</td>
</tr>
<tr>
<td>P$RF</td>
<td>8-20</td>
</tr>
<tr>
<td>P$RI</td>
<td>8-20</td>
</tr>
<tr>
<td>P$ROUND</td>
<td>8-21</td>
</tr>
<tr>
<td>P$RSTR</td>
<td>8-21</td>
</tr>
<tr>
<td>P$RTIME</td>
<td>8-21</td>
</tr>
<tr>
<td>P$RTMSG</td>
<td>8-22</td>
</tr>
<tr>
<td>P$RUNTIM</td>
<td>8-22</td>
</tr>
<tr>
<td>P$SFRAME</td>
<td>8-23</td>
</tr>
<tr>
<td>P$TIME</td>
<td>8-23</td>
</tr>
<tr>
<td>P$TIMER</td>
<td>8-23</td>
</tr>
<tr>
<td>P$TRACE</td>
<td>8-23</td>
</tr>
<tr>
<td>P$TRUNC</td>
<td>8-24</td>
</tr>
<tr>
<td>P$WB</td>
<td>8-24</td>
</tr>
<tr>
<td>P$WCH</td>
<td>8-24</td>
</tr>
<tr>
<td>P$WEOF</td>
<td>8-25</td>
</tr>
<tr>
<td>P$WI</td>
<td>8-25</td>
</tr>
<tr>
<td>P$WO</td>
<td>8-25</td>
</tr>
<tr>
<td>P$WRFIX</td>
<td>8-26</td>
</tr>
<tr>
<td>P$WRFLT</td>
<td>8-26</td>
</tr>
<tr>
<td>P$WRITE</td>
<td>8-26</td>
</tr>
<tr>
<td>P$WRITLN</td>
<td>8-27</td>
</tr>
<tr>
<td>P$WSTR</td>
<td>8-27</td>
</tr>
</tbody>
</table>

9. **MULTITASKING SUBPROGRAMS**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK ROUTINES</td>
<td>9-1</td>
</tr>
<tr>
<td>TASK CONTROL ARRAY</td>
<td>9-6</td>
</tr>
<tr>
<td>LOCK ROUTINES</td>
<td>9-7</td>
</tr>
<tr>
<td>EVENT ROUTINES</td>
<td>9-9</td>
</tr>
<tr>
<td>UTILITY SUBPROGRAMS</td>
<td>9-11</td>
</tr>
</tbody>
</table>
A. ALGORITHMS

ACOS AND ACOSV .................................................. A-1
 ALOG AND ALOGV .................................................. A-2
 ATAN AND ATANV .................................................. A-2
 ATAN2 AND ATANZV ................................................ A-3
 CABS AND CAVBSV ................................................ A-3
 CCOS AND CCOSV .................................................. A-4
 CEXP AND CEXPV .................................................. A-4
 CILOG AND CILOGV ................................................ A-4
 COS AND COSV; COSS AND COSSV .................................. A-5
 COSH AND COSHV .................................................. A-5
 COT AND COTV ..................................................... A-7
 CSIN AND CSINV .................................................. A-7
 CSQRT AND CSQRTV ................................................ A-8
 CTOCSS, CTOCSV, CTOCVS, AND CTOCVVV ........................ A-8
 CTOISS, CTOISV, CTOIVS, AND CTOIVVV ........................ A-9
 CTORSS, CTORSV, CTOIVS, AND CTORIVV ........................ A-9
 DACOS AND DACOSV ................................................ A-9
 DASIN AND DASINV ................................................ A-10
 DATAN AND DATANV ................................................. A-10
 DCOS AND DCOSV .................................................. A-11
 DCOSH AND DCOSHV ............................................... A-11
 DCOT AND DCOTV .................................................. A-11
 DEXP AND DEXPV .................................................. A-11
 DLOG AND DLOGV .................................................. A-12
 DMOD AND DMODV .................................................. A-13
 DSINH AND DSINHV ................................................ A-13
 DSQRT AND D SQRTV ............................................... A-13
 DTAN AND DTANV .................................................. A-14
 DTANH AND DTANHV ............................................... A-14
 DTODSS, DTODSV, DTODVS, AND DTODVV ........................ A-14
 DTOISS, DTOISV, DTOIVS, AND DTOIVVV ........................ A-14
 EXP AND EXPV ..................................................... A-14
 ITOISS ............................................................. A-15
 ITOIVS, ITOIVVV .................................................. A-15
 RANF .............................................................. A-16
 RANFV ............................................................. A-16
 RANGE T ............................................................ A-17
 RANSET ............................................................. A-17
 RTOISS, RTOIVS, AND RTOIVVV ................................ A-17
 RTORSS ............................................................. A-18
 RTOIVS ............................................................. A-18
 RTORVV ............................................................. A-18
 SQRT AND SQRTV .................................................. A-19
 TANH AND TANHV .................................................. A-19

BIBLIOGRAPHY ....................................................... A-20

SR-0014 xiv
B. PERFORMANCE STATISTICS

COMMON MATHEMATICAL SUBPROGRAMS

SCIENTIFIC APPLICATIONS SUBPROGRAMS

C. SORT ENTRY POINTS

FIGURES

5-1 Group name
5-2 Variable entry
5-3 Array entry
5-4 Logical read
5-5 Logical write
7-1 Exchange Package printout
7-2 Parameter Control Table
7-3 Parameter control entry
7-4 Message Control Table

TABLES

2-1 Subprogram summary
2-2 Pascal subprogram summary
3-1 Logarithmic routines
3-2 Exponential routines
3-3 Square root routines
3-4 Trigonometric routines
3-5 Hyperbolic routines
3-6 Boolean arithmetic routines
3-7 Values raised to a power
3-8 Double-precision arithmetic routines
3-9 Triple-precision arithmetic routines
3-10 64-bit integer division
3-11 Character comparison functions called from FORTRAN
3-12 Character comparison functions called from CAL
3-13 ASCII conversion
3-14 Miscellaneous math routines
3-15 Random number routines
4-1 Basic linear algebra subprograms (BLAS)
4-2 Other linear algebra subprograms
4-3 Single-precision LINPACK routines
4-4 Single-precision EISPACK routines
4-5 Arguments for Fourier transform routines
4-6 ISRCH routines
4-7 WHEN routines
4-8 Sort times in seconds for ORDERS
4-9 Sort times in seconds with ASCII key
5-1 FORTRAN I/O routines
TABLES (continued)

5-2 OPEN specifiers and their meanings ........................................ 5-36
5-3 CLOSE specifiers and their meanings ....................................... 5-37
5-4 INQUIRE specifiers and their meanings .................................... 5-38
5-5 Conversion mode descriptions ................................................ 5-80
5-6 Conversion return conditions ................................................ 5-81
5-7 Error codes for record-addressable, random access dataset
   I/O routines .......................................................................... 5-86
5-8 CLOSMS statistics .................................................................. 5-92
5-9 WOPEN statistics .................................................................. 5-104
5-10 Error codes for word-addressable, random access dataset
   I/O routines .......................................................................... 5-106
7-1 Performance counter group descriptions ................................... 7-86
B-1 Statistics for single-precision, single-argument subprograms . B-2
B-2 $SCILIB timings and comparisons .......................................... B-6
B-3 $SCILIB timings and MFLOP rates ......................................... B-9

INDEX
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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Call by Address</th>
<th>Call by Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>RTE</td>
<td>RTE%</td>
</tr>
<tr>
<td>Vector</td>
<td>%RTE</td>
<td>%RTE%</td>
</tr>
</tbody>
</table>

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 B65g and T00 through T67g. Registers B70g through B77g and T70g through T77g 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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italic</em></td>
<td>Define generic terms representing words or symbols to be supplied by the user and identify new terms</td>
</tr>
<tr>
<td>[] Brackets</td>
<td>Enclose optional portions of a command format</td>
</tr>
<tr>
<td>(S1), (S2), etc.</td>
<td>Content of register S1, S2, etc., respectively</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CPT Standard</th>
<th>Call From CAL Only</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>ABORT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Aborts job with traceback</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS</td>
<td>REAL</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes absolute value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCESS</td>
<td>ACS</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Accesses a permanent dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACOS</td>
<td>ACOS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes arccosine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACOSV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized arccosine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACPTBAD</td>
<td>RCVBAD</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Transfers bad data to a specified buffer for the caller</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard Call From CAL Only</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFT Call Type</td>
<td>Library</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>ACQUIRE</td>
<td>AQR</td>
<td>ACQUIRE</td>
<td>S</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td>ACTTABLE</td>
<td>ACTTABLE</td>
<td>ACTTABLE</td>
<td>S</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td>ADJUST</td>
<td>ADJ</td>
<td>ADJUST</td>
<td>S</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td>AIMAG</td>
<td>COMPLX</td>
<td>AIMAG</td>
<td>SF</td>
<td>$ARLIB</td>
</tr>
<tr>
<td>AINT</td>
<td>REAL</td>
<td>AINT</td>
<td>SF</td>
<td>$ARLIB</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard Call From CAL Only</td>
<td>CPT Call Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ALF</td>
<td>ALF</td>
<td>$ALF</td>
<td>S</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td>AMOD</td>
<td>REAL</td>
<td>AMOD</td>
<td>SF</td>
<td>$ARLIB</td>
</tr>
<tr>
<td>AMU</td>
<td>TM</td>
<td>$AMU</td>
<td>$SYSLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td>AND</td>
<td>BOOLEAN</td>
<td>AND</td>
<td>SF</td>
<td>$ARLIB</td>
</tr>
<tr>
<td>ANINT</td>
<td>REAL</td>
<td>ANINT</td>
<td>SA</td>
<td>$ARLIB</td>
</tr>
<tr>
<td>ARERP</td>
<td>ARERP</td>
<td>ARERP%</td>
<td>$ARLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ARGPLIMQ</td>
<td>FLOW ARGPLIMQ</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>ASCDC</td>
<td>ASCDC ASCDC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>ASIN</td>
<td>ACOS ACOSV</td>
<td>SF VF</td>
<td>$ARLIB $ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>ASIN% %ASIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSIGN</td>
<td>ASS ASSIGN</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td>ATAN</td>
<td>ATAN ATANV</td>
<td>SF VF</td>
<td>$ARLIB $ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>ATAN% %ATAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ATAN2</td>
<td>ATAN2</td>
<td>ATAN2</td>
<td>ATAN2%</td>
<td>SF</td>
</tr>
<tr>
<td>ATAN2V</td>
<td></td>
<td>%ATAN2%</td>
<td>%ATAN2%</td>
<td>VF</td>
</tr>
<tr>
<td>ATS</td>
<td>TM</td>
<td></td>
<td>$ATS</td>
<td></td>
</tr>
<tr>
<td>AXPY</td>
<td>CAXPY</td>
<td>CAXPY</td>
<td>S</td>
<td>$SCILIB</td>
</tr>
<tr>
<td></td>
<td>SAXPY</td>
<td>SAXPY</td>
<td>S</td>
<td>$SCILIB</td>
</tr>
<tr>
<td></td>
<td>SPAXPY</td>
<td>SPAXPY</td>
<td>S</td>
<td>$SCILIB</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>BACK</td>
<td>BACKFILE</td>
<td>$BACK</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>BKFFILE</td>
<td></td>
<td></td>
<td>$BKSP</td>
</tr>
<tr>
<td></td>
<td>BKSP</td>
<td></td>
<td></td>
<td>$BKSPF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>BICON</td>
<td>BICONV</td>
<td></td>
<td>BICONV</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>BICONZ</td>
<td></td>
<td>BICONZ</td>
<td>S</td>
</tr>
<tr>
<td>BTD</td>
<td>BTD</td>
<td></td>
<td>BTD%</td>
<td>$BTD</td>
</tr>
<tr>
<td></td>
<td>BTD</td>
<td></td>
<td>BTD%</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td>BTDR</td>
<td></td>
<td>BTDR%</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td>BTDL</td>
<td></td>
<td>BTDL%</td>
<td>SF</td>
</tr>
<tr>
<td>BTO</td>
<td>BTO</td>
<td></td>
<td>BTO%</td>
<td>$BTO</td>
</tr>
<tr>
<td></td>
<td>BTO</td>
<td></td>
<td>BTO%</td>
<td>SF</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>BTO</td>
<td></td>
<td>primary</td>
<td>UPDATE</td>
<td>CFT Standard</td>
</tr>
<tr>
<td>BTO continued</td>
<td></td>
<td>primary</td>
<td>UPDATE</td>
<td>CFT Standard</td>
</tr>
<tr>
<td>BTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTOR</td>
<td>BTOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTOL</td>
<td>BTOL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CABS</td>
<td>CABS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CABS</td>
<td>%CABS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CABSV</td>
<td>%CABS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCOS</td>
<td>CCOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCOS</td>
<td>%CCOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCOSV</td>
<td>%CCOS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Converting binary to octal ASCII right justified, zero filled.

Converting binary to octal ASCII left justified, zero filled.

Calculates complex absolute value

Calculates vectorized complex absolute value

Computes complex cosine

Computes vectorized complex cosine
<table>
<thead>
<tr>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDC</td>
<td>CDCI</td>
<td>$CDCI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Translates CDC formatted input data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDCO</td>
<td>$CDCO</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Translates CDC formatted output data</td>
<td></td>
</tr>
<tr>
<td>CEXPR</td>
<td>CEF</td>
<td>CEXPR</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td>Transforms an expression character string to a Reverse Polish Table</td>
</tr>
<tr>
<td>CHAR</td>
<td>CARCON</td>
<td>$MOVE</td>
<td>$FTLIB</td>
<td>Ind.</td>
<td>Transfers one character item to result</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCARCON</td>
<td>$PAD</td>
<td>$FTLIB</td>
<td>Ind.</td>
<td>Terminates transfer for store</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$CCI</td>
<td>$FTLIB</td>
<td>Ind.</td>
<td>Initializes concatenation for store</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$CCT</td>
<td>$FTLIB</td>
<td>Ind.</td>
<td>Transfers one character item to result</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARCPR</td>
<td>$CCF</td>
<td>$FTLIB</td>
<td>Ind.</td>
<td>Terminates transfer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$GE</td>
<td>SF</td>
<td>$FTLIB</td>
<td>Compares ASCII arguments for greater than or equal to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$GT</td>
<td>SF</td>
<td>$FTLIB</td>
<td>Compares ASCII arguments for greater than</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CFT Standard</td>
<td>Call From CALL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Address Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR continued</td>
<td>LLT</td>
<td>$LT</td>
<td>SF</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Compares ASCII arguments for less than</td>
</tr>
<tr>
<td></td>
<td>LLE</td>
<td>$LE</td>
<td>SF</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Compares ASCII arguments for less than or equal to</td>
</tr>
<tr>
<td></td>
<td>$EQ</td>
<td>$EQ</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Compares ASCII arguments for equality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$NE</td>
<td>$NE</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Compares ASCII arguments for nonequality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDEX INDEX</td>
<td>SF</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Finds position of second argument as substring of first argument</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEN LEN</td>
<td>SF</td>
<td>$PTLIB</td>
<td>Ind.</td>
<td>Finds length of argument</td>
<td></td>
</tr>
<tr>
<td>CHCONV</td>
<td>CHCONV CHCONV</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts ASCII character to integer</td>
<td></td>
</tr>
<tr>
<td>CLEARBT</td>
<td>BTMODE CLEARBT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Temporarily disables bidirectional memory transfers</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>CLEARBTS</td>
<td>BTMODE</td>
<td>CLEARBTS</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>CLEARFI</td>
<td>FIMODE</td>
<td>CLEARFI</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>CLEARFIS</td>
<td>FIMODE</td>
<td>CLEARFIS</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>CLOCK</td>
<td>CLOCK</td>
<td>CLOCK</td>
<td>S/SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>CLOSE</td>
<td>CLOSE</td>
<td>$CLS</td>
<td>SA</td>
<td>$IOLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td>SYMDBC</td>
<td>CLOSE</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Address</th>
<th>Value</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPLX</td>
<td>REAL</td>
<td>CMPLX</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Converts two reals to complex</td>
</tr>
<tr>
<td>COMPL</td>
<td>BOOLEAN</td>
<td>COMPL</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Forms logical complement</td>
</tr>
<tr>
<td>CONJG</td>
<td>COMPLX</td>
<td>CONJG</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes complex conjugate</td>
</tr>
<tr>
<td>COPY</td>
<td>CCOPY</td>
<td>CCOPY</td>
<td></td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Copies complex array into complex array</td>
</tr>
<tr>
<td></td>
<td>SCOPY</td>
<td>SCOPY</td>
<td></td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Copies real array into real array</td>
</tr>
<tr>
<td>COPYD</td>
<td>COPY</td>
<td>COPYD</td>
<td></td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Copies one dataset to another; EOD not copied</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>CFT Standard Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>COPYF</td>
<td>COPY</td>
<td>COPYF</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Copies files from one dataset to another; EOD not written</td>
<td></td>
</tr>
<tr>
<td>COPYR</td>
<td>COPY</td>
<td>COPYR</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Copies records from one dataset to another; EOF not written</td>
<td></td>
</tr>
<tr>
<td>COPYU</td>
<td>COPYU</td>
<td>COPYU</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Copies data to EOD in unblocked format</td>
<td></td>
</tr>
<tr>
<td>COS</td>
<td>COS</td>
<td>COS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes cosine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COSV</td>
<td>%COS%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized cosine</td>
<td></td>
</tr>
<tr>
<td>COSH</td>
<td>COSH</td>
<td>COSH%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes hyperbolic cosine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COSHV</td>
<td>%COSH%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized hyperbolic cosine</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>COSS</td>
<td>COSS</td>
<td>COSS%</td>
<td>SF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes cosine and sine</td>
</tr>
<tr>
<td></td>
<td>COSSV</td>
<td>%COSS</td>
<td>VF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes vectorized cosine and sine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%COSS%</td>
<td>VF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Same as %COSS</td>
</tr>
<tr>
<td>COSSH</td>
<td>COSH</td>
<td>COSSH%</td>
<td>SF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes hyperbolic cosine and sine</td>
</tr>
<tr>
<td></td>
<td>COSHV</td>
<td>%COSSH</td>
<td>VF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes vectorized hyperbolic cosine and sine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%COSSH%</td>
<td>VF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Same as %COSSH</td>
</tr>
<tr>
<td>COT</td>
<td>COT</td>
<td>COT%</td>
<td>SF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes cotangent</td>
</tr>
<tr>
<td></td>
<td>COTV</td>
<td>%COT</td>
<td>VF $ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes vectorized cotangent</td>
</tr>
<tr>
<td>CRACK</td>
<td>CRACK</td>
<td>CRACK</td>
<td>S $SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Reformats a user-supplied string into verb, separators, keywords and values</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>CS</td>
<td>CCS</td>
<td>$CS$</td>
<td>$CS$</td>
<td>S</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>CSIN</td>
<td>CSIN</td>
<td>CSIN%</td>
<td>%CSIN%</td>
<td>SF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>CSINV</td>
<td></td>
<td>%CSIN%</td>
<td>%CSIN%</td>
<td>VF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>CTOC</td>
<td>CTOCSS</td>
<td>CTOC%</td>
<td>SF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTOCSV</td>
<td>CTO%C%</td>
<td>VF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTOCVS</td>
<td>%CTOC%</td>
<td>VF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTOCVV</td>
<td>%CTOC%</td>
<td>VF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTOI</td>
<td>CTOISS</td>
<td>CTOI%</td>
<td>SF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTOISV</td>
<td>CTO%i%</td>
<td>VF</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>CTOI</td>
<td>CTOIVS</td>
<td>$CTOI$%</td>
<td>VF</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises complex vector to integer scalar power</td>
</tr>
<tr>
<td></td>
<td>CTOIVV</td>
<td>$CTOI$%I%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Raises complex vector to integer vector power</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTOI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard Address</td>
<td>CPT Standard Value</td>
<td>Call From CAL Only</td>
<td>CPT Call Type</td>
<td>Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>DATAN2</td>
<td>DATAN</td>
<td>DATAN2</td>
<td>DATAN2%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes double-precision 2-argument arctangent</td>
</tr>
<tr>
<td></td>
<td>DATANV</td>
<td>%DATAN2%</td>
<td>%DATAN2</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized double-precision 2-argument arctangent</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATE</td>
<td>DATE</td>
<td>S/SP</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns current date in \textit{mmdd:yy} format</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JDATE</td>
<td>JDATE</td>
<td>S/SP</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns current Julian date in \textit{yydd} format</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMEF</td>
<td>TIMEF</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns time since initial TIMEF call in milliseconds</td>
<td></td>
</tr>
<tr>
<td>DBLE</td>
<td>REAL</td>
<td>DBLE</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Converts type real to double precision</td>
<td></td>
</tr>
<tr>
<td>DCOS</td>
<td>DCOS</td>
<td>DCOS%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes double-precision cosine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCOSV</td>
<td>%DCOS%</td>
<td>%DCOS</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized double-precision cosine</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>DDIM</td>
<td>DDIM DDIMV</td>
<td>DDIM</td>
<td>%DDIM%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Double-precision positive real difference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%DDIM%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECODE</td>
<td>RFD</td>
<td>$DFI</td>
<td>$DFV $DFA $DFV%</td>
<td>$DFV $DFF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initializes for decode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE</td>
<td>DELETE</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Removes a saved dataset from the Dataset Catalog</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELTSK</td>
<td>DELTSK</td>
<td>$DELTSK%</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Deletes calling task</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIM</td>
<td>REAL</td>
<td>DIM</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Computes positive real difference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$ARLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Reference Name</th>
<th>Call From CAL Only</th>
<th>CFT Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>DINT</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Truncates double-precision numbers to integers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DINTV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPOSE</td>
<td>DISPOSE</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Directs a dataset to the specified queue</td>
<td></td>
</tr>
<tr>
<td>DMOD</td>
<td>DMOD</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes division remainder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DMOD%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNINT</td>
<td>DNINT</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Calculates nearest integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DNINTV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOT</td>
<td>CDOT</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the conjugated dot product of two complex arrays</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDOTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDOTU</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the unconjugated dot product of two complex arrays</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Reference Name</th>
<th>Entry Type</th>
<th>CFT Standard Address</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOT</td>
<td>SDOT</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the dot product of two real arrays</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPDOT</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the dot product of two real arrays; one array is referenced indirectly.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPREC</td>
<td>DASS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision addition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDSS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision division</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DMSS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision multiplication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSSS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision subtraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAVV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision addition, scalar+vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAVS</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision addition, vector+scalar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAVV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision addition, vector+vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDSSV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision division, scalar/vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDVVS</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision division, vector/scalar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDVV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision division, vector/vector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>DPREC</td>
<td>DVVV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMSV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision multiplication, scalar x vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMVS%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision multiplication, vector x scalar</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMVV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision multiplication, vector x vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSVV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSSV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision subtraction, scalar-vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSVS%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision subtraction, vector-scalar</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSVV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs double-precision subtraction, vector-vector</td>
<td></td>
</tr>
<tr>
<td>DPROD</td>
<td>DBLE</td>
<td>DPROD</td>
<td></td>
<td>SA</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs double-precision product of two real arguments</td>
<td></td>
</tr>
<tr>
<td>DRIVER</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Allows a user to directly program a CRAY channel on an IOS</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard</td>
<td>Call From CAL Only</td>
<td>CPT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>DSASC</td>
<td>DSASC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Converts display code character to ASCII character</td>
</tr>
<tr>
<td>DSIGN</td>
<td>DBLE</td>
<td>SA</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Transfers sign from one double-precision number to another</td>
</tr>
<tr>
<td>DSIN</td>
<td>DCOS</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Computes double-precision sine</td>
</tr>
<tr>
<td></td>
<td>DCOSV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Computes vectorized double-precision sine</td>
</tr>
<tr>
<td>DSNDSP</td>
<td>DSNDSP</td>
<td>$DSN DSP</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td>Searches Logical File Table (LFT) in user's I/O area for dataset name</td>
</tr>
<tr>
<td>DTB</td>
<td>DTB</td>
<td>$DTB</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Converts ASCII decimal integer number to binary</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>DTOD</td>
<td>DTODSS</td>
<td>DTOD%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision scalar to double-precision scalar power</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODSV</td>
<td>DTO%D%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision scalar to double-precision vector power</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODVS</td>
<td>DTO%D%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision vector to double-precision scalar power</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODVV</td>
<td>DTO%D%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision vector to double-precision vector power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTOI</td>
<td>DTOIIS</td>
<td>DTOI%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision scalar to integer scalar power</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTOISV</td>
<td>DTO%I%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises double-precision scalar to integer vector power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CPT Standard</td>
<td>Entry Type</td>
<td>CPT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>DTOI</td>
<td>DTOIVS</td>
<td>%DTOI%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to integer scalar power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTOIVV</td>
<td>%DTOI%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to integer vector power</td>
<td></td>
</tr>
<tr>
<td>DTOI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTOIW%</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to integer vector power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTO%I%</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to integer vector power</td>
<td></td>
</tr>
<tr>
<td>DTOI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTO%R%</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision scalar to real vector power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODSV</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision scalar to real vector power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODVS</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision scalar to real vector power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODVV</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to real vector power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTODVV</td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Raises double-precision vector to real vector power</td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td>CRAYDUMP</td>
<td>CRAYDUMP</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Prints a memory dump to a specified dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PDUMP</td>
<td>DUMP</td>
<td>$DUMP</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td>Dumps memory and aborts job</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PDUMP</td>
<td>PDUMP</td>
<td>$PDUMP</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td>Dumps memory</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1. Subprogram summary (continued)
<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DUMPJOB</strong></td>
<td>DUMPJOB</td>
<td>DUMPJOB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Creates an unblocked dataset containing the user job area image</td>
</tr>
<tr>
<td><strong>ECHO</strong></td>
<td>ECHO</td>
<td>ECHO</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Allows user to turn on and off classes of messages to user logfile</td>
</tr>
<tr>
<td><strong>EISPACK</strong></td>
<td></td>
<td></td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>See table 4-4.</td>
</tr>
<tr>
<td><strong>ENCODE</strong></td>
<td>WFD</td>
<td></td>
<td>$EFI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initializes for encode</td>
</tr>
<tr>
<td></td>
<td>WFD</td>
<td></td>
<td>$EFV $EFV%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Cracks format; encodes output.</td>
</tr>
<tr>
<td></td>
<td>WFD</td>
<td></td>
<td>$EFA</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Finalizes encode</td>
</tr>
<tr>
<td><strong>END</strong></td>
<td>END</td>
<td></td>
<td>$END</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Terminates current job step and advances job to next job step</td>
</tr>
<tr>
<td></td>
<td>END$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Entry Type</td>
<td>CFT Call From CAL Only</td>
<td>Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------</td>
<td>------------</td>
<td>------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ENDFILE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EODW</td>
<td>EODW</td>
<td></td>
<td></td>
<td>$EODW</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>$EODW</td>
<td>EODW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EOFR</td>
<td>EOFR</td>
<td></td>
<td></td>
<td>$EOFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOFW</td>
<td>EOFW</td>
<td></td>
<td></td>
<td>$EOFW</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOFW</td>
<td>EOFW</td>
<td></td>
<td></td>
<td>$EOFW</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td></td>
<td>EOSFR</td>
<td>EOSFR</td>
<td></td>
<td></td>
<td>$EOSFR</td>
<td>$SYSLIB</td>
</tr>
<tr>
<td></td>
<td>EOR</td>
<td>EOR</td>
<td></td>
<td></td>
<td>$EOR</td>
<td>$IOLIB</td>
</tr>
<tr>
<td>Primary</td>
<td>UPDATE</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From</td>
<td>CFT Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>Reference</td>
<td>Deck Name</td>
<td></td>
<td></td>
<td>CAL Only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>SF</td>
<td>$IOLIB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EOF</td>
<td>$IEOF</td>
<td>SF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQV</td>
<td>BOOLEAN</td>
<td>EQV</td>
<td>SF</td>
<td>$ARLIB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVREXIT</td>
<td>ABORT</td>
<td>EVREXIT</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVASGN</td>
<td>EVASGN</td>
<td>EVASGN</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVCLEAR</td>
<td>EVCLEAR</td>
<td>EVCLEAR</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVPOST</td>
<td>EVPOST</td>
<td>EVPOST</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1. Subprogram summary (continued)
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVREL</td>
<td>EVREL</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Releases event identifier</td>
</tr>
<tr>
<td>EVTEST</td>
<td>EVTEST</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Tests event for posted state</td>
</tr>
<tr>
<td>EVWAIT</td>
<td>EVWAIT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Delays calling task until event is posted</td>
</tr>
<tr>
<td>EXCHANGE</td>
<td>XPFMT</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Format Exchange Package</td>
</tr>
<tr>
<td></td>
<td>FXPF</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Print Exchange Package</td>
</tr>
<tr>
<td></td>
<td>XPFMT</td>
<td>$FXP</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Format octal representation of Exchange Package</td>
</tr>
<tr>
<td>EXIT</td>
<td>EXIT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Terminates current job step</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard</td>
<td>Call From CAL Only</td>
<td>CPT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Address</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td></td>
<td></td>
<td>CEXP</td>
<td>CEXP%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CEXPV</td>
<td>%CEXP%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEXP</td>
<td>DEXP%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEXPV</td>
<td>%DEXP%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXP</td>
<td>EXP%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPV</td>
<td>%EXP%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>FETCH</td>
<td></td>
<td></td>
<td>FETCH</td>
<td>FETCH</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td>CRFFT2</td>
<td>CRFFT2</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFFT2</td>
<td>CFFT2</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RCFFT2</td>
<td>RCFFT2</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>----------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>FP6064</td>
<td>FP6064</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP6460</td>
<td>FP6460</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILTER</td>
<td>FILTERG</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILTERS</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPFILT</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINDCH</td>
<td>FINDCH</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>FLOAT</td>
<td>INTEGER</td>
<td>FLOAT</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Converts type integer to real</td>
<td></td>
</tr>
<tr>
<td>FTERP</td>
<td>FTERP</td>
<td>FTERP</td>
<td>$FTLIB</td>
<td>Dep.</td>
<td>Processes $FTLIB errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>ARGPLIMQ</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Controls argument printing for flowtrace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>FLOWENTR</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Processes CFT flowtrace option for calls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>FLOWEXIT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Processes CFT flowtrace option for RETURN statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>FLOWLIM</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Sets limit on the number of subroutine calls to be traced</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>FLOWSTOP</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Processes CFT flowtrace option</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOW</td>
<td>SETPLIMQ</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Controls CALL printing for flowtrace</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOLR</td>
<td>FOLR</td>
<td>FOLR</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Solves first-order linear recurrences</td>
</tr>
<tr>
<td>FOLRN</td>
<td>FOLRN</td>
<td>FOLRN</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Solves for the last term of a first-order linear recurrence</td>
</tr>
<tr>
<td>FOLRP</td>
<td>FOLRP</td>
<td>FOLRP</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Solves first-order linear recurrences</td>
</tr>
<tr>
<td>FOLR2</td>
<td>FOLR2</td>
<td>FOLR2</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Solves first-order linear recurrences</td>
</tr>
<tr>
<td>FOLR2P</td>
<td>FOLR2P</td>
<td>FOLR2P</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Combination of FOLRP &amp; FOLR2</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>Call From CAL Only</td>
<td>CFT Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------</td>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFT Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Address</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATHER</td>
<td>GATHER</td>
<td>GATHER</td>
<td></td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Gathers a vector from a source vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETBL</td>
<td>SYMDBC</td>
<td>GETBL</td>
<td></td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Returns address of subroutine name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETDSP</td>
<td>GTDSP%</td>
<td>GETDSP</td>
<td></td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Locates a Dataset Parameter Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETLPP</td>
<td>GLPP%</td>
<td>GETLPP</td>
<td></td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Returns lines from JCLPP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETNAMEQ</td>
<td>GNAMEQ</td>
<td>GETNAMEQ</td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns ASCII left-justified, space-filled name of routine that called FLOWENTR or FLOWEXIT</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPOS</td>
<td>GETPOS</td>
<td>GETPOS</td>
<td>$GPOS</td>
<td>SF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Gets dataset position</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GPOS</td>
<td>$GPOS</td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Gets tape dataset position</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GTPOS</td>
<td>GTPOS%</td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETREGS</td>
<td>NAMEQ</td>
<td>GETREGS</td>
<td>S</td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Returns register usage statistics for FLOWENTR</td>
<td></td>
</tr>
<tr>
<td>GPARAM</td>
<td>GPARAM</td>
<td>GETPARAM</td>
<td>$GP</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>All three routines transfer control statement parameter values to an array provided by the calling routine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$GPARAM</td>
<td></td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PAL</td>
<td></td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Address</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEAP MANAGER</td>
<td>HPALLOC</td>
<td>HPALLOC</td>
<td>ALLOC%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Allocates a block of memory from the heap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPCHECK</td>
<td>HPCHECK</td>
<td>HCHECK%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Checks integrity of the heap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPCLMOVE</td>
<td>HPCLMOVE</td>
<td>CLMOVE%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Changes length of a heap block, and moves block if it cannot be extended in place</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPDEALLC</td>
<td>HPDEALLC</td>
<td>DEALLC%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns a block of memory to the heap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPDUMP</td>
<td>HPDUMP</td>
<td>HPDUMP%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Writes information about the heap to a dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPGROW</td>
<td>HPGROW</td>
<td>HPGROW%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Heap expansion routine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPLEN</td>
<td>HPLEN</td>
<td>HPLEN%</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns length of a heap block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPMEM</td>
<td>HPMEM</td>
<td>HPMEM%</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Memory request routine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPMERGE</td>
<td>HPMERGE</td>
<td>HPMERGE%</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Heap merge routine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPNEWLEN</td>
<td>HPNEWLEN</td>
<td>NEWLEN%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Changes length of a heap block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPSHRINK</td>
<td>HPSHRINK</td>
<td>SHRINK%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns memory from heap to operating system</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPSTAT</td>
<td>HPSTAT</td>
<td>HPSTAT%</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Returns information about the heap</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>----------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>I00DEL</td>
<td>I00DEL</td>
<td>I00DEL</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Deletes SKOL string or substring and returns the length of resulting string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I00ERR</td>
<td>I00ERR</td>
<td>I00ERR</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Handles run time errors in SKOL programs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I00MVC</td>
<td>I00MVC</td>
<td>I00MVC</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Replaces SKOL string or substring with simple character and returns the length of resulting string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I00MVM</td>
<td>I00MVM</td>
<td>I00MVM</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Replaces SKOL string or substring with another SKOL string or substring and returns the length of the resulting string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>I00ORD</td>
<td>I00ORD</td>
<td></td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Returns internal SKOL code for a given ASCII character</td>
<td></td>
</tr>
<tr>
<td>I00READ</td>
<td>I00READ</td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Reads a logical record in Al format and converts each word containing an ASCII character, left-justified, space-filled, to its internal SKOL code</td>
<td></td>
</tr>
<tr>
<td>I00SETUP</td>
<td>I00SETUP</td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Initializes a SKOL program's table for direct translation of ASCII character codes to internal ordinal numbers</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>IOWRITE</td>
<td>IOOWRITE</td>
<td>IOOWRITE</td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>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.</td>
</tr>
<tr>
<td>IABS</td>
<td>INTEGER</td>
<td>IABS</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes integer absolute value</td>
</tr>
<tr>
<td>IBM</td>
<td>IBMI</td>
<td></td>
<td></td>
<td>$IBMI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Translates input IBM formatted data</td>
</tr>
<tr>
<td></td>
<td>IBMO</td>
<td></td>
<td></td>
<td>$IBMO</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Translates output IBM formatted data</td>
</tr>
<tr>
<td>ICAMAX</td>
<td>ICAMAX</td>
<td>ICAMAX</td>
<td></td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the first index of the maximum absolute value of a complex array</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>ICEIL</td>
<td>ICEIL</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Returns integer ceiling of a rational number represented as two integer parameters</td>
</tr>
<tr>
<td>IDIM</td>
<td>INTEGER</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Computes positive integer difference</td>
</tr>
<tr>
<td>IDNINT</td>
<td>IDNINT</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Finds nearest integer to double-precision number</td>
</tr>
<tr>
<td>IFDNT</td>
<td>IFDNT</td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Determines if a dataset is local to the job</td>
</tr>
<tr>
<td>IGTYBT</td>
<td>IGTYBT</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Fetches bytes</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>IILZ</td>
<td>IILZ</td>
<td>IILZ</td>
<td>IILZ</td>
<td>Ind.</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Returns the number of zero values before the first nonzero value</td>
</tr>
<tr>
<td>IIN</td>
<td>NCON</td>
<td>$IIN</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Contains integer powers of 10 in range of $10^0$ to $10^{18}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILLZ</td>
<td>ILLZ</td>
<td>ILLZ</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Returns the number of values that do not have the first bit set before the first value that does have the first bit set</td>
<td></td>
</tr>
<tr>
<td>ILSUM</td>
<td>ILSUM</td>
<td>ILSUM</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Returns total number of true values in array declared LOGICAL</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CPT Standard Address</td>
<td>CPT Standard Value</td>
<td>Call From CAL Only</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>IMX</td>
<td>NCON</td>
<td>$IMX</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Contains double-precision floating-point representation of negative powers of 10 in range of $10^0$ to $10^{-4096}$</td>
</tr>
<tr>
<td>INQUIRE</td>
<td>INQUIRE</td>
<td>$INQ</td>
<td>SA $IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
<td>Returns the status of a unit or a dataset</td>
</tr>
<tr>
<td>INSASCI</td>
<td>INSASCI%</td>
<td>S $SYSLIB</td>
<td></td>
<td></td>
<td></td>
<td>Ind.</td>
<td>Inserts ASCII parameters into a message</td>
</tr>
<tr>
<td>INT</td>
<td>REAL INT</td>
<td>SF $ARLIB</td>
<td></td>
<td></td>
<td></td>
<td>Ind.</td>
<td>Truncates to integer value</td>
</tr>
<tr>
<td>INT</td>
<td>TM</td>
<td>$INT</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
<td>Initializes table pointers</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call From CAL Only</td>
<td>Address</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT6064</td>
<td>INT6064</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Converts CDC integer to Cray integer</td>
</tr>
<tr>
<td>INT6460</td>
<td>INT6460</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Converts Cray integer to CDC integer</td>
</tr>
<tr>
<td>IOERP</td>
<td>IOERP</td>
<td>IOERP% NLERP%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Processes I/O errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Processes NAMELIST errors</td>
</tr>
<tr>
<td>IOSTAT</td>
<td>IOSTAT</td>
<td>SF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Returns EOF status; clears UEOF in DSP.</td>
</tr>
<tr>
<td>IPX</td>
<td>NCON</td>
<td>$IPX</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Contains double-precision floating-point representation of positive powers of 10 in range of $10^0$ to $10^{-4096}$</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard</td>
<td>Call From CAL Only</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>ISAMAX</td>
<td>ISAMAX</td>
<td></td>
<td>ISAMAX</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the first index of the largest absolute value in a real array</td>
</tr>
<tr>
<td>ISAMIN</td>
<td>ISAMIN</td>
<td></td>
<td>ISAMIN</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds first index of the smallest absolute value in a real vector</td>
</tr>
<tr>
<td>ISIGN</td>
<td>INTEGER</td>
<td></td>
<td>ISIGN</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Transfers sign from one integer to another</td>
</tr>
<tr>
<td>ISRCH</td>
<td>ISEARCH</td>
<td></td>
<td>ISRCHEQ</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Returns the first location of a true condition</td>
</tr>
<tr>
<td></td>
<td>ISRHNE</td>
<td></td>
<td>ISRHFLI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISRHFLT</td>
<td></td>
<td>ISRHFLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISRHFGT</td>
<td></td>
<td>ISRHFGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISRHFGE</td>
<td></td>
<td>ISRHFGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISRHILT</td>
<td></td>
<td>ISRHILT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Entry Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CFT Standard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISRCH</td>
<td>ISRCHILE</td>
<td>ISRCHILE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>continued</td>
<td>ISRCHIGT</td>
<td>ISRCHIGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISMXP</td>
<td>ISMAX</td>
<td>ISMAX</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the first index of the maximum of an array</td>
</tr>
<tr>
<td>ISMIN</td>
<td>ISMIN</td>
<td>ISMIN</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Finds the first index of the minimum of an array</td>
</tr>
<tr>
<td>ITOI</td>
<td>ITOISS</td>
<td>ITOI%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises integer scalar to integer scalar power</td>
</tr>
<tr>
<td></td>
<td>ITOIVS</td>
<td>%ITOI%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises integer vector to integer scalar power</td>
</tr>
<tr>
<td></td>
<td>ITOIVVV</td>
<td>ITOI%I%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises integer scalar to integer vector power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%ITO%I%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises integer vector to integer vector power</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>----------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>JNAME</td>
<td>JNAME</td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KOMSTR</td>
<td>KOMSTR</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEADZERO</td>
<td>BOOLEAN</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td>LENGTH</td>
<td>SF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINPACK</td>
<td></td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGO</td>
<td>LGO</td>
<td></td>
<td>LGO</td>
<td>$LGO</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Loads an absolute program from a local dataset containing the binary image as the first record</td>
</tr>
<tr>
<td>LOADF</td>
<td>LOADF</td>
<td></td>
<td>$STOREF</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Performs run time array bounds checking (Performed by all LOADF routines)</td>
</tr>
<tr>
<td>LOC</td>
<td>LOC</td>
<td></td>
<td>SF</td>
<td></td>
<td>$FTLIB</td>
<td>Ind.</td>
<td></td>
<td>Returns first word address of argument</td>
</tr>
<tr>
<td>LOCKASGN</td>
<td>LOCKASGN</td>
<td></td>
<td>SF</td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Identifies lock</td>
</tr>
</tbody>
</table>

SR-0014 2-48
<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard Address</th>
<th>Call From CAL Only Value</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKOFF</td>
<td>LOCKOFF</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Tests lock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCKON</td>
<td>LOCKON</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Tests lock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCKREL</td>
<td>LOCKREL</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Tests lock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG</td>
<td>ALOG</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes natural logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALOGV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vector natural logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALOG10</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes common logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALOG10V</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vector common logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLOG</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes complex logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLOGV</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vector complex logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>LOG</td>
<td>DLOG</td>
<td>LOG</td>
<td>DLOG</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes double-precision natural logarithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLOGV</td>
<td>DLOGV</td>
<td>%DLOG%</td>
<td>%DLOG</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vector double-precision natural logarithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DLOG</td>
<td>DLOG10</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes double-precision common logarithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DLOGV</td>
<td>%DLOG10%</td>
<td>%DLOG10</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vector double-precision common logarithm</td>
</tr>
<tr>
<td>LOGECCHO</td>
<td>LOGECCHO</td>
<td>LOGECCHO</td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Writes last line formatted by $WFD as a message to $LOG file</td>
<td></td>
</tr>
<tr>
<td>MASK</td>
<td>BOOLEAN</td>
<td>MASK</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Forms ones mask from left of argument bits if $0&lt;arg&lt;63$; forms ones mask from right of $(128-argument)$ bits if $64&lt;arg&lt;128$.</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard Address</td>
<td>Call From CAL Only Value</td>
<td>CPT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>MEM</td>
<td>TM</td>
<td>$MEM</td>
<td>$SYSLIB Dep.</td>
<td>Requests memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMORY</td>
<td>MEM</td>
<td>MEMORY</td>
<td>S $SYSLIB Dep.</td>
<td>Determines or changes the amount of memory assigned to a job</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINV</td>
<td>MINV</td>
<td>MINV</td>
<td>S $SCILIB Ind.</td>
<td>Solves linear equations, using a partial pivot search and Gauss-Jordan reduction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>DMOD</td>
<td>DMOD%</td>
<td>SF $ARLIB Ind.</td>
<td>Performs double-precision modulo arithmetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DMODV</td>
<td>%DMOD%</td>
<td>%DMOD VF $ARLIB Ind.</td>
<td>Performs vectorized double-precision modulo arithmetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD continued</td>
<td>LDIVV</td>
<td>MOD%</td>
<td>MOD</td>
<td>MOD%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDSV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit integer division, scalar/vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDVS%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit integer division, vector/scalar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDVV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit integer division, vector/vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODSS%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit integer division, scalar/scalar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%MOD%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Same as MOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODVV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit modulo arithmetic on two integer vectors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODSV%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Same as %MOD%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODVS%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Performs 64-bit modulo arithmetic on integer scalar and integer vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>CFT Standard Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>MODIFY</td>
<td>MFY</td>
<td>MODIFY</td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td>MOVBIT</td>
<td>MOVBIT</td>
<td>MOVBIT</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td>Ind.</td>
</tr>
<tr>
<td>MSC</td>
<td>TM</td>
<td>$MSC</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSIO</td>
<td>RAIO</td>
<td>OPENMS</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITMS</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READMS</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLOSMS</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STINDEX</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FINDMS</td>
<td>S</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
</tr>
<tr>
<td>MVC</td>
<td>MVC</td>
<td>MVC</td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td>Ind.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>MVE</td>
<td>TM</td>
<td>$MVE</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td>Moves memory words to table</td>
<td></td>
</tr>
<tr>
<td>MXM</td>
<td>MXM</td>
<td>S</td>
<td>$SCILIB</td>
<td></td>
<td></td>
<td>Performs matrix multiply with fixed row and column spacing</td>
<td></td>
</tr>
<tr>
<td>MXMA</td>
<td>MXMA</td>
<td>S</td>
<td>$SCILIB</td>
<td></td>
<td></td>
<td>Performs matrix multiply with arbitrary row and column spacing</td>
<td></td>
</tr>
<tr>
<td>MXV</td>
<td>MXV</td>
<td>S</td>
<td>$SCILIB</td>
<td></td>
<td></td>
<td>Performs matrix-vector multiply with fixed row and column spacing</td>
<td></td>
</tr>
<tr>
<td>MXVA</td>
<td>MXVA</td>
<td>S</td>
<td>$SCILIB</td>
<td></td>
<td></td>
<td>Performs matrix-vector multiply with arbitrary row and column spacing</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>ENTRY TYPE</th>
<th>CFT Standard Address</th>
<th>Value</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACSED</td>
<td>PDDED</td>
<td>NACSED</td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td>Returns edition number of last dataset that was accessed, acquired, or saved</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>RNL</td>
<td>RNL</td>
<td>$RNL</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Reads NAMELIST input</td>
</tr>
<tr>
<td></td>
<td>RNLSKIP</td>
<td>RNLSKIP</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Determines action for wrong NAMELIST group</td>
</tr>
<tr>
<td></td>
<td>RNLTYPE</td>
<td>RNLTYPE</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Determines action for NAMELIST type mismatch across equal sign</td>
</tr>
<tr>
<td></td>
<td>RNLECHO</td>
<td>RNLECHO</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Specifies unit for NAMELIST error messages and input echo</td>
</tr>
<tr>
<td></td>
<td>RNLFLAG</td>
<td>RNLFLAG</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Adds or removes NAMELIST echo-initiating characters</td>
</tr>
<tr>
<td></td>
<td>RNLDELM</td>
<td>RNLDELM</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Adds or removes NAMELIST group name delimiting character</td>
</tr>
<tr>
<td></td>
<td>RNLSEP</td>
<td>RNLSEP</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Adds or removes NAMELIST separator character</td>
</tr>
<tr>
<td></td>
<td>RNLREP</td>
<td>RNLREP</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Adds or removes NAMELIST replacement character</td>
</tr>
<tr>
<td></td>
<td>RNLCOMM</td>
<td>RNLCOMM</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td>Adds or removes NAMELIST trailing comment characters</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard Address</td>
<td>Call From CAL Only Value</td>
<td>CPT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------------------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>NAMELIST continued</td>
<td>WNL</td>
<td>$WNL</td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Writes NAMELIST output</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WNLLONG</td>
<td></td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Specifies NAMELIST output line length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WNLDELM</td>
<td></td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Specifies NAMELIST delimiting character</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WNLSEP</td>
<td></td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Specifies NAMELIST separator character</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WNLREP</td>
<td></td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Specifies NAMELIST replacement character</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WNLFLAG</td>
<td></td>
<td>S $IOLIB</td>
<td>Ind.</td>
<td>Specifies first character of first line</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEQV</td>
<td>BOOLEAN</td>
<td>NEQV</td>
<td>SF $ARLIB</td>
<td>Ind.</td>
<td>Computes logical difference</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NICV</td>
<td>NICV</td>
<td>NICV%</td>
<td>S $UTLIB</td>
<td>Ind.</td>
<td>Converts numeric input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$NICV</td>
<td></td>
<td>S $UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NICONV</td>
<td>NICONV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CPT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>NINT</td>
<td>INTEGER</td>
<td>NINT</td>
<td></td>
<td></td>
<td>SA</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Calculates nearest integer</td>
<td></td>
</tr>
<tr>
<td>NOCV</td>
<td>NOCV</td>
<td>NOCONV</td>
<td>NOCV%</td>
<td>$NOCV</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts numeric output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NOCONV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORERUN</td>
<td>RERUN</td>
<td>NORERUN</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Controls monitoring of conditions causing job to be flagged as not rerunnable</td>
<td></td>
</tr>
<tr>
<td>NUMBLKS</td>
<td>NUMBLKS</td>
<td>NUMBLKS</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Returns current size of dataset in 512-word blocks</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>SYMDBC</td>
<td>OPEN</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Opens a random, unblocked dataset. 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.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>OPEN</td>
<td>$OPN</td>
<td>SA</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>BOOLEAN</td>
<td>OR</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Forms logical sum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORDERS</td>
<td>ORDERS</td>
<td>ORDERS</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Internally sorts fixed-length records</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTB</td>
<td>OTB</td>
<td>$OTB</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Converts octal ASCII number to binary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVERLAY</td>
<td>OVERLAY</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td>Processes overlays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P32</td>
<td>P32</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Packs 32-bit words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6460</td>
<td>P6460</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Packs 60-bit words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACK</td>
<td>PACK</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Packs a specified number of words into a packed list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAUSE</td>
<td>STOP</td>
<td>$PAUSE</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Suspends program execution or terminates job step; installation dependent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>PBN</td>
<td>PBN</td>
<td>primary</td>
<td>$APBN</td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Asynchronously positions dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$PBN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Synchronously positions dataset</td>
<td></td>
</tr>
<tr>
<td>PERF</td>
<td>PERF</td>
<td>primary</td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
<td>Hardware performance monitor</td>
<td></td>
</tr>
<tr>
<td>PERMIT</td>
<td>PER</td>
<td>primary</td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td>Dep.</td>
<td>Allows the owner of a permanent dataset to control the manner in which other users can use the dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Processes keywords for a given directive</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CPT Standard Address</th>
<th>Call From CAL Only</th>
<th>Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPCNT</td>
<td>BOOLEAN</td>
<td>POPCNT</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Tallies number of bits set in a word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POPPAR</td>
<td>BOOLEAN</td>
<td>POPPAR</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Parity of number of bits set in a word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCW</td>
<td>PRCW</td>
<td>$PRCW</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Positions dataset after an RCW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTS</td>
<td>TM</td>
<td>$PTS%</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Presets table space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUTBYT</td>
<td>PUTBYT</td>
<td>PUTBYT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Stores byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>RANDOM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANF</td>
<td></td>
<td></td>
<td>RANF</td>
<td>RANF%</td>
<td>$ARLIB</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>RANFV</td>
<td></td>
<td></td>
<td>%RANF%</td>
<td>RANF</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>RANGET</td>
<td></td>
<td></td>
<td>RANGET</td>
<td>RANGET%</td>
<td>$ARLIB</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>RANSET</td>
<td></td>
<td></td>
<td>RANSET</td>
<td>RANSET%</td>
<td></td>
<td>S</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>RANFI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANSED</td>
<td></td>
<td>RANFI</td>
<td></td>
<td></td>
<td>$ARLIB</td>
<td></td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>RANFS</td>
<td></td>
<td>RANFS</td>
<td></td>
<td></td>
<td>$ARLIB</td>
<td></td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>---------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBN</td>
<td>RBN</td>
<td>RBN</td>
<td>$RBN</td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Ind. Replaces trailing blanks with nulls</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCW</td>
<td>RCW</td>
<td></td>
<td>$RCHP</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads characters in partial record mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RCHR</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads characters in record mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RCWP</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads words in partial record mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RCWR</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads words in record mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDIN</td>
<td>SYMDBC</td>
<td>RDIN</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads a buffer of data from a random, unblocked dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDYQUE</td>
<td>RDYQUE</td>
<td></td>
<td>$RDYQUE%</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Readies a queue of tasks for execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDYTSK</td>
<td>RDYTSK</td>
<td></td>
<td>$RDYTSK%</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Readies a task for execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>READ</td>
<td>READ</td>
<td>READ</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Reads words, full record mode</td>
</tr>
<tr>
<td>READC</td>
<td>READC</td>
<td>READC</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Reads characters, full record mode</td>
</tr>
<tr>
<td>READCP</td>
<td>READCP</td>
<td>READCP</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Reads characters, partial record mode</td>
</tr>
<tr>
<td>READIBM</td>
<td>READIBM</td>
<td>READIBM</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Reads two IBM 32-bit floating-point words from each Cray 64-bit word</td>
</tr>
<tr>
<td>READP</td>
<td>READP</td>
<td>READP</td>
<td>S</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Reads words, partial record mode</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>Type</td>
<td>Reference</td>
<td>Call Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------</td>
<td>--------------------</td>
<td>------</td>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>REAL</td>
<td>COMPLX</td>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td></td>
</tr>
<tr>
<td>RELEASE</td>
<td>RLS</td>
<td>RELEASE</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
</tr>
<tr>
<td>REMARK</td>
<td>REMARK</td>
<td>REMARK</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMARK</td>
<td>REMARK2</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMARK</td>
<td>REMARKF</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td></td>
</tr>
<tr>
<td>REPRIEVE</td>
<td>RPV</td>
<td>SETRPV</td>
<td></td>
<td>$SETRPV</td>
<td></td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENDRPV</td>
<td></td>
<td>$ENDRPV</td>
<td></td>
<td>S</td>
<td>$SYSLIB</td>
<td></td>
</tr>
</tbody>
</table>

SR-0014  
2-65  
I
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RERUN</td>
<td>RERUN</td>
<td>RERUN</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Allows user to set job rerunnability status</td>
</tr>
<tr>
<td>REWD</td>
<td>WCW</td>
<td>$REWD</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Rewinds dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFD</td>
<td>RFD</td>
<td>$RFI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initializes for formatted read</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RFV $RFV%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Cracks format; reads input.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RFA</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Vectorizes formatted read</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RFF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Terminates formatted read</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RCHK</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Read formatted, check;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$RNOCHK</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Read formatted, no check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLB</td>
<td>RLB</td>
<td>$RLB</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Reads data directly from user area</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard</th>
<th>Call From CAL Only</th>
<th>CFT Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLD</td>
<td>RLD</td>
<td></td>
<td>$RLI</td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Reads list-directed input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RLA</td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RLF</td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
</tr>
<tr>
<td>RNB</td>
<td>RNB</td>
<td></td>
<td>$RNB</td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td>Replaces trailing nulls with blanks</td>
</tr>
<tr>
<td>ROT</td>
<td>CROT</td>
<td>CROT</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Applies complex Givens plane rotation</td>
</tr>
<tr>
<td></td>
<td>CROTG</td>
<td>CROTG</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Constructs complex Givens plane rotation</td>
</tr>
<tr>
<td></td>
<td>SROT</td>
<td>SROT</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Performs Givens plane rotation</td>
</tr>
<tr>
<td></td>
<td>SROTG</td>
<td>SROTG</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Constructs Givens plane rotation</td>
</tr>
<tr>
<td></td>
<td>SROTM</td>
<td>SROTM</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Performs modified Givens plane rotation</td>
</tr>
<tr>
<td></td>
<td>SROTMG</td>
<td>SROTMG</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td></td>
<td>Calculates a rotation matrix as parameters for modified Givens plane rotation SROTM</td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Address</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTOI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTOISS</td>
<td></td>
<td></td>
<td>RTOI%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real scalar to integer scalar power</td>
</tr>
<tr>
<td>RTOIVS</td>
<td></td>
<td></td>
<td>%RTOI%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real vector to integer scalar power</td>
</tr>
<tr>
<td>RTOIVV</td>
<td></td>
<td></td>
<td>RTO%I%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real scalar to integer vector power</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%RTO%I%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real vector to integer vector power</td>
</tr>
<tr>
<td>RTOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTORSS</td>
<td></td>
<td></td>
<td>RTOR%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real scalar to real scalar power</td>
</tr>
<tr>
<td>RTORSV</td>
<td></td>
<td></td>
<td>RTO%R%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real scalar to real vector power</td>
</tr>
<tr>
<td>RTORVS</td>
<td></td>
<td></td>
<td>%RTOR%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real vector to real scalar power</td>
</tr>
<tr>
<td>RTORVV</td>
<td></td>
<td></td>
<td>%RRO%R%</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Raises real vector to real vector power</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Call Address</td>
<td>Call From CAL Only Type Value</td>
<td>CFT Call Type Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>-----------------------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td>RU</td>
<td></td>
<td>$RUI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initializes for unformatted read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RUV $RUV%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Reads unformatted input</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RUA</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Reads unformatted input into a vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%$RUV%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Terminates unformatted read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$RUF</td>
<td>$IOLIB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUT</td>
<td>RUT</td>
<td></td>
<td>RUTI%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initializes foreign dataset read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RUTD%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Reads foreign dataset, full record mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RUTDP%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Reads foreign dataset, partial record mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RUTF%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Terminates foreign dataset read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FDPOS%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Gets foreign dataset position</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FDSPOS%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Sets foreign dataset position</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FDBKSP%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Backspaces foreign dataset</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard Address</th>
<th>Value</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE</td>
<td>SVS</td>
<td>SAVE</td>
<td>S SYSLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Makes a dataset permanent</td>
</tr>
<tr>
<td>SCAL</td>
<td>CSCAL</td>
<td>CSCAL</td>
<td>S SCILIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Scales a complex array by a complex factor</td>
</tr>
<tr>
<td></td>
<td>CSSCAL</td>
<td>CSSCAL</td>
<td>S SCILIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Scales a complex array by a real factor</td>
</tr>
<tr>
<td></td>
<td>SSCAL</td>
<td>SSCAL</td>
<td>S SCILIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Scales a real array by a real factor</td>
</tr>
<tr>
<td>SCATTER</td>
<td>SCATTER</td>
<td>SCATTER</td>
<td>S SCILIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Scatters a vector into another vector</td>
</tr>
<tr>
<td>SCERP</td>
<td>SCERP</td>
<td>SCERP</td>
<td>SCERP</td>
<td>S</td>
<td></td>
<td>S SCILIB</td>
<td></td>
<td></td>
<td>Processes $SCILIB errors; issues logfile error message, then aborts with traceback.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>Entry Type</td>
<td>Call From</td>
<td>CFT Standard</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCHED</td>
<td>UPDATE Deck Name</td>
<td>CAL Only</td>
<td>$SCHED%</td>
<td>$UTLIB Dep. Schedules logical CPUs for user tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCNRM2</td>
<td>SCNRM2</td>
<td>SF</td>
<td>$SCILIB Ind. Calculates the Euclidean norm ((l_2)) of a complex array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDACCESS</td>
<td>SDACC</td>
<td>S</td>
<td>$SYSLIB Dep. Allows a FORTRAN program to access system datasets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDSP</td>
<td>SDSP</td>
<td>$SDSP</td>
<td>$SYSLIB Dep. Searches Dataset Parameter Tables for a dataset name and returns DSP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEARCH</td>
<td>OSRCHI</td>
<td>S</td>
<td>$SCILIB Ind. Searches an ordered array for a target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OSRCHF</td>
<td>S</td>
<td>$SCILIB Ind.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1. Subprogram summary (continued)
<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Call From CAL Only</th>
<th>Primary Type</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECOND</td>
<td>SECOND</td>
<td>SECOND</td>
<td>S/ST</td>
<td>$ULID</td>
<td>Ind.</td>
<td></td>
<td>Returns time since start of job in floating-point seconds</td>
</tr>
<tr>
<td>SEGRES</td>
<td>SEGRES</td>
<td>$SEGRES $SEGCALL</td>
<td></td>
<td>$ULID</td>
<td>Dep.</td>
<td></td>
<td>Initializes execution of a segmented program and services intersegment subroutine calls</td>
</tr>
<tr>
<td>SENSEBT</td>
<td>BTMODE</td>
<td>SENSEBT</td>
<td>S</td>
<td>$ULID</td>
<td>Dep.</td>
<td></td>
<td>Returns mode indicating bidirectional memory transfers are enabled or disabled</td>
</tr>
<tr>
<td>SENSEFI</td>
<td>FIMODE</td>
<td>SENSEFI</td>
<td>S</td>
<td>$ULID</td>
<td>Dep.</td>
<td></td>
<td>Returns mode indicating floating-point interrupts permitted or prohibited</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETBT</td>
<td>BTMODE</td>
<td>SETBT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETBTS</td>
<td>BTMODE</td>
<td>SETBTS</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETFI</td>
<td>FIMODE</td>
<td>SETFI</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETFIS</td>
<td>FIMODE</td>
<td>SETFIS</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPLIMQ</td>
<td>FLOW</td>
<td>SETPLIMQ</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From Library</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETPOS</td>
<td>SETPOS</td>
<td>S</td>
<td>$IOLIB</td>
<td>Set dataset position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPOS</td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STPOS</td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td></td>
<td>ASPOS</td>
<td>Dep. Asynchronously positions current dataset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td></td>
<td>$FSPOS</td>
<td>$SYSLIB</td>
<td>Finishes the asynchronous dataset positioning request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$GPOS</td>
<td>$SYSLIB</td>
<td>Returns current dataset position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SPOS</td>
<td>$SYSLIB</td>
<td>Positions dataset to a word address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STPOS%</td>
<td>$SYSLIB</td>
<td>Positions tape dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SETPOS</td>
<td>$IOLIB</td>
<td>Returns current dataset position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF</td>
<td>$SYSLIB</td>
<td>Positions dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFN</td>
<td>RNB</td>
<td>$RNB</td>
<td>SF</td>
<td>Replaces trailing nulls with blanks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>BOOLEAN</td>
<td>SHIFT</td>
<td>SF</td>
<td>Shifts left circularly</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>Entry Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------</td>
<td>--------------------</td>
<td>------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>SHIFTL</td>
<td>BOOLEAN</td>
<td>SHIFTL</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>SHIFTR</td>
<td>BOOLEAN</td>
<td>SHIFTR</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>REAL</td>
<td>SIGN</td>
<td></td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>COS</td>
<td>SIN</td>
<td>SIN%</td>
<td>%SIN</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>COSV</td>
<td>SIN%</td>
<td>%SIN</td>
<td>%SIN</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>SINH</td>
<td>COSH</td>
<td>SINH</td>
<td>SINH%</td>
<td>%SINH</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>COSHV</td>
<td>SINH%</td>
<td>%SINH</td>
<td>%SINH</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>SKIP</td>
<td>SKIP</td>
<td>SKIPR</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Bypasses a specified number of records in a dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SKIPF</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Bypasses a specified number of files in a dataset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SKIPD</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Positions a dataset at end of data</td>
<td></td>
</tr>
<tr>
<td>SKIPBAD</td>
<td>RCVRBAD</td>
<td>SKIPBAD</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Skips to the first good data encountered</td>
<td></td>
</tr>
<tr>
<td>SKIPU</td>
<td>SKIPU</td>
<td>SKIPU</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Skips sectors on unblocked dataset</td>
<td></td>
</tr>
<tr>
<td>SLERP</td>
<td>SLERP</td>
<td>-</td>
<td>SLERP%</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td>Processes $SYSLIB errors; aborts with traceback.</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>SLFT</td>
<td>SDSP</td>
<td>$SLFT</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Searches Logical File Table for dataset name and returns LFT address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMACH</td>
<td>SMACH</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Real function of an integer argument that returns Cray machine constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMACH</td>
<td>CMACH</td>
<td>SF</td>
<td>$SCLIB</td>
<td>Ind.</td>
<td>Computes complex Cray constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNAP</td>
<td>SNAP</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Prints current register contents on $OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNRM2</td>
<td>SNRM2</td>
<td>SF</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Calculates Euclidean norm ($L_2$) of a real array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLR</td>
<td>SOLR</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Solves second-order linear recurrences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>CFT Standard Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>SOLRN</td>
<td>SOLRN</td>
<td>Primary UPDATE</td>
<td>SOLRN</td>
<td>SOLRN</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>SOLR3</td>
<td>SOLR3</td>
<td>CFT</td>
<td>SOLR3</td>
<td>SOLR3</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>SQRT</td>
<td>CSQRT</td>
<td>CSQRT%</td>
<td>CSQRT</td>
<td>CSQRT%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>CSQRTV</td>
<td>%CSQRT%</td>
<td>%CSQRT</td>
<td>%CSQRT</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>DSQRT</td>
<td>DSQRT%</td>
<td>DSQRT</td>
<td>DSQRT%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>DSQRTV</td>
<td>%DSQRT%</td>
<td>%DSQRT</td>
<td>%DSQRT</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>SQRT</td>
<td>SQRT%</td>
<td>SQRT</td>
<td>SQRT%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td></td>
<td>SQRTV</td>
<td>%SQRT%</td>
<td>%SQRT</td>
<td>%SQRT</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>SRC</td>
<td>TM</td>
<td>$SRC</td>
<td>$SYSLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSWITCH</td>
<td>SSWITCH SSWITCH</td>
<td>S/SF</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STACK</td>
<td>STACKAL</td>
<td>$STKOPEN</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$STKCR</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$STKUFEX</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STACKDE</td>
<td>$STKDE</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$STKUFCK</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>STOP</td>
<td>$STOP</td>
<td>$UTLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

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

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Address</th>
<th>Call From CAL Only Value</th>
<th>Entry Type</th>
<th>CFT Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAP</td>
<td>CSWAP</td>
<td>CSWAP</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Exchanges specified elements of complex arrays</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSWAP</td>
<td>SSWAP</td>
<td>S</td>
<td>$SCILIB</td>
<td>Ind.</td>
<td>Exchanges specified elements of two real arrays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYMDEBUG</td>
<td>SYMDEBUG</td>
<td>SYMDEBUG</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Produces a symbolic dump</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEADBUG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNCH</td>
<td>SYNCH</td>
<td>SYNCH%</td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Synchronizes tape dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE MANAGER</td>
<td>TM</td>
<td>TMADW</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Adds a word to a table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMAMU</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Reports TMGR statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMATS</td>
<td>SF</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Allocates table space</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMINIT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Initializes managed tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>Update Deck Name</td>
<td>CFT Standard Address</td>
<td>CFT Standard Value</td>
<td>Call From</td>
<td>Call Only CAL</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>TABLE MANAGER continued</td>
<td>TM</td>
<td>TMMEM</td>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Requests memory</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMMSC</td>
<td>SF</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Searches table with mask</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMVSC</td>
<td>SF</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Moves words</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMMVE</td>
<td>S</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Presets table space</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMSRC</td>
<td>SF</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Searches table</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>TMVSC</td>
<td>SF</td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Dep.</td>
<td>Searches vector table</td>
</tr>
<tr>
<td>TABLES</td>
<td>NCON</td>
<td>$IIN</td>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Table of integer powers of ten</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IMX</td>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Table of negative real powers of ten</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IPX</td>
<td></td>
<td></td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Table of positive real powers of ten</td>
</tr>
<tr>
<td></td>
<td>PDD</td>
<td>PDD</td>
<td></td>
<td></td>
<td></td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Table of permanent dataset definitions</td>
</tr>
<tr>
<td></td>
<td>RANSED</td>
<td>RANFI</td>
<td></td>
<td></td>
<td></td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Table for random number generator</td>
</tr>
<tr>
<td>TADD</td>
<td>TPREC</td>
<td>TADD</td>
<td>TASS</td>
<td></td>
<td></td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs triple-precision addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$TADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TASS$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>TAN</td>
<td>COT</td>
<td>TAN</td>
<td>TAN%</td>
<td>%TAN</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes tangent</td>
</tr>
<tr>
<td></td>
<td>COTV</td>
<td></td>
<td></td>
<td></td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Computes vectorized tangent</td>
</tr>
<tr>
<td>TANH</td>
<td>TANH</td>
<td>TANH</td>
<td>TANH%</td>
<td>%TANH</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Calculates hyperbolic tangent</td>
</tr>
<tr>
<td></td>
<td>TANHV</td>
<td></td>
<td>%TANH</td>
<td>%TANH</td>
<td>VF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Calculates vectorized hyperbolic tangent</td>
</tr>
<tr>
<td>TDIV</td>
<td>TDSS</td>
<td>TDIV</td>
<td>TDSS</td>
<td>TDSS%</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td>Performs triple-precision division</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$TDSS</td>
<td>$TDIV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCR</td>
<td>TIBCR</td>
<td></td>
<td>$TIBCR%</td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Builds task information block</td>
<td></td>
</tr>
<tr>
<td>TIBDE</td>
<td>TIBDE</td>
<td></td>
<td>$TIBDE%</td>
<td></td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Deletes task information block</td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entry Type</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTTS</td>
<td>DTTS</td>
<td>$S$</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td>Converts from date and time to timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MTTS</td>
<td>MTTS</td>
<td>$SF$</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td>Converts from a real-time value to the corresponding timestamp value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSDT</td>
<td>TSDT</td>
<td>$S$</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td>Converts between timestamps and the date and time as ASCII strings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSMT</td>
<td>TSMT</td>
<td>$SF$</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td>Converts from a timestamp to the corresponding real-time clock value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNITTS</td>
<td>UNITTS</td>
<td>$SF$</td>
<td>$SYSLIB$</td>
<td>Dep.</td>
<td>Returns the number of timestamp units in a specified number of standard time units</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMLT</td>
<td></td>
<td>TMLT</td>
<td>TMSS</td>
<td>$ARLIB$</td>
<td>Ind.</td>
<td>Performs triple-precision multiplication</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TMSS% $TMLT$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td></td>
<td>TR</td>
<td>$S$</td>
<td>$UTLIB$</td>
<td>Ind.</td>
<td>Translates characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>TRBK</td>
<td>TRBK</td>
<td>primary</td>
<td>$TRBK</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Prints a list of subroutines showing the path from the main program to the current subprogram</td>
</tr>
<tr>
<td></td>
<td>TRBK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRBKLV</td>
<td></td>
<td>TRBKLV%</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Aids the traceback mechanism by returning information for the current level</td>
</tr>
<tr>
<td></td>
<td>TRBKLV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tremain</td>
<td>Tremain</td>
<td></td>
<td>Tremain</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Returns time remaining for job execution in floating-point seconds</td>
</tr>
<tr>
<td>TSKSTART</td>
<td>TSKSTART</td>
<td></td>
<td>TSKSTART</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Initiates a task</td>
</tr>
<tr>
<td>TSKTEST</td>
<td>TSKTEST</td>
<td></td>
<td>TSKTEST</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td>Returns a value indicating whether the indicated task exists</td>
</tr>
</tbody>
</table>

Table 2-1. Subprogram summary (continued)
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Address</th>
<th>CFT Standard Value</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSKTUNE</td>
<td>TSKTUNE</td>
<td>TSKTUNE</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Modifies scheduling parameters used by a multitasking subroutine</td>
</tr>
<tr>
<td>TSKVALUE</td>
<td>TSKVALUE</td>
<td>TSKVALUE</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Retrieves user id specified in the task control array used to create the executing task</td>
</tr>
<tr>
<td>TSKWAIT</td>
<td>TSKWAIT</td>
<td>TSKWAIT</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Waits for the indicated task to complete execution</td>
</tr>
<tr>
<td>TSUB</td>
<td>TSUB</td>
<td>TSUB</td>
<td>TSSS% TSSS $TSUB</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Performs triple-precision subtraction</td>
</tr>
<tr>
<td>U32</td>
<td>U32</td>
<td>U32</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td></td>
<td></td>
<td>Unpacks 32-bit words</td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>Entry Type</th>
<th>CFT Standard Address</th>
<th>Call From CAL Only Value</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>U6064</td>
<td>U6064</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Unpacks 60-bit words</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UEOF</td>
<td>UEOFCL</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td>Sets the uncleared end-of-file flag in DSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$UEOFCL</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$UEOFKIL</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$UEOFSET</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT</td>
<td>UNITLB</td>
<td>SF</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Waits for buffer I/O completion and returns status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNPACK</td>
<td>UNPACK</td>
<td>S</td>
<td>$SYSLIB</td>
<td>Ind.</td>
<td>Expands full words of data into a larger number of partial words</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USCCTC</td>
<td>USCCTC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts EBCDIC character to ASCII character</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>Value</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>USCCTI</td>
<td>USCCTI</td>
<td>USCCTI</td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts ASCII character to EBCDIC character</td>
</tr>
<tr>
<td>USICTP</td>
<td>USICTP</td>
<td>USICTP</td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts integer to IBM packed decimal field</td>
</tr>
<tr>
<td>USDCTC</td>
<td>USDCTC</td>
<td>USDCTC</td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts IBM double-precision to Cray double-precision</td>
</tr>
<tr>
<td>USDCTI</td>
<td>USDCTI</td>
<td>USDCTI</td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts Cray single-precision to IBM double-precision</td>
</tr>
<tr>
<td>USICTC</td>
<td>USICTC</td>
<td>USICTC</td>
<td></td>
<td></td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts IBM integer to Cray integer</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard</td>
<td>Call From</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>USICTI</td>
<td>USICTI</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts Cray integer to IBM integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USLCTC</td>
<td>USLCTC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts IBM Logical to Cray Logical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USLCTI</td>
<td>USLCTI</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts Cray logical to IBM logical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USPCTC</td>
<td>USPCTC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts IBM packed decimal field to integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USSCTC</td>
<td>USSCTC</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind.</td>
<td>Converts IBM single-precision to Cray single-precision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard Address</td>
<td>CFT Standard Value</td>
<td>Call From CAL Only</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>-------------------</td>
<td>------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USSCTI</td>
<td>USSCTI</td>
<td>USSCTI</td>
<td>S</td>
<td>$UTLIB</td>
<td>Ind. Converts Cray single-precision to IBM single-precision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTERP</td>
<td>UTERP</td>
<td>UTERP%</td>
<td>$UTLIB</td>
<td>Dep. Processes $UTLIB errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIO</td>
<td>WAIO</td>
<td>WOPEN</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep. Opens a dataset on a disk and specifies dataset as word-addressable, random access</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTWA</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep. Writes a number of words from memory to a word-addressable, random access dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GETWA</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep. Reads a number of words from a word-addressable, random access dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WCLOSE</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep. Closes a word-addressable, random access dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEEK</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep. Allows user to asynchronously read data into specified dataset buffers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>Update Deck Name</th>
<th>CFT Standard Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WB</td>
<td>WB</td>
<td>$WB</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Initiates buffered output</td>
</tr>
<tr>
<td>WB</td>
<td>WB</td>
<td>$WB</td>
<td>S</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Writes characters in partial record mode</td>
</tr>
<tr>
<td>WC</td>
<td>WCH</td>
<td>$WCHP</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Writes characters in record mode</td>
</tr>
<tr>
<td>WC</td>
<td>WCH</td>
<td>$WCHR</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WE</td>
<td>WWD</td>
<td>$WEOD</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Writes EOF and/or EOR, if necessary</td>
</tr>
<tr>
<td>WE</td>
<td>WWD</td>
<td>$WEOD</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Writes EOD; writes EOF and/or EOR, if necessary</td>
</tr>
<tr>
<td>WFBUFFER</td>
<td>WFBUFFER</td>
<td>WFBUFFER</td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td>Loads a character from $WFD's buffer</td>
</tr>
<tr>
<td>WF</td>
<td>WFD</td>
<td>$WFI</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td>Initializes for formatted output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$WFV $WFV% $WFV%</td>
<td>$IOLIB</td>
<td>Dep.</td>
<td></td>
<td>Cracks format; writes output.</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>CFT Standard Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>WF continued</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$WFA</td>
<td></td>
<td>$IOLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$WFF</td>
<td></td>
<td>$IOLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$WCHK</td>
<td></td>
<td>$IOLIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$WNOCHK</td>
<td></td>
<td>$IOLIB</td>
<td></td>
</tr>
<tr>
<td>WHEN</td>
<td></td>
<td></td>
<td>$SCILIB</td>
<td>S</td>
<td></td>
<td>Ind.</td>
</tr>
<tr>
<td>WHENEQ</td>
<td></td>
<td></td>
<td>WHENEQ</td>
<td>WHENEQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENNE</td>
<td></td>
<td></td>
<td>WHENNE</td>
<td>WHENNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENFLT</td>
<td></td>
<td></td>
<td>WHENFLT</td>
<td>WHENFLT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENFLE</td>
<td></td>
<td></td>
<td>WHENFLE</td>
<td>WHENFLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENFGT</td>
<td></td>
<td></td>
<td>WHENFGT</td>
<td>WHENFGT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENPGE</td>
<td></td>
<td></td>
<td>WHENPGE</td>
<td>WHENPGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENE EQ</td>
<td></td>
<td></td>
<td>WHENE EQ</td>
<td>WHENE EQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENE NE</td>
<td></td>
<td></td>
<td>WHENE NE</td>
<td>WHENE NE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENE LT</td>
<td></td>
<td></td>
<td>WHENE LT</td>
<td>WHENE LT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENI LE</td>
<td></td>
<td></td>
<td>WHENI LE</td>
<td>WHENI LE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENI GT</td>
<td></td>
<td></td>
<td>WHENI GT</td>
<td>WHENI GT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHENI GE</td>
<td></td>
<td></td>
<td>WHENI GE</td>
<td>WHENI GE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLB</td>
<td></td>
<td></td>
<td>$WLB</td>
<td></td>
<td>$SYSLIB</td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-1. Subprogram summary (continued)

<table>
<thead>
<tr>
<th>Primary Reference Name</th>
<th>UPDATE Deck Name</th>
<th>CFT Standard Address</th>
<th>Value</th>
<th>Call From CAL Only</th>
<th>CFT Call Type</th>
<th>Library</th>
<th>OS Dep</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLD</td>
<td>WLD</td>
<td>$WLI $WLA $WLF</td>
<td></td>
<td></td>
<td></td>
<td>$IOLIB</td>
<td>Dep.</td>
<td>Writes list-directed data</td>
</tr>
<tr>
<td>WRITE</td>
<td>WRITE</td>
<td>WRITE</td>
<td>S</td>
<td></td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td>Writes words, full record mode</td>
</tr>
<tr>
<td>WRITEC</td>
<td>WRITEC</td>
<td>WRITEC</td>
<td>S</td>
<td></td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td>Writes characters, full record mode</td>
</tr>
<tr>
<td>WRITECP</td>
<td>WRITECP</td>
<td>WRITECP</td>
<td>S</td>
<td></td>
<td>$IOLIB</td>
<td>Ind.</td>
<td></td>
<td>Writes characters, partial record mode</td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>Entry Type</td>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITIBM</td>
<td>WRITIBM</td>
<td>WRITIBM</td>
<td><strong>WU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$WUI$</td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$WUV$ $WUVV$</td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$WUA$</td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$WUF$</td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WUT</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WUTI%</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WUTD%</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WUTDP%</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WUTF%</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>FDWEOF%</strong></td>
<td>$IOLIB$</td>
<td>Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Reference Name</td>
<td>UPDATE Deck Name</td>
<td>CFT Standard</td>
<td>Call From CAL Only</td>
<td>CFT Call Type</td>
<td>Library</td>
<td>OS Dep</td>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>WW</td>
<td>WCW</td>
<td>$WWDP</td>
<td>$WWPU</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Writes words in partial mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$WWDR</td>
<td>$WWDS</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Writes words in partial record mode with unused bit count</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>BOOLEAN</td>
<td>XOR</td>
<td>SF</td>
<td>$ARLIB</td>
<td>Ind.</td>
<td></td>
<td>Forms logical difference</td>
<td></td>
</tr>
<tr>
<td>2TS</td>
<td>TM</td>
<td>$2TS%</td>
<td>$SYSLIB</td>
<td>Dep.</td>
<td></td>
<td>Clears table space</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-2. Pascal subprogram summary

<table>
<thead>
<tr>
<th>Primary Ref. and UPDATE Deck Name</th>
<th>Pascal Call Standard</th>
<th>Call from CAL only (X)</th>
<th>Dep.</th>
<th>Call Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>P$$HPAD</td>
<td>P$$HPAD</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>Get address of heap control block</td>
</tr>
<tr>
<td>P$ABORT</td>
<td>P$ABORT</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Abort job step</td>
</tr>
<tr>
<td>P$CALLR</td>
<td>P$CALLR</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>Get name of calling routine</td>
</tr>
<tr>
<td>P$CBV</td>
<td>P$CBV</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Call a call-by-value routine</td>
</tr>
<tr>
<td>P$CONNEC</td>
<td>P$CONNEC</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Set file name</td>
</tr>
<tr>
<td>P$DATE</td>
<td>P$DATE</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Get date</td>
</tr>
<tr>
<td>P$DBP</td>
<td>P$DBP</td>
<td>P$DIVMOD</td>
<td>Ind.</td>
<td>SF</td>
<td>Breakpoint checking</td>
</tr>
<tr>
<td>P$DIST</td>
<td>P$DIST</td>
<td>P$DIVMOD</td>
<td>Ind.</td>
<td>S</td>
<td>Dispose heap area</td>
</tr>
<tr>
<td>P$DIVMOD</td>
<td>P$DIVMOD</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>Integer division</td>
</tr>
<tr>
<td>P$ENDP</td>
<td>P$ENDP</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>End program</td>
</tr>
<tr>
<td>P$EOF</td>
<td>P$EOF</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>End of file check</td>
</tr>
<tr>
<td>P$EOLN</td>
<td>P$EOLN</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>End of line check</td>
</tr>
<tr>
<td>P$GET</td>
<td>P$GET</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Read record</td>
</tr>
<tr>
<td>P$HALT</td>
<td>P$HALT</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Terminate program</td>
</tr>
<tr>
<td>P$JTIME</td>
<td>P$JTIME</td>
<td></td>
<td>Ind.</td>
<td>SF</td>
<td>Get job CPU time</td>
</tr>
<tr>
<td>P$LOGMSG</td>
<td>P$LOGMSG</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Write $LOG message</td>
</tr>
<tr>
<td>P$LSTREW</td>
<td>P$LSTREW</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Rewrite file without rewind</td>
</tr>
<tr>
<td>P$MEMRY</td>
<td>P$MEMRY</td>
<td>P$MOD</td>
<td>Ind.</td>
<td>SF</td>
<td>Memory management</td>
</tr>
<tr>
<td>P$MOD</td>
<td>P$MOD</td>
<td>P$MOD</td>
<td>Ind.</td>
<td>SF</td>
<td>Integer modulus</td>
</tr>
<tr>
<td>P$NEW</td>
<td>P$NEW</td>
<td></td>
<td>Ind.</td>
<td>S</td>
<td>Allocate heap area</td>
</tr>
<tr>
<td>P$OSDBS</td>
<td>P$OSDBS</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Rewind dataset</td>
</tr>
<tr>
<td>P$OSDDT</td>
<td>P$OSDDT</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Get date</td>
</tr>
<tr>
<td>P$OSDEP</td>
<td>P$OSDEP</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Open dataset</td>
</tr>
<tr>
<td>P$OSDJT</td>
<td>P$OSDJT</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Get job CPU time</td>
</tr>
<tr>
<td>P$OSDLM</td>
<td>P$OSDLM</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Write $LOG message</td>
</tr>
<tr>
<td>P$OSDPF</td>
<td>P$OSDPF</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Set prompt string</td>
</tr>
<tr>
<td>P$OSDQI</td>
<td>P$OSDQI</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Query if dataset is interactive</td>
</tr>
<tr>
<td>P$OSDRC</td>
<td>P$OSDRC</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Read characters</td>
</tr>
<tr>
<td>P$OSDRP</td>
<td>P$OSDRP</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Enable reprieve</td>
</tr>
<tr>
<td>P$OSDRW</td>
<td>P$OSDRW</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Read words</td>
</tr>
<tr>
<td>P$OSDTM</td>
<td>P$OSDTM</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Get time of day</td>
</tr>
<tr>
<td>P$OSDWC</td>
<td>P$OSDWC</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Write characters</td>
</tr>
<tr>
<td>P$OSDFW</td>
<td>P$OSDFW</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Write EOF</td>
</tr>
<tr>
<td>P$OSDWR</td>
<td>P$OSDWR</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Write record</td>
</tr>
<tr>
<td>P$OSDXP</td>
<td>P$OSDXP</td>
<td></td>
<td>Dep.</td>
<td>S</td>
<td>Exit program</td>
</tr>
</tbody>
</table>
### Table 2-2. Pascal subprogram summary (continued)

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

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:
\[ \text{arg}_1 \]Address of first argument
\[ \text{arg}_2 \]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:
arg1 First word address of first argument
arg2 Address of first increment
arg3 First word address of second argument (if present)
arg4 Address of second increment (if present)
(VL) Vector length

Exit:
(VL) 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:
(VL) 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 $|x|<10^{2466}$. 

---
Table 3-1. Logarithmic routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments No.</th>
<th>Type</th>
<th>Range</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural log.</td>
<td>ALOG</td>
<td>SA</td>
<td>log(_e)(x) or (\ln(x))</td>
<td>1</td>
<td>R</td>
<td>0&lt;x&lt;(\infty)</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>ALOG%</td>
<td>SVa</td>
<td>ln(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%ALOG</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%ALOG%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common log.</td>
<td>ALOG10</td>
<td>SA</td>
<td>log(_{10})(x)</td>
<td>1</td>
<td>R</td>
<td>0&lt;x&lt;(\infty)</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>ALOG10%</td>
<td>SVa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%ALOG10</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%ALOG10%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex log.</td>
<td>CLOG</td>
<td>SA</td>
<td>ln</td>
<td>z</td>
<td>+ i arctan(y/x)</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>CLOG%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CLOG</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CLOG%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. Natural log.</td>
<td>DLOG</td>
<td>SA</td>
<td>log(_e)(x) or (\ln(x))</td>
<td>1</td>
<td>D</td>
<td>0&lt;x&lt;(\infty)</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DLOG%</td>
<td>SVb</td>
<td>ln(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DLOG</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DLOG%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. Common log.</td>
<td>DLOG10</td>
<td>SA</td>
<td>log(_{10})(x)</td>
<td>1</td>
<td>D</td>
<td>0&lt;x&lt;(\infty)</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DLOG10%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DLOG10</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DLOG10%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3-2. Exponential routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex exponentiation</td>
<td>CEXP</td>
<td>SA</td>
<td>e&lt;sup&gt;x&lt;/sup&gt;cos(y)+ie&lt;sup&gt;x&lt;/sup&gt;sin(y)</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>CEXP%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CEXP</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CEXP%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. exponentia-</td>
<td>DEXP</td>
<td>SA</td>
<td>e&lt;sup&gt;x&lt;/sup&gt;</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>tion</td>
<td>DEXP%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DEXP</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DEXP%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exponentiation</td>
<td>EXP</td>
<td>SA</td>
<td>e&lt;sup&gt;x&lt;/sup&gt;</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>EXP%</td>
<td>SVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%EXP</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%EXP%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3-3. Square root routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex square root</td>
<td>CSQRST</td>
<td>SA</td>
<td>(\sqrt{1/2(</td>
<td>z</td>
<td>+x)}+)</td>
</tr>
<tr>
<td></td>
<td>CSQRST%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CSQRST</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CSQRST%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. square root</td>
<td>DSQRST</td>
<td>SA</td>
<td>(\sqrt{x}) or (x^{1/2})</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DSQRST%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DSQRST</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DSQRST%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square root</td>
<td>SQRT</td>
<td>SA</td>
<td>(\sqrt{x}) or (x^{1/2})</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>SQRT%</td>
<td>SVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%SQRT</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%SQRT%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## TRIGONOMETRIC ROUTINES

### Table 3-4. Trigonometric routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arccosine</td>
<td>ACOS</td>
<td>SA VVa</td>
<td>arccos(x)</td>
<td>1 R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACOS% %ACOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arcsine</td>
<td>ASIN</td>
<td>SA VVa</td>
<td>arcsin(x)</td>
<td>1 R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASIN% %ASIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arctangent</td>
<td>ATAN</td>
<td>SA VVa</td>
<td>arctan(x)</td>
<td>1 R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATAN% %ATAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two-arg.</td>
<td>ATAN2</td>
<td>SA VVa</td>
<td>arctan(x/y)</td>
<td>2 R</td>
<td></td>
</tr>
<tr>
<td>arctangent</td>
<td>ATAN2% %ATAN2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec.</td>
<td>DACOS</td>
<td>SA VVa</td>
<td>arccos(x)</td>
<td>1 D</td>
<td></td>
</tr>
<tr>
<td>arccosine</td>
<td>DACOS% %DACOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec.</td>
<td>DASIN</td>
<td>SA VVa</td>
<td>arcsine(x)</td>
<td>1 D</td>
<td></td>
</tr>
<tr>
<td>arcsine</td>
<td>DASIN% %DASIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Purpose</td>
<td>Entry Names</td>
<td>Call Seq. Code</td>
<td>Definition</td>
<td>Arguments</td>
<td>Func. Value Type</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>----------------</td>
<td>------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Double-prec. arctangent</td>
<td>DATAN</td>
<td>SA</td>
<td>arctan(x)</td>
<td>1 D</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DATAN%</td>
<td>SVb</td>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>%DATAN</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DATAN</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. two-arg. arctan.</td>
<td>DATAN2</td>
<td>SA</td>
<td>arctan(x/y)</td>
<td>2 D</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DATAN2%</td>
<td>SVb</td>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>%DATAN2</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DATAN2</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cosine</td>
<td>COS</td>
<td>SA</td>
<td>cos(x)</td>
<td>1 R</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>%COS</td>
<td>SVa</td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%COS</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COS%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex cosine</td>
<td>CCOS</td>
<td>SA</td>
<td>cos(x)cosh(y) + i sin(x)sinh(y)</td>
<td>1 C</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>CCOS%</td>
<td>SVb</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>%CCOS</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CCOS%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. cosine</td>
<td>DCOS</td>
<td>SA</td>
<td>cos(x)</td>
<td>1 D</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DCOS%</td>
<td>SVb</td>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>%DCOS</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DCOS</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sine</td>
<td>SIN</td>
<td>SA</td>
<td>sin(x)</td>
<td>1 R</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>%SIN</td>
<td>SVa</td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%SIN</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%SIN%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex sine</td>
<td>CSIN</td>
<td>SA</td>
<td>sin(x)cosh(y) + i cos(x)sinh(y)</td>
<td>1 C</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>CSIN%</td>
<td>SVb</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>%CSIN</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CSIN%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3-4. Trigonometric routines (continued)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-prec. sine</td>
<td>DSIN</td>
<td>SA</td>
<td>sin(x)</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DSIN%</td>
<td>SVb</td>
<td></td>
<td>[x</td>
<td>&lt; 2^48</td>
</tr>
<tr>
<td></td>
<td>%DSIN</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DSIN%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cosine and sine†</td>
<td>COSS</td>
<td>SA</td>
<td>y1=cos(x)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>COSS%</td>
<td>SVb</td>
<td>y2=sin(x)</td>
<td>[x</td>
<td>&lt; 2^24</td>
</tr>
<tr>
<td></td>
<td>%COSS</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COSS%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tangent</td>
<td>TAN</td>
<td>SA</td>
<td>tan(x)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>TAN%</td>
<td>SVa</td>
<td></td>
<td>[x</td>
<td>&lt; 2^24</td>
</tr>
<tr>
<td></td>
<td>%TAN</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%TAN%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. tangent</td>
<td>DTAN</td>
<td>SA</td>
<td>tan(x)</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DTAN%</td>
<td>SVb</td>
<td></td>
<td>[x</td>
<td>&lt; 2^46</td>
</tr>
<tr>
<td></td>
<td>%DTAN</td>
<td>VA</td>
<td></td>
<td>[x-n</td>
<td>&gt; 0,</td>
</tr>
<tr>
<td></td>
<td>%DTAN%</td>
<td>VVb</td>
<td></td>
<td>[n</td>
<td>= 1,3,5,...</td>
</tr>
<tr>
<td>Cotangent</td>
<td>COT</td>
<td>SA</td>
<td>cot(x)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>COT%</td>
<td>SVa</td>
<td></td>
<td>[x</td>
<td>&lt; 2^24</td>
</tr>
<tr>
<td></td>
<td>%COT</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COT%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. cotangent</td>
<td>DCOT</td>
<td>SA</td>
<td>cot(x)</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DCOT%</td>
<td>SVb</td>
<td></td>
<td>[x</td>
<td>&lt; 2^46</td>
</tr>
<tr>
<td></td>
<td>%DCOT</td>
<td>VA</td>
<td></td>
<td>[x-n</td>
<td>&gt; 0,</td>
</tr>
<tr>
<td></td>
<td>%DCOT%</td>
<td>VVb</td>
<td></td>
<td>[n</td>
<td>= 1,3,5,...</td>
</tr>
</tbody>
</table>

† Not FORTRAN callable
Table 3-5. Hyperbolic routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperbolic cosine</td>
<td>COSH</td>
<td>SA</td>
<td>((e^x + e^{-x})/2)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%COSH</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COSH%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SINH</td>
<td>SA</td>
<td>((e^x - e^{-x})/2)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%SINH</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%SINH%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperbolic cosine and sine†</td>
<td>COSSH</td>
<td>SA</td>
<td>(y_1 = (e^x + e^{-x})/2)</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%COSSH%</td>
<td>SVb</td>
<td>(y_2 = (e^x - e^{-x})/2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COSSH%</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%COSSH%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperbolic tangent</td>
<td>TANH</td>
<td>SA</td>
<td>((e^x - e^{-x})/(e^x + e^{-x}))</td>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>%TANH%</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%TANH%</td>
<td>VVa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. hyperbolic cosine</td>
<td>DCOSH</td>
<td>SA</td>
<td>(\cosh(x))</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>%DCOSH%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DCOSH%</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DCOSH%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-prec. hyperbolic sine</td>
<td>DSINH</td>
<td>SA</td>
<td>(\sinh(x))</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>%DSINH%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DSINH%</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DSINH%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Not FORTRAN callable
<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Names</th>
<th>Call Seq. Code</th>
<th>Definition</th>
<th>Arguments</th>
<th>Func. Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-prec. hyperbolic tangent</td>
<td>DTANH</td>
<td>SA</td>
<td>tanh(x)</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>DTANH</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DTANH</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DTANH%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Arguments</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Computes logical product 0011</td>
<td>2</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>0010</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>COMPL</td>
<td>Computes logical complement 01</td>
<td>1</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>EQV</td>
<td>Computes logical equivalence</td>
<td>2</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td></td>
<td>0011</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>0110</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>LEADZ</td>
<td>Counts the number of leading zero bits</td>
<td>1</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td>MASK</td>
<td>Returns a bit mask of ones. If 0&lt;arg&lt;63, the mask is left-justified. If 64&lt;arg&lt;128, a right-justified mask of (128-argument) bits is returned.</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>NEQV</td>
<td>Computes logical difference (same as XOR)</td>
<td>2</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td>OR</td>
<td>Computes logical sum 0011</td>
<td>2</td>
<td>I,R,L,B</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>0110</td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>
Table 3-6. Boolean arithmetic routines (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Arguments</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPCNT</td>
<td>Counts the number of bits set to 1</td>
<td>1 I,R,L,B</td>
<td>I</td>
</tr>
<tr>
<td>POPPAR</td>
<td>Returns 0 if even number of bits set; returns 1 if odd number of bits set</td>
<td>1 I,R,L,B</td>
<td>I</td>
</tr>
<tr>
<td>SHIFT</td>
<td>Performs circular shift of (arg&lt;sub&gt;1&lt;/sub&gt;) to the left by (arg&lt;sub&gt;2&lt;/sub&gt;) bits</td>
<td>2 I,R,L,B</td>
<td>B</td>
</tr>
<tr>
<td>SHIFTL</td>
<td>Performs left shift of (arg&lt;sub&gt;1&lt;/sub&gt;) by (arg&lt;sub&gt;2&lt;/sub&gt;) bits with zero fill</td>
<td>2 I,R,L,B</td>
<td>B</td>
</tr>
<tr>
<td>SHIFTR</td>
<td>Performs right shift of (arg&lt;sub&gt;1&lt;/sub&gt;) by (arg&lt;sub&gt;2&lt;/sub&gt;) bits with zero fill</td>
<td>2 I,R,L,B</td>
<td>B</td>
</tr>
<tr>
<td>XOR</td>
<td>Computes logical difference</td>
<td>0011 1010 1001</td>
<td>2 I,R,L,B</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex base raised to a complex power</td>
<td>CTOC%</td>
<td>C S1,S2 C S3,S4 C S1,S2</td>
<td></td>
</tr>
<tr>
<td>(C**C)</td>
<td>CTO%C%</td>
<td>C S1,S2 C V3,V4 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTOC%</td>
<td>C V1,V2 C S3,S4 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTO%C%</td>
<td>C V1,V2 C V3,V4 C V1,V2</td>
<td></td>
</tr>
<tr>
<td>Complex base raised to an integer power</td>
<td>CTOI%</td>
<td>C S1,S2 I S3 C S1,S2</td>
<td></td>
</tr>
<tr>
<td>(C**I)</td>
<td>CTOI%</td>
<td>C S1,S2 I V3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTOI%</td>
<td>C V1,V2 I S3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTO%I%</td>
<td>C V1,V2 I V3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td>Complex base raised to a real power</td>
<td>CTOR%</td>
<td>C S1,S2 R S3 C S1,S2</td>
<td></td>
</tr>
<tr>
<td>(C**R)</td>
<td>CTO%R%</td>
<td>C S1,S2 R V3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTOR%</td>
<td>C V1,V2 R S3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CTO%R%</td>
<td>C V1,V2 R V3 C V1,V2</td>
<td></td>
</tr>
<tr>
<td>Double-precision base raised to a</td>
<td>DTOD%</td>
<td>+D S1,S2 D S3,S4 D S1,S2</td>
<td></td>
</tr>
<tr>
<td>double-precision power (D**D)</td>
<td>DTOD%</td>
<td>+D S1,S2 D V3,V4 D V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DTOD%</td>
<td>+D V1,V2 D S3,S4 D V1,V2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DTOD%</td>
<td>+D V1,V2 D V3,V4 D V1,V2</td>
<td></td>
</tr>
</tbody>
</table>
Table 3-7. Values raised to a power (continued)

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-precision base raised to an integer power</td>
<td>DITO%</td>
<td>D S1,S2 I S3</td>
<td>D S1,S2</td>
</tr>
<tr>
<td>(D**I)</td>
<td>DITO%I%</td>
<td>D S1,S2 I V3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td></td>
<td>%DITO%</td>
<td>D V1,V2 I S3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td></td>
<td>%DITO%I%</td>
<td>D V1,V2 I V3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td>Double-precision base raised to a real power (D**R)</td>
<td>DTO%R%</td>
<td>+D S1,S2 R S3</td>
<td>D S1,S2</td>
</tr>
<tr>
<td></td>
<td>DTO%R%%</td>
<td>+D S1,S2 R V3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td></td>
<td>%DTO%R%</td>
<td>+D V1,V2 R S3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td></td>
<td>%DTO%R%%</td>
<td>+D V1,V2 R V3</td>
<td>D V1,V2</td>
</tr>
<tr>
<td>Integer base raised to an integer power (I**I)</td>
<td>ITOI%</td>
<td>I S1 I S2</td>
<td>I S1</td>
</tr>
<tr>
<td></td>
<td>ITOI%I%</td>
<td>I S1 I V2</td>
<td>I V1</td>
</tr>
<tr>
<td></td>
<td>%ITOI%</td>
<td>I V1 I S2</td>
<td>I V1</td>
</tr>
<tr>
<td></td>
<td>%ITOI%I%</td>
<td>I V1 I V2</td>
<td>I V1</td>
</tr>
<tr>
<td>Real base raised to an integer power (R**I)</td>
<td>RTOI%</td>
<td>R S1 I S2</td>
<td>R S1</td>
</tr>
<tr>
<td></td>
<td>RTOI%I%</td>
<td>R S1 I V2</td>
<td>R V1</td>
</tr>
<tr>
<td></td>
<td>%RTOI%</td>
<td>R V1 I S2</td>
<td>R V1</td>
</tr>
<tr>
<td></td>
<td>%RTOI%I%</td>
<td>R V1 I V2</td>
<td>R V1</td>
</tr>
<tr>
<td>Real base raised to a real power (R**R)</td>
<td>RTOR%</td>
<td>+R S1 R S2</td>
<td>R S1</td>
</tr>
<tr>
<td></td>
<td>RTOR%R%</td>
<td>+R S1 R V2</td>
<td>R V1</td>
</tr>
<tr>
<td></td>
<td>%RTOR%</td>
<td>+R V2 R S2</td>
<td>R V1</td>
</tr>
<tr>
<td></td>
<td>%RTOR%R%</td>
<td>+R V2 R V2</td>
<td>R V1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-precision addition (D+D)</td>
<td>DASS%</td>
<td>D S1,S2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DASV%</td>
<td>D S1,S2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td></td>
<td>DAVS%</td>
<td>D V1,V2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DAVV%</td>
<td>D V1,V2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td>Double-precision division (D/D)</td>
<td>DDSS%</td>
<td>D S1,S2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DDSV%</td>
<td>D S1,S2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td></td>
<td>DDVS%</td>
<td>D V1,V2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DDVV%</td>
<td>D V1,V2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td>Double-precision multiplication (D*D)</td>
<td>DMSS%</td>
<td>D S1,S2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DMSV%</td>
<td>D S1,S2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td></td>
<td>DMVS%</td>
<td>D V1,V2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DMVV%</td>
<td>D V1,V2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td>Double-precision subtraction (D-D)</td>
<td>DSSS%</td>
<td>D S1,S2</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DSSV%</td>
<td>D S1,S2</td>
<td>D V3,V4</td>
</tr>
<tr>
<td></td>
<td>DSVS%</td>
<td>D V3,V4</td>
<td>D S3,S4</td>
</tr>
<tr>
<td></td>
<td>DSVV%</td>
<td>D V3,V4</td>
<td>D V3,V4</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Name</th>
<th>Call Type</th>
<th>Entry Conditions</th>
<th>Exit Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple-precision addition</td>
<td>$TADD$</td>
<td>Value</td>
<td>(S1)=address of addend (S2)=address of augend (S3)=address of result</td>
<td>Result to address in S3</td>
</tr>
<tr>
<td></td>
<td>TADD$^+$</td>
<td>Address</td>
<td>$arg_1$=address of addend $arg_2$=address of augend $arg_3$=address of result</td>
<td>Result to address in $arg_3$</td>
</tr>
<tr>
<td></td>
<td>TASS</td>
<td>Address</td>
<td>$arg_1$=address of addend $arg_2$=address of augend $arg_3$=address of result</td>
<td>$(S1),(S2),(S3)=result$</td>
</tr>
<tr>
<td></td>
<td>TASS$^&amp;$</td>
<td>Value</td>
<td>$(S1),(S2),(S3)=addend (S4),(S5),(S6)=augend</td>
<td>$(S1),(S2),(S3)=result$</td>
</tr>
</tbody>
</table>

$^+$ FORTRAN entry point
<table>
<thead>
<tr>
<th>Definition</th>
<th>Name</th>
<th>Call Type</th>
<th>Entry Conditions</th>
<th>Exit Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple-precision</td>
<td>$TDIV</td>
<td>Value</td>
<td>$(S1) =$address of dividend $(S2) =$address of divisor $(S3) =$address of result</td>
<td>Result to address in S3</td>
</tr>
<tr>
<td>division</td>
<td>TDIV</td>
<td>Address</td>
<td>$arg_1 =$address of dividend $arg_2 =$address of divisor $arg_3 =$address of result</td>
<td>Result to address in $arg_3$</td>
</tr>
<tr>
<td></td>
<td>TDSS</td>
<td>Address</td>
<td>$arg_1 =$address of dividend $arg_2 =$address of divisor $arg_3 =$address of result</td>
<td>$(S1),(S2),(S3)=result</td>
</tr>
<tr>
<td></td>
<td>TDSS%</td>
<td>Value</td>
<td>$(S1),(S2),(S3)=result $(S4),(S5),(S6)=divisor</td>
<td>$(S1),(S2),(S3)=result</td>
</tr>
<tr>
<td>Triple-precision</td>
<td>$TMLT</td>
<td>Value</td>
<td>$(S1) =$address of multiplier $(S2) =$address of multiplicand $(S3) =$address of result</td>
<td>Result to address in S3</td>
</tr>
<tr>
<td>multiplication</td>
<td>TMLT</td>
<td>Address</td>
<td>$arg_1 =$address of multiplier $arg_2 =$address of multiplicand $arg_3 =$address of result</td>
<td>Result to address in $arg_3$</td>
</tr>
<tr>
<td></td>
<td>TMSS</td>
<td>Address</td>
<td>$arg_1 =$address of multiplier $arg_2 =$address of multiplicand $arg_3 =$address of result</td>
<td>$(S1),(S2),(S3)=result</td>
</tr>
<tr>
<td></td>
<td>TMSS%</td>
<td>Value</td>
<td>$(S1),(S2),(S3)=result $(S4),(S5),(S6)=multiplicand</td>
<td>$(S1),(S2),(S3)=result</td>
</tr>
</tbody>
</table>

† FORTRAN entry point
Table 3-9. Triple-precision arithmetic routines (continued)

<table>
<thead>
<tr>
<th>Definition</th>
<th>Name</th>
<th>Call Type</th>
<th>Entry Conditions</th>
<th>Exit Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple-precision subtraction</td>
<td>$\text{TSUB}$</td>
<td>Value</td>
<td>$(S1)=$address of minuend $(S2)=$address of subtrahend $(S3)=$address of result</td>
<td>Result to address in $S3$</td>
</tr>
<tr>
<td></td>
<td>$\text{TSUB}^+$</td>
<td>Address</td>
<td>$\text{arg}_1=$address of minuend $\text{arg}_2=$address of subtrahend $\text{arg}_3=$address of result</td>
<td>Result to address in $\text{arg}_3$</td>
</tr>
<tr>
<td></td>
<td>$\text{TSSS}$</td>
<td>Address</td>
<td>$\text{arg}_1=$address of minuend $\text{arg}_2=$address of subtrahend</td>
<td>$(S1),(S2),(S3)=$result</td>
</tr>
<tr>
<td></td>
<td>$\text{TSSS}%$</td>
<td>Value</td>
<td>$(S1),(S2),(S3)=$minuend $(S4),(S5),(S6)=$subtrahend</td>
<td>$(S1),(S2),(S3)=$result</td>
</tr>
</tbody>
</table>

$^+$ FORTRAN entry point

Example of FORTRAN use:

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

Example 1 of CAL use:

<table>
<thead>
<tr>
<th>Location</th>
<th>Result</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>TASS, (ARG1, ARG2)</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Definition</th>
<th>Name</th>
<th>Registers</th>
<th>Exit</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Entry</td>
<td>Exit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quo.</td>
<td>Rem.</td>
<td></td>
</tr>
<tr>
<td>Scalar/scalar</td>
<td>LDSS%</td>
<td>S1,S2</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>Scalar/vector</td>
<td>LDSV%</td>
<td>S1,V2,VL</td>
<td>V1</td>
<td>V2</td>
</tr>
<tr>
<td>Vector/scalar</td>
<td>LDVS%</td>
<td>V1,S2,VL</td>
<td>V1</td>
<td>V2</td>
</tr>
<tr>
<td>Vector/vector</td>
<td>LDVV%</td>
<td>V1,V2,VL</td>
<td>V1</td>
<td>V2</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII compare for GE</td>
<td>LGE</td>
<td>Character, Character</td>
<td>L S1</td>
</tr>
<tr>
<td>ASCII compare for GT</td>
<td>LGT</td>
<td>Character, Character</td>
<td>L S1</td>
</tr>
<tr>
<td>ASCII compare for LE</td>
<td>LLE</td>
<td>Character, Character</td>
<td>L S1</td>
</tr>
<tr>
<td>ASCII compare for LT</td>
<td>LLT</td>
<td>Character, Character</td>
<td>L S1</td>
</tr>
<tr>
<td>Find position of second argument as substring of first argument</td>
<td>INDEX</td>
<td>Character, Character</td>
<td>I S1</td>
</tr>
<tr>
<td>Find length of argument</td>
<td>LEN</td>
<td>Character</td>
<td>I S1</td>
</tr>
</tbody>
</table>

Example:

Call from CAL:

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

\[ \text{arg}_1 \quad \text{Address of character operand 1} \]

\[ \text{arg}_2 \quad \text{Address of character operand 2} \]

Exit:

\( \text{(Sl)} \quad \text{Logical result of comparison} \)

Call from FORTRAN:

\[
\text{logical} = \text{LGE(} \text{chr}_1, \text{chr}_2) \]

\( \text{logical} \quad \text{Logical result of comparison} \)

\( \text{chr}_1 \quad \text{Character operand 1} \)

\( \text{chr}_2 \quad \text{Character operand 2} \)

The character functions in table 3-12 are called with the character address of the first operand in register \( \text{Sl} \) and the address of the second operand in register \( \text{S2} \). These routines are called only from CAL.

Table 3-12. Character comparison functions called from CAL

<table>
<thead>
<tr>
<th>Definition</th>
<th>Function Name</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \text{Operand 1} )</td>
<td>( \text{Operand 2} )</td>
</tr>
<tr>
<td>ASCII compare for GE</td>
<td>$GE$</td>
<td>Character</td>
<td>Character</td>
</tr>
<tr>
<td>ASCII compare for GT</td>
<td>$GT$</td>
<td>Character</td>
<td>Character</td>
</tr>
<tr>
<td>ASCII compare for LE</td>
<td>$LE$</td>
<td>Character</td>
<td>Character</td>
</tr>
<tr>
<td>ASCII compare for LT</td>
<td>$LT$</td>
<td>Character</td>
<td>Character</td>
</tr>
<tr>
<td>ASCII compare for EQ</td>
<td>$EQ$</td>
<td>Character</td>
<td>Character</td>
</tr>
<tr>
<td>ASCII compare for NE</td>
<td>$NE$</td>
<td>Character</td>
<td>Character</td>
</tr>
</tbody>
</table>
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 CPT 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

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Ent. Name</th>
<th>Call Seq. Code</th>
<th>Argument 1</th>
<th>Argument 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converts binary to decimal ASCII (right-justified, blank-filled)</td>
<td>BTD</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'D'99999999$</td>
<td>One word ASCII string (right-justified, blank-filled, decimal conversion)</td>
</tr>
<tr>
<td>Converts binary to decimal ASCII (left-justified, zero-filled)</td>
<td>BTDL</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'D'99999999$</td>
<td>One word ASCII string (left-justified, zero-filled, decimal conversion)</td>
</tr>
<tr>
<td>Converts binary to decimal ASCII (right-justified, zero-filled)</td>
<td>BTDR</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'D'99999999$</td>
<td>One word ASCII string (right-justified, zero-filled, decimal conversion)</td>
</tr>
<tr>
<td>Converts binary to octal ASCII (right-justified, blank-filled)</td>
<td>BTOL</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'O'77777777$</td>
<td>One word ASCII string (right-justified, blank-filled, octal conversion)</td>
</tr>
<tr>
<td>Converts binary to octal ASCII (left-justified, zero-filled)</td>
<td>BTOLN</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'O'77777777$</td>
<td>One word ASCII string (left-justified, zero-filled, octal conversion)</td>
</tr>
<tr>
<td>Converts binary to octal ASCII (right-justified, zero-filled)</td>
<td>BTOR</td>
<td>SA</td>
<td>I</td>
<td>$0^&lt;X'O'77777777$</td>
<td>One word ASCII string (right-justified, zero-filled, octal conversion)</td>
</tr>
<tr>
<td>Converts decimal ASCII to binary</td>
<td>DTF</td>
<td>SA</td>
<td>I</td>
<td>Decimal ASCII (left-justified, zero-filled)</td>
<td>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.</td>
</tr>
<tr>
<td>Converts octal ASCII to binary</td>
<td>OTB</td>
<td>SA</td>
<td>I</td>
<td>Octal ASCII (left-justified, zero-filled)</td>
<td>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.</td>
</tr>
</tbody>
</table>
Example:

Call from FORTRAN:

\[
\text{result}=\text{BTD}(\text{arg})
\]

\text{result} \quad \text{Decimal ASCII result (right-justified, blank-filled)}
\text{arg} \quad \text{Integer argument}

Call from CAL:

\[
\text{CALLV BTD\#,S1}
\]

Entry:
(S1) \quad \text{Integer value}

Exit:
(S1) \quad \text{Decimal ASCII result (right-justified, blank-filled)}

Example:

Call from FORTRAN:

\[
\text{result}=\text{DTB}(\text{arg,errcode})
\]

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

Call from CAL:

\[
\text{CALLV DTB\#,S1}
\]

Entry:
(S1) \quad \text{Decimal ASCII (left-justified, zero-filled)}

Exit:
(S1) \quad \text{Integer value}
(S0) \quad \text{Error code} \quad 0 \text{ if conversion successful}
\quad \quad \quad -1 \text{ if error}
PSEUDO VECTORIZATION ROUTINES


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
Table 3-14 contains the miscellaneous math routines.

### Table 3-14. Miscellaneous math routines

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Name</th>
<th>Call Seq. Code</th>
<th>Argument Type</th>
<th>Result Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real absolute value</td>
<td>ABS</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Find the imaginary portion of a complex number</td>
<td>AIMAG</td>
<td>SA</td>
<td>C</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Truncate to integral value</td>
<td>AINT</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Real modulo arithmetic (y=x_1-x_2[x_1/x_2])</td>
<td>AMOD</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Calculates nearest whole number (y=[x+.5]) if (x&gt;0) (y=[x-.5]) if (x&lt;0)</td>
<td>ANINT</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
Table 3-14. Miscellaneous math routines (continued)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Name</th>
<th>Call Seq. Code</th>
<th>Argument Type</th>
<th>Result Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex absolute value</td>
<td>CABS</td>
<td>SA</td>
<td>C</td>
<td>R</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>CABS%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CABS</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%CABS%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer to character conversion</td>
<td>CHAR</td>
<td>I</td>
<td></td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>Convert two reals to a complex</td>
<td>CMPLX</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td>C</td>
</tr>
<tr>
<td>Complex conjugate</td>
<td>CONJG</td>
<td>SA</td>
<td>C</td>
<td>C</td>
<td>(</td>
</tr>
<tr>
<td>Determine double-precision absolute value</td>
<td>DABS</td>
<td>SA</td>
<td>D</td>
<td>D</td>
<td>(</td>
</tr>
<tr>
<td>Convert real to double-precision</td>
<td>DBLE</td>
<td>SA</td>
<td>R</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Double-precision positive real difference MAX(0,x-y)</td>
<td>DDIM</td>
<td>SA</td>
<td>D</td>
<td>D</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DDIM%</td>
<td>SVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DDIM</td>
<td>VA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>%DDIM%</td>
<td>VVb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Purpose</td>
<td>Entry Name</td>
<td>Call Seq. Code</td>
<td>Argument Type</td>
<td>Result Type</td>
<td>Restrictions</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>----------------</td>
<td>---------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Positive real difference MAX(0, x-y)</td>
<td>DIM</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Truncate double-precision numbers, $y=[x]$ fraction last no rounding</td>
<td>DINT</td>
<td>SA</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DINT%</td>
<td>SVb</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>%DINT</td>
<td>VA</td>
<td>D</td>
<td></td>
<td>( 0&lt;</td>
</tr>
<tr>
<td></td>
<td>%DINT%</td>
<td>VVb</td>
<td>D</td>
<td></td>
<td>( 0&lt;</td>
</tr>
<tr>
<td>Double-precision modulo arithmetic $y=x_1-x_2[x_1/x_2]$</td>
<td>DMOD</td>
<td>SA</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DMOD%</td>
<td>SVb</td>
<td>D</td>
<td></td>
<td>( 0&lt;</td>
</tr>
<tr>
<td></td>
<td>%DMOD</td>
<td>VA</td>
<td>D</td>
<td></td>
<td>( 0&lt;</td>
</tr>
<tr>
<td></td>
<td>%DMOD%</td>
<td>VVb</td>
<td>D</td>
<td></td>
<td>( 0&lt;</td>
</tr>
<tr>
<td>Calculates nearest integer; defined as $\lfloor x+0.5 \rfloor$ if $x \geq 0$</td>
<td>DNINT</td>
<td>SA</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>DNINT%</td>
<td>SVb</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>%DNINT</td>
<td>VA</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>%DNINT%</td>
<td>VVb</td>
<td>D</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td>Double-precision product of two real arguments</td>
<td>DPROD</td>
<td>SA</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-14. Miscellaneous math routines (continued)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Name</th>
<th>Call Seq. Code</th>
<th>Argument Type 1</th>
<th>Argument Type 2</th>
<th>Result Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfers sign from one double-precision number to another defined as y=</td>
<td>DSIGN</td>
<td>SA</td>
<td>D (to which sign is transferred)</td>
<td>D (from which sign is transferred)</td>
<td>D</td>
<td>$</td>
</tr>
<tr>
<td>number to another defined as</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y=</td>
<td>x_1</td>
<td>$ if $x_2&gt;0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y=</td>
<td>x_1</td>
<td>$ if $x_2&lt;0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert integer to real</td>
<td>FLOAT</td>
<td>SA</td>
<td>I</td>
<td></td>
<td>R</td>
<td>$</td>
</tr>
<tr>
<td>Integer absolute value</td>
<td>IABS</td>
<td>SA</td>
<td>I</td>
<td></td>
<td>I</td>
<td>$</td>
</tr>
<tr>
<td>Character to integer conversion</td>
<td>ICHAR</td>
<td>CH</td>
<td></td>
<td>I</td>
<td>I</td>
<td>$</td>
</tr>
<tr>
<td>Positive integer difference MAX(0,x-y)</td>
<td>IDIM</td>
<td>SA</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>$</td>
</tr>
</tbody>
</table>
Table 3-14. Miscellaneous math routines (continued)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Name</th>
<th>Call Seq. Code</th>
<th>Argument Type</th>
<th>Result Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest integer to a double-precision number; defined as (</td>
<td>x+0.5</td>
<td>) if (x\geq 0) (</td>
<td>x-0.5</td>
<td>) if (x&lt;0)</td>
<td>IDNINT IDNINT% %IDNINT %IDNINT%</td>
</tr>
<tr>
<td>Truncate to integral value</td>
<td>INT</td>
<td>SA</td>
<td>R</td>
<td>I</td>
<td>(</td>
</tr>
<tr>
<td>Transfer sign from one integer to another defined as (y=</td>
<td>x_1</td>
<td>) if (x_2\geq 0) (y=-</td>
<td>x_1</td>
<td>) if (x_2&lt;0)</td>
<td>ISIGN</td>
</tr>
<tr>
<td>Perform 64-bit modulo arithmetic on two integer scalars (y=x_1-x_2\lfloor x_1/x_2 \rfloor)</td>
<td>MOD MOD% MODSS%</td>
<td>SA SVb SVb</td>
<td>I (x_1) I (x_2)</td>
<td>I</td>
<td>(</td>
</tr>
<tr>
<td>General Purpose</td>
<td>Entry Name</td>
<td>Call Seq. Code</td>
<td>Argument Type 1</td>
<td>Argument Type 2</td>
<td>Result Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Perform 64-bit modulo arithmetic on two integer vectors ( y =</td>
<td>x_1 - x_2</td>
<td>) ( (x_1/x_2) )</td>
<td>%MOD% MODVV%</td>
<td>VVb</td>
<td>I ( x_1 )</td>
</tr>
<tr>
<td>Perform 64-bit modulo arithmetic on integer scalar and integer vector</td>
<td>MODSV%</td>
<td>I ( x_1 ) (dividend) ( (S1) )</td>
<td>I ( x_2i- ) (divisor) ( (V2) )</td>
<td>I ( y ) (V1-remainder V2-quotient)</td>
<td></td>
</tr>
<tr>
<td>Perform 64-bit modulo arithmetic on integer vector and integer scalar</td>
<td>MODVS%</td>
<td>I (dividend) ( (V1) )</td>
<td>I (divisor) ( (S2) )</td>
<td>I (V1-remainder S2-quotient)</td>
<td></td>
</tr>
<tr>
<td>Calculate nearest integer ( y = \lceil x + 0.5 \rceil ) if ( x &gt; 0 ) ( y = \lfloor x - 0.5 \rfloor ) if ( x &lt; 0 )</td>
<td>NINT</td>
<td>SA</td>
<td>I</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td>Return real portion of a complex number</td>
<td>REAL</td>
<td>SA</td>
<td>C</td>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>
### Table 3-14. Miscellaneous math routines (continued)

<table>
<thead>
<tr>
<th>General Purpose</th>
<th>Entry Name</th>
<th>Call Seq. Code</th>
<th>Argument Type 1</th>
<th>Argument Type 2</th>
<th>Result Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer sign from one real number to another; defined as $y =</td>
<td>x_1</td>
<td>$ if $x_2 &gt; 0$; $y =</td>
<td>x_1</td>
<td>$ if $x_2 &lt; 0$</td>
<td>SIGN</td>
<td>SA</td>
</tr>
</tbody>
</table>

Examples:

**Call from CAL:**

```plaintext
CALL F eager(y)
```

**Entry:**

`arg1` Address of integer argument

**Exit:**

(S1) Real result

**Call from FORTRAN:**

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

**real** Real result

**integer** Integer result
### Table 3-15. Random number routines

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

**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.

$\text{lIN}$ Contains integer powers of 10 in the range of $10^0$ to $10^{18}$

$\text{LMX}$ Contains double-precision floating-point representation of negative powers of 10 in the range of $10^0$ to $10^{-4096}$

$\text{LPX}$ 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

RANFSP Contains 128 random number seeds in a buffer
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 $inox$ and $inoy$ 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 $x_1, x_2, \ldots, x_n$, the contents of memory spaces $SX(1), SX(2), \ldots, SX(n)$ are

$x_1, x_2, \ldots, x_n$ if $inox = 1$, or $x_n, x_{n-1}, \ldots, x_1$ if $inox = -1$. 

SR-0014 4-2 I
That is, element $x_i$ is in location $SX(l + (i-l) \times incx)$ if $incx > 0$ or in location $SX(l + (n-i) \times incx)$ if $incx < 0$. Location $SX(l)$ 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.

$$SASUM(5, X, 2) = (((1.0 + 3.0) + 5.0) + 7.0) + 9.0), \text{ and}$$
$$SASUM(5, X, -2) = (((9.0 + 7.0) + 5.0) + 3.0) + 1.0)$$

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

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

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Name (parameter list)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of element with maximum absolute value</td>
<td>ISAMAX($n$, $sx$, $incx$)</td>
<td>Integer function</td>
</tr>
<tr>
<td></td>
<td>ICAMAX($n$, $cx$, $incx$)</td>
<td></td>
</tr>
<tr>
<td>Sum of the absolute values</td>
<td>SASUM($n$, $sx$, $incx$)</td>
<td>Real function</td>
</tr>
<tr>
<td></td>
<td>SCASUM($n$, $cx$, $incx$)</td>
<td></td>
</tr>
<tr>
<td>Constant times a vector plus another vector</td>
<td>SAXPY($n$, $sa$, $sx$, $incx$, $sy$, $incy$)</td>
<td>Routine</td>
</tr>
<tr>
<td></td>
<td>CAXPY($n$, $sa$, $cx$, $incx$, $cy$, $incy$)</td>
<td></td>
</tr>
<tr>
<td>Copy one array into another</td>
<td>SCOPY($n$, $sx$, $incx$, $sy$, $incy$)</td>
<td>Routine</td>
</tr>
<tr>
<td></td>
<td>CCOPY($n$, $cx$, $incx$, $cy$, $incy$)</td>
<td></td>
</tr>
<tr>
<td>Euclidean norm of array</td>
<td>SNRM2($n$, $sx$, $incx$)</td>
<td>Real function</td>
</tr>
<tr>
<td></td>
<td>SCNRM2($n$, $cx$, $incx$)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Name (parameter list)</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Dot product</td>
<td>SDOT(n,sx,\text{incx},sy,\text{incy}))</td>
<td>Real function</td>
</tr>
<tr>
<td></td>
<td>CDOTC(n,ax,\text{incx},cy,\text{incy}))</td>
<td>Complex function</td>
</tr>
<tr>
<td></td>
<td>CDOTU(n,ax,\text{incx},cy,\text{incy}))</td>
<td></td>
</tr>
<tr>
<td>Construct Givens plane rotation</td>
<td>SROTG(a,b,c,s))</td>
<td>Routine</td>
</tr>
<tr>
<td>Apply Givens plane rotation</td>
<td>SROT(n,sx,\text{incx},sy,\text{incy},c,s))</td>
<td>Routine</td>
</tr>
<tr>
<td>Construct Givens modified plane rotation</td>
<td>SROTMG (dl,d2,bl,b2,param) )</td>
<td>Routine</td>
</tr>
<tr>
<td>Apply Givens modified plane rotation</td>
<td>SROTM(n,sx,\text{incx},sy,\text{incy},param) )</td>
<td>Routine</td>
</tr>
<tr>
<td>Scale array</td>
<td>SSCAL(n,sa,sx,\text{incx}) )</td>
<td>Routine</td>
</tr>
<tr>
<td></td>
<td>CSSCAL(n,sa,ax,\text{incx}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSCAL(n,ca,ax,\text{incx}) )</td>
<td></td>
</tr>
<tr>
<td>Swap two arrays</td>
<td>SSWAP(n,sx,\text{incx},sy,\text{incy}) )</td>
<td>Routine</td>
</tr>
<tr>
<td></td>
<td>CSWAP(n,ax,\text{incx},cy,\text{incy}) )</td>
<td></td>
</tr>
</tbody>
</table>

**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

\[
|x_i| = \max |x_j|: j=1,...,n.
\]

where \(x_j\) is an element of a real vector.
Call from FORTRAN:

\[
\text{imax} = \text{ISAMAX}(n, sx, inax)
\]

\( n \)  \hspace{1cm} \text{Number of elements to process in the vector to be searched (} n=\text{vector length if } inax=1; \ n=\text{vector length}/2 \text{ if } inax=2; \text{ etc.)}

\( sx \)  \hspace{1cm} \text{Real vector to be searched}

\( inax \)  \hspace{1cm} \text{Skip distance between elements of } sx. \text{ For contiguous elements, } inax=1.

ICAMAX determines the first index \( i \) such that

\[ |\text{Real}(x_i)| + |\text{Imag}(x_i)| = \max \{|\text{Real}(x_j)| + |\text{Imag}(x_j)| : j=1,\ldots,n \}. \]

where \( x_j \) is an element of a complex vector.

Call from FORTRAN:

\[
\text{imax} = \text{ICAMAX}(n, cx, inax)
\]

\( n \)  \hspace{1cm} \text{Number of elements to process in the vector to be searched (} n=\text{vector length if } inax=1; \ n=\text{vector length}/2 \text{ if } inax=2; \text{ etc.)}

\( cx \)  \hspace{1cm} \text{Complex vector to be searched}

\( inax \)  \hspace{1cm} \text{Skip distance between elements of } cx. \text{ For contiguous elements, } inax=1.

SUM OF THE ABSOLUTE VALUES

These real functions sum the absolute values of a vector.

SASUM computes

\[ \sum_{i=1}^{n} |x_i| \]

where \( x_i \) is an element of a real vector.
Call from FORTRAN:

\[
\text{sum=SASUM}(n, sx, inax)
\]

- \( n \) Number of elements in the vector to be summed
- \( sx \) Real vector to be summed
- \( inax \) Skip distance between elements of \( sx \). For contiguous elements, \( inax=1 \).

SCASUM computes

\[
\text{sum}= \frac{1}{n} \sum_{i=1}^{n} \left( |\text{Real}(x_i)| + |\text{Imag}(x_i)| \right)
\]

where \( x_i \) is an element of a complex vector.

Call from FORTRAN:

\[
\text{sum=SCASUM}(n, cx, inax)
\]

- \( n \) Number of elements in the vector to be summed
- \( cx \) Complex vector to be summed
- \( inax \) Skip distance between elements of \( cx \). For contiguous elements, \( inax=1 \).

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:

\begin{verbatim}
CALL SAXPY(n,sa,sx,incx,sy,incy)
\end{verbatim}

- \( n \): 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:

\begin{verbatim}
CALL CAXPY(n,ca,cx,incx,cy,incy)
\end{verbatim}

- \( n \): 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

\[ y_i = x_i : i = 1, \ldots, n \]

where \( x_i \) and \( y_i \) are elements of real vectors.

Call from FORTRAN:

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

- \( n \): 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
- \( iny \): Skip distance between elements of \( sy \). For contiguous elements, \( iny = 1 \).

CCOPY copies a complex vector

\[ y_i = x_i : i = 1, \ldots, n \]

where \( x_i \) and \( y_i \) are elements of complex vectors.

Call from FORTRAN:

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

- \( n \): 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
- \( iny \): Skip distance between elements of \( cy \). For contiguous elements, \( iny = 1 \).
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:

\[
dot = \text{SDOT}(n, sx, inax, sy, inay)
\]

- \( n \): Number of elements in the vectors
- \( sx \): Real vector operand
- \( inax \): Skip distance between elements of \( sx \). For contiguous elements, \( inax = 1 \).
- \( sy \): Real vector operand
- \( inay \): Skip distance between elements of \( sy \). For contiguous elements, \( inay = 1 \).

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:

\[
cdot = \text{CDOTC}(n, ox, inax, oy, inay)
\]
n  Number of elements in vector

cx  Complex vector operand

ixon  Skip distance between elements of cx. For contiguous elements, icx=1.

cy  Complex vector operand

incy  Skip distance between elements of cy. For contiguous elements,incy=1.

CDOTU computes

\[ odot = \sum_{i=1}^{n} x_i y_i \]

where \( x_i \) and \( y_i \) are elements of complex vectors.

Call from FORTRAN:

\[ odot = \text{CDOTU}(n, cx, icx, cy, icy) \]

n  Number of elements in vector

cx  Complex vector operand

ixon  Skip distance between elements of cx. For contiguous elements, icx=1.

cy  Complex vector operand

incy  Skip distance between elements of cy. For contiguous elements, icy=1.

EUCLIDEAN NORM OF AN ARRAY (\( L_2 \) NORM)

These real functions compute the Euclidean or \( L_2 \) norm of a vector.

SNRM2 computes

\[ \text{euonorm} = \left( \sum_{i=1}^{n} |x_i|^2 \right)^{1/2} \]

where \( x_i \) is an element of a real vector.
Call from FORTRAN:

\[ \text{eucnorm} = \text{SNRM2}(n, sx, inx) \]

\( n \quad \text{Number of elements in vector} \)

\( sx \quad \text{Real vector operand} \)

\( inx \quad \text{Skip distance between elements of } sx. \text{ For contiguous elements, } inx=1. \)

SCNRM2 computes

\[ \text{eucnorm} = \left( \sum_{i=1}^{n} x_i \bar{x}_i \right)^{1/2} \]

where \( x_i \) is a complex vector and \( \bar{x}_i \) is the complex conjugate of \( x_i \).

Call from FORTRAN:

\[ \text{eucnorm} = \text{SCNRM2}(n, cx, inx) \]

\( n \quad \text{Number of elements in vector} \)

\( cx \quad \text{Complex vector operand} \)

\( inx \quad \text{Skip distance between elements of } cx. \text{ For contiguous elements, } inx=1. \)

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:

\[ \text{CALL SROTG}(a, b, c, s) \]
Scalar a of equation 1

Scalar b of equation 1

Scalar cosine of equation 1

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

\[ r = \text{sgn}(a) * \sqrt{a^2 + b^2} \text{ if } |a| > |b| \]

\[ = \text{sgn}(b) * \sqrt{a^2 + b^2} \text{ if } |a| < |b| \]

and

\[ c = a/r \text{ if } r \neq 0 \]

\[ = 1 \text{ if } r = 0 \]

\[ s = b/r \text{ if } r \neq 0 \]

\[ = 0 \text{ if } r = 0. \]

Parameter z is

\[ z = s \text{ if } |a| > |b| \text{ or } a=b=0 \]

\[ = 1/c \text{ if } 0 < |a| < |b| \]

\[ = 1 \text{ if } |b| > |a| = 0 \]

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 \( c^2 + s^2 = 1.0 \), 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:

```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.

```fortran
CALL SROTMG(d1, d2, b1, b2, param)
```
SROTMG sets up parameters \( \text{param} \) from inputs \( d_1, d_2, b_1, b_2 \). The following is a brief description.

An application of the Givens plane rotation

\[
\begin{bmatrix}
  x' \\
  y
\end{bmatrix} =
\begin{bmatrix}
  c & s \\
  -s & c
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix} G & x \\ y \end{bmatrix}
\]

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

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = D^{1/2} H \begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]

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:

\[
\begin{bmatrix}
  x' \\
  o
\end{bmatrix} = D^{1/2} H \begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]
Equation 4:
\[
\begin{pmatrix}
\sqrt{d_1} & h_{11} & \sqrt{d_1} & h_{12} \\
\sqrt{d_2} & h_{21} & \sqrt{d_2} & h_{22}
\end{pmatrix}
= \begin{pmatrix}
\sqrt{d_1} & d_2s \\
-\sqrt{d_1} & d_2c
\end{pmatrix}
\]

Coefficients \(c\) and \(s\) are determined by equations 5 and 6.

Equation 5:
\[
c = \frac{x}{\sqrt{x^2+y^2}} = \frac{d_1b_1}{\sqrt{d_1b_1^2 + d_2b_2^2}}
\]

Equation 6:
\[
s = \frac{y}{\sqrt{x^2+y^2}} = \frac{d_2b_2}{\sqrt{d_1b_1^2 + d_2b_2^2}}
\]

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 \(|s|>|c|\), 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 \(|d_1b_1^2|>|d_2b_2^2|\)), the solutions for equation 4 are determined by equation 7.

Equation 7:
\[
h_{11}=h_{22}=1
\]

Case 2:

If \(|s|>|c|\) (which is \(|d_2b_2^2|>|d_1b_1^2|\)), equation 8 is chosen.
Equation 8:

\[ h_{12} = -h_{21} = 1. \]

Distinguishing the two cases, \(|c| > \frac{1}{\sqrt{2}}\) or \(|s| > \frac{1}{\sqrt{2}}\) 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 \(|d_{11} b_{11}^2| > |d_{22} b_{22}^2|\), the following solutions for \(H\) are chosen:

\[
\begin{align*}
    h_{11} &= 1, & h_{12} &= \frac{d_{22} b_{22}}{d_{11} b_{11}} \\
    h_{21} &= \frac{-b_{22}}{b_{11}}, & h_{22} &= 1
\end{align*}
\]

and scale factors \(d_1, \ d_2\) are updated to

\[
\begin{align*}
    d'_1 &= d_1 / u = \sigma^2 \ d_1 \\
    d'_2 &= d_2 / u = \sigma^2 \ d_2
\end{align*}
\]

where

\[
u = \det (H) = 1 + \frac{d_{22} b_{22}^2}{d_{11} b_{11}^2}
\]

and since \(x' = r, y' = 0\), and \(b_1' = x' / d_1\) then \(b_1 = b_1 \cdot u\) is updated.

Case 2:

In case 2, where \(|s| > |c|\) or \(|d_{11} b_{11}^2| < |d_{22} b_{22}^2|\), the following solutions for \(H\) are chosen.

\[
\begin{align*}
    h_{11} &= \frac{d_{11} b_{11}}{d_{22} b_{22}}, & h_{12} &= 1 \\
    h_{21} &= -1, & h &= \frac{b_1}{b_2}
\end{align*}
\]
and scale factors $d_i$ are updated to

\[
\begin{align*}
d_1' &= d_2/u \\
d_2' &= d_1/u
\end{align*}
\]

with

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

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

\[
(m)^{-2} < |d_i| < (m)^2
\]

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.

\[
\begin{align*}
d_i' &= d_i' (m)^2, \\
h_{i1}' &= h_{i1}' (m)^{-1}, \\
b_{1}' &= b_{1}' (m)^{-1}, \\
h_{i2}' &= h_{i2}' (m)^{-1}.
\end{align*}
\]

Overflow:

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

\[
\begin{align*}
d_i' &= d_i' (m)^{-2}, \\
h_{i1}' &= h_{i1}' (m), \\
b_{1}' &= b_{1}' (m), \\
h_{i2}' &= h_{i2}' (m).
\end{align*}
\]

SROTMG modifies the input parameters D1, D2, and B1 and returns the array PARAM according to the following schedule:
Case 4:

If \( \text{ABS}(D_1 \cdot B_1 \cdot B_1) \cdot \text{ABS}(D_2 \cdot B_2 \cdot B_2) < \), then

\[
\begin{align*}
\text{PARAM}(1) &= 0 \\
\text{PARAM}(3) &= -B_2/B_1 \\
\text{PARAM}(4) &= D_2 \cdot B_2 / D_1 \cdot B_1
\end{align*}
\]

and parameters \( D_1, D_2, \) and \( B_1 \) are written over by

\[
\begin{align*}
D_1 &= D_1 / U \\
D_2 &= D_2 / U \\
B_1 &= B_1 \cdot U
\end{align*}
\]

where

\[
U = 1 + (D_2 \cdot B_2 \cdot B_2) / (D_1 \cdot B_1 \cdot B_1).
\]

Case 5:

If \( \text{ABS}(D_2 \cdot B_2 \cdot B_2) \cdot \text{ABS}(D_2 \cdot B_2 \cdot B_2) \cdot \text{ABS}(D_1 \cdot B_1 \cdot B_1), \) then

\[
\begin{align*}
\text{PARAM}(1) &= 1. \\
\text{PARAM}(2) &= D_1 \cdot B_1 / D_2 \cdot B_2 \\
\text{PARAM}(5) &= B_1 / B_2
\end{align*}
\]

and parameters \( D_1, D_2, \) and \( B_1 \) are written over according to the following sequence.

\[
\begin{align*}
\text{TEMP} &= D_1 / U \\
D_1 &= D_2 / U \\
D_2 &= \text{TEMP} \\
B_1 &= B_2 \cdot U
\end{align*}
\]

\[
U = 1 + (D_1 \cdot B_1 \cdot B_1) / (D_2 \cdot B_2 \cdot B_2).
\]
Case 6:

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

\[(m)^{\text{-2}} \leq |D1| \leq (m)^{\text{2}}\]

\[(m)^{\text{-2}} \leq |D2| \leq (m)^{\text{2}}\]

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

- \text{PARAM}(1) = -1.
- \text{PARAM}(2) = H11
- \text{PARAM}(3) = H21
- \text{PARAM}(4) = H12
- \text{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, \(h_{11}=h_{12}=h_{21}=h_{22}=0\))

\text{PARAM}(1) = -1, and the rest of the elements of \text{PARAM} contain zero.

Case 7:

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

\text{PARAM}(1) = -2.0 only.

\text{APPLY MODIFIED GIVENS PLANE ROTATION}

\text{SROTM} applies the modified Givens plane rotation constructed by \text{SROTMG}.

\[
\text{CALL SROTM}(n, sx, incx, sy, incy, param)\]

SR-0014 4-19 I
computes

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

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,inax)

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

\[ X = aX \]

where \( a \) is a real number and \( X \) is a complex vector.

Call from FORTRAN:

```fortran
CALL CSSCAL(n,sa,ex,inex)
```

- \( n \) Number of elements in vector
- \( sa \) Real scaling factor
- \( ex \) Complex vector to be scaled
- \( inex \) Skip distance between elements of \( ex \). For contiguous elements, \( inex=1 \).

CSCAL computes

\[ Y = aY \]

where \( a \) is a complex number and \( Y \) is a complex vector.

Call from FORTRAN:

```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:

\[
\text{CALL SSWAP}(n, \text{x}, \text{incx}, \text{y}, \text{incy})
\]

- \( n \) Number of elements in vector
- \( \text{x} \) One real vector
- \( \text{incx} \) Skip distance between elements of \( \text{x} \). For contiguous elements, \( \text{incx}=1 \).
- \( \text{y} \) Another real vector
- \( \text{incy} \) Skip distance between elements of \( \text{y} \). For contiguous elements, \( \text{incy}=1 \).

CSWAP exchanges two complex vectors.

Call from FORTRAN:

\[
\text{CALL CSWAP}(n, \text{a}, \text{inax}, \text{ay}, \text{incy})
\]

- \( n \) Number of elements in vector
- \( \text{a} \) One complex vector
- \( \text{inax} \) Skip distance between elements of \( \text{a} \). For contiguous elements, \( \text{inax}=1 \).
- \( \text{ay} \) Another complex vector
- \( \text{incy} \) Skip distance between elements of \( \text{ay} \). For contiguous elements, \( \text{incy}=1 \).

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.

\[
\begin{array}{l}
\text{DO 10 } I=1,N \\
10 \quad \text{SY(INDEX(I))=SA*SX(I)+SY(INDEX(I))}
\end{array}
\]

Call from FORTRAN:

\[
\text{CALL SPAXPY}(n,sa,sx,sysy,index)
\]

- \textit{n} Number of elements in the vectors
- \textit{sa} Real scalar multiplier
- \textit{sx} Real vector operand
- \textit{sy} Real vector operand
- \textit{index} Vector of indexes

SPODT is defined in FORTRAN in the following way.

\[
\begin{array}{l}
\text{DO 10 } I=1,N \\
10 \quad \text{PDOT=PDOT+SY(INDEX(I))*SX(I)}
\end{array}
\]

Call from FORTRAN:

\[
\text{pdot=SPDOT}(n,sy,index,sx)
\]

- \textit{n} Number of elements in the vectors
- \textit{sy} Real vector operand
- \textit{sx} Real vector operand
- \textit{index} Vector of indexes
### Table 4-2. Other linear algebra subprograms

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

**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

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

where $x_j$ is an element of a real vector.
Call from FORTRAN:

\[ \text{imax}=\text{ISMAX}(n,ax,\text{incx}) \]

- \( n \): Number of elements to process in the vector to be searched
  (\( n \)=vector length if \( \text{incx}=1 \); \( n \)=vector length/2 if \( \text{incx}=2 \); etc.)
- \( ax \): Real vector to be searched
- \( \text{incx} \): Skip distance between elements of \( ax \). For contiguous elements, \( \text{incx}=1 \).

\[ |x_i|=\min \{ |x_j| : j=1, \ldots, n \} \]

where \( x_j \) is an element of a real vector.

Call from FORTRAN:

\[ \text{imin}=\text{ISMIN}(n,ax,\text{incx}) \]

- \( n \): Number of elements to process in the vector to be searched
  (\( n \)=vector length if \( \text{incx}=1 \); \( n \)=vector length/2 if \( \text{incx}=2 \); etc.)
- \( ax \): Real vector to be searched
- \( \text{incx} \): Skip distance between elements of \( ax \). For contiguous elements, \( \text{incx}=1 \).

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.

\[ |x_i|=\min \{ |x_j| : j=1, \ldots, n \} \]

where \( x_j \) is an element of a real vector.
Call from FORTRAN:

\[ \text{imin} = \text{ISAMIN}(n, sx, \text{incx}) \]

- \( n \): Number of elements to process in the vector to be searched
  - \( n \) = vector length if \( \text{incx} = 1 \);
  - \( n \) = vector length/2 if \( \text{incx} = 2 \); etc.

- \( sx \): Real vector to be searched

- \( \text{incx} \): Skip distance between elements of \( sx \). For contiguous elements, \( \text{incx} = 1 \).

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:

\[ \text{sum} = \text{SSUM}(n, sx, \text{incx}) \]

- \( n \): Number of elements in vector

- \( sx \): Real vector to be summed

- \( \text{incx} \): Skip distance between elements of \( sx \). For contiguous elements, \( \text{incx} = 1 \).

CSUM sums the elements of a complex vector.

Call from FORTRAN:

\[ \text{sum} = \text{CSUM}(n, cx, \text{incx}) \]

- \( n \): Number of elements in vector

- \( cx \): Complex vector to be summed

- \( \text{incx} \): Skip distance between elements of \( cx \). For contiguous elements, \( \text{incx} = 1 \).
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:

```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:

```fortran
CALL CROT(n,cx,incx,cy,incy,cc,cs)
```
\( n \) Number of elements in vector
\( cx \) Complex vector to be modified
\( \text{inex} \) Skip distance between elements of \( cx \). For contiguous elements, \( \text{inex} = 1 \).
\( cy \) Complex vector to be modified
\( \text{inay} \) Skip distance between elements of \( cy \). For contiguous elements, \( \text{inay} = 1 \).
\( cc \) 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:
\[
\text{\texttt{x=SMACH}}(\text{\texttt{job}})
\]

SMACH returns the following information.

\[
\begin{align*}
\text{\texttt{job}} &= 1 &\quad 0.7105E-14 &\quad \text{The machine epsilon (the smallest number } \varepsilon \text{ such that } 1.\pm \varepsilon \neq 1.)
\text{\texttt{job}} &= 2 &\quad 0.1290E-2449 &\quad \text{A number close to smallest normalized, representable number}
\text{\texttt{job}} &= 3 &\quad 0.7750E+2450 &\quad \text{A number close to largest normalized, representable number}
\end{align*}
\]

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

Call from FORTRAN:
\[
\text{\texttt{x=CMACH}}(\text{\texttt{job}})
\]
CMACH returns the following information.

for \(job=1\) \[0.7105E-14\] The machine epsilon (the smallest number \(\epsilon\) such that \(1.1 \pm \epsilon \neq 1\.)

\(=2\) \[.1348E12\] A number close to the square root of the smallest normalized, representable number

\(=3\) \[.7421E12\] 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:

\[c_1=b_1\]

\[c_i=-a_i c_{i-1}+b_i\] for \(i=2,3\ldots,n\)

or in FORTRAN,

\[
\text{EQUIVALENCE (B,C)}
\]

\[C(1)=B(1)\]

\[\text{DO 10 } I=2,N\]

\[10 \quad C(I)=-A(I)*C(I-1)+B(I)\]

Call from FORTRAN:

\[
\text{CALL FOLR}(n,a,inca,b,incb)
\]
\begin{verbatim}
\textbf{CALL FOLRP}(n,a,inca,b,inab)
\end{verbatim}

\begin{itemize}
    \item \textbf{n} \quad \text{Length of linear recurrence}
    \item \textbf{a} \quad \text{Vector } a \text{ of length } n \text{ of equation 12. (}A(1)\text{ is arbitrary.)}
    \item \textbf{inca} \quad \text{Skip distance between elements of the vector operand } A. \text{ For contiguous elements, } \text{inca}=1.
    \item \textbf{b} \quad \text{Vector } b \text{ of equation 12 on input and vector } c \text{ of equation 12 on output. (The output overwrites the input.)}
    \item \textbf{inab} \quad \text{Skip distance between elements of the vector operand } b \text{ and result } C. \text{ For contiguous elements, } \text{inab}=1.
\end{itemize}

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

Equation 12:
\[
c_1 = b_1 \\
c_i = a_i c_{i-1} + b_i \quad \text{for } i=2,3\ldots,n
\]

or in FORTRAN:
\[
\text{EQUIVALENCE}(B,C) \\
C(1)=B(1) \\
\text{DO } 10 \text{ I}=2,N \\
10 \quad C(I)=A(I)*C(I-1)+B(I)
\]

Call from FORTRAN:
FOLR2 solves first-order linear recurrences as in equation 11. The solution, however, is written to a vector \( a \), 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, \( incb = 1 \).
a

Vector a 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

$$
\begin{align*}
  r_1 &= b_1 \\
  r_i &= a_i r_{i-1} + b_i & i &= 2, 3, \ldots, n
\end{align*}
$$

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.

\[
\begin{align*}
  \text{Let } p(x) &= \sum_{i=0}^{n} b_i x^{n-i} \\
  \text{then } p(a) &= (((b_0 a + b_1) a + b_2) a + \ldots + b_n) \text{ Horner's rule.}
\end{align*}
\]
In FORTRAN:

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

or equivalently

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

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

Equation 12:

\[ c_{i+2} = a_i c_{i+1} + b_i c_i \quad \text{for } i=1,2,\ldots \]

or in FORTRAN,

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

Call from FORTRAN:

```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; \( C(3), C(4), \ldots, C(N+2) \) 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

\[
\begin{pmatrix}
  a_i & b_i \\
  1 & 0
\end{pmatrix}
\begin{pmatrix}
  c_2 \\
  c_1
\end{pmatrix}
= \begin{pmatrix}
  r_n \\
  r_{n-1}
\end{pmatrix}
\]

with the following call.

\[
m = \text{SOLRN}(n,a,1,b,1,c,1)
\]

The equivalent FORTRAN follows.

```fortran
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

\[
c_1 = c_1 \\
c_2 = c_2 \\
c_i = c_{i-3} + a_{i-2} c_{i-1} + b_{i-2} c_{i-2} \quad i = 3, \ldots, n
\]

Call from FORTRAN:

```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.

\[
Lx = \begin{pmatrix}
1 & 0 & 0 & \cdots & 0 \\
0 & e_1 & 1 & 0 & \cdots \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & f_{n-2} & e_{n-1} \\
0 & 0 & \cdots & 0 & f_n \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_n \\
\end{pmatrix}
= \begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_n \\
\end{pmatrix}
\]

then there is

\[
x_1 = b_1 \\
x_2 = b_2 - e_1 x_1 \\
x_i = b_i - e_{i-1} x_{i-1} - f_{i-2} x_{i-2} \quad i=3, \ldots, n
\]

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)
```

SR-0014
SINGLE-PRECISION REAL AND COMPLEX LINPACK Routines


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, SPOS, STSLS, 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.

Table 4-3. Single-precision LINPACK routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix or Decomposition</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGECO</td>
<td>Real general</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>SGEFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>SGESL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>SGEDI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
<tr>
<td>CGECO</td>
<td>Complex general</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>CGEFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>CGESL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>CGEDI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
<tr>
<td>SGBCO</td>
<td>Real general banded</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>SGBFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>SGBSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>SGBDI</td>
<td></td>
<td>Compute determinant</td>
</tr>
<tr>
<td>CGBCO</td>
<td>Complex general banded</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>CGBFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>CGBSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>CGBDI</td>
<td></td>
<td>Compute determinant</td>
</tr>
<tr>
<td>SPOCO</td>
<td>Real positive definite</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>SPOFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>SPDSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>SPODI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
<tr>
<td>CPOCO</td>
<td>Complex positive definite</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>CPOFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>CPOS L</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>CPODI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
<tr>
<td>SPPCO</td>
<td>Real positive definite packed</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>SPPFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>SPSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>SPPDI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
<tr>
<td>CPPCO</td>
<td>Complex positive definite packed</td>
<td>Factor and estimate condition</td>
</tr>
<tr>
<td>CPPFA</td>
<td></td>
<td>Factor</td>
</tr>
<tr>
<td>CPPSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>CPPDI</td>
<td></td>
<td>Compute determinant and inverse</td>
</tr>
</tbody>
</table>
Table 4-3. Single-precision LINPACK routines (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix or Decomposition</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPBCO</td>
<td>Real positive definite banded</td>
<td>Factor and estimate condition Factor Solve Compute determinant</td>
</tr>
<tr>
<td>SPBFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPBSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPBDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBCO</td>
<td>Complex positive definite banded</td>
<td>Factor and estimate condition Factor Solve Compute determinant</td>
</tr>
<tr>
<td>CPBFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPBDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSICO</td>
<td>Symmetric indefinite</td>
<td>Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse</td>
</tr>
<tr>
<td>SSIFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSISL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSIDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHICO</td>
<td>Hermitian indefinite</td>
<td>Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse</td>
</tr>
<tr>
<td>CHIFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHISL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSPCO</td>
<td>Symmetric indefinite packed</td>
<td>Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse</td>
</tr>
<tr>
<td>SSPFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSPSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSPDII</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHPCO</td>
<td>Hermitian indefinite packed</td>
<td>Factor and estimate condition Factor Solve Compute inertia, determinant, and inverse</td>
</tr>
<tr>
<td>CHPFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHPSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHPDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRCO</td>
<td>Real triangular</td>
<td>Factor and estimate condition Factor Solve Compute determinant and inverse</td>
</tr>
<tr>
<td>STRSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTRCO</td>
<td>Complex triangular</td>
<td>Factor and estimate condition Factor Solve Compute determinant and inverse</td>
</tr>
<tr>
<td>CTRSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTRDI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4-3. Single-precision LINPACK routines (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix or Decomposition</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGTSL</td>
<td>Real tridiagonal</td>
<td>Solve</td>
</tr>
<tr>
<td>CGTSL</td>
<td>Complex tridiagonal</td>
<td>Solve</td>
</tr>
<tr>
<td>SPTSL</td>
<td>Real positive definite</td>
<td>Solve</td>
</tr>
<tr>
<td></td>
<td>tridiagonal</td>
<td></td>
</tr>
<tr>
<td>CPTSL</td>
<td>Complex</td>
<td>Solve</td>
</tr>
<tr>
<td>SCHDC</td>
<td>Real Cholesky decomposition</td>
<td>Decompose</td>
</tr>
<tr>
<td>SCHDD</td>
<td></td>
<td>Downdate</td>
</tr>
<tr>
<td>SCHUD</td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>SCHEX</td>
<td></td>
<td>Exchange</td>
</tr>
<tr>
<td>CCHDC</td>
<td>Complex Cholesky decomposition</td>
<td>Decompose</td>
</tr>
<tr>
<td>CCHDD</td>
<td></td>
<td>Downdate</td>
</tr>
<tr>
<td>CCHUD</td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>CCHEX</td>
<td></td>
<td>Exchange</td>
</tr>
<tr>
<td>SQRDC</td>
<td>Real</td>
<td>Orthogonal factorization</td>
</tr>
<tr>
<td>SQRSL</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>CQRDC</td>
<td>Complex</td>
<td>Orthogonal factorization</td>
</tr>
<tr>
<td>CQRS</td>
<td></td>
<td>Solve</td>
</tr>
<tr>
<td>SSVDC</td>
<td>Real</td>
<td>Singular value</td>
</tr>
<tr>
<td>CSVDC</td>
<td>Complex</td>
<td>decomposition</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix or Decomposition</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>Complex symmetric</td>
<td></td>
</tr>
<tr>
<td>RG</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>RGG</td>
<td>Real general generalize</td>
<td>Find eigenvalues and eigenvectors (as desired)</td>
</tr>
<tr>
<td></td>
<td>$Ax = \lambda Bx$</td>
<td></td>
</tr>
<tr>
<td>RS</td>
<td>Real symmetric</td>
<td></td>
</tr>
<tr>
<td>RSB</td>
<td>Real symmetric band</td>
<td></td>
</tr>
<tr>
<td>RSG</td>
<td>Real symmetric generalize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$Ax = \lambda Bx$</td>
<td></td>
</tr>
<tr>
<td>RSGAB</td>
<td>Real symmetric generalize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$ABx = \lambda x$</td>
<td></td>
</tr>
<tr>
<td>RSGBA</td>
<td>Real symmetric generalize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$BAb = \lambda x$</td>
<td></td>
</tr>
<tr>
<td>RSP</td>
<td>Real symmetric packed</td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td>Real symmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>Special real tridiagonal</td>
<td></td>
</tr>
<tr>
<td>BALANC</td>
<td>Real general</td>
<td>Balances matrix and isolates eigenvalues whenever possible</td>
</tr>
<tr>
<td>CBAL</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Matrix or Decomposition</td>
<td>Purpose</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>ELMHES</td>
<td>Real general</td>
<td>Reduce matrix to upper Hessenberg form</td>
</tr>
<tr>
<td>ORTHES</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>COMHES</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>COMTH</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>ELTRAN</td>
<td>Real general</td>
<td>Accumulate transformations used in the reduction to upper Hessenberg form done by ELMHES, ORTHES</td>
</tr>
<tr>
<td>ORTRAN</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>BALBAK</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>ELMBAK</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>ORTBAK</td>
<td>Real general</td>
<td></td>
</tr>
<tr>
<td>COMBAK</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>CORTB</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>CBABK2</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>REBAK</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>REBAKB</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>TREOI</td>
<td>Complex general</td>
<td></td>
</tr>
<tr>
<td>TRED1</td>
<td>Real symmetric</td>
<td>Reduce to symmetric tridiagonal</td>
</tr>
<tr>
<td>TRED2</td>
<td>Real symmetric</td>
<td></td>
</tr>
<tr>
<td>TRED3</td>
<td>Real symmetric</td>
<td></td>
</tr>
<tr>
<td>TRBAK1</td>
<td>Real symmetric</td>
<td>Form eigenvectors by back transforming those of the corresponding matrices determined by TRED1 or TRED3</td>
</tr>
<tr>
<td>TRBAK3</td>
<td>Real symmetric</td>
<td></td>
</tr>
<tr>
<td>IMTQLV</td>
<td>Symmetric tridiagonal</td>
<td>Find eigenvalues and/or eigenvectors by implicit QL method</td>
</tr>
<tr>
<td>IMTQL1</td>
<td>Symmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>IMTQL2</td>
<td>Symmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>RATQR</td>
<td>Symmetric tridiagonal</td>
<td>Find the smallest or largest eigenvalues by rational QR method with Newton corrections</td>
</tr>
<tr>
<td>Name</td>
<td>Matrix or Decomposition</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TQLRAT</td>
<td>Symmetric tridiagonal</td>
<td>Find the eigenvalues by rational QL method</td>
</tr>
<tr>
<td>TQL1</td>
<td>Symmetric tridiagonal</td>
<td>Find the eigenvalues and/or eigenvectors by the rational QL or QL method</td>
</tr>
<tr>
<td>TQL2</td>
<td>Symmetric tridiagonal</td>
<td>Find eigenvalues and/or eigenvectors which lie in a specified interval using bisection and/or inverse iteration</td>
</tr>
<tr>
<td>BISECT</td>
<td>Symmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>TRIDIB</td>
<td>Nonsymmetric tridiagonal</td>
<td>Reduce to symmetric tridiagonal with the same eigenvalues</td>
</tr>
<tr>
<td>TSTURM</td>
<td>Nonsymmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>TINVIT</td>
<td>Nonsymmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>FIGI</td>
<td>Nonsymmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>FIGI2</td>
<td>Nonsymmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>BAKVEC</td>
<td>Nonsymmetric tridiagonal</td>
<td></td>
</tr>
<tr>
<td>HQR</td>
<td>Real upper Hessenberg</td>
<td>Find eigenvalues and/or eigenvectors by QR method</td>
</tr>
<tr>
<td>HQR2</td>
<td>Complex upper Hessenberg</td>
<td></td>
</tr>
<tr>
<td>COMQR</td>
<td>Complex upper Hessenberg</td>
<td></td>
</tr>
<tr>
<td>COMQR2</td>
<td>Complex upper Hessenberg</td>
<td></td>
</tr>
<tr>
<td>INVIT</td>
<td>Upper Hessenberg</td>
<td>Find eigenvectors corresponding to specified eigenvalues</td>
</tr>
<tr>
<td>CINVIT</td>
<td>Complex upper Hessenberg</td>
<td></td>
</tr>
<tr>
<td>BANDR</td>
<td>Real symmetric banded</td>
<td>Reduce to a symmetric tridiagonal matrix</td>
</tr>
<tr>
<td>BANDV</td>
<td>Real symmetric banded</td>
<td>Find those eigenvectors corresponding to specified eigenvalues using inverse iteration</td>
</tr>
</tbody>
</table>
Table 4-4. Single-precision EISPACK routines (continued)

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

```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(n,nd) + 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:

```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,nbe)
**Example 1:**

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

\[
\begin{array}{ccc}
(1,1)^* & (1,2)^* & (1,3)^* \\
\downarrow & (a) & \\
(2,1) & (2,2) & (2,3) \\
\downarrow & (b) & \to (c)
\end{array}
\]

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.

\[
\begin{array}{ccc}
(1,1) & (1,2) & (1,3) \\
(a) & (b) \\
(2,1) & (2,2) & (2,3) \\
(3,1) & (3,2) & (3,3)
\end{array}
\]

Matrix $A$

\[
\begin{array}{ccc}
(1,1) & (2,1) & (3,1) \\
(a) & (b) & (c)
\end{array}
\]

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.

\[
\text{CALL MXMA}(A,1,3,\ldots)
\]

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

\[
\text{CALL MXMA}(A,3,1,\ldots)
\]

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

Call from FORTRAN:

\[
\text{CALL MXVA}(a,na,iad,b,nb,c,nar,nbr)
\]

- $a$ 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: \((\text{init}, \text{ix}, n, x, \text{work}, y)\).

- \( \text{init} \) Initialization flag
- \( \text{ix} \) Analysis/synthesis flag
- \( n \) Size of transform
- \( x \) Input vector
- \( \text{work} \) Working storage vector
- \( y \) Result vector

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>CFPT2</th>
<th>RCPFT2</th>
<th>CRFFT2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Complex $n$</td>
<td>Real $n$</td>
<td>Complex $(n/2)+1$</td>
</tr>
<tr>
<td>$x$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$work$</td>
<td>Complex $(5/2)n$</td>
<td>Complex $(3/2)n+2$</td>
<td>Complex $(3/2)n+2$</td>
</tr>
<tr>
<td>$y$</td>
<td>Complex $n$</td>
<td>Complex $(n/2)+1$</td>
<td>Real $n$</td>
</tr>
</tbody>
</table>

CFPT2 calculates equation 13.

Equation 13:

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

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 CFPT2(init,ix,n,x,work,y)
```

- $init \neq 0$ Generates sine and cosine tables in work
- $init = 0$ Calculates Fourier transforms using sine and cosine tables of previous call
- $ix > 0$ Calculates Fourier analysis
- $ix < 0$ 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.
\[ y_{k}=\sum_{j=0}^{n-1} x_{j} \exp\left(\pm \frac{2\pi i}{n} jk\right) \]

for \( k = 0, 1, \ldots, (n/2) \)

where \( x_{i} i=0,1,\ldots,n-1 \) are stored in \( X(I), I=1,N \)

\( y_{i} i=0,1,\ldots,n/2 \) are stored in \( Y(I), I=1,(N/2)+1 \)

and the sign of the exponent is determined by \( \text{SIGN}(IX) \).

Call from FORTRAN:

```fortran
CALL RCFPT2(init,ix,n,x,work,y)
```

- \( \text{init} \neq 0 \) Generates sine and cosine tables in \( \text{work} \)
- \( \text{init} = 0 \) Calculates Fourier transforms using sine and cosine tables of previous call
- \( \text{init} < 0 \) Calculates Fourier synthesis

- \( \text{ix} > 0 \) Calculates Fourier analysis
- \( \text{ix} < 0 \) Calculates Fourier synthesis

- \( n \) Size of Fourier transform; \( 2^{m} \) where \( 3 \leq m \).

- \( x \) Input vector. Vector of \( n \) real values
  Range: \( 10^{-2466}/n \geq x(i) > n \cdot (10^{-2466}) \) for \( i=1,n \).

- \( \text{work} \) 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 $j=0,1,\ldots,(n/2)$. Only the first $(n/2)+1$ elements are stored in $X$.

Equation 13:

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

for $k=0,1,\ldots,n-1$

where the $x_j$ elements are complex and are related by $x_j = x_{n-j}$

for $j=1,2,3,\ldots,(n/2)$

Call from FORTRAN:

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

`init`  
$\neq 0$ Generates sine and cosine tables in `work`  
$=0$ Calculates Fourier transforms using sine and cosine tables of previous call

`ix`  
$>0$ Calculates Fourier analysis  
$\leq 0$ 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: $10^{-2466}/n \leq x(i) \leq 10^{-2466}$; $i=1,n$.

`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.

SR-0014 4-53 I
FILTERG computes a convolution of two vectors.

Given:

\[(a_i) \quad i=1, \ldots, m \quad \text{Filter coefficients}\]
\[(d_j) \quad j=1, \ldots, n \quad \text{Data}\]

FILTERG computes the following.

\[o_i = \sum_{j=1}^{m} a_j d_{i+j-1} \quad i=1, \ldots, n-m+1\]

Call from FORTRAN:

\[
\text{CALL FILTERG}(a, m, d, n, o)
\]

\[a \quad \text{Vector of filter coefficients}\]
\[m \quad \text{Number of filter coefficients}\]
\[d \quad \text{Input data vector}\]
\[n \quad \text{Number of data points}\]
\[o \quad \text{Output vector}\]

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

Given:

\[(c_i) \quad i=1, \ldots, \lceil m/2 \rceil \quad (\lceil m/2 \rceil = m/2 \text{ for } m \text{ even and } (m+1)/2 \text{ for } m \text{ odd, called the ceiling function.})\]
\[(d_i) \quad j=1, \ldots, n\]

FILTERS computes the following.

\[m \text{ odd: } o_i = \sum_{j=1}^{(m-1)/2} a_j (d_{i+j-1} + d_{i+m-j})\]
\[+ \sum_{j=1}^{\lceil m/2 \rceil} d_{i+j-1} d_j \quad i=1, \ldots, n-m+1\]

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

$$t_{ij}=R(k) \quad \text{for } |j-i|+1=k$$
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 $2m$
- **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.

\[a_i = b_j\] where \(i=1,\ldots,n\)

In FORTRAN:

\[A(I) = B(INDEX(I))\]

where \(I=1,N\)

Call from FORTRAN:

\[
\text{CALL GATHER}(n,a,b,index)
\]

- \(n\) Number of elements in each vector
- \(a\) Output vector
- \(b\) Source vector
- \(index\) Vector of indexes
SCATTER is defined in the following way.

\[ a_j = b_i \text{ where } i=1,\ldots,n \]

In FORTRAN:

\[ A(INDEX(I)) = B(I) \]

where \( I=1,N \)

Call from FORTRAN:

\begin{verbatim}
CALL SCATTER(n, a, index, b)
\end{verbatim}

- \( n \): 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

ILLZ 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:

\[ kount=ILLZ(n,l,\text{incl}) \]

\( n \) 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

\( \text{incl} \) Skip distance between elements of the vector operand. For contiguous elements, \( \text{incl}=1 \).

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:

\[ kount=ILLZ(n,l,\text{incl}) \]

\( n \) 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

\( \text{incl} \) Skip distance between elements of the vector operand. For contiguous elements, \( \text{incl}=1 \).

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:

\[ kount=ILSUM(n,l,\text{incl}) \]
Number of elements to process in the vector \( n = \text{vector length}, \) if \( incl = 1; \) \( n = \text{vector length}/2, \) if \( incl = 2; \) etc.

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

<table>
<thead>
<tr>
<th>Name parameter list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISRCHEQ(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is equal to the real target</td>
</tr>
<tr>
<td>ISRCHNE(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is not equal to the real target</td>
</tr>
<tr>
<td>ISRCHFLT(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is less than the real target</td>
</tr>
<tr>
<td>ISRCHFLE(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is less than or equal to the real target</td>
</tr>
<tr>
<td>ISRCHFGT(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is greater than the real target</td>
</tr>
<tr>
<td>ISRCHFGE(( n ),array,inc,target)</td>
<td>Returns the first location in a real array that is greater than or equal to the real target</td>
</tr>
</tbody>
</table>
Table 4-6. ISRCH routines (continued)

<table>
<thead>
<tr>
<th>Name (parameter list)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISRCHEQ(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is equal to the integer target</td>
</tr>
<tr>
<td>ISRCHNE(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is not equal to the integer target</td>
</tr>
<tr>
<td>ISRCHILT(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is less than the integer target</td>
</tr>
<tr>
<td>ISRCHILE(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is less than or equal to an integer target</td>
</tr>
<tr>
<td>ISRCHIGT(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is not equal to an integer target</td>
</tr>
<tr>
<td>ISRCHIGE(n,iarray,inc,itarget)</td>
<td>Returns the first location in an integer array that is not equal to an integer target</td>
</tr>
</tbody>
</table>

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:

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

- \( n \) Number of elements to be searched. If \( n < 0 \), 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.

```fortran
FUNCTION ISRCHNE(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 ISRCHNE=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:

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

n Number of elements to be searched. If n<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:

```fortran
location=ISRCHFLT(n,array,inc,target)
```
\( n \) \hspace{1cm} \text{Number of elements to be searched. If } n < 0, \text{ then } 0 \text{ is returned.}

\text{array} \hspace{1cm} \text{First element of real array to be searched}

\text{inc} \hspace{1cm} \text{Skip distance between elements of the searched array}

\text{target} \hspace{1cm} \text{Real value searched for in array. If } target \text{ 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:

\begin{verbatim}
location=ISRCHFLE(n,array,inc,target)
\end{verbatim}

\( n \) \hspace{1cm} \text{Number of elements to be searched. If } n < 0, \text{ then } 0 \text{ is returned.}

\text{array} \hspace{1cm} \text{First element of real array to be searched}

\text{inc} \hspace{1cm} \text{Skip distance between elements of the searched array}

\text{target} \hspace{1cm} \text{Real value searched for in array. If } target \text{ 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:

\begin{verbatim}
location=ISRCHFGT(n,array,inc,target)
\end{verbatim}

\( n \) \hspace{1cm} \text{Number of elements to be searched. If } n < 0, \text{ then } 0 \text{ is returned.}

\text{array} \hspace{1cm} \text{First element of real array to be searched}

\text{inc} \hspace{1cm} \text{Skip distance between elements of the searched array}

\text{target} \hspace{1cm} \text{Real value searched for in array. If } target \text{ 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<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<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<0`, then 0 is returned.
space

\textit{iarray} \quad \text{First element of integer array to be searched}

\textit{inc} \quad \text{Skip distance between elements of the searched array}

\textit{itarget} \quad \text{Integer value searched for in array. If \textit{itarget} is not found, then the returned value is } n+1.

\textsc{ISrchigt} is used when the integer array element is greater than the integer target.

\textbf{Call from FORTRAN:}

\begin{verbatim}
location=ISrchigt(n,iarray,inc,itarget)
\end{verbatim}

\textit{n} \quad \text{Number of elements to be searched. If } n<0, \text{ then } 0 \text{ is returned.}

\textit{iarray} \quad \text{First element of integer array to be searched}

\textit{inc} \quad \text{Skip distance between elements of the searched array}

\textit{itarget} \quad \text{Integer value searched for in array. If \textit{itarget} is not found, then the returned value is } n+1.

\textsc{ISrchigE} is used when the integer array element is greater than or equal to the integer target.

\textbf{Call from FORTRAN:}

\begin{verbatim}
location=ISrchigE(n,iarray,inc,itarget)
\end{verbatim}

\textit{n} \quad \text{Number of elements to be searched. If } n<0, \text{ then } 0 \text{ is returned.}

\textit{iarray} \quad \text{First element of integer array to be searched}

\textit{inc} \quad \text{Skip distance between elements of the searched array}

\textit{itarget} \quad \text{Integer value searched for in array. If target is not found, then the returned value is } n+1.

\textsc{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.

SR-0014  

4-64
<table>
<thead>
<tr>
<th>Name (parameter list)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHENEQ(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are equal to the real target</td>
</tr>
<tr>
<td>WHENNE(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are not equal to the real target</td>
</tr>
<tr>
<td>WHENFLT(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are less than the real target</td>
</tr>
<tr>
<td>WHENFLE(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are less than or equal to the real target</td>
</tr>
<tr>
<td>WHENFGT(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are greater than the real target</td>
</tr>
<tr>
<td>WHENFGE(n,array,inc,target,index,nval)</td>
<td>Returns all locations in a real array that are greater than or equal to the real target</td>
</tr>
<tr>
<td>WHENEQ(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are equal to the integer target</td>
</tr>
<tr>
<td>WHENNE(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are equal to the integer target</td>
</tr>
<tr>
<td>WHENILT(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are less than the integer target</td>
</tr>
<tr>
<td>WHENILE(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are less than or equal to the integer target</td>
</tr>
<tr>
<td>WHENIGT(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are greater than the integer target</td>
</tr>
<tr>
<td>WHENIGE(n,iarray,inc,itarget,index,nval)</td>
<td>Returns all locations in an integer array that are greater than or equal to the integer target</td>
</tr>
</tbody>
</table>
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, in~, 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:

\[
\text{CALL WHENFLE}(n,\text{array},\text{inc},\text{target},\text{index},\text{nval})
\]

\begin{itemize}
  \item \textit{n} \quad \text{Number of elements to be searched}
  \item \textit{array} \quad \text{First element of real array to be searched}
  \item \textit{inc} \quad \text{Skip distance between elements of the searched array}
  \item \textit{target} \quad \text{Real value searched for in array}
  \item \textit{index} \quad \text{Integer array containing the index of the found target in the array}
  \item \textit{nval} \quad \text{Number of values put in the index array}
\end{itemize}

\textsc{WHENFGT} is used when the real array element is greater than the real target.

Call from FORTRAN:

\[
\text{CALL WHENFGT}(n,\text{array},\text{inc},\text{target},\text{index},\text{nval})
\]

\begin{itemize}
  \item \textit{n} \quad \text{Number of elements to be searched}
  \item \textit{array} \quad \text{First element of real array to be searched}
  \item \textit{inc} \quad \text{Skip distance between elements of the searched array}
  \item \textit{target} \quad \text{Real value searched for in array}
  \item \textit{index} \quad \text{Integer array containing the index of the found target in the array}
  \item \textit{nval} \quad \text{Number of values put in the index array}
\end{itemize}

\textsc{WHENFGE} is used when the real array element is greater than or equal to the real target.

Call from FORTRAN:

\[
\text{CALL WHENFGE}(n,\text{array},\text{inc},\text{target},\text{index},\text{nval})
\]
\begin{verbatim}
\textbf{n} \quad \text{Number of elements to be searched}
\textbf{array} \quad \text{First element of real array to be searched}
\textbf{inc} \quad \text{Skip distance between elements of the searched array}
\textbf{target} \quad \text{Real value searched for in array}
\textbf{index} \quad \text{Integer array containing the index of the found target in the array}
\textbf{nval} \quad \text{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:

\begin{verbatim}
CALL WHENILT(n,iarray,inc,itarget,index,nval)
\end{verbatim}

\textbf{n} \quad \text{Number of elements to be searched}
\textbf{iarray} \quad \text{First element of integer array to be searched}
\textbf{inc} \quad \text{Skip distance between elements of the searched array}
\textbf{itarget} \quad \text{Integer value searched for in array}
\textbf{index} \quad \text{Integer array containing the index of the found target in the array}
\textbf{nval} \quad \text{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:

\begin{verbatim}
CALL WHENILE(n,iarray,inc,itarget,index,nval)
\end{verbatim}

\textbf{n} \quad \text{Number of elements to be searched}
\textbf{iarray} \quad \text{First element of integer array to be searched}
\textbf{inc} \quad \text{Skip distance between elements of the searched array}
\end{verbatim}

SR-0014 4-69
**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:

```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:

```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, \( index = n + 1 \)
**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:**

```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 \( n < 1 \), \( \text{index} = 0 \)
2. If no equal array elements, \( \text{index} = n+1 \)

- **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, \( \text{index} = \text{iwhere} \).)

**Exceptional case:** if \( n \) is less than 1, \( \text{iwhere} = 0 \)
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,iradix)
```

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*irealth)-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 two's 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 iradsis=1, or K=65537 if iradsis=2

data Array dimensioned irealth by N containing the N records of length irealth 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
irealth Length of each record as a number of 64-bit words. Default is 1. irealth is used as a skip for vector loads and stores. Therefore, irealth should be chosen to avoid bank conflicts.

ikeylth Length of each key as a number of 8-bit bytes. Default is eight bytes (1 word).

iradsize Radix of the sort. iradsize is the number of bytes processed per pass over the records. Default is 1. See section on large radix sorting for iradsize=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 irealth. Therefore, irealth 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 ikeylth/iradsize. Using ORDERS with iradsize=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 l-S Computer System.

Table 4-8. Sort times in seconds for ORDERS

<table>
<thead>
<tr>
<th>Length of key in bytes</th>
<th>N</th>
<th>7</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>.00057</td>
<td>.00065</td>
<td>.00065</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>.00059</td>
<td>.00067</td>
<td>.00067</td>
</tr>
<tr>
<td>One byte per pass</td>
<td></td>
<td>.00060</td>
<td>.00069</td>
<td>.00070</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>.00080</td>
<td>.00092</td>
<td>.00095</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>.00105</td>
<td>.00120</td>
<td>.00125</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>.00299</td>
<td>.00342</td>
<td>.00370</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>.00543</td>
<td>.00616</td>
<td>.00678</td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td>.02480</td>
<td>.02824</td>
<td>.03118</td>
</tr>
<tr>
<td>10,000</td>
<td></td>
<td>.04906</td>
<td>.05585</td>
<td>.06173</td>
</tr>
<tr>
<td>Two bytes per pass</td>
<td></td>
<td>.01515</td>
<td>.01520</td>
<td>.01513</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>.01510</td>
<td>.01511</td>
<td>.01519</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>.01507</td>
<td>.01511</td>
<td>.01512</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>.01522</td>
<td>.01522</td>
<td>.01525</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>.01540</td>
<td>.01542</td>
<td>.01547</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>.01651</td>
<td>.01642</td>
<td>.01679</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>.01781</td>
<td>.01786</td>
<td>.01838</td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td>.02486</td>
<td>.02861</td>
<td>.03156</td>
</tr>
<tr>
<td>5,000</td>
<td></td>
<td>.04216</td>
<td>.04216</td>
<td>.04808</td>
</tr>
</tbody>
</table>
Table 4-9. Sort times in seconds with ASCII key

<table>
<thead>
<tr>
<th>N</th>
<th>Two passes 8-byte Key</th>
<th>One pass 16-byte Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.00131</td>
<td>0.00065</td>
</tr>
<tr>
<td>5</td>
<td>0.00135</td>
<td>0.00068</td>
</tr>
<tr>
<td>10</td>
<td>0.00140</td>
<td>0.00072</td>
</tr>
<tr>
<td>50</td>
<td>0.00191</td>
<td>0.00104</td>
</tr>
<tr>
<td>100</td>
<td>0.00258</td>
<td>0.00145</td>
</tr>
<tr>
<td>500</td>
<td>0.00796</td>
<td>0.00485</td>
</tr>
<tr>
<td>1,000</td>
<td>0.01470</td>
<td>0.00907</td>
</tr>
<tr>
<td>5,000</td>
<td>0.06874</td>
<td>0.04307</td>
</tr>
<tr>
<td>10,000</td>
<td>0.13602</td>
<td>0.08531</td>
</tr>
</tbody>
</table>

Example 1:

PROGRAM SORT1

C Sort on a 2-word (16-byte) key that is at the beginning of a 5-word (inclusive) record

DIMENSION DATA(5,N)
DIMENSION INDEX(N)
DIMENSION WORK(65537)
N=10000

CALL ORDERS(MODE,WORK,DATA,INDEX,N,5,16,2)

Print out the keys in increasing alphabetic order

DO 100 I=1,N
   WRITE(6,200) DATA(1,INDEX(I)), DATA(2, INDEX(I))
200 FORMAT(1X,2A8)
100 CONTINUE

Example 2:

PROGRAM SORT2
This program uses two calls to ORDERS to completely sort an array of double-precision numbers. The sign bit of the first word is used to change the second word into a text key that preserves the ordering. A sort is done on these six bytes of the second word. (The changes made to the second word are reversed after the call.) Last, a sort is done on the first word as a real key using the initial ordering from the previous call.

DOUBLE PRECISION DATA(IOO)
INTEGER IATA(200)
EQUIVALENCE(IATA, DATA)
INTEGER INDEX(IOO), WORK(257)
N=12
DO 5 I=1, N
   DATA(I)=(-1, DO)**IO.DO**(-20)*DBLE(RANF())
   CONTINUE

First the second word key is changed

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

Sort on second word

CALL ORDERS(0, WORK, IATA(2), INDEX, N, 2, 6, 1)

Restore second word to original form

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

Sort on the first word using the initial ordering

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
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

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

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])
```

- arg1: Address of unit name or number, or internal file
- arg2: Address of FORMAT specification
- arg3: Address of error exit address, ERR=
- arg4: Address of end exit address, END=
- arg5: Address of IOSTAT parameter
- arg6: Address of record number (direct access)

$RUI initializes FORTRAN unformatted read.

Call from CAL:

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

- arg1: Address of unit name or number
- arg2: Unused
- arg3: Address of error exit address, ERR=
- arg4: Address of end exit address, END=
- arg5: Address of IOSTAT parameter
- arg6: Address of record number (direct access)

$DFI initializes FORTRAN formatted decode.

Call from CAL:

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

SR-0014 5-3
$RLI$ initializes list-directed reads.

Call from CAL:

\[
\text{CALL } \$RLI, (\text{arg}_1[] , \text{arg}_3[] , \text{arg}_4[] , \text{arg}_5[])
\]

\begin{itemize}
  \item \text{arg}_1 \quad \text{Address of unit name or number}
  \item \text{arg}_2 \quad \text{Unused}
  \item \text{arg}_3 \quad \text{Address of error exit for ERR=}
  \item \text{arg}_4 \quad \text{Address of exit address for END=}
  \item \text{arg}_5 \quad \text{Address of IOSTAT parameter}
\end{itemize}

Output initialization routines

$WFI$ initializes \texttt{FORTRAN} formatted write.

Call from CAL:

\[
\text{CALL } \$WFI, (\text{arg}_1[] , \text{arg}_2[] , \text{arg}_3[] , \text{arg}_5[] , \text{arg}_6[])
\]

\begin{itemize}
  \item \text{arg}_1 \quad \text{Address of unit name or number, or internal file}
  \item \text{arg}_2 \quad \text{Address of FORMAT specification}
  \item \text{arg}_3 \quad \text{Address of error exit address}
  \item \text{arg}_4 \quad \text{Unused}
  \item \text{arg}_5 \quad \text{Address of IOSTAT parameter}
  \item \text{arg}_6 \quad \text{Address of record number (direct access)}
\end{itemize}

$WUI$ initializes \texttt{FORTRAN} unformatted write.

Call from CAL:

\[
\text{CALL } \$WUI, (\text{arg}_1[] , \text{arg}_3[] , \text{arg}_5[] , \text{arg}_6[])
\]

\begin{itemize}
  \item \text{arg}_1 \quad \text{Address of unit name or number}
  \item \text{arg}_2 \quad \text{Unused}
  \item \text{arg}_3 \quad \text{Address of error exit address}
  \item \text{arg}_4 \quad \text{Unused}
  \item \text{arg}_5 \quad \text{Address of IOSTAT parameter}
  \item \text{arg}_6 \quad \text{Address of record number (direct access)}
\end{itemize}
$EFI$ 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:

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Type of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$xnam+0$</td>
<td>Typeless (Boolean) or no I/O list or type checking present</td>
</tr>
<tr>
<td>Entry point</td>
<td>Type of data</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>xnam+3</td>
<td>Integer</td>
</tr>
<tr>
<td>xnam+6</td>
<td>Real</td>
</tr>
<tr>
<td>xnam+9</td>
<td>Double-precision</td>
</tr>
<tr>
<td>xnam+12</td>
<td>Complex</td>
</tr>
<tr>
<td>xnam+15</td>
<td>Logical</td>
</tr>
<tr>
<td>xnam+18</td>
<td>Character</td>
</tr>
<tr>
<td>xnam+21</td>
<td>Short integer</td>
</tr>
</tbody>
</table>

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,(arg_1,arg_2,arg_3)
CALL $RUA+offset,(arg_1,arg_2,arg_3)
CALL $DFA+offset,(arg_1,arg_2,arg_3)
```

- $arg_1$: 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,(arg_1,arg_2,arg_3)
```

- $arg_1$: 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. 
%RU% reads FORTRAN unformatted, vectorized data.

Call from CAL:

```
CALL $RFV%+offset
CALL $RU%+offset
```

Entry:
- (VL) Vector length

Exit:
- (VL) (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:
- arg1 Address of unit name or number
- arg2 Address of mode specifier (mode<0 indicates partial record transfer; mode>0 indicates full record transfer).
- arg3 First word address of destination
- arg4 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])
```
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GN</td>
<td>0-63</td>
<td>8-bit ASCII name, left-justified, zero-filled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>0</td>
<td>0-63</td>
<td>8-bit ASCII character name, left-justified, zero-filled</td>
</tr>
<tr>
<td>Unused</td>
<td>1</td>
<td>0-10</td>
<td>Zero-filled</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>11-13</td>
<td>Mode. Number of dimensions in the array (1-7)</td>
</tr>
<tr>
<td>Unused</td>
<td>1</td>
<td>14-19</td>
<td>Zero-filled</td>
</tr>
<tr>
<td>TL</td>
<td>1</td>
<td>20-31</td>
<td>Internal CFT type and length description:</td>
</tr>
<tr>
<td>Field</td>
<td>Word</td>
<td>Bits</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Unused</td>
<td>1</td>
<td>32-33</td>
<td>Zero-filled</td>
</tr>
</tbody>
</table>
| 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 |

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

**Figure 5-3. Array entry**
<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>1</td>
<td>32-33</td>
<td>Zero-filled</td>
</tr>
</tbody>
</table>
| 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)
```

arg1  First word address destination  
arg2  Address of word count  
arg3  Address of increment between destination addresses

SR-0014  5-11  I
$WLA$ writes list-directed data by address.

Call from CAL:

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

arg1  
First word address of input

arg2  
First word address of item count

arg3  
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:

```call
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:

```call
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:
- $arg1$: Address of unit name or number
- $arg2$: Address of mode specifier (mode<0 indicates partial record transfer; mode>0 indicates full record transfer).
- $arg3$: First word address of destination
- $arg4$: 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)
```

- $arg1$: Address of unit name or number
- $arg2$: Address of NAMELIST group entries
- $arg3$: 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 $WUF

$WUF finalizes FORTRAN unformatted write.

Call from CAL:

CALL $WUF

$WUF finalizes list-directed write.

Call from CAL:

CALL $WUF

$WUF finalizes FORTRAN formatted encode.

Call from CAL:

CALL $WUF

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)
```

arg1 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
```

Output buffer management routines

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.
$SCDCI 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.

CDl20I% 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.

Output data management format routines

%II160 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.

%ICHR0 translates 8-bit IBM EBCDIC character to 8-bit ASCII character on output.
%IC640 translates 64-bit IBM complex floating-point to 128-bit Cray complex floating-point on output.

CI600% translates 64-bit Cray integer to CDC 60-bit integer on output.

CF600% translates 64-bit Cray floating-point to CDC 60-bit floating-point on output.

CD1200% translates 128-bit Cray double-precision floating-point to CDC 120-bit double-precision on output.

CC1200% 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.

CL600% 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.
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.

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:**

```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
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isb</code></td>
<td>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 <code>dest</code>.</td>
</tr>
<tr>
<td><code>num</code></td>
<td>Number of Cray floating-point numbers to convert. Type integer variable, expression, or constant.</td>
</tr>
<tr>
<td><code>ier</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>inc</code></td>
<td>Memory increment for fetching the number to be converted. Optional parameter of type integer variable, expression, or constant. Default value is 1.</td>
</tr>
</tbody>
</table>

**CRAZY 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(fp, dest, isb, num, ier[, inc])
```

- `fp` 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:**

```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:**

```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 characters 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)
```
**PACK 32-BIT WORDS ROUTINE**

The P32 subroutine packs 32-bit words into Cray 64-bit words.

Call from FORTRAN:

```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:

```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-precision 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 boundaries.

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(ollist)
```

ollist  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(ist)
```

`ist` 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**

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Data Type</th>
<th>Meaning</th>
<th>Input (I) or Return value (RV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT=(u)</td>
<td>Integer</td>
<td>External unit specifier</td>
<td>(I) Unit number</td>
</tr>
<tr>
<td>IOSTAT=io8</td>
<td>Integer variable or array element</td>
<td>Error status specifier</td>
<td>(I) if no error condition exists. If error condition exists, error message number that corresponds to error (see CRAY-OE Message Manual, publication SR-0039 for error message descriptions.)</td>
</tr>
<tr>
<td>ERR=(a)</td>
<td>Statement label</td>
<td>Statement label where control is transferred if error condition exists</td>
<td>(I) FORTRAN statement label</td>
</tr>
<tr>
<td>FILE=(f\in)</td>
<td>Character expression</td>
<td>File specifier</td>
<td>(I) Name of dataset to be connected</td>
</tr>
<tr>
<td>STATUS=std</td>
<td>Character expression</td>
<td>Disposition specifier (Default, 'UNKNOWN')</td>
<td>(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'.</td>
</tr>
<tr>
<td>ACCESS=acc (\d)</td>
<td>Character expression</td>
<td>Access specifier (Default, 'SEQUENTIAL')</td>
<td>(I) 'SEQUENTIAL' is access method; 'DIRECT' is access method.</td>
</tr>
<tr>
<td>FORM=f'ed</td>
<td>Character expression</td>
<td>Form specifier (Default, 'UNFORMATTED' if access is direct; 'FORMATTED' if access is sequential.)</td>
<td>(I) 'FORMATTED', formatted I/O; 'UNFORMATTED', unformatted I/O.</td>
</tr>
<tr>
<td>RBLCL=pl</td>
<td>Positive integer expression</td>
<td>Record length for direct access method (omitted for sequential access)</td>
<td>(I) For formatted I/O, number of characters per record; For unformatted I/O, 8 times the number of words.</td>
</tr>
<tr>
<td>BLANK=blnk</td>
<td>Character expression</td>
<td>Blank specifier (Default, 'NULL')</td>
<td>(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.</td>
</tr>
</tbody>
</table>

\(\d\) UNIT= does not need to be included in the unit specification if \(u\) is the first item in olist.

\(\d\)\(\d\) CPT allows formatted and unformatted records in the same dataset (non-ANSI).
Table 5-3. CLOSE specifiers and their meanings

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Data Type</th>
<th>Meaning</th>
<th>Input (I) or Return value (RV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT=nn</td>
<td>Integer</td>
<td>External unit specifier</td>
<td>(I) Unit number</td>
</tr>
<tr>
<td>ERR=ς</td>
<td>Statement label</td>
<td>Statement label where control is transferred if error condition exists</td>
<td>(I) FORTRAN statement label</td>
</tr>
<tr>
<td>STATUS=ςς</td>
<td>Character expression</td>
<td>Disposition specifier (Default, 'KEEP' if OPEN status is 'OLD', 'NEW', or 'UNKNOWN'. Default, 'DELETE' if OPEN status is 'SCRATCH' or dataset is memory resident.)</td>
<td>(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.</td>
</tr>
<tr>
<td>IOSTAT=ιοι</td>
<td>Integer variable or array element</td>
<td>Error status specifier</td>
<td>(RV) 0 if no error condition exists, 1 if error condition exists, error message number that corresponds to error (see CRAY-OS Message Manual, publication SR-0019 for error message descriptions).</td>
</tr>
</tbody>
</table>

† UNIT= does not need to be included in the unit specification if u is the first item in ellist.

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:

```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

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Data Type</th>
<th>Meaning</th>
<th>Input (I) or Return value (RV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOSTAT=io8</td>
<td>Integftr variable or array element</td>
<td>Error status specifier</td>
<td>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.)</td>
</tr>
<tr>
<td>ERR=se</td>
<td>Statement label</td>
<td>Statement label where control is transferred if error condition exists</td>
<td>I FORTRAN statement label</td>
</tr>
<tr>
<td>EXIST=se</td>
<td>Logical variable or array element</td>
<td>Existence specifier</td>
<td>RV .TRUE. if unit or file exists; else, .FALSE.</td>
</tr>
<tr>
<td>OPENED=od</td>
<td>Logical variable or array element</td>
<td>Connection specifier</td>
<td>RV .TRUE. if unit and dataset are connected; else, .FALSE.</td>
</tr>
<tr>
<td>NUMBER=num</td>
<td>Integer variable or array element</td>
<td>External unit specifier</td>
<td>RV Unit currently connected; if no unit, num is undefined</td>
</tr>
<tr>
<td>NAMED=nm</td>
<td>Logical variable or array element</td>
<td>Unit name specifier</td>
<td>RV .TRUE. if unit has a name; else, .FALSE.</td>
</tr>
<tr>
<td>RECL=rc1</td>
<td>Integer variable or array element</td>
<td>Record length of unit or file connected for direct access</td>
<td>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, rc1 is undefined.</td>
</tr>
<tr>
<td>NEXTRBC=nr</td>
<td>Integer variable or array element</td>
<td>Next record</td>
<td>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.</td>
</tr>
<tr>
<td>NAME=fn</td>
<td>Character variable or array element</td>
<td>File name</td>
<td>RV File name if file has a name; else, fn is undefined.</td>
</tr>
<tr>
<td>ACCESS=ac0</td>
<td>Character variable or array element</td>
<td>Access specifier</td>
<td>RV 'SEQUENTIAL' is access method; 'DIRECT' is access method.</td>
</tr>
<tr>
<td>SEQUENTIAL=seq</td>
<td>Character variable or array element</td>
<td>Sequential as possible access method</td>
<td>RV 'YES' if sequential is allowed; 'NO' if sequential is not allowed; 'UNKNOWN' if unable to determine.</td>
</tr>
<tr>
<td>DIRECT=dir</td>
<td>Character variable or array element</td>
<td>Direct as possible access method</td>
<td>RV 'YES' if direct is allowed; 'NO' if direct is not allowed; 'UNKNOWN' if unable to determine.</td>
</tr>
<tr>
<td>FORM=fm</td>
<td>Character variable or array element</td>
<td>Format specifier</td>
<td>RV 'FORMATTED' if file is connected for formatted I/O; 'UNFORMATTED' if file is connected for unformatted I/O.</td>
</tr>
<tr>
<td>FORMATTED=fm</td>
<td>Character variable or array element</td>
<td>Formatted as a possible allowed form</td>
<td>RV 'YES' if formatted is allowed; 'NO' if formatted is not allowed; 'UNKNOWN' if unable to determine.</td>
</tr>
<tr>
<td>UNFORMATTED=utf</td>
<td>Character variable or array element</td>
<td>Unformatted as a possible allowed form</td>
<td>RV 'YES' if unformatted is allowed; 'NO' if unformatted is not allowed; 'UNKNOWN' if unable to determine.</td>
</tr>
<tr>
<td>BLANK=blnk</td>
<td>Character variable or array element</td>
<td>Blank control specifier</td>
<td>RV 'NULL' if null blank control is in effect; 'ZERO' if zero blank control is in effect. Blank control applies only to formatted records.</td>
</tr>
</tbody>
</table>

* 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.

```fortran
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:**

```fortran
CALL SKIPD(dn[,istat])
```
**$GPOS** and **GETPOS**

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:

\[
\text{CALLV \$GPOS}
\]

**Entry:**
- (AI) 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:**
- (AI) 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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End-of-record (EOR) flag</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>0</td>
<td>Dataset is either at BOD or in the middle of a record.</td>
</tr>
<tr>
<td>1</td>
<td>Unused</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>End-of-data flag. One indicates the dataset is currently at EOD.</td>
</tr>
<tr>
<td>31-63</td>
<td>Word address. This word address is the current physical word address within the dataset, including RCWs but not including block control words (BCW).</td>
</tr>
</tbody>
</table>
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, CRRI 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:
(Al) Absolute DSP address or negative DSP offset relative to the
DSP base (JCDSP)

Exit:
(Al) Absolute DSP address
(Sl) 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

(S2) Dataset position information for finding the beginning of
    the current block (Cray physical record). For disk
data sets, 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
data sets, 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
\[ \neq 0 \] 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:

\[
\begin{align*}
\text{CALLV } & \text{SPOS} \\
\text{CALLV } & \text{ASPOS}
\end{align*}
\]

Entry:

(AI) Address of Dataset Parameter Table (DSP) or negative DSP offset relative to DSP base (JCDS), that is, contents of second word of Open Dataset Name Table (ODN).

(SI) Dataset position

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-30</td>
<td>Unused</td>
</tr>
<tr>
<td>31-63</td>
<td>Word address. The desired physical word address within the dataset aligned on a record boundary. Record control words are included.</td>
</tr>
</tbody>
</table>
Exit:

<table>
<thead>
<tr>
<th>(A1)</th>
<th>DSP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S1)</td>
<td>Dataset position</td>
</tr>
<tr>
<td>(S6)</td>
<td>RCW after which the dataset is positioned; (S6)=0 if at BOD.</td>
</tr>
</tbody>
</table>

$FSPOS$ 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:

<table>
<thead>
<tr>
<th>(A1)</th>
<th>Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S1)</td>
<td>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.</td>
</tr>
<tr>
<td>(S2)</td>
<td>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.</td>
</tr>
</tbody>
</table>
(S3) Volume number request sign. This register must be set to '+1', '-1', or '1'. 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 '+1'. A block request (register S2) must not be specified with plus or minus signs.

For backward volume positioning, set register S3 to '-1'. A block request (register S2) must not be specified with plus or minus signs.

For absolute volume positioning, set register S3 to '1'.

(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 'VOL1'.

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) 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):

```
BACKSPACE(u, fin, alist)
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>u</strong></td>
<td>Unit identifier</td>
<td></td>
</tr>
<tr>
<td><strong>fin</strong></td>
<td>File identifier whose value specifies the name of the file</td>
<td></td>
</tr>
<tr>
<td><strong>alist</strong></td>
<td>The following set of identifiers:</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{[UNIT]} &= u \\
\text{IOSTAT} &= ios \\
\text{ERR} &= s
\end{align*}
\]

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:

(AI) 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:

(AI) Address of DSP
(S6) Contains record control word (RCW) after which dataset is positioned (equals 0 if at BOD)

Call from FORTRAN (also clears UEQF 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 $REW'D

Entry:

(AI) 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:

(AI) Address of DSP
Call from FORTRAN:

\[
\text{REWIND}(\{ \text{unit} \})
\]

\begin{itemize}
\item \text{unit} \quad \text{Unit identifier. If the dataset is $\text{IN}$, the file is positioned after the first EOF. Otherwise, it is positioned at BOD.}
\item \text{fin} \quad \text{File identifier whose value specifies the name of a file}
\item \text{alist} \quad \text{The following set of specifiers must contain a single external unit specifier and can contain, at most, one of each of the other specifiers.}
\end{itemize}

\begin{align*}
&[\text{UNIT}]=\text{u} \\
&\text{IOSTAT}=\text{ios} \\
&\text{ERR}=\text{s}
\end{align*}

See the UNIT, IOSTAT, and ERR specifiers described for the OPEN or CLOSE statement, table 5-2, or table 5-3.

**Position dataset**

$\text{PBN}$ and $\text{APBN}$ position a dataset to a specific block, update $\text{DPOUT}$ of $\text{DSP}$, clear $\text{DPRCW}$ of $\text{DSP}$, and update $\text{DPIN}$ of $\text{DSP}$, if a read is from disk. $\text{PBN}$ synchronously positions the dataset; $\text{APBN}$ asynchronously positions the dataset.

Call from CAL:

\begin{verbatim}
CALL $\text{PBN}$
CALL $\text{APBN}$
\end{verbatim}

**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 responsible for checking on completion of the read and any outstanding errors. The only valid exit value is the DSP address found in Al.

Exit:
- (Al) Address of DSP
- (A4) New block number
- (A5) DPPRST 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:

\[
\text{CALLV $PRCW}
\]

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:
(Al) Absolute DSP address or negative DSP offset relative to the DSP base (JCDSP)
(Sl) Processing direction:
    =0 Input dataset
    =1 Output dataset

Exit:
(Al) Absolute DSP address
(Sl) 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:
(Al) 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:
(Al) 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:

```
(Sl) 0 if UEOP was not set
1 if UEOP 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)
```

SR-0014 5-57 I
\textbf{iexit} \hspace{1em} -1 \hspace{1em} \text{EOD on last operation} \\
\hspace{1em} 0 \hspace{1em} \text{Neither EOD nor EOF on last operation} \\
\hspace{1em} +1 \hspace{1em} \text{EOF on last operation}

\textit{dn} \hspace{1em} \text{Dataset name or unit number}

\textbf{EOF} returns a real value EOF status and clears UEOF flag in DSP.

Call from FORTRAN:
\begin{verbatim}
  iexit=EOF(dn)
\end{verbatim}

\textit{rexit} \hspace{1em} -1.0 \hspace{1em} \text{EOD on last operation} \\
\hspace{1em} 0.0 \hspace{1em} \text{Neither EOD nor EOF on last operation} \\
\hspace{1em} +1.0 \hspace{1em} \text{EOF on last operation}

\textit{dn} \hspace{1em} \text{Dataset name or unit number}

\textbf{IOSTAT} returns end-of-file status.

Call from FORTRAN:
\begin{verbatim}
  iexit=IOSTAT(dn)
\end{verbatim}

\textit{iexit} \hspace{1em} 0 \hspace{1em} \text{No error} \\
\hspace{1em} 1 \hspace{1em} \text{UEOF cleared (EOF)} \\
\hspace{1em} 2 \hspace{1em} \text{UEOD cleared (EOD)}

\textit{dn} \hspace{1em} \text{Dataset name or unit number}

\textbf{$EOATEST$} 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:
\begin{verbatim}
  CALLV $EOATEST
\end{verbatim}

Entry:
\begin{verbatim}
  (A1) DSP address
\end{verbatim}
Exit (if job not aborted):
\begin{verbatim}
  (A1) DSP address
\end{verbatim}
$UEOFSET sets the uncleared End-of-file flag in the DSP.

Call from CAL:

```
CALLV $UEOFSET
```

Entry:

(Al) DSP address

Exit:

(Al) DSP address

$UEOPTCL clears the uncleared End-of-file flag in the DSP and indicates whether it had been set or not.

Call from CAL:

```
CALLV $UEOPTCL
```

Entry:

(Al) DSP address

Exit:

(Al) DSP address

(Sl) 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:

(Al) DSP address

Exit (if job not aborted):

(Al) 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).
- `mode = 0` Skip past group.
- `mode < 0` Abort job or go to ERR=address.

RNCTYPE determines the action if a type mismatch occurs across the equal sign on an input card.

Call from FORTRAN:

```
CALL RNCTYPE(mode)
```

- `mode = 0` Abort job or go to ERR=address.
- `mode ≠ 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).
- `unit ≠ 0` Error messages and echoed input go to specified unit. All input is echoed.

The following four routines (RNFLAG, RNLDEL, 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:

\[
\begin{array}{cccccccccccccc}
A & B & C & D & E & F & G & H & \hline
I & J & K & L & M & N & O & P & \hline
Q & R & S & T & U & V & W & X & \hline
Y & Z & \Delta & \Delta & \Delta & \Delta & \Delta & \Delta & \hline
\end{array}
\]

\[\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \hline
8 & 9 & \Delta & \Delta & \Delta & \Delta & \Delta & \Delta & \hline
\end{array}\]

$RWDP\rightarrow (A2)_{\text{RCW}}\rightarrow (A3)=2\Delta\rightarrow (A3)=2\Delta\times 2

Data in I/O buffer

READ reads words, full record mode. READP reads words, partial record mode.

Call from FORTRAN:

\[
\begin{align*}
\text{CALL READ}(dn, word, count, status[, ubc]) \\
\text{CALL READP}(dn, word, count, status[, ubc])
\end{align*}
\]

\begin{itemize}
  \item \textit{dn} \hspace{1cm} \text{Dataset name or unit number}
  \item \textit{word} \hspace{1cm} \text{Word-receiving data area}
  \item \textit{count} \hspace{1cm} \text{On entry: number of words requested} \hspace{1cm} \text{On exit: number of words actually transferred}
  \item \textit{status} \hspace{1cm} \text{On exit:}
    \begin{itemize}
      \item -1 \hspace{1cm} \text{Words remain in record.}
      \item 0 \hspace{1cm} \text{EOR}
      \item 1 \hspace{1cm} \text{Null record}
      \item 2 \hspace{1cm} \text{EOF}
      \item 3 \hspace{1cm} \text{EOD}
      \item 4 \hspace{1cm} \text{Unrecovered data error encountered}
    \end{itemize}
  \item \textit{ubc} \hspace{1cm} \text{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.}
\end{itemize}

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:

Example of $RWDP:

\[ (A2) \quad \text{T} \quad \text{H} \quad \text{I} \quad \text{S} \]
\[ (A3) = 16 \]

Data in I/O buffer

User data area

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 $WWDP$:

Example of $WWDP$:

WRITE writes words, full record mode.

Call from FORTRAN:

\[
\text{CALL WRITE}(dn, \text{word}, \text{count}[, \text{ubc}])
\]

\begin{itemize}
  \item \textit{dn} \quad \text{Dataset name or unit number}
  \item \textit{word} \quad \text{Data area containing words}
  \item \textit{count} \quad \text{Word count. A value of 0 causes an end-of-record record control word to be written.}
  \item \textit{ubc} \quad \text{Optional unused bit count. Number of unused bits contained in the last word of the record.}
\end{itemize}

WRITEP writes words, partial record mode.

Call from FORTRAN:

\[
\text{CALL WRITEP}(dn, \text{word}, \text{count}[, \text{ubc}])
\]

\begin{itemize}
  \item \textit{dn} \quad \text{Dataset name or unit number}
  \item \textit{word} \quad \text{Data area containing words}
  \item \textit{count} \quad \text{Word count}
  \item \textit{ubc} \quad \text{Optional unused bit count. Number of unused bits contained in the last word of the record.}
\end{itemize}
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 $WCHP$:

```
(A2) __ __ __ __ __ __ __ __ __ __ O __ __ __ __ __ __ __ __ __ __ U
                            __ __ __ __ __ __ __ __ __ __ T
                            __ __ __ __ __ __ __ __ __ __ P
                            __ __ __ __ __ __ __ __ __ __ U
                            __ __ __ __ __ __ __ __ __ __ T
                            __ __ __ __ __ __ __ __ __ __ Δ
                            __ __ __ __ __ __ __ __ __ __ Δ
                            __ __ __ __ __ __ __ __ __ __ D
                            __ __ __ __ __ __ __ __ __ __ A
                            __ __ __ __ __ __ __ __ __ __ T
                            __ __ __ __ __ __ __ __ __ __ A
```

User data area

SR-0014 5-73 I
WRITEC writes characters, full record mode. WRITECP writes characters, partial record mode.

Call from FORTRAN:

\[
\text{CALL WRITEC}(dn, \text{char}, \text{count}) \\
\text{CALL WRITECP}(dn, \text{char}, \text{count})
\]

\(dn\)  
Dataset name or unit number

\(\text{char}\)  
Data area containing characters

\(\text{count}\)  
Character count

Write IBM words

WRITIBM writes two IBM 32-bit floating-point words from each Cray 64-bit word.

Call from FORTRAN:

\[
\text{CALL WRITIBM}(dn, \text{fwa}, \text{value}, \text{increment})
\]

\(dn\)  
Dataset name or unit number

\(\text{fwa}\)  
FWA of user data area

\(\text{value}\)  
Number of values to be written

\(\text{increment}\)  
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:

\[
\text{CALLV$WLB}
\]
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, termond)
```

- **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.

- **termond**: 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,wrdbnt,termcnd,ubont)
```

- **dn**: Dataset name or unit number
- **uda**: User data area to receive the bad data; length must be 512 words.
- **wrdbnt**: 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
- **ubont**: On exit, address of unused bit count. Only defined if termination condition is 0, and wrdbnt 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**: Character address pointer
- **Exit**: Length of item in words
- **(Al)** Length of item in words
- **(VL)** Length of item adjusted to word boundary
- **(VI)** 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:
- (Al) 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:
- (Al) 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(foa,fw,dp,pf,mode,br,stat)
```

- **foa** 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.
High-order binary result followed by low-order binary result. This parameter must be two Cray 64-bit words long.

Errors and warnings returned from the conversion routines. See table 5-6 for descriptions.

Table 5-5. Conversion mode descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>SVDPART</td>
<td>Set if decimal places field present</td>
</tr>
<tr>
<td>50</td>
<td>SVCFT</td>
<td>Set if called from library; 0 if called from CFT.</td>
</tr>
<tr>
<td>51</td>
<td>SVPLS</td>
<td>Set if requested + sign output</td>
</tr>
<tr>
<td>52</td>
<td>SVEXPS</td>
<td>Set if exponent size defined</td>
</tr>
<tr>
<td>53</td>
<td>SVSEXPM</td>
<td>Sign of exponent</td>
</tr>
<tr>
<td>54</td>
<td>SVSMAN</td>
<td>Sign of fraction or number</td>
</tr>
<tr>
<td>55</td>
<td>SVDFLD</td>
<td>D conversion</td>
</tr>
<tr>
<td>56</td>
<td>SVGFLD</td>
<td>G conversion</td>
</tr>
<tr>
<td>57</td>
<td>SVEFLD</td>
<td>E conversion</td>
</tr>
<tr>
<td>58</td>
<td>SVFFLD</td>
<td>F conversion</td>
</tr>
<tr>
<td>59</td>
<td>SVIFLD</td>
<td>I conversion</td>
</tr>
<tr>
<td>60</td>
<td>SVZFLD</td>
<td>Hex conversion</td>
</tr>
<tr>
<td>61</td>
<td>SVOFLD</td>
<td>Octal conversion</td>
</tr>
<tr>
<td>62, 63</td>
<td>SVBZR,SVBNL</td>
<td>Blank conversion indicators:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 Blanks treated as delimiters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 Blanks ignored (as in FORTRAN source)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Blanks treated as zeros (as in FORTRAN run-time input)</td>
</tr>
</tbody>
</table>

NOCV% and NOCONV perform numeric output conversion.

Call from CAL:

```call
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
P factor
Mode in bits 49 to 63 (symbols defined in $IOLIB$). See table 5-5 for description of bits.

Exit:
Last output character address

Table 5-6. Conversion return conditions

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0'0077</td>
<td>Typeless</td>
</tr>
<tr>
<td>0'4027</td>
<td>24-bit integer</td>
</tr>
<tr>
<td>0'4077</td>
<td>64-bit integer</td>
</tr>
<tr>
<td>0'6077</td>
<td>64-bit real</td>
</tr>
<tr>
<td>0'6177</td>
<td>128-bit real</td>
</tr>
<tr>
<td>-1</td>
<td>Illegal character</td>
</tr>
<tr>
<td>-2</td>
<td>Overflow</td>
</tr>
<tr>
<td>-3</td>
<td>Exponent underflow</td>
</tr>
<tr>
<td>-4</td>
<td>Exponent overflow</td>
</tr>
<tr>
<td>-5</td>
<td>Null field</td>
</tr>
</tbody>
</table>

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

\[ >0 \quad \text{No error in conversion} \]
\[ <0 \quad \text{Error in conversion. (Current version of numeric output conversion has no error conditions to return.)} \]

$\$NICV$ performs numeric input conversion. This routine is obsolete.

Call from CAL:

\[ \text{R } \$NICV \]

Entry:
\begin{itemize}
  \item (A3) Field width (optional)
  \item (A4) Decimal places (optional)
  \item (A5) P factor (optional)
  \item (A6) First character address
  \item (S6) Mode in bits 49 to 63 (symbols defined in \$IOLIB$). See table 5-5 for bit descriptions.
\end{itemize}

Exit:
\begin{itemize}
  \item (A1) Next character
  \item (A2) Mode (in octal). See table 5-6 for descriptions.
  \item (A6) Last input character address+1
  \item (S1) High-order result
  \item (S2) Low-order result
\end{itemize}

$\$NOCV$ performs numeric output conversion. This routine is obsolete.

Call from CAL:

\[ \text{R } \$NOCV \]

Entry:
\begin{itemize}
  \item (A3) Field width
  \item (A4) Decimal places
  \item (A5) P factor
  \item (A7) First output character address
  \item (S1),(S2) Binary number to be output
  \item (S6) Conversion mode in bits 49 to 61 (symbols defined in \$IOLIB$). See table 5-5 for bit descriptions.
\end{itemize}

Exit:
\begin{itemize}
  \item (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:

```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 \(2^n_{rec}\) if \(it=1\) or \(n_{rec}\) if \(it=0\). \(n_{rec}\) 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

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Routines Affected</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>OPENDR, OPENMS, WRITDR, WRITMS, READDR, READMS, STINDR, STINDX, CLOSDR, CLOSMS, CHECKDR, CHECKMS, WAITDR, WAITMS, ASYNCDR, ASYNCSMS, SYNCDR, SYNCMS</td>
<td>The dataset name or unit number is illegal.</td>
</tr>
<tr>
<td>-2</td>
<td>OPENDR, OPENMS</td>
<td>The user-supplied index length is less than or equal to 0.</td>
</tr>
<tr>
<td>-3</td>
<td>OPENDR, OPENMS</td>
<td>The number of datasets has exceeded memory or size availability.</td>
</tr>
<tr>
<td>-4</td>
<td>OPENDR, OPENMS</td>
<td>The dataset index length read from the dataset is greater than the user-supplied index length (nonfatal message).</td>
</tr>
<tr>
<td>-5</td>
<td>OPENDR, OPENMS</td>
<td>The user-supplied index length is greater than the index length read from dataset (nonfatal message).</td>
</tr>
</tbody>
</table>
Table 5-7. Error codes for record-addressable, random access dataset I/O routines (continued)

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Routines Affected</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6</td>
<td>WRITMS</td>
<td>The user-supplied named index is illegal.</td>
</tr>
<tr>
<td></td>
<td>READMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FINDMS</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>WRITMS</td>
<td>The named record index array is full.</td>
</tr>
<tr>
<td></td>
<td>READMS</td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>WRITMS</td>
<td>The index number is greater than the maximum on the dataset.</td>
</tr>
<tr>
<td></td>
<td>READMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FINDMS</td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td>WRITMS</td>
<td>Rewrite record exceeds the original.</td>
</tr>
<tr>
<td></td>
<td>READMS</td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>READMS</td>
<td>The named record was not found in the index array.</td>
</tr>
<tr>
<td></td>
<td>FINDMS</td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>OPENMS</td>
<td>The index word address read from the dataset is less than or equal to 0.</td>
</tr>
<tr>
<td>-12</td>
<td>OPENMS</td>
<td>The index length read from the dataset is less than 0.</td>
</tr>
<tr>
<td>-13</td>
<td>OPENMS</td>
<td>The dataset has a checksum error.</td>
</tr>
<tr>
<td>-14</td>
<td>OPENMS</td>
<td>OPENMS already has opened the dataset.</td>
</tr>
<tr>
<td>-15</td>
<td>WRITMS</td>
<td>OPENMS/OPENDR was not called on this dataset.</td>
</tr>
<tr>
<td></td>
<td>WRITDR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>READMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLOSMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STINDR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STINDX</td>
<td></td>
</tr>
</tbody>
</table>
Table 5-7. Error codes for record-addressable, random access dataset I/O routines (continued)

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Routines Affected</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-15</td>
<td>FINDMS CHECKMS CHECKDR WAITMS WAITDR ASYNCEMS ASYNCDR SYNCYMS SYNCDR WAITDR WAITMS</td>
<td>OPENMS/OPENDR was not called on this dataset.</td>
</tr>
<tr>
<td>-16</td>
<td>STINDR STINDX</td>
<td>A STINDX/STINDR call cannot change the index type.</td>
</tr>
<tr>
<td>-17</td>
<td>WRITMS WRITDR READMS FINDMS</td>
<td>The index entry is less than or equal to 0 in the user's index array.</td>
</tr>
<tr>
<td>-18</td>
<td>WRITMS WRITDR READMS FINDMS</td>
<td>The user-supplied word count is less than or equal to 0.</td>
</tr>
<tr>
<td>-19</td>
<td>WRITMS WRITDR READMS FINDMS</td>
<td>The user-supplied index number is less than or equal to 0.</td>
</tr>
<tr>
<td>-20</td>
<td>OPENMS OPENDR</td>
<td>Dataset created by WRITDR/WRITMS</td>
</tr>
</tbody>
</table>
Writtms/WrittDr writes data from user memory to a record in a random access dataset on disk and updates the current index.

Call from FORTRAN:

```
CALL WRITTMS(dn, ubuff, n, irec, rrflag, s[, ierr])
CALL WRITTDR(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. WrittDR 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 \( ierr \).
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.

Type INTEGER variable, expression, or constant. A sub-index flag.

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
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.

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 \( ierr \geq 0 \), 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

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL ACCESSES=</td>
<td>Number of accesses</td>
</tr>
<tr>
<td>READS =</td>
<td>Number of reads</td>
</tr>
<tr>
<td>WRITES =</td>
<td>Number of writes</td>
</tr>
</tbody>
</table>
Table 5-8. CLOSMS statistics (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENTIAL READS =</td>
<td>Number of sequential reads</td>
</tr>
<tr>
<td>SEQUENTIAL WRITES =</td>
<td>Number of sequential writes</td>
</tr>
<tr>
<td>REWRITES IN PLACE =</td>
<td>Number of rewrites in place</td>
</tr>
<tr>
<td>WRITES TO EOI =</td>
<td>Number of writes to EOI</td>
</tr>
<tr>
<td>TOTAL WORDS MOVED =</td>
<td>Number of words moved</td>
</tr>
<tr>
<td>MINIMUM RECORD =</td>
<td>Minimum record size</td>
</tr>
<tr>
<td>MAXIMUM RECORD =</td>
<td>Maximum record size</td>
</tr>
<tr>
<td>TOTAL ACCESS TIME =</td>
<td>Total access time</td>
</tr>
<tr>
<td>AVERAGE ACCESS TIME =</td>
<td>Average access time</td>
</tr>
</tbody>
</table>

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:

```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/ASYNC Dr. 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 ASYNC Dr. (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/ASYNC Dr., `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/ASYNC Dr.:

- `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 SY NC DR (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).
isrp

Type integer variable. Error control and code. If isrp is supplied on the call to SYNCMS/SYNCDR, isrp returns any error codes to the user. If isrp>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:

- **isrp=0** No errors detected
- **<0** Error detected. isrp 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 one 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:

```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, S12, 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 (KEYS(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, S12, NAME)
25  WRITE(6,5120) (IARRAY(J), J=1,S12)
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:

1  PROGRAM INITIAL
2  INTEGER IARRAY(512)
3  INTEGER INDEX (512)
4  CALL OPENMS ('MASTER',INDEX,101,1)
5  CALL WRITMS ('MASTER',IARRAY,512,NAME,0,0)
6  CONTINUE
7  CALL CLOSMS ('MASTER')

Program UPDATE:

1  PROGRAM UPDATE
2  INTEGER INEWRC(512)
3  INTEGER INDX (512)
4  CALL OPENMS ('MASTER',INDEX,101,1)
5  READ (6,610) N
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
  SINDEX(I)=0
C CALL STINDEX TO CHANGE INDEX TO SINDEX.
CALL STINDEX (1,SINDEX,SLEN,0,ERR)
IF (ERR.NE.0) THEN
   PRINT*, ' Error on STINDEX, 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
   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
20 CONTINUE
C "TOGGLE" THE INDEX BACK TO THE MASTER AND
C WRITE THE SECONDARY INDEX TO THE DATASET.
CALL STINDEX (1,PINDEX,PLEN,0)
C NOTE THE ABOVE STINDEX CALL DOES NOT USE THE
C OPTIONAL ERROR PARAMETER, AND WILL ABORT
C IF STINDEX DETECTS AN ERROR.
CALL WRITMS (1,SINDEX,SLEN,K,-1,DUMMY,ERR)
IF (ERR.NE.0) THEN
   PRINT*, ' Error on STINDEX, 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])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn</td>
<td>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).</td>
</tr>
<tr>
<td>blocks</td>
<td>Type integer variable, expression, or constant. The maximum number of 512-word blocks that the word-addressable package can use for a buffer.</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ierr</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERS USED =</td>
<td>Number of 512-word buffers used by this dataset</td>
</tr>
<tr>
<td>TOTAL ACCESSES =</td>
<td>Number of accesses. This is the sum of the GETWA and PUTWA calls.</td>
</tr>
<tr>
<td>GETS =</td>
<td>Number of times the user calls GETWA</td>
</tr>
<tr>
<td>PUTS =</td>
<td>Number of times the user calls PUTWA</td>
</tr>
<tr>
<td>FINDS =</td>
<td>Number of times the user calls SEEK</td>
</tr>
<tr>
<td>HITS =</td>
<td>Number of times word addresses desired were resident in memory</td>
</tr>
<tr>
<td>MISSSES =</td>
<td>Number of times no word addresses desired were resident in memory</td>
</tr>
<tr>
<td>PARTIAL HITS =</td>
<td>Number of times that some but not all of the word addresses desired were in memory</td>
</tr>
<tr>
<td>DISK READS =</td>
<td>Number of physical disk reads done</td>
</tr>
<tr>
<td>DISK WRITES =</td>
<td>Number of times a physical disk was written to</td>
</tr>
<tr>
<td>BUFFER FLUSHES =</td>
<td>Number of times buffers were flushed</td>
</tr>
<tr>
<td>WORDS READ =</td>
<td>Number of words moved from buffers to user</td>
</tr>
<tr>
<td>WORDS WRITTEN =</td>
<td>Number of words moved from user to buffers</td>
</tr>
<tr>
<td>TOTAL WORDS =</td>
<td>TOTAL WORDS. Sum of WORDS READ and WORDS WRITTEN</td>
</tr>
<tr>
<td>TOTAL ACCESS TIME =</td>
<td>Real time spent in disk transfers</td>
</tr>
<tr>
<td>AVER ACCESS TIME =</td>
<td>TOTAL ACCESS TIME divided by the sum of DISK READS and DISK WRITES</td>
</tr>
<tr>
<td>EOD BLOCK NUMBER =</td>
<td>Number of the last block of the dataset</td>
</tr>
<tr>
<td>DISK WORDS READ =</td>
<td>Count of number of words moved from disk to buffers</td>
</tr>
<tr>
<td>DISK WDS WRITTEN =</td>
<td>Count of number of words moved from buffers to disk</td>
</tr>
</tbody>
</table>
Table 5-9. WOPEN statistics (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL DISK XFERS =</td>
<td>Sum of DISK WORDS READ and DISK WDS WRITTEN</td>
</tr>
<tr>
<td>BUFFER BONUS % =</td>
<td>TOTAL WORDS divided by value TOTAL DISK XFERS multiplied by 100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Routines Affected</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>WOPEN APUTWA PUTWA GETWA WCLOSE SEEK</td>
<td>Illegal unit number</td>
</tr>
</tbody>
</table>
Table 5-10. Error codes for word-addressable, random access dataset I/O routines (continued)

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Routines Affected</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>WOPEN APUTWA PUTWA GETWA SEEK</td>
<td>Number of datasets has exceeded memory or size availability.</td>
</tr>
<tr>
<td>-3</td>
<td>GETWA SEEK</td>
<td>User attempt to read past end of data</td>
</tr>
<tr>
<td>-4</td>
<td>APUTWA PUTWA GETWA SEEK</td>
<td>User-supplied word address less than or equal to 0</td>
</tr>
<tr>
<td>-5</td>
<td>APUTWA PUTWA GETWA SEEK</td>
<td>User-requested word count greater than maximum</td>
</tr>
<tr>
<td>-6</td>
<td>WOPEN APUTWA PUTWA GETWA WCLOSE SEEK</td>
<td>Illegal dataset name</td>
</tr>
<tr>
<td>-7</td>
<td>APUTWA PUTWA GETWA SEEK</td>
<td>User word count less than or equal to 0</td>
</tr>
</tbody>
</table>
WCLOSE finalizes the additions and changes to the word-addressable dataset and closes the dataset.

Call from FORTRAN:

```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:

```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.
**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)
```

<table>
<thead>
<tr>
<th>dn</th>
<th>Dataset name or unit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Offset into Dataset Parameter Table (DSP) area</td>
</tr>
<tr>
<td>iaddr</td>
<td>Value to get buffer address</td>
</tr>
</tbody>
</table>

WDSET sets dataset names for word-addressable datasets.

Call from FORTRAN:

```
CALL WDSET(dn,idn)
```

<table>
<thead>
<tr>
<th>dn</th>
<th>Dataset name or unit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>idn</td>
<td>On exit, returns dataset name or 0 if error</td>
</tr>
</tbody>
</table>
DATASET MANAGEMENT
SUBPROGRAMS

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(ipto,'DN'L,EXAMPL,'U'L,'MR'L,'DC'L,IDC)
```

`ipto` 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:
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
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:
   (AI) Address of dataset name or unit number
Exit:
   (AI) 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:

\[ val = \text{NUMBLKS}(dn) \]

\begin{itemize}
  \item \textit{val} Number of blocks returned as integer value. If number of blocks cannot be determined, a negative function value is returned.
  \item \textit{dn} Dataset name or unit number
\end{itemize}

**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:

\[ \text{CALL OPEN}(dst,dn,ds,dstat) \]

\begin{itemize}
  \item \textit{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)
  \item \textit{dn} Dataset name or unit number
  \item \textit{ds} Dataset size
  \item \textit{dstat} Dataset status
\end{itemize}

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.
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.
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.
Status:  
<0 if error  
=0 if no error  
>0 if no error and current level is main level

Name of current level (ASCII, left-justified, blank-filled)

Parcel address from where call to current level was made

Parcel address of current level's entry point

Line sequence number corresponding to call address; 0 if none.

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:

\[
\text{CALL PDUMP}(\text{fwd}, \text{lwd}, \text{type})
\]

- \textit{fwd} First word to be dumped
- \textit{lwd} Last word to be dumped
- \textit{type} Dump type code:
  - 0 or 3 Octal dump
  - 1 Floating-point dump
  - 2 Integer dump

$DUMP$ and DUMP dump memory to $\text{OUT}$ and abort the job.

Call from FORTRAN:

\[
\text{CALL DUMP}(\text{fwd}, \text{lwd}, \text{type})
\]

- \textit{fwd} First word to be dumped
- \textit{lwd} Last word to be dumped
- \textit{type} Dump type code:
  - 0 or 3 Octal dump
  - 1 Floating-point dump
  - 2 Integer dump

---

\textbf{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)
```

\textit{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)
```

\textit{regs} Code indicating registers to be copied:
\begin{itemize}
  \item 1 B registers
  \item 2 T registers
  \item 3 B and T registers
  \item 4 V registers
  \item 5 B and V registers
  \item 6 T and V registers
  \item 7 B, T, and V registers
\end{itemize}

\textit{control} Control word (not currently used)

\textit{form} Code indicating format of dump. Dumps from registers S, T, and V are controlled by the following type codes:
\begin{itemize}
  \item 0 Octal
  \item 1 Floating-point
  \item 2 Decimal
  \item 3 Hexadecimal
\end{itemize}
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
```

\textit{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:

```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-I 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).

- **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-I 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
Address of output Dataset Parameter Area (DSP)

xp
Address of Exchange Package

vm
Vector mask (VM) to be formatted

ret
Contents of B00 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 $2^{63} - 1$. 
Number of low-order bits of \( v \) to convert to ASCII character representation \((1 \leq n \leq 64)\). 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. \[ \ldots + \ldots 1 \ldots + \ldots 2 \ldots + \ldots 3 \ldots + \ldots 4 \ldots + \ldots 5 \ldots + \ldots 6 \ldots \]

2. \[ \begin{array}{cccccccc}
P & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
IBA & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
ILA & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
XA & 0 & 0 & 0 & V & L & 0 & 0 \\
DBA & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
DLA & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

3. \[ \begin{array}{cccccccc}
A0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

4. \[ \begin{array}{cccccccc}
aaa & MODES & FLGS \\
OFF & ON & OFF & ON \\
MM & MM & PCI & PCI \\
IUM & IUM & ORE & ORE \\
IMA & IMA & PRE & PRE \\
SEI & SEI & ME & ME \\
FPS & FPS & EEX & EEX \\
WS & WS & NEX & NEX \\
S2 & S2 & S3 & S3 \\
AAA & AAA & AAA & AAA \\
AAAAA & AAAAA & AAAAA & AAAAA \\
ICP & ICP & ICP & ICP \\
\end{array} \]

\[ \text{PROCESSOR} = 0 \quad \text{CLUSTER} = 0 \quad \text{PS} = 0 \]
\[ \text{ERROR TYPE} = \text{aaa} \quad \text{VU} = 0 \]
\[ \text{CHIP SLCT} = \text{aaa} \quad \text{BANK} = 00 \]
\[ \text{READ MODE} = \text{aaa} \quad \text{SYNDROME} = 000 \]

\( a \) means alpha
\( o \) means octal

1. Derived from \textit{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.

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>CAL</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMINIT</td>
<td>$INT</td>
<td>Initiates managed tables</td>
</tr>
<tr>
<td>TMATS</td>
<td>$ATS</td>
<td>Allocates space to a table</td>
</tr>
<tr>
<td>TMADW</td>
<td>$ADW</td>
<td>Adds a word to a table</td>
</tr>
<tr>
<td>TMSRC</td>
<td>$MSC, $MSCO, $SRC</td>
<td>Searches a table with or without masking</td>
</tr>
<tr>
<td>TMPTS</td>
<td>$PTS%, $ZTS%</td>
<td>Presets memory with any given value (default is 0)</td>
</tr>
<tr>
<td>TMMVE</td>
<td>$MVE</td>
<td>Moves memory</td>
</tr>
<tr>
<td>TMMEM</td>
<td>$MEM</td>
<td>Requests additional memory</td>
</tr>
<tr>
<td>TMAMU</td>
<td>$AMV</td>
<td>Returns table management operation statistics</td>
</tr>
</tbody>
</table>

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.
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(022,1X),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:

- (Al) 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,inc)
```

index Index of change
number Table number
inc 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, mword)

index Table index of match, if found; -1 if not found.

tabnum Table number

mask Field being searched for within entry

sword Search word

mword 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

SR-0014 7-17 I
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

SR-0014  7-18  I
TMVSC searches vector table for the search argument.

Call from FORTRAN:

\[
\text{index} = \text{TMVSC}(\text{tabnum}, \text{arg}, \text{nword})
\]

index Table index match, if found; -1 if not found.

\text{tabnum} Table number

\text{arg} Search argument

\text{nword} Number of words per entry group

$\$AMU$\$ returns total allotted table space.

Call from CAL:

\[
\text{CALLV } \$\text{AMU}
\]

Entry:
No arguments required

Exit:

(A2) Allocated length of tables
(A3) Number of table entries

TMAMU reports TMGR statistics.

Call from FORTRAN:

\[
\text{CALL TMAMU}(\text{len}, \text{tabnum}, \text{tabmov}, \text{tabmar}, \text{nword})
\]

\text{len} Allocated length of table

\text{tabnum} Number of tables used

\text{tabmov} Number of table moves

\text{tabmar} Maximum amount of memory used throughout Table Manager

\text{nword} Number of words moved
$PTS%$ presets table space.

Call from CAL:

\[
\begin{array}{ll}
\text{CALLV} & \text{PTS}
\end{array}
\]

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:

\[
\begin{array}{ll}
\text{CALLV} & \text{ZTS}
\end{array}
\]

Entry:
(A1) Base address of space
(A2) Length to be preset or zeroed

Exit:
(V0) Zero vector

\text{TMPTS}$ presets table space.

Call from FORTRAN:

\[
\begin{array}{ll}
\text{CALL} & \text{TMPTS(start,len,preset)}
\end{array}
\]

\begin{itemize}
\item \text{start} Starting address
\item \text{len} Length to be preset
\item \text{preset} Preset value
\end{itemize}

$MVE$ moves memory words to table.
Call from CAL:

\[
\text{CALLV $MVE}
\]

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:

\[
\text{CALL TMMVE(from,to,count)}
\]

\begin{array}{ll}
\text{from} & \text{Address from where words are to be moved} \\
\text{to} & \text{Address where words are to be moved} \\
\text{count} & \text{Number of words to be moved} \\
\end{array}

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:

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
0 & 8 & 16 & 24 & 32 & 40 & 48 & 56 & 63 \\
\hline
0 & \text{GROW} & | & \text{ASEG} \\
1 & \text{HWM} & | & \text{SIZE} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHGROW</td>
<td>0</td>
<td>0-31</td>
<td>The number of times the stack has grown</td>
</tr>
<tr>
<td>SHASEG</td>
<td>0</td>
<td>31-63</td>
<td>Size of additional increments to the stack. Zero means overflow is a fatal error.</td>
</tr>
<tr>
<td>SHHWM</td>
<td>1</td>
<td>0-31</td>
<td>The largest size of the stack so far.</td>
</tr>
<tr>
<td>SHSIZE</td>
<td>1</td>
<td>32-63</td>
<td>Current size of the stack; includes usable stack space for all stack segments</td>
</tr>
</tbody>
</table>

Stack segment control words exist at the top of each stack segment.

Format:

```
  0  8  16  24  32  40  48  56  63
0  .  .  .  .  .  .  .  .
  127
  128
      SIZE     |
  129  \------------------------------------------/ | PSEG
      \------------------------------------------/ |
  130  \------------------------------------------/ | TCPT
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD AREA</td>
<td>0-127</td>
<td>0-63</td>
<td>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.</td>
</tr>
<tr>
<td>SSSIZE</td>
<td>128</td>
<td>0-31</td>
<td>Size of stack segment. Size includes the length of the stack header but does not include the length of stack segment control words.</td>
</tr>
</tbody>
</table>
Stack management routines perform the following functions.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$STKOFEN</td>
<td>Manages stack overflow</td>
</tr>
<tr>
<td>$STKCR</td>
<td>Creates initial stack segment</td>
</tr>
<tr>
<td>$STKUFCK</td>
<td>Manages stack underflow</td>
</tr>
<tr>
<td>$STKUFEX</td>
<td>Releases a topmost stack segment</td>
</tr>
<tr>
<td>$STKDE%</td>
<td>Releases all stack segments</td>
</tr>
</tbody>
</table>

$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)

$STKUPCK determines whether an exit sequence has produced stack underflow. This routine releases the topmost segment if underflow has occurred.

Call From CAL:

CALLV $STKUPCK

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.

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>CAL</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPALLOC</td>
<td>ALLOC%</td>
<td>Allocate a block of memory from the heap</td>
</tr>
<tr>
<td>HPDEALLC</td>
<td>DEALLC%</td>
<td>Return a block of memory to the heap</td>
</tr>
<tr>
<td>HPNEWLEN</td>
<td>NEWLEN%</td>
<td>Change the length of a heap block</td>
</tr>
<tr>
<td>HPCIMOVE</td>
<td>CIMOVE%</td>
<td>Change the length of a heap block and move the block if it cannot be extended in place</td>
</tr>
<tr>
<td>IHPLEN</td>
<td>HPLEN%</td>
<td>Return the length of a heap block</td>
</tr>
<tr>
<td>HPSHRINK</td>
<td>SHRINK%</td>
<td>Return memory from the heap to the operating system</td>
</tr>
<tr>
<td>HPCHECK</td>
<td>HCHECK%</td>
<td>Check the integrity of the heap</td>
</tr>
<tr>
<td>IHPSTAT</td>
<td>HPSTAT%</td>
<td>Return information about the heap</td>
</tr>
<tr>
<td>HPDUMP</td>
<td>HPDUMP%</td>
<td>Write information about the heap to a dataset</td>
</tr>
</tbody>
</table>

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

Deallocation 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, errocode, abort)
```

- `addr` First word address of the block to deallocate (input)
- `errocode` 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:

\[
\text{length} = \text{IHPLLEN}(\text{addr}, \text{ercode}, \text{abort})
\]

*length* Length of the block starting at \text{addr} (output)

*addr* First-word-address of the block (input)

*ercode* 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:

\[
\text{CALLV SHRINK}
\]

No arguments

Call from FORTRAN:

\[
\text{CALL HP SHRINK}
\]

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:

(Sl) 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:

(Sl) 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:

(Sl) Requested value
Call from FORTRAN:

\[ value = \text{IHPSTAT}(\text{code}) \]

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:

\begin{verbatim}
CALLV HPDUMP%
\end{verbatim}

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:

\begin{verbatim}
CALL HPDUMP (code,daname)
\end{verbatim}

code Code for the type of dump requested (see CAL entry point)

daname Name of the dataset to which the dump is to be written. 
daname 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:
(SL) Number of words in pending allocate request
Exit:
(SL) 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:

- `(Al)` Reprieve code entry address
- `(A2)` Reprieve table address
- `(Sl)` Mask
Call from FORTRAN:

```fortran
CALL SETRPV(rpvcoderrpvtab,mask)
```

- `rpvcoderrpvtab` - Routine where control is transferred
- `mask` - A 40-word array reserved for system use
- `mask` - User mask specifying reprieveable 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)
```

**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 Sl 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, ss88)
```

- `ts` Timestamp on entry (type integer)
- `date` Word to receive ASCII date `mm/dd/yy`
- `hhmmss` Word to receive ASCII time `hh:mm:ss`
- `ss88` Word to receive ASCII fractional seconds, `ss88`

**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:

\[
\text{irtc} = \text{TSMT}(ts)
\]

- \text{irtc} Real-time clock value corresponding to specified timestamp
- \text{ts} Timestamp to be converted (type integer)

MTTS converts from a real-time clock value to the corresponding timestamp value.

Call from FORTRAN:

\[
\text{ts} = \text{MTTS}(\text{irtc})
\]

- \text{ts} Timestamp corresponding to real-time clock value (type integer)
- \text{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:

\[
\text{ts} = \text{UNITTS}(\text{periods}, \text{units})
\]

- \text{ts} Number of timestamp units in \text{periods} and \text{units} (type integer)
- \text{periods} Number of timestamp units wanted in standard time units (that is, number of seconds, minutes, etc.); type integer.
- \text{units} Specification for the units in which \text{periods} is expressed. The following values are accepted: 'DAYS'H, 'HOURS'H, 'MINUTES'H, 'SECONDS'H, 'MSEC'H (milliseconds), 'USEC'H (milliseconds), 'USEClOO'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 than 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...'.
```

<table>
<thead>
<tr>
<th>W@JCCPR+0</th>
<th>V</th>
<th>E</th>
<th>R</th>
<th>B</th>
<th>----------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>017</td>
</tr>
<tr>
<td>2</td>
<td>K</td>
<td>E</td>
<td>Y</td>
<td>W</td>
<td>O</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>H</td>
<td>I</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>L</td>
<td>I</td>
<td>T</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>L</td>
<td>V</td>
<td>A</td>
<td>L</td>
<td>U</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>W@JCCPR+</th>
<th>A</th>
<th>C</th>
<th>Q</th>
<th>U</th>
<th>I</th>
<th>R</th>
<th>E</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>D</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>D</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The calling program can then interpret this portion of the control statement. The second CCS call replaces the cracked parameter list with:

<table>
<thead>
<tr>
<th>W@JCCPR+</th>
<th>D</th>
<th>E</th>
<th>P</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>D</td>
<td>E</td>
<td>P</td>
</tr>
</tbody>
</table>

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
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(A1)</td>
<td>table (Address)</td>
</tr>
<tr>
<td>(A2)</td>
<td>number (Actual number)</td>
</tr>
<tr>
<td>(A3)</td>
<td>params (Address)</td>
</tr>
</tbody>
</table>

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*8/
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=Al234567:B1234567 (Two 8-character parameters)
A=Al234567B1234567 (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,obuf,alen,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.
- `obuf` Array to receive the cracked image
- `alen` Integer length in words of the array `obuf`

**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(abuf,atable,ltable,outarray,stattbl)
```

- `abuf` Array to receive the cracked image
- `atable` 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.

<table>
<thead>
<tr>
<th>Array element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal termination</td>
</tr>
<tr>
<td>1</td>
<td>Required keyword not found</td>
</tr>
<tr>
<td>2</td>
<td>Output keyword overflow</td>
</tr>
<tr>
<td>3</td>
<td>Syntax error</td>
</tr>
<tr>
<td>4</td>
<td>Unknown or duplicate keyword</td>
</tr>
<tr>
<td>5</td>
<td>Unexpected separator encountered</td>
</tr>
<tr>
<td>6</td>
<td>Keyword cannot be equated</td>
</tr>
<tr>
<td>7</td>
<td>Keyword must have value</td>
</tr>
<tr>
<td>8</td>
<td>Maximum of 64 keywords exceeded</td>
</tr>
<tr>
<td>9</td>
<td>Invalid return status; cannot recover.</td>
</tr>
<tr>
<td>2</td>
<td>Keyword in error</td>
</tr>
<tr>
<td>3</td>
<td>Ordinal keyword value</td>
</tr>
</tbody>
</table>

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:

\[
\text{CALL CEXPR(char, out, lmt, size)}
\]

- **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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMi</td>
<td>0-55</td>
<td>15-character data item from the character string</td>
</tr>
<tr>
<td>T</td>
<td>56-57</td>
<td>2-bit type field describing ITEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 1- through 8-character symbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 1-word binary value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 1- through 15-character literal string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(delimiting double quotes not included)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 Null, OC is valid</td>
</tr>
</tbody>
</table>
Field | Bits | Description
--- | --- | ---
OC | 58-63 | 6-bit operator code.

<table>
<thead>
<tr>
<th>Operator code (octal)</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>.XOR.</td>
</tr>
<tr>
<td>10</td>
<td>.OR.</td>
</tr>
<tr>
<td>20</td>
<td>.AND.</td>
</tr>
<tr>
<td>30</td>
<td>.NOT.</td>
</tr>
<tr>
<td>40</td>
<td>.EQ.</td>
</tr>
<tr>
<td>41</td>
<td>.NE.</td>
</tr>
<tr>
<td>42</td>
<td>.LT.</td>
</tr>
<tr>
<td>43</td>
<td>.GT.</td>
</tr>
<tr>
<td>44</td>
<td>.LE.</td>
</tr>
<tr>
<td>45</td>
<td>.GE.</td>
</tr>
<tr>
<td>50</td>
<td>+</td>
</tr>
<tr>
<td>51</td>
<td>-</td>
</tr>
<tr>
<td>52</td>
<td>+ (unary)</td>
</tr>
<tr>
<td>53</td>
<td>- (unary)</td>
</tr>
<tr>
<td>60</td>
<td>*</td>
</tr>
<tr>
<td>61</td>
<td>/</td>
</tr>
</tbody>
</table>

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

SR-0014 7-55 I
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.

\[
\text{len} = \text{I00DEL}(\text{string}, \text{max}, \text{olen}, \text{fchar}, \text{lchar})
\]

- **len**: New length of the string addressed by `string`
- **string**: A SKOL string
- **max**: Maximum size of the string
- **olen**: 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.

\[
\text{len} = \text{I00MVC}(\text{string}, \text{max}, \text{olen}, \text{fchar}, \text{lchar}, \text{ichar}, \text{lnum})
\]

- **len**: New length of string addressed by `string`
- **string**: A SKOL string
- **max**: Maximum size of string
- **olen**: 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.

\[
\text{len} = \text{I00MVM}(\text{string}, \text{max}, \text{olen}, \text{fchar}, \text{lchar}, \text{sstri}, \text{sfchar}, \text{slchar}, \text{lnum})
\]
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

sfochar  Index of first character in second string to be inserted

slochar  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.

--------------------------------------------------------------------------------

\[
\text{ord} = \text{I00ORD}(\text{char}, \text{lnum})
\]

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 Al 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

IOOSETUP must be called before any call to IOOREAD. IOOSETUP is described later in this section.

CALL IOOREAD(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 IOOREAD

IOOWRITE 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 IOOWRITE(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
fport Index of first ordinal to be output
lord Index of last ordinal to be output
lnum SKOL source line number of call to IOOWRITE
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, ARnnn
FTERP% processes $FTLIB errors.

Call from CAL:

```
CALLVFTERP%
```

Entry:
(S1) Error message ID defined in $SYSTXT, FTnnn

IOERP% processes $IOLIB errors.

Call from CAL:

```
CALLVIOERP%
```

Entry:
(S1) Error message ID defined in $SYSTXT, FTnnn
(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:

```
CALLVNLERP%
```

Entry:
(S1) Error message ID defined in $SYSTXT, FTnnn
(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, SCnnn

SLERP% processes $SYSLIB errors and abort with traceback.

Call from CAL:

```
CALLV SLERP%
```

Entry:
(S1) Error message ID defined in $SYSTXT, SLnnn

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, UTnnn
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(s1, j1, s2, j2, k)

- **s1**: Word address of the Hollerith source string.
- **j1**: 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).
- **s2**: 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).

\[
\text{CALL MVC(ARRAY,1,ARRAY,2,K-1)}
\]

**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:

\[
\text{CALL PUTBYT(string,position,value)}
\]

- **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:

\[ ik = \text{KOMSTR}(src, isb, num, dest, idb) \]

\( ik \) 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:

\[ \text{CALL FINDCH}(chrs, len, str, ls, nb, ifnd) \]

\( chrs \) 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*

**lz**
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:
\[ n = \frac{(nw \times nbits)}{64} \]

- \( nw \)  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 \( nw \times nbits \) to a multiple of 64.

Call from FORTRAN:

\[
\text{CALL PACK}(p, nbits, u, nw) \]

- \( p \)  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 \( nw \times nbits \) to be a multiple of 64.

Call from FORTRAN:

\[
\text{CALL UNPACK}(p, nbits, u, nw) \]

- \( p \)  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:

```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:

```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.

entry 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 \textit{status} if monitoring is disabled. \textit{to} is required, \textit{sevents} is ignored. \textit{levents} and \textit{oevents} are set by the system. If \textit{to} is 0, an installation-defined default, I@TODEF, is used. If \textit{to} is specified, but less than the installation defined minimum, I@TOMIN, the installation minimum is used with no notification. If \textit{levents} is 0 on return, timeout is the only event that occurred.

'RETURN' Requests \textit{levents} and \textit{oevents} be set by the system; all other words are ignored. An error is returned in \textit{status} if monitoring is disabled.

\textit{status} Status; integer variable set by the system. \textit{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').

\textit{sevents} Integer array set by the user containing the events to be monitored. \textit{levents} is the number of events specified in \textit{sevents}. The events can be selected from the following.

- 'IJ' Inter-job message received
- 'UD' Unsolicited operator message received
- 'OR' Operator reply received

The following are privileged:

- 'CH' Channel driver done
- 'IQ' SDT placed in Input queue
- 'QQ' SDT placed in Output queue

\textit{to} Timeout duration in milliseconds (rightmost 24 bits); integer variable set by the user.

\textit{oevents} Integer array set by the system to the occurred events. \textit{levents} is the number of event words that have been placed in \textit{oevents} by the system. See \textit{sevents} for possible values.
**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**.

**GETBl** returns the contents of register Bl (address of calling subroutine name) in register Sl.

Call from FORTAN:

```
bl=GETBl
```

**(bl**  
Calling subroutine name address (Bl)

**GETLPP** returns the lines per page from field **JCLPP** of the JCB in register Sl.

Call from FORTAN:

```
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 FORTAN:

```
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:

\[
\text{byte=IGTBYT(string,position)}
\]

**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:

\[
\text{CALL IJCOM(array,larray,lentry,nentry,status)}
\]

**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 'IJMS$OPEN' to open a path. 'IJMS$BP' in STAT means the path was busy).

- 'IJMS$NOP' \(\text{Subfunction is a no op.}\)
- 'IJMS$REC' \(\text{Subfunction marks the job as receptive. RCB is required; all other words are ignored.}\)
- 'IJMS$OPEN' \(\text{Subfunction initiates an attempt to open a communication path with another job. HLEN, TID, and NCB are required; all other words are ignored.}\)
'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.

\[ \text{Array} \] Length of array; integer variable set by the user.

\[ \text{Length} \] Length of each parameter block entry in array; integer variable set by the user.

\[ \text{Number} \] Number of parameter blocks in array; integer variable set by the user. Default is 1.

\[ \text{Status} \] 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):

```
<table>
<thead>
<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>
</thead>
<tbody>
<tr>
<td></td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PCA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 7-2. Parameter Control Table**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCNP</td>
<td>0-7</td>
<td>Number of parameters to be inserted</td>
</tr>
<tr>
<td>PCPCA</td>
<td>40-63</td>
<td>Address of parameter control entry (PCE)</td>
</tr>
</tbody>
</table>

Entry (one per parameter):

```
<table>
<thead>
<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>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SW</td>
<td>SB</td>
</tr>
</tbody>
</table>
```

**Figure 7-3. Parameter control entry**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCSW</td>
<td>40-45</td>
<td>Starting word in text (0, 1,...)</td>
</tr>
<tr>
<td>PCSB</td>
<td>46-51</td>
<td>Starting bit in starting word (0...63)</td>
</tr>
<tr>
<td>PCLEN</td>
<td>52-63</td>
<td>Length in bits (maximum is 64 bits)</td>
</tr>
</tbody>
</table>
Figure 7-4. Message Control Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCNM</td>
<td>0-7</td>
<td>Number of messages defined in MCT</td>
</tr>
<tr>
<td>MCDBA</td>
<td>8-31</td>
<td>Address of buffer in which to assemble message</td>
</tr>
<tr>
<td>MCPCA</td>
<td>40-63</td>
<td>Address of Parameter Control Table (PCT)</td>
</tr>
<tr>
<td>MCPCO</td>
<td>0-7</td>
<td>Offset into PCT; selects desired entry.</td>
</tr>
<tr>
<td>MCMWL</td>
<td>36-39</td>
<td>Message length in words</td>
</tr>
<tr>
<td>MCMWA</td>
<td>40-63</td>
<td>Message address in words</td>
</tr>
</tbody>
</table>

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**: (Sl) Dataset name containing the absolute load module
CALL from FORTRAN:

\[
\text{CALL LGO('dn'L)}
\]

\(dn\)  Dataset name containing the absolute load module

LOC returns memory address of specified variable or array.

CALL from FORTRAN:

\[
\text{address=LOC(arg)}
\]

\(address\)  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:

\[
\text{CALL LOGECHO}
\]

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:

\[
\text{CALL MEMORY}(\text{code}, \text{value})
\]

code

Determines what information or action is requested (blank-filled).

'UC'  
\(\text{value}\) 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'  
\(\text{value}\) specifies the number of words of field length to be allocated to the job. If FL is specified and \(\text{value}\) 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. \(\text{value}\) is ignored.

'AUTO'  
The job is put in automatic field length reduction mode. \(\text{value}\) is ignored.

'MAXFL'  
The maximum field length allowed the job is returned in \(\text{value}\).

'CURFL'  
The current field length is returned in \(\text{value}\).

'TOTAL'  
The total amount of unused space in the job is returned in \(\text{value}\).

\(\text{value}\)

An integer value or variable when \text{code} is 'UC' or 'FL'.
An integer variable that is to contain a returned value if \text{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, lev1, lev2[, recall])
```

- \dn: Dataset in which overlay resides
- \lev1: Overlay level 1 (LEV1)
- \lev2: 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, buf1)
```

- \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.
  - '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.

buf1  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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMRSF</td>
<td>0</td>
<td>Subfunction (PM$ON, PM$OFF, PM$REP, PM$RST)</td>
</tr>
<tr>
<td>HMRGN</td>
<td>1</td>
<td>Group number (0 through 3) for PM$ON</td>
</tr>
<tr>
<td>HMRNW</td>
<td>2</td>
<td>Length of request block</td>
</tr>
<tr>
<td>HMRNU</td>
<td>3</td>
<td>Number of words used</td>
</tr>
<tr>
<td>HMRBH</td>
<td>4</td>
<td>Number of words in block header</td>
</tr>
<tr>
<td>HMRTS</td>
<td>5</td>
<td>Set nonzero if block too small</td>
</tr>
<tr>
<td>HMRCT</td>
<td>6</td>
<td>Offset to first group counter in subblock</td>
</tr>
<tr>
<td>HMRCP</td>
<td>7</td>
<td>Offset to first group accounted cycles</td>
</tr>
<tr>
<td>HMRGE</td>
<td>8</td>
<td>Length of counter group entry in subblock</td>
</tr>
<tr>
<td>HMRNC</td>
<td>9</td>
<td>Number of counters in each group entry</td>
</tr>
<tr>
<td>HMRNG</td>
<td>10</td>
<td>Number of groups in each subblock</td>
</tr>
<tr>
<td>HMRLE</td>
<td>11</td>
<td>Length of subblock entries</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMTN</td>
<td>0</td>
<td>User task number</td>
</tr>
<tr>
<td>HMGRP</td>
<td>1</td>
<td>Latest performance monitor group number</td>
</tr>
<tr>
<td>HMCNT0</td>
<td>2-9</td>
<td>Group 0, counter 0 through 7</td>
</tr>
<tr>
<td>HMCCY0</td>
<td>10</td>
<td>Group 0 accounted CP cycles</td>
</tr>
<tr>
<td>HMCNT1</td>
<td>11-18</td>
<td>Group 1, counter 0 through 7</td>
</tr>
<tr>
<td>Field</td>
<td>Word</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>HMCCY1</td>
<td>19</td>
<td>Group 1 accounted CP cycles</td>
</tr>
<tr>
<td>HMCNT2</td>
<td>20-27</td>
<td>Group 2, counter 0 through 7</td>
</tr>
<tr>
<td>HMCCY1</td>
<td>28</td>
<td>Group 1 accounted CP cycles</td>
</tr>
<tr>
<td>HMCNT3</td>
<td>29-36</td>
<td>Group 3, counter 0 through 7</td>
</tr>
<tr>
<td>HMCCY1</td>
<td>37</td>
<td>Group 1 accounted CP cycles</td>
</tr>
</tbody>
</table>

$RBN$ and $RNB$ convert trailing blanks to nulls.

Call from CAL:

```
CALLV $RBN
```

- **Entry:** (SL) Argument to be converted
- **Exit:** (SL) 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

<table>
<thead>
<tr>
<th>group</th>
<th>Performance Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Number of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Instructions issued</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CPUs holding issue</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Fetches</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>I/O references</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>CPU references</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Floating-point add operations</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Floating-point multiply operations</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Floating-point reciprocal operations</td>
</tr>
<tr>
<td>1</td>
<td>Hold issue conditions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Semaphores</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Shared registers</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A registers and functional units</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>S registers and functional units</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>V registers</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>V functional units</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Scalar memory</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Block memory</td>
</tr>
<tr>
<td>2</td>
<td>Number of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Fetches</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Scalar references</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Scalar conflicts</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>I/O references</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>I/O conflicts</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Block references</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Block conflicts</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Vector memory references</td>
</tr>
<tr>
<td>3</td>
<td>Number of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>000 - 017 instructions</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>020 - 137 instructions</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>140 - 157, 175 instructions</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>160 - 174 instructions</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>176, 177 instructions</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Vector integer operations</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Vector floating-point operations</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Vector memory references</td>
</tr>
</tbody>
</table>
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 columns 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 ('PODOl - 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:

```plaintext
STATUS=SYSTEM(function,arg_1,arg_2)
```

<table>
<thead>
<tr>
<th>function</th>
<th>Executive request number</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg_1</td>
<td>Optional argument (required by some requests)</td>
</tr>
<tr>
<td>arg_2</td>
<td>Optional argument (required by same requests)</td>
</tr>
<tr>
<td>STATUS</td>
<td>Status returned in SO (function dependent)</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 WBUFFER,(arg)
```

Entry:
- \textit{arg} \hspace{1cm} \text{Index of the desired character}

Exit:
- \textit{(S1)} \hspace{1cm} \text{Requested character right-justified, zero-filled}

Users can call procedures from a CFT or CAL program, using the \texttt{F$\$CSP} system call. FORTRAN users should use the \texttt{SYSTEM} routine with the appropriate parameters. CAL users should set up the call parameters in the proper registers and \texttt{EX}.
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

\[ \text{WRITE (10)} \]

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.

\[
\text{FUNCTION P$$HPAD: INTEGER; EXTERNAL;}
\]

\[
\begin{align*}
\text{.} \\
\text{.} \\
\text{.} \\
\text{Integer variable := P$$HPAD}
\end{align*}
\]
P$ABORT

P$ABORT calls P$OSDXP to abort the current job step. No parameters are required.

PROCEDURE P$ABORT; EXTERNAL;

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;

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.

```plaintext
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.

```plaintext
PROCEDURE P$ENDP;EXTERNAL;

P$ENDP
```
**P$EOF**

P$EOF returns the end-of-file status for a Pascal text/record file.

```pascal
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.

```pascal
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 ARRAY (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.

```pascal
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.

```pascal
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.

```pascal
FUNCTION P$MEMRY (REQUEST: INTEGER): INTEGER; EXTERNAL;

Integer variable := P$MEMRY (request code)
```
Request codes for P$MEMRY:

<table>
<thead>
<tr>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>QUERY SIZE OF HEAP</td>
</tr>
<tr>
<td>2</td>
<td>QUERY HIGH WATER MARK</td>
</tr>
<tr>
<td>3</td>
<td>QUERY LOW WATER MARK</td>
</tr>
<tr>
<td>4</td>
<td>QUERY NUMBER OF ALLOCATED HEAP AREAS</td>
</tr>
<tr>
<td>5</td>
<td>QUERY NUMBER OF TIMES HEAP HAS GROWN</td>
</tr>
<tr>
<td>6</td>
<td>QUERY NUMBER OF TIMES HEAP HAS SHRUNK</td>
</tr>
<tr>
<td>7</td>
<td>QUERY NAME OF LAST HEAP ROUTINE USED</td>
</tr>
<tr>
<td>8</td>
<td>QUERY NAME OF LAST ROUTINE TO CALL A HEAP ROUTINE</td>
</tr>
<tr>
<td>9</td>
<td>QUERY LAST HEAP ITEM ALLOCATED/DEALLOCATED/ETC.</td>
</tr>
<tr>
<td>10</td>
<td>QUERY SIZE OF LARGEST FREE BLOCK</td>
</tr>
<tr>
<td>11</td>
<td>QUERY AMOUNT HEAP CAN SHRINK</td>
</tr>
<tr>
<td>12</td>
<td>QUERY AMOUNT HEAP CAN GROW</td>
</tr>
<tr>
<td>13</td>
<td>QUERY HEAP F.W.A.</td>
</tr>
<tr>
<td>14</td>
<td>QUERY HEAP L.W.A.</td>
</tr>
<tr>
<td>15</td>
<td>QUERY NUMBER OF TIMES HPALLOC CALLED</td>
</tr>
<tr>
<td>16</td>
<td>QUERY NUMBER OF TIMES HPDEALLC CALLED</td>
</tr>
<tr>
<td>17</td>
<td>DO A HEAP SHRINK</td>
</tr>
<tr>
<td>18</td>
<td>DO A HEAP INTEGRITY CHECK</td>
</tr>
<tr>
<td>19</td>
<td>DUMP HEAP STATISTICS TO $OUT</td>
</tr>
<tr>
<td>20</td>
<td>DUMP HEAP CONTROL WORDS TO $OUT(ALL BLKS)</td>
</tr>
<tr>
<td>21</td>
<td>DUMP HEAP CONTROL WORDS TO $OUT(FREE BLKS VIA NEXT)</td>
</tr>
<tr>
<td>22</td>
<td>DUMP HEAP CONTROL WORDS TO $OUT(FREE BLKS VIA LAST)</td>
</tr>
</tbody>
</table>

P$MOD

P$MOD (call by value from CAL only) returns the modulus in register Sl. P$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)
POSDEP

POSDEP 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 POSDxx routines that perform I/O).

```
FUNCTION POSDEP (DSN: ALFA): INTEGER; EXTERNAL;
  
  Integer variable := POSDEP ('dataset name')
```

POSDJT

POSDJT is an operating system dependent routine that returns the number of CPU seconds used by the job. No parameters are required.

```
FUNCTION POSDJT: REAL; EXTERNAL;
  
  Real variable := POSDJT
```

POSDLM

POSDLM 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 POSDLM (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$OSDRC (DSP: INTEGER; VAR BUF: BUFFER;
BUFLLENGTH: INTEGER; VAR RETURNINFO: INTEGER); EXTERNAL;
    .
    .
P$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$OSDPR (PROCEDURE P; EP: EXCHANGEPACKAGE;
MASK: INTEGER);
    .
    .
P$OSDPR (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;
BUFSIZE: 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.

```pascal
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.

```pascal
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$REWRIT, the file pointer is positioned at the beginning of the file.
PROCEDURE P$REWRT (VAR F:FILE OF type);EXTERNAL;
  
  
  P$REWRT (file variable)

P$REWRT 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

```pascal
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$REWRT for information on opening Pascal textfiles. See procedures P$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$FRAME**

P$FRAME returns a pointer to the base of the stack frame of the calling procedure. No parameters are required.

```
FUNCTION P$FRAME: INTEGER; EXTERNAL;
  
  Integer variable := P$FRAME
```

**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$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.

```pascal
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.

```pascal
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`.

```pascal
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.

```pascal
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.

```pascal
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 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:

```c
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 c create task to execute second half of the data
task2ary(1)=3
task2ary(3)="task 2"
```

SR-0014 9-2
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 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 control array (described under subtitle task 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-l 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(keyword1,value1,keyword2,value2,...)

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>
<thead>
<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>
</thead>
<tbody>
<tr>
<td>LENGTH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK VALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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.

EWAIT 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 EWAIT (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.
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:

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

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

$$\text{arcsin}(-x) = -\text{arcsin}(x).$$

1. Calculate a reduced argument, $r$, a multiplier, $m$, and an addend, $a$, depending on $x$:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$r$</th>
<th>$m$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq x \leq \sin(\pi/6)$</td>
<td>$x$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\sin(\pi/6) &lt; x \leq \sin(\pi/3)$</td>
<td>$T_2(x) = 2x^2 - 1$</td>
<td>1/2</td>
<td>$\pi/4$</td>
</tr>
<tr>
<td>$\sin(\pi/3) &lt; x \leq \sin(5\pi/12)$</td>
<td>$T_4(x) = 8x^4 - 8x^2 + 1$</td>
<td>1/4</td>
<td>$3\pi/8$</td>
</tr>
<tr>
<td>$\sin(5\pi/12) &lt; x \leq 1$</td>
<td>$\sqrt{1-x^2}$</td>
<td>-1</td>
<td>$\pi/2$</td>
</tr>
</tbody>
</table>

2. Compute $\text{arcsin}(r)$ using a rational function of the form

$$r^P(r^2)/Q(r^2)$$

where $P$ and $Q$ are polynomials of degree 4 in $r^2$. 

SR-0014
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 \( 1/\sqrt{2} < f < \sqrt{2} \). 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 \( 1/2 < c < 1 \). When \( 1/2 < c < 1/\sqrt{2} \), then \( 1 < 2c < \sqrt{2} \).

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 \( 1/\sqrt{2} < c < 1 \), 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 \( \omega = (z - t)/(1 + tz) \)
4. Compute $\arctan(\omega)$ using a polynomial $P(\omega)$ of degree 9.

5. Let $a = 0$ if $a \leq 1$ and $a = \pi/4$ if $a > 1$.

6. Let $c = \arctan(t)$.

7. Form $\arctan(\alpha) = (a - c) + c - P(\omega)$.

8. Insert the result sign depending on the magnitude of $a$, the sign of $x$ and the sign of $\arctan(\alpha)$.

**ATAN2 AND ATAN2V**

Use the same algorithm as $\text{ATAN}$ and $\text{ATANV}$.

**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 $v \sqrt{1 + (u/v)^2} = v \sqrt{\omega}$ where $\omega$ lies in the interval $[1,2]$. This expression avoids possible overflow in the computation $x^2 + y^2$.

4. Compute an initial approximation to $\sqrt{\omega}$ as

$$ a_0 = 33/32 + 3/8 \left(\omega - 33/32\right) $$

5. Apply the Newton-Raphson iteration three times to compute $\sqrt{\omega}$.

$$ a_{n+1} = \frac{1}{2} \left(a_n + \omega/a_n\right) $$

three times to compute $\sqrt{\omega}$.

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 \( n = \left[ \frac{\pi}{4} |x| \right] \) 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 \( [\#a \&(d \backslash c)] \times [a \&(d \backslash c)] \).

COSH AND COSHV

1. Compute the integer \( n \) nearest to \( |x|/\ln 2 \). \( x = n \ln 2 + f \) where \( |f| < \frac{\ln 2}{2} \).

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
   
   \[ e^x = e^{n \ln 2 + f} = (e^{\ln 2})^n e^f = 2^n e^f \]

\( \backslash = \) 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

\[ \sinh(x) = \frac{e^x - e^{-x}}{2} = \frac{2^n e^f - 2^{-n} e^{-f}}{2} = 2^{n-1} e^f - 2^{-n-1} e^{-f} \]

\[ = \left(\frac{e^f + e^{-f}}{2} + \frac{e^f - e^{-f}}{2}\right) 2^{n-1} - \left(\frac{e^f + e^{-f}}{2} - \frac{e^f - e^{-f}}{2}\right) 2^{-n-1} \]

\[ = (\cosh(f) + \sinh(f)) 2^{n-1} - (\cosh(f) - \sinh(f)) 2^{-n-1} \]

\[ = [1 + (\cosh(f) - 1) + \sinh(f)] 2^{n-1} - [1 + (\cosh(f) - 1) - \sinh(f)] 2^{-n-1} \]

\[ = [1 + (P + Q)] 2^{n-1} - [1 + (P - Q)] 2^{-n-1} \]

\[ = 2^{-1} + (P + Q)] 2^{n-1} - [1 + (P - Q)] 2^{-n-1} + 2^{n-3} + 2^{n-3} \]

\[ = 2^{n-3} + (2^{n-3} - [1 + (P - Q)] 2^{-n-1}) + [2^{-1} + (P + Q)] 2^{n-1} \]

\[ = 2^{n-3} + (2^{n-3} - [1 + (P - Q)] 2^{-n-1}) + [2^{-1} + (P + Q)] 2^{n-1} \]
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, \( a = 0 \).

   \[ b = \text{Low-order bit of } m \]
   \[ c = \text{High-order bit of } m \]
   \[ d = \text{Argument sign} \]

4. Compute the reduced argument \( r = |x| - \pi/4(n + 6) \) in double-precision.

5. Define \( t = a\backslash b\backslash c \).

6. Compute \( \tan(r) \) 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 \( c\backslash d \).

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 COSH% 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 \( \sin(z) \).

\[ \backslash\equiv \text{Logical difference} \]
CSQRT AND CSQRTV

The complex square root of \( z = x + iy \) is defined by

\[
\sqrt{z} = \sqrt{\frac{1}{2}(|z| + x)} + i \sqrt{\frac{1}{2}(|z| - x)} = u + iv
\]

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

\[
z^a = e^{a \ln(z)} = \exp(a \ln|z| + i \arctan(z)) = \exp((a \ln|z| - \arctan(y/x)) + i(\arctan(y/x) + \ln|z|))
\]

\[
= e^s \cos(t) + is \sin(t)
\]

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 than 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

\[
e = 2^0 + 2^{-1} + ... + 2^{-n} + e_n
\]

where \( e_i \) is either 0 or 1.

4. Form

\[
e_i = \pi \cdot 2^i
\]

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
\[
\arcsin(x) = \arctan\left(\frac{x}{\sqrt{1 - x^2}}\right)
\]

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 \(y/x < 2^{-17}\), 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 \(q\).

<table>
<thead>
<tr>
<th>(q)</th>
<th>(r)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &lt; (q) &lt; (\tan(\pi/8))</td>
<td>((q - a)/(1 + qa))</td>
<td>(\pi/16)</td>
</tr>
<tr>
<td>(\tan(\pi/8) \leq q &lt; 1)</td>
<td>((q - b)/(1 + qb))</td>
<td>(3\pi/16)</td>
</tr>
<tr>
<td>(1 \leq q &lt; \sqrt{2})</td>
<td>((1 - qb)/(q + b))</td>
<td>(5\pi/16)</td>
</tr>
<tr>
<td>(\sqrt{2} \leq q &lt; \infty)</td>
<td>((1 - qa)/(q + a))</td>
<td>(7\pi/16)</td>
</tr>
</tbody>
</table>

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 $e^w$ 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 $1/\sqrt{2} \leq f \leq \sqrt{2}$. 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 $z = f - 1/f + 1$.

$$
a_0 = c_1 z + c_2 z^3 + c_5 z^5 + c_7 z^7.
$$

3. Apply the Newton-Raphson iteration

$$
a_{n+1} = a_n - \frac{q(a_n)}{q'(a_n)} = a_n - \frac{c_n - a_n}{q'(a_n)} = a_n - (1 - f e^{-a_n})
$$

two times to compute $\ln(f)$. $a_2$ is computed directly from $a_0$ using the formula

$$
a_2 = [(a_0 - t_u) - t_1] - t_u^2 (\frac{1}{2} + \frac{t_u}{3}).
$$

The formula is derived by neglecting terms that are insignificant.

Let $r = f e^{-a_0}$ and $t = 1 - r = t_u + t_1$ where $t_u$ is the most significant part of $t$ and $t_1$ is the least significant. Then

$$
a_2 = a_1 - (1 - f e^{-a_1}) = a_1 - [1 - f e^{-a_0 - t_u}]
$$

$$
= a_1 - (1 - f e^{-a_0} e^{t_u}) = a_1 - (1 - r e^{t_u}) = a_0 - t_u - (1 - r e^{t_u})
$$

ignoring the least significant part of $t$.

Expanding $e^{t_u}$ as a power series truncated to four terms, we have

$$
a_0 - t_u - (1 - r e^{t_u}) \approx a_0 - t_u - [1 - r(1 + t_u + \frac{t_u^2}{2} + \frac{t_u^3}{3})]
$$

$$
\approx a_0 - t_u - (1 - r - rt_u - \frac{rt_u^2}{2} - \frac{rt_u^3}{6})
$$

$$
\approx a_0 - t_u - [t_u + t_1 - (t_u - t_u^2) - (\frac{t_u^2}{2} - \frac{t_u^3}{2} - \frac{t_u^3}{6})]
$$

$$
\approx a_0 - t_u - t_1 - \frac{t_u^2}{2} - \frac{t_u^3}{3}
$$

$$
\approx (a_0 - t) - t_u^2 (\frac{1}{2} + \frac{t_u}{3})
$$
Note that in the derivation above, some terms have been ignored.

\[ 1 - r = t_u + t_l = r = 1 - t_u - t_l \]

\[ r t_u = (1 - t_u - t_l) t_u = t_u - t_u^2 - t_l t_u = t_u - t_u^2 \]

\[ \frac{r t_u^2}{2} = \frac{1 - t_u - t_l}{2} t_u^2 = \frac{t_u}{2} - \frac{t_u^3}{2} - \frac{t_u^3}{2} \]

\[ \frac{r t_u^3}{6} = \frac{1 - t_u - t_l}{6} t_u^3 = \frac{t_u}{6} - \frac{t_u^3}{6} - \frac{t_u^3}{6} \]

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 \( \text{SQRT}\% \) 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
\[ \tan(x) = \frac{\sin(x)}{\cos(x)} \]

DTANH AND DTANHV

Compute using the identity
\[ \tanh(x) = \frac{\sinh(x)}{\cosh(x)} \]

For \( x = 0 \), return \( \tanh(0) = 0 \)
For \( x < 0 \), \( \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \)
For \( x > 0 \), \( \tanh(x) = \frac{1 - e^{-2x}}{e^{-2x} + 1} \)

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 DTOIVVV

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
   \[ y = x \cdot \frac{2}{\ln 2} = 2n_1 + n_2 + f \]
where \( n_1 \) and \( n_2 \) are integers, \( 0 \leq n_2 < 2 \) and \( 0 < f < 1 \). Observe that
\[
\begin{align*}
e^x &= e^{(x/\ln 2)(\ln 2)} = (e^{\ln 2})^{x/\ln 2} = 2^{x/\ln 2} \\
&= 2^{(n_1 + n_2/2 + f/2)} = 2^{n_1}2^{n_2/2}2^{f/2}.
\end{align*}
\]

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 \( \text{ABS}(i) \) as the initial product.

2. Using \( \text{ABS}(j) \) 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 \( j \mod 2 \neq 0 \), 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.
Let \( \langle S(n) \rangle \) be the linear congruential sequence produced by setting
\[
S_{n+1} = mS_n \mod 2^{48}, \quad n \geq 0.
\]
Select a suitable multiplier \( m \) and starting seed \( S_0 \). Use \( S_n, \quad n > 0 \), 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 \((i \mod N)(j \mod N) = ij \mod N\) for integers \(i,j,N\), the result is
\[
S_{n+64} = m^{64}S_n \mod 2^{48}
\]
enabling computation of 64 seeds at a time using vector instructions and 64 initial values \( S_i, \quad 1 \leq i \leq 64 \). 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 $\text{RANSET}$ 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 $\prod_{i=1}^{e_i} z^i$
**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

\[
\frac{y_n}{2^{4-n}} = \left( \frac{y_n - 1}{2^{4-(n-1)}} \right) + \left( \frac{2^{4-(n-1)}}{y_{n-1}} \right) \frac{x}{2^{4-(n-1)} \cdot 2^{4-(n-1)}}
\]

4. Using a full-precision divide approximation, compute a final Newton-Raphson iteration yielding

\[
\sqrt{x} \approx y_4 = \frac{y_3}{2} + \frac{2}{y_3} \frac{x}{4}
\]

TANH AND TANHV

1. If \( x > 17.3287 \), 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 \( \tanh(x) = 1 - 2/1 + e^{2x} \).
BIBLIOGRAPHY


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

<table>
<thead>
<tr>
<th>Entry</th>
<th>Domain</th>
<th>Accuracy</th>
<th>Average Result Rate (μ Secs/Result)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Max. Error (x10^-14)</td>
<td>Stdd Dev (x10^-15)</td>
</tr>
<tr>
<td>ACOS%</td>
<td>(-1, 1)</td>
<td>2.910</td>
<td>4.233</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>2.134</td>
<td>3.368</td>
<td>2.2</td>
</tr>
<tr>
<td>(0, 1/2)</td>
<td>0.944</td>
<td>2.695</td>
<td>2.0</td>
</tr>
<tr>
<td>(1/2, sin π/3)</td>
<td>1.064</td>
<td>3.150</td>
<td>2.2</td>
</tr>
<tr>
<td>(sin π/3, sin5 π/12)</td>
<td>2.469</td>
<td>5.635</td>
<td>2.5</td>
</tr>
<tr>
<td>(sin5 π/12, 1)</td>
<td>2.134</td>
<td>4.564</td>
<td>4.2</td>
</tr>
<tr>
<td>ALOG%</td>
<td>(1/2, 2)</td>
<td>1.639</td>
<td>3.886</td>
</tr>
<tr>
<td>(0, ∞)</td>
<td>1.562</td>
<td>4.282</td>
<td>2.0</td>
</tr>
<tr>
<td>ALOG10%</td>
<td>(1/2, 2)</td>
<td>1.500</td>
<td>4.077</td>
</tr>
<tr>
<td>(0, ∞)</td>
<td>1.786</td>
<td>3.924</td>
<td>1.9</td>
</tr>
<tr>
<td>ASIN%</td>
<td>(-1, 1)</td>
<td>1.765</td>
<td>3.888</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1.655</td>
<td>3.874</td>
<td>2.0</td>
</tr>
<tr>
<td>(0, 1/2)</td>
<td>1.985</td>
<td>4.008</td>
<td>1.8</td>
</tr>
<tr>
<td>(1/2, sin π/3)</td>
<td>1.289</td>
<td>3.093</td>
<td>2.0</td>
</tr>
<tr>
<td>(sin π/3, sin5 π/12)</td>
<td>1.349</td>
<td>5.594</td>
<td>2.2</td>
</tr>
<tr>
<td>(sin5 π/12, 1)</td>
<td>0.578</td>
<td>2.152</td>
<td>3.9</td>
</tr>
<tr>
<td>ATAN%</td>
<td>(-π/2, π/2)</td>
<td>1.057</td>
<td>3.549</td>
</tr>
<tr>
<td>ATAN%</td>
<td>(-π/4, π/4)</td>
<td>0.474</td>
<td>1.656</td>
</tr>
<tr>
<td>COS%</td>
<td>(-π/4, π/4)</td>
<td>0.100</td>
<td>2.312</td>
</tr>
<tr>
<td>(-π/100, 100)</td>
<td>50.043</td>
<td>16.824</td>
<td>2.1</td>
</tr>
<tr>
<td>COSH%</td>
<td>(-0.301, 0.301)</td>
<td>0.712</td>
<td>3.177</td>
</tr>
<tr>
<td>(-5677, 5677)</td>
<td>1.319</td>
<td>3.980</td>
<td>2.1</td>
</tr>
<tr>
<td>COT%</td>
<td>(-π/4, π/4)</td>
<td>1.835</td>
<td>5.058</td>
</tr>
<tr>
<td>(-100, 100)</td>
<td>60.847</td>
<td>24.833</td>
<td>2.3</td>
</tr>
</tbody>
</table>

* Samples are exponentially distributed.*
Table B-1. Statistics for single-precision, single-argument subprograms (continued)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Domain</th>
<th>Accuracy Max. Error (x10^-14)</th>
<th>Accuracy Std. Dev (x10^-15)</th>
<th>Average Result Rate (π Secs/Result)</th>
<th>Vector Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP%</td>
<td>(-1, 1)</td>
<td>2.170</td>
<td>7.482</td>
<td>1.6</td>
<td>2.5 1.3 0.7 0.3 0.2 0.2</td>
</tr>
<tr>
<td>%EXP%</td>
<td>(-5677, 5677)</td>
<td>2.045</td>
<td>7.408</td>
<td>1.6</td>
<td>2.5 1.3 0.7 0.3 0.2 0.2</td>
</tr>
<tr>
<td>SIN%</td>
<td>(-π/4, π/4)</td>
<td>0.698</td>
<td>2.244</td>
<td>1.8</td>
<td>3.9 2.0 1.1 0.5 0.3 0.3</td>
</tr>
<tr>
<td>%SIN%</td>
<td>(-π, π)</td>
<td>0.927</td>
<td>2.108</td>
<td>1.9</td>
<td>3.8 2.0 1.1 0.5 0.3 0.3</td>
</tr>
<tr>
<td></td>
<td>(-100, 100)</td>
<td>20.809</td>
<td>8.512</td>
<td>2.0</td>
<td>3.8 2.0 1.1 0.5 0.3 0.3</td>
</tr>
<tr>
<td>SINH%</td>
<td>(-0.301, 0.301)</td>
<td>0.686</td>
<td>2.273</td>
<td>2.3</td>
<td>4.0 2.1 1.1 0.6 0.4 0.3</td>
</tr>
<tr>
<td>%SINH%</td>
<td>(-5677, 5677)</td>
<td>1.032</td>
<td>3.974</td>
<td>2.2</td>
<td>4.0 2.1 1.1 0.6 0.4 0.3</td>
</tr>
<tr>
<td>SQRT%</td>
<td>(0, 1)</td>
<td>0.767</td>
<td>1.931</td>
<td>1.5</td>
<td>2.3 1.2 0.6 0.3 0.1 0.1</td>
</tr>
<tr>
<td>%SQRT%</td>
<td>(0, ∞)†</td>
<td>0.724</td>
<td>2.099</td>
<td>1.5</td>
<td>2.3 1.2 0.6 0.3 0.1 0.1</td>
</tr>
<tr>
<td>TAN%</td>
<td>(-π/4, π/4)</td>
<td>1.444</td>
<td>4.274</td>
<td>1.8</td>
<td>3.4 1.8 0.9 0.5 0.3 0.2</td>
</tr>
<tr>
<td>%TAN%</td>
<td>(-100, 100)</td>
<td>60.456</td>
<td>24.830</td>
<td>2.1</td>
<td>3.4 1.8 0.9 0.5 0.3 0.2</td>
</tr>
<tr>
<td>TANH%</td>
<td>(-20, 20)</td>
<td>5.671</td>
<td>5.822</td>
<td>2.8</td>
<td>4.6 2.4 1.2 0.6 0.3 0.3</td>
</tr>
<tr>
<td>%TANH%</td>
<td>(-1/2, 1/2)</td>
<td>8.127</td>
<td>13.481</td>
<td>2.7</td>
<td>4.6 2.3 1.2 0.6 0.3 0.3</td>
</tr>
</tbody>
</table>

† Samples are exponentially distributed.
This section presents timings of selected $SCILIB routines and some comparisons 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=$SCILIB,ID=myown.
ACCESS,DN=CFT,ID=myown.
ACCESS,DN=SCILBPL,ID=myown.
UPDATE,P=SCILBPL,I=0,Q=TIMINGS.
CFT,I=$CPL.
CFT.
LDR.
/EOF
   CALL TIMINGS
   STOP
   END
```

<table>
<thead>
<tr>
<th>Subroutine name</th>
<th>name of the $SCILIB routine and a brief description of its function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop length</td>
<td>the number of passes through the FORTRAN DO-loop</td>
</tr>
<tr>
<td>CFT time</td>
<td>seconds used computing the results with CFT version 1.10</td>
</tr>
<tr>
<td>$SCILIB time</td>
<td>seconds used computing the same result with a call to a $SCILIB subroutine</td>
</tr>
<tr>
<td>$SCILIB/CFT</td>
<td>the ration of $SCILIB time to CFT time</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Subroutine Name (Function)</th>
<th>Loop Length</th>
<th>CFT Time (x10^-6 sec)</th>
<th>$SCILIB$ Time (x10^-6 sec)</th>
<th>$SCILIB$/CFT (Ratio)</th>
<th>Clocks Per Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOLR (first order linear recurrence)</td>
<td>1</td>
<td>2.5</td>
<td>3.9</td>
<td>1.58</td>
<td>315.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.1</td>
<td>4.1</td>
<td>1.32</td>
<td>165.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.7</td>
<td>4.3</td>
<td>1.18</td>
<td>87.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4.3</td>
<td>4.6</td>
<td>1.08</td>
<td>61.33</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.8</td>
<td>4.8</td>
<td>1.00</td>
<td>48.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>7.6</td>
<td>5.9</td>
<td>.78</td>
<td>26.33</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>16.</td>
<td>9.3</td>
<td>.58</td>
<td>15.50</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>30.</td>
<td>15.</td>
<td>.50</td>
<td>12.18</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>58.</td>
<td>26.</td>
<td>.45</td>
<td>10.59</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>140.</td>
<td>60.</td>
<td>.42</td>
<td>9.63</td>
</tr>
<tr>
<td>GATHER</td>
<td>1</td>
<td>3.0</td>
<td>4.6</td>
<td>1.57</td>
<td>371.</td>
</tr>
<tr>
<td>A(I)=B(INDEX(I))</td>
<td>2</td>
<td>3.4</td>
<td>4.8</td>
<td>1.43</td>
<td>193.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.8</td>
<td>5.0</td>
<td>1.32</td>
<td>133.67</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4.2</td>
<td>5.1</td>
<td>1.22</td>
<td>102.75</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.6</td>
<td>4.9</td>
<td>1.07</td>
<td>79.2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6.7</td>
<td>5.6</td>
<td>.84</td>
<td>44.7</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>13.</td>
<td>7.5</td>
<td>.58</td>
<td>24.</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>23.</td>
<td>11.</td>
<td>.46</td>
<td>17.10</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>44.</td>
<td>17.</td>
<td>.39</td>
<td>13.65</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>110.</td>
<td>36.</td>
<td>.34</td>
<td>11.58</td>
</tr>
<tr>
<td>ISAMIN (finds the first position of the minimum absolute value of a vector)</td>
<td>1</td>
<td>2.5</td>
<td>6.2</td>
<td>2.48</td>
<td>493.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.4</td>
<td>6.2</td>
<td>1.83</td>
<td>494.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.2</td>
<td>6.2</td>
<td>1.46</td>
<td>247.50</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.1</td>
<td>6.2</td>
<td>1.22</td>
<td>165.33</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.0</td>
<td>6.2</td>
<td>1.04</td>
<td>124.25</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10.</td>
<td>7.1</td>
<td>.69</td>
<td>63.11</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>23.</td>
<td>9.0</td>
<td>.39</td>
<td>30.08</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>45.</td>
<td>11.0</td>
<td>.25</td>
<td>18.04</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>88.</td>
<td>51.</td>
<td>.58</td>
<td>41.55</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>220.</td>
<td>70.</td>
<td>.32</td>
<td>22.33</td>
</tr>
<tr>
<td>Subroutine Name</td>
<td>Loop Length</td>
<td>CPT Time (x10^-6 sec)</td>
<td>$SCILIB$ Time (x10^-6 sec)</td>
<td>$SCILIB$/CPT (Ratio)</td>
<td>Clocks Per Operation</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-----------------------------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>ISRCHEQ (searches a vector for a word match)</td>
<td>1</td>
<td>4.3</td>
<td>5.5</td>
<td>1.27</td>
<td>438.00</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.7</td>
<td>5.5</td>
<td>1.55</td>
<td>219.49</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5.2</td>
<td>5.5</td>
<td>1.06</td>
<td>146.67</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.6</td>
<td>5.5</td>
<td>0.97</td>
<td>110.25</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.1</td>
<td>5.5</td>
<td>0.90</td>
<td>88.40</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>8.3</td>
<td>5.6</td>
<td>.67</td>
<td>44.70</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>15.</td>
<td>5.8</td>
<td>.38</td>
<td>18.48</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>28.</td>
<td>6.1</td>
<td>.21</td>
<td>9.74</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50.</td>
<td>7.0</td>
<td>.14</td>
<td>5.59</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>110.</td>
<td>9.1</td>
<td>.07</td>
<td>2.90</td>
</tr>
<tr>
<td>MXM M (full matrix multiply)</td>
<td>1</td>
<td>6.6</td>
<td>5.5</td>
<td>.84</td>
<td>443.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24.</td>
<td>6.5</td>
<td>.46</td>
<td>57.44</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>25.</td>
<td>8.0</td>
<td>.32</td>
<td>18.37</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>40.</td>
<td>9.2</td>
<td>.23</td>
<td>8.07</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>59.</td>
<td>12.</td>
<td>.20</td>
<td>5.04</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>220.</td>
<td>30.</td>
<td>.13</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>1700.</td>
<td>260.</td>
<td>.16</td>
<td>.71</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>8900.</td>
<td>1800.</td>
<td>.20</td>
<td>.59</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>59000.</td>
<td>14000.</td>
<td>.24</td>
<td>.57</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>770000.</td>
<td>210000.</td>
<td>.27</td>
<td>.54</td>
</tr>
<tr>
<td>MXM V (matrix times a vector)</td>
<td>1</td>
<td>5.8</td>
<td>5.6</td>
<td>.97</td>
<td>447.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7.6</td>
<td>6.4</td>
<td>.85</td>
<td>102.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9.3</td>
<td>6.5</td>
<td>.70</td>
<td>40.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>11.0</td>
<td>7.0</td>
<td>.63</td>
<td>22.44</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>13.0</td>
<td>7.3</td>
<td>.56</td>
<td>12.4</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>23.0</td>
<td>9.8</td>
<td>.42</td>
<td>4.35</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>66.</td>
<td>24.</td>
<td>.37</td>
<td>1.62</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>180.</td>
<td>76.</td>
<td>.43</td>
<td>1.24</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>580.</td>
<td>280.</td>
<td>.49</td>
<td>1.14</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>3100.</td>
<td>1700.</td>
<td>.55</td>
<td>1.09</td>
</tr>
<tr>
<td>SAXPY (add a scalar multiple of one vector to another)</td>
<td>1</td>
<td>3.3</td>
<td>5.3</td>
<td>1.60</td>
<td>212.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.3</td>
<td>5.3</td>
<td>1.60</td>
<td>106.5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.3</td>
<td>5.3</td>
<td>1.59</td>
<td>71.17</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.4</td>
<td>5.3</td>
<td>1.59</td>
<td>53.50</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.4</td>
<td>5.4</td>
<td>1.59</td>
<td>42.90</td>
</tr>
<tr>
<td>Subroutine Name (Function)</td>
<td>Loop Length</td>
<td>CFT Time (x10^-6 sec)</td>
<td>$SCILIB$ Time (x10^-6 sec)</td>
<td>$SCILIB$/CFT (Ratio)</td>
<td>Clocks Per Operation</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
<td>------------------------</td>
<td>----------------------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>SAXPY (continued)</td>
<td>10</td>
<td>3.5</td>
<td>5.5</td>
<td>1.58</td>
<td>21.95</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>4.0</td>
<td>6.0</td>
<td>1.50</td>
<td>9.66</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>5.0</td>
<td>7.0</td>
<td>1.41</td>
<td>5.58</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>7.3</td>
<td>9.3</td>
<td>1.28</td>
<td>3.72</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>14.0</td>
<td>16.0</td>
<td>1.15</td>
<td>2.52</td>
</tr>
<tr>
<td>SCATTER</td>
<td>1</td>
<td>2.7</td>
<td>6.0</td>
<td>2.24</td>
<td>482.00</td>
</tr>
<tr>
<td>A(INDEX(I))=B(I)</td>
<td>2</td>
<td>3.0</td>
<td>6.0</td>
<td>2.00</td>
<td>241.5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.0</td>
<td>6.1</td>
<td>1.81</td>
<td>161.33</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.7</td>
<td>6.1</td>
<td>1.66</td>
<td>121.25</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.0</td>
<td>6.1</td>
<td>1.52</td>
<td>97.20</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>5.6</td>
<td>6.9</td>
<td>1.23</td>
<td>55.20</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>10.0</td>
<td>8.8</td>
<td>.83</td>
<td>28.00</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>19.0</td>
<td>12.0</td>
<td>.63</td>
<td>18.66</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>35.0</td>
<td>19.0</td>
<td>.53</td>
<td>14.90</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>84.0</td>
<td>39.0</td>
<td>.46</td>
<td>12.42</td>
</tr>
<tr>
<td>SDOT</td>
<td>1</td>
<td>2.9</td>
<td>5.5</td>
<td>1.89</td>
<td>439.33</td>
</tr>
<tr>
<td>(computes the dot product of 2 real vectors)</td>
<td>2</td>
<td>7.1</td>
<td>5.5</td>
<td>.77</td>
<td>146.33</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.1</td>
<td>5.5</td>
<td>.77</td>
<td>87.80</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.2</td>
<td>5.5</td>
<td>.77</td>
<td>62.71</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.2</td>
<td>5.5</td>
<td>.76</td>
<td>48.78</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>7.3</td>
<td>5.6</td>
<td>.76</td>
<td>23.63</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>8.2</td>
<td>6.0</td>
<td>.73</td>
<td>9.78</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>9.8</td>
<td>6.6</td>
<td>.68</td>
<td>5.34</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>13.0</td>
<td>7.8</td>
<td>.59</td>
<td>3.15</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>23.0</td>
<td>12.0</td>
<td>.51</td>
<td>1.93</td>
</tr>
<tr>
<td>SSUM</td>
<td>1</td>
<td>2.6</td>
<td>3.9</td>
<td>1.52</td>
<td>314.63</td>
</tr>
<tr>
<td>(sums the elements of a real vector)</td>
<td>2</td>
<td>6.6</td>
<td>4.1</td>
<td>.62</td>
<td>326.90</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.6</td>
<td>4.2</td>
<td>.64</td>
<td>169.33</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6.6</td>
<td>4.4</td>
<td>.66</td>
<td>116.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.6</td>
<td>4.5</td>
<td>.68</td>
<td>90.50</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6.8</td>
<td>4.8</td>
<td>.71</td>
<td>42.89</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>7.2</td>
<td>5.0</td>
<td>.69</td>
<td>16.71</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>8.2</td>
<td>5.3</td>
<td>.65</td>
<td>8.69</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>10.0</td>
<td>7.0</td>
<td>.67</td>
<td>5.66</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>17.0</td>
<td>9.1</td>
<td>.55</td>
<td>2.92</td>
</tr>
</tbody>
</table>
Table B-3. $\text{SCILIB}$ timings and MFLOP rates

<table>
<thead>
<tr>
<th>Subroutine Name (Function)</th>
<th>Dimension of matrix</th>
<th>Execution time (seconds)</th>
<th>MFLOP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELMHES</td>
<td>5</td>
<td>.000057</td>
<td>3.63</td>
</tr>
<tr>
<td>(Reduction of full matrix</td>
<td>10</td>
<td>.00026</td>
<td>6.32</td>
</tr>
<tr>
<td>to upper Hessenberg form)</td>
<td>25</td>
<td>.0020</td>
<td>13.30</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.010</td>
<td>20.23</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.064</td>
<td>25.83</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>.83</td>
<td>31.42</td>
</tr>
<tr>
<td>HQR</td>
<td>5</td>
<td>.00053</td>
<td>2.25</td>
</tr>
<tr>
<td>(Reduction form upper Hessenberg form to upper triangular)</td>
<td>10</td>
<td>.0022</td>
<td>4.70</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>.011</td>
<td>11.41</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.043</td>
<td>21.05</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.18</td>
<td>36.02</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>1.4</td>
<td>62.62</td>
</tr>
<tr>
<td>SGEFA</td>
<td>5</td>
<td>.00062</td>
<td>1.35</td>
</tr>
<tr>
<td>(L-U decomposition)</td>
<td>10</td>
<td>.00019</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>.0011</td>
<td>9.62</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.0048</td>
<td>17.41</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.026</td>
<td>25.51</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>.30</td>
<td>35.00</td>
</tr>
<tr>
<td>SGESL</td>
<td>5</td>
<td>.00049</td>
<td>1.73</td>
</tr>
<tr>
<td>(Backward and forward solving)</td>
<td>10</td>
<td>.000056</td>
<td>3.59</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>.00015</td>
<td>8.58</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.00033</td>
<td>15.01</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.0089</td>
<td>22.01</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>.039</td>
<td>32.58</td>
</tr>
</tbody>
</table>
SORT ENTRY POINTS

SORT consists of library subroutines accessed through FORTRAN calls. The user-callable entry points are as follows:

- SAMSORT
- SAMFILE
- SAMKEY
- SAMGO
- SAMEQU
- 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.

<table>
<thead>
<tr>
<th>Module PSRVSIO entry points</th>
<th>Module IOXOXOI entry points</th>
</tr>
</thead>
<tbody>
<tr>
<td>V8PUT</td>
<td>P87DNT</td>
</tr>
<tr>
<td>V8GET</td>
<td>P87DOD</td>
</tr>
<tr>
<td>V8WAIT</td>
<td>P87CLS</td>
</tr>
<tr>
<td>V8POS</td>
<td>P87DLT</td>
</tr>
<tr>
<td>V8CLOSE</td>
<td>P87WDC</td>
</tr>
<tr>
<td>V8OPEN</td>
<td>P87RDC</td>
</tr>
<tr>
<td>V8STATS</td>
<td>P87LBN</td>
</tr>
<tr>
<td>V8LINTHDS</td>
<td>P87OPN</td>
</tr>
<tr>
<td></td>
<td>P87WAIT</td>
</tr>
<tr>
<td></td>
<td>P87SNSE</td>
</tr>
</tbody>
</table>

For more details, see the SORT Reference Manual, CRI publication SR-0074.
INDEX

Abort current job step, 8-2
Abort with traceback, 2-4
ABORT, 2-4, 7-36
ABS, 2-4, 3-29
Absolute program, 2-48
Absolute value
minimum, 4-26
sum, 4-3, 4-5
ACCESS, 2-4, 6-2
ACOS, 2-4, 3-8, A-1
ACPTBAD, 2-4, 5-76
ACQUIRE, 2-5, 6-3
ACTTABLE, 2-5, 7-70
Addition, double-precision, 2-22
ADDLFT, 2-6, 6-5
ADJUST, 2-5, 6-2
ADW, 7-16
AIMAG, 2-5, 3-29
AINT, 2-5, 3-29
ALF, 6-5
Algorithms, A-1
ALLOC, 2-37, 7-26
Allocate routines, 7-25
ALOG, 2-49, 3-5, A-2
ALOG10, 2-49, 3-5
AMOD, 2-6, 3-29
AMU, 2-6, 7-19
AND, 2-6, 3-13
ANINT, 2-6, 3-29
APHN, 2-60, 5-53
APUTWA, 5-105
Arccosine, 2-4, 3-8, A-1, A-9
Arcsine, 2-7, 3-8, A-2
Arctangent
double-precision, 2-18, A-10
double-precision 2-argument, 2-19
two-argument, A-3, 2-8
one-argument, 2-7, 3-8, 3-9, A-2
ARERP, 2-6, 7-60
ARGPIMQ, 2-33, 2-7, 7-3
Argument
as substring, 2-12
length, 2-12
first word address, 2-48
Argument values, 2-7
Arithmetic routines
double-precision, 3-17
triple-precision, 3-18
triple-precision, 3-19
triple-precision, 3-20
Array
bounds checking, 2-48, 7-12
complex, 2-8, 2-70, 2-71, 2-80, 2-81
copy, 4-3
Euclidean norm of, 4-3
indexed, 4-64
interchange, 4-22
largest absolute value in, 2-45
maximum of, 2-46
minimum of, 2-46
ordered, 2-71
real, 2-8, 2-70, 2-80, 2-81
scale, 4-4
scaling, 2-70
search for target, 2-71, 4-71
swap, 4-22
swap two, 4-4
task control, 2-86
true values in, 2-42
with integer target, 4-60, 4-65
with real target, 4-59, 4-65
with true relational value to
  target, 2-92
zero values in, 4-58
ASCDC, 2-7, 5-33
ASCII conversion, 3-25, 3-26
ASIN, 2-7, 3-8
ASPOS, 2-74, 5-46
ASSIGN, 2-7, 6-3
ASYNCMS/ASYNCDR, 5-96
ATAN, 2-7, 3-8, A-2
ATAN2, 2-8, 3-8, A-3
ATS, 2-8, 7-15
B2OCT, 2-30, 7-10
BACK, 2-8
BACKFILE, 2-8, 5-52
Backspace
record, 2-8
file, 2-8
BACKSPACE, 5-51
Bad data, 2-4, 5-75
Basic Linear Algebra Subprograms (BLAS)
description, 4-1
list, 4-3
Bibliography, A-20
BICONV, 2-9, 7-68
BICONZ, 2-9, 7-68
Bidirectional memory transfer, 2-12 and
  2-13, 2-72, 73, 7-40
Binary image, 2-48
Bit setting, 2-42
Bits
move, 2-53

tally number set, 2-61
parity of number set, 2-61
number set to 1, 3-14

BKSP, 2-8, 5-50
BKSPF, 2-8, 5-52
BLAS, see Basic Linear Algebra Subprograms
Boolean arithmetic routines, 3-13, 14

Breakpoint checking (Pascal), 2-96

BKSP,
2-8,
5-50

BKSPF,
2-8,
5-52

BLAS,
see Basic Linear Algebra
Subprograms

Boolean arithmetic routines, 3-13, 14

Number set to 1, 3-14

Bucket sort, 4-75

BUFFER
I/O, 2-47, 2-87

Buffer management
input, 5-16
output, 5-17

Buffered output, 2-91

Byte and bit manipulation, 7-63
Byte storage, 2-61
Bytes
fetches, 2-41
compares, 2-47
move, 2-80

CABS, 2-10, 3-30, A-3
CAL I/O interface, 5-75
Call-by-value routine (Pascal), 2-96
Calling routine (Pascal), 2-96
Calling task, 2-20
CAXPY, 2-8, 4-3, 4-7
CBIO, 5-75
CCI201, 5-19
CCI200, 5-20
CCP, 2-11, 3-25
CHCNO, 5-20
CCI, 2-11, 3-24
CCOPY, 2-14, 4-3, 4-8, 2-10
CCOS, 3-9, A-4
CSS, 2-17, 7-46
CCT, 2-11, 3-24
CD1201, 5-19
CD1200, 5-20
CDCI, 2-11, 5-18
CDO, 2-11, 5-18
CDXC, 4-4, 4-9
CDXCY, CDTU, 2-21
CDTOU, 4-10, 4-4

Ceiling of rational number, 2-41

CEXP, 2-31, 3-6, A-4
CEXPR, 2-11, 7-54
CF60I, 5-19
CF600, 5-20
CFPT2, 2-31, 4-51
CPT call type, 2-3
CPT in-line functions, 3-13
CPT linkage macros, 1-2
CPT linkage methods, 1-2
Change length and move routines, 7-29

CHAR, 3-30
Character address
increment, 5-78
Character functions
called from CAL, 3-23
called from FORTRAN, 3-22
Character routines, 5-77
Character string
searches for occurrence of, 2-32
CHOCONV, 2-12
CHOCONV, 7-67
CHECKMS/CHKDR, 5-97
Cl601, 5-19
Cl600, 5-20
Circular shift, 3-14
Cl601, 5-19
Cl600, 5-20
CLEAREBS, 2-12, 7-41
CLEARBTS, 2-13, 7-42
CLEARPI, 2-13, 7-39
CLEARPIS, 2-13, 7-40
CLMOVE, 2-37, 7-29
CLOCK, 2-13, 7-42
CLOG, 2-49, 3-5, A-4
CLOSE, 2-13, 5-34, 6-8
CLOSEWNA, 5-110
CLOSEMS, 2-53
CLOSEMS/ClOSDR, 5-91, 5-92
CMACH, 2-77, 4-30
CMPLX, 2-14, 3-30
Common block TM, 7-14
Common logarithm, 3-5
Common mathematical subprograms, 2-97
Compare
ASCII, 3-22 and 3-23
Compares
for greater than, 2-11
for less than, 2-12
for less than or equal to, 2-12
for nonequality, 2-12
COMPL, 2-14, 3-13
Complex
absolute value, A-3, 3-30
conjugate, 3-30
cosine, A-4, 3-9
exponential, A-4
exponentiation, 3-6
imaginary portion of, 3-29
logarithm, A-4
real portion of, 3-34, 4-7
sine, 3-9, A-7
square root, 3-7, A-8
vector, 4-6, 4-8
Complex absolute value, 2-10
Complex array
copies, 2-14
Complex conjugate, 2-14
Complex sine, 2-17
Complex cosine, 2-10
Complex logarithm, 3-5
Complex number
imaginary part, 2-5
real part of, 2-65
Complex sine, 2-17
Compute absolute value, 2-4
Concatenation
initialize for store, 3-24
CONJG, 2-14, 3-30
Contangent, d.p., A-11
Control information tables, 7-12
Control statement
acks, 2-17
Control statement processing, 7-46
Conventions in manual, 1-3
Conversion
ASCII, 3-26
character to integer, 3-32
Cray/other vendors, 5-20
integer to character, 3-30
return conditions, 5-81
Convert
ASCII/display code, 5-33
ASCII/EBCDIC, 5-27
ASCII to EBCDIC, 2-88
ASCII to integer, 7-67
between timestamps and date and
time, 2-84, 7-44
binary to decimal, 3-26
binary to octal, 2-9
Cray/CDC integer to IBM integer, 2-89
Cray/CDC single-precision, 5-31
Cray/CDC integer, 5-31
Cray/CDC single-precision, 5-31
Cray/IBM integer, 5-26
Cray/IBM logical, 5-28
Cray/IBM single precision, 5-24
Cray integer/IBM decimal, 5-28
Cray single-precision to IBM
double-precision, 2-88, 5-25
Cray to CDC, 2-32, 2-44
date and time to timestamp, 2-84
decimal to binary, 3-26
display code/ASCII character, 2-7
display code to ASCII character, 2-32
EBCDIC/ASCII, 5-22
from date and time to timestamp, 7-44
from real-time clock value to
timestamp, 7-45
from timestamp to real-time clock
value, 7-45
IBM/Cray integer, 5-21
IBM/Cray logical, 5-23
IBM integer to Cray integer, 2-88
IBM packed decimal integer, 5-23
integer to ASCII, 7-68
integer to binary, 2-24
integer to IBM packed decimal, 2-88
integer to real, 2-33, 3-32
nearest integer to double-precision,
3-32
numeric input, 2-56
numeric output, 2-57
octal to binary, 2-59, 3-26
real to double-precision, 2-19, 3-30
real-time to corresponding timestamp
value, 2-84
timestamp to corresponding real-time
clock value, 2-84
Convert (continued)
trailing blanks to nulls, 7-85
trailing nulls to blanks, 7-85
two reals to a complex, 3-30
Converts
ASCII character to display code
character, 2-7
ASCII to integer, 2-12
binary to decimal, 2-9
binary to octal, 2-9
Cray integer to IBM integer, 2-89
Cray logical to IBM logical, 2-89
Cray single-precision to IBM
double-precision, 2-90
EBCDIC to ASCII, 2-87
IBM logical to Cray logical, 2-89
IBM packed decimal field to
integer, 2-89
IBM single-precision to Cray
single-precision, 2-89
integer to ASCII, 2-9
IBM/Cray single-precision, 5-20
IBM/Cray single-precision, 5-21
Copy
vector, 4-8
COPYD, 5-39
COPYF, 2-15, 5-37
COPYR, 2-15, 5-35
COPYU, 2-15, 5-39
COS, 2-15, 3-9, A-5
COSH, 2-15, 3-11, A-5
Cosine
double-precision, 2-19
d.p., hyperbolic, A-11
Cosine and sine, 3-10
Cosine, 2-15, 2-16, 3-9, A-5
Cosine, complex, A-4
Cosine, d.p., A-11
COSSS, 2-16, 3-10, A-5
COSHH, 2-16, 3-11
COT, 2-16, 3-10, A-7
Cotangent, 2-16, 3-10, A-7
CRACK, 2-16, 7-52
CRAY channel
program on an IOS, 2-23
Cray machine constants, 4-29
CRAYDUMP, 2-26, 7-8
CRFFT2, 2-31, 4-53
CROT, 2-67, 4-28
CROTG, 2-67, 4-28
CS, 2-17, 7-46
CSCL, 2-70, 4-22, 4-4
CSIN, 2-17, 3-9, A-7
CSQRT, 2-78, 3-7, A-8
CSSCAL, 2-70, 4-22, 4-4
CSSUM, 2-80, 4-27
CSWAP, 2-81, 4-23, 4-4
CTOC, 2-17, 3-15, A-8
CTOI, 2-17, 3-15, A-9
CTOR, 2-18, 3-15, A-9
Current date, 2-19, 8-10, 8-14, 8-4
Current time, 8-23
Table of Contents

Dataset (continued)
  terminates connection of, 2-13
termination, 5-55
  unblocked, 2-76, 7-6
  word-addressable, 5-109 and 5-110
  write, 2-94

Dataset Catalog
  removes saved dataset from, 2-20

Dataset catalog, 6-2

Dataset control, 5-34

Dataset management, 6-1

Dataset Parameter Table (DSP)
  search, 6-4
  search for address, 6-6

Dataset parameter tables
  search, 2-71

Dataset position, 2-36

Datasets
  system, 2-71
  FORTRAN access, 2-71
  word-addressable, 5-111

Date (Pascal)
  obtain, 2-96

Date and time, 7-42

DATE, 2-19, 7-42

DAVS, 2-22, 3-17

DAVV, 2-22, 3-17

DBLE, 2-19, 3-30

DCOS, 2-19, 3-9, A-10

DCOSH, 3-11, A-11

DCO'1', 3-10, A-11

DDIM, 2-20, 3-30

DDSS, 2-22, 3-17

DDSV, 2-22, 3-17

DDVS, 3-17

DDVV, 2-22, 3-17

DEADBOG, 2-81, 7-7

DEALLC, 2-37, 7-27

Deallocate routines, 7-26

Debug aid, 7-1

Decode
  formatted, 5-3, 5-7, 5-14

Decoding routines, 2-20

Definition and control, 6-3

DELETE, 2-20, 6-2

DELETES, 2-20, 6-2

DELSK, 2-20, 9-13

DEV, 2-20

DEXP, 2-31, 3-6, A-11

DFA, 2-20, 5-7

DFF, 2-20, 5-14

DFI, 5-3

DFV, 5-7

Difference
  positive integer, 2-41

DIM, 2-20, 3-31

DINT, 2-21, 3-31

Display code conversion, 2-24

DISPOSE, 2-21, 6-3

Divide, 8-5

Division
  compute remainder, 2-6
  double-precision, 2-22
  integer, 2-96
  64-bit, 2-52
  64-four-bit integer, 3-21

Dataset accessed, 6-6
assigns characteristics to, 2-7
asynchronously positions, 2-60
attributes, 6-3
buffers, 2-90
bypass records in, 2-76
bypass files in, 2-76
characteristics, 2-53
change records of, 2-53
CLOSE, 5-34
close, 2-13, 5-110, 6-3, 6-8
closing, 5-92
connect, 2-58
copy, 5-35
copy, 5-39
current size, 6-7
definitions table, 6-9
direct to specified queue, 2-21
direct-to-file number, 2-55
foreign, 2-69, 2-94
front-end resident, 2-5
interactive, 8-12
load absolute program from, 7-78
local to job, 2-41
mass storage, 5-42, 5-46, 5-48
OPEN, 5-34
open, 2-58, 2-90, 2-7, 2-96, 6-3, 6-7, 8-11
open local, 5-84
permanent, 2-82, 6-2
places into input queue, 2-80
position, 2-61, 5-53
positioning, 2-69, 2-74, 2-76, 5-42, 5-47
query interactive, 2-96
random access, 2-90, 5-89, 5-90, 5-91, 5-103
random, unblocked, 6-7, 2-63, 6-8
read, 2-69
release, 2-65
rewind, 2-96, 5-52, 8-10, 2-66
saved, 6-2
search, 6-4
size, 2-57
skip, 5-40, 5-41
staging, 6-2
status, 2-43, 5-35
synchronize, 5-55
synchronously positions, 2-60
tape, 2-81, 5-44, 5-45, 5-47

SR-0014

Index-4
Division remainder, 2-21
DLOG, 2-50, 3-5, A-12
DLOG10, 3-5
DMOD, 2-21, 2-51, A-13
DMSS, 2-22, 3-17
DMSV, 2-23, 3-17
DMVV, 2-23, 3-31
Dot product, 2-21, 2-22
Dot product, 4-4
Double-precision
absolute value, 2-18, 3-30
addition, 2-22, 3-17
arcosine, 3-8, A-9
arcsine, 3-8, A-10
arctangent, 2-18, 2-19, 3-9, A-10
arithmetic routines, 3-17
common logarithm, 3-5
conversion, 2-68
cosine, 2-19, 3-9, A-11
cotangent, 3-10, A-11
division, 2-22, 3-17
exponential, 2-31
exponentiation, 3-6, A-12
hyperbolic cosine, 3-11, A-11
hyperbolic sine, 3-11, A-13
hyperbolic tangent, 3-12, A-14
logarithm, 2-50
modulo arithmetic, 2-51, 3-31, A-13
multiplication, 2-22, 2-23, 3-17
natural logarithm, 3-5, A-12
nearest integer to a, 3-32
negative powers of 10, 2-43
positive powers of 10, 2-44
positive real difference, 2-20, 3-30
product, 2-23
product of two real arguments, 3-31
raise power, 2-25, 2-26
sine, 2-24, 3-10
square root, 3-7, A-13
subtraction, 2-22, 2-23, 3-17
tangent, 3-10, A-14
transfer sign, 3-32
truncate, 2-21, 3-31
DPROD, 2-23
DRIVER, 2-230 7-70
DSASC, 2-24, 5-32
DSIGN, 2-24, 3-32
DSIN, 2-24, 3-10
DSINH, 3-11, A-13
DSNDSP, 2-24, 6-5
DSP
set end-of-file flag, 2-87
DSQRT, 2-78, 3-7
DSSS, 2-22, 3-17
DSSV, 2-23, 3-17
DSSV, 2-23, 3-17
DSSV, 2-23, 3-17
DTAN, 3-10, A-14
DTANH, 3-12, A-14
DTB, 2-24, 3-26
DTOD, 2-25, 3-15, A-14
DTOI, 2-25, 3-16, A-14
DTOR, 2-26, 3-16
DTTS, 2-84, 7-44
Dump heap control word routines, 7-33
Dump routines, 7-5
DUMP, 2-26, 7-6, 8-4
DUMPJOB, 2-27, 7-6
ECHO, 2-27, 7-71
EFA encode
formatted, 5-11
EFA, 2-27
EFF, 2-27, 5-15
EFI, 2-27, 5-5
EFV, 2-27, 5-12
Eigenvalue problem, 4-38
EISPACK routines
descriptions, 4-38
summary, 4-42
EISPACK, 2-27
Encode formatted, 5-5, 5-12, 5-15
End
program, 2-96
day of file check, 2-96
day of line check, 2-96
End of file mark, 8-15
End of file mark, 8-25
END, 2-27, 7-36
End-of-file status, Pascal, 8-6
End-of-line status, Pascal, 8-6
ENDFILE, 5-56
ENDRVP, 2-28, 2-65, 7-36
Entry type, 2-2
E0TEST, 2-28, 5-58
EOD, 2-91
EODW, 2-28, 5-56
EOF, 2-29, 2-91, 5-58
EDFW, 2-28, 5-56
EQ, 2-29, 2-8
EOR, 2-91
EQ, 2-12, 3-23
Equations Weiner-Levinson, 2-32
EQV, 2-29, 3-13
ERECALL, 7-72
ERREXIT, 2-29, 7-36
Error processing, 2-6
Error processing routines, 7-60
Error recovery, 5-75
Errors
1/0, 2-44
NAMELIST, 2-44
processes $FPTLIB, 2-33
process $SCLIB, 2-70
process $UTLIB, 2-90
processes $SYSLIB, 2-76
Euclidean norm, 2-71, 2-77, 4-10, 4-3
EVASGN, 2-29, 9-9
EVCCLEAR, 2-29, 9-10
Event processing, 2-29, 2-30
Event routines, 9-9
EVPOST, 2-29, 9-9
EVREL, 2-30, 9-10
EVTEST, 2-30, 9-10
EVWAIT, 2-30, 9-9
Exchange package
printout, 7-11
write to output dataset, 7-9
Exchange Package processing, 2-30, 7-8
Exit program, 2-96
EXIT, 2-30, 7-36
EXP, 2-31, 3-6, A-14
Explicit data conversion, 5-20
Exponential
complex, 2-31, A-4
double-precision, 2-31
Exponential routines, 3-6
Exponentiation, complex, A-8, A-9
Exponentiation, d.p., A-11
Fast Fourier Transform, 2-31, 4-50
FILEBACK, 2-69, 5-51
FDGPOS, 2-69, 2-69, 5-44
FDPOS, 2-69, 5-48
FNDPOS, 2-94
FETCH, 2-31, 6-3
FFS, 5-78
File
backspace, 5-52
reset, 2-97, 8-19
rewrite, 2-97
rewrite without rewind, 2-96
File name set, 2-96
Files
copies, 2-15
copy, 5-37
skip, 5-40
Filter analysis and design, 4-53
Filter coefficients, 2-32
FILTER, 2-32, 4-54
FILTERS, 2-32, 4-54
Finalization
input, 5-14
output, 5-14
FINALIZE, 2-32, 7-66
PINDUM, 2-53, 5-94
FLOAT, 2-33, 5-32
Floating-point interrupt
permanent, 7-39
Floating-point interrupt
temporary, 7-39
Floating-point interrupt
test routine, 7-39
Floating-point interrupts, 2-72, 2-73
Floating-point seconds, 2-85
Flow trace, 7-2
FLOWENTRY, 2-33, 7-2
FLOWEXIT, 2-33, 7-2
FLOWLIMIT, 2-33, 7-4
FLOWSTOP, 2-33, 7-2
Flowtrace processing, 2-33, 2-73
FOLR, 2-34, 4-30
FOLR2, 2-34, 4-32
FOLR2P, 2-34, 4-32
FOLRN, 2-34, 4-33
FOLRF, 2-34, 4-31
Format search, 5-78
Formatted output, 2-91
FORTRAN I/O
description, 5-1
finalization, 5-14
initialization, 5-3
summary, 5-2
transfer, 5-5
FP6064, 2-32, 5-31
FP6460, 2-32, 5-33
FPPOS, 2-74, 5-47
FTERP, 2-33, 7-61
FPX, 2-30, 7-10, 7-9
GATHER, 2-35, 4-56
Gauss-Jordan elimination, 4-46
Gauss-Jordan reduction, 2-51
GE, 2-11, 3-23
GETBI, 2-35, 7-74
GETDSP, 2-35
GETLP, 2-35, 7-74
GETNAMEQ, 2-35, 7-2
GETPARAM, 2-36, 7-49
GETPOS, 2-36, 2-74, 5-45
GETREGS, 7-3
GETDSP, 7-6
GETMA, 2-90, 5-106
Givens plane rotation, 2-67, 4-11, 4-13,
4-19, 4-28, 4-4
GP, 2-36
GPARAM, 2-36, 7-50
GPOS, 2-36, 2-74, 5-42
GT, 2-11, 3-23
GTDSP, 6-6
GTPOS, 2-36, 5-44
Hardware performance monitor, 2-60, 7-82
HCHECK, 2-37, 7-32
Heap block length routines, 7-30
Heap control block, 8-1
Heap expansion routine, 7-34
Heap integrity check routines, 7-31
Heap manager, 7-25
Heap memory request routine, 7-34
Heap merge routine, 7-35
Heap processing (Pascal), 2-96
Heap processing, 2-37, 8-8
Heap shrink routines, 7-31
Heap statistics routines, 7-32
HMERGE, 2-37, 7-35
HPALLOC, 2-37, 7-26
HPCHECK, 2-37, 7-32
HPCLMOVE, 2-37, 7-30
HPDELETE, 2-37
HPDUMP, 2-37, 7-33
HPEGROW, 2-37, 7-34
HPLEN, 2-37, 7-30
HPMEM, 2-37, 7-34
HPNENLEN, 2-37, 7-28
HPSHRINK, 2-37, 7-31
HPSTAT, 2-37, 7-32
Hyperbolic cosine and sine, 2-16, 3-11
Hyperbolic cosine, 3-11, A-5
Hyperbolic routines, 3-11, 3-12
Hyperbolic sine, 2-75, 3-11
Hyperbolic tangent, 2-83, 3-11, A-19
Hyperbolic tangent, d.p., A-14

I/O mode, 5-96
I/O status, 5-57
IODEL, 2-38, 7-57
IODEL, 2-38, 7-60
IOMVC, 2-38, 7-57
IOMVC, 2-38, 7-57
IORD, 2-39, 7-58
IOREAD, 2-39, 7-58
IODESETUP, 2-39, 7-60
IOWRITE, 2-40, 7-59
IAAB, 2-40, 3-32
IBMI, 2-40, 5-18
IBMIO, 2-40, 5-18
IC641, 5-19
ID640, 5-19, 5-20
ICAMAX, 2-40, 4-3, 4-5
ICEIL, 2-41, 7-74
ICHAR, 3-32
ICHRI, 5-19
ICHRO, 5-19
IDIM, 2-41, 3-32
IDMINT, 2-41, 3-33
IEOF, 2-29, 5-57
IF321, 5-19
IF320, 5-19
IFDNT, 2-41, 6-6
IGSBYT, 2-41, 7-74
IHPLEN, 2-37, 7-31
IHPSSTAT, 2-37, 7-33
III61, 5-18
III6G, 5-19
II321 translate
IBM integer to Cray integer, 5-18
II320, 5-19
IIIZ, 2-42, 4-58
IIIN, 2-42, 2-82, 3-38
IJC0M, 7-75
II81, 5-19
II80, 5-19
IIIZ, 2-42, 4-58
IIJSUM, 2-42, 4-58
ILXX, 2-43, 2-82, 3-38
INDEX, 2-12, 3-22
Initialization
input, 5-3, 5-4
Inline code, 1-1
Input
buffered, 2-62
Input transfer
buffered, 5-8
formatted and unformatted, 5-6
namelist, 5-8
Input/output subprograms, 5-1
INQ, 2-43
INQUIRE, 2-43, 5-35
INSASCI, 2-43, 7-76
INT, 2-43, 3-33, 7-15
INT6064, 2-44, 5-31, 5-32

Integer absolute value, 2-40, 3-32
Integer value
truncate to, 2-43
Interset subroutine calls, 2-72
Integral value
truncate to, 2-5
IORDER, 2-44, 7-61
IOSTAT, 2-44, 5-56
IPX, 2-44, 2-82, 3-38
ISAMAX, 2-45, 4-3, 4-4
ISAMIN, 2-45, 4-26
ISEARCH, 2-45
ISIGN, 2-45, 3-33
ISMAX, 2-46, 4-25
ISMIN, 2-46, 4-26
ISRCCH routines, summary, 4-59
ISRCHEQ, 2-45, 4-60
ISRCHPGE, 2-45, 4-62
ISRCHPGT, 2-45, 4-62
ISRCHPLT, 2-45, 4-61
ISRCHEGE, 2-46, 4-64
ISRCHEGT, 2-46, 4-64
ISRCHEL, 2-46, 4-63
ISRCHELT, 2-46, 4-63
ISRCHENE, 2-45, 4-61
ITOI, 2-46, 3-16, A-15

JCLPP
returns lines from, 2-35
JDATE, 2-19, 7-43
JNAME, 2-47, 7-78
Job Accounting Table (JAT), 2-5
Job area image, 2-27
Job control language symbol routines, 7-55
Job control routine, 7-35
Job CPU time, 2-96
Job name, 2-47
Job step
abort, 2-29
terminate and advance, 2-27
terminates, 2-30
Job step (Pascal)
abort, 2-96
JSYMGET, 7-56
JSYMSET, 7-55
Julian date, 2-19

Keyword processing, 2-60
KOMSTR, 2-47, 7-66

LCI, 5-77
LDSS, 2-52, 3-21
LDSS, 2-52, 3-21
LDSV, 2-52, 3-21
LE, 2-12, 3-23
Leading zero bits, 3-13
LEADZ, 2-47, 3-13
Left shift, 3-14
LEN, 2-12, 3-22
LENGTH, 2-47, 5-57
usi

RESLEN, 2-37, 7-28
NICONV, 2-56, 5-79
NICV, 2-56, 5-79, 5-82
NINT, 2-57, 3-34
NLERP, 2-44, 7-61
NOCONV, 2-57, 5-80, 5-81
NOCV, 2-57, 5-80, 5-82
NORERUN, 2-57, 7-37
NUMBLKS, 2-57, 6-7
Numeric conversion, 5-78

Online translation, v
OPEN, 2-58, 5-34, 6-7
OPENMS, 2-53
OPENMS/OPENDR, 5-84
OPENWA, 5-109
OPFILT, 2-32, 4-55
OPTION, 7-82
OR, 2-58, 3-13
O~E~ description, 4-73
examples, 4-77
method, 4-75
sort times, 4-76
os dependency, 2-3
OSRCHF, 2-71, 4-72
OSReHI, 2-71, 4-71
OTB, 2-59, 3-26
Output transfer
buffered, 5-13
formatted and unformatted, 5-11
name list, 5-13
OVERLAY, 2-59, 7-82

P$HPAD, 2-96, 8-1
PSABORT, 2-96, 8-2
PBREAK, 8-2
PCALLR, 2-96, 8-2
P$CBV, 2-96, 8-3
P$CONNEC, 2-96, 8-3
PSDATE, 2-96, 8-4
PSDBF, 2-96, 8-4
PSDEBUG, 2-97, 8-4
PSDISP, 2-96, 8-4
PSDIVMOD, 2-96, 8-5
P$ENDP, 2-96, 8-5
P$EOF, 2-96, 8-6
P$EOLN, 2-96, 8-6
P$GET, 2-96, 8-6
P$HALT, 2-96, 8-7
PJLTIME, 2-96, 8-7
PSLOMMSG, 2-96, 8-8
PSLISTREW, 2-96, 8-8
P$MEMRY, 2-96, 8-9
P$MOD, 2-96, 8-9
P$NEW, 2-96, 8-9
P$OSDDB, 2-96, 8-10
P$OSDDT, 2-96, 8-10
P$OSDEP, 2-96, 8-11
P$OSDFT, 2-96, 8-11
P$OSDLRM, 2-96, 8-11
P$OSDRF, 2-96, 8-12

Pack
32-bit words, 2-59, 5-30
60-bit words, 2-59, 5-29
words into packed list, 2-59
PACK, 2-59, 7-68
PAD, 2-11
Page start new, 2-97
Page-eject, 8-16
PAL, 2-36
Pascal subprogram summary, 2-96
Pascal, 8-1
PAUSE, 2-59, 7-38
PBN, 2-60, 5-53
PDD, 2-62, 6-9
PDDUMP, 2-26, 7-6
PERF, 2-60, 7-82
Performance counter group descriptions, 7-86
Performance monitor, 2-60
Performance statistics, B-1

SR-0014
Index-9
Permanet dataset
access, 2-4, 2-5
acquire front-end resident, 2-5
expand contract, 2-5
PERMIT, 2-60, 6-2
Plane rotation, Givens, 2-67, 4-11, 4-13, 4-19, 4-28, 4-4
Pocket sort, 4-75
POPCNT, 2-61, 3-14
POPPAR, 2-61, 3-14
Positive integer difference, 3-32
Positive real difference, 2-20
Positive real difference, 3-31
Power
scalar to scalar, 2-17, 2-18
scalar to vector, 2-17, 2-18
vector to scalar, 2-17, 2-18
vector to vector, 2-17 2-18
Power raised
double-precision, 2-25
double-precision, 2-26
scalar/vector, 2-68
single-precision, 2-46
Power raised, 3-14, 3-16
Powers of 10, 2-42, 2-82
PPL, 2-60, 7-53
PRBREAK, 2-96
PROW, 2-61, 5-54
Preface, v
Primary reference name, 1-1, 2-2
Product inner, 4-9
Prompt string, 2-96
Pseudo vectorization, 3-28
PTS, 2-61, 7-20
Publication list, v
PUTBYT, 2-61, 7-65
PUTWA, 2-90, 5-105
Radix sort, 4-75
Random access I/O, 5-83
Random number, A-16
Random number generator, 2-82
Random number processing, 2-62
Random number routines, 3-36
RANF, 2-62, 3-36, A-16
RANFI, 2-62, 2-82, 3-38
RANFES, 2-62, 2-82, 3-38
RANGST, 2-62, 3-36, A-17
RANSET, 2-62, 3-36, A-17
Rational number integer ceiling of, 2-41
RB, 2-62, 5-8
RENN, 2-63, 7-85
RCPFFT2, 2-31, 4-52
RCHP, 2-63, 5-67
RCHR, 2-63, 5-67
RCWP, 2-63, 2-63
RDIN, 2-63, 6-8
RDYQUE, 2-63, 9-12
RDYTSK, 2-63, 9-11
Read
asynchronously, 5-95, 5-108
Boolean, 2-97
buffer, 2-63
Read (continued)
buffered, 5-8, 5-17
CDC file format, 5-18
characters, 5-66, 5-67, 2-97
characters, 2-63, 2-64, 2-96
characters, full record mode, 5-68
characters, partial record mode, 5-68
character record, 8-12
directly from user area, 2-66
floating-point, 2-97
formatted, 5-3, 5-7, 2-66, 5-14
formatted, vectorized, 5-8
FORTRAN namelist, 5-8
IBM file format, 5-18
IBM floating point words, 2-64
IBM words, 5-69
integer, 2-97
list-directed, 2-67, 5-4, 5-14
logical record, 8-6
logical record in AI format, 7-58
new line, 2-97
one buffer of data, 6-8
record, 2-96, 2-97, 8-13
string, 2-97
synchronous/asynchronous, 5-96
unblocked data, 5-69
unformatted, 2-6, 5-3, 5-7, 5-14
unformatted, vectorized, 5-8
words, 2-63, 2-64, 2-96, 5-110
words, full record mode, 5-64, 5-66
words, partial record mode, 5-64, 5-66
READ, 2-64
Read/write check, 2-28
READC, 2-64, 5-68
READCP, 2-64, 5-68
READIBM, 2-64, 5-69
READMS, 2-53
READMS/READDR, 5-90
READP, 2-64, 5-66
Reads
random access, 5-90
READWA, 5-110
REAL, 2-65, 3-34
Real absolute value, 3-29
Real array
copies, 2-14
Record
backspace, 5-50
logical I/O, 5-63
Record format management
input, 5-18
output, 5-18
Record mode writes characters in, 2-91
Records
bypass, 5-40
copies, 2-15
copy, 5-35
RELEASE, 2-65, 6-3
REMARK, 2-65, 7-87
REMARK2, 2-65, 7-87
REMARKF, 2-65, 7-87
Reprieve processing (Pascal), 2-96
REPRIEVE processing, 2-65
Reprieve processing, 2-97, 8-13
RERUN, 2-66, 7-37
Reverse Polish Table, 2-11
REW, 2-66, 5-52
REWIND, 5-53
RF, 5-7
RF, 2-66
RF, 5-14
RFT, 5-3
RFV, 5-7, 5-8
Right shift, 3-14
RLA, 2-67
RLA read list-directed, 5-7
RLB, 2-66, 5-69
RLF, 2-67, 5-14
RLI, 2-67, 5-4
RNB, 2-67, 2-74, 7-85, 7-87
RNLCOMM, 2-67, 2-74, 7-87
RNLDVM, 2-55
RNLGRP, 2-55
RNLIM, 5-61
RNLIND, 2-55
RNLINP, 2-55, 5-61
RNLSKIP, 2-55, 5-60
RNLTYP, 2-55, 5-60
RTOL, 2-68, 3-16, A-17
RTOR, 2-68, 3-16, A-18
RUA, 2-69, 5-7
RUF, 5-14
RUI, 2-69, 5-3
Runtime errors, 8-19
Runtime initialization routine, 2-97
Runtime messages, 2-97
Runtime timing, 2-97, 2-97
RUTD, 2-69, 5-17
RUTDP, 2-69
RTTP, 2-69, 5-17
RTU, 2-69, 5-16
RUV, 2-69, 5-7, 5-8
RWDP, 5-64
RWDR, 5-64

Sectors
  bypass, 5-42
  copy, 5-39
SEEK, 2-90, 5-108
SECOALL, 2-72, 7-88
Segmented program, 2-72, 7-88
SEGRES, 2-72, 7-88
Sense switch, 2-79, 7-88
SENSEBT, 2-72, 7-40
SENSEFI, 2-72, 7-39
Set new length routines, 7-27
SETDP, 2-72, 7-41
SETETS, 2-73, 7-42
SETFI, 2-73, 7-39
SETFIS, 7-40
SETPLIM, 2-33, 2-73, 7-3
SETPOS, 2-74, 5-49
SETREW, 2-65, 7-37
SFR, 2-74
SHIFT, 2-74, 3-14
Shift left, 2-74, 2-75
Shift right, 2-75
SHIFTL, 2-75, 3-14
SHIFTR, 2-75, 3-14
SHRINK, 2-37, 7-31
SIGN, 2-75, 3-35
Sign
  transfer, 2-45
  transfer from one integer to another, 3-33
  transfer from one real number to another, 3-35
SIN, 2-75, 3-9
Sign
  complex, A-7
  double-precision, 2-24
SINE, 2-16, 2-75, 3-10, 3-9, A-5
SINE, d.p., hyperbolic, A-13
SINH, 2-75, 3-11
Skip sectors, 2-76
Skip to first good data, 2-76
SKIPBAD, 2-76, 5-76
SKIPD, 2-76, 5-41
SKIPF, 2-76, 5-40
SKIPR, 2-76, 5-40
SKIFU, 2-76, 5-42
SKOL processing, 2-38, 2-39, 2-40
SKOL run-time support, 7-56
SLEHP, 2-76, 7-62
SLFT, 2-77, 6-4
SMAC, 2-77, 4-29
SNMVE, 2-82
SNAP, 2-77, 7-7
SNRM2, 2-77, 4-10, 4-3
SOLR, 2-77, 4-34
SOLR3, 2-78, 4-36
SOLRN, 2-78, 4-35
Sort
  bucket, 4-75
  fixed-length records, 2-58
  multipass, 4-76
  pocket, 4-75
  radix, 4-75
Sort entry points, C-1
SORT ROUTINE, 4-73
Sparse matrix primitives, 4-24
SPAXPY, 2-8, 4-24
Special purpose subprograms, 7-1
SPYDAT, 4-24
SPOS, 2-46
SQRT, 2-78, 3-7, A-19
Square root complex, A-8
Square root routines, 3-7
Square root, 2-78, A-19
SREC, 2-79, 7-18
SROT, 2-67, 4-13, 4-4
SROTG, 2-67, 4-11, 4-4
SROTM, 2-67, 4-19, 4-4
SROTMG, 2-67, 4-13, 4-4
SSCAL, 2-70, 4-21, 4-4
SSOM, 2-80, 4-27
SSWAP, 2-81, 4-23, 4-4
SSWITCH, 2-79, 7-88
Stack header control word, 7-21
Stack management, 7-21
Stack processing, 2-79, 2-97
Stack walkback, 8-23
STINDX, 2-53
STKCR, 2-79, 7-23
STKDE, 2-79, 7-24
STKOPEN, 2-79, 7-23
STKFCK, 2-79, 7-24
STKUFEX, 2-79, 7-24
Stop, 2-79, 7-38
Store character item, 5-78
STORF, 2-48, 7-12
STPOS, 2-74, 5-47
STTMOV, 2-80, 7-63
Sub-index, 5-93
Submit, 2-80, 6-3
Subprogram summary, 2-1
Subroutine name address of, 2-35
Subroutines prints a list, 2-85
Subtraction double-precision, 2-23
SUSTSK, 2-80, 9-12
Symbolic dump, 2-81, 7-7
SYMDEBUG, 2-81, 7-7
SYNCH, 2-81, 5-55
SYNCRD, 2-81, 9-6
System clock, 2-13
System directory, 6-3
SYSTEM, 7-89

Table
  clears space, 2-95
  move words to, 2-54
  permanent dataset definitions, 6-9
  preset space, 2-61
  search, 2-79
  search with mask, 2-53
Table Base Table (BTAB), 7-12
Table Length Table (LTAB), 7-12
Table management, 7-12
Table manager, 2-81
Table of contents, vii

Table pointers, 2-43
Table space total allotted, 2-6
TADD, 2-82, 3-18
TAN, 2-83, 3-10
Tangent, 2-83, 3-10
Tangent, d.p., A-14
Tangent, hyperbolic, A-19
Tangent, hyperbolic d.p., A-14
TANH, 2-83, 3-11, A-19
Tape dataset, 5-55
Tape dataset position information, 5-44
Tape translation
  buffer management, 5-16
  data format management, 5-18
  description, 5-15
  record format management, 5-18
Task
  initiates, 2-85
  suspends execution, 2-80
Task information block, 2-83
Task routines, 9-1
Task, 2-86
Tasks
  read for execution, 2-63
  schedule logical CPUs for, 2-71
TASS, 2-82, 3-18
TDIV, 2-83, 3-19
TDSS, 2-83, 3-19
Temporarily prohibits
  floating-point interrupts, 2-13
Terminate program, 2-96
TMCR, 2-83, 9-11
TIBDE, 2-83, 9-11
Time
  obtain, 2-96, 2-97
  since start of job, 2-72
Time and date, 7-42
TIMEF, 2-19, 7-43
Timestamp routines, 7-44
Timestamp units, 7-45
Timestamp, 2-84
TM common block, 7-14
TMADW, 2-81, 7-16
TMAMU, 2-81, 7-19
TMATS, 2-81, 7-16
TMINIT, 2-81, 7-15
TMLT, 2-84, 3-19
TMEM, 2-82, 7-15
TMMSC, 2-82, 7-17
TMMEP, 7-21
Tmnts, 2-82, 7-20
TMSC, 2-82, 7-18
TMSS, 2-84, 3-19
TMVSC, 2-82, 7-19
TR, 2-84, 7-89
Traceback, 2-85, 7-4
Trailing blanks, 2-67, 2-74
Transfer
  character to result, 2-11
  one character item to result, 3-24
  termination, 2-11, 3-25
  sign, 2-24, 2-45, 2-75, 3-32
Translate

ASCII CDC display code, 5-20
CDC d.p. to Cray d.p. f.p., 5-19
CDC f.p. to Cray f.p., 5-19
CDC complex to Cray complex f.p., 5-19
CDC display code to ASCII, 5-19
CDC logical to Cray logical, 5-19
Cray integer to CDC integer, 5-20
Cray d.p. f.p. to CDC d.p., 5-20
Cray f.p. to CDC f.p., 5-20
Cray complex f.p. to CDC complex, 5-20
Cray logical to CDC logical, 5-20
IBM complex f.p. to Cray complex f.p., 5-20
IBM integer to Cray integer, 5-20
IBM logical to Cray logical, 5-20
IBM f.p. to Cray f.p., 5-20
IBM EBCDIC to ASCII, 5-19
internal format, 5-18

Translates

CDC formatted input, 2-11
CDC formatted output, 2-11
characters, 2-84
IBM formatted data, 2-40

Trigonometric routines, 3-10, 3-8, 3-9
Triple-precision
addition, 2-82, 3-18
arithmetic routines, 3-19, 3-20, 3-18
division, 2-83, 3-19
multiplication, 2-84, 3-19
subtraction, 2-86, 3-20
True condition
first location of, 2-45
Truncate
double-precision, 2-21
double-precision numbers, 3-31
to integral value, 3-29, 3-33

TSDT, 2-84, 7-44
TSKLIST, 9-7
TSKSTART, 2-85, 9-1
TSKTEST, 2-85, 9-4
TSKTUNE, 2-86, 9-5
TSKVALUE, 2-86, 9-4
TSKWAIT, 2-86, 9-3
TSM, 2-84, 7-45
TSSS, 2-86, 3-20
TSUB, 2-86, 3-20

U32, 2-86, 5-30
U6064, 2-87, 5-29
UEOFCL, 2-87
UEOFKL, 2-87, 5-59
UEOFPSK, 2-87, 5-59
UEOPTCL, 5-59
UIO, 5-78
UNIT, 2-87, 5-57
UNITTS, 2-84, 7-45
Unpack
32-bit words, 5-30
60-bit words, 2-87, 5-29

Vector
absolute values of, 4-5
add a scalar multiple, 4-6
complex, 4-6
cpy copy complex, 4-8
false values in, 4-58
gerather, 2-35, 4-56
largest absolute value of, 404
largest or smallest element of, 4-25
scale, 4-21
scatter, 4-56
smallest absolute value in, 2-45
sum of values, 4-27
ture values, 4-58
write to output, 2-92

Vector functions, 1-1
Vectorization
pseudo, 3-28

Vectors
convolution, 4-54
exchange, 4-23

WAITMS/WAITDR, 5-98
WB, 2-91, 5-13
WCHR, 2-92
WCFP, 2-91, 5-73
WCHF, 2-91, 5-73
WCLOSE, 2-90, 5-108
WDSET, 5-111
WDSETB, 5-111
WEOG, 2-91, 5-56
WEOF, 2-91, 5-56
WFA, 2-92, 5-11
WFBUFFER, 2-91, 7-89
WFF, 2-92, 5-14
WFF, 2-91, 5-4
WVF, 2-91, 5-12
WHEN routines, summary, 4-65
WHENSEQ, 2-92, 4-66
WHENFG, 2-92, 4-68
WHENFSL, 2-92, 4-68
WHENFGE, 2-92, 4-67
WHENFLT, 2-92, 4-67
WHENGE, 2-92, 4-70
WHENIGT, 2-92, 4-70
WHENILE, 2-92, 4-69
WHENILT, 2-92, 4-69
WHENNE, 2-92, 4-66

Whole number
  calculates nearest, 3-29
  nearest, 2-6

WLA, 2-93, 5-12
WLB, 2-92, 5-74
WLF, 2-93, 5-15
WLL, 2-93, 5-5
WLV, 5-13

WNL write FORTRAN namelist, 5-13
WNL, 2-96
WNLDELM, 5-62
WNLFLAG, 5-63
WNLLONG, 5-62
WNLREP, 5-63
WNLSEP, 5-63
WNOCHK, 2-92
WOPEN, 2-90, 5-103

Write
  Boolean, 2-97
  Boolean argument, 8-24
  buffered, 5-13, 5-17
  CDC file format, 5-18
  characters, 2-96, 2-97, 8-24
  characters, full record mode, 5-73, 5-74
  characters, partial record mode, 5-73, 5-74
  characters defined by the TYPE CHAR
    statement, 7-59
  character record, 8-14
  contents of exchange package, 7-9
  EOF, 2-96
  EOF, EOR, 5-56
  end-of-file, 2-97
  end of file mark, 8-25
  end-of-line, 2-97
  formatted, 2-92, 5-4, 5-11, 5-12, 5-14
  formatted, vectorized, 5-12
  floating-point, 2-94
  full record mode, 2-94
  IBM file format, 5-18
  IBM words, 5-74
  integer, 2-97
  into user area, 2-92
  list-directed, 2-93, 5-5, 5-12, 5-13, 5-15
  logical record, 8-17
  octal integer, 2-97
  partial record mode, 2-93, 2-94
  real number, 2-97
  record, 2-96, 2-97, 8-15
  packed character string, 8-27
  record mode, 2-95
  string, 2-97
  unblocked data, 5-74
  unformatted, 2-94, 5-4, 5-11, 5-12, 5-15
  unformatted, vectorized, 5-12
  words, 5-110
  words, full record mode, 5-70, 5-71
  words, partial record mode, 5-70, 5-71
WRITE, 2-93, 5-71

WRITEC, 2-93, 5-74
WRITECP, 2-93, 5-74
WRITEP, 2-93, 5-71

Write
  master index, 5-91
  random access, 5-89
WRITEA, 5-110
WRITEIBM, 2-94, 5-74
WRITMS, 2-53
WRITMS/WRITDR, 5-89
WUA, 2-94, 5-11
WUF, 2-94, 5-15
WUI, 2-94, 5-4
WUTD, 2-94, 5-17
WUTDP, 2-94
WUTF, 2-94, 5-18
WUTI, 2-94, 5-17
WUV, 2-94, 5-12
WWV, 2-95, 5-70
WWDP, 2-95, 5-70
WWDS, 2-95, 5-70
WWPU, 2-95, 5-70

XOR, 2-95, 3-14
XPFMT, 2-30, 7-9

Zero bits tally leading, 2-47
Zero/nonzero, 2-42
ZTS, 2-95, 7-20

SR-0014  Index-14
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 _______
READERS COMMENT FORM

Library Reference Manual

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.
Attention:
PUBLICATIONS

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