The following staff members authored sections or chapters of this manual:

Margaret Drake Chapter 5, Library Files
PLIB
ULIB

Janet Ivarie Index

Roy Jenne and Dennis Joseph Chapter 6, Library Routines that
process and Access Data
RPTOUT/RPTIN
GBYTE

Bernard O'Lear BUFRD
OVERLAY
SAVE

David Robertson Chapter 4, Plotting Library Routines

Vincent Wayland BRANRD
RANRD

The sustained and competent help of Janet Ivarie in the preparation of this volume is also gratefully acknowledged.

Portions of this manual have been edited from Control Data copyrighted publications with the permission of the Control Data Corporation.
CONTENTS

FOREWORD ........................................ iii
1. IN-LINE FUNCTIONS ............................... 1.1
   Description ..................................... 1.1
   Table of In-Line Functions Available on NCAR Compiler .... 1.2
2. MATHEMATICAL LIBRARY ROUTINES .............. 2.1
   Description ..................................... 2.1
   Call-by-Value Routines .......................... 2.1
   Call-by-Name Routines ........................... 2.3
   Table of Fortran Library Functions ............... 2.7
   Writeups of Mathematical Routines ............... 2.9
   ACOS  CLOG DLOG DLOG10 DCOS  DMAX1/DMIN1  DEXP  RANF
   ALOG  CLOG DLOG DLOG10 DCOS  DMAX1/DMIN1  RBAIEX  RBAIEX
   ALOG10  COS DLOC DLOG10 RBAIEX
   AMAX/AMIN  CSIN DLOG10 RBAIEX
   ASIN  CSQRT DMAX1/DMIN1  SIN
   ATAN  CUBRT DMOD SGL
   ATAN2  DATAN DSIGN SQRT
   ACOS  CABS DABEX DSQRT TAN
   COS  DBABS DSQRT TANH
   ATAN2  DATAN2  DSIN TAN
   DBAIEX  DRAIEX EXP TIMEF
   ACOS  DCOS  DEXP  RANF
   ALOG  ALOG10  CSIN  DLOG  DLOG10  RBAIEX
   AMAX/AMIN  CSQRT  DMAX1/DMIN1  SIN
   ATAN  CUBRT  DMOD  SGL
   ATAN2  DATAN  DSIGN  SQRT
   ACOS  CABS  DABEX  DSQRT  TAN
   COS  DBABS  DSQRT  TANH
   ATAN2  DATAN2  DSIN  TAN
   DBAIEX  DRAIEX  EXP  TIMEF

3. UTILITY LIBRARY SUBROUTINES .................. 3.1
   Description ................................... 3.1
   Writeups of Utility Subroutines .................. 3.3
   ACGOER  GBYTE  NEWLAB  SAVEF
   BACKSP  INPUTB  OUTPUTC  SORT
   BRANRD  INPUTC  OUTPUTS  SSW
   BUFFEI  INPUTS  OVERLAY  SYSRCL
   BUFRD  IOPROC  PAUSE  ULIB
   DEBUG  JOBID  PDUMP  UNLOAD
   DUMP  KORDER  PLIBRD
   ENDFIL  KRAKER  RANRD
   EXIT  MACHINE  RPTIN/RPTOUT

4. PLOTTING LIBRARY ROUTINES .................. 4.1
   General Description .......................... 4.1
   PLOT ........................................ 4.5
   Scaling ....................................... 4.6
   SET  PScale  PORGN  PORGN (MX,MY)
Plotting ........................................... 4.12
FRSTPT POINT (X,Y) LINE
VECTOR PSYM *CURVE
Writing ........................................... 4.16
PWRT ............................................. 4.16
Internal Flags for PWRT .................. 4.20
Control ......................................... 4.22
FLUSH OPTION MXMY
FRAME DASHLN
Backgrounds .................................... 4.29
GRID PERIML GRIDALL LABMOD
GRIDL HALFAX TICKS
PERIM AXES TICK4
Plot Instruction Manipulation .......... 4.41
FLASH1 FLASH3 LTPEN
FLASH2 FLASH4
Instruction Format ......................... 4.46
Normal Format ............................... 4.47
Tabular Format ............................. 4.47
Other Useful Routines .................... 4.48
CALCNT IDIOT SUPMAP
5. ULIB--LIBRARY OF SPECIAL PURPOSE SUBROUTINES .......... 5.1
General Description ...................... 5.1
Algebraic Zeros ........................... 5.3
Approximation and Curve Fitting .... 5.5
Elementary Functions ................... 5.9
Fast Fourier Transforms ................. 5.12
Graphic Display .......................... 5.13
Input/Output ................................ 5.15
Matrix Analysis ............................ 5.16
Miscellaneous ............................. 5.20
Nonalgebraic Zeros ..................... 5.23
Ordinary Differential Equations ....... 5.24
Partial Differential Equations ....... 5.25
Quadrature ................................. 5.26
Random Number Generators ............. 5.28
Statistics .................................. 5.29
6. LIBRARY ROUTINES THAT PROCESS AND ACCESS DATA .......... 6.1
   Tape Copy and Edit ............................................. 6.1
   USER USERF  .................................................. 6.1
   Data Control Cards ............................................. 6.4
   Mode and Type Options .......................................... 6.5
   Control Words .................................................. 6.6
   Copy Output .................................................... 6.7
   Other Data Handling Routines .................................... 6.9
   CDS360 RDTAPE and WRTAPE
   CDTAPE and CDTAPEB REPBIN
   CHCONV RPTIN and RPTOUT
   CORFOR SORTUU and SORTBB
   GBITS and SBITS TDUMP
   GBYTES and SBYTES UBLOK and UZBLOK

7. CENTRAL MONITOR ROUTINES ...................................... 7.1
   General Description ............................................. 7.1
   Error Message .................................................. 7.1
   Monitor Communications Area .................................... 7.1
   Registers ....................................................... 7.7
   Ascent Calling Sequence ....................................... 7.7
   Arguments for a CM Call ....................................... 7.7
   RETURN UNIT LWA
   OP STATUS
   CNT FWA
   Arguments for a CM MACRO Call .................................. 7.12
   I/O File Handling ............................................... 7.12

8. EXECUTION DIAGNOSTICS ......................................... 8.1
   General Discussion ............................................. 8.1
   Mathematical Library Functions .................................. 8.5
   Utility Routines ................................................ 8.6
   Loader Diagnostics ............................................. 8.24
   Loader Table Size Limits ....................................... 8.26

9. INDEX .......................................................... 9.1
The code for in-line functions is placed directly in the user's program by the compiler. These functions are available only through a Fortran reference. The logic is inserted in the object code of a Fortran program.

Example

The statement

\[ E = \text{FLOAT}(J) \]

is compiled, and inserted in Ascent as follows:

- SA4 J Bring J in from core
- PX7 B0,X4 Pack the bias 2000
- NX6 B0,X7 Normalize
- SA6 E Store in E

The operation is to take the value of the integer J and place it in E as a normalized floating-point number.

In-line function names do not appear in the loader map. Routines supplied by the user should not be given the same names as in-line functions. Although such routines will be loaded, they will not be called; the in-line function of the same name will be compiled instead.

A table giving the in-line functions available on the NCAR compiler follows. Fortran II names appear in brackets.
### Table 1: In-Line Functions Available on NCAR Compiler

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>FORMS</th>
<th>PARAMETER MODE</th>
<th>RESULT MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Absolute Value</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>ABS(X)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[ABSF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>IABS(I)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XABSF(I)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>DABS(D)</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conversion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To truncate a real argument</td>
<td>AINT(X)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[INTF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To truncate a real number</td>
<td>INT(X)</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XINTF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFIX(X)</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XFIXF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To float an integer number to floating point</td>
<td>FLOAT(I)</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[FLOATF(I)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain the imaginary part of a complex argument</td>
<td>AIMAG(C)</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>To obtain a double-precision number from a real number</td>
<td>DBLE(X)</td>
<td>Real</td>
<td>Double</td>
</tr>
<tr>
<td>To obtain the real part of a complex argument</td>
<td>REAL(C)</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>To convert real to complex</td>
<td>CMPLX(X₁,X₂)</td>
<td>Real</td>
<td>Complex</td>
</tr>
<tr>
<td>To obtain the complex conjugate of an argument</td>
<td>CONJG(C)</td>
<td>Complex</td>
<td>Complex</td>
</tr>
</tbody>
</table>
### 1.3 In-Line Functions

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>FORMS</th>
<th>PARAMETER MODE</th>
<th>RESULT MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modulus</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_1 - (X_1/X_2)X_2$</td>
<td>$\text{AMOD}(X_1,X_2)$</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>$[\text{MODF}(X_1,X_2)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>where</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(X_1/X_2) = \text{integer part}$</td>
<td>$\text{MOD}(I_1,I_2)$</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>$[\text{XMODF}(I_1,I_2)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sign</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign of $X_2$ times absolute value of $X_1$</td>
<td>$\text{SIGN}(X_1,X_2)$</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>$[\text{SIGNF}(X_1,X_2)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign of $I_2$ times absolute value of $I_1$</td>
<td>$\text{ISIGN}(I_1,I_2)$</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>$[\text{XSIGNF}(I_1,I_2)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location$^1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To determine the address of the variable name stated in the argument</td>
<td>$\text{ALOC}(\text{VAR}_1)$</td>
<td>Not applicable</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>$[\text{LOCF}(\text{VAR}_1)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{LOC}(\text{VAR}_1)$</td>
<td>Not applicable</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>$[\text{XLOCF}(\text{VAR}_1)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Positive Difference</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1 - \text{MIN}(a_1,a_2)$</td>
<td>$\text{DIM}(X_1,X_2)$</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>$[\text{DIMF}(X_1,X_2)]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{IDIM}(I_1,I_2)$</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>$[\text{XDIMF}(I_1,I_2)]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^1$The "real" mode of result may not be practical when dealing with addresses. Integer is the mode of result used with most applications.
MATHEMATICAL LIBRARY ROUTINES

DESCRIPTION

Mathematical functions and subprograms that are used frequently are stored in the library and need not be submitted with the program (job). A reference to these library routines will load them from the system along with the routines submitted. A library routine reference contains at least one parameter. These library routines are call-by-value or call-by-name routines.

CALL-BY-VALUE ROUTINES

The call-by-value routines are primarily those library functions with one argument. These function-type routines return a value associated with the function name; the mode of the function is determined by a type indicator or the name of the function.

The function call may be included anywhere in an arithmetic statement. For example, \( E = (4.0^X)/\text{ALOG}(X) \) is an arithmetic statement where the value of \( \text{ALOG}(X) \) is returned in an X-register for inclusion in the statement. When \( \text{ALOG} \) is called in this way, the value of \( X \) is placed in an X-register on entry to the subprogram. "Values" of parameters are sent to the subprogram and returned by the subprogram via X-registers. Function subprograms on the library tape save all B-registers and restore them before returning to the main code.
2.2
Mathematical Library Routines

**CALL-BY-VALUE Routines**

(Continued)

**Example 1**

\[ E = \text{ALOG}(X) \]

- **SAl** X
- **RJ** LOGF
- **BX6** X1
- **SA6** E

In the example, the compiled program establishes the address of the argument X, and places the contents of X in register X1 by setting A1 to X. A return jump (RJ) to the function is generated. The logarithm function code is loaded into core as a separate routine along with the other programs and subprograms used. A system routine of this type uses the value of the argument in X1 directly and returns the result of the function in X1 directly. The value associated with the logarithm of X is placed in X1 in the subprogram, and a jump is made to the calling program. In order to store the value in X1 in E, X1 is transferred to X6 and the address register A6 is set to the address of E.

Both Fortran II (LOGF) and Fortran IV (ALOG) names are acceptable. The compiler may use Fortran II names in the RJ whenever Fortran II names have been defined. Both names (LOGF and ALOG) appear in the loader map.

If a double-precision or complex routine is called and two values are returned, they are placed in X1 and X2, as shown in the following example.
Example 2

\[ Z = \text{CCOS}(C) \]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>C</td>
</tr>
<tr>
<td>SA2</td>
<td>C+1</td>
</tr>
<tr>
<td>RJ</td>
<td>CCOS</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>BX7</td>
<td>X2</td>
</tr>
<tr>
<td>SA6</td>
<td>Z</td>
</tr>
<tr>
<td>SA7</td>
<td>A6+1B</td>
</tr>
</tbody>
</table>

In this example the contents of C (i.e., the complex argument stored in C and C+1) are placed in X1 and X2. After the RJ, the value of the real part of the answer is in X1 and the value of the imaginary part is in X2. Z is type-complex. The real part (in X1) is placed in X6 and stored in Z, which is type-complex, by placing the address of Z in A6. The imaginary part (in X2) is placed in X7 and stored by setting A7 to the address of Z (in A6) +1. A similar procedure is generated for double-precision routines where the most significant part of the number returns in X1 and the least significant part in X2.

**CALL-BY-NAME Routines**

Library routines may also be *call-by-name*. Addresses or names of parameters are sent to the subprogram. The subprogram must then set an A-register to the name in order to fetch the value of A in an X-register. Call-by-name functions return values according to the rules for functions stated *above* where results are returned by value in an X-register. Subroutines only return values by name as they appear in the subroutine call.
2.4
Mathematical Library Routines

CALL-BY-NAME
Routines
(Continued)

Example 1

A=X**Y

<table>
<thead>
<tr>
<th>RJ</th>
<th>RBAREX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>**+3</td>
</tr>
<tr>
<td>CON</td>
<td>X</td>
</tr>
<tr>
<td>CON</td>
<td>Y</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>SA6</td>
<td>A</td>
</tr>
</tbody>
</table>

In this example the function locates the addresses of X and Y in the two locations following the RJ to RBAREX. When the contents of one of the addresses are needed, the function sets an A-register to that address and the contents are returned.

Example 2

An example of Ascent code in the subprogram RBAREX to fetch arguments from the previously compiled call to RBAREX is:

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>RBAREX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBAREX</td>
<td></td>
</tr>
<tr>
<td>CON</td>
<td>0</td>
</tr>
<tr>
<td>SA1</td>
<td>RBAREX</td>
</tr>
<tr>
<td>LX1</td>
<td>30</td>
</tr>
<tr>
<td>SA2</td>
<td>X1+1</td>
</tr>
<tr>
<td>SA3</td>
<td>A2+1</td>
</tr>
<tr>
<td>SA4</td>
<td>X2</td>
</tr>
<tr>
<td>SA5</td>
<td>X3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EQ</td>
<td>RBAREX</td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
In this example, the address X in A4 fetches the contents of X in X4 and the address of Y in A5 fetches the contents of Y in X5. The locations of these parameter names are found by locating the return address placed in the entry point when a RJ is executed. Full word addressing is therefore possible for any parameters.

All subprograms written in Fortran or in Ascent and supplied by the programmer are compiled as call-by-name subprograms. Function-type subprograms must save and restore any B-registers used in the function, as the compiler assumes these will not be changed by a function call. Function subprograms return a value for the name of the function in X1. If the routine uses A0, the ** operator (RBAREX, RBAIEX, CBAIIX, etc.) requires that A0 be saved and restored as well as the B-registers.

On the other hand, the compiler does not require B-register restoration by subroutines. If a user subprogram has the same name as a library routine, the user subprogram must be written in Ascent in order to match the calling sequence generated for the library routines by the compiler. This manual gives Ascent equivalents for each name used. User subprograms must exactly match the library routines, including the multiply entry points.
The compiler does not generate the call-by-name sequence of code to pick up arguments in Fortran subprograms, however. The compiler does a RJ to an initialization routine called Q8QRSD, which picks up the formal parameter list and presets the parameter addresses directly in core for all instructions containing address parameters. The number of addresses to be preset is maintained in a use count tabulated by the compiler. If subsequent calls to this subprogram repeat the argument list, Q8QRSD will not preset these arguments in instructions a second time. It branches out and uses the instructions already set by the previous call.

Table 2 is a list of the Fortran library functions; the NCAR library routines, listed alphabetically, are described in detail following the table.

Fortran II names appear in brackets.

All error messages from these subroutines are the result of a jump to Q8QERR. When the message is printed, the execution terminates (see Chapter 8 for details).

An asterisk (*) after the word Method in the descriptions indicates that the routine has been checked for the given method. If no asterisk appears, the method is copied from Control Data Corporation literature, and is the method used in routines supplied by the Control Data Corporation.
### Table 2: Fortran Library Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>PARAMETER MODE</th>
<th>RESULT MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS(X)</td>
<td>Arcosine</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ALOG(X)</td>
<td>Natural log of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ALOG10(X)</td>
<td>Log to the base 10 of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>AMAXO(I₁, I₂, ..., Iₙ)</td>
<td>Maximum of n arguments¹</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>AMINO(I₁, I₂, ..., Iₙ)</td>
<td>Minimum of n arguments¹</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>ASIN(X)</td>
<td>Arcsine</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ATAN(X)</td>
<td>Arctangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ATAN2(X₁, X₂)</td>
<td>Arctangent X₁/X₂</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>CABS(C)</td>
<td>Absolute value</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>CBAIEX(C, I)</td>
<td>Complex C**I</td>
<td>Complex</td>
<td>(to integer)</td>
</tr>
<tr>
<td>CCOS(C)</td>
<td>Complex cosine</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CEXP(C)</td>
<td>Complex exponential</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CLOCKF</td>
<td>Current real time</td>
<td>--</td>
<td>DPC</td>
</tr>
<tr>
<td>CLOG(C)</td>
<td>Complex log function</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>COS(X)</td>
<td>Cosine X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>CSIN(C)</td>
<td>Complex sine</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CSQRK(C)</td>
<td>Complex square root</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CURRT(Y)</td>
<td>Cube root</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>DATAN(D)</td>
<td>Double arctangent</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DATAN2(D₁, D₂)</td>
<td>Double arctangent D₁/D₂</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DDATEF</td>
<td>Current date</td>
<td>--</td>
<td>DPC</td>
</tr>
<tr>
<td>²DBINDEX(D₁, D₂)</td>
<td>D₁**D₂</td>
<td>Double</td>
<td>(to double)</td>
</tr>
<tr>
<td>²DBINDEX(D₁, I)</td>
<td>D₁**I</td>
<td>Double</td>
<td>(to integer)</td>
</tr>
<tr>
<td>²DBINDEX(D₁, X)</td>
<td>D₁**X</td>
<td>Double</td>
<td>(to real)</td>
</tr>
<tr>
<td>DCOS(D)</td>
<td>Double cosine</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DEXP(D)</td>
<td>Double exponential function (e^D)</td>
<td>Double</td>
<td>Double</td>
</tr>
</tbody>
</table>

¹For detailed forms of the function called, see AMAX/AMIN writeup.

²These routines must be called with a ** operator. They may not be used as functions. These routines must save and restore A₀ as well as B-registers used. Other library functions need only save the B-registers.
### Table 2: Fortran Library Functions (continued)

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>PARAMETER MODE</th>
<th>RESULT MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLOG(D)</td>
<td>Natural log of D (base e)</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DLOG10(D)</td>
<td>Log to base 10 of D</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMAXI(D₁,D₂,...,Dₙ)</td>
<td>Maximum of n arguments</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMINI(D₁,D₂,...,Dₙ)</td>
<td>Minimum of n arguments</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMD(D₁,D₂)</td>
<td>D₁ modulo D₂</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>D1-(AINT(D1/D2)D2)</td>
<td>D₁-(AINT(D1/D2)D2)</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DSIGN(D₁,D₂)</td>
<td>Sign of D₂ times</td>
<td>D₁</td>
<td></td>
</tr>
<tr>
<td>DGIN(D)</td>
<td>Sine of double-precision argument</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DSORT(D)</td>
<td>Square root of double</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>EXP(X)</td>
<td>e to x\textsuperscript{th} power (e\textsuperscript{x})</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>²IBAEX(I₁,I₂)</td>
<td>I₁\textsuperscript{I₂}</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>IDINI(D)</td>
<td>Convert double-precision argument to integral floating-point number (rounded)</td>
<td>Double</td>
<td>Real</td>
</tr>
<tr>
<td>RANF(X)</td>
<td>Random number generator; results between 0 and 1; rectangular distribution</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>²RBAEX(X₁,I)</td>
<td>X₁\textsuperscript{I}</td>
<td>Real (to integer)</td>
<td>Real</td>
</tr>
<tr>
<td>²RBAREX(X₁,X₂)</td>
<td>X₁\textsuperscript{X₂}</td>
<td>Real (to real)</td>
<td>Real</td>
</tr>
<tr>
<td>SIN(X)</td>
<td>Sine X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>SNGL(D)</td>
<td>Convert double-precision argument to real</td>
<td>Double</td>
<td>Real</td>
</tr>
<tr>
<td>SQRT(X)</td>
<td>Square root of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>TAN(X)</td>
<td>Tangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>TANH(X)</td>
<td>Hyperbolic tangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>TIMEF(l)</td>
<td>Elapsed CPU time in milliseconds</td>
<td>--</td>
<td>Real</td>
</tr>
</tbody>
</table>

²These routines must be called with a \textsuperscript{²} operator. They may not be used as functions. These routines must save and restore A0 as well as B-registers used. Other library functions need only save the B-registers.
ACOS

**Note:** This routine calculates both ASIN and ACOS, using multiple entry points (see ASIN, p. 2.15).

**Purpose**
To compute the single-precision arccosine of a single-precision argument.

**Fortran Function**

\[ B = \text{ACOS}(X) \quad [B = \text{ACOSF}(X)] \]

**Entry Point**

ACOS[ACOSF]

**Ascent Calling Sequence**

<table>
<thead>
<tr>
<th>Ascent Calling</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ</td>
<td>ACOS</td>
</tr>
<tr>
<td>BX6</td>
<td>XL</td>
</tr>
<tr>
<td>SA6</td>
<td>B</td>
</tr>
</tbody>
</table>

**Normal Return**
The result in the range \(-\pi/2\) to \(\pi/2\) is returned in XL and stored in B.

**Error Message**
ARG GT 1, OR INDEFINITE

**Storage**

1338 locations

**Accuracy**

1.E-12 in the worst case

**Timing**

6600: 69 µsec 7600: 12 µsec

**Method**
The method used is a power series telescoped by the use of Chebyshev polynomials. The range of X is partitioned into intervals, and values of constants are assigned in the approximation equation accordingly. Near the origin for values of \(X < 0.05\), a Maclaurin series is used. The results are in quadrant I or quadrant IV. ACOS is computed as \(\pi/2 - \text{ASIN}(X)\). The result returns \(0 < B < \pi\); that is, in the first or second quadrant.


**ALOG**

**Purpose**
To calculate the natural logarithm of \( x \) in single precision where \( x \) is single precision.

**Fortran Function**
\[ E = \text{ALOG}(X) \ [E = \text{LOGF}(X)] \]

**Entry Point**
ALOG[LOGF]

**Ascent Calling Sequence**
SA1 X
RJ ALOG
BX6 XL
SA6 E

**Normal Return**
The result returned in XL is stored in E.

**Error Message**
ARG ZERO OR NEG

**Storage**
358 locations

**Accuracy**
In the range \([0.1, 10.0]\) the relative error is approximately \(10^{-13}\). Outside this range the relative error is approximately \(10^{-14}\).

**Timing**
6600: 33 μsec    7600: 6.9 μsec
To calculate $\log_e x$ a range reduction is performed so that

$$x = 2^{N-\frac{3}{2}} s, \quad s \in \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right],$$

then the approximation\(^3\)

$$\log_e s \approx z \frac{P(z^2)}{Q(z^2)}$$

where $z = \frac{s-\frac{1}{2}}{s+1}$

is used giving $\log_e x = (N-\frac{3}{2}) \log_e 2 + \log_e s$. The polynomials $P(z^2)$ and $Q(z^2)$ are given by

$$P(z^2) = P_0 + P_1 z^2 + P_2 z^4$$

$$Q(z^2) = Q_0 + Q_1 z^2 + z^4$$

where

$$P_0 = 8.28955848560451297$$
$$P_1 = -6.491111394895693$$
$$P_2 = 0.573146236434104$$
$$Q_0 = 4.14477924280070217$$
$$Q_1 = -4.627148777339487$$

Special properties of ALOG are that

- if $x = 1.0$, then $\ln(x) = 0$, and
- if $x \leq 0$, an error return is generated.

---

\(^3\)This approximation is taken from The SIAM series in Applied Mathematics - Computer Approximations, John Wiley and Sons, 1968, and is approximation number 2703.
2.12
Mathematical Library Routines

**ALOG10**

**Purpose**
To calculate to the base 10 the logarithm of a floating-point argument.

**Fortran Function**
\[ A = \text{ALOG10}(X) \]

**Entry Point**
ALOG10

**Ascent Calling Sequence**
SA1 X
RJ ALOGL0
BX6 XI
SA6 A

**Normal Return**
The result returned in XI is stored in A.

**Error Message**
ARG ZERO OR NEGATIVE
INFINITE OR INDEFINITE ARG

**Storage**
56\(_8\) locations

**Accuracy**
\[ 10^{-15}. \text{ This is relative error using the double-precision logarithm as base.} \]

**Timing**
6600: 43 \(\mu\text{sec} \quad 7600: 8 \mu\text{sec}

**Method**
\[ \log_{10}(x) = 0.43429448190325 \log_e(x) \]
## AMAX/AMIN

### Purpose
To calculate the maximum (or minimum) in \( n \) arguments.

### Entry Points

<table>
<thead>
<tr>
<th>Entry Forms</th>
<th>Parameter Mode</th>
<th>Result Mode</th>
<th>Timing (( \mu \text{sec} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMAXO(I₁,I₂,...,Iₙ)</td>
<td>Integer</td>
<td>Real</td>
<td>36</td>
</tr>
<tr>
<td>[MAXOF(I₁,I₂,...,Iₙ)]</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>AMAX1(X₁,X₂,...,Xₙ)</td>
<td>Real</td>
<td>Real</td>
<td>37</td>
</tr>
<tr>
<td>[MAX1F(X₁,X₂,...,Xₙ)]</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>MAXO(I₁,I₂,...,Iₙ)</td>
<td>Integer</td>
<td>Integer</td>
<td>35</td>
</tr>
<tr>
<td>[XMAXOF(I₁,I₂,...,Iₙ)]</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>MAX1(X₁,X₂,...,Xₙ)</td>
<td>Real</td>
<td>Integer</td>
<td>38</td>
</tr>
<tr>
<td>[XMAX1F(X₁,X₂,...,Xₙ)]</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>AMINO(I₁,I₂,...,Iₙ)</td>
<td>Integer</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>[MINOF(I₁,I₂,...,Iₙ)]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMIN1(X₁,X₂,...,Xₙ)</td>
<td>Real</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>[MIN1F(X₁,X₂,...,Xₙ)]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINO(I₁,I₂,...,Iₙ)</td>
<td>Integer</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>[XMINOF(I₁,I₂,...,Iₙ)]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN1(X₁,X₂,...,Xₙ)</td>
<td>Real</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>[XMIN1F(X₁,X₂,...,Xₙ)]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Ascent Calling Sequence
For \( R = \text{AMAXO}(I,J,K) \)

- **RJ**  MAXOF
- **EQ**  \#+4
- **CON**  I
- **CON**  J
- **CON**  K
- **BX6**  X1
- **SA6**  R
AMAX/AMIN
(Continued)

Normal Return
The result returned in XI is stored in R.

Error Message
THERE MUST BE AT LEAST TWO ARGS

Storage
103, locations

Accuracy
The formula is exact.

Method*
A loop is set up to determine the maximum (minimum) argument by successive subtractions.
**ASIN**

This routine calculates both ASIN and ACOS, using multiple entry points (see ACOS, p. 2.9).

**Purpose**

To compute the single-precision arcsine of a single-precision argument.

**Fortran Function**

\[ B = \text{ASIN}(X) \quad [B = \text{ASINF}(X)] \]

**Entry Point**

ASIN[ASINF]

**Ascent Calling Sequence**

| SA1 | X |
| RJ  | ASIN |
| BX6 | X1 |
| SA6 | B |

**Normal Return**

The result in the range \(-\pi/2 \to \pi/2\) is returned in X1 and stored in B.

**Error Message**

ARG GT 1, OR INDEFINITE

**Storage**

133, locations

**Accuracy**

1.E-12 in the worst case

**Timing**

6600: 65 µsec 7600: 12 µsec

**Method**

The method used is a power series telescoped by the use of Chebyshev polynomials. The range of X is partitioned into intervals, and values of constants are assigned in the approximation equation accordingly. Near the origin for values of X < 0.05, a Maclaurin series is used. The results are in quadrant I or quadrant IV. ACOS is computed as \(\pi/2 - \text{ASIN}(X)\). The result returns 0 < B < \(\pi\); that is, in the first or second quadrant.
ATAN

Purpose To calculate the arctangent of X in radians.
\((-\pi/2 < \text{result} < \pi/2)\)

Fortran Function \(A = \text{ATAN}(X) \ [A = \text{ATANF}(X)]\)

Entry Point \(\text{ATAN}[\text{ATANF}]\)

Ascent Calling Sequence

- \(\text{SA1} \ X\)
- \(\text{RJ} \ \text{ATAN}\)
- \(\text{BX6} \ X1\)
- \(\text{SA6} \ A\)

Normal Return The result returned in \(X1\) is stored in \(A\). The result is in radians, and is between \(-\pi/2\) and \(\pi/2\).

Error Message \(\text{INDEFINITE OR INFINITE ARG}\)

Storage \(728\) locations

Accuracy \(10^{-15}\). This is a relative error compared with double-precision arctangent as base.

Timing \(6600: 37\ \mu\text{sec} \quad 7600: 8\ \mu\text{sec}\)
Method

Let $A = \text{ATAN}(X)$, then $-\pi/2 < A < \pi/2$

Let $P = \tan(\pi/16)$, $T = \tan(3\pi/16)$

$\text{ATAN}(W) = \text{sign}(W) \cdot \text{ATAN}(V)$, $V = |W|$

$\text{ATAN}(V) = \text{ATAN}(R) + C$

$R$ and $C$ defined below:

- $0 < V < P$
  - $R = V$
  - $C = 0.0$

- $P < V < 2^{3/2} - 1$
  - $R = (V-P)/(1+V*P)$
  - $C = \pi/16$

- $2^{3/2} - 1 \leq V < 1$
  - $R = (V-T)/(1+V*T)$
  - $C = 3\pi/16$

- $1 \leq V < 2^{3/2} + 1$
  - $R = (1-V*T)/(V+T)$
  - $C = 5\pi/16$

- $2^{3/2} + 1 \leq V$
  - $R = (1-V*P)/(V+P)$
  - $C = 7\pi/16$

$\text{ATAN}(R) = R - R^2Q$, $Z = R^2$

$$Q = \frac{n_0 + n_1 Z + n_2 Z^2 + n_3 Z^3}{d_0 + d_1 Z + d_2 Z^2 + d_3 Z^3}$$
ATAN2

Purpose
To calculate the arctangent of Y/X in the range
\(-\pi < \text{result} < \pi\).

Fortran Function
B = ATAN2(Y,X)

Entry Point
ATAN2

Ascent Calling Sequence
RJ ATAN2
EQ \#+3
CON Y
CON X
BX6 XL
SA6 B

Normal Return
The result returned in XL is stored in B. The result in
radians is between \(-\pi\) and \(\pi\).

Error Message
INDEFINITE OR INFINITE ARG OR X = Y = 0

Storage
1168 locations

Accuracy
10^{-15}. This is relative error compared with the double-
precision arctangent 2.

Timing
6600: 58 \mu sec
7600: 11 \mu sec
Method

Let \( B = \text{ATAN2}(Y,X) \), then \( B \) is the argument of the complex number \( X = iY \) and \(-\pi \leq B \leq \pi\).

\[
B = \begin{cases} 
\text{sign}(Y) \pi/2 & X = 0 \\
\text{ATAN}(Y/X) & X > 0 \\
\text{ATAN}(Y/X) + \text{sign}(Y) \pi & X < 0
\end{cases}
\]

Let \( P = \tan(\pi/16) \), \( T = \tan(3\pi/16) \), \( W = Y/X \)

\( \text{ATAN}(w) = \text{sign}(w) \text{ATAN}(v), \ V = \text{ABS}(W) \)

\( \text{ATAN}(V) = \text{ATAN}(R) + C \)

\( R \) and \( C \) defined below:

\[
\begin{align*}
0 \leq V < P & \quad R = V & C = 0.0 \\
P \leq V < 2^{1/2} - 1 & \quad R = (V-P)/(1+V\times P) & C = \pi/16 \\
2^{1/2} - 1 \leq V < 1 & \quad R = (V-T)/(1+V\times T) & C = 3\pi/16 \\
1 \leq V < 2^{1/2} + 1 & \quad R = (1-V\times T)/(V+T) & C = 5\pi/16 \\
2^{1/2} + 1 \leq V & \quad R = (1-V\times P)/(V+P) & C = 7\pi/16
\end{align*}
\]

\( \text{ATAN}(R) = R - R\times Q \), \( Z = R^2 \)

\[
Q = \frac{n_0 + n_1 Z + n_2 Z^2 + n_3 Z^3}{d_0 + d_1 Z + d_2 Z^2 + d_3 Z^3}
\]
CABS

Purpose
To calculate the magnitude of a complex number.

Fortran Function
\[ X = \text{CABS}(Z) \]

Entry Point
CABS

Ascent Calling Sequence
\begin{align*}
\text{SA1} & \quad Z \\
\text{SA2} & \quad Z + 1 \\
\text{RJ} & \quad \text{CABS} \\
\text{EX6} & \quad X_1 \\
\text{SA6} & \quad X \\
\end{align*}

Normal Return
The result (a real number) returned in \( X_1 \) is stored in \( X \).

Error Message
None

Storage
348 locations

Accuracy
The relative error is \( 10^{-15} \).

Timing
\begin{align*}
6600: & \quad 28 \mu\text{sec} \\
7600: & \quad 6 \mu\text{sec}
\end{align*}

Method
The result approximates
\[ |Z| = (x^2 + y^2)^{\frac{1}{2}} \]
where \( Z = X + iY \).
CBAIEX

Purpose
To compute $Z^I$ where $Z$ is complex and $I$ is integer. The real part is in $Z$ and the imaginary part in $Z+1$.

Fortran Function
$C = Z^I$ where $C$ is complex.

Entry Point
CBAIEX

Ascent Calling Sequence
RJ CBAIEX
EQ **+3
CON Z
CON I
LX6 X1
BX7 X2
SA6 C
SA7 C+l

Normal Return
The real part of the result is in X1, the imaginary in X2. The result is stored in C.

Error Message
ZERO TO ZERO OR NEGATIVE POWER

Storage
338 locations

Timing
<table>
<thead>
<tr>
<th></th>
<th>6600</th>
<th>7600</th>
</tr>
</thead>
<tbody>
<tr>
<td>For an exponent of 2:</td>
<td>33 μsec</td>
<td>6 μsec</td>
</tr>
<tr>
<td>For an exponent of 10:</td>
<td>45 μsec</td>
<td>7 μsec</td>
</tr>
</tbody>
</table>

Method*
Successive complex multiplications are performed in a loop depending on the exponent $I$. If $Z = 0$ and $I > 0$, $C = 0$. If $Z = 0$ and $I < 0$, an exit will be forced, and the panel and an error message will be printed. If $Z \neq 0$ and $I < 0$, $C = 1/C$. 

CCOS

Purpose
To calculate the cosine of the complex argument Z.

Fortran Function
\[ C = \text{CCOS}(Z) \]

Entry Point
CCOS

Externals
COSF, SINF, EXPF

Ascent Calling Sequence
\[ \begin{align*}
SA1 & : Z \\
SA2 & : Z+1 \\
RJ & : \text{CCOS} \\
BX6 & : X1 \\
LX7 & : X2 \\
SA6 & : C \\
SA7 & : C+1
\end{align*} \]

Normal Return
The real part is in X1, the imaginary in X2. The complex result is stored in C.

Error Message
INFINITE OR INDEFINITE ARG

Storage
368 locations

Accuracy
The relative error is \(10^{-15}\).

Timing
6600: 121 µsec 7600: 23 µsec

Method
\[ \text{CCOS}(Z) = \text{CSIN} \left( \frac{\pi}{2} - Z \right) \]
CEXP

Purpose
To calculate the exponential of a complex argument Z.

Fortran Function
C = CEXP(Z)

Entry Point
CEXP

Externals
SINF, COSF, EXPF

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ CEXP
EX6 X1
SA6 B0+C
IX7 X2
SA7 A6+1B

Normal Return
The real part is in X1, the imaginary in X2. The complex result is stored in C.

Error Message
None in CEXP. See EXP, SIN, COS.

Storage
158 locations

Accuracy
About $10^{-13}$

Timing
6600: 106 μsec 7600: 22 μsec

Method*
For complex $Z = x + iy$

$$e^Z = e^x \cos(y) + ie^x \sin(y)$$

EXP, COS, SIN are called to evaluate this.
**CLOCKF/DATEF**

**Purpose**
To obtain the current real time and the date.

**Fortran Function**

\[
R = \text{CLOCKF}(1.0) \\
R = \text{DATEF}(1.0)
\]

**Entry Points**

CLOCKF, DATEF

**Ascent Calling Sequence**

<table>
<thead>
<tr>
<th>RJ</th>
<th>CLOCKF</th>
<th>RJ</th>
<th>DATEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX6</td>
<td>X1</td>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>SA6</td>
<td>R</td>
<td>SA6</td>
<td>R</td>
</tr>
</tbody>
</table>

**Normal Return**

The result in X1 is stored in R.

**Error Message**

None

**Storage**

128 locations

**Method**

The central monitor reads the clock cell or the date cell. The result is left-justified alphanumeric (DPC) information, returned in X1. Notice that R and CLOCKF are real. Thus there are no mixed-mode changes in the DPC characters.
**CLOG**

**Purpose**
To calculate the complex logarithm of a complex argument Z to the base e.

**Fortran Function**
\[ C = \text{CLOG}(Z) \]

**Entry Point**
CLOG

**Ascent Calling Sequence**
\[
\begin{align*}
\text{SA1} & : Z \\
\text{SA2} & : Z+1 \\
\text{RJ} & : \text{CLOG} \\
\text{BX6} & : X1 \\
\text{LX7} & : X2 \\
\text{SA6} & : C \\
\text{SA7} & : C+1
\end{align*}
\]

**Normal Return**
The real part is in X1, the imaginary in X2. The complex result is stored in C.

**Error Message**
ARG(0.,0.)

**Storage**
146 locations

**Accuracy**
About \(10^{-14}\)

**Timing**
6600: 71 μsec 7600: 14 μsec

**Method**
\[
X + iY = \text{CLOG}(C + CI) \\
X = 0.5 \log_e(C^2 + CI^2) \\
Y = \text{ATAN2}(CI, C)
\]
COS

Note: See SIN, p. 2.58.

Purpose
To evaluate the cosine of a floating-point argument X, where X is in radians.

Fortran Function
Y = COS(X)

Entry Point
COS[COSF]

Ascent Calling Sequence
SA1 X
RJ COS
BX6 XL
SA6 Y

Normal Return
The result returned in XL is stored in Y.

Error Message
ARG TOO LARGE

Storage
658 locations

Accuracy
The relative error is about $10^{-14}$. Double-precision cosine is used as base.

Timing
6600: 33 μsec  7600: 7 μsec

Method*
Similar to SIN.
### CSIN

**Purpose**
To calculate the sine of a complex number.

**Fortran Function**
\[ R = \text{CSIN}(Z) \]

**Entry Point**
CSIN

**Externals**
COSF, SINF, EXPF

**Ascent Calling Sequence**
| SA1 | Z     |
| SA2 | Z+1   |
| RJ  | CSIN  |
| BX6 | X1    |
| SA6 | R     |
| LX7 | X2    |
| SA7 | R+1   |

**Normal Return**
The real part is in X1, the imaginary in X2. The complex result is stored in R.

**Error Message**
INFINITE OR INDEFINITE ARG

**Storage**
368 locations

**Accuracy**
The relative error is \(10^{-15}\)

**Timing**
6500: 121 µsec
7600: 23 µsec

**Method**
Where \(z = x + iy\)

\[
\text{csin}(z) = \sin(x) \frac{e^y + e^{-y}}{2} + i\cos(x) \frac{e^y - e^{-y}}{2}
\]
**CSQRT**

**Purpose**
To calculate the square root of a complex number Z.

**Fortran Function**
\[ X = \text{CSQRT}(Z) \]

**Entry Point**
CSQRT

**Ascent Calling Sequence**
- SA1  Z
- SA2  Z+1
- RJ   CSQRT
- BX6  X1
- SA6  X
- LX7  X2
- SA7  X+1

**Normal Return**
The real part is X1, the imaginary in X2. The complex result is stored in X.

**Error Message**
None

**Storage**
538 locations

**Accuracy**
At worst, \(10^{-13}\)

**Timing**
6600: 51 µsec  
7600: 11 µsec

**Method**
For \(Z = A + iB\)
\[ X + iY = (A+iB)^{\frac{1}{2}} \]
\[ X = \sqrt{\left((A^2 + B^2)^{\frac{1}{2}} + A\right)/2} \]
\[ Y = B/2X \]

Results are returned between \(-\pi/2\) and \(\pi/2\).
**CUBRT**

**Purpose**
To calculate the cube root of a real number.

**Fortran Function**
\[ X = \text{CUBRT}(Y) \]

**Entry Points**
CUBRT, CUBRTF

**Ascent Calling Sequence**

<table>
<thead>
<tr>
<th>SA1</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ</td>
<td>CUBRT</td>
</tr>
<tr>
<td>BX6</td>
<td>XL</td>
</tr>
<tr>
<td>SA6</td>
<td>X</td>
</tr>
</tbody>
</table>

**Normal Return**
The result returned in XL is stored in X.

**Error Message**
INDEFINITE OR INFINITE ARGUMENT

**Storage**
478 locations

**Accuracy**
The relative error is approximately \(10^{-14}\).

**Timing**
6600: 38 µsec  
7600: 7 µsec
Method*

\[ Y = 2^{3n+k} \cdot X, \text{ the first decomposition} \]
where \( \frac{1}{2} \leq X < 1 \) and \( n \) and \( k \) are integers, \( k = 0, 1, 2 \).

Then \( Z = \sqrt[3]{Y} = 2^n 2^{k/3} \sqrt[3]{X} \)

An initial approximation, \( B \), to \( \sqrt[3]{X} \) is obtained from the expression

\[
B = a_0 - \frac{a_1 (X + b_1)}{(X + b_1)(X + b_2) - a_2}
\]

with

- \( a_0 = 2.502926 \)
- \( a_1 = 8.045125 \)
- \( b_1 = 4.612244 \)
- \( a_2 = 0.3598496 \)
- \( b_2 = 0.3877552 \)

\( \sqrt[3]{X} \) is approximated by two Newton iterations using \( B \) for an initial guess.

\[
u = \frac{4}{9} \left( B + \frac{X}{2B^2} \right) + \frac{3}{4} \left[ \frac{X}{B + \frac{X}{2B^2}} \right]^{2}
\]

Finally, \( Z = 2^{n_2 k/3} \sqrt[3]{X} \approx 2^{n_2 k/3} u \)
DATAN

Purpose
To compute the double-precision arctangent of a double-precision argument X. Results are in radians where $-\pi/2 < Y < \pi/2$.

Fortran Function
\[ Y = DATAN(X) \]

Entry Point
DATAN

Ascent Calling Sequence
SA1 X
SA2 X+1
RJ DATAN
BX6 X1
LX7 X2
SA6 Y
SA7 A6+1

Normal Return
The most significant part is left in X1, the least significant in X2. The result is stored in Y.

Error Message
INDEFINITE OR INFINITE ARG

Storage
2028 locations

Accuracy
$10^{-28}$ (average)

Timing
6600: 136 μsec
7600: 24 μsec

Method
DATAN is computed from a polynomial telescoped from the Taylor-Maclaurin power series 39. The result is set to the argument where $-1.0 \times 10^{-10} < \text{arg} < 1.0 \times 10^{-10}$. 
**DATAN2**

**Purpose**
To compute in double-precision the angle in radians when the tangent is given in terms of coordinates X and Y, where X and Y are double-precision. The angle in radians is between $-\pi$ and \( \pi \) ($-\pi < \text{result} < \pi$).

**Fortran Function**
\[ R = \text{DATAN2}(Y, X) \]

**Entry Point**
DATAN2

**Ascent Calling Sequence**
\[
\begin{align*}
&\text{RJ} \quad \text{DATAN2} \\
&EQ \quad *+3 \\
&\text{CON} \quad Y \\
&\text{CON} \quad X \\
&BX6 \quad X1 \\
&LX7 \quad X2 \\
&\text{SA6} \quad R \\
&\text{SA7} \quad A6+1
\end{align*}
\]

**Normal Return**
The most significant part is left in X1, the least significant in X2. The result is stored in R. The result between $-\pi$ and \( \pi \) is in radians.

**Error Message**
\[ X = Y = 0 \text{ OR INDEFINITE OR INFINITE ARGUMENT} \]

**Storage**
2158 locations

**Accuracy**
Relative error about $10^{-28}$

**Timing**
6600: 146 usec 7600: 25 usec
DATAN2 establishes the correct quadrant based on the sign of the coordinates X and Y. It is computed from a polynomial telescoped from the Taylor-Maclaurin power series 39.

If $X = 0$, $R = \text{sign}(Y)\pi/2$

$X > 0$, $R = \text{DATAN}(Y/X)$

$X < 0$, $R = \text{DATAN}(Y/X) + \text{sign}(Y)\pi$

For values of $Y/X < 10^{-10}$, the result is set to the value of the argument.
**DBADEX**

**Purpose**
To calculate the D to the DD power, where D and DD are double-precision.

**Fortran Function**

\[ D2 = D^{DD} \]

**Entry Point**

DBADEX

**Ascent Calling Sequence**

- RJ DBADEX
- EQ \(*+3\)
- CON D
- CON DD
- BX6 X1
- LX7 X2
- SA6 D2
- SA7 D2+1

**Normal Return**
The most significant part is in X1, the least significant in X2. The double-precision result is stored in D2.

**Error Message**
NEG BASE WITH REAL EXPONENT

**Storage**
2178 locations

**Accuracy**
About \(10^{-25}\)

**Timing**
6600: 238 μsec  
7600: 40 μsec
DBAIEX

Purpose To calculate $D^I$ ($D^I$) where the argument $D$ is double-precision and $I$ is integer.

Fortran Function $DD = D^I$

Entry Point DBAIEX

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>EQ</th>
<th>CON</th>
<th>BX6</th>
<th>LX7</th>
<th>SA6</th>
<th>SA7</th>
</tr>
</thead>
<tbody>
<tr>
<td>*+3</td>
<td>D</td>
<td>I</td>
<td>X1</td>
<td>X2</td>
<td>DD</td>
<td>DD+1</td>
</tr>
</tbody>
</table>

Normal Return The most significant part is in X1, the least significant in X2. The double-precision result is stored in DD.

Error Message None; an overflow will be returned from the monitor.

Storage 358 locations

Accuracy The error obtained by comparing the result with that of an explicit succession of double-precision multiplies is $10^{-26}$.

<table>
<thead>
<tr>
<th></th>
<th>6600</th>
<th>7600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing For I = 2:</td>
<td>34 μsec</td>
<td>6 μsec</td>
</tr>
<tr>
<td>For I = 12:</td>
<td>50 μsec</td>
<td>8 μsec</td>
</tr>
</tbody>
</table>

Note: Time increases as I increases.

Method* Successive double-precision multiplies are taken. Where the exponent is negative the result is divided into 1.0 after the appropriate number of multiplies.
DBAREX

Purpose
To calculate the double-precision result of double base to real exponent.

Fortran Function
D = DD**R

Entry Point
DBAREX

Ascent Calling Sequence
RJ  DBAREX
EQ  #+3
CON  DD
CON  R
BX6  X1
LX7  X2
SA6  D
SA7  D+1

Normal Return
The most significant part is in X1, the least significant in X2. The double-precision result is stored in D.

Error Message
NEG BASE WITH REAL EXPONENT

Storage
2178 locations

Accuracy
About 10^-26

Timing
6600: 247 μsec  7600: 41 μsec
DCOS

Purpose
To calculate the double-precision cosine of a double-precision argument.

Fortran Function
Y = DCOS(Z)

Entry Point
DCOS

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ DCOS
BX6 X1
LX7 X2
SA6 Y
SA7 Y+1

Normal Return
The most significant part is in X1, the least significant in X2. The double-precision result is stored in Y.

Error Message
ABS(ARG) = PI X 2 TO 94

Storage
1618 locations

Accuracy
About $10^{-28}$

Timing
6600: 101 μsec 7600: 17 μsec

Method
Similar to DSIN.
2.38
Mathematical Library Routines

**DEXP**

**Purpose**
To compute the double-precision natural exponential of a double-precision argument \( D \).

**Fortran Function**
\[
Y = \text{DEXP}(D)
\]

**Entry Point**
DEXP

**Ascent Calling Sequence**
- SA1 D
- SA2 D+1
- RJ DEXP
- BX6 X1
- LX7 X2
- SA6 Y
- SA7 A6+1

**Normal Return**
The most significant part is in X1, the least significant in X2. The double-precision result is stored in Y.

**Error Message**
ARG GREATER THAN 743

**Storage**
1308 locations

**Accuracy**
Average is \( 10^{-29} \). Tests performed by Control Data for 5000 values of X uniformly distributed in the range \(|X| < \log_e(2)/2\) found the maximum observed relative error was 3.6E-29. For 5000 values < 600., it was 8.0E-29.

**Timing**
| 6600: | 117 μsec |
| 7600: | 19 μsec |
Method

Let $N = \left\lfloor \frac{X}{\log_e(2.0)} + 0.5 \right\rfloor$, and $R = R_1 + R_2 = X - N \log_e(2.0)$, $|R| \leq \frac{\log_e(2.0)}{2}$. $R_1$ and $R_2$ represent the most significant and least significant parts of $R$.

$$e^X = 2^N e^{R_1} e^{R_2}$$

$e^{R_1}$ is evaluated from a polynomial of degree 17. This polynomial was telescoped from a truncated Maclaurin power series 20.

$$e^{R_2} = 1 + R_2$$
### DLOG

**Purpose**
To compute the double-precision natural logarithm of a double-precision argument D.

**Fortran Function**
\[ Y = \text{DLOG}(D) \]

**Entry Point**
DLOG

**Ascent Calling Sequence**
- \( \text{SA1} \quad D \)
- \( \text{SA2} \quad D+1 \)
- \( \text{RJ} \quad \text{DLOG} \)
- \( \text{BX6} \quad X1 \)
- \( \text{LX7} \quad X2 \)
- \( \text{SA6} \quad Y \)
- \( \text{SA7} \quad Y+1 \)

**Normal Return**
The most significant part is in X1, the least significant in X2. The double-precision result is stored in Y.

**Error Message**
ARG ZERO OR NEGATIVE

**Storage**
1568 locations

**Accuracy**
Average is \(10^{-29} \). In tests done by Control Data for 2000 values of X uniformly distributed in the range \(0.5 \leq X \leq 2\), the maximum observed absolute error was 2.4E-29.

**Timing**
- 6600: 133 μsec
- 7600: 22 μsec
Method

\[ x = 2^W \quad 0.5^W < W < 2^W \]

\[ \log_e (x) = K \log_e 2 + \log_e W \]

\( \log_e W \) is approximated by the equation

\[ \log_e W = C_1 t + C_3 t^3 + C_5 t^5 + C_7 t^7 \]

where \( t = (W-1)/(W+1) \).

If \( X \leq 0 \), an error exit is taken and a diagnostic printed.
DLOG10

Purpose
To compute the logarithm of a double-precision argument to the base 10.

Fortran Function
DD = DLOG10(D)

Entry Point
DLOG10

Ascent Calling Sequence
SA1 D
SA2 D+1
RJ DLOG10
BX6 X1
LX7 X2
SA6 DD
SA7 DD+1

Normal Return
The most significant part is in X1, the least significant in X2. Double-precision results are stored in DD.

Error Message
ARG ZERO OR NEGATIVE

Storage
1648 locations

Accuracy
About $10^{-28}$. The argument is compared with the double-precision $10^{|\text{arg}|}$. It is an absolute difference.

Timing
6600: 139 μsec  7600: 23 μsec

Method
\[ \log_{10}(x) = \log_{10}(e) \log_{e}(x) \]
DMAX1/DMIN1

Purpose
To calculate the maximum (minimum) of n arguments.

Fortran Function
D1 = DMAX1(D1,D2,...,Dn)
DD = DMIN1(D1,D2,...,Dn)

Entry Points
DMAX1, DMIN1

Ascent Calling Sequence
For D = DMAX1(D1,D2,D3)

<table>
<thead>
<tr>
<th>RJ</th>
<th>DMAX1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>#+4</td>
</tr>
<tr>
<td>CON</td>
<td>D1</td>
</tr>
<tr>
<td>CON</td>
<td>D2</td>
</tr>
<tr>
<td>CON</td>
<td>D3</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>BX7</td>
<td>X2</td>
</tr>
<tr>
<td>SA6</td>
<td>D</td>
</tr>
<tr>
<td>SA7</td>
<td>D+1</td>
</tr>
</tbody>
</table>

Normal Return
The double-precision result is returned in X1 and X2.

Error Message
None

Storage
258 locations

Accuracy
The formula is exact.

Method*
The most significant portion is checked for a maximum (minimum). If the difference between two arguments is zero, then the least significant portion is checked.
DMOD

Purpose
To calculate the value of the double-precision argument D1 modulo D2.

Fortran Function
D = DMOD(D1,D2)

Entry Point
DMOD

Ascent Calling Sequence
RJ DMOD
EQ *+3
CON D1
CON D2
BX6 X1
BX7 X2
SA6 D
SA7 D+1

Normal Return
The most significant part is in X1, the least significant in X2. Double-precision results are stored in D.

Error Message
INDEF OR INF ARG, X2 = 0, OR INTEGER PART TOO LARGE

Storage
438 locations

Accuracy
The formula is exact.

Timing
6600: 51 µsec 7600: 8 µsec

Method*
D = D1 - \left( \frac{D1}{D2} \right) D2

where \frac{D1}{D2} = integral part
**DSIGN**

**Purpose**
To transfer the sign of argument D2 to the absolute value of argument D1.

**Fortran Function**
\[ D = \text{DSIGN}(D1, D2) \]

**Entry Point**
DSIGN

**Ascent Calling Sequence**

<table>
<thead>
<tr>
<th>RJ</th>
<th>DSIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>*+3</td>
</tr>
<tr>
<td>CON</td>
<td>D1</td>
</tr>
<tr>
<td>CON</td>
<td>D2</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>BX7</td>
<td>X2</td>
</tr>
<tr>
<td>SA6</td>
<td>D</td>
</tr>
<tr>
<td>SA7</td>
<td>D+1</td>
</tr>
</tbody>
</table>

**Normal Return**
The most significant part is in X1, the least significant in X2. The double-precision result is stored in D.

**Error Message**
None

**Storage**
148 locations

**Accuracy**
The formula is exact.

**Timing**
6600: 17 μsec 7600: 3 μsec

**Method***
The sign of the second argument D2 is assigned to both the least and most significant parts of D1 without regard to the sign of D1.
DSIN

Purpose
To calculate the double-precision sine of a double-precision argument.

Fortran Function
Y = DSIN(Z)

Entry Point
DSIN

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ DSIN
BX6 X1
LX7 X2
SA6 Y
SA7 Y+1

Normal Return
The most significant part is in X1, the least significant in X2. The double-precision result is stored in Y.

Error Message
ABS(ARG) = PI X 2 TO 94

Storage
1618 locations

Accuracy
About $10^{-28}$. This is relative error compared with the first three terms of the Taylor series.

Timing
6600: 91 µsec 7600: 16 µsec

Method
$N = \frac{x}{\pi/2} + .5$
$R = x - \frac{N\pi}{2}$, then $|R| < \frac{\pi}{4}$
$K = N(\text{MOD}4)$
Then $\sin(X) = \sin\left(R + \frac{N\pi}{2}\right)$
$= \sin(R)\cos\left(\frac{K\pi}{2}\right) + \cos(R)\sin\left(\frac{K\pi}{2}\right)$
Polynomials were telescoped using truncated Taylor-Maclaurin power series 25 and 24.
**DSQRT**

**Purpose**
To calculate the double-precision square root of a double-precision argument.

**Fortran Function**

\[ DD = \text{DSQRT}(D) \]

**Entry Point**

DSQRT

**Ascent Calling Sequence**

SA1  D  
SA2  D+1  
RJ  DSQRT  
BX6  XL  
LX7  X2  
SA6  DD  
SA7  DD+1

**Normal Return**
The most significant part is in X1, the least significant in X2. Double-precision results are stored in DD.

**Error Message**
NEGATIVE ARG

**Storage**
508 locations

**Accuracy**

\[ 10^{-27} \]. This is percent relative error compared with its square.

**Timing**

6600: 43 μsec  
7600: 7 μsec

**Method**
Two Newton iterations are done in single precision and two more in double precision.
**EXP**

**Purpose**
To calculate the exponential of a floating-point argument. The range of the argument is \( x \leq 741.667 \).

**Fortran Function**
\[ Y = \text{EXP}(X) \]

**Entry Point**
EXP(EXPF)

**Ascent Calling Sequence**
\[
\begin{align*}
\text{SA}1 & \quad X \\
\text{RJ} & \quad \text{EXP} \\
\text{BX6} & \quad \text{XL} \\
\text{SA6} & \quad Y
\end{align*}
\]

**Normal Return**
The results returned in XL are stored in Y.

**Error Message**
ARG IN EXP GT 741

**Storage**
568 locations

**Accuracy**
The maximum relative error is \( 1.3E-14 \) with roughly half the errors above and half the errors below zero.

**Timing**
6600: 20 µsec    7600: 4.4 µsec
Method* 4

\[ e^x = 2^y \quad \text{where} \quad y = x \log_2(e) \]

\[ 2^y = 2^{n+z} = 2^n \cdot 2^z \quad \text{with the integer} \quad n \quad \text{and} \quad |z| \leq \frac{1}{2} \]

For \( |z| \leq \frac{1}{2} \),

\[ 2^z \approx \frac{Q(z^2) + zP(z^2)}{Q(z^2) - zP(z^2)} \]

where \( Q(w) = w^2 + Q_1 w + Q_0 \) and \( P(w) = P_2 w^2 + P_1 w + P_0 \)

with coefficients:

\[ Q_0 = 4.368211662727558E+3 \]
\[ Q_1 = 2.331842114274816E+2 \]
\[ P_0 = 1.513906799054339E+3 \]
\[ P_1 = 2.020206565128693E+1 \]
\[ P_2 = 2.309334775375023E-2 \]

---

**IBAIEX**

**Purpose**
To calculate the integer result of integer base to integer exponent ($I^J$).

**Fortran Function**

\[ K = I^J \]

**Entry Point**

IBAIEX

**Ascent Calling Sequence**

\[
\begin{align*}
& RJ \quad \text{IBAIEX} \\
& EQ \quad \ast+3 \\
& \text{CON} \quad I \\
& \text{CON} \quad J \\
& \text{BX6} \quad X1 \\
& \text{SA6} \quad K
\end{align*}
\]

**Normal Return**

The results returned in X1 are stored in K.

**Error Message**

0 TO 0TH POWER OR RESULT EXCEEDS 2 TO THE 48TH

**Storage**

256 locations

**Accuracy**

The method is exact.

**Timing**

<table>
<thead>
<tr>
<th></th>
<th>6600</th>
<th>7600</th>
</tr>
</thead>
<tbody>
<tr>
<td>For an exponent of 3:</td>
<td>30 μsec</td>
<td>5 μsec</td>
</tr>
<tr>
<td>For an exponent of 12:</td>
<td>48 μsec</td>
<td>6 μsec</td>
</tr>
</tbody>
</table>

**Method**

Form $I^J$ by successive multiplications of the base.

- If $J = 0$, $I \neq 0$, set to 1.
- If $J = 0$, $I = 0$, error exit.
- If $J \neq 0$, $I = 0$, set to 0.

**Note:** If $J$, the exponent, is a constant and equals zero, the compiler will generate code which sets the result to 1. No error exit will occur, as there is no entry to IBAIEX in this case.
IDINT

Purpose
To convert the double-precision argument to a floating-point integral argument.

Fortran Function
A = IDINT(D)

Entry Point
IDINT

Ascent Calling Sequence
SA1 D
SA2 D+1
RJ IDINT
BX6 X1
SA6 A

Normal Return
The result returned in X1 is stored in A.

Error Message
None

Storage
258 locations

Accuracy
The method is exact.

Timing
6600: 23 μsec 7600: 4 μsec

Method*
The argument is shifted to integer and the least significant part of the integer is added to get an exact result as a full word integer, which is packed and normalized.

Example
5.43169999999999D+13 converts to 54317000000000.00
A floating-point integral number is returned.
RANF

Purpose
To generate a uniformly distributed sequence of random numbers between 0 and 1 in floating-point format, by repeatedly using RANF as a function for the set. Call RANSET as a subroutine to change the generating number.

Fortran Function
X = RANF(A)

Entry Points
RANF, RANSET, RANGET

ENTRY POINT RANF

Ascent Calling Sequence
RJ RANF
BX6 X1
SA6 X

Normal Return
The results returned in X1 are stored in X.

Error Message
None

Storage
264 locations

Accuracy
The $\chi^2$ test on 10,000 numbers was satisfactory.

Timing
6600: 12 μsec 7600: 2 μsec

Method*
Number_{i+1} = Number_i \cdot K$ where $Number$ and $K$ are packed as floating integers.

$$X = \text{Number}_{i+1} \pmod{2^{24}}$$

$K = 2^{24} - 3$

Argument A is a dummy and is not used by the function. Use a floating-point argument to avoid extra pack and normalize instructions added by the compiler; where RANSET has not previously been called, the generative number is set to 9.
ENTRY POINT RANSET

Fortran Call

CALL RANSET(I)

The generative number (NUMBER) is reset to the integer specified by the argument I. Following a CALL RANSET(I), the repeated use of \( X = \text{RANF}(1.0) \) generates a set of random numbers based on I. To repeat a set of random numbers, this generative number must be reset to the generative number of the original set. I should be an odd integer between 1 and 220. An octal constant may also be used as the argument.

Ascent Calling Sequence

RJ       RANSET
EQ       \#+2
CON      I

ENTRY POINT RANGET

Fortran Call

CALL RANGET(I)

The generative number currently in use by RANF is returned in I including its mask. Use an O20 format to print. To repeat a set of random numbers, this generative number may be reentered with or without the mask by a CALL to RANSET.

Ascent Calling Sequence

RJ       RANGET
EQ       \#+2
CON      I
RBAIEX

Purpose
To evaluate $X^I$, floating base to integer exponent.

Fortran Function
$Y = X^I$

Entry Point
RBAIEX

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>RBAIEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>$I+3$</td>
</tr>
<tr>
<td>CON</td>
<td>X</td>
</tr>
<tr>
<td>CON</td>
<td>I</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>SA6</td>
<td>Y</td>
</tr>
</tbody>
</table>

Normal Return
The results returned in X1 are stored in Y.

Error Message
ZERO BASE TO ZERO OR NEG EXPONENT

Storage
208 locations

Accuracy
The method is exact. $N = \log_2 |I|$ is an upper bound for the number of binary bits lost due to rounding error.

Timing

<table>
<thead>
<tr>
<th></th>
<th>6600</th>
<th>7600</th>
</tr>
</thead>
<tbody>
<tr>
<td>For an exponent of 3:</td>
<td>24 µsec</td>
<td>4 µsec</td>
</tr>
<tr>
<td>For an exponent of -8:</td>
<td>29 µsec</td>
<td>5 µsec</td>
</tr>
</tbody>
</table>

Method

$I = \sum_{i=0}^{n} a_i 2^i \quad a_i = 1 \text{ or } 0$

then

$X^I = \prod_{i=0}^{n} X a_i 2^i$

Form $X a_i 2^i \quad i = 1,n$ by successive squarings and multiply into the result if $a_i = 1$ in the binary configuration of I.
RBAREX

Purpose
To calculate \( X^Y \) where the arguments \( X \) and \( Y \) are real.

Fortran Function
\( C = X^Y \)

Entry Point
RBAREX

Ascent Calling Sequence
\[
\begin{align*}
&\text{RJ} \quad \text{RBAREX} \\
&\text{EQ} \quad \#+3 \\
&\text{CON} \quad X \\
&\text{CON} \quad Y \\
&\text{BX6} \quad XL \\
&\text{SA6} \quad C \\
\end{align*}
\]

Normal Return
The result returned in XL is stored in C.

Error Message
EXponent OVERFLOW OR Neg Base IN X TO Y

Storage
658 locations

Accuracy
When \( x^y \) is in the range \([0.1, 1.0]\) then the relative error is approximately \(< 10^{-14}\). When \( x^y \) becomes very small, \(< 10^{-100}\), or very large, \(> 10^{100}\), then the relative error can be on the order of \(10^{-12}\).

This error behavior can be seen as follows. The quantity \( z = x^y \) is computed as \( 2^y \log_2 x \). If we compute the quantity \( y' = y \log_2 x \) with relative error, \( \varepsilon \), then we wish to determine the relative error, \( \delta \) of \( z = 2^{y'} \);
i.e.,
\[2y'(1+\varepsilon) = z(1+\delta)\]
\[2y' 2y - z = \delta z\]
\[2y'(2y - 1) = \delta z\]

but,
\[z = 2y\]

thus,
\[\delta = 2y' - 1 = \varepsilon y' \ln_2 e - 1\]
\[\delta \sim 1 + \varepsilon y' \ln_2 e - 1 = \varepsilon y' \ln_2 e\]

The largest that \(y'\) can become is roughly \(\pm 1000\), thus
\[|\delta| \leq 1000 |\varepsilon|\]

If \(|\varepsilon| < 10^{-14}\), then \(|\delta| < 10^{-11}\). Thus, the relative error, \(\delta\), in \(z = x^y\) depends on the magnitude of \(z\).

Timing

6600: 57 µsec 7600: 11.5 µsec

Method*

Special properties:

if \(x = 0\), then \(x^y = 0\)
if \(y = 0\), then \(x^y = 1.0\)
if \(x = 1.0\), then \(x^y = 1.0\)
if \(x^y\) is less than the machine range, then \(x^y = 0\)
if \(x < 0\) or \(x^y\) is greater than the machine range, an error return is generated.
Method* Approximation Method:

\( x^y \) is calculated by \( 2^{y \log_2 x} \). Thus, two approximations are needed; one for \( \log_2 x \) and one for \( 2^y \). To calculate \( \log_2 x \) a range reduction is first performed so that \( x = 2^{N-\frac{1}{2}} S \), \( S \in \left[ \frac{1}{\sqrt{2}}, \sqrt{2} \right] \); then the approximation:

\[
\log_e S \sim z \frac{P(z^2)}{Q(z^2)} \quad \text{where} \quad z = \frac{S-1}{S+1} \quad \text{is used.}
\]

The polynomials \( P(z^2) \) and \( Q(z^2) \) are given by

\[
P(z^2) = P_0 + P_1 z^2 + P_2 z^4
\]
\[
Q(z^2) = Q_0 + Q_1 z^2 + z^4
\]

where

\[
P_0 = 8.2895584560451297
\]
\[
P_1 = -6.49111394895693
\]
\[
P_2 = 0.573146236434104
\]
\[
Q_0 = 4.14477924280070217
\]
\[
Q_1 = -4.627148777339487
\]

Then \( \log_2 x = N-\frac{1}{2} + \log_e \log_e S \).

Letting \( y' = y \log_2 x \), a range reduction is performed to obtain \( y' = N + 2z \) where \( z \in [0, \frac{1}{2}] \), and the approximation:

\[
z^2 \sim \frac{Q(z^2) + zP(z^2)}{Q(z^2) - zP(z^2)}
\]

\[\text{These approximations are taken from The SIAM Series in Applied Mathematics - Computer Approximations, and are approximation number 2703 for } \log_e x \text{ and approximation number 1065 for } 2^x.\]
is used. The polynomials $P(z^2)$ and $Q(z^2)$ are given by

\[ P(z^2) = P_0 + P_1 z^2 \]
\[ Q(z^2) = Q_0 + Q_1 z^2 + z^b \]

where

\[ P_0 = 2525.0425576241933744 \]
\[ P_1 = 28.875563776168927289 \]
\[ Q_0 = 7285.7336028361108885189 \]
\[ Q_1 = 375.021654220866600213 \]

Finally, one has

\[ x^y \sim 2^N \cdot 2^z \]
**SIN**

*Note:* See also COS, p. 2.26.

**Purpose**
To evaluate the sine of a floating-point argument \( X \), where \( X \) is in radians.

**Fortran Function**
\( Y = \text{SIN}(X) \)

**Entry Point**
SIN[SINF]

**Ascent Calling Sequence**

\[
\begin{align*}
\text{SA1} & \quad X \\
\text{RJ} & \quad \text{SIN} \\
\text{BX6} & \quad XL \\
\text{SA6} & \quad Y
\end{align*}
\]

**Normal Return**
The result returned in \( XL \) is stored in \( Y \).

**Error Message**
ARG TOO LARGE

**Storage**
658 locations

**Accuracy**
The relative error is about \( 10^{-14} \). Double-precision sine is used as base.

**Timing**

\[
\begin{align*}
6600: & \quad 33 \ \mu \text{sec} \\
7600: & \quad 7 \ \mu \text{sec}
\end{align*}
\]
Method* Let $N = \left\lfloor \frac{X}{\pi/2} + .5 \right\rfloor$

$R = X - N\pi/2$

$|R| \leq \pi/4$

$K = N$ modulo 4, thus $K = 0, 1, 2, \text{ or } 3.$

Then

$$\sin (x) = \sin(R + N\pi/2)$$
$$= \sin(R + K\pi/2)$$
$$= \sin(R)\cos(K\pi/2) + \cos(R)\sin(K\pi/2)$$

Polynomials for $R$ were telescoped from a truncated Taylor-Maclaurin power series 14 and 15.

The argument is first reduced to $X \leq \pi/4$. Before reduction, for large values of $X (> 1000 \text{ radians})$, increasing loss of accuracy appears in both the sine and cosine. This effect is inherent in the behavior of these functions when the argument has a fixed number of significant digits. Because of the periodicity, an integral number of periods is subtracted from the argument before evaluation; the number of significant digits in the remainder is thus approximately equal to the number of digits in the argument less than the number of digits in the integer part. The significance of the function value is approximately that of the remainder.
**SNGL**

**Purpose**
To convert a double-precision argument to a single-precision result which has been rounded.

**Fortran Function**
\[ R = \text{SNGL}(D) \]

**Entry Point**
SNGL

**Ascent Calling Sequence**
- SA1: D
- SA2: D+1
- RJ: SNGL
- BX6: X1
- SA6: R

**Normal Return**
The most significant part is returned in X1 and stored in R as a single-precision number.

**Error Message**
None

**Storage**
108 locations

**Accuracy**
The method is exact.

**Timing**
- 6600: About 10 μsec
- 7600: 2 μsec

**Method**
The most and least significant parts of D are added using the round floating-add to effect a round. A zero test on the argument (D) sets X1 to zero if D = 0.
**SQRT**

**Purpose**
To evaluate the square root of a real number.

**Fortran Function**
\[ Y = \text{SQRT}(X) \]

**Entry Point**
SQRT[SQRTF]

**Ascent Calling Sequence**
- SA1 X
- RJ SQR
- BX6 X1
- SA6 Y

**Normal Return**
The result returned in X1 is stored in Y.

**Error Message**
NEGATIVE ARG IN SQRT

**Storage**
318 locations

**Accuracy**
Average relative error is \(10^{-13}\).

**Timing**
6600: 21 µsec 7600: 4 µsec
Method
For $X > 0$

$.5 < W < 1$

$k = 0$ or integer with same sign as $r$

$r = -1, 0, +1$

$X = 2^k r W$

$\sqrt{X} = 2^k 2^{r/2} W^{3/2}$

The initial approximation to $W^{3/2}$ is

$B = .585786W + .420495$

Let $C$ be the result of two Newton iterations using $B$ as the initial guess for $W^{3/2}$. Then

$4C = \frac{(B^2 + W)^2 + 4B^2W}{B(B^2 + W)}$

$D = 2X^{3/2} = 2^{K-1} 2^{r/2} (4C)$

$Y = \sqrt{X} \approx .25D + X/D$

Note: If $Y_1 = B$ (linear approximation to $B$ given above), then

$Y_2 = B - \frac{(B^2 - W)}{2B} = \frac{B^2 + W}{2B}$

$C = Y_3 = \frac{B^2 + W}{2B} - \left[ \frac{(B^2 + W)^2 - W}{2 \left( \frac{B^2 + W}{2B} \right)} \right]$
**TAN**

**Purpose**
To calculate the tangent of X in radians.

**Fortran Function**
Y = TAN(X)

**Entry Point**
TAN[TANF]

**Ascent Calling Sequence**
SA1   X
RJ    TAN
BX6   X1
SA6   Y

**Normal Return**
The result returned in X1 is stored in Y.

**Error Message**
None

**Storage**
728 locations

**Accuracy**
Approximately $10^{-14}$. In the worst case, near 90°, it is about $10^{-10}$.

**Timing**
6600: 36 μsec  7600: 7 μsec
TAN
(Continued)

Method*

To find the tangent of $x$ the argument is first reduced as follows:

$x > 2\pi$, $x' = x - \lfloor x/2\pi \rfloor 2\pi$; $\tan(x) = \tan(x')$

where $\lfloor \rfloor$ is greatest integer notation.

$x' > \pi$, $x'' = 2\pi - x$; $\tan(x') = -\tan(x'')$

$x'' > \pi/2$, $x''' = \pi - x$; $\tan(x'') = -\tan(x''')$

$x''' > \pi/4$, $x'''' = x - \pi/2$; $\tan(x''') = -\cotn(x''')$

where $\cotn(x) = \frac{1}{\tan(x)}$.

Then

$$\tan(x) \approx \frac{2027025.x - 270270.x^3 + 6930.x^5 - 36x^7}{2027025. - 945945.x^2 + 51975.x^4 - 630.x^6 + x^8}$$

which is derived from the continued fraction

$$\tan(x) = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{4 - \frac{x^2}{5 - \frac{x^2}{6 - \ldots}}}}}$$
TANH

Purpose  
To calculate the hyperbolic tangent of X.

Fortran Function  
Y = TANH(X)

Entry Point  
TANH[TANHF]

Ascent Calling Sequence  
SA1 X
RJ TANH
BX6 X1
SA6 Y

Normal Return  
The result returned in X1 is stored in Y.

Error Message  
None

Storage  
608 locations

Accuracy  
About $10^{-12}$. The error increases rapidly to $10^{-8}$ and worse as the argument approaches zero between $-10^{-4}$ and $10^{-4}$. This is percent relative error.

Timing  
6600: 53 µsec  7600: 9 µsec

Method  
TANH(0) = 0

\[
\begin{align*}
\text{TANH}(X) &= \frac{e^{2X} - 1}{e^{2X} + 1} \quad \text{for } |X| < 30. \\
\text{TANH}(X) &= \pm 1 \quad \text{for } |X| \geq 30.
\end{align*}
\]
**TIMEF**

**Purpose**
To obtain the current reading of the clock in milliseconds.

**Fortran Function**
\[ Z = \text{TIMEF}(1.0) \]

**Entry Point**
TIMEF

**Ascent Calling Sequence**
- RJ
- BX6
- SA6
- X1
- Z

**Normal Return**
The result returned in X1 is stored in Z.

**Error Message**
None

**Storage**
7 locations

**Method**
The central monitor reads the total elapsed central processing unit (CPU) time in milliseconds, and the value is packed and normalized. Argument 1. is a dummy variable.

*Note:* Even though the argument is a dummy, it should be floating-point. Otherwise, the compiler will pack and normalize it, and these instructions are not necessary.
DESCRIPTION

A number of miscellaneous subroutines are included in the library. Since they are of such varied character, no attempt is made to classify them. Instead, the use of each one is described briefly.

A subroutine communicates with the calling program by means of parameters in the list defining the call, or by means of common. The formal parameters may be variables or arrays and are call-by-name parameters; i.e., addresses rather than values.

A call is compiled as follows:

```
CALL ROUT(A,B)
```

```
RJ ROUT
EQ *+3
CON A
CON B
```

In this example, the addresses of A and B are stored in the locations following the RJ to the subroutine. Answers are returned by storing results in the names in the parameter list (or common). The compiler generates a table of addresses for up to 64 parameters. The compiler calls a subroutine and the subroutine initializes the parameter addresses given in the subprogram instruction. If the parameter addresses do not change, this procedure is done only on the first call to that particular subroutine; the addresses remain in the instruction word for any subsequent calls.
3.2 Utility Library Subroutines

**DESCRIPTION**

A list of the subroutines described in this section follows.

- ACOER
- BACKSP (entry points: IFENDF, REWINM)
- BRANRD (entry points: BRANWT, BRANCK, BRANST)
- BUFFEI (entry points: BUFFEO, IOCHEK, LENGTHF)
- BUFRD (entry points: BUFINT, BUFWT, BUFCL)
- DEBUG
- DUMP
- ENDFIL
- EXIT (entry points: END, STOP)
- GBYTE (entry points: SBYTE, GBYTES, SBYTES)
- INPUTB (entry point: OUTPTB)
- INPUTC
- INPUTS
- IOPROC (entry points: BSTAPE, RDTAPE, WRTAPE, IOWAIT)
- JOBID
- KODER
- KRAKER
- MACHINE
- NEWLAB
- OUTPTC
- OUTPTS
- OVERLAY
- PAUSE
- PDUMP
- PLIBRD (entry points: PLIBREW, PLIBWT, PLIBCK)
- RANRD (entry points: RANINT, RANWT, RANOUT, RANIN, RANCK)
- RPTIN (entry point: RPTOUT)
- SAVEF
- SSW (entry point: SLT)
- SYSRCL
- TAPECY (see Chapter 6 for description)
- ULIB (entry points: ULIBREW, ULIBRD, ULIBCK)
- UNLOAD
ACGOER

Purpose

To output an error message for an incorrectly computed GO TO statement.

Error Message

ERROR IN COMPUTED GO TO

Storage

3 locations

Method

A branch to this routine is compiled in Fortran programs using a computed GO TO statement.
3.4 Utility Library Subroutines

**BACKSP**

**Entry Points** BACKSP, IFENDF, REWINM

**Storage** 1258 locations

**ENTRY POINT BACKSP**

**Purpose** To backspace one Fortran record on a tape. The unit number is contained in X1.

**Fortran Reference** BACKSPACE i

**Ascent Calling Sequence** SAL I

**Error Message** POSITIONING ERROR DURING BACKSPACE. This occurs when trying to backspace a buffered record when the program is anticipating a Fortran record.
ENTRY POINT IFENDF

Purpose
To check for an end-of-file (EOF) on unit i of the previous input/output (I/O) statement. The branch is to \( n_1 \) for an EOF, and to \( n_2 \) if no EOF is encountered.

Fortran Reference
\[
\text{IF(EOF,i) } n_1, n_2
\]

Ascent Calling Sequence
\[
\text{SA1 I RJ IFENDF ZR X1, statement } n_2 \text{ EQ B0, B0, statement } n_1
\]

ENTRY POINT REWINM

Purpose
To rewind the tape to loadpoint.

Fortran Reference
\[
\text{REWIND i}
\]

Ascent Calling Sequence
\[
\text{SA1 I RJ REWINM}
\]
BRANRD

**Purpose**

To provide a buffered I/O operation from the random file to the drum (disk). The routine has true buffering capability on two channels.

**Entry Points**

BRANRD, BRANWT, BRANCK, BRANST

**Error Messages**

- ATTEMPTED ANOTHER OPERATION WITHOUT CHECKING LAST
- CANNOT RESET FILE
- CANNOT SET UP FILE
- EOF ENCOUNTERED
- RECORD LENGTH MUST BE GT 0
- UNSUCCESSFUL READ OR WRITE

**Storage**

242 locations

**Method**

BRANRD sends even-named records to one of the two random files available and odd-named records to the other. An even name has bit $0 = 0$; an odd name has bit $0 = 1$. Most alpha-numeric names will be odd-named files because the word will be blank-filled and bit $0 = 1$. Integer names should have type integer variable names; if these are floated, the pack and normalize will force all integers to be even, since bit $0 = 0$. Two channels transmit records simultaneously.

The BRANRD routine is preferred to RANRD (p. 3.85). Any record can be written, read, or rewritten at random to the file. Each record may have a different length. The length of the record given on the first write must be the maximum length that will be used. The maximum number of file names is 200. If more are required, use the ULIB version of BRANRD, where the number of file names may be changed. See a systems programmer.
ENTRY POINT BRANRD

Purpose
To start a read.

Fortran Calling
CALL BRANRD(NAME, ARRAY, LENGTH)

Sequence
BRANRD starts a buffered read from the random record NAME into ARRAY. The length of the record is LENGTH. Before the array is used, BRANCK should be called to assure completion of the read. The arguments are:

NAME = Any collection of numbers or characters identifying the record. If bit 0 = 1, the name is odd; if bit 0 = 0, the name is even. This can be any 60-bit configuration except all 0's.

ARRAY = First word address in memory of the array to be transmitted.

LENGTH = Number of words to be transmitted.

ENTRY POINT BRANWT

Purpose
To start a write.

Fortran Calling
CALL BRANWT(NAME, ARRAY, LENGTH)

Sequence
BRANWT starts a buffered write from ARRAY to a random record called NAME. The length of the record written is LENGTH. Before destroying the array, BRANCK should be called to assure completion of the write. The arguments are:

NAME = Any collection of numbers or characters identifying the record. If bit 0 = 1, the name is odd; if bit 0 = 0, the name is even. This can be any 60-bit configuration except all 0's.

ARRAY = First word address in memory of the array to be transmitted.

LENGTH = Number of words to be transmitted.
ENTRY POINT BRANCK

Purpose
To check for completion of the I/O operation and for transmission error.

Fortran Calling Sequence
CALL BRANCK(NAME)

BRANCK will wait for completion of the I/O operation on the record called NAME. It will check the status of the operation. If the I/O was unsuccessful or the record was missing on the file, a message is printed and the program terminated. NAME is placed in X2 on termination. BRANCK can be called with a dummy name. If the record is not being read or written, or if the name is a dummy, the call is a NOP (no operation). The argument is:

NAME = NAME assigned to record.

Note: BRANCK should be called to complete the operation on the last even-named record before the next even-named record is read or written.
ENTRY POINT BRANST

Purpose
To test for the completion of an I/O operation on a record, without having to wait for completion.

Fortran Calling Sequence
CALL BRANST(NAME,NSTAT)

BRANST, an optional call, will test the I/O of the record called NAME. If the I/O is incomplete, the contents of NSTAT will be -1 on return. If the I/O is complete, the contents of NSTAT = 0 and BRANCK are automatically called by BRANST. The I/O is considered complete if the name is a dummy name used to call this routine before reading or writing. The arguments are:

NAME = Name assigned to record
NSTAT = -1, I/O not complete
= 0, I/O complete
BUFFEI

Entry Points BUFFEI, BUFFEO, IOCHEK, LENGTHF

Storage 2338 locations

ENTRY POINT BUFFEI

Purpose To initiate a read of a physical record.

Fortran Reference BUFFER IN (i,p)(A(1),A(100))

Ascent Calling Sequence For BUFFER IN (I,1)(A(1),A(100)):

| SB2 | I   |
| SB1 | 1   |
| RJ  | BUFFEI |
| SB7 | A   |
| SB2 | 0   |
| RJ  | BUFFEI |
| SB7 | A+144B |
| SB2 | 0   |
| RJ  | BUFFEI |

The tape unit number or the address is stored in B2, the parity code in B1, after which a RJ to BUFFEI is made. The starting address of the block is stored in B7 with another RJ. Finally, the last word address is stored in B7 with the last call to BUFFEI. B2 is set to zero after the first call.

Error Messages
- TAPE UNIT NUMBER NOT IN 1-6764B RANGE
- TOO MANY UNITS ASSIGNED
Utility Library Subroutines

Method

The length is given in the call, and one of two modes, BCD or binary (even or odd parity), may be specified. A wait (RECALL) is not issued with the call so that control may be returned to the caller, thus allowing programmer-designed buffering.

Limitation

- Buffering in an EOF on a tape will place a BCD 17 in the first buffer word.

ENTRY POINT BUFFEO

Purpose

To buffer out data.

Fortran Reference

BUFFER OUT (i,p)(A(l),A(100))

Ascent Calling Sequence

For BUFFER OUT (I,l)(A(1),A(100)):

SB2    I
SB1    1
RJ    BUFFEO
SB7    A
SB2    0
RJ    BUFFEO
SB7    A+144B
SB2    0
RJ    BUFFEO

Error Messages

TAPE UNIT NUMBER NOT IN 1-6764B RANGE
TOO MANY UNITS ASSIGNED
ENTRY POINT IOCHEK

Purpose
To complete the call made with a previous BUFFER OUT or BUFFER IN call.

Fortran Reference
IF(UNIT,i) n1,n2,n3,n4

Ascent Calling Sequence

SA1 I
RJ IOCHEK
NG X1,n1 (not ready--there is a wait)
ZR X1,n2 (ready)
SX1 X1-l
ZR X1,n3 (EOF)
EQ n4 (tape error)

The unit I is stored in X1 and a RJ is taken to IOCHEK. The return flag is X1.

X1 = 0 if unit is ready
X1 = negative if unit is not ready
X1 = 1 if EOF exists
X1 = 2 if tape error exists

Branches are taken on the value of X1 to statements n1, n2, n3, and n4. A RJ to this entry checks the status of the unit in buffered I/O.

Error Message
IF UNIT CALL BEFORE I/O CALL

Method
A RECALL is issued if the previous I/O operation is incomplete; no return is made unless the operation is complete. A status is returned immediately if the previous operation is complete.
3.13 Utility Library Subroutines

ENTRY POINT LENGTHF

Purpose
To return the number of words transferred in a previous BUFFER IN or BUFFER OUT call.

Fortran Function
\[ K = \text{LENGTHF}(I) \]

Ascent Calling Sequence
\[ \text{SA}1 \quad I \\
\text{RJ} \quad \text{LENGTHF} \\
\text{BX}6 \quad \text{XL} \\
\text{SA}6 \quad K \]

The address of the unit number is stored in XL and a RJ to LENGTHF is taken. The answer is returned in XL. This entry returns the number of words in a completed buffered I/O operation on Unit I.

Error Message
LENGTHF CALL BEFORE I/O CALL

Method
A RECALL is issued if the previous I/O operation is incomplete. The length is returned immediately if the previous I/O operation is complete.
Utility Library Subroutines

**BUFRD**

**Purpose**
To enable a user to design the data movement schemes for special I/O problems so that concurrent input and output can be maintained during a compute phase by providing an internal data processing system.

**Entry Points**
BUFINT, BUFWT, BUFRD, BUFCL

**Storage**
3368 locations

**Description**
According to the user's needs intermediate storage, called BUFFER, is provided in core, between the central memory and the peripheral drum memory. There are two buffers, one for input data and one for output data. Each buffer is divided in half and the data is passed alternately through the halves in a synchronized "flip-flop" technique.

These programs can be used to maintain concurrent input or output during a compute phase, only if the problem is one in which computations can be made on one set of data while the other set is being loaded or unloaded. Each set of computations must be of such length that all I/O can be completed. The following pages include descriptions of both input and output phases. It is assumed that initialization has been done.
Input Phase

Assume that 50 logical records have been placed on drum -A- during initialization phase. Drum -A- is then rewound and the procedure starts.

The program calls for Record 1, which moves into the first half of the input buffer. A wait condition exists until it is completely in. Record 2 then starts into the second half of the input buffer, but no wait condition exists. During this period instead, Record 1 moves into a designated array of the call, and control returns to the caller, who uses the data produced by Record 1 in his problem.

When the program calls for Record 2, checks are made to ascertain its status. If the compute phase for Record 1 was long enough, all of Record 2 will be in the input buffer. However, if Record 2 is still moving into the buffer, a wait condition will be activated. As soon as Record 2 is in the buffer, it moves into a designated array of the call, Record 3 starts into the first half of the input buffer, and control returns to the caller. This flip-flop procedure continues until all 50 records have moved into alternate buffer halves, have been used in the computation, and have moved to the designated arrays.
Output Phase

As soon as the calculation using Record 1 data is complete, the output phase begins. Assume that drum -B- has been rewound, and the program calls for the results of the first calculation, Result Set 1, to be put on the drum. (Input and output record sizes are the same.)

Result Set 1 moves into the first half of the output buffer from its designated array. Control returns immediately to the caller.

The next call is to move Result Set 2 onto the drum. At the same time Result Set 1 is moving to the drum; no wait condition exists at this point. Result Set 2 moves to the output buffer from its designated array, and control returns to the caller.

The next call is to move Result Set 3 onto the drum. If the checks made at this time indicate that all of Result Set 1 has moved out of the first half of the buffer onto the drum, Result Set 3 will begin its move to the drum. However, if the compute phase for Result Set 1 was not long enough to permit the move, a wait condition will exist until the move is completed. Result Set 3 then moves to the first half of the output buffer from its designated array, and control returns to the caller. This procedure continues until all 50 result sets have been loaded on drum -B-.

Rewind and Exchange Units

After all 50 records have been read, the computation sets completed, and the 50 result sets put on the drum -B-, the drum unit designations are rewound and exchanged. Drum -B- becomes the input unit, and drum -A- the output unit. The flip-flop procedure begins again, using the results calculated in the previous cycle as input.
ENTRY POINT BUFINT

Purpose
To initialize the program.

Fortran Calling
CALL BUFINT(EXYT,ENTRY,LENGTH,UNIT1,UNIT2)

Sequence
BUFINT must be called first, and is called only once. Two arrays, in addition to the data arrays, are required for buffering, and two logical unit numbers must be assigned to the drum. If the data array is dimensioned for 1000, then EXYT and ENTRY must be dimensioned for 2000 in the calling program.

Example


The parameters for BUFINT are:

EXYT An array which is twice the length of the data array that is to be moved to the drum memory.

ENTRY An array, equal in length to the exit array, which processes data from the drum memory.

LENGTH The number of elements in EXYT. (EXYT and ENTRY must have the same dimensions in the calling program.)

UNIT1 A logical unit number to be used for the data passing through the exit array to the drum. This number must appear on an ASSIGN card, e.g., *ASSIGN,DRUM=i.

UNIT2 A second logical unit number to be used for the data passing through the ENTRY array from the drum. This number must also appear on an ASSIGN card, e.g., *ASSIGN,DRUM=i+1.
ENTRY POINT BUFINT
(Continued)

Ascent Calling
Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>BUFINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>6</td>
</tr>
<tr>
<td>CON</td>
<td>EXYT</td>
</tr>
<tr>
<td>CON</td>
<td>ENTRY</td>
</tr>
<tr>
<td>CON</td>
<td>LENGTH</td>
</tr>
<tr>
<td>CON</td>
<td>UNIT1</td>
</tr>
<tr>
<td>CON</td>
<td>UNIT2</td>
</tr>
</tbody>
</table>

ENTRY POINT BUFWT

Purpose
To send the data array (called ARRAY) to the drum memory.

Fortran Calling
Sequence
CALL BUFWT(ARRAY,NERROR,NUNIT)

The BUFWT routine moves the data from the ARRAY into the exit array, and then to the drum memory. Alternate halves of the EXYT routine are used (in the flip-flop technique previously described) to move data to the drum. After two times through the loop, data are entering one half the EXYT array from ARRAY, while data are moving concurrently to the drum from the other half of the EXYT array.
The arguments are defined as follows:

**ARRAY**
The array containing data to be buffered.

**NERROR**

- **NERROR = 0** No write error.
- **NERROR ≠ 0** An error occurred during the previous attempt to move data to or from the drum. This number should be printed out for debugging purposes. This is a fatal error, and the user should terminate the subroutine.

**NUNIT**
The unit number on which the error occurred.

**Ascent Calling Sequence**

- **RJ** BUFWT
- **EQ** *
- **CON** ARRAY
- **CON** NERROR
- **CON** NUNIT

**ENTRY POINT BUFRD**

**Purpose**
To load the data array from the drum memory.

**Fortran Calling Sequence**

`CALL BUFRD(ARRAY,NERROR,NUNIT)`

The BUFRD routine moves the data from alternate halves of the ENTRY array into ARRAY and returns control to the caller. BUFRD then starts the next read from the drum memory to have data ready for the next call. (See Entry Point BUFWT for a definition of arguments.)
ENTRY POINT BUFCL

Purpose

To rewind the units assigned to the EXYT and ENTRY arrays.

Fortran Calling Sequence

CALL BUFFCL(N)

BUFCL must be called when repositioning is required. It is used in a manner similar to the ordinary rewind instruction in tape usage.

N = 0  The unit numbers are exchanged so that the input unit will be used for output and the output unit for input.

N = 1  The rewind takes place, but the unit numbers are not exchanged.

Ascent Calling Sequence

RJ  BUFCL
EQ  #+2
CON  N
In this sample, both the N1 and N2 arrays are used to make it possible to check that data are moving in and out correctly. BUFRD needs only one of these arrays, in addition to the exit and entry array, to do true buffering. The array N1 is initialized and moved to the drum before the buffering loop starts.
3.22
Utility Library Subroutines

Computer Sample 1
Program BUF
(continued)

0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095

000075
000075
000075
000075
000075
000111
000112
000117
000120
000120
000121
000122
000122
000122
000122
000122
000122
000122
000122
000136
000152
000152
000152
000152
000152
000152
000154
000160
000162
000164
000201
000201
000201
000202

-

RUFINT

C
C
C

CALL BUFCL (0)
70 CONTINUE
CALL EXIT
500 WRITE (6,501)NEX,J,I,NER,NUNI
501 FORMfT(iHi,10X,6HNEX = ,I2,3X,4HJ
16HNER = ,I4,3X,7HNUNI = ,12
)
CALL EXIT
END
BUF

SUBROUTINES CALLED
BUFWT
BUFCL

PROGRAM SPACF IS
FNTRY POINT
BUF
080ERR
EXIT
END
STOP
KODER
OUTPTC
BUFINT
BUFWT
BUFRD
BUFCL
CCMMONBLOCKS

BUFROD

NX
I
KE

- 017500C00
- 000215
- 000207

OUTPTC

061161

LOCATION
000004

= ,I2,3X,4HI = ,12,

000257

55

LOCATION
056704
057163
057175
057175
057175
057202
060224
060623
060646
060753
061067

= ,15,
= ,15)

REWIND AND EXCHANGE UNITS FOR COMPLETE NEW CYCLE

VARIABLE ASSIGNMENTS
000000000
N2
- 007640C00
000217
NMOX
- 000216
000211
NEX
- 000210
000203

COMPILE TIME=

ANC ERROR CHECK

44 CONTINUE
CALL BUFWT (N1,NER,NLNI)
IF (hER)45,48,45
45 NEX=45
GO TC 500
48 CONTINUE
KDIF=O
KEO=C
00 55 K=l,4000
IF (NI(K)-N2(K))50,52,50
50 KDIF=KDIF+i
GO TC 55
52 KEO=KEQ+l
55 CONTINUE
41 WRITE (6,43)KOI ,KE,KDIF,KEO
43 FORMAT (
5X,14HPASS I KDI
= ,I5,5X,6HKE
1
5X.14HPASS 2 KDIF = ,I5,SX,6HKEO
60 CONTINUE

LENGTH OF ROUTINE
N1
NXA
J
KEQ

35 KE =KE +1
42 CONTINUE
DEFINE A NEW N1 ARRAY

C
C
C

ROUTINE ORIGIN
056704
057163
057175
057175
057175
057202
060224
060623
060623
060623
060623

EXIT

NN
NER
KOI

- 037200C00
- 000214
- 000206

NU1
NUNI
K

- 000221
- 000213
- 000205

NU2
KP
KDIF

- 000220
- 000212
- 000204


3.24
Utility Library Subroutines

Computer Sample 1
Program BUF
(continued)
**DEBUG**

**Purpose**
To print the panel and a selective dump, during program execution, in any of the following three ways:

1. On the program instruction count (P count)
2. When a variable name occurs as an operand in the object code
3. When branch instructions appear in the object code between certain instructions

**Fortran Call**
CALL DEBUG

**Entry Point**
DEBUG

**Ascent Calling Sequence**
RJ DEBUG

**Error Message**
An error message will be printed if DEBUG is violated by a LOAD or STORE not within range. The program terminates.

*Note:* Addresses on snap cards outside the field length of the program are ignored. The dump will stop at the field limit address; no message is printed.

**Storage**
7438 locations
The call reads appropriate snap cards in the data set and forms an address table with the information read. If the call to DEBUG precedes the first read, the snap cards should be the first cards in the deck. Snap cards must appear in the data set wherever the call to DEBUG occurs.

To initiate the routine, DEBUG need be called only once. A second DEBUG call turns off the DEBUG option. The second call should be placed as soon as the last referenced address has been concluded. The portion of the program to be referenced by snap cards should be between two DEBUG calls.

*Note:* Introducing DEBUG slows the program down about 100 to 1. For example, a program that runs 70 sec without DEBUG will run about 7000 sec with DEBUG. To minimize program slowdown, the DEBUG call should be placed immediately preceding referenced addresses.
Snap Cards

All snap cards are followed by an END card which flags the last card in the snap set.

P Card

Format:

Col. 1
P

Indicates a snap on the P count during program execution

Col. 2
blank

Blank column

Col. 3-10
IDENT

The identifier printed to label which snap dump has occurred

Col. 11
SSSSSS LLLLL UUUUUU
SS LLLL UUUU

The snap dump addresses always start in Col. 11. The addresses are 1 to 6 characters long, and are separated by one blank.

SSSSSS

Snap dump when the P count reaches this address

LLLLLL
UUUUUU

Dump from this lower address to this upper address
3.28
Utility Library Subroutines

DEBUG
(Continued)

Snap Cards
(continued)

Format:

Col. 12345678...
    U IDENT SSSSS\(^1\) LLLLL UUUUU
    U IDENT SSSS LLLL UUUU
    U IDENT SS LLL UUUU
    END

Col. 1
    U
    Indicates an operand address (address of variable name) on which to dump

Col. 2
    blank
    Blank column

Col. 3-10
    IDENT
    Indicates which snap dump has occurred on output

Col. 11
    SSSSSS LLLLL LLLLL UUUUUU
    SSSS LLLL UUUU
    SSSSSS
    LLLLL
    UUUUUU
    The snap dump address always starts in Col. 11.
    The addresses may be 1 to 6 characters long, and are separated by blanks.
    Dump whenever this address appears in the instruction code as an operand
    Dump from this lower address to this upper address

\(^1\)If S = 0 or 1, the snap does not work correctly.
T Card

Format:

Col. 1
T

Col. 2
Z

Requests a trace on branch operation codes

If Z ≠ blank, a trace is performed on all branches. If Z = blank, a trace is performed only on those operations selected by non-blank characters in the X-field.

Col. 3
X ≠ blank
Trace on 01 branches

Col. 4
X ≠ blank
Trace on 02 branches

Col. 5
X ≠ blank
Trace on 03 branches

Col. 6
X ≠ blank
Trace on 04 branches

Col. 7
X ≠ blank
Trace on 05 branches

Col. 8
X ≠ blank
Trace on 06 branches

Col. 9
X ≠ blank
Trace on 07 branches

Col. 10
blank

Blank column

Col. 11
LLLLL UUUUU
LLLL UUU

The lower and upper addresses of the program code where the specified branch is to be traced
Computer Sample 2
Program TEST
Illustrating Use of DEBUG and Snap Cards

In this sample, the trace card has a 1 in Col. 2, so all branches between program instructions 4 and 11 will be searched.

The trace lines mean that 01 branches were taken from 6 to 144 (the entry point for INPUTC), 10 to 144, and 11 to 144. There were no other branch instructions in this range.

The SNAP2 U card used the operand 102 (4 + 76 is B in the program) at location 12. At 12 (relative) there was a SA7 B instruction, so at this point there was a snap dump on operand B.

The SNAP1 dump occurred at program location 77, as specified on the P card. The dump was from 4 to 114.

Before making up the snap cards, a *FORTRAN,L is needed to get the correct locations from the program listing.
This demonstrates the use of DEBUG. It must be called twice once to initialize, then later to turn the option off.

```fortran
C
C THIS DEMONSTRATES THE USE OF DEBUG. IT MUST BE CALLED TWICE 
C ONCE TO INITIALIZE, THEN LATER TO TURN THE OPTION OFF

COPPLEX C,D,R
INTEGER X,Y

READ (5,110) A
110 FORMAT (F10.1)
A=2.2
B=3.4
O=CMPLX(A,B)
C=(1.0,2.0)
A=2.2
9=3.4
O=CMPLX(A,B)
C=(1.0,2.0)
C=LOC FUNCTION
Y=4.635
X=LOC(Y)
Z=LOC(C)
PRINT 100,C,D
100 FORMAT (2B0,CMPLX*4F10.2)
E=AIMAG(D)
F=REAL(C)
R=CONJG(C)
PRINT 101,E,F,R
101 FORMAT (*OAIMAG,REAL,CONJG*4F10.1)
GO TO 8
END
```
Utility Library Subroutines

Computer Sample 2

Program TEST
(continued)
Utility Library Subroutines

LENGTH OF ROUTINE TEST 000121

DEBUG SUBROUTINES CALLED

END OF DEBUG CARDS

1-000006 TO 000144

SNAP1 P 000007 A0 000008 X8 7777777777777777
A1 000237 X1 0000000000000000
A2 000112 X2 7777777777777777
A3 000000 X3 0000000000000000
A4 000000 X4 0000000000000000
A5 000000 X5 0000000000000000
A6 000055 X6 0000000000000000
A7 000276 X7 0000000000000000

THE TOTAL PROGRAM AND BUFFER SPACE REQUIRED IS 003744

P SNAP1 07 4 111
U SNAP2 102 4 40
T1 4 111

END OF DEBUG CARDS
### DUMP

**Purpose**

To call a peripheral processor program which generates an octal core dump and then calls EXIT.

**Fortran Call**

CALL DUMP

**Entry Point**

DUMP

**Ascent Calling Sequence**

RJ DUMP

A RJ to DUMP is taken. There are no options and no argument list.

**Information Message on Output**

CENTRAL PROCESSOR DUMP

*Note:* This message is printed if there is no *TRAP card in the deck.

**Storage**

3 locations

**Method**

DUMP is written in ASCENT, and a *TRAP card must be included among the monitor control cards. DUMP output is always sent to the dd80.
**ENDFIL**

**Purpose**
To write an EOF on a specified tape unit.

**Fortran Reference**
END FILE i

**Entry Point**
ENDFIL

**Ascent Calling Sequence**
SA1 I
RJ ENDFIL

X1 contains the unit number

**Storage**
108 locations
EXIT

Purpose
To terminate a program or subprogram with a "normal" exit.

Fortran Call and References
CALL EXIT
STOP
END

Entry Points
EXIT, END, STOP

Error Message
None

Storage
5 locations

Method
A central monitor call to EXIT terminates the program.
GBYTE

Purpose To provide the capability to format in terms of numbers of bits rather than numbers of characters.

Entry Points GBYTE, GBYTES, SBYTE, SBYTES

Fortran Call CALL GBYTE(NPACK,ISAM,IBIT,NBITS)

Ascent Calling Sequence RJ GBYTE
               EQ $+5
               CON NPACK
               CON ISAM
               CON IBIT
               CON NBITS

Storage 1248 locations

Timing About 18 μsec per byte. This is about three and a half times as fast as ENCODE/DECODE for handling 6-bit characters, and about seven times as fast as ENCODE/DECODE when the latter uses an I4 format.
ENTRY POINT GBYTE

Purpose
To unpack bits or bytes from NPACK to ISAM.

Method
Call GBYTE(NPACK, ISAM, IBIT-offset, NBITS in byte) to get NBITS after skipping IBIT-offset bits in word NPACK.

Put the byte into ISAM right-adjusted, with the rest of cell ISAM set to zero. The maximum byte size is 60. The permissible range of IBIT-offset is 0-59.

Example
CALL GBYTE(NPA, ISA, 6, 6)

NPA 00770000000000000000000

ISA 00000000000000000000077
ENTRY POINT SBYTE

Purpose
To reverse the process that occurred in GBYTE.

Method
The byte size in NPACK is cleared and the byte put in; data surrounding the packed byte are not affected.

Example
DIMENSION NA(2)
CALL SBYTE(NA,NB,56,6)

\[
\begin{array}{c|c}
\text{NB} & 00000000000000000000000077 \\
\hline
\text{NA(1),NA(2)} & 00000000000000000017 600000000000000000000000 \\
\end{array}
\]

In binary the two words NA(1) and NA(2) look like this:

\[
\begin{array}{c|c}
\text{NA(1)} & \text{NA(2)} \\
\hline
\text{binary} & 0 \ldots 0000001111 1100000000 \ldots 0 \\
\text{octal equivalent} & 1 7 6 \\
\end{array}
\]

In the example, the offset is 56 bits and byte size is 6 bits, so that the total number of bits (62) is greater than one word (60 bits). Since word boundaries are ignored in packing bytes, SBYTE continues packing into the next word.
ENTRY POINT GBYTES

Purpose
To get an ITER number of bytes from NPACK into the ISAM array.

Method
Call GBYTES(NPACK,ISAM,IBIT-offset,NBITS,NSKIP,ITER). After the first byte, specified as in GBYTE, there is a skip of NSKIP bits. The next byte (of the same size and the same skip) then begins, and so on. The maximum byte size is 60 bits, but the skip can be longer than 60 bits. The permissible range of IBIT-offset is 0-59.

Example

DIMENSION ISB(2)
CALL GBYTES(NPB,ISB,3,6,9,2)

NPB  \begin{array}{c}
0770007700000000000000 \\
\end{array}

ISB(1),ISB(2) \begin{array}{c}
00000000000000000077000000000077 \\
\end{array}

Skip 3 bits as specified in the IBIT-offset, take a 6-bit byte (the two 7's), skip 9 bits (the next three 0's) as specified in NSKIP, and do this twice (ITER=2).
ENTRY POINT SBYTES

Purpose
To reverse the process that occurred in GBYTES.

Method
SBYTES uses the same arguments as GBYTES.

Example

```
DIMENSION ISB(2)
CALL SBYTES(NPC,ISB,45,6,3,2)
```

```
  59 0 59
ISB(1),ISB(2) 00000000000000000000000000000000770777077
  0
NPC 000000000000000000000000077077077
      59 0

45 bits (IBIT-offset) are skipped in NPC. Then the rightmost 6 bits (NBITS) in ISB(1) are packed into NPC, 3 bits are skipped in NPC (NSKIP), and the process is repeated (ITER=2) from ISB(2).
```
**INPUTB**

**Entry Points**

INPUTB, OUTPTB

**Error Messages**

- ATTEMPT TO WRITE 2 FILE MARKS, LOGICAL UNIT NO.
- ATTEMPT TO WRITE TAPE WITH NO RING REQUESTED, LOGICAL UNIT NO.
- ATTEMPT TO WRITE WITH ZERO WORD COUNT, LOGICAL UNIT NO.
- CHECK SUM ERROR, LOGICAL UNIT NO.
- DISK/DRUM ERROR, LOGICAL UNIT NO.
- DISK/DRUM SFF IS ILLEGAL, LOGICAL UNIT NO.
- DRUM/DISK ILLEGAL OP REQUESTED, LOGICAL UNIT NO.
- DRUM/DISK ILLEGAL OP SEQUENCE, LOGICAL UNIT NO.
- END OF FILE READ ON LOGICAL UNIT NO.
- END OF TAPE ENCOUNTERED ON LOGICAL UNIT NO.
- LOGICAL UNIT NOT IN SYSIOU TABLE =
- NEGATIVE RECORD LENGTH, LOGICAL UNIT NO.
- NOT ENOUGH SPACE LEFT TO COMPLETE OP, LOGICAL UNIT NO.
- PARITY ERROR ON SKIP BAD SPOT OPERATION ON LOGICAL UNIT NO.
- RECORD LENGTH EXCEEDED, LOGICAL UNIT NO.
- R. L. CONTINUATION LIST ERROR, LOGICAL UNIT NO.
- TAPE ERROR, LOGICAL UNIT NO.
- UNIT CONTROL TABLE OVERFLOW, LOGICAL UNIT NO.
- UNRECOVERABLE TAPE READ ERROR, LOGICAL UNIT NO.
- UNRECOVERABLE TAPE WRITE ERROR, LOGICAL UNIT NO.
- WRITE, WEF OR SFF PRECEDE READ OP, LOGICAL UNIT NO.

**Storage**

2418 locations plus buffer size.

**ENTRY POINT INPUTB**

**Purpose**

To transfer information in binary mode (odd parity) into storage from logical unit i.

**Fortran Reference**

READ TAPE 7,A,B,C (assume A is dimensioned for 100)
ENTRY POINT INPUTB

(Continued)

### Ascent Calling Sequence

<table>
<thead>
<tr>
<th>SB2</th>
<th>SB1</th>
<th>RJ</th>
<th>SB2</th>
<th>SB1</th>
<th>RJ</th>
<th>SB2</th>
<th>SB1</th>
<th>RJ</th>
<th>SB2</th>
<th>SB1</th>
<th>RJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>INPUTB</td>
<td>144B</td>
<td>A</td>
<td>INPUTB</td>
<td>B0</td>
<td>B</td>
<td>INPUTB</td>
<td>B0</td>
<td>C</td>
<td>INPUTB</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>INPUTB</td>
<td>-1</td>
<td>-1</td>
<td>INPUTB</td>
<td>-1</td>
<td>-1</td>
<td>INPUTB</td>
<td>-1</td>
<td>-1</td>
<td>INPUTB</td>
</tr>
</tbody>
</table>

The unit number is put in B2, B1 is set to zero to signal the first pass into the routine, and a RJ to INPUTB is taken. A separate RJ to INPUTB is taken for each variable on the list; B1 is set to the address of the variable and B2 is set to the dimension if the variable is an array, or B2 = 0 if the variable is single-valued. When the last parameter of the list is reached, B1 is set to -1, which signals that this RJ to INPUTB is the last pass into the subroutine.
ENTRY POINT OUTPTB

Purpose
To transfer information in binary mode (odd parity) from storage locations specified in the list to a specified logical unit.

Fortran Reference
WRITE TAPE 9, (Z(I), I=I1,I2)

Ascent Calling Sequence

| SB2  | 9    | Tape unit |
| SB1  | 0    | Bl=0 signals first entry |
| RJ   | OUTPTB |
| SA1  | I1   | Initialize I |
| RX7  | X1   |
| SA7  | I    |
| LOC SA2 | I | Loop on I for each element of Z |
| SB1  | X2   |
| SB1  | Bl+Z-1 |
| SB2  | 0    |
| RJ   | OUTPTB |
| SA3  | I    |
| SX0  | 1    |
| SA4  | I2   |
| IX6  | X3+X0 |
| SA6  | A3   |
| IX5  | X4-X6 |
| PL   | X5,LOC |
| SB1  | -1   | Last entry to OUTPTB |
| RJ   | OUTPTB |

In this case the indexing is explicit and a separate call to OUTPTB is generated for each loop element over I. This I/O statement call is slower than the implicit call in the example under INPUTB above where only one call is generated for the array A dimensioned for 100.
INPUTC

Purpose To process the input list and the FORMAT statement associated with the READ statement for that list.

Fortran Reference READ(n,FMT)LIST or READ INPUT TAPE n,Format,LIST

Entry Point INPUTC (uses KRAKER as an external)

Ascent Calling Sequence For READ (9,100)A,B:

```
SB3    Format address 100
SB2    9
SB1    B0
RJ     INPUTC
SB2    B0
SB1    A
RJ     INPUTC
SB2    B0
SB1    B
RJ     INPUTC
SB1    777776B
RJ     INPUTC
```

B3 contains the reference to the format statement, B2 the tape number. B1 is zero and flags the first entry into the routine, and a RJ to INPUTC is taken. The list is then processed, the address of the variable is loaded into B1 (B2 is the dimension of the variable and equals zero if it is not dimensioned), and a RJ is taken. A separate RJ is generated for each item on the list. After the last item in the list is processed B1 is set to -1 to indicate the last pass into the subroutine.

An implicit loop for input, such as READ(5,100)A, where A is dimensioned for N, is faster than READ(5,100)(A(I),I=I,N). One call is needed to INPUTC for A in the first case; in the second, N calls to INPUTC are generated by the compiler, one for each A(I).
3.47
Utility Library Subroutines

Error Messages

- DATA OVERFLOW-EXPONENT OUT OF RANGE
- EXCEEDED RECORD SIZE
- EXPONENT TOO LARGE ON DATA INPUT
- HOLLERITH FORMAT WITH LIST
- ILLEGAL DATA ENCOUNTERED
- ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
- INTEGER TOO LARGE ON DATA INPUT
- MULTIPLE DECIMAL POINTS IN DATA
- PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
- TAPE ERROR DURING READ OPERATION
- TAPE UNIT NUMBER NOT IN 1-6761B RANGE GIVEN UNIT NUMBER =
- ZERO FIELD WIDTH IN FORMAT STATEMENT

Storage

- 3468 locations

Method

INPUTC is a central processor Fortran data input routine which is entered each time a READ n,L, READ INPUT TAPE, or READ(n,FMT)L statement is executed.

The data passes from the external device, through an I/O buffer, and into central memory, where it is converted into display code, one record at a time. Each record is assembled in central memory according to the format specification and transmitted to the location specified by the compiler for the list.
Utility Library Subroutines

INPUTS

Purpose
To process the DECODE list and the FORMAT statement associated with the DECODE statement.

Fortran Reference
DECODE(c,n,V),L

Entry Point
INPUTS (uses subroutine KRAKER as an external)

Ascent Calling Sequence
For DECODE (10,103,IB)A:

| SB4 | 10 | Number of characters |
| SB3 | B0 + Format address(n) | Flag first entry |
| SB1 | B0 | |
| RJ | INPUTS |
| SB1 | IB |
| SB2 | B0 |
| RJ | INPUTS |
| SB2 | B0 |
| SB1 | A |
| RJ | INPUTS |
| SB1 | -1 | Flag last entry |
| RJ | INPUTS |

Error Messages
DATA OVERFLOW—EXponent OUT OF RANGE
EXCEEDED RECORD SIZE
EXponent TOO LARGE ON DATA INPUT
HOLLERITH FORMAT WITH LIST
ILLEGAL DATA ENCOUNTERED
ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
INTEGER TOO LARGE ON DATA INPUT
MULTIPLE DECIMAL POINTS IN DATA
PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
TAPE ERROR DURING READ OPERATION
TAPE UNIT NUMBER NOT IN 1-6761B RANGE GIVEN UNIT NUMBER =
ZERO FIELD WIDTH IN FORMAT STATEMENT
Storage

271₈ for INPUTS + 574₈ for KRAKER

Method

INPUTS is a central processor routine which is entered each time a DECODE statement is executed. The data, which are in display code, are assembled in central memory according to the FORMAT specification, one record at a time. The data are then transmitted to the location specified by the compiler for the list.
IOPROC

Purpose
To perform the functions of buffer statements plus certain tape I/O functions which the buffer statements cannot do.

Entry Points
BSTAPE, RDTAPE, WRTAPE, IOWAIT

Error Messages
ATTEMPT TO USE AN ILLEGAL MODE NUMBER
ATTEMPT TO USE AN UNASSIGNED UNIT NUMBER
ATTEMPT TO USE ILLEGAL UNIT NUMBER

Storage
3478 locations

Note: Tape densities are specified on *ASSIGN cards.

ENTRY POINT BSTAPE

Purpose
To backspace one physical record.

ENTRY POINT RDTAPE

Purpose
To read a record.

Limitation
● When reading an EOF with RDTAPE, the first word of the buffer read in for the EOF will contain a BCD 17.
CALL RDTAPE(NUNIT,MODE,NTYPE,NADDR,NWDCNT)

**NUNIT**  A logical tape unit

**MODE**

- 0 = Even parity (BCD mode), no character conversion
- 1 = Odd parity (binary mode), no character conversion
- 2 = Even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record)

**NTYPE**

- 0 = Used for all tape writes and to read tapes created on the NCAR system. The NWDS from IOWAIT ignores any partial words. (1 provides the same option.)
- 2 = Used to read records from another computer. The NWDS from CALL IOWATT includes a count for any partial word at the end of the record.

**NADDR**  Address of the first word in the record

**NWDCNT**  Number of words to be read or written from the record, or maximum possible record size

---

Note: Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error. Thus a type 4 is the same as type 0 but with no reread on parity errors.

---

Character conversion applies only to tapes. The drum never does character conversion on a read or write.
ENTRY POINT WRTAPE

Purpose
To write a record.

Fortran Call
CALL WRTAPE (same arguments as RDTAPE)

ENTRY POINT IOWAIT

Purpose
To wait for the completion of a read or write.

Fortran Calling Sequence
CALL IOWAIT(NUNIT,NSTATE,NWDS)

NUNIT A logical tape unit
NSTATE Status:
  0 = good read or write
  1 = EOF
  2 = parity error on read or unable to write
  3 = end of tape
NWDS Number of 6600/7600 60-bit words read or written

It is possible to call IOWAIT and get a good return before a unit has been used.

Ascent Calling Sequence
RJ IOWAIT
EQ **4
CON NUNIT
CON NSTATE
CON NWDS

Limitations
- After a read or a write, IOWAIT must be called before reading or writing again from the same unit.
- In a call to RDTAPE or WRTAPE, the array size must be equal to or larger than the NWDCNT specified.
JOBID

Purpose
To return to the user job identification data in display code.

Fortran Call
CALL JOBID(IWHERE)

Entry Point
JOBID

Ascent Calling Sequence
RJ JOBID
JP *+2
CON IWHERE

Normal Return
IWHERE is a four-word array in which the following job identification is returned:

| IWHERE(1) | Sequence number |
| IWHERE(2) | Name            |
| IWHERE(3) | Scientist number|
| IWHERE(4) | Project number  |

All items are left-justified with blank fill.

Error Message
None

Storage
108 locations

Method
The central monitor reads a table from the user's buffer area containing this information.
### KODER

**Purpose**
To scan the format and convert the data to the format specified for output. KODER is called by OUTPTC/OUTPTS.

**Fortran**
No access

**Ascent**
No access

**Error Message**
Refer to error messages for OUTPTC and OUTPTS.

**Storage**
10158 locations
KRAKER

Purpose       To scan the format and convert the data to the format specified for input. KRAKER is called by INPUTC/INPUTS.

Fortran       No access

Ascent        No access

Error Messages Refer to error messages for INPUTC and INPUTS.

Storage       575, locations
**MACHINE**

**Purpose**
To identify whether the user's job has been run on the Control Data 6600 computer or on the 7600 computer.

**Fortran Function**
\[ N = \text{MACHINE}(l) \]

**Entry Point**
MACHINE

**Ascent Calling Sequence**
RJ MACHINE

**Normal Return**
X1 contains the code to identify the 6600 or the 7600 as follows:

- 6600: \( X1 = 0 \) (00B)
- 7600: \( X1 = 11 \) (13B)

**Error Message**
None

**Storage**
3 locations

**Method**
Bits 52-55 in location 1 are non-zero on the 7600. Bits 52-55 are zero on the 6600. Location 1 is read and X1 is set accordingly. Note that the argument is a dummy.
NEWLAB

Purpose
To change the label of a previously assigned tape.

Fortran Call
CALL NEWLAB(UNIT,LABEL)

UNIT An unsigned integer designating the logical tape having a label changed.

LABEL An integer type variable containing a maximum of five characters left-justified in DPC.

Entry Point
NEWLAB

Ascent Calling Sequence
RJ NEWLAB
JP *+3
CON UNIT
CON LABEL

UNIT CON Assigned unit number
LABEL DPC *ABCD...* left-justified

Normal Return
The tape label is changed. No other action is performed.

Error Message
None

Storage
108 locations

Method
A new label is provided. Note that after NEWLAB is called, the programmer must unload the unit number of the previous tape mount and subsequently refer to that unit again following the UNLOAD call. The reference to the unit after the UNLOAD will display on the scope the request to the operator to mount the new tape.
OUTPTC

Purpose
To process the output list and associated FORMAT statement.

Fortran Reference
PRINT, PUNCH, WRITE OUTPUT TAPE 6, WRITE(6,100)

Entry Point
OUTPTC (uses KODER as an external)

Ascent Calling Sequence
For WRITE (6,100)A,B:

| SB3 | Format address |
| SB2 | 6776B          |
| SB1 | B0             |
| RJ  | OUTPTC         |
| SB2 | B0             |
| SB1 | A              |
| RJ  | OUTPTC         |
| SB2 | B0             |
| SB1 | B              |
| RJ  | OUTPTC         |
| SB1 | -1             |
| RJ  | OUTPTC         |

B3 contains the format address and B2 the tape number.
B1 = 0 flags the first entry into the routine, and a RJ to OUTPTC is generated. For each variable in the list, a separate RJ is generated where B2 = the dimension of the variable and B1 the address. B2 = 0 if there are no dimensions. The final pass to OUTPTC is flagged by setting B1 = -1.

Note: Only one call to OUTPTC is generated if a dimensioned variable is output implicitly, as it would be, for example, if only the base address of the array in B1 and the dimension of A in B2 were specified. However, if an explicit loop is programmed in Fortran, as for example,

WRITE (6,100)(A(I),I=1,N)

a RJ to OUTPTC is generated for each A(I). Thus N return jumps to OUTPTC would be taken in a Fortran-compiled loop.
OUTPTC
(Continued)

Ascent Calling Sequence
(continued)

For PUNCH 102, \((A(I), I=M, MM, 10)\):

\[
\begin{align*}
SB2 & \quad 6775B \\
SB3 & \quad \text{Format address} \\
SB1 & \quad B0 \\
RJ & \quad \text{OUTPTC} \\
SA4 & \quad M \\
BX7 & \quad X4 \\
SA7 & \quad I \\
LOC & \quad \text{SA1} \\
SB1 & \quad X1 \\
SB1 & \quad B1+A-1 \\
SB2 & \quad 0 \\
RJ & \quad \text{OUTPTC} \\
SA5 & \quad I \\
BX0 & \quad 10 \\
SA2 & \quad MM \\
IX6 & \quad X5+X0 \\
SA6 & \quad A5 \\
IX3 & \quad X2-X6 \\
PL & \quad X3, LOC \\
SB1 & \quad -1 \\
RJ & \quad \text{OUTPTC}
\end{align*}
\]

Note: The punch unit is 6775B and, since the list indexing is explicit, a RJ to OUTPTC is done for each value of I.

Error Messages

- FIELD WIDTH LESS THAN DECIMAL WIDTH
- HOLLERITH FORMAT WITH LIST
- ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
- OUTPUT RECORD LENGTH EXCEEDED
- PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
- SINGLE ARRAY, D FORMAT
- TAPE ERROR ON OUTPUT
- UNIT NO. NOT PUNCH UNIT, OR IN 1-6761B RANGE GIVEN UNIT NUMBER = ZERO FIELD WIDTH IN FORMAT STATEMENT

Storage

- 3778 for OUTPTC + 10158 for KODER
Method

OUTPTC is a central processor Fortran data output routine which is entered each time a PRINT, PUNCH, or WRITE OUTPUT TAPE statement is executed. The data in the list are converted from their internal to external representations. The display code records of information are assembled, one at a time, in central memory according to the FORMAT specification. As each record is assembled, it is passed to a central I/O buffer where it is transmitted to an external device by other system routines.
OUTPTS

Purpose
To process the ENCODE list and associated FORMAT statement.

Fortran Reference
ENCODE(c,n,V)L

Entry Point
OUTPTS (uses KODER as an external)

Ascent Calling Sequence
SB4 B0+c
SB3 B0+n Format address
SB1 B0
RJ OUTPTS
SB1 V
SB2 B0
RJ OUTPTS
SB1 B0+L
SB2 B0
RJ OUTPTS
SB1 777776B
RJ OUTPTS

Error Messages
FIELD WIDTH LESS THAN DECIMAL WIDTH
HOLLERITH FORMAT WITH LIST
ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
OUTPUT RECORD LENGTH EXCEEDED
PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
SINGLE ARRAY, D FORMAT
UNIT NO. NOT PUNCH UNIT, OR IN 1-6761B RANGE GIVEN UNIT NUMBER =
TAPE ERROR ON OUTPUT
ZERO FIELD WIDTH IN FORMAT STATEMENT

Storage
2728 for OUTPTS + 10228 for KODER

Method
OUTPTS is a central processor routine which is entered each
time an ENCODE statement is executed. The data in the list
are converted from machine language representation to display
code, according to the format specification. One display
code record of information is assembled in central memory
and transmitted to a location specified by the ENCODE
statement.
OVERLAY

Purpose
To provide a technique for bringing routines into the central memory from the peripheral drum, so that several routines occupy the same storage locations at different times.

Description
OVERLAY is used when the total storage requirements for instructions exceed the available main storage. The program is divided into separate parts which may be called and executed as needed providing the user with an almost unlimited program size. The parts are one control program, overlays of the control program, and segments of the overlays.

The overlay subroutine can be used to run as one job the series of jobs with program A generating output to be used as input for program B. The case may be that A and B are too large to fit in central memory together, or A must be completely processed before the data is ready for B.

A control program is constructed which moves programs A and B, as overlays, in a sequential manner to operate on a data set. This procedure accomplishes with one deck what formerly took several decks and a great deal of time. Other potential uses for this procedure include input options for different integration schemes to be called as overlays, or for different output programs to be called as overlays.

Note: OVERLAY does not provide the user with unlimited data storage capabilities.

Rules
- Only the control program, one overlay program, and one segment program may occupy central memory at a given time.

- A list may be transferred to an overlay or segment program. The preferred means of variable transfer is through blank common.
Control Program

A control program is a sequence of instructions which prescribes the steps to be taken in moving a set of overlay programs into core where they will perform their individual operations on a data set. A control program may or may not be considered a main program since a given set of main programs can be used as overlays and moved by a control program.

- The control program may contain subprograms.
- The control program always remains in central memory during execution.
- The control program may only call overlays (not segments).
- The control program is identified by an OVERLAY(0,0) card.
- Subroutines which are not from the system but are loaded with the control program are identified with an OVERLAY(0,0) card.
- All subroutines loaded with the control program may be called from the control program, overlay program, segment program, or other subroutine.

Overlay Program

An overlay program is a sequence of instructions which performs an operation on data as does a subroutine. An overlay program is moved into core only on command of the control program.

- An overlay program is always a subprogram to the control program.
Overlay programs are loaded from the drum when called by the control program.

Overlay programs may call only their associated segment programs.

An overlay program may not call another overlay program.

An overlay program is identified by an OVERLAY(IO,0) card.

An overlay program is called into the central processor with a CALL OVERLAY(IO,0,LIST) type of call statement.

An overlay program may have subroutines associated with it only. These subroutines can only be called by this overlay program and its associated segments.

The subroutines associated only with a given overlay program are identified with an OVERLAY(IO,0,NAME) card. NAME is the overlay entry point name. These subroutines are called in the ordinary way in the overlay.

An overlay program may call any subroutine associated with the control program.

Labeled common is common only to its overlay and segments.

An overlay is brought into the computer in its original form each time it is called. Values are not maintained in storage locations or in labeled common from one overlay call to another, since the overlay is never written out in its current form.
A *segment program* is a sequence of instructions which performs an operation on data as does a subroutine. The segment program is associated with only one overlay program and can only be called into core by this overlay program.

- A segment program is always a subprogram to an overlay program.

- A segment program cannot be used as a subprogram to the control program.

- Segment programs are loaded from the drum when called by their associated overlay program.

- Segment programs may not call the control program, any overlay program, or any other segment program.

- A segment program is identified by an OVERLAY(IO,IS,NAME) card.

- A segment program is called into the central processor with a CALL OVERLAY(IO,IS,LIST) call statement.

- A segment program may have subroutines associated with it only. These subroutines can only be called by this segment program.

- The subroutines associated only with a given segment are identified with an OVERLAY(IO,IS,NAME). NAME is the controlling subroutine in a segment. These subroutines are called in the ordinary way in the segment.

- A segment program may call any subroutine associated with its overlay program or the control program.
The overlay identification card is used to tell the loader what kind of program is currently being processed and with which program it is associated. This card is placed immediately after the PROGRAM NAME or SUBROUTINE NAME card. It has the following format:

OVERLAY(IO,IS,NAME)

The program identification card for an overlay program, where

IO = Overlay number
IS = 0 = Segment number for control program
NAME = Entry point name of the controlling subroutine in the overlay. The entry point name is the NAME on the SUBROUTINE NAME card associated with the given overlay or segment program.

This card is used with an overlay or a segment program and is used when:

a. A binary deck is to be punched and subsequently loaded with nonbinary decks.

b. The overlay or segment program contains dimensioned variables not in blank or labeled common.
Overlay Call Statements

Overlay call statements are Fortran call statements for an overlay or segment program which have one of the following formats:

1. CALL OVERLAY(IO,IS) where
   - IO = overlay number
   - IS = segment number when applicable

2. CALL OVERLAY(IO,IS,LIST) where
   - IO = overlay number
   - IS = segment number when applicable
   - LIST = parameter LIST(I,A) that appeared on the SUBROUTINE NAME(I,A) card

Error Messages

OVERLAY LIST DOES NOT CONTAIN OVLY/SEG
SYSTEM OR HARDWARE FAILURE DURING READ ATTEMPT ON OVLY/SEG

Storage

2058 locations

Central Memory Allocation

Central memory allocation is as follows:

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>Central monitor</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>Overlay control table</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>44</td>
<td>Blank common</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>LOC1</td>
<td>Control program and dimension</td>
</tr>
<tr>
<td></td>
<td>+ major subroutine</td>
</tr>
<tr>
<td></td>
<td>+ system subroutines (e.g., SIN, COS, PLOT, etc.)</td>
</tr>
<tr>
<td>LOC2</td>
<td>Labeled common</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>LOC3</td>
<td>Overlay origin</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>LOC4</td>
<td>Segment origin</td>
</tr>
</tbody>
</table>

End Allocation

LOC1 through LOC4 are arbitrary core locations depending on the relative size of the areas.
If different kinds of programs are loaded together with subroutine OVERLAY, the following rules apply.

- **Fortran**
  
  Fortran decks immediately after the control program will be loaded with the central program unless overlay cards signal otherwise. Decks may be placed out of order but loaded with any overlay if the proper overlay card is present.

- **Ascent**
  
  An Ascent program may be used as an overlay or a segment. An overlay card is placed immediately after the Ascent card and has the following format:

  Col. 11 Col. 20
  OVERLAY IO, IS, NAME

  where IO, IS, and NAME are defined above.

- **Cosy**
  
  The Cosy deck must have been punched from a Fortran or Ascent deck containing the proper overlay card.

- **Binary**
  
  The binary deck must have been punched from a Fortran or Ascent deck containing the proper overlay card.
If a binary deck is being loaded with non-binary decks, the following rules apply.

- If the binary deck is the control program or is used by the control program, an OVERLAY(0,0) card must have been used when the deck was punched.

- If the binary deck is an overlay or a segment, then an OVERLAY(IO,IS,NAME) card must have been used when the deck was punched.

- If the binary deck is a subroutine to be loaded with an overlay or a segment, then an OVERLAY(IO,IS,NAME) card must have been used when the deck was punched. NAME is the name of the controlling subroutine in the overlay with which this particular binary subroutine is associated.
Computer Sample 3 illustrates the use of OVERLAY. An explanation of the loader map follows the sample.

```
CARD NUMBER    PROGRAM LOCATION
000000  *FORTRAN
000001  PROGRAM CONTROL
000002  OVERLAY (0,0)
000003  COMMON A(10),B(10),C(10),D(10)
000004  DIMENSION E(10)
000005  XI=0.0
000006  XA=1.0
000007  DO 50 I=1,10
000008  SUM=0.0
000009  LOAD A ARRAY - INPUT PHASE
000010  DO 10 J=1,10
000011  A(J)=XI
000012  XI=XI+XA
000013  CALL 1ST OVERLAY - CALCULATION PHASE
000014  CALL OVERLAY (1,0,XI,XA)
000015  FORM NEW D ARRAY - DATA SUMMARY PHASE
000016  CALL OVERLAY (2,0)
000017  FORM FINAL ANSWER FOR ITH SET
000018  DO 20 J=1,10
000019  SUM= A(J)+B(J)+C(J)+D(J)
000020  E(I)=SUM
000021  20 CONTINUE
000022  XI=0.0
000023  XA=XH+1
000024  CALL OUTPUT PROGRAM
000025  CALL OVERLAY (3,0,E)
000026  CALL COS
000027  CALL SIN
000028  CALL EXIT
000029  END PROGRAM
```

**VARIABLE ASSIGNMENTS**

- \( E \) = 000000
- \( A \) = 000000C00
- \( B \) = 000012C00
- \( C \) = 000072C00
- \( D \) = 000036C00
- \( XI \) = 000076
- \( XA \) = 000075
- \( I \) = 000074
- \( SUM \) = 000073
- \( J \) = 000072

**SUBROUTINES CALLED**

- OVERLAY
- EXIT
- COS
- SIN

**Compile Time** = 21
Utility Library Subroutines

Computer Sample 3
Program CONTROL
(continued)

CARD Approximate Number Program Location
0001 000000 SUBROUTINE OVLY10 (YI,YA)
0002 000000 OVERLAY (1,0,OVLY10) THIS IS THE FIRST OVERLAY
0003 000000 C COMMON A(I),B(I),C(I),D(I)
0004 000000 YC=YI-YA
0005 000020 CALL SUB1
0006 000002 C CALL 1ST SEGMENT TO THIS OVERLAY - FORM B
0007 000006 CALL OVERLAY (1,0,YC) CALL 2ND SEGMENT TO THIS OVERLAY - FORM C
0008 000002 C
0009 000006 SUBROUTINES CALLED
0010 000013 CALL OVERLAY (1,2,YS)
0011 000020 RETURN
0012 000020 END
0013 000020

LENGTH OF ROUTINE OVLY10 00036

VARIABLE ASSIGNMENTS
VY - 000000 YA - 000000 A - 00000000 B - 00000000 C - 00000000 D - 00000000
VG - 000027 YS - 000026

SUBROUTINES CALLED
OVERLAY OVLY10

COMPILE TIME= 7

CARD Approximate Number Program Location
0001 000000 SUBROUTINE SUB1
0002 000000 OVERLAY (1,0,OVLY10) NOTE THE OVERLAY IDENTIFIER IS OVLY10
0003 000000 C
0004 000000 A=Y
0005 000004 RETURN
0006 000004 END

LENGTH OF ROUTINE SUB1 00013

VARIABLE ASSIGNMENTS
A - 000010 V - 000007

COMPILE TIME= 2
3.71
Utility Library Subroutines

CARD NUMBER
0001 000000
0002 000000
0003 000000
0004 000000
0005 000000
0006 000000
0007 000000
0008 000000
0009 000000
0010 000000

PROGRAM LOCATION
SUBROUTINE SEG11 (Z)
OVERLAY (1,1,SEG11)
THIS IS THE 1ST SEGMENT OF THE 1ST OVERLAY
COMMON A(10),B(10),C(10),D(10)
COMPUTE AN ARRAY B
DO 10 I=1,10
B(I) = SINF (A(I)-(.5Z))
10 CONTINUE
RETURN
END

LENGTH OF ROUTINE SEG11 000027
VARIABLE ASSIGNMENTS
Z - 000000 A - 000000 B - 000000 C - 000000 D - 000000 I - 000000

SUBROUTINES CALLED
SINF

COMPILE TIME= 6

CARD NUMBER
0001 000000
0002 000000
0003 000000
0004 000000
0005 000000
0006 000000
0007 000000
0008 000000
0009 000000
0010 000000

PROGRAM LOCATION
SUBROUTINE SEG12 (S)
OVERLAY (1,2,SEG12)
THIS IS THE 2ND SEGMENT OF THE 1ST OVERLAY
COMMON A(10),B(10),C(10),D(10)
COMPUTE AN ARRAY C
DO 20 I=1,10
C(I) = COSF (A(I)-(.5*S))
20 CONTINUE
RETURN
END

LENGTH OF ROUTINE SEG12 000027
VARIABLE ASSIGNMENTS
S - 000000 A - 000000 B - 000000 C - 000000 D - 000000 I - 000000

SUBROUTINES CALLED
COSF

COMPILE TIME= 6

CARD NUMBER
0001 000000
0002 000000
0003 000000
0004 000000
0005 000000
0006 000000
0007 000000
0008 000000
0009 000000
0010 000000

PROGRAM LOCATION
SUBROUTINE OVLY20
OVERLAY (2,0,OVLY20)
THIS IS THE SECOND OVERLAY
COMMON A(10),B(10),C(10),D(10)
COMPUTE AN ARRAY D
DO 10 I=1,10
D(I) = 2.0*(A(I)-B(I)+C(I))
10 CONTINUE
RETURN
END

LENGTH OF ROUTINE OVLY20 000023
VARIABLE ASSIGNMENTS
A - 000000 B - 000000 C - 000000 D - 000000 I - 000000

COMPILE TIME= 7
3.72 Utility Library Subroutines

Computer Sample 3
Program CONTROL
(continued)

CARD APPROXIMATE NUMBER PROGRAM LOCATION
0001 000000 SUBROUTINE OUTPUT (E)
0002 000000 OVERLAY (J, I, OUTPUT)
0003 000000 C
0004 000000 DIMENSION E(10) THIS IS THE THIRD OVERLAY
0005 000000 WRITE (6,10) (E(I), I=1,10)
0006 000014 10 FORMAT (1X,IX,10(E10.3))
0007 000014 RETURN
0008 000014 END

LENGTH OF ROUTINE OUTPUT 000032

VARIABLE ASSIGNMENTS
E - 000000 I - 000022

SUBROUTINES CALLED
OUTPTC Q8QRSD

COMPILE TIME= 5

PROGRAM SPACE IS 002235
CONTROL PROGRAM LENGTH 00531

ENTRY POINT LOCATION ROUTINE ORIGIN
Q8QRSD 003220 000220
EXIT 003232 000232
END 003232 000232
STOP 003232 000232
SIN 003237 000237
COS 003237 000237
OVERLAY 00324 000324

COMMON BLOCKS LOCATION
000044

OVERLAY 000001 000000 LOCATION 00531 LENGTH 00102

ENTRY POINT LOCATION ROUTINE ORIGIN
OVERLAY 000001 000000 SUBI 000067 000067
Q8QRSD 000062 000062

SEGMENT 000001 000001 LOCATION 00633 LENGTH 00027
ENTRY POINT LOCATION ROUTINE ORIGIN
SEG11 000633 000633

SEGMENT 000001 000002 LOCATION 00633 LENGTH 00027
ENTRY POINT LOCATION ROUTINE ORIGIN
SEG12 000633 000633

OVERLAY 000002 000000 LOCATION 00531 LENGTH 00023

ENTRY POINT LOCATION ROUTINE ORIGIN
OVERLAY 000003 000000 SUBI 000067 000067
Q8QRSD 000062 000062

OVERLAY 000003 000000 LOCATION 00531 LENGTH 001504
ENTRY POINT LOCATION ROUTINE ORIGIN
OVERLAY 000003 000000 SUBI 000067 000067
Q8QRSD 000062 000062

THE TOTAL PROGRAM AND BUFFER SPACE REQUIRED IS 002255
Utility Library Subroutines

<table>
<thead>
<tr>
<th>Time in Milliseconds</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
<th>131922</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Time</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>PPU Time</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>Pages Printed</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>Total Resources Used</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

Total CPU Time in Milliseconds: 126
PPU Time in Milliseconds: 3967
Pages Printed: 10
Total Resources Used: .61
The loader map on p. 3.72 follows the usual format up to the point at which overlay is loaded.

a. OVERLAY 000001 000000

This represents the overlay card which is placed before the SUBROUTINE OVLY10 card, and is read OVERLAY(1,0).

b. LOCATION 000531 LENGTH 00102

This gives the location at which OVERLAY(1,0) will be loaded when it is called by the control program. It will occupy the next 1028 cells.

c. ENTRY POINT LOCATION ROUTING ORIGIN
   OVLY10 000531 000531
   SUB1 000567 000567
   Q8QRSD 000602 000602

The first entry point is the name of the subroutine which was labeled as OVERLAY(1,0). The location of this entry point follows. The last number is the origin of the routine in which this entry point is located.

d. SEGMENT 000001 000001

This represents the overlay card placed before the SUBROUTINE SEG11 card and read OVERLAY(1,1).
e. LOCATION 000616 LENGTH 000027

This gives the location at which this segment will be loaded when it is called by its associated OVERLAY. It will occupy the next 27 bytes after the overlay is loaded.

f. ENTRY POINT LOCATION ROUTINE ORIGIN
   SD311 000616 000616

This entry point is the name of the subroutine which was labeled OVERLAY(1,1). The rest of the description is the same as for item c above.
PAUSE

Purpose To stop program execution and place the program on the drum so that the operator can perform tasks such as setting sense switches. Program execution is then resumed.

Fortran Reference PAUSE or PAUSEn

Entry Point PAUSE

Ascent Calling Sequence

| N | DPC | *11111* |
| N | SAL | PAUSE |

In XL the left-most 30 bits (30-59) contain the PAUSE identifier in DPC. A RJ to PAUSE is then taken.

Storage 7 locations

Method PAUSE stops program execution with the words PAUSEn HOLD TIL SIGNAL appearing on the console after the program's control point number under the heading STATUS. If n = 0, only PAUSE is printed. Instructions for the operator are written on the job card in the space provided.

Example of Console Printout

```
CP SEQ. PROG PTTG TTG SSW STATUS
... 03 3246 ADAMS 002M 12S PAUSE 5 HOLD TIL SIGNAL
```

To restart execution, the operator types the request GO after typing the control point number, in this case 03, at the console.

Example

```
03 GO
```
**PDUMP**

**Purpose**
To write an octal dump of the entire program or parts of the program, depending on the arguments specified.

**Entry Point**
PDUMP

**Error Message**
ARGUMENT EXCEEDS FIELD LENGTH

**Storage**
340₈ locations

**Method**
This is a central processor dump-and-proceed written in Ascent. The peripheral dump is not called.

**Registers Saved**
The A registers are never saved. All other registers are saved and restored.

**Variable Arguments**
PDUMP uses a variable argument list. A choice of one of the four options depends on whether there are no, one, two, or three arguments. The test for the argument count is made using the RJ address.

**No Arguments**

```plaintext
CALL PDUMP
```

Ascent calling sequence:

```
RJ PDUMP
```

PDUMP will dump from zero to the field length (specified in FL on the panel).
Utility Library Subroutines

**PDUMP**

(Continued)

*One Argument*

CALL PDUMP (500B)

Ascent calling sequence:

```
RJ       PDUMP
EQ       *+2
CON      NUM     (where NUM contains 500B)
```

The starting address of the dump is the single argument, 500B. A dump will be taken from this argument to the field length.

*Two Arguments*

CALL PDUMP (500B, 1000B)

Ascent calling sequence:

```
RJ       PDUMP
EQ       *+3
CON      NUM1    (contains 500B)
CON      NUM2    (contains 1000B)
```

The starting address of the dump is in NUM1 which will dump to the address in NUM2. If the first argument = 0, the dump starts at zero; if the second argument = 0, the dump terminates at the field length.

**Example 1**

```
CALL PDUMP (0,1000B)
```

This dumps from 0 to 1000B.

**Example 2**

```
CALL PDUMP (1500B, 0)
```

This dumps from 1500B to field length.
Three Arguments

CALL PDUMP (A(1), B(30), 1)

Ascent calling sequence:

<table>
<thead>
<tr>
<th>RJ</th>
<th>PDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>+4</td>
</tr>
<tr>
<td>CON</td>
<td>A</td>
</tr>
<tr>
<td>CON</td>
<td>B</td>
</tr>
<tr>
<td>CON</td>
<td>N</td>
</tr>
</tbody>
</table>

(wh... where N = 1)

The flag (N=1) signals that three arguments are present and that the first two arguments contain the actual dump address (i.e., not a reference to the dump address). Thus the dump in this case proceeds from A(1) to B(30). This format will also dump a whole routine or certain routines if desired. For example, a CALL PDUMP(EXIT, PDUMP, 1) would start at the beginning of EXIT and stop at the beginning of PDUMP. In order to use this option, three Fortran arguments must be present in the call. EXIT and PDUMP must be declared external with the Fortran statement EXTERNAL EXIT, PDUMP.
PLIBRD

Description
PLIB is a permanent storage library located on the disk. The user can create and use files that are labeled with his project number (as specified on his *JOB card) and labeled with a name for the file specified by the user. Program decks can be stored in this library by using the options available on the *FORTRAN, *ASCENT, and *MOD control cards.

This program provides the user with a more general capability. The Fortran user may call this subroutine from his program to read or write a series of records to a file. The records may contain any kind of information, and the record size may be of any length. If the file does not already exist, it will be set up when the first rewind is executed. A typical use of a file is to store binary information frequently used by one or more programs. The file replaces a tape as the source of this data, and thus eliminates mounting that tape for every run.

Method
To reset a file call PLIBREW. To read a record call PLIBRD. To write a record call PLIBWT. The read and write are buffered routines, i.e., the operation starts and control returns to the caller. To check that the operation is complete, call PLIBCK.

File Identification
All files are associated with the project number on the *JOB card.

Entry Points
PLIBREW, PLIBRD, PLIBWT, PLIBCK

Storage
Length = 1138 locations
ENTRY POINT PLIBREW

CALL PLIBREW(NAME,NERR)

Fortran Calling Sequence

This call initializes a file or repositions the file at the first record.

NAME
Name of the file, which can be any 60-bit configuration. If, however, later use will be made of the editing features of the system, DPC characters left-adjusted with blank fill should be used. NAME = 4HFL23 where FL23 is the file name.

NERR
A location the subroutine will use to pass back an error code. NERR = 0 means the call was successful. NERR ≠ 0 means call was improper or the subroutine is unable to do the request. NERR error codes are:

NERR = 0 Successful operation
  = 1 EOF, or attempted to read more records than available in file
  = 3 Unsuccessful operation
  = 4 Attempted another read or write before checking previous one
ENTRY POINT PLIBRD

Fortran Calling Sequence

CALL PLIBRD(NAME, ARRAY, LENGTH, NERR)

This call will read one record from the file called NAME. Call PLIBREW before reading the first record from a file.

NAME
As in PLIBREW

ARRAY
Field of words to be filled by the read

LENGTH
Number of words to be read into array

NERR
As in PLIBREW

ENTRY POINT PLIBWT

Fortran Calling Sequence

CALL PLIBWT(NAME, ARRAY, LENGTH, NERR)

This call will write one record to a file called NAME. Call PLIBREW before writing the first record to the file.

NAME
As in PLIBREW

ARRAY
Field of words to be written to file

LENGTH
Number of words to be written

NERR
As in PLIBREW
ENTRY POINT PLIBCK

**Fortran Calling Sequence**

CALL PLIBCK(LNTH)

This call should be made after each read or write to check that the read or write sequence is complete and successful. If the sequence is not complete, PLIBCK will wait until it is. If the read or write was unsuccessful, PLIBCK will put an error code in the NERR location of the read/write call.

**LNTH**

A location in which PLIBCK will put the actual number of words read or written. This is useful when reading records shorter than expected or when a failure has occurred in a read or write.
Notes:

- No assign card is necessary for PLIB.

- A particular file on PLIB can be thought of as a tape. Before each sequence of reads or writes to a file, PLIBREW should be called to position the file at the first record. Care should be taken when writing several records on two or more files. All the records of one particular file should be read or written before the next file is reset and read or written. That is, one cannot interlace reading or writing records to several files.

- Because the system has a very useful editing feature for PLIB which is controlled by parameters on the *FORTRAN, *ASCENT, and *MOD cards, compatibility is sometimes desirable. The editor will expect the name of the file to be DPC characters left-justified in a word with blank fill; these records must be 4096 words long. The editor produces, and expects to find on the PLIB file, a Cosy-like deck. Consult a systems programmer for assistance in formatting the data. See the Fortran Reference Manual for a discussion of the editing specifications for creating a PLIB file. It is under the section on File Manipulation Modifiers in Chapter 12.
Utility Library Subroutines

RANRD

Note: For true buffering, and therefore optimum use of random files, use BRANRD, BRANWT, BRANCK.

Purpose

To provide the user with the capability to write, read, and rewrite records of differing lengths, at random, to and from the peripheral drum.

Description

There are two sets of programs. One set (RANINT, RANWT, RANRD) provides the capability for unbuffered read/writes. The other set (RANOUT, RANIN, RANCK) provides the capability for buffered read/writes. (Do not assign the drum; the system will do it for this routine.)

Entry Points

RANINT, RANWT, RANRD
RANOUT, RANIN, RANCK

Storage

3028 locations

ENTRY POINT RANINT

Purpose

To set up certain tables and flags which are subsequently used by RANWT and RANRD.

Description

The RANINT subroutine is the initializing routine and must be called first.

Fortran Calling Sequence

CALL RANINT(ID, IDLEN, NERROR)

The core space for the tables and flags must be supplied by the user and are transmitted in the calling sequence of RANINT. The core space to be supplied is defined as follows:
ENTRY POINT RANINT ID

(Continued)

An array of at least two core cells for tables. The ID array will be divided into two tables to provide information for each record, as follows.

Table 1

Name table. Each record must have a unique identifying name.

Table 2

Length table. The length of each record.

IDLEN

The length of the ID array. In order to supply enough ID TABLE area the user must reserve twice the number of uniquely named records that he plans to write.

NERROR

An error flag cell checked by the user in his program after each call to RANINT, RANWT and RANRD. If NERROR = 0, all operations of the last call were successful. If NERROR ≠ 0, an error in the last call occurred. By printing NERROR in an I2 format, the user can check the NERROR number against the error flag table, which follows:

NERROR = 1 EDF or record not in file
= 3 Unsuccessful operation
= 4 Attempted another operation before calling RANCK to check previous operation
= 5 No more room in ID array or ID array has not been specified with at least two core cells
= 6 Incorrect calling sequence--initially written record does not specify length of record or attempt to change record length

Error Messages

UNABLE TO ESTABLISH RANFILE
UNABLE TO RESET RANFILE
ENTRY POINT RANWT

Purpose
To provide two calling sequences for writing records.

Fortran Calling Sequence
CALL RANWT(NAME,ARRAY1,NERROR,LENGTH)
CALL RANWT(NAME,ARRAY1,NERROR)

Method
The first calling sequence has one extra argument, LENGTH, and is used for writing a given record for the first time. The second calling sequence is used for rewrites of previously written records. Once a given record is written with the first calling sequence, its LENGTH cannot be changed and is the length used for all subsequent reads and rewrites.

The arguments are defined as follows:

NAME
A variable name containing one 60-bit word of any configuration, other than an indefinite, which is unique for each record.

ARRAY1
The array of data to be written.

NERROR
The error flag cell.

LENGTH
The number of words of ARRAY1 which is to be written. This is also the permanent length of the record with this NAME and will be used for all subsequent operations pertaining to this record.

Error Message
RECORD LENGTH MUST BE GT 0
ENTRY POINT RANRD

Purpose
To read the data identified by NAME into ARRAY2 with the length given when the record was previously written.

Fortran Calling Sequence
CALL RANRD(NAME,ARRAY2,NERROR)

Method
The arguments are:

NAME
The record identifier as described in entry RANWT.

ARRAY2
The array into which the data is read.

NERROR
The error flag cell.

Rules
- Call RANINT first and only once.
- Call RANWT (four arguments) for new record writes.
- Check the error flag after each call.
- Special provisions must be made at RESTART time when using the SAVE routine together with the random read/write routines. All records must be rewritten using the original length after RESTART. Normal flow may then be entered.
- The maximum number of names is 200. If more names are required, use the ULIB version of RANRD, where the number of file names may be changed. See a systems programmer.
ENTRY POINT RANOUT

Purpose
To start a buffered write to the random file, and immediately return control to caller.

Fortran Calling Sequence
CALL RANOUT(NAME,ARRAY1,LENGTH,NERROR)

ARRAY1 should not be changed before RANCK is called to assure that the write is complete. The arguments are:

NAME
The record identifier, as described in entry RANWT.

ARRAY1
The array of data to be written.

LENGTH
The number of words of ARRAY1 which are to be written. Unlike RANWT, this parameter is in every call. It may vary from call to call, but the first write of a particular record must have the maximum length of that record.

NERROR
The error flag cell. This cell must be checked after each call to RANOUT, RANIN, and RANCK. The error flag table is:

NERROR = 1 EOF or record not in file
= 3 Unsuccessful operation
= 4 Attempted another operation before calling RANCK to check previous operation
= 5 No more room in ID array or ID array has not been specified with at least two core cells
= 6 Incorrect calling sequence—initially written record does not specify length of record or a subsequent attempt to write a record longer than initial record written by RANOUT

Error Message
RECORD LENGTH MUST BE GT 0
ENTRY POINT RANIN

Purpose
To start a buffered read from the random file and immediately return control to the caller.

Fortran Call
CALL RANIN(NAME, ARRAY1, LENGTH, NERROR)

Method
ARRAY1 should not be used before RANCK is called to assure that the read is complete. The arguments are the same as those for RANOUT.

ENTRY POINT RANCK

Purpose
To check for completion of the previous I/O operation.

Fortran Call
CALL RANCK (no arguments)

Method
If the I/O operation is not complete, RANCK will wait until it is. It returns a status for the operation in the NERROR cell. RANCK must be called after each read or write. Only one logical unit may be used at one time; only one I/O operation is buffered.
Computer Sample 4
Program RANREAD
Illustrating Use
of RANRD Subroutines

CARD NUMBER APPROXIMATE LOCATION
0001  000000  *FORTRAN
0002  000000  PROGRAM RANREAD
0003  000000  DIMENSION A(100),B(100),ID(20)
0004  000000  C ID MUST BE DIMENSIONED FIVE TIMES THE NUMBER OF UNIQUELY
0005  000000  NAMED RECORDS WE PLAN TO WRITE
0006  000000  C THE CALL TO RANINT SETS POINTERS AND MUST PRECEDE ANY
0007  000000  CALLS TO RANWT OR RANRD
0008  000000  CALL RANINT(10,20,NERROR)
0009  000000  IF(NERROR.NE.0)GO TO 10
0010  000000  DO 1=2,100
0011  000000  A(1)-.10
0012  000000  DO I=2,100
0013  000000  A(I)=A(I-1)+.1
0014  000000  C ANY 60 BIT CONFIGURATION DEFINES THE RECORD NAMES
0015  000000  FIRST=18
0016  000000  SECOND=1.36E-9
0017  000000  THIRD=MARY
0018  000000  FOURTH=1.0
0019  000000  C ON THE FIRST CALL TO ANY UNIQUE RECORD THE RECORD
0020  000000  LENGTH MUST BE INCLUDED
0021  000000  CALL RANWT(FIRST,A(1),NERROR,10)
0022  000000  IF(NERROR.NE.0)GO TO 11
0023  000000  CALL RANWT(SECOND,A(51),NERROR,20)
0024  000000  C SOMETHING THAT WAS PREVIOUSLY WRITTEN
0025  000000  WITH RANWT MAY NOW BE BROUGHT IN WITH RANRD
0026  000000  CALL RANRD(FIRST,B(1),NERROR)
0027  000000  IF(NERROR.NE.0)GO TO 12
0028  000000  WRITE(6,115)(B(I),I=1,10)
0029  000000  CALL RANWT(THIRD,A(11),NERROR,5)
0030  000000  IF(NERROR.NE.0)GO TO 11
0031  000000  CALL RANWT(FOURTH,A(16),NERROR,25)
0032  000000  C AFTER THE FIRST CALL TO ANY UNIQUELY NAMED RECORD THE
0033  000000  LENGTH OF THE RECORD TO BE WRITTEN MAY BE OMITTED
0034  000000  CALL RANWT(FIRST,A(51),NERROR)
0035  000000  IF(NERROR.NE.0)GO TO 11
0036  000000  WRITE(6,115)(B(I),I=1,10)
0037  000000  CALL RANRD(FOURTH,B(11),NERROR)
0038  000000  IF(NERROR.NE.0)GO TO 12
0039  000000  WRITE(6,115)(B(I),I=1,10)
0040  000000  WRITE(6,115)(A(),I=1,100)
0041  000000  WRITE(6,115)(B(I),I=1,100)
0042  000000  LENGTH OF ROUTINE RANREAD 100035
Computer Sample 4
Program RANREAD
(continued)

VARIABLE ASSIGNMENTS
A - 000000 B - 000144 ID - 000310 MERROR - 000652 N - 000601 FIRST - 000600
SECOND - 000577 THIRD - 000576 FOURTH - 000575

SUBROUTINES CALLED
RANINT RANWD RANRD OUTPTC EXIT

COMPILE TIME = 56
PROGRAM SPACE IS 002456
ENTRY POINT LOCATION ROUTINE ORIGIN
RANREAD 000340 000004 Q0QERR 000641 000641 EXIT 000653 000653 END 000653 000653 STOP 000660 000660 Q0QERR 000660 000660 RANINT 002370 002402 RANWT 002402 002402 RANRD 002405 002301 RANOUT 002405 002301 RANIN 002405 002301 RANCK 002405 002301 RANINT 002370 002301 RANWD 002370 002301 RANRD 002370 002301

THE TOTAL PROGRAM AND BUFFER SPACE REQUIRED IS 002476

TOTAL CPU TIME IN MILLISECONDS = 56
PPU TIME IN MILLISECONDS = 96
PAGES PRINTED = 3
TOTAL RESOURCES USED = .15
**RPTIN/RPTOUT**

**Purpose**
RPTIN is used to read logical records from a tape made by RPTOUT. RPTOUT is used to pack short variable-length logical records into larger physical records on tape.

**Description**
The records are under checksum control. Logical records can be 1 to 352 words long, but the first 12 bits of each logical record must be reserved for use by RPTOUT. Several tapes can be written or read at once, but each unit must have a separate buffer dimensioned in the calling program.

**Storage**
570₈ locations

**Method**
In general, the I/O operators are used by the programmer ignoring that the records are blocked. The exceptions to this are:

1. A special buffer dimensioned 360 must be set aside by the programmer for use by RPTIN/RPTOUT in blocking and unblocking the logical records (reports). There must be a separate buffer for each unit in use at the same time.

2. There is no backspace command.

3. Before putting an end file on a tape, or otherwise terminating writes on a tape, a special RPTOUT command with JL=2 is required (see below).

4. After a rewind or before using an NBUF buffer for another tape, NBUF(1) must be set to zero, and the counters in NBUF may be reinitialized if desired.
ENTRY POINT RPTIN

CALL RPTIN (KUNIT,KBUF,KLOC,KWDS,JJ,KLMAX,JEOF)

The flow of data into or out of the subroutine is:

CALL RPTIN(IN,OUT,OUT,OUT,IN,IN,OUT)

KUNIT

Tape numbers 1-10 (not 5 or 6)

KBUF

An array to use for unpacking the records. Dimension KBUF(360) KBUF(1) must be set to 0 before the first read. RPTIN will start a new read when it sees the 0.

KBUF(2-6) Must be cleared when desired
KBUF(2) Will have a count of logical records read
KBUF(3) Will have a count of physical records read
KBUF(4) Will have a count of words read from the tape
KBUF(5) Will have the words in this physical record
KBUF(6) Will be the same as JEOF

KLOC

Array to put report in.

KWDS

The total number of words in this report.

JJ

= 1 Return only two words of this report and do not move on to the next report (except for EOF case).
= 10 Return full report up to a maximum of KLMAX words and position on next report. (This is the option normally used.)
Fortran Calling Sequence (continued)

KLMAX

Maximum number of words in a report to move to KLOC.

JE0F

= 0 Good report returned.
= 1 EOF.
= 2 Report returned from a record with a bad checksum.

Error Messages

RPTIN-SHORT PHYSICAL RECORD. UNIT, PHYSICAL RECORD, LENGTH, EXPECTED LENGTH, STATUS. (Bad tape read or records were not created by RPTOUT. This diagnostic often simply indicates noise in a record gap.)

RPTIN-BAD PHYSICAL LENGTH. UNIT, PHYSICAL RECORD, LENGTH, EXPECTED LENGTH, STATUS. (Bad tape read or records were not created by RPTOUT.)

RPTIN-BAD CHECKSUM. UNIT, PHYSICAL RECORD, LENGTH, EXPECTED LENGTH, STATUS. (Bad tape read or records were not created by RPTOUT.)

RPTIN-BAD LOGICAL LENGTH. UNIT, LOGICAL RECORD, LENGTH. (Probably the result of one of the previous errors.)
ENTRY POINT RPTOUT

Fortran Calling Sequence

CALL RPTOUT (NUNIT, NBUF, LOCRPT, NWDS, JL)

The calling program provides the data in the arguments labeled IN; the subroutine puts data in the arguments labeled OUT. For this subroutine the calling program is:

CALL RPTOUT(IN, WORK-OUT, IN, IN, IN)

NUNIT
Tape numbers 1-10 (not 5 or 6)

NBUF
An array in which the records are built.
Dimension NBUF(360)

NBUF(1) Must be 0 before the first write on a tape. RPTOUT will start a record when it sees the 0.

NBUF(2-4) Must be cleared if the counters need to be reinitialized.

NBUF(2) Will have a count of logical records output.

NBUF(3) Will have a count of physical records output.

NBUF(4) Will be a count of the words output. But words in reports = NBUF(4) - 2 x NBUF(3).

LOCRPT
Location of the report for output. The first 12 bits of the report will be used by RPTIN/RPTOUT. Any data contained in the first 12 bits of the first word will be destroyed.

NWDS
Number of words in the report.

JL
= 0 Output of this report.
= 2 No report to output; output the reports in NBUF onto the tape. This is used to output the last buffer onto the tape. This also sets NBUF(1) to zero.
Layout of a Physical Record Made by RPTOUT

This is a sample layout of a physical record which has only three logical records with lengths of 18, 5, and 32 words respectively. The record then has the following form,

```
| word count = 57 |
| count = 18 |
| 1st logical record |
| count = 5 |
| 2nd logical record |
| count = 32 |
| 3rd logical record |
| checksum |
```

with the detailed format:

Word 1 Has a right-adjusted binary count of the words in the physical record. In this example the count is $1 + 18 + 5 + 32 + 1 = 57$.

Word 2 The left-most 12 bits contain a count of 18 in binary. This means that there are 18 words of data in the first logical record.

Words 2-19 The remainder of word 2 and all of words 3-19 contain the data in this logical record.

Word 20 Contains a count of 5 in the first 12 bits.

Words 20-24 The data in the second logical record.

Word 25 Contains a count of 32 in the first 12 bits.

Words 25-56 The data in the third logical record.

Word 57 Is a checksum of the data in words 1-56. This sum is made by the equivalent of an add-and-carry-logical instruction.

Error Messages

RPTOUT-BAD PHYSICAL WRITE. UNIT, PHYSICAL RECORD, LENGTH, STATUS. (This usually indicates some problems with the physical tape, tape drive, or machine.)

RPTOUT-BAD LOGICAL LENGTH. UNIT, LOGICAL RECORD NUMBER, LENGTH. (The user tried to write either a zero-length record or a record longer than 352 words.)
SAVEF

Purpose
To save on logical unit 3 the complete internal state of central memory whenever SAVEF is called. SAVEF also makes provision for saving all overlays used with the program and allows the user to write a data file on the end of the save tape.

Description
The SAVEF subroutine is used in jobs requiring more than 15 minutes of machine time. The advantages of such a routine are obvious for long-running programs. Standard procedure is to call SAVEF approximately every 15 minutes. This allows the job to be restarted in the event of a machine malfunction without losing more than 15 minutes of machine time.

Fortran Calling Sequence
FARG=SAVEF(ISAV)
FARG=SAVEF(ISAV,IOVLY)
FARG=SAVEF(ISAV,IOVLY,THISW)

Method
1. FARG=SAVEF(ISAV)

This is the standard method of calling SAVEF where ISAV is either 0 or 1 and one sense switch setting is allowed, as follows:

ISAV = 0
If sense switch 6 is on, save the state of the computation on logical unit 3 and terminate execution. If sense switch 6 is off, continue without interruption.

ISAV = 1
If sense switch 6 is on, save the state of the computation on logical unit 3 and terminate execution. If sense switch 6 is off, save the state of the computation on logical unit 3 and continue the computation.
Setting sense switch 6 allows the operator to stop a program containing a SAVEF call after a given length of real time, or in case of any emergency, provided that the programmer has made such allowances by checking for sense switch 6 at regular intervals which are shorter than the regular save intervals. The check is placed at a logical stopping point in case sense switch 6 is set.

When restarting from a save tape, all logical units are at the load point and must be repositioned by the user.

2. FARG=SAVEF(ISAV,IOVLY)

This calling sequence is used when a program contains overlays and segments. (See description of subroutine OVERLAY, p. 3.61.)

   ISAV
       As described in calling sequence 1 above.

   IOVLY = 0
       A flag to the save program indicating that there are no overlays to be moved from the drum to the save tape.

   IOVLY = 1
       A flag to the save program indicating there are overlays to be moved to the save tape.

3. FARG=SAVEF(ISAV,IOVLY,IHISW)

This calling sequence is used when a program is to write an additional file of history data to be used for initialization purposes at restart time. Provision has been made in the SAVEF program to jump to a user-constructed program which will write a history file on logical unit 3. When the IHISW option is in effect, this calling sequence is used whether or not a program contains overlays. The IOVLY flag is set for the appropriate condition in the case being run.
SAVEF
(Continued)

ISAV
As described in calling sequence 1 above.

IOVLY
As described in calling sequence 2 above.

IHISW = 0
No additional information is to be added to the save tape.

IHISW = 1
Jump to a user program called WTHIS, which will write a logical file of data on the save tape starting at its current position.

Rules
• The program which calls SAVEF must have a tape assignment to logical unit 3 such as *ASSIGN,B021=3. B021 is a tape in the user's name, assigned to him by the operator. This tape will be used to restart the job which has been saved on this tape.

• Logical unit 3 is reserved for the exclusive use of SAVEF. When more than one argument is used with SAVEF, logical unit 3 may not be the drum.

• The operators must be supplied with a restart deck. A restart deck consists of the same control cards as were used with the program which called SAVEF, with the following changes:
  a. Replace the *RUN card with the *RESTART card.
  b. Add a *LIMIT,C=NNNNN card to specify the core reservation limit. This number is equal to or greater than the total core used by the run which made the save tape. The amount of core needed may be determined from the loader map when the job is first saved.
- A time limit and output limit are specified on a *LIMIT card in the restart deck.

**Example**

```
*JOB,5000,F0540004,SMITH H
*ASSIGN,S1315
*LIMIT,T=30,C=101023,PR=300
*RESTART
   5  6  3  789
   2.6698E-04  56332  54
*END
```

- SAVEF may be called from an overlay but not from a segment.

- If SAVEF is called from an overlay, it must be loaded with the control program.

- If overlays are used with SAVEF, the save tape will have the following format:
  
  a. First logical file--central memory
  b. EOF mark
  c. Second logical file--overlays and their associated segments
  d. EOF mark (written by SAVEF)

- If overlays and the history file option are used with SAVEF, the save tape will have the following format:
  
  a. First logical file--central memory
  b. EOF mark
  c. Second logical file--overlays and their associated segments
  d. Third logical file--user-constructed data file
  e. EOF mark (written by SAVEF)
SAVEF
(Continued)

Rules
(continued)

• If only the history file option is used with SAVEF, the save tape will have the following format:
  a. First logical file—central memory
  b. EOF mark
  c. Second logical file—user-constructed data file
  d. EOF mark (written by SAVEF)

• If the history file option is used (i.e., THISW=1), then the user must furnish his own subroutine called WTHIS. SAVEF will do a RJ to this routine. The routine may perform any operations necessary to move data onto the save tape. The normal return will enable SAVEF to finish. (Do not rewind the save tape.)

• When the history file is used, the following sequence of events will take place at restart time:
  a. The first logical file containing central memory will be moved to central memory.
  b. If overlays were used, they will be moved to the drum.
  c. Control will be returned to the program which called SAVEF at the actual statement which performed the call.
  d. The save tape will now be positioned at the beginning of the logical data file previously written by the user's WTHIS subroutine.
  e. It is up to the user's program to move the data file back from the save tape to its original storage position.
  f. Computation will continue as before.

• The call statements should be placed at the logical termination of a calculation cycle and before another cycle is started.
• In some cases it is necessary to program special logic for restart purposes, since on a restart the program is started back through the FARG=SAVEF(ISAV) statement.

• FARG is a floating-point argument which is non-zero when the program is being restarted through the use of a restart deck.

• FARG equals zero upon return from all other calls.
There are two argument returns from an FARG=SAVEF(ISAV) type of call:

1. The normal return, where execution did not stop and calculations are to continue in a normal fashion:
   
   \[ \text{FARG} = 0 \text{ for ISAV} = 0 \text{ or ISAV} = 1 \]

2. The restart return, where control is being returned to the calling program after a restart:
   
   \[ \text{FARG} \neq 0 \text{ for ISAV} = 0 \text{ or ISAV} = 1 \]

Error Messages

- CHECKSUM ERROR DURING RESTART PHASE
- DRUM READ ERROR
- DRUM WRITE ERROR
- ERROR IN READING OVERLAYS FROM DRUM
- ERROR READING LOM DATA FROM SAVE TAPE
- ERROR READING OVERLAYS FROM PROGRAMMER LIB. FILE
- ERROR WRITING OVERLAYS ON PROGRAMMER LIBRARY FILE
- FAILED TO READ RESIDENT OVERLAY FROM DRUM
- FAILURE TO WRITE RESIDENT OVERLAY TO DRUM
- SW6 TERM BY SAVE TAPE READ ERROR
- TAPE WRITE ERROR
- UNABLE TO ESTABLISH PROGRAMMER FILE
- UNABLE TO ESTABLISH RANDOM FILE BUFFER ON RESTART

Storage

6338 locations

Summary

The following table summarizes sense switch setting and ISAV=0 or =1 in using the SAVEF function:

<table>
<thead>
<tr>
<th>Sense Switch 6</th>
<th>ISAV = 0</th>
<th>ISAV = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Save and terminate</td>
<td>Save and terminate</td>
</tr>
<tr>
<td>OFF</td>
<td>No action (program continues)</td>
<td>Save (program continues)</td>
</tr>
</tbody>
</table>
SORT

Purpose
To sort an array A (or K) in ascending order from A(I) to A(J).

Fortran Function
CALL SORT (A,I,J)

Entry Point
SORT

Ascent Calling Sequence
RJ SORT
EQ $^{*+4}$
CON A
CON I
CON J

Normal Return
A is reordered in ascending order.

Error Message
None

Storage
2128 locations

Accuracy
Not applicable

Timing
Depends on the size of the array (6600: 15 usec for 200 elements of A)

Method
SORT will sort Hollerith data in alphabetic order. For each column, the order will be: (1) letters A-Z, (2) numbers 0-9, (3) blanks, (4) dashes.

The array may be fixed or floating-point. The ordering is done by integer subtraction. If the array is floating-point, the numbers must be in normalized form. The array may be dimensioned for $2^{16}$ elements.
Arguments:

A = The array to be sorted.

I = The first element of the array to be sorted.

J = The last element of the array to be sorted.

I should be less than J. The array is repeatedly split into segments to create a fast sort.
**SSW**

**Purpose**
To set and test sense switches and sense lights.

**Fortran Reference**
To test the sense switch or light use:

```
IF (SENSE SWITCH i)n_1, n_2
IF (SENSE LITE i)n_1, n_2
```

To set the sense light use:

```
SENSE LITE i
```

Sense switches are set at the console and cannot be set by the programmer.

**Entry Points**
SSW, SLT

**Storage**
248 locations

*Note:* This routine maintains compatibility with early machines. It is recommended that other control parameter logic be substituted for sense lights and switches.
SYSRCL

Purpose
This routine recalls the operating system from an executing program in order that a subsequent compilation, loading, and execution can be done.

Fortran Call
CALL SYSRCL

Ascent Call
RJ SYSRCL

Sample Deck Setup
*JOB,............
*FORTRAN (Program 1)
  :
  CALL SYSRCL
  :
  *RUN
    data1
*FORTRAN (Program 2)
  PROGRAM NAME
  :
  CALL EXIT
  END
  *RUN
    data2
  *END

Storage
428 locations
Information may be communicated between Program 1 and Program 2 using logical units 6762B-6764B inclusive. For example, Program 1 could write a tape that Program 2 would read by using logical units 6762B-6764B. These units must not be assigned and can only be referenced by BUFFER statements or Ascent routines using central monitor calls. If Program 1 does not read all of the data, Program 2 may never be reached. A control card option, S=PSCR, causes system input to be read from unit 6762B. However, information on the PSCR file must be in system Cosy format. There are routines available for this purpose.
ULIB

Purpose

To reset (rewind) a file on ULIB and to read records from the file. To reset a file call ULIBREW. To read a record call ULIBRD. The read is a buffered routine; i.e., the operation starts and control returns to the caller. To check that the operation is complete, call ULIBCK.

Entry Points

ULIBREW, ULIBRD, ULIBCK

Storage

1078 locations

Notes:

• No assign card is necessary for ULIB.

• A particular file on ULIB can be thought of as a tape. Before each sequence of reads to a file, ULIBREW should be called to position the file at the first record. Care should be taken when reading several records on two or more files. All the records of one particular file should be read before the next file is reset and read. That is, one cannot interlace reading records to several files.

• Because ULIB is a read-only library, the user may not create or change any files. To have a file put on ULIB contact the librarian.
ENTRY POINT ULIBREW

Fortran Call

CALL ULIBREW(NAME,NERR)

This call initializes a file or repositions the file at the first record.

NAME

Name of the file, which can be any 60-bit configuration. If, however, later use will be made of the editing features of the system, DPC characters left-adjusted with blank fill should be used. NAME = 'HFL23 where FL23 is the file name.

NERR

A location the subroutine will use to pass back an error code. NERR = 0 means the call was successful. NERR ≠ 0 means the call was improper or the subroutine is unable to do the request. NERR error codes are:

NERR = 0  Successful operation
          = 1  End-of-file, or attempted to read more records than available in file
          = 3  Unsuccessful operation
          = 4  Attempted another read before checking previous one


ENTRY POINT ULIBRD

Fortran Call

CALL ULIBRD(NAME, ARRAY, LENGTH, NERR)

This call will read one record from the file called NAME. Call ULIBREW before reading the first record from a file.

NAME
As in ULIBREW

ARRAY
Field of words to be filled by the read

LENGTH
Number of words to be read into array

NERR
As in ULIBREW

ENTRY POINT ULIBCK

Fortran Call

CALL ULIBCK(LNTH)

This call should be made after each read to check that the read sequence is complete and successful. If the sequence is not complete, ULIBCK will wait until it is. If the read was unsuccessful, ULIBCK will put an error code in the NERR location of the read call.

LNTH
A location in which ULIBCK will put the actual number of words read or written. This is useful when reading records shorter than expected or when a failure has occurred in a read.
UNLOAD

Purpose To unload the tape units given in the calling sequence.

Fortran Call CALL UNLOAD(I_1,I_2,...) (Variable argument list)

Entry Point UNLOAD

Ascent Calling Sequence RJ UNLOAD
EQ *+N Number of arguments
CON I
CON J

Normal Return To program execution

Error Message None

Storage 178 locations

Method The central monitor issues the rewind unload command.
A list of entry points into the PLOT routine described in this chapter follows. The routine is divided into the following categories: scaling, plotting, writing, control, backgrounds, and plot instruction manipulation. Information on instruction format is included at the end of this chapter, as well as other useful plotting routines in the ULIB file.

The dd80C is a cathode ray tube (CRT) plotter that presents the computer output in a graphic display. Designed to run on-line with the Control Data 6600/7600 computer, it comprises a CRT, a 35-mm microfilm camera, and a logical control section.

A 1024 by 1024 horizontal (X) and vertical (Y) coordinate system controls plotting beam positioning. Displays are of three types: lines, points, and symbols. Symbols are drawn by a character generator in the hardware, and may be of four sizes, two orientations, and two intensities, with the additional option of italics in one direction. Available symbols include the entire alphabet in capitals and lower case, the digits 0-9, the Greek alphabet, and a variety of punctuation and special symbols.
## GENERAL DESCRIPTION

(Continued)

<table>
<thead>
<tr>
<th>PLOT Entry Points</th>
<th>Arguments</th>
<th>Alternate Arguments</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling</strong></td>
<td></td>
<td></td>
<td>4.6</td>
</tr>
<tr>
<td>SET</td>
<td>XA,YA,YB,XC,YD,YTYP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSCALE</td>
<td>SCALEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PORGN</td>
<td>X,Y</td>
<td>MX,MY</td>
<td></td>
</tr>
<tr>
<td><strong>Plotting</strong></td>
<td></td>
<td></td>
<td>4.12</td>
</tr>
<tr>
<td>FRSTPT</td>
<td>X,Y</td>
<td>MX,MY</td>
<td></td>
</tr>
<tr>
<td>VECTOR</td>
<td>X,Y</td>
<td>MX,MY</td>
<td></td>
</tr>
<tr>
<td>POINT</td>
<td>X,Y</td>
<td>MX,MY</td>
<td></td>
</tr>
<tr>
<td>PSYM</td>
<td>X,Y,ICHR,ISIZ,ICAS,IP</td>
<td>MX,MY,ICHR,ISIZ,ICAS,IP</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td>XA,YA,XB,YB</td>
<td>MXA,MYA,MXB,MYB</td>
<td></td>
</tr>
<tr>
<td>CURVE</td>
<td>XAD,YAD,N</td>
<td>MXAD,MYAD,N</td>
<td></td>
</tr>
<tr>
<td><strong>Writing</strong></td>
<td></td>
<td></td>
<td>4.16</td>
</tr>
<tr>
<td>PWRT</td>
<td>X,Y,LOC,N,ISIZ,IOR</td>
<td>MX,MY,LOC,N,ISIZ,IOR</td>
<td></td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td></td>
<td></td>
<td>4.22</td>
</tr>
<tr>
<td>FLUSH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>ICAS,INT,ITAL,IOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DASHLN</td>
<td>IPATRN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMXY</td>
<td>MX,MY</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Backgrounds</strong></td>
<td></td>
<td></td>
<td>4.29</td>
</tr>
<tr>
<td>GRID</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRIDL</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERIM</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERIML</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALFAX</td>
<td>MGRX,MINRX,MGRY,MINRY, X,Y,TXLAB,TYLAB</td>
<td>MGRX,MINRX,MGRY,MINRY, X,Y,TXLAB,TYLAB</td>
<td></td>
</tr>
<tr>
<td>AXES</td>
<td>X,Y</td>
<td>MX,MY</td>
<td></td>
</tr>
<tr>
<td>GRIDALL</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ticks</td>
<td>MAJOR,MINOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICK4</td>
<td>MGRX,MINRX,MGRY,MINRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LABMOD</td>
<td>IFMTX,IFMTY,NUMX,NUMY,ISIZX, ISTIZY,IXDEC,TYDEC,IXOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Plot Instruction Manipulation</strong></td>
<td></td>
<td></td>
<td>4.41</td>
</tr>
<tr>
<td>FLASH1</td>
<td>IFIELD,MAXLEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH2</td>
<td>IFORM,LENGTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH3</td>
<td>IFORM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH4</td>
<td>IFWA,LWA,IFORM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LTPEN</td>
<td>ID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notes on arguments X, Y and MX, MY:

Wherever the coordinate arguments of a routine are X and Y (floating point), alternate arguments MX and MY (integer) may be used.

The floating-point arguments are used to determine a CRT address using the following formulas:

\[
\text{CRT ADDRESS (i) = } a^X \text{ (or logX)} + b \\
\text{CRT ADDRESS (j) = } c^Y \text{ (or logY)} + d
\]

The CRT addresses are integers in the range \(0 \leq \text{CRT address} \leq 1023\).

The a, b, c, and d are established initially as 1023., 0., 1023., and 0., respectively, and can be modified by SET, PORGN, and PScale. (See p. 4.7 for discussion of a, b, c, and d.)

The integer arguments determine the i and j by the following formulas:

\[
\text{CRT ADDRESS (i) = MX - 1} \\
\text{CRT ADDRESS (j) = MY - 1}
\]

Note that while the range of CRT addresses (i and j) is 0 \(\rightarrow\) 1023, the range of MX and MY is 1 \(\rightarrow\) 1024. This is because it is impossible to distinguish between an integer zero and a floating-point zero on the 6600/7600. The routines, therefore, always treat a zero as a floating-point number.
Example

POINT may be called in any of these ways:

CALL POINT (X,Y)   CALL POINT (X,MY)
CALL POINT (MX,Y)   CALL POINT (MX,MY)

The alternate arguments MX and MY apply to the X and Y coordinates of the following routines:

PORGN    CURVE
FRSTPT   PWRT
VECTOR   HALFAx
POINT    AXES
PSYM     GRIDALL
LINE

4.4
Plotting Library Routines
**PLOT**

**Purpose**
To provide graphic output. (PLOT is written in machine language and is on the library tape.)

**Fortran Functions**
See list on p. 4.2

**Entry Points**
PLOT currently has 30 entry points as listed above; 10 of them have alternate argument lists.

**Error Messages**
ARG 5-8 IN SET ZERO OR NEGATIVE WITH LOG SCALING
FLASH BUFFER OVERFILL
OPERATOR DROP OF DD80 ONLINE JOB (DD80 FAULT)
TRIED TO START FLASH WHILE IN FLASH
X OR Y COOR ZERO OR NEGATIVE WITH LOG SCALING

**Storage**
21638 locations

**Timing**
Variable, depending on what is plotted.
The scaling routines establish a mapping between the user's Cartesian X,Y plane and the dd80's I,J plane. The I,J plane is defined by 1024 possible addresses in I and J.

The mapping equation used is one of the following:

1. \( i = ax + b \)  \( j = cy + d \)
2. \( i = a \log(x) + b \)  \( j = c \log(y) + d \)

b and d are initially set to 0.; the type of mapping is linear-linear.
SET

Purpose
To establish a mapping of the type specified by LTYPE between the area of the CRT given by XA,XB,YA,YB and a user's plane with an X range of XC (left) to XD (right) and a Y range of YC (bottom) to YD (top).

Note: This scaling will be in effect for all subsequent plotting until changed by another call to SET, PORGN, or PScale.

Arguments
XA,XB,YA,YB,XC,XD,YC,YD,LTYPE

XA,XB,YA,YB (floating)
Numbers between 0. and 1. defining the portion of the CRT to be used. (0.,1.,0.,1.) specifies the entire area. The following restrictions apply:

0. ≤ XA < XB ≤ 1.
0. ≤ YA < YB ≤ 1.

Alternate arguments MXA, MXB, MYA, MYB may be used. These specify CRT addresses, as explained on p. 4.3.

XC,XD,YC,YD (floating)
Numbers indicating minimum and maximum values of the array to be plotted in the CRT area specified by arguments 1-4.

XC maps to XA
XD maps to XB
YC maps to YA
YD maps to YB

In either linear or log mapping, the only restrictions are that

XC ≠ XD
YC ≠ YD

That is, XD may be larger or smaller than XC; similarly, YD may be larger or smaller than YC.
## SET

(Continued)

### Arguments

(continued)

<table>
<thead>
<tr>
<th>LTYPE</th>
<th>X Direction</th>
<th>Y Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>linear</td>
<td>linear</td>
</tr>
<tr>
<td>2</td>
<td>linear</td>
<td>log</td>
</tr>
<tr>
<td>3</td>
<td>log</td>
<td>linear</td>
</tr>
<tr>
<td>4</td>
<td>log</td>
<td>log</td>
</tr>
</tbody>
</table>

### Example

CALL SET (.1,.8,.3,.5,0.,20.,10.,1000.,2)

<table>
<thead>
<tr>
<th>Linear Case (X)</th>
<th>Log Case (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINX = .1 x 1023. = 102</td>
<td>MINY = .3 x 1023. = 307</td>
</tr>
<tr>
<td>MAXX = .8 x 1023. = 818</td>
<td>MAXY = .5 x 1023. = 512</td>
</tr>
<tr>
<td>Range of CRT addresses (RCRT) is 818 - 102 = 716</td>
<td>RCRT = 512 - 307 = 25</td>
</tr>
<tr>
<td>Range of calling addresses (RCA) is 20. - 0. = 20.</td>
<td>RCA = log 10000, - log 10, = 3.0</td>
</tr>
<tr>
<td>X scale factor (a) then c = $\frac{205}{3} = 68.33$ equals $\frac{716}{20}$. = 35.8</td>
<td></td>
</tr>
<tr>
<td>X zero point address (b) is 102 = b + 35.8*$d$</td>
<td>307 = d + 68.33*$\log(10)$</td>
</tr>
<tr>
<td>b = 102</td>
<td>d = 239</td>
</tr>
</tbody>
</table>
The figure shows the mapping described in the example:
PScale

Purpose To set directly the X and Y scale factors (a and c) in one of the following mapping equations:

1. \( i = ax + b \quad j = cy + d \)
2. \( i = a \log x + b \quad j = c \log y + d \)

a and c are initially set to 1023.; mapping is linear-linear.

Arguments SCALEX,SCALEY (floating)

SCALEX
X scale factor

SCALEY
Y scale factor

Method
a = SCALEX
c = SCALEY
PORGN

Purpose
To set new zero point addresses (b and d) in the mapping equations:

1. \( i = ax + b \) \hspace{1cm} j = cy + d
2. \( i = a\log x + b \) \hspace{1cm} j = c\log y + d

b and d are initially set to 0.; mapping is linear-linear.

Arguments
\( X,Y \) (floating)

Coordinates in terms of current scale factors and zero point addresses of desired new zero point addresses.

Method
\[
\begin{align*}
    b(\text{new}) &= b(\text{old}) + a\times X \\
    d(\text{new}) &= d(\text{old}) + c\times Y
\end{align*}
\]

PORGN (MX,MY)

Alternate routine for PORGN.

Purpose
To set zero-point addresses using the I,J coordinate system.

Arguments
\( MX,MY \) (integer)

New zero point coordinates in range

\[ 1 \leq MX \text{ or } MY \leq 1024 \]

Method
\[
\begin{align*}
    b &= MX - 1 \\
    d &= MY - 1
\end{align*}
\]
The plotting routines enable the user to draw points, lines, and individual symbols. Scaling is as specified by initial conditions or by the scaling routines. When LTYPE is other than 1, the appropriate log of the calling number is plotted if floating-point coordinates are specified (see notes on p. 4.3).

**FRSTPT**

**Purpose**
To establish the base point of a line. No plotting is done by FRSTPT.

**Arguments**

X,Y (floating)

Coordinates of the beginning point of a line

**VECTOR**

**Purpose**
To plot a straight line segment from the (X,Y) coordinates of the immediately previous plotter instruction to the coordinates given by (X,Y). A curve is often drawn with an initial call to FRSTPT followed by as many successive calls to VECTOR as there are points along the line.

**Arguments**

X,Y (floating)

Coordinates of an end point of a line
POINT (X,Y)  

X,Y (floating)  
Coordinates of point to be plotted  

Purpose  
To plot a point at (X,Y).  

PSYM  

Purpose  
To draw the symbol specified by ICHR and ICAS of size ISIZ at (X,Y) connected by a line to previous plot point if IP = 2. If an octal number is used instead of a 1H (character), it should be left-justified within the word. Intensity, orientation, and italics may be controlled by a prior call to OPTION.  

Arguments  
X,Y,ICHR,ISIZ,ICAS,IP  
X,Y (floating)  
Coordinates of point  
ICHR (Hollerith)  
Character to be plotted  
ISIZ (integer)  
Size of character (see Table 1, p. 4.18)  
ICAS (integer)  
0 = upper case; 1 = lower case  
IP (integer)  
1 = symbol only; 2 = symbol plus line to symbol from most recent plot point
LINE

Purpose To draw a line from (XA,YA) to (XB,YB).

Arguments XA,YA,XB,YB (floating)

XA,YA
Coordinates of beginning point

XB,YB
Coordinates of end point

CURVE

Purpose To draw a series of line segments connecting the N points specified in the arguments. The routine assumes the X and Y arrays are ordered. If integers are in the array, they will be plotted without scaling.

Arguments XAD,YAD,N

XAD (floating)
An array of X coordinates

YAD (floating)
An array of Y coordinates

N (integer)
The number of points on the curve
Computer Sample 5

Example of plotting routine use.

Program CIRCLE

Illustrating
Use of dd80

SUBROUTINE CIRCLE
DIMENSION X(100), Y(100)
CALL GRID(1,1,1,1)
PI = 3.1416
R = 3.
DO I = 1, 73
X(I) = R * COS(D * I)
Y(I) = R * SIN(D * I)
1 CALL PORGN(300,400)
CALL PSCALE(70.,50.)
CALL CURVE(X,Y,73)
CALL PCINT(0.,0.)
CALL PORGN(600, 700)
CALL PSCALE(100.,100.)
CALL FRSTPT(X(1), Y(1))
DO Z = 2, 73
2 CALL VECTOR(X(I), Y(I))
CALL LINE(600,700, X(1), Y(1))
CALL PORGN(100, 200)
CALL PSCALE(30.,30.)
CALL PSYM(0.,900,1HP,3,0,1)
CALL CURVE(X,Y,73)
CALL FRAME
RETURN
END
**WRITING**

**PWRT**

**Purpose**
To permit writing Hollerith information on the dd80. 118 characters are available; they may be written in any of four sizes, two orientations, two intensities, and in either block print or italics.

**Arguments**

- **X,Y** (floating)
  Coordinates of the center of the first character to be written

- **LOC** (integer)
  The display code (DPC) array to be written

- **N** (integer)
  The number of characters to be written. LOC should be dimensioned appropriately if more than 10 characters are to be written. All control characters and function codes are included in the count.

- **ISIZ** (integer)
  Character size (see Table 1, p. 4.18)

- **IOR** (integer)
  Character (and line) orientation (see Table 2, p. 4.18)
There are three other variables which can be preset and affect the tabular output: intensity (INT), case (ICAS), and italics (ITAL) (see Table 2, p. 4.18). These variables are set by a call to OPTION. ICAS can also be set by a call to PSYM. Whenever one of these variables is set, it remains unchanged until modified by the calling program.

Note: When using the dd80 as a line printer with a *TRAP or a WRITE statement assigned to the dd80, the line limit is 127 characters (128 characters is the printer limit).
### Table 1
**CHARACTER SIZES**

<table>
<thead>
<tr>
<th>ISIZ</th>
<th>Maximum Char./Line</th>
<th>Maximum Lines/Frame</th>
<th>Increment Along Line</th>
<th>Increment Between Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>64</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>86</td>
<td>43</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>22</td>
<td>24</td>
<td>48</td>
</tr>
</tbody>
</table>

### Table 2
**CHARACTER CONTROLS**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Controls</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Controls</td>
<td>0 (Normal)</td>
<td>1</td>
</tr>
<tr>
<td>IOR</td>
<td>Orientation</td>
<td>Writes in +X direction</td>
<td>Writes in +Y direction</td>
</tr>
<tr>
<td>INT</td>
<td>Intensity</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>ICAS</td>
<td>Case</td>
<td>Upper</td>
<td>Lower</td>
</tr>
<tr>
<td>ITAL</td>
<td>Italics</td>
<td>Block print</td>
<td>Italics (permissible only with IOR = 0)</td>
</tr>
</tbody>
</table>

### Table 3
**INTERNAL FLAGS FOR PWRT**

<table>
<thead>
<tr>
<th>Tab Bit</th>
<th>Case (ICAS)</th>
<th>Carriage Return</th>
<th>Italics (ITAL)</th>
<th>Not Used</th>
<th>High Intensity (INT)</th>
<th>Octal Value</th>
<th>Hollerith (PWRT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>04</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>05</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>14</td>
<td>$4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>15</td>
<td>$5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>25</td>
<td>E</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>H</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>31</td>
<td>I</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>34</td>
<td>)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>35</td>
<td>$E</td>
</tr>
</tbody>
</table>

Bit = 1 means YES; Bit = 0 means NO.
### Table 4

#### dd80 CHARACTER SET

<table>
<thead>
<tr>
<th>dd80 Characters</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
<th>dd80 Characters</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>00</td>
<td>-</td>
<td>6</td>
<td>τ</td>
<td>6</td>
<td>41</td>
</tr>
<tr>
<td>A</td>
<td>a</td>
<td>01</td>
<td>21</td>
<td>7</td>
<td>θ</td>
<td>7</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>02</td>
<td>22</td>
<td>8</td>
<td>χ</td>
<td>8</td>
<td>43</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>03</td>
<td>23</td>
<td>9</td>
<td>ψ</td>
<td>9</td>
<td>44</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>04</td>
<td>24</td>
<td>+ blank</td>
<td>+</td>
<td>45</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>05</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>06</td>
<td>26</td>
<td>×</td>
<td>×</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td>G</td>
<td>g</td>
<td>07</td>
<td>27</td>
<td>/</td>
<td>/</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>H</td>
<td>h</td>
<td>10</td>
<td>30</td>
<td>( blank</td>
<td>(</td>
<td>51</td>
<td>74</td>
</tr>
<tr>
<td>I</td>
<td>i</td>
<td>11</td>
<td>31</td>
<td>) flag only</td>
<td>)</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>J</td>
<td>j</td>
<td>12</td>
<td>41</td>
<td>$</td>
<td>$</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>k</td>
<td>13</td>
<td>42</td>
<td>=</td>
<td>=</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>L</td>
<td>l</td>
<td>14</td>
<td>43</td>
<td>blank blank</td>
<td>blank</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>15</td>
<td>44</td>
<td>, blank blank</td>
<td>,</td>
<td>56</td>
<td>73</td>
</tr>
<tr>
<td>N</td>
<td>n</td>
<td>16</td>
<td>45</td>
<td>. blank blank</td>
<td>.</td>
<td>57</td>
<td>33</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>17</td>
<td>46</td>
<td>$2 blank</td>
<td>$2</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>P</td>
<td>p</td>
<td>20</td>
<td>47</td>
<td>$6 blank blank</td>
<td>$6</td>
<td>61</td>
<td>16</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>21</td>
<td>50</td>
<td>$5 blank blank</td>
<td>$5</td>
<td>62</td>
<td>15</td>
</tr>
<tr>
<td>R</td>
<td>r</td>
<td>22</td>
<td>51</td>
<td>&lt; function code</td>
<td>&lt;</td>
<td>63</td>
<td>17</td>
</tr>
<tr>
<td>S</td>
<td>s</td>
<td>23</td>
<td>62</td>
<td>$E</td>
<td>$E</td>
<td>65</td>
<td>35</td>
</tr>
<tr>
<td>T</td>
<td>t</td>
<td>24</td>
<td>63</td>
<td>$B</td>
<td>$B</td>
<td>66</td>
<td>32</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>25</td>
<td>64</td>
<td>$G</td>
<td>$G</td>
<td>67</td>
<td>37</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>26</td>
<td>65</td>
<td>$F</td>
<td>$F</td>
<td>70</td>
<td>36</td>
</tr>
<tr>
<td>W</td>
<td>w</td>
<td>27</td>
<td>66</td>
<td>$K</td>
<td>$K</td>
<td>71</td>
<td>52</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>30</td>
<td>67</td>
<td>$O</td>
<td>$O</td>
<td>72</td>
<td>56</td>
</tr>
<tr>
<td>Y</td>
<td>y</td>
<td>31</td>
<td>70</td>
<td>$P</td>
<td>$P</td>
<td>74</td>
<td>57</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>32</td>
<td>71</td>
<td>none</td>
<td>none</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>33</td>
<td>00</td>
<td>$v</td>
<td>$v</td>
<td>76</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>34</td>
<td>01</td>
<td>$S</td>
<td>$S</td>
<td>77</td>
<td>72</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>35</td>
<td>02</td>
<td>$4</td>
<td>$4</td>
<td>78</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>36</td>
<td>03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>37</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>40</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is possible to modify intensity, case, and italics while writing the array. In addition, the writing can be forced to start a new line at any time (the new line starts directly under the beginning of the previous line) by signaling the d980 with an octal 77 that the next six bits are function control bits. The apostrophe on the keypunch is interpreted as a 778. The proper six bits to follow the 778 may be selected from Table 3 (p. 4.18).

If the tab bit is on, the arithmetic sum of the lower five bits (from 0 to 3110) is multiplied by 32 and the resulting number is used as the new coordinate along the line (X in 0 orientation, Y in 1 orientation). When the tab bit is used, the case, carriage return, italics, and intensity are not changed.

Example

To write

This is the WAY to CHANGE INTENSITY, Style and case.

the DPC array to PWRT should be

T'+HIS IS THE '+WAY'+ TO 'OCHANGE '+INTENSITY,S'+ETYLE AND '+CASE '+D$S
Notes on LOC:

LOC is the display code (DPC) array to be written. Each character is defined as a six-bit binary constant. The dd80 uses an entirely different code from DPC but also has a six-bit constant for each character. There are only 48 legal DPC characters (convertible from keypunch data) and 63 legal dd80 codes. To make the full dd80 set available, a $ convention has been adopted. For instance, to get a $ (ICAS = 0) on the dd80, the two characters $P are used. The $ itself is not written on the dd80 but causes 108 to be added to the dd80 code for P(47), giving a 57, the dd80 code for $A. The disadvantage to this system is that the $ itself is never available as a character during FWRT. (It is, however, available as a single character in PSYM.)

If LOC is defined as an octal constant(s) in the program instead of a Hollerith set, the $A could be directly expressed as a 748.

When calculating the number of characters (argument N), all flags and the $ must be included, even though they do not appear in the final output.

The four easiest ways to generate LOC are:

1. Read it in directly as data in an A format specification
2. Define it arithmetically or in a DATA statement as a Hollerith or octal constant
3. Encode it. For example, the following calculation is made in the sequence shown:

   DIMENSION LTTT(3)
   Z = SINF(A)
   ENCODE (26,100,LTIT) A,Z
   100 FORMAT (10HTHE SIN OF,F6.2,3H IS,F7.3)
   CALL FWRT (X,Y,LTIT,26,ISIZ,IOR)

4. Use it directly as an argument in FWRT, for example,

   CALL FWRT(X,MY,19HTHIS IS AN ARGUMENT,19,2,0)
CONTROL

To increase execution speed, plotter instructions from the PLOT routine are not sent directly to the dd80. Instead, up to 425 instructions are collected in an array (or buffer) internal to PLOT and are then sent to the system output file in the dd80.

This procedure has two potential disadvantages. First, when the program terminates execution, there may be instructions left in the array which will not reach the output file (dd80) unless a specific request is made to send them there. Routine FLUSH and each call to FRAME (which calls FLUSH after generating the frame advance instruction) will clear the array.

- Any program using the PLOT package should have as its last call either FLUSH or FRAME.

Second, if printer output is assigned to the dd80, plotter instructions generated by the PLOT routine will intermix with instructions generated by WRITE or PRINT, and the instruction sequence in the output file will be different than intended.

- To maintain the correct order, a FLUSH or FRAME call should precede each change from PLOT instructions to PRINT or WRITE instructions.
Care should be exercised in controlling whether the printing assigned to the dd80 is on the same frame as the plots.

- If the last dd80 instruction to reach the system output file (not the PLOT buffer) was from PLOT or if there have been no previous printer lines, printout will begin at the top of a new page (line one).

- A frame advance is issued only if the carriage character requests it (LHL), or the line count reaches 64.

The PLOT buffer may be bypassed altogether by use of the FLASH routines which cause plotter instructions to be accumulated in a user-designated array and sent to the output file only when requested by the user (FLASH3) (see p. 4.42 for details).
FLUSH

Purpose
To send all calculated but unexecuted plotter instructions to the dd80.

Description
PLOT has a 255-word instruction buffer which holds 425 plotter instructions. This buffer is normally dumped to the output file dd80 only when full. FLUSH forces this buffer to dump to the output file when it is not full. FLUSH or FRAME must be called at the end of plotting in any program to make sure all calculated instructions actually reach the dd80. It should also be called before switching from graphic output to printed output assigned to the dd80 by an *ASSIGN card.

FRAME

Purpose
To advance the film one frame. Since FRAME calls FLUSH, it also forces the PLOT buffer to dump to the output file.

Arguments
None
Although the following routines do not plot, they are included here because they control some special hardware and software capabilities.

**OPTION**

**Purpose**

To set flags for the listed options. ICAS, ITAL, and IOR apply only to symbols and text (PSYM and PWRT). INT applies to all plotting. The normal value of all these flags is 0. Once set, these flags are not changed until another OPTION call is made or a call to a routine using the flag as an argument occurs.

*Note:* GRID, PERIM, GRIDL, PERIML, GRIDALL, and HALFAX leave the dd80 in high intensity.

**Arguments**

ICAS, INT, ITAL, IOR (integer)

ICAS
- 0 = upper case
- 1 = lower case

INT
- 0 = low intensity
- 1 = high intensity

ITAL
- 0 = block print
- 1 = italics

IOR
- 0 = +X direction (horizontal)
- 1 = +Y direction (vertical)
DASHLN

Purpose
To make a variety of dashed line types available by controlling the blank-unblank circuitry in the dd80. A 1 bit is an unblank; a 0 bit is a blank.

Description
The pattern set by DASHLN is the same for all plotting until changed by another DASHLN call or a call to PWRT (which sets the pattern back to 17778). For example,

- A bit pattern of 17778 (=11111111112) makes a solid line.
- A bit pattern of 14308 (=11000110002) makes a line 2/10 solid, 3/10 blank, 2/10 solid, 3/10 blank.

The DASHLN feature is based on the timing clock of the dd80 and different line lengths are a function both of the number of clock cycles and of the beam speed. Therefore the individual dashes from a given DASHLN pattern will not necessarily be the same length in line segments of different lengths.

Argument
IPATRN (integer)
A 10-bit octal constant 0 ≤ IPATRN ≤ 17778 defining the dash pattern for vectors.
Example


Dashed Line Patterns

111111100 Binary or 1774 Octal

111110000 Binary or 1760 Octal

1110000000 Binary or 1700 Octal

1100000000 Binary or 1400 Octal

1100010000 Binary or 1430 Octal

111011110 Binary or 1736 Octal

111110110 Binary or 1766 Octal

1The # symbols printed on this listing are punched as ' on the keypunch.
MXMY

Purpose
To return the integer coordinates of the last plotter instruction to the calling program. (This is useful in some kinds of titling.)

Arguments
MX,MY (integer)
Names of two core cells in the program in which the most recent CRT coordinates generated by any plot call will be placed.
The following routines are convenient for drawing graph paper, axes, and other backgrounds. They use the linear-log scaling combination if SET is called and linear-linear if SET is not called.

Arguments 1-4 of SET specify the area over which the backgrounds are to be drawn. GRIDALL, GRID, GRIDL, PERIM, PERIML, and HALFAX have arguments MGRX,MINRX,MGRY,MINRY which control the number of major and minor divisions in the graph paper or perimeter. Major division lines, or tick marks, are drawn in high intensity, and minor lines or ticks in low intensity. The number of divisions refers to the holes between lines rather than the lines themselves. This means that there is always one more major division line than the number of major divisions. Similarly there is one less minor division line than minor divisions (per major division). All background routines leave the dd80 in high intensity.
The meaning of the argument in the log case is completely different from the meaning in the linear case, as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Linear</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGRX X-axis</td>
<td>The number of major divisions along the X- or Y-axis. Divisions are the holes, not the lines.</td>
<td>The exponent of $10^\text{MGRX(Y)}$. For normal log paper MGRX(Y) is 1, meaning a major division type line for each log cycle.</td>
</tr>
<tr>
<td>MGRY Y-axis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINRX X-axis</td>
<td>The number of minor divisions per major division. Again, this is the number of holes, not lines.</td>
<td>If MINRX(Y) ≤ 10 and MGRX(Y) = 1, the eight normal subcycle divisions will be drawn. If MINRX(Y) ≥ 10 or MGRX(Y) = 1, these subdivisions will be suppressed.</td>
</tr>
<tr>
<td>MInRY Y-axis</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GRID

Purpose To draw graph paper.

Description This routine will draw graph lines in the portion of the CRT specified by a SET call, with the number of major and minor divisions specified by the arguments. Lines at major divisions are high intensity; minor ones are low intensity. On log-type axes, the eight subcycle divisions are not marked if MGRX (or MGRY) is other than 1.

Arguments MGRX, MINRX, MGRY, MINRY (integer)

For definitions see p. 4.30.

GRIDL

Purpose Same as GRID, but each major division is marked with its numerical value.

Arguments MGRX, MINRX, MGRY, MINRY

See GRID.
### PERIM

**Purpose**
Same as GRID but interior lines are replaced with tick marks along the edges.

**Arguments**
MGRX, MINRX, MGRY, MINRY

See GRID, p. 4.31.

### PERIML

**Purpose**
Same as PERIM, but each major division is marked with its numerical value.

**Arguments**
MGRX, MINRX, MGRY, MINRY

See GRID, p. 4.31.
HALFAX

Purpose
To produce orthogonal axes intersecting at X,Y with division tick marks according to MGRX, MINRX, MGY, and MINRY. These can be labeled if desired.

Arguments
MGRX, MINRX, MGY, MINRY, X, Y, IXLAB, IYLAB

MGRX, MINRX, MGY, MINRY (integer)
See GRID, p. 4.31.

X, Y (floating or integer)
The intersection point of the X and Y axes. If floating-point, the coordinate is scaled according to SET. If integer, it is assumed to be an absolute CRT coordinate.

IXLAB, IYLAB (integer)
Flags for line-value labels:

<table>
<thead>
<tr>
<th>IXLAB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No X labels</td>
</tr>
<tr>
<td>1</td>
<td>X-axis labels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IYLAB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Y labels</td>
</tr>
<tr>
<td>1</td>
<td>Y-axis labels</td>
</tr>
</tbody>
</table>

AXES

Purpose
To draw a set of perpendicular axes through point (X,Y) to the minimum and maximum points specified by the first four arguments of SET. If SET has not been called, the axes will be drawn to the edges of CRT.

Arguments
X, Y (floating)

Coordinates of axis intersection.
GRIDALL

Purpose
To provide a general entry point for all backgrounds with the option of line labeling on each axis.

Arguments
MGRX, MINRX, MGRY, MINRY, IXLAB, IYLAB, IHGP (integer)

MGRX, MINRX, MGRY, MINRY
See GRID, p. 4.31.

IXLAB, IYLAB
Flags for line value labels along axes:

<table>
<thead>
<tr>
<th>IXLAB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No X-axis labels</td>
</tr>
<tr>
<td>1</td>
<td>X-axis labels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TYLAB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Y-axis labels</td>
</tr>
<tr>
<td>1</td>
<td>Y-axis labels</td>
</tr>
</tbody>
</table>

IHGP
Flag for background type:

<table>
<thead>
<tr>
<th>IHGP</th>
<th>X-Axis Background</th>
<th>Y-Axis Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GRID</td>
<td>GRID</td>
</tr>
<tr>
<td>1</td>
<td>GRID</td>
<td>PERIM</td>
</tr>
<tr>
<td>2</td>
<td>GRID</td>
<td>HALFAX</td>
</tr>
<tr>
<td>4</td>
<td>PERIM</td>
<td>GRID</td>
</tr>
<tr>
<td>5</td>
<td>PERIM</td>
<td>PERIM</td>
</tr>
<tr>
<td>6</td>
<td>PERIM</td>
<td>HALFAX</td>
</tr>
<tr>
<td>8</td>
<td>HALFAX</td>
<td>GRID</td>
</tr>
<tr>
<td>9</td>
<td>HALFAX</td>
<td>PERIM</td>
</tr>
<tr>
<td>10</td>
<td>HALFAX</td>
<td>HALFAX</td>
</tr>
</tbody>
</table>

If IHGP = 10, the axis intersection will be taken at the lower left corner of the area specified by SET.
TICKS

Purpose
To allow program control of tick mark length in PERIM, PERIML, GRIDALL, and HALFAX. MAJOR is initially set to 12, and MINOR to 8.

Arguments
MAJOR, MINOR (integer)

CRT length of division tick marks. Positive arguments will produce inward-pointing ticks; negative arguments will produce outward-pointing ticks.

TICK4

Purpose
To allow program control of tick mark length in PERIM, PERIML, GRIDALL, and HALFAX. TICK4 allows separate control of X- and Y-axis tick marks. MGRX and MGRY are initially set to 12, and MINRX and MINRY to 8.

Arguments
MGRX, MINRX, MGRY, MINRY (integer)

MGRX, MGRY
Length in CRT units of major division tick marks on the X- or Y-axis.

MINRX, MINRY
Length in CRT units of minor division tick marks on the X- or Y-axis.
LABMOD

Purpose To enable greater flexibility in graph paper generation. LABMOD itself does no plotting; it only presets parameters for the graphing routines.

Rules
- LABMOD must be called *after* the SET call, as it needs the information derived by SET.

- It must be called *before* the background routines for which it is presetting parameters.

Arguments IFMTX, IFMTY, NUMX, NUMY, ISIZX, ISIZY, IXDEC, IYDEC, IXOR (integer)

IFMTX, IFMTY
A Hollerith format specification for X- or Y-axis numerical labels in GRIDL, PERIML, GRIDALL, or HALFAX. The specification must start with a left parenthesis and end with a right parenthesis, and should not use more than 10 characters. Only F and E conversions are permissible, such as IFMTX=6H(F8.2) and IFMTY=7H(E10.0).

NUMX, NUMY
The number of characters specified by IFMTX and IFMTY. For the above examples, these would be NUMX=8 and NUMY=10.
ISIZX, ISIZY

The character size codes for the X and Y labels. Codes are the same as for PWRT and PSYM.

<table>
<thead>
<tr>
<th>ISIZX or ISIZY</th>
<th>CRT Units/Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
</tbody>
</table>

IXDEC

The decrement in CRT units from the MINX position (specified by the first argument of SET) to the starting X-address of the label specified by IFMTY, NUMY, and ISIZY. For example, if the first argument of SET is .1, MINX is 102 (.1*1024). If IXDEC is 60, the label will start at 62 (102-60). To save ordinary condition calculations, the following conventions are used.

- If IXDEC = 0, it is set to properly position the Y-axis labels to the left of the Y-axis. The plot package calculates it by:
  \[
  \text{decrement} = (\text{NUMY} + 1) \times \text{character size Y}
  \]
  Thus if NUMY = 7 and ISIZY = 2 and IXDEC is set to zero, the X decrement becomes \((7+1) \times 16 = 120\).

- If IXDEC = 1, it is reset to produce labels to the right of the right Y-axis. The plot package calculates it as follows:
  \[
  \text{X decrement} = -(\text{MAXX} - \text{MINX} + \text{character size Y})
  \]
  Thus if the first two arguments of SET are .1 and .8, ISIZY = 3 and IXDEC = 1, the X decrement is
  \[-((.1*1024) - (.8*1024) + 24) = -741\].
LABMOD
(Continued)

TYDEC

The decrements in CRT units from the MINY position (specified by the third argument of SET) to the starting Y-address of the label specified by IFMTX, NUMX, and ISIZX. For example, if the third argument of SET is .2, MINY is \( .2 \times 1024 = 205 \). If TYDEC = 30, the label will start at \( 205 - 30 = 175 \). To save ordinary condition calculations, there are two conventions:

- If TYDEC = 0, the decrement is set to produce labels along the bottom. The calculations depend on whether the X-labels are in normal or 90° orientation (see IXOR). For TYDEC = 0, IXOR = 0, the decrement equals the character size; for example, if ISIZX = 2, the decrement equals 16. For TYDEC = 0, IXOR = 1:
  \[ \text{Y decrement} = (\text{NUMX} + \frac{1}{2}) \times \text{character size X} \]

- If TYDEC = 1 and IXOR = 0 or 1, the labels will go along the top of the graph. The PLOT package makes the following calculations:
  \[ \text{Y decrement} = -(\text{MAXY} - \text{MINY} + \text{character size X}). \]

IXOR

Orientation of the X-axis labels.

\[
\begin{align*}
\text{IXOR} & = 0 \quad \text{normal or } +X \\
& = 1 \quad 90° \text{ or } +Y
\end{align*}
\]

In normal orientation the actual number of non-blank digits are centered under the line or tick to which they apply.
Computer Sample 6
Program GRAPHS
Illustrating Use of Backgrounds

SUBROUTINE GRAPHS
CALL SET(0, 1, 0, 1, 0, 1, 0, 1, 1)
CALL GRID(1, 1, 1, 1)
CALL SET(5, 6, 5, 95, 1, 1000, 3, 1, 5, 3)
CALL GRID(1, 1, 5, 3)
CALL SET(1, 0, 0.7, 0.07, 0.5, 0.95, 0.0, 1.1, 1.1e100, 2)
CALL TICKS(-15, -7)
CALL PERIM(10, 10, 20, 1)
CALL SET(5, 6, 5, 95, 0, 1, -1e100, 1, 1, 1)
CALL LABMOD(6H(F5.2), 6H(E9.1), 5, 9, 1, 0, 0, 0, 0)
CALL TICKS(60, 15)
CALL PERIM(4, 2, 18, 1)
CALL SET(4.7, 4.9, 5.9, 0.1, 1000, 0, 1)
CALL LABELMOD(6H(F5.2), 6H(E9.1), 5, 9, 1, 1, 2, 1, 1, 1)
CALL GRID(15, 2, 10, 4)
CALL PART(270, 45, 0, PERIM, 5, 2, 0)
CALL PART(400, 30, 0, PERIM, 5, 3, 0)
CALL PART(270, 90, 0, GRID, 5, 8, 0)
CALL PART(400, 490, 0, 0, GRID, 4, 1, 9)
CALL FRAME
RETURN
END
Plotting Library Routines

Computer Sample 7
Program SAMP
Illustrating Backgrounds
Generated by GRIDALL

SUBROUTINE SAMP
DIMENSION L(3), IHGP(3,3)
DATA XA,YA,R,S/.1,.05,.25,.05/
DATA L/GRID,GRIDPERIM,HALFAX/
DATA IHGP/2,6,10,1,5,9,0,4,8/
IX=1
DO 1 I=1,3
IY=(XA+(I-1)*(R+S)+.5*R)*1024.+
1 CALL PWRT(IX, IY, L(4-I), 6, 2, 0)
IY=975
DO 2 I=1,3
IX=((XA+(I-1)*(R+S)+.5*R)*1024.5)-40.
2 CALL PWRT(IX, IY, L(I), 6, 2, 0)
DO 3 I=1,3
X1=X4+(I-1)*(R+S)
X2=X1+R
IX=512.* (X1*X2)/40.
3 CALL PWRT(IX, IY, L(I), 6, 2, 0)
Y1=Y4+(J-1)*(R+S)
Y2=Y1+R
IY=512.* (Y1*Y2)
CALL SET(X1, X2, Y1, Y2, 0., 1., 0., 1., 1)
CALL GRIDALL(5, 3, 6, 4, 0, 0, IHGP(I, J))
ENCCGE(7, 100, M)IHGP(I, J)
100 FORMAT (5HIHGP=, 12)
3 CALLPWRT(IX, IY, M, 7, 2, 0)
CALL FRAME
RETURN
END

GRID

PERIM

HALFAX
The FLASH routines (FLASH1, FLASH2, FLASH3, and FLASH4) permit plotting of repetitive data, such as complicated grids, on demand without recalculating the plotter instructions each time. This is particularly useful for movie applications. Up to 11 separate blocks of instructions can be created and regenerated at any time during the program. (See discussion of normal instruction handling under PLOT Buffer Control, p. 4.22.)

FLASH1

Purpose
To initiate storage of plotter instructions in the array IFIELD instead of sending them to the dd80.

Arguments
IFIELD, MAXLEN (integer)

IFIELD
The name of an array in the program that is to receive completed plotter instructions.

MAXLEN
The maximum number of words in IFIELD that may be used (normally the dimension of IFIELD).
FLASH2

Purpose
To terminate the process started by FLASH1. After a FLASH2 call, plot instructions again go out to the dd80 normally. All instructions between the FLASH1 and FLASH2 calls are stored in IFIELD.

Arguments
IFORM, LENGTH (integer)

IFORM
An integer between 0 and 10 designating which block of instructions the FLASH call has generated; i.e., a label which will be used by subsequent FLASH3 calls.

LENGTH
The actual number of words in IFIELD that have been filled. This is returned in LENGTH.

FLASH3 (IFORM)

Purpose
To send the complete set of instructions generated between a particular set of FLASH1 and FLASH2 calls to the dd80.

Argument
IFORM (integer)
An integer between 0 and 10 that was used in FLASH2 for the particular block of instructions to be plotted.
FLASH4

Purpose

To establish the necessary parameters which have not been generated by FLASH1 and FLASH2 in the program being run for use by FLASH3 as instructions.

Arguments

IFWA, LWA,IFORM (integer)

IFWA

The first word of a block of plotter instructions not generated by a FLASH1 and FLASH2 sequence in the program being run.

LWA

The last word of the block. The total length must be 1 plus a multiple of 3, because of the method of plot- ter instruction packing.

IFORM

An integer between 0 and 10 designating which block of instructions is to be used in FLASH3 calls.
Example of FLASH Routine Use

PROGRAM MOVIE
DIMENSION IA(10000),IB(10000), etc.
NN=0
CALL SET (appropriate arguments)
1 CALL FLASH1 (IA,10000)
CALL CONTIN (a fictitious continental outline program)
CALL FLASH2(1,NN)
WRITE(6,100)NN
100 FORMAT(1H0,'NUMBER OF WORDS USED FOR CONT OUTLINE = *15)
2 WRITE(7)(IA(K),K=1,NN) (To save continental outline instructions on tape for future runs)
DO 20 J=1,1000
CALL DATA(J) (a fictitious data generation routine)
CALL FLASH1(IB,10000)
CALL CONTOUR (a fictitious contouring routine)
CALL FLASH2(2,NN)
WRITE(6,101)NN
101 FORMAT(1H0,'NUMBER OF WORDS USED FOR CONTOUR = *15)
DO 10 I=1,3
CALL FLASH3(1) (continental outline)
CALL FLASH3(2) (contour)
CALL LABEL(J) (fictitious titling subroutine)
10 CALL FRAME (contains buffer flush)
20 CONTINUE
END
In this example, the routine would generate the plotter instructions once for the continental outline in IA. Then for each set of new data, it would generate the plotter instructions for the contour in IB. Three identical frames would be made: the continental outline which comes from IA, the contour which comes from IB, and the label and frame which advance directly. In this routine of 3000 frames, the continental outline is calculated only once, and 1000 instead of 3000 contour maps are generated. To eliminate recalculating all the plotter instructions for the continental outline on subsequent runs of the routine, statements 1 through 2 could be replaced with

```
1 READ(5,100)NN
100 FORMAT (I5)
   READ 7(IA(K),K=1,NN)
2 CALL FLASH4 (IA(1),IA(NN),1)
```

**LTPEN**

**Purpose**
To send messages to the GRID interactive terminal. This routine is ignored by the dd80. See the GRID manual for appropriate use.

**Argument**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000B</td>
<td>means enable light pen</td>
</tr>
<tr>
<td>2000B</td>
<td>means disable light pen</td>
</tr>
<tr>
<td>1-77B</td>
<td>identification code</td>
</tr>
</tbody>
</table>
INSTRUCTION

The dd80 receives a 36-bit instruction word from the computer in one of two formats: normal mode or tabular mode. The function to be performed is defined by a 3-bit operation code, and the location by 10-bit X- and Y-coordinates. The bit arrangement is shown below.

FORMAT

```
<table>
<thead>
<tr>
<th>Orientation</th>
<th>Italic</th>
<th>Point</th>
<th>Intensity</th>
<th>Operation code</th>
<th>Size</th>
<th>Case</th>
<th>X-coordinate</th>
<th>Y-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
NORMAL FORMAT

Operation codes 0-5 are defined in bits 27-29.

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Generates a symbol plot. The symbol code in bits 30-35 includes orientation, italics, point, and intensity.</td>
</tr>
<tr>
<td>1</td>
<td>Sets up a vector origin at X,Y; with bit 31, set (1) plots a point at X,Y. This operation code may also be used to preset orientation, italics, point, and intensity without doing any actual plotting.</td>
</tr>
<tr>
<td>2</td>
<td>Draws a vector from the previous X,Y position to the new one.</td>
</tr>
<tr>
<td>3</td>
<td>Initiates tabular mode for generating text. This code sets up the tabular mode options and starting X,Y coordinates. The text, in a different format, follows this initial tabular mode instruction.</td>
</tr>
<tr>
<td>4</td>
<td>Advances frame.</td>
</tr>
<tr>
<td>5</td>
<td>Produces a dashed-line pattern. The 10-bit dash pattern (instead of an X-coordinate) is in bits 12-21.</td>
</tr>
</tbody>
</table>

TABULAR FORMAT

The tabular word format is used for text following an operation code of 3. In this format 6-bit symbol codes are packed six to an instruction until a 768 symbol code signals the end of text. At this point the dd80 reverts to normal word mode for the next instruction words.
OTHER USEFUL ROUTINES

The following plotting routines are written in Fortran and are in the ULIB file.

CALCNT

Purpose
To contour a two-dimensional field.

Fortran Call
CALL CALCNT(AZ,M,N,FLO,HI,FINC,NSET,NHI,NDOT)

AZ (floating)
The array to be contoured

M (integer)
X dimension of AZ

N (integer)
Y dimension of AZ

FLO (floating)
Value of lowest contour line

HI (floating)
Value of highest contour line

FINC (floating)
Increment between successive contour line values:
1. If FLO and HI are both zero, they will be set to the minimum and maximum values of AZ, respectively.
2. IF FINC is 0, it will be set to (HI-FLO)/16.
3. If FINC is negative, it will be set to (HI-FLO)/(-FINC).
NSET (integer)
1. If 0, SET and PERIM will be called by CALCNT.
2. If positive, SET and PERIM will not be called by CALCNT.

NHI (integer)
1. If 0, highs and lows will be marked with an "H" or "L" as appropriate, and the value of the high and low will be written under the symbol.
2. If positive, three digits and sign of each AZ value will be written with the sign indicating the data point location.
3. If negative, neither of the above will be done.

NDOT (integer)
A 10-bit octal constant designating the desired DASHLN pattern of negative-valued contour lines on the map. (See DASHLN, p. 4.26.) If NDOT is set to 0, 1, or 17778, solid lines will be drawn.

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>CALCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP</td>
<td>#+10</td>
</tr>
<tr>
<td>CON</td>
<td>AZ</td>
</tr>
<tr>
<td>CON</td>
<td>M</td>
</tr>
<tr>
<td>CON</td>
<td>N</td>
</tr>
<tr>
<td>CON</td>
<td>FLO</td>
</tr>
<tr>
<td>CON</td>
<td>HI</td>
</tr>
<tr>
<td>CON</td>
<td>FINC</td>
</tr>
<tr>
<td>CON</td>
<td>NSET</td>
</tr>
<tr>
<td>CON</td>
<td>NHI</td>
</tr>
<tr>
<td>CON</td>
<td>NDOT</td>
</tr>
</tbody>
</table>
CALCNT
(Continued)

Error Message
None

Storage
3410, including a 1520, word labeled common block /CONT/ and subroutines SCAN, LINEAR, HILO, and PTVALU.

Timing
About 2 sec for an average smoothness field dimensioned 40 x 70.

Method
Linear interpolation between cells whose AZ value locations are specified by subscripts; i.e., AZ(1,1) is at the lower left corner of map; AZ(M,N) is at the upper right corner. Scaling, if done by CALCNT (NSET = 0) makes each cell a square.
IDIOT

Purpose

To make a single subroutine call produce a complete, annotated X-Y plot.

Fortran Call

CALL IDIOT (FLDX,FLDY,N,LTYPE,IPAT,LABX,LABY,LTIT,LFRAME)

FLDX (floating)
The array of X values to be plotted.

FLDY (floating)
The array of Y values to be plotted.

N (integer)
The number of data points to be plotted (normally the dimension of FLDX and FLDY). Setting N to -N causes scaling to be skipped. This option may be exercised only after IDIOT has been called at least once with N positive.

LTYPE (integer)
Designation of log-linear combination of plotting.
The code is:

<table>
<thead>
<tr>
<th>LTYPE</th>
<th>X-Plotting</th>
<th>Y-Plotting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>linear</td>
<td>linear</td>
</tr>
<tr>
<td>2</td>
<td>linear</td>
<td>log</td>
</tr>
<tr>
<td>3</td>
<td>log</td>
<td>linear</td>
</tr>
<tr>
<td>4</td>
<td>log</td>
<td>log</td>
</tr>
</tbody>
</table>

(See p. 4.8 for more complete discussion.)

IPAT (integer)

A 10-bit octal constant defining the line pattern of the curve to be drawn. If IPAT = 1777B, the line will be solid. A typical dashed line is 1740B. (For further discussion see p. 4.26.)

LABX (integer)

A 4-word array of Hollerith information for labeling the abscissa.
IDIOT
(Continued)

Fortran Call (continued)

LABY (integer)
A 4-word array of Hollerith information for labeling the ordinate.

LTIT (integer)
A 4-word array of Hollerith information for a main title.

Note: The easiest way to generate LABX, LABY, or LTIT is to read one array per card with a 4A10 format. If the title is centered, it will be centered in the plot field.

LFRAME (integer)
A flag for frame advance. If LFRAME is
zero, no frame advance will be given.
negative, a frame advance will be called before plotting is started.
positive, a frame advance will be given after plotting is completed.

Ascent Calling Sequence

RJ  IDIOT
JP  *10
CON  FLDX
CON  FLDY
CON  N
CON  LTYPE
CON  IPAT
CON  LABX
CON  LABY
CON  LTIT
CON  LFRAME
Error Message  None

Storage  10008

Method  IDIOT searches the field, rounds the minimum and maximum field values appropriately, draws graph paper with a suitable number of lines, and plots and labels a curve on the graph.

Additional curves may be made on the same graph by the following sequence:

CALL DASHLN(IPATRN) to make an identifiable line
CALL CURVE(FLDX,FLDY,N) to plot the curve

IDIOT uses the routines LINRD and LGRD (in the package).
SUPMAP

Purpose

To draw maps in a variety of projections with an option of world coast lines and USA political boundaries.

Fortran Call

CALL SUPMAP(JPROJ, POLAT, POLONG, ROT, PL1, PL2, PL3, PL4, JLTS, JGRID)

Entry Points

SUPMAP, SUPCON, MAPLOT

Program Size

There are three subroutines:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPMAP</td>
<td>01455</td>
</tr>
<tr>
<td>MAPLOT</td>
<td>02750</td>
</tr>
<tr>
<td>SUPCON</td>
<td>0461</td>
</tr>
</tbody>
</table>
A number of utility subroutines are available to the user from a disk library called ULIB. These subroutines represent many of the most commonly used algorithms in numerical analysis. Each subroutine or group of subroutines is in the library under a unique file name. The file consists of the source cards of the subroutine that have been put in Cosy form. The user gains access to these files by using the control card options available on the *FORTRAN and *ASCENT cards. For example:

*FORTRAN,S=ULIB,N=NOLLS
*COSY

These cards will cause a search of the ULIB library for the file called NOLLS and put the contents of the file into the input file immediately after the *COSY card. The subroutines may then be used, edited, or punched in the same way any other subroutine in the input file is manipulated.

The ULIB library is a read-only library. The user may not write or edit any file in the library. If a user wishes to add a routine to the library or requires a program not available in the library, the library group would be pleased to work on the development of such a program. Detailed descriptions of all routines are available from the librarian.
Each routine has been classified into one of the following fourteen groups:

1. Algebraic zeros
2. Approximation and curve fitting
3. Elementary functions
4. Fast Fourier transforms
5. Graphic display
6. Input/output
7. Matrix analysis
8. Miscellaneous
9. Nonalgebraic zeros
10. Ordinary differential equations
11. Partial differential equations
12. Quadrature
13. Random number generators
14. Statistics

A listing and brief description of the available routines follow. The access control cards for ULIB are given for each routine.
ALGEBRAIC ZEROS  

**BNDZRO**  
Finds individual bounds for a set of approximate zeros. Access cards for BNDZRO:

*FORTRAN,S=ULIB,N=BNDZRO  
*COSY

**CPOLY**  
Traub's three-stage variable-shift iteration for zeros of complex polynomials. Access cards for CPOLY:

*FORTRAN,S=ULIB,N=CPOLY  
*COSY

**CUBIC**  
Algebraic solution of a third-order polynomial with real or complex coefficients. Access cards for CUBIC with real coefficients:

*FORTRAN,S=ULIB,N=RCUBIC  
*COSY

Access cards for CUBIC with complex coefficients:

*FORTRAN,S=ULIB,N=CCUBIC  
*COSY

**PROOT**  
Newton-Bairstow technique for a polynomial with real coefficients (NYU). Access cards for PROOT:

*FORTRAN,S=ULIB,N=PROOT  
*COSY

**QUARTC**  
Algebraic solution of a fourth-order polynomial with complex coefficients. Access cards for QUARTC:

*FORTRAN,S=ULIB,N=QUARTC  
*COSY
ALGEBRAIC ZEROS

(Continued)

QUARTD

Algebraic solution of a fourth-order polynomial with real coefficients in double precision.
Access cards for QUARTD:

*FORTRAN,S=ULIB,N=QUARTD
*COSY

RPOLY

Traub's three-stage variable-shift iteration for zeros of real polynomials. Access cards for RPOLY:

*FORTRAN,S=ULIB,N=RPOLY
*COSY

TRAUB

Traub's globally convergent method for a polynomial with complex coefficients. Access cards for TRAUB:

*FORTRAN,S=ULIB,N=TRAUB
*COSY
Approximation and Curve Fitting

Least Squares

LINLSQ Computes $a_i$'s so that the data are approximated in the least squares sense by a function of the form $y = a_1 f_1(x) + a_2 f_2(x) + ... + a_n f_n(x)$ where the $f_i(x)$ are specified by the user. Access cards for LINLSQ:

*FORTRAN, S=ULIB, N=LINLSQ
*COSY

LSTSQR Computes the coefficients of a least squares polynomial approximation to the data. Access cards for LSTSQR:

*FORTRAN, S=ULIB, N=LSTSQR
*COSY

MPAKGE Nonlinear multivariate least squares package. MPAKGE computes $a_1, a_2, ..., a_m$ so that data $y_1, y_2, ..., y_n$ are approximated in the least squares sense by a function of the form $y = f(x_1, x_2, ..., x_n, a_1, a_2, ..., a_m)$. Access cards for MPAKGE:

*FORTRAN, S=ULIB, N=MPAKGE
*COSY

NOLLS Nonlinear least squares package. NOLLS computes the parameters $a_1$ so that the data $y_i$ are approximated in the least squares sense by a function of the form $y = f(x, a_1, a_2, ..., a_n)$. (Note that the difference between NOLLS and LINLSQ is that the coefficients $a_i$ can occur nonlinearly in NOLLS.) Access cards for NOLLS:

*FORTRAN, S=ULIB, N=NOLLS
*COSY
**Rational Approximation**

**RATION**

Produce a rational function (i.e., a function of the form \( \frac{P(x)}{Q(x)} \) where \( P \) and \( Q \) are polynomials) which approximates an explicitly given function in the sense that the maximum relative error is minimized. Access cards for RATION:

\*FORTRAN,S=ULIB,N=RATION
\*COSY

**Spline Fitting**

**PSPLIN**

Periodic spline approximation of a function. Access cards for PSPLIN:

\*FORTRAN,S=ULIB,N=PSPLIN
\*COSY

**SINGRT**

SPLINE integration of function over any interval (rectangle) in its domain. Routine uses coefficients produced by XYZSPLN. Access cards for SINGRT:

\*FORTRAN,S=ULIB,N=SINGRT
\*COSY

**SINTRP**

Spline interpolation to approximate function, first or second derivatives at a point. Routine uses coefficients computed in XYZSPLN. Access cards for SINTRP:

\*FORTRAN,S=ULIB,N=SINTRP
\*COSY
SPLINE Cubic spline approximation of a function. Access
cards for SPLINE:

*FORTRAN,S=ULIB,N=SPLINE2
*COSY

SPLINE PACKAGE A one- or two-dimensional spline fitting approx-
imation of a function with options for different
boundary conditions. In the one-dimensional case,
a sequence of nodes may be set and a least squares
approximation of the function by spline curves
occurs. This routine may be used to interpolate
the function and its first and second derivatives.
Access cards for SPLINE PACKAGE:

*FORTRAN,S=ULIB,N=SPLPACA
*COSY
*ASCENT,L,S=ULIB,N=SPLPACB
*COSY Use both

SPLINT One-dimensional interpolation using a spline
under tension. Access cards for SPLINT:

*FORTRAN,S=ULIB,N=SPLINT
*COSY

SPLVN Least squares cubic spline approximation with
variable knots. Access cards for SPLVN:

*FORTRAN,S=ULIB,N=SPLVN
*COSY

XYZSPLN Calculates spline coefficients for numerical
interpolation, differentiation, or integration,
or any combination of these. The function inter-
polated may have one or two independent variables.
Access cards for XYZSPLN:

*FORTRAN,S=ULIB,N=XYZSPLN
*COSY
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBTRP</td>
<td>Simple cubic interpolation of a function given at four points. Access card for CUBTRP:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=CUBTRP</td>
</tr>
<tr>
<td></td>
<td>*COSY</td>
</tr>
<tr>
<td>DARIV</td>
<td>Approximates the derivative of a function given by an array. Access card for DARIV:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=DARIV</td>
</tr>
<tr>
<td></td>
<td>*COSY</td>
</tr>
<tr>
<td>DERIV</td>
<td>Approximates the derivative of a function given analytically. Access card for DERIV:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=DERIV</td>
</tr>
<tr>
<td></td>
<td>*COSY</td>
</tr>
<tr>
<td>DUP</td>
<td>Simple two-dimensional cubic interpolation. Access cards for DUP:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=DUP</td>
</tr>
<tr>
<td></td>
<td>*COSY</td>
</tr>
<tr>
<td>SEARCH</td>
<td>Linear interpolation in a two-dimensional array. Access cards for SEARCH:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=SEARCH</td>
</tr>
<tr>
<td></td>
<td>*COSY</td>
</tr>
</tbody>
</table>
**ELEMENTARY FUNCTIONS**

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
</table>
| BESLI      | Modified Bessel function of first kind, $I_n(x)$. | *FORTRAN, S=ULIB, N=BESLIK
*COSY |
| BESLJ      | Bessel function of first kind, $J_n(x)$. | *FORTRAN, S=ULIB, N=BESLJY
*COSY |
| BESLK      | Modified Bessel function of first kind, $K_n(x)$. | *FORTRAN, S=ULIB, N=BESLIK
*COSY |
| BESLY      | Bessel function of first kind, $Y_n(x)$. | *FORTRAN, S=ULIB, N=BESLJY
*COSY |
| BSJ        | Spherical Bessel functions. | *FORTRAN, S=ULIB, N=BSJ
*COSY |
| COMBES     | Bessel functions for complex argument and complex order. | *FORTRAN, S=ULIB, N=COMBES
*COSY |
### ELEMENTARY FUNCTIONS

(Continued)

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
</table>
| **CX** | Computes $z^x$ where $z$ is complex and $x$ is real; $z^x = e^x(\log|z| + i \arg Z)$ where $0 \leq \arg Z < 2\pi$. | *FORTRAN, S=ULIB, N=CX
*COSY |
| **CXERFC** | Complex error function (10-digit accuracy). | *FORTRAN, S=ULIB, N=CXERFC
*COSY |
| **EI** | Exponential integral (15-digit accuracy). | *FORTRAN, S=ULIB, N=EI
*COSY |
| **ELIP** | Complete elliptic integrals. | *FORTRAN, S=ULIB, N=ELIP
*COSY |
| **ERF** | Error function. | *FORTRAN, S=ULIB, N=ERF
*COSY |
| | Access cards for fast ERF with 8-digit accuracy: | *FORTRAN, S=ULIB, N=ERFFAST
*COSY |
EXPON  Fast exponential (6-digit accuracy). Access cards for EXPON:

*ASCENT,L,S=ULIB,N=EXPON
*COSY

GAMLN  Computes natural log of gamma function. Access cards for GAMLN:

*FORTRAN,S=ULIB,N=GAMLN
*COSY

INCGAM Incomplete gamma function. Access cards for INCGAM:

*FORTRAN,S=ULIB,N=INCGAM
*COSY

SCHMDT Computes Schmidt functions and the first derivatives for arbitrary degree n and order m ≤ n, or both. Access cards for SCHMDT:

*FORTRAN,S=ULIB,N=SCHMDT
*COSY
**FAST FOURIER TRANSFORMS**

### One-Dimensional

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
</table>
| **FFA**    | Forward transform, where the number of real data points is a power of two. Access cards for FFA: | *FORTRAN, S=ULIB, N=FFA  
*COSY |
| **FFS**    | Reverse transform (to be used in conjunction with FFA). Access cards for FFS: | *FORTRAN, S=ULIB, N=FFS  
*COSY |
|            | Access cards when using both FFA and FFS: | *FORTRAN, S=ULIB, N=FFAFFS  
*COSY |

### Multi-Dimensional

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
</table>
| **FOR2D**  | FOUR2 with data kept on direct access storage. Access cards for FOR2D: | *FORTRAN, S=ULIB, N=FOR2D  
*COSY |
| **FOUR2**  | Forward or reverse transform where the number of complex data points in each dimension is a power of two. Access cards for FOUR2: | *FORTRAN, S=ULIB, N=FOUR2  
*COSY |
| **FOURT**  | Forward or reverse transform with no restriction on the number of complex data points in any dimension. Access cards for FOURT: | *FORTRAN, S=ULIB, N=FOURT  
*COSY |
GRAPHIC DISPLAY

Contouring

CALCNT Contour plots of dd80. Access cards for CALCNT:

*FORTRAN,S=ULIB,N=CALCNT
*COSY

Access cards for quick CALCNT:

*FORTRAN,S=ULIB,N=CALCQK
*COSY

MAP Simple contouring routine with no labels. Access cards for MAP:

*FORTRAN,S=ULIB,N=MAP
*COSY

Lettering

PWRX A subroutine to letter and title on microfilm. This subroutine has an extensive set of symbols and greater flexibility than the system subroutine PWRT. Access cards for PWRX:

*FORTRAN,S=ULIB,N=PWRX
*COSY
GRAPHIC DISPLAY

(Continued)

Plotting

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIST</td>
<td>Plots a histogram on dd80. Access cards for HIST:</td>
<td>*FORTRAN, S=ULIB, N=HIST *COSY</td>
</tr>
<tr>
<td>IDIOT</td>
<td>Curve plotting on dd80. Access cards for IDIOT:</td>
<td>*FORTRAN, S=ULIB, N=IDIOT *COSY</td>
</tr>
<tr>
<td>MAP80</td>
<td>Plots one or more curves in one call on the dd80. Access cards for MAP80:</td>
<td>*FORTRAN, S=ULIB, N=MAP80 *COSY</td>
</tr>
<tr>
<td>SOLIDS</td>
<td>Draws a perspective picture of a function of two variables defined by an array. Access cards for SOLIDS:</td>
<td>*FORTRAN, S=ULIB, N=SOLIDS *COSY</td>
</tr>
<tr>
<td>STRMLN</td>
<td>To display the magnitude and flow direction of a two-dimensional vector field. Primarily used for wind field display. Access cards for STRMLN:</td>
<td>*FORTRAN, S=ULIB, N=STRMLN *COSY</td>
</tr>
<tr>
<td>SUPMAP</td>
<td>Plots the continental outline according to a specified projection. Access cards for SUPMAP:</td>
<td>*FORTRAN, S=ULIB, N=SUPMAP *COSY</td>
</tr>
</tbody>
</table>
INPUT/OUTPUT

REcin
Buffered handling of a large number of small records on tape, drum, or disk. Access cards for REcin and RECOT:

*FORTRAN,S=ULIB,N=REcin  
*COSY

RECOT

*COSY

TAPECY
Copies, combines, and edits tapes (see p. 6.1 for detailed description). Access cards for TAPECY:

*FORTRAN,S=ULIB,N=TAPECY
*COSY
### MATRIX ANALYSIS

**Eigenvalues and Eigenvectors of a Matrix**

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPXQR</td>
<td>Eigenvalues and eigenvectors of a complex matrix. Access cards for CMPXQR:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=CMPXQR *COSY</td>
</tr>
<tr>
<td>HEVAL</td>
<td>Eigenvalues of Hermitian matrix. Access cards for HEVAL:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=HEVAL *COSY</td>
</tr>
<tr>
<td>HOUSESR</td>
<td>Eigenvalues and eigenvectors of a real symmetric matrix. Access cards for HOUSESR:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=HOUSESR *COSY</td>
</tr>
<tr>
<td>REALQR</td>
<td>Eigenvalues and eigenvectors of a real matrix. Access cards for REALQR:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=REALQR *COSY</td>
</tr>
<tr>
<td>RSYMQR</td>
<td>Eigenvalues and eigenvectors of a real symmetric matrix. Access cards for RSYMQR:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=RSYMQR *COSY</td>
</tr>
<tr>
<td>STDQR</td>
<td>Eigenvalues of a real symmetric tridiagonal matrix. Access cards for STDQR:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN,S=ULIB,N=STDQR *COSY</td>
</tr>
</tbody>
</table>
ULIB--A Library of
Special Purpose Subroutines

Matrix Inversion

**INVMTX**  Inverse of a matrix (Gauss elimination with full pivoting). Access cards for INVMTX:

*FORTRAN, S=ULIB, N=INVMTX
*COSY

Orthogonalization

**ORTHO**  Normalizes a system of independent vectors. Access cards for ORTHO:

*FORTRAN, S=ULIB, N=ORTHO
*COSY

Determinants

**DET**  Calculates determinant of a matrix. Access cards for DET:

*FORTRAN, S=ULIB, N=DET
*COSY

Reduction to Hessenberg Form (i.e., $a_{ij} = 0$ for $i < j+1$)

**HESREL**  Householder reduction of a real matrix to upper Hessenberg form. Access cards for HESREL:

*FORTRAN, S=ULIB, N=HESREL
*COSY

**HESRSM**  Householder reduction of a real symmetric matrix to tridiagonal form. Access cards for HESRSM:

*FORTRAN, S=ULIB, N=HESRSM
*COSY
MATRIX ANALYSIS

Solution of a System of Equations

(Continued)

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAND3</td>
<td>Computes the solution of a system of linear equations whose matrix is tridiagonal using Gaussian elimination with partial pivoting. Access cards for BAND3:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN, S=ULIB, N=BAND3 *COSY</td>
</tr>
<tr>
<td>BNDLVL</td>
<td>Solves the matrix equation $Ax = B$ where $A$ is a banded $n \times n$ real matrix ($A$ banded means that there exists an integer $M &gt; 0$ such that $a_{ij} = 0$ whenever $</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN, S=ULIB, N=BNDLVL *COSY</td>
</tr>
<tr>
<td>EQSLV</td>
<td>Solves the matrix equation $Ax = B$ where $A$ is a real $n \times n$ matrix. Access cards for EQSLV:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN, S=ULIB, N=EQSLV *COSY</td>
</tr>
<tr>
<td>TRIDI</td>
<td>Solves the matrix equation $Ax = B$ where $A$ is a real tridiagonal matrix. Access cards for TRIDI:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN, S=ULIB, N=TRIDI *COSY</td>
</tr>
<tr>
<td>TRIDIP</td>
<td>Solves the matrix equation $Ax = B$ where $A$ is a real tridiagonal matrix except for the first and last equations which describe the periodic behavior of the unknowns (can be used to solve differential equations with periodic boundary conditions). Access cards for TRIDIP:</td>
</tr>
<tr>
<td></td>
<td>*FORTRAN, S=ULIB, N=TRIDIP *COSY</td>
</tr>
</tbody>
</table>
Matrix Package

Access cards for matrix package:

\[
\begin{align*}
\text{FORTRAN}, &\text{ S=ULIB, N=MATXPACA} \\
\text{COSY} \\
\text{ASCENT}, &\text{ L, S=ULIB, N=MATXPACB} \\
\text{COSY} \\
\end{align*}
\]

\{ Use both \}

DECOMP Decomposes a matrix into upper and lower triangular parts.

IMPRUV Iterative improvement of the SOLVE solution.

SOLVE Solves the matrix equation \( Ax = B \) using the above decomposition of \( A \).
## MISCELLANEOUS

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSEARCH</td>
<td>Binary search of a floating-point table.</td>
<td>*ASCENT, L=S=ULIB, N=BSEARCH *COSY</td>
</tr>
<tr>
<td>CONV360</td>
<td>Program to convert 360 decks punched in EBCDIC to Hollerith code acceptable to the 6600.</td>
<td>*FORTRAN, S=ULIB, N=CONV360A *COSY *ASCENT, L=S=ULIB, N=CONV360B *COSY</td>
</tr>
<tr>
<td>DATE</td>
<td>Given the number of hours after 1920, program computes year, month, day, hour, and day of week.</td>
<td>*FORTRAN, S=ULIB, N=DATE *COSY</td>
</tr>
<tr>
<td>EDIT</td>
<td>A program to create and maintain a library of source decks on various storage media (tape, disk, or cards).</td>
<td>*FORTRAN, S=ULIB, N=EDITF *COSY *ASCENT, L=S=ULIB, N=EDITA *COSY</td>
</tr>
<tr>
<td>EOT</td>
<td>Logical function to test for end of tape when using BUFFERIN/BUFFEROUT.</td>
<td>*ASCENT, L=S=ULIB, N=EOT *COSY</td>
</tr>
</tbody>
</table>

HOURS  Given year, month, and day, program computes number of hours after 1920. Access cards for HOURS:

*FORTRAN, S=ULIB, N=HOURS
*COSY

LCKSUM  Very fast checksum routine. Access cards for LCKSUM:

*ASCENT, L, S=ULIB, N=LCKSUM
*COSY

LNRPRG  Minimizes or maximizes one of the variables in a system of linear equations by the simplex method. Access cards for LNRPRG:

*FORTRAN, S=ULIB, N=LNRPRG
*COSY

MOVE   Very fast routine to move arrays from one place in core to another. Access cards for MOVE:

*ASCENT, L, S=ULIB, N=MOVE
*COSY

NDATE  Given a date, NDATE determines the number of days from a given base date. Using a different entry point, NDATE determines the date from the number of days from a base date. Access cards for NDATE:

*FORTRAN, S=ULIB, N=NDATE
*COSY
MISCELLANEOUS
(Continued)

PERMUT Determines all permutations of a set of integers.
Access cards for PERMUT:

*FORTRAN,S=ULIB,N=PERMUT
*COSY

PSARITH Adds, subtracts, multiplies, divides, differentiates, or integrates a truncated power series.
Access cards for PSARITH:

*FORTRAN,S=ULIB,N=PSARITH
*COSY

RENO Renumbers Fortran statement numbers. Access cards for RENO:

*FORTRAN,S=ULIB,N=RENO
*COSY

RICH Richardson's extrapolation to the limit to improve convergence of many iterative techniques.
Access cards for RICH:

*FORTRAN,S=ULIB,N=RICH
*COSY
NONALGEBRAIC

ZEROS

Roots of Nonlinear Equations

LAGUER
Laguerre's method for the zeros of a complex function. Access cards for LAGUER:

*FORTRAN,S=ULIB,N=LAGUER
*COSY

MINSER
Determines a local minimum of a function of several variables without use of derivatives by an approximation to the conjugate gradient method. Access cards for MINSER:

*FORTRAN,S=ULIB,N=MINSER
*COSY

MULROOT
Muller's algorithm to compute the complex zeros of a complex function. Access cards for MULROOT:

*FORTRAN,S=ULIB,N=MULROOT
*COSY

NLEQS
Kiener's method for the solution of a nonlinear equation. Access cards for NLEQS:

*FORTRAN,S=ULIB,N=NLEQS
*COSY

RTSRH
Binary search for the real zeros of a function. Access cards for RTSRH:

*FORTRAN,S=ULIB,N=RTSRH
*COSY

SIPMIN
Determines a local minimum of a function of several variables without use of derivatives by a variation of the simplex method. Access cards for SIPMIN:

*FORTRAN,S=ULIB,N=SIPMIN
*COSY
ORDINARY DIFFERENTIAL EQUATIONS

Solution of a System of Ordinary Differential Equations

EZND A Runge-Kutta with error control. Access cards for EZND A:

```
*FORTRAN, S=ULIB, N=EZND A
*COSY
```

NDA Hamming's method. Access cards for NDA:

```
*FORTRAN, S=ULIB, N=NDA
*COSY
```

HPCG To solve a system of first-order ordinary general differential equations with given initial values. Hamming's modified predictor-corrector method is used. Access cards for HPCG:

```
*FORTRAN, S=ULIB, N=HPCG
*COSY
```

HPCL To solve a system of first-order ordinary linear differential equations with given initial values. Hamming's modified predictor-corrector method is used. Access cards for HPCL:

```
*FORTRAN, S=ULIB, N=HPCL
*COSY
```

RNGKUT Runge-Kutta with no error control. Access cards for RNGKUT:

```
*FORTRAN, S=ULIB, N=RNGKUT
*COSY
```

Eigenvalues of Ordinary Differential Equations

ODEIG Solves eigenvalue problem for a linear system of ordinary differential equations. Access cards for ODEIG:

```
*FORTRAN, S=ULIB, N=ODEIG
*COSY
```
### Partial Differential Equations

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>Solution of Poisson's equation on a rectangle (alternating direction-implicit technique).</td>
<td>*FORTRAN, S=ULIB, N=ADI *COSY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHARAC</td>
<td>Solves a quasilinear hyperbolic system of equations.</td>
<td>*FORTRAN, S=ULIB, N=CHARAC *COSY</td>
</tr>
<tr>
<td>XYPOIS</td>
<td>Buneman's technique to solve Poisson's equation on a rectangle.</td>
<td>*FORTRAN, S=ULIB, N=XYPOIS *COSY</td>
</tr>
</tbody>
</table>
QUADRATURE

One-Dimensional

ALGUER
Laguerre quadrature on the interval \((0, \infty)\). Access cards for ALGUER:

*FORTRAN, S=ULIB, N=ALGUER
*COSY

CONVOL
Evaluates the convolution of two tabulated functions. Access cards for CONVOL:

*FORTRAN, S=ULIB, N=CONVOL
*COSY

GAUS
Gaussian quadrature. Access cards for GAUS:

*FORTRAN, S=ULIB, N=GAUS
*COSY

ROMB
Romberg integration. Access cards for ROMB:

*FORTRAN, S=ULIB, N=ROMB
*COSY

SIMPNE
Simpson quadrature for non-equally spaced abscissa. Access cards for SIMPNE:

*FORTRAN, S=ULIB, N=SIMPNE
*COSY

SIMPSON
Simpson quadrature for either an even or an odd number of points. Access cards for SIMPSON:

*FORTRAN, S=ULIB, N=SIMPSON
*COSY

SPI
Spline integration of a function. Access cards for SPI:

*FORTRAN, S=ULIB, N=SPI
*COSY
Multi-Dimensional

HASEL2  Monte Carlo technique using quasi-random numbers. Access cards for HASEL2:
*FORTRAN,S=ULIB,N=HASEL2
*COSY

MROMB  Romberg multiple integration. Access cards for MROMB:
*FORTRAN,S=ULIB,N=MROMB
*COSY
ULIB--A Library of Special Purpose Subroutines

**RANDOM NUMBER GENERATORS**

**GAUSSR**
Randomly drawn Gaussian deviates $G$.
$-4.417 \leq G \leq 4.417$. Access cards for GAUSSR:

```
*FORTRAN,S=ULIB,N=GAUSSR
*COSY
```

**GETRAN**
Random numbers with normal distribution or with uniform distribution. Access cards for GETRAN:

```
*FORTRAN,S=ULIB,N=GETRAN
*COSY
```

**GPROB**
Random numbers with normal distribution. Access cards for GPROB:

```
*FORTRAN,S=ULIB,N=GPROB
*COSY
```

**INTRAN**
Random integer $N$. $0 \leq N \leq 9$. Integers read from tape. Prime source of digits is Douglas Aircraft location 'A' rerandomization of basic RAND random digit generator output. Access cards for INTRAN:

```
*FORTRAN,S=ULIB,N=INTRAN
*COSY
```

**RANF**
Random numbers with uniform distribution (NYU) with restart capabilities, Ascent. Access cards for RANF:

```
*ASCENT,L,S=ULIB,N=RANF
*COSY
```

**RANF2**
Random number $X$. $0 \leq X \leq 1$. $X$ formed by using 14 digits from INTRAN. Access cards for RANF2:

```
*FORTRAN,S=ULIB,N=INTRAN
*COSY
```

**RNDEV**
Random numbers with normal distribution. Access cards for RNDEV:

```
*FORTRAN,S=ULIB,N=RNDEV
*COSY
```
### STATISTICS

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Access Cards</th>
</tr>
</thead>
</table>
| ASA        | Computes autocorrelation coefficients and spectral densities for a single series. Access cards for ASA: | *FORTRAN, S=ULIB, N=ASA  
*COSY |
| BMD04M     | Discriminant analysis for two groups. Access cards for BMD04M: | *FORTRAN, S=ULIB, N=BMD04M  
*COSY |
| BMD06M     | Canonical analysis. Access cards for BMD06M: | *FORTRAN, S=ULIB, N=BMD06M  
*COSY |
| BMD02R     | Stepwise regression. Access cards for BMD02R: | *FORTRAN, S=ULIB, N=BMD02R  
*COSY |
| BMD05R     | Polynomial regression. Access cards for BMD05R: | *FORTRAN, S=ULIB, N=BMD05R  
*COSY |
| BMD02S     | Contingency table analysis. Access cards for BMD02S: | *FORTRAN, S=ULIB, N=BMD02S  
*COSY |
| BMD02V     | Analysis of variance for factorial design. Access cards for BMD02V: | *FORTRAN, S=ULIB, N=BMD02V  
*COSY |
STATISTICS
(Continued)

CORR Determines correlation between two series. Access cards for CORR:

*FORTRAN,S=ULIB,N=CORR
*COSY

FACT Factor analysis. Access cards for FACT:

*FORTRAN,S=ULIB,N=FACT
*COSY

HARMON Uses Goertzel's algorithm to determine the coefficients of a trigonometric expansion. Access cards for HARMON:

*FORTRAN,S=ULIB,N=HARMON
*COSY

KOLMO Tests the difference in absolute value between an empirical distribution and a theoretical distribution, using Kolmogorov-Smirnov's limiting distribution. Access cards for KOLMO:

*FORTRAN,S=ULIB,N=KOLMO
*COSY

NDTR Computes \( Y = P(x) = \text{PROB} (X \leq x) \), where \( X \) is a random variable, distributed normally with mean zero and variance one. Access cards for NDTR:

*FORTRAN,S=ULIB,N=NDTR
*COSY

POWERX Computes the power spectrum of two series and the coherence, phase, and gain. Access cards for POWERX:

*FORTRAN,S=ULIB,N=POWERX
*COSY
SMIRN Computes the values of Kolmogorov-Smirnov's limiting distribution for a given argument. Access cards for SMIRN:

*FORTRAN,S=ULIB,N=SMIRN
*COSY

SPAL Calculates spectral estimates for a one-dimensional stationary time series. Access cards for SPAL:

*FORTRAN,S=ULIB,N=SPAL
*COSY
This chapter discusses routines for processing and accessing data, including tape copying and editing routines. For help with data processing problems, including where to find the data, how to decode problem formats, tape information, volume compaction, etc., see a data analyst.¹

The option for editing and copying tapes is not available as a routine on the system tapes.

Purpose

To copy, combine, and edit tapes.

Description

The routine includes a main driver program, a tape copy subroutine, and two user subroutines which may be modified for specific copy jobs.

The main driver program contains a LTH parameter which must be set to at least the maximum record length. Dimension IBUF by two times LTH.

6.2
Library Routines that
Process and Access Data

USER

Purpose
To allow the user to examine each record to determine whether or not to copy it.

Note: If the error option on the control card has been set to 1., this subroutine is not called when there is a parity error.

Fortran Reference
SUBROUTINE USER(IBUF,IWORDS,NFILE,NREC,ISTATE,NN)

IBUF
Address of the first word of the current record.

IWORDS
Length of the current record.

NFILE
Number of the current file.

NREC
Number of records read in the current file.

ISTATE
Status of current read.
0 = good read
2 = parity error

NN
Flag set in USER to determine whether or not the record is to be copied.
0 = do not copy
1 = copy
USERF

Purpose
To return control to the user after each file in a multiple file copy.

Fortran Reference
SUBROUTINE USERF(NRSUM,NRECS,NFILE,NF)

NRSUM
Total number of records read under control of the current control card.

NRECS
Number of records in this file.

NFILE
Number of the file just completed.

NF
A flag set for action taken after this call.
0 = continue copy with next file and do not write an end file on the output tape
1 = same as 0, but write an end file on the output tape
2 = go to next control card and do not write an end file on the output tape
3 = same as 2 but write an end file on the output tape

Note: For simple copies of a given tape the USER routine consists simply of the card NN=1, and the USERF routine of the card NF=1.
DATA CONTROL CARDS

Main program control is accomplished by using control cards in the data section of the program. Each control card is processed sequentially and is printed before it is processed.

Format

The general card format is:

<table>
<thead>
<tr>
<th>Card Column</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Control word (left-justified)</td>
</tr>
<tr>
<td>6-10</td>
<td>NREC = number of records or files</td>
</tr>
<tr>
<td>21-22</td>
<td>UNIT1 = a logical tape unit (1-10)</td>
</tr>
<tr>
<td>27-28</td>
<td>UNIT2 = a logical tape unit (1-10)</td>
</tr>
<tr>
<td>38</td>
<td>Mode of tape on UNIT1</td>
</tr>
<tr>
<td>40</td>
<td>Mode of tape to be written on UNIT2</td>
</tr>
<tr>
<td>48</td>
<td>Type of tape on UNIT1</td>
</tr>
<tr>
<td>50</td>
<td>Type of tape to be written on UNIT2 (always zero)</td>
</tr>
<tr>
<td>57</td>
<td>Error option</td>
</tr>
</tbody>
</table>

1 = copy only good records and delete those with parity errors
2 = copy all records
MODE AND TYPE OPTIONS

Mode and type options are the same as those listed under IOPROC (see p. 3.50):

**MODE**
- 0 = even parity (BCD mode), no character conversion
- 1 = odd parity (binary mode), no character conversion
- 2 = even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record)

**NTYPE**
- 0 or 1 = read (or write) a record in the NCAR system format
- 2 = read a record from another computer

*Note:* Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error. Thus a type 6 is the same as a type 2, but with no reread on parity errors.
The valid control words and their functions are listed below.

**REM**
Remark card which can contain any remarks in columns 6-80

**REW**
Rewind the unit specified in the field UNIT1 on the data control card.

**WEOF**
End file on the unit specified in the field UNIT1 on the data control card.

**COPY**
Copy NREC records (or one file, whichever comes first) from UNIT1 to UNIT2 with the modes, types, and error option as specified

**CPFLE**
Copy NREC files as for COPY. The copy will continue until NREC files have been copied or the copy is terminated in the USERF routine

**SKIP**
Skip NREC records (or one file, whichever comes first) on UNIT1. The output parameters must be specified even though they are not used

**DONE**
The job is terminated
COPY OUTPUT

The copy output will indicate whether there are any parity errors, the number of records copied, and the range of lengths in the records copied.

Listing

Computer Sample 8 is output from a tape copy which, for reasons of illustration, is unusually complicated and contains a high number of errors. This output would result from use of the control cards described above with the user routines also described.

This procedure would copy two files from the tape assigned to logical unit 1, to the tape assigned to logical unit 8. Then it would copy 50 records, skip 25 records, and copy to the next end file. The print lines beginning with control words are images of the control cards while the other print lines indicate the results of the previous control card. Note that the first file had one redundancy (record with a parity error) at record number 57. The second file had a parity error at record number 16. The first file contained 138 records and the second contained 97. Since the copy 50 record had the error option set to 1 and encountered a parity error at record number 31, it was not copied. The copy 9999 record card was terminated when an end file was encountered after 37 records, each 250 words long. The trouble on a write message here indicates that an attempt to write a zero-length record failed because of the previous parity error on the read. Trouble in a write indicates number of record in, number of record out, length in, length out, and status of the write.
Computer Sample 8
Showing Use of
Control Cards with
USER Subroutines

REM COPY OUTPUT EXAMPLE
REM CARD COLUMNS
REM 10 22 28 40 50 57
REM 01
REM 8
CFILE 2 FLs FROM 1 TO 8 MODE 1.1 TYPE 0.0 ERR 2
PARITY ERROR IN RECORD 57.
TROUBLE ON A WRITE: 57 57 0 0 2
138 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
PARITY ERROR IN RECORD 16.
TROUBLE ON A WRITE: 16 16 0 0 2
97 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
COPY 50 REC FROM 1 TO 8 MODE 1.1 TYPE 0.0 ERR 1
PARITY ERROR IN RECORD 31.
50 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
SKIP 25 REC FROM 1 TO 8 MODE 1.1 TYPE 0.0 ERR 2
25 RECORDS READ WITH LENGTHS OF 250 TO 250 WORDS.
COPY 9999 REC FROM 1 TO 8 MODE 1.1 TYPE 0.0 ERR 2
37 RECORDS READ WITH LENGTHS OF 250 TO 250 WORDS.
WEOF 8
DONE
OTHER DATA HANDLING ROUTINES

Note: Further details about the routines listed here are available from a data analyst, as is information regarding additional similar programs.¹

CDS360
A routine to convert from IBM 360 punched card codes to NCAR-compatible punched code cards (on ULIB as CONV360).

CDTAPE AND CDTAPEB
Routines to put punched cards onto tape in RPTOUT format (see RPTIN/RPTOUT, p. 3.93). The input cards may be Hollerith or binary.

CHCONV
A character conversion routine to convert from one six-bit code to another six-bit code (such as from BCD to display code).

CORFOR
A routine to correct Fortran-written tapes and to copy records created with the Fortran WRITE(NUNIT) statement. The Fortran READ(NUNIT) routine terminates execution on a tape error, and the CORFOR program facilitates recovery or deletion of these bad records.

GBITS AND SBITS
Routines for manipulating bits with a procedure similar to format control.

GBYTES AND SBYTES
Routines for manipulating bits in single bytes or arrays of equal-length bytes (see Entry Point GBYTES, p. 3.41).

¹See footnote on p. 6.1.
OTHER DATA HANDLING ROUTINES (Continued)

RDTAPE AND WRTAPE

Tape I/O routines similar to Fortran BUFFERIN/BUFFEROUT. They provide facilities not available under the Fortran operations (see IOPROC, p. 3.50).

REPBIN

A routine to reproduce card decks regardless of format. This routine and others similar to it are available in the machine room.

RPTIN AND RPTOUT

Routines to handle unblocking and blocking of short variable-length records (see RPTIN/RPTOUT, p. 3.90).

SORTUU AND SORTBB

Tape sort routines which operate on unblocked or RPTOUT format tapes and sequence the records in a file. The part of the record used to sequence the records is completely flexible and is determined in a user-written subroutine.¹

TDUMP

A tape-to-printer routine for dumping tape records.

UBLOK AND UZBLOK

Routines to facilitate unblocking of fixed-length logical records from tapes.

¹See footnote on p. 6.1.
Central monitor calls pass all I/O requests from an executing central memory program to the peripheral monitor in PP0, the interrupt driver. Necessary action is also taken by executive programs resident in the CPU.

Error Messages

Appropriate error messages will be returned when an executing program is interrupted (see Chapter 8).

Monitor Communications Area

The first four cells of every executing program contain the area of communications with the central monitor. These cells are assigned as follows:

```
Cell   0   1   2   3
CON 0  EQ 1  CM CON 0  EQ 1
```

When an executing program has an I/O request, a RJ to CM implements this request. The pseudo operation

```
CM EQU 2
```

establishes the central monitor jump address.

In the above code sequence, the RJ to CM places the address +1 in location 2 (CM), indicating the return address in the executing program immediately ahead of the arguments for the CM call. Execution proceeds to location 3 which jumps to location 1. Location 1 contains a jump to itself with the program counter equal to 1 for that control point. Peripheral processor zero is the interrupt driver. When the P count is set to 1, an interrupt is sensed by PP0 and an exchange jump is executed.
Cell 0 contains an error code number in bits 0-11, which is used to identify the error message printed at the end of the program output. (Error messages are listed in Chapter 8 of this manual.) If required, auxiliary error information, such as a unit number or an additional error address, is contained in bits 12-29. Bits 30-47 contain the address where the error is detected, i.e., the P count. Bits 48-50 identify hardware and software interrupts (H/S code). If these three bits are zero, a software interrupt has occurred. If they are non-zero, a hardware interrupt has occurred.

<table>
<thead>
<tr>
<th>Bits 48-50</th>
<th>Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (001₂)</td>
<td>Memory bounds</td>
</tr>
<tr>
<td>2 (010₂)</td>
<td>Floating-point overflow</td>
</tr>
<tr>
<td>4 (100₂)</td>
<td>Indefinite operand</td>
</tr>
</tbody>
</table>

For hardware interrupts, bits 0-29 are zero.

For a software interrupt on the 6600, Cell 0 contains the error code in the low-order bits.
Example 1, Software Interrupt

CARD NUMBER LOCATION
00000 000000 PROGRAM TEST
00001 000000 *FORTRAN*
00002 000000 A=800.
00003 000000 Y=EXP(X)
00004 000000 STOP
00005 000000 END

LENGTH OF ROUTINE TEST 000011

VARIABLE ASSIGNMENTS
X = 000007 Y = 000006

SUBROUTINES CALLED
EXPF
EXIT

COMPILE TIME 25

PROGRAM SPACE IS 000075

ENTRY POINT LOCATION ROUTINE ORIGIN
TEST 000004 000004
GOERR 000015 000015
EXIT 000027 000027
STOP 000027 000027
EXP 000034 000034
EXPF 000034 000034

THE TOTAL PROGRAM AND BUFFER SPACE REQUIRED IS 001371

ARG IN EXP GST 741.
ERROR OCCURRED AT 000006 IN TEST
For a hardware interrupt on the 6600, the error code number for INDEFINITE OPERAND is 0000.

Example 2, Hardware Interrupt

Monitor Communications Area (continued)
For a software interrupt, cell 0 is the same in bits 0-47 as the 6600 cell 0.

Example 1, Software Interrupt

The error occurred in location 6. 1001 is the code for the message ARG IN EXP GT 741.
Monitor Communications Area (continued)

For a hardware interrupt, cell 0 contains the contents of the PSD register in bits 0-17.

Example 2, Hardware Interrupt

An INDEFINITE was encountered at location 53.

If cell 0 is destroyed by the user's program, an arbitrary error comment is produced but this is unrelated to the termination status.
Central Monitor Routines

Registers

An exchange jump is taken by PPO. All registers are saved.

Ascent Calling Sequence

All I/O calls from a central memory program are made by calling CM with a calling sequence in the following form:

- RJ CM: CM is always location 2
- JP RETURN: Return location
- EQ OP: Operation code (equivalence)
- EQ CNT: Argument count (equivalence)
- EQ UNIT: Unit address (address)
- EQ STATUS: Status address (address)
- EQ FWA: First word address
- EQ LWA: Last word address +1

Arguments for a CM Call

RETURN

The address to which control passes following the call to CM. This may simply be $+4$ to jump around the calling sequence and continue the program, if the calling sequence contains the fifth word--FWA, LWA+1.

OP

The operation code. (Example: REW EQU 4)

- RECALL EQU 400000B
- SFT EQU 1: Search file forward
- BSF EQU 2: Backspace file
- WEF EQU 3: Write end of file
- REW EQU 4: Rewind
- RWU EQU 5: Rewind unload
- FSP EQU 6: Forward space a record
- BSP EQU 7: Backspace a record
- RFC EQU 10B: Read a BCD record (even parity)
- RFB EQU 11B: Read a binary record (odd parity)
- WRC EQU 12B: Write a BCD record (even parity)
- WRB EQU 13B: Write a binary record (odd parity)
- STATUS EQU 14B: Status request from device
- FLUSH EQU 15B: Flush a buffer
- SBS EQU 16B: Skip a bad spot
- RFNT EQU 17B: Request file name table entries
Arguments for a CM Call (continued)

If 4000008 is added to any of the above operation codes, the CM routine will retain control until the operation is complete. An operation code of 4000008 with a call that includes the status address of a previous call will also result in an exchange jump to another central program unless the status word indicates the operation is already complete. Thus the 4000008 modifier serves as a WAIT operation code. If an operation code occurs with no wait, control returns to the calling program before the operation is complete.

A number of special requests may be made to the central monitor by using the following OP codes.

<table>
<thead>
<tr>
<th>OP Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCL</td>
<td>100001B</td>
<td>Read the wall clock</td>
</tr>
<tr>
<td>MCL</td>
<td>100002B</td>
<td>Read the millisecond clock</td>
</tr>
<tr>
<td>DATE</td>
<td>100004B</td>
<td>Read the date</td>
</tr>
<tr>
<td>EXIT</td>
<td>100007B</td>
<td>Normal exit</td>
</tr>
<tr>
<td>DUMQ</td>
<td>100015B</td>
<td>Dump and stop</td>
</tr>
</tbody>
</table>

The millisecond clock is returned in X1 following a MCL request. Following WCL or DATE, results are returned in X1 in display code, left-justified in the word.

An example, using the MACRO call, is:

```
IOC WCL,0,0,0
```

The count. (Example: CNT EQU 0)

For calls that do not request a first word address (FWA), last word +1 (LWA) pointer CNT is 0, and FWA,LWA word is omitted. Where data is to be transmitted by means of FWA,LWA pointers, CNT is 1.
UNIT
The logical unit number.
(Example: UNIT CON 6776B)

UNIT is the address of a core cell containing the logical unit number referenced by the call. Logical units 1-6760 are assigned with an *ASSIGN control card. Unless an *ASSIGN control card has been used to assign logical unit 5 or 6, these units will be assigned as the standard input unit (card reader) and the standard output unit (printer) respectively. This is done to allow their use in the Fortran statements without the presence of *ASSIGN cards.

READ INPUT TAPE 5, format, list
WRITE OUTPUT TAPE 6, format, list

Four other units which need not be assigned are

- 6776B printer
- 6777B card reader
- 6774B dd80
- 6775B card punch

STATUS
The status cell. (Example: STATUS CON 0)

STATUS is the address of a cell in which status bits for the operation are returned to the central program. The lower 18 bits contain the number of central processor words transmitted. The upper 24 bits are set for the following conditions:

<table>
<thead>
<tr>
<th>Status Flags</th>
<th>Actual Device Status</th>
<th>Device Code Number</th>
<th>Word Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>48</td>
<td>47</td>
<td>36 35 24 23 18 17 0</td>
</tr>
</tbody>
</table>
Arguments for a CM Call (continued)

<table>
<thead>
<tr>
<th>Bits 59-48</th>
<th>Status Replies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 59 4XXX</td>
<td>Operation unsuccessful</td>
</tr>
<tr>
<td>Bit 55 X2XX</td>
<td>Load point on tape</td>
</tr>
<tr>
<td>Bit 54 X1XX</td>
<td>End of tape</td>
</tr>
<tr>
<td>Bit 52 XX2X</td>
<td>Unrecoverable read error</td>
</tr>
<tr>
<td>Bit 49 XXX2</td>
<td>EOF</td>
</tr>
<tr>
<td>Bit 48 XXX1</td>
<td>Program termination</td>
</tr>
</tbody>
</table>

If bit 59 is set then bits 51-56 have an error code number, which means the following:

a. Unit is a tape 2 Logical unit no. not in SYSIOU table
   3
   ...
   6

b. Unit is disk/drum 25
   ...

A calling error of the forms FWA > LWA, FWA > FL, LWA > FL, sensed in bit 48 will terminate the program and a diagnostic will be printed.
Central Monitor Routines

Bits 47-36

<table>
<thead>
<tr>
<th>Bit</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>4XXX</td>
</tr>
<tr>
<td>46</td>
<td>2XXX</td>
</tr>
<tr>
<td>45</td>
<td>1XXX</td>
</tr>
<tr>
<td>44</td>
<td>X4XX</td>
</tr>
<tr>
<td>43</td>
<td>X2XX</td>
</tr>
<tr>
<td>42</td>
<td>X1XX</td>
</tr>
<tr>
<td>41</td>
<td>XX4X</td>
</tr>
<tr>
<td>40</td>
<td>XX2X</td>
</tr>
<tr>
<td>39</td>
<td>XX1X</td>
</tr>
<tr>
<td>38</td>
<td>XXX4</td>
</tr>
<tr>
<td>37</td>
<td>XXX2</td>
</tr>
<tr>
<td>36</td>
<td>XXX1</td>
</tr>
</tbody>
</table>

If STATUS = 0, no status will be returned by the peripheral processor. If a STATUS cell is specified, it is cleared before CM passes the request.

The device code numbers which guarantee that the status is returned non-zero, are

1 for disk or drum
4 for tape

FWA

The first word address of the data to be read or written

LWA

The last word address +1 of the data to be read or written
An I/O calling sequence might be defined in a MACRO as follows:

MACRO IOC, OP, CNT, UNIT, STATUS, FWA, LWA
BSS 0
RJ CM First word
JP *+3+CNT Second word
EQ OP Third word
EQ CNT
EQ UNIT Fourth word
EQ STATUS
IFN CNT,2,LST Pseudo operation to establish number of arguments
EQ FWA Fifth word
EQ LWA
ENDM

The CM routine blocks and unblocks all information for the standard input and output files on the disk. The input file may be read in either BCD or binary mode and the output file may be written in either mode. The only legal OP codes on these units are RFC or RFB for the input file and WRC or WRB for the output file.

Calls referencing units 5 and 6777B are reading the input unit, and a unit record or card image is unpacked from the input buffer and placed in the area starting at FWA. Only the number of words requested from the card are transmitted to the program. The buffer pointers are then moved to the start of the next card.
Data from calls referencing units 6, 6776B, 6774B, and 6775B are flagged with the unit name and placed in the output buffer. The buffer with all data types is written on logical unit 6776B. If the dd80 has been assigned for use during the execution phase, data from calls referencing unit 6774B are sent directly to the dd80. If cards are to be punched, they must be on unit 6775B, and the user must be careful to transmit exactly 16 words per card. If the call is not a multiple of 16 words, the cards will be incorrectly punched.

CM handles the input and output files in a wait mode. The data will always be moved before control is returned to the main program. If an EOF is sensed on the input medium, the status cell will be set with an EOF flag. Otherwise, CM does not set anything in the status cell for calls referencing these units.

Rule

- After a calling sequence to CM, the status word should not be modified until after an I/O operation is complete.
Example 1

The following code will read one card image from a BCD tape written on another computer and mounted on logical unit 3:

```
CM EQU 2
RFC EQU 10B
;
BSS 0
RJ CM Return jump to CM
JP **+4 Jump around calling sequence
EQ RFC+400000B Read BCD and wait
EQ 1 FWA and LWA included
EQ UNIT Address of cell containing logical unit number
EQ STAT Status cell
EQ CARD FWA
EQ CARD+8 LWA (last word +1)
;
UNIT CON 3 Logical unit 3 must be assigned
STAT CON 0 Storage for status word
CARD BSS 8 Storage for card image
```

Using the MACRO call, the code would be:

```
CM EQU 2
RFC EQU 400010B
IOC RFC,1,UNIT,STATUS,CARDA,CARDB
UNIT CON 3
STAT CON 0
CARD BSS 8
```
Example 2

The following subroutine will write a comment on the printer:

```
ENTRY COMNT
CM EQU 2
WRC EQU 12B
COMNT CON 0 Entry/exit
+ RJ CM Return jump to CM
JP COMNT Exit
EQ WRC+400000B Write BCD and wait
EQ 1 FWA,LWA included
EQ OUT
EQ STAT
EQ MSG1 FWA
EQ MSG2 LWA (last word +1)
OUT CON 6776B
STAT CON 0
MSG1 DPC *THIS IS A COMMENT*
MSG2 EQU *+1
END
```

The MACRO call would be:

```
IOC,400012B,1,OUT,STAT,MSG1,MSG2
```

where the variables are defined as above.
If a program terminates during execution, the monitor prints a list of diagnostics. The routine in which the fault occurred is printed first. A brief error message is printed second, and the octal location in the program from which the subroutine was entered is printed last.

Example

IBAIEX
0 TO 0TH POWER OR RESULT EXCEEDS 2 TO THE 48TH
ERROR AT LOCATION 000165

In the example, the routine which had the fault during execution of the job was IBAIEX. In the Fortran designation, I**J, the mnemonics of the name suggests integer base to an integer exponent. The error message tells the user that the answer is larger than $2^{48}$. The location in core from which this routine was called is 1658. If 4, the routine origin in the loader map, is subtracted from 1658, the result, which is 1618, indicates that the jump to IBAIEX is 1618 locations into the program which started at 4. The location field in a Fortran program printout is the second column from the left. It is a "relative address" since these locations start at 0 for each routine and are relative to the routine origin found in the loader map.
An alphabetical list of the subroutines and their associated diagnostics begins on p. 8.5. All of the messages are printed as a result of a RJ to Q8QERR. A typical exit to Q8QERR is in IBAIEX, as shown in the following example.

Example

```
EXT Q8QERR
SB3 ER
SB2 IBAIEX
SB1 B0
RJ Q8QERR
ER EQU 1036B
```

The code on an error exit in IBAIEX is: B3 contains the error code number and B2 indicates the entry point which contains the absolute address from which the routine was called.

- B1=0, if no additional error condition exists.
- B1≠0, if additional error conditions are to be printed from B1.
When a *TRAP card is used and an execution diagnostic is sensed, the computer will trace the error starting in the routine at fault. The trace is limited to 50 calls, and does not trace system exit mode (memory bounds, overflow, and indefinite operand). Computer Sample 9 on p. 8.4 shows a trace following a trace dump.

```plaintext
PROGRAM MAIN
A=C
CALL SUB1
CALL EXIT
END

SUBROUTINE SUB1
B=T
CALL SUB5
END

SUBROUTINE SUB2
B=T
CALL SUB7
END

SUBROUTINE SUB3
B=T
CALL SUB2
END

SUBROUTINE SUB4
B=T
CALL SUB6
END

SUBROUTINE SUB5
B=T
CALL SUB3
END

SUBROUTINE SUB6
DIMENSION A(1500)
B=T
CALL SUB9
END

SUBROUTINE SUB7
B=T
CALL SUB8
END

SUBROUTINE SUB8
B=T
CALL SUB4
END

SUBROUTINE SUB9
S=800.
y=EXP(S)
END
```
8.4
Execution Diagnostics

Computer Sample 9
Program MAIN
(continued)

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>000004</td>
<td>000004</td>
</tr>
<tr>
<td>SUN1</td>
<td>000012</td>
<td>000012</td>
</tr>
<tr>
<td>SUN2</td>
<td>000025</td>
<td>000025</td>
</tr>
<tr>
<td>SUN3</td>
<td>000040</td>
<td>000040</td>
</tr>
<tr>
<td>SUN4</td>
<td>000053</td>
<td>000053</td>
</tr>
<tr>
<td>SUN5</td>
<td>000066</td>
<td>000066</td>
</tr>
<tr>
<td>SUN6</td>
<td>000235</td>
<td>000235</td>
</tr>
<tr>
<td>SUB7</td>
<td>000500</td>
<td>000500</td>
</tr>
<tr>
<td>SUB8</td>
<td>000663</td>
<td>000663</td>
</tr>
<tr>
<td>SUB9</td>
<td>003076</td>
<td>003076</td>
</tr>
<tr>
<td>DGRERR</td>
<td>003114</td>
<td>003114</td>
</tr>
<tr>
<td>EXIT</td>
<td>003126</td>
<td>003126</td>
</tr>
<tr>
<td>END</td>
<td>003126</td>
<td>003126</td>
</tr>
<tr>
<td>STOP</td>
<td>003126</td>
<td>003126</td>
</tr>
<tr>
<td>EXP</td>
<td>003133</td>
<td>003133</td>
</tr>
<tr>
<td>EXPF</td>
<td>003133</td>
<td>003133</td>
</tr>
</tbody>
</table>

The total program and buffer space required is 003214

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>RAS</td>
<td>000012</td>
<td></td>
</tr>
<tr>
<td>FLS</td>
<td>000052</td>
<td></td>
</tr>
<tr>
<td>PCD</td>
<td>000066</td>
<td></td>
</tr>
<tr>
<td>FLR</td>
<td>000080</td>
<td></td>
</tr>
<tr>
<td>NFA</td>
<td>000100</td>
<td></td>
</tr>
<tr>
<td>EEA</td>
<td>000126</td>
<td></td>
</tr>
<tr>
<td>SCW</td>
<td>000151</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>000174</td>
<td></td>
</tr>
</tbody>
</table>

Total CPU time in milliseconds = 61
PPU time in milliseconds = 1074
Pages printed = 11
Total resources used = 95
## MATHEMATICAL LIBRARY FUNCTIONS

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOG</td>
<td>ARG ZERO OR NEG, INDEFINITE OR INFINITE ARGUMENT</td>
</tr>
<tr>
<td>ALOG10</td>
<td>ARG ZERO OR NEGATIVE, INDEFINITE OR INFINITE ARGUMENT</td>
</tr>
<tr>
<td>AMAX/AMIN</td>
<td>THERE MUST BE AT LEAST 2 ARGS</td>
</tr>
<tr>
<td>ASIN/ACOS</td>
<td>ARG GT 1, OR INDEFINITE</td>
</tr>
<tr>
<td>ATAN</td>
<td>INDEFINITE OR INFINITE ARG</td>
</tr>
<tr>
<td>ATAN2</td>
<td>INDEFINITE OR INFINITE ARG, OR X=Y=0</td>
</tr>
<tr>
<td>CBAIEX</td>
<td>ZERO TO ZERO OR NEGATIVE POWER</td>
</tr>
<tr>
<td>CLOG</td>
<td>ARG (0.,0.)</td>
</tr>
<tr>
<td>CSIN/CCOS</td>
<td>INFINITE OR INDEFINITE ARG</td>
</tr>
<tr>
<td>CUBRT</td>
<td>INDEFINITE OR INFINITE ARG</td>
</tr>
<tr>
<td>DATAN</td>
<td>INDEFINITE OR INFINITE ARG</td>
</tr>
<tr>
<td>DATAN2</td>
<td>X=Y=0 OR INDEFINITE OR INFINITE ARG</td>
</tr>
<tr>
<td>DBADEX</td>
<td>NEG BASE TO EXP OR ZERO BASE TO NEG EXP</td>
</tr>
<tr>
<td>DBAREX</td>
<td>NEG BASE TO EXP OR ZERO BASE TO NEG EXP</td>
</tr>
<tr>
<td>DCOS</td>
<td>ABS(ARG) = PI X 2 TO 94</td>
</tr>
<tr>
<td>DEXP</td>
<td>ARG GT 741.67 OR LT -675.82</td>
</tr>
<tr>
<td>DLOG</td>
<td>ARG ZERO OR NEGATIVE</td>
</tr>
<tr>
<td>DLOG10</td>
<td>ARG ZERO OR NEGATIVE</td>
</tr>
<tr>
<td>DMOD</td>
<td>INDEF OR INF ARG, X2=0, OR INTEGER PART TOO LARGE</td>
</tr>
<tr>
<td>DSIN</td>
<td>ABS(ARG) = PI X 2 TO 94</td>
</tr>
<tr>
<td>DSQRT</td>
<td>NEGATIVE ARG</td>
</tr>
<tr>
<td>EXP</td>
<td>ARG IN EXP GT 741.</td>
</tr>
<tr>
<td>IRAIEX</td>
<td>0 TO OTH POWER OR RESULT EXCEEDS 2 TO THE 48TH</td>
</tr>
<tr>
<td>RBAIEX</td>
<td>ZERO BASE TO ZERO OR NEG EXPONENT</td>
</tr>
<tr>
<td>RBAREX</td>
<td>EXPONENT OVERFLOW OR NEG BASE IN X TO Y</td>
</tr>
<tr>
<td>SIN/COS</td>
<td>ARGUMENT TOO LARGE</td>
</tr>
<tr>
<td>SQRT</td>
<td>NEGATIVE ARG IN SQRT</td>
</tr>
</tbody>
</table>
## UTILITY ROUTINES

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOGER</td>
<td>ERROR IN COMPUTED GO TO</td>
<td>The program has specified a number which is less than the first branch or greater than the Nth branch and a BRANCH TO statement has gone outside the specified range.</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>POSITIONING ERROR DURING BACKSPACE</td>
<td>The user has attempted to backspace a tape in the wrong format. Two different tape formats are written: the standard format from OUTPTC or OUTPTB, and the special format written by BUFFER OUT. The format is assumed to be standard unless a BUFFER statement has been executed prior to the backspace.</td>
</tr>
<tr>
<td>BRANRD</td>
<td>ATTEMPTED ANOTHER OPERATION WITHOUT CHECKING LAST</td>
<td>A BRANCK call has been omitted.</td>
</tr>
<tr>
<td></td>
<td>CANNOT RESET FILE</td>
<td>This is a hardware problem. See a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>CANNOT SET UP FILE</td>
<td>This is a system problem. See a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>EOF ENCOUNTERED</td>
<td>Check program logic.</td>
</tr>
<tr>
<td></td>
<td>UNSUCCESSFUL READ OR WRITE</td>
<td>Attempted to write a record larger than initial record by the same name, or the status cell has bit 59 set.</td>
</tr>
<tr>
<td>BUFFEI/O</td>
<td>TAPE UNIT NUMBER NOT IN 1-6764B RANGE</td>
<td>The logical unit specified in a BUFFER statement has not been requested on an ASSIGN card. Tapes 5 and 6 may not be used.</td>
</tr>
<tr>
<td></td>
<td>TOO MANY UNITS ASSIGNED</td>
<td>More than seven different units in one BUFFER program have been assigned.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONTROL</td>
<td>A COSY DECK WAS CALLED FOR FOR AND WAS NOT THERE</td>
<td>A Cosy deck is omitted or in the wrong place.</td>
</tr>
<tr>
<td></td>
<td>AN ALTER CARD SPECIFIED A REPLACE OR DELETE PAST THE END OF THE DECK</td>
<td>Check alter card punching.</td>
</tr>
<tr>
<td></td>
<td>ASSIGN LOGICAL UNIT</td>
<td>An *ASSIGN card is missing from the deck or is incorrectly punched. Check the logical unit number on the *ASSIGN card and the tape unit specified in the Fortran program. If the unit number (i) is missing from the comment printed on the output, an attempt was made to reference logical unit zero which was not assigned on an *ASSIGN card.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPTED TO DELETE FROM THE SYSTEM LIBRARY FILE</td>
<td>A file may not be deleted from ULIB.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPTED TO REWIND DD80</td>
<td>Do not rewind the dd80.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPTED TO REWIND PUNCH</td>
<td>Do not rewind punch.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPTED TO REWRITE THE SYSTEM LIBRARY FILE</td>
<td>A systems error. See a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>BINARY READ REQUEST OF A DPC FILE</td>
<td>A binary card occurred without a *BINARY control card, or a Cosy deck was not preceded by a *COSY card. This message is printed only when reading the input file.</td>
</tr>
<tr>
<td></td>
<td>BREAKPOINT</td>
<td>Termination occurred when breakpoint address flag matched P count. This is used for system debugging; see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>BUT ONE OF THESE AT A TIME-- PS,PC, PB, AF, RF</td>
<td>Only one of these requests may be specified on a monitor card such as *FORTRAN or *ASCENT.</td>
</tr>
</tbody>
</table>
## UTILITY ROUTINES

(Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL</td>
<td>CARD LIMIT EXCEEDED</td>
<td>Either the *LIMIT card specification for the punch is too small, or the *LIMIT card is not in the deck and the program punched more than 100 cards (the assumed limit).</td>
</tr>
<tr>
<td>CHECKSUM ERROR ON A COSY CARD</td>
<td></td>
<td>The original checksum on the Cosy card is different from the checksum recalculated at the time of a read.</td>
</tr>
<tr>
<td>DD60 I/O REQUEST NOT ALLOWED</td>
<td></td>
<td>A program may not write the dd60.</td>
</tr>
<tr>
<td>DD80 REQUEST WRONG LENGTH (MUST BE MULT OF 3 + 1)</td>
<td></td>
<td>The dd80 request must be a multiple of 3 + 1.</td>
</tr>
<tr>
<td>DPC CARD MUST FOLLOW A COSY DECK</td>
<td></td>
<td>Either a binary or other non-Cosy card is in the deck following a Cosy card.</td>
</tr>
<tr>
<td>DPC READ REQUEST OF A BINARY FILE</td>
<td></td>
<td>An Ascent card may be in a Cosy deck, or a Fortran card in a binary deck. This message is printed only when reading an input file.</td>
</tr>
<tr>
<td>END OF FILE FOUND ON INPUT FILE WHILE TRYING TO TERMINATE A COSY DECK READ</td>
<td></td>
<td>An unexpected control card appears, or a source deck card is omitted.</td>
</tr>
<tr>
<td>END OF FILE ON READER, PLIB, OR SLIB SOURCE</td>
<td></td>
<td>Library search reveals no such file. Check to see if a control card or a Cosy deck has been omitted.</td>
</tr>
<tr>
<td>END OF MODPACK NOT FOUND (END FILE, ERROR, OR BINARY CARD ENCOUNTED)</td>
<td></td>
<td>After the modpack, a deck to be modified is expected. The *SOURCE or *COSY card may have been omitted.</td>
</tr>
<tr>
<td>ERROR READING OR WRITING SYSTEM FILE ON A DRUM</td>
<td></td>
<td>A system file error has been sensed—see a systems programmer.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FRAME LIMIT EXCEEDED</td>
<td>The frame count exceeds the number of frames specified on the #LIMIT card. If the #LIMIT card is not in the deck, the frame count is assumed to be 100.</td>
<td></td>
</tr>
<tr>
<td>FUNCTION REQUESTED RESERVED FOR MONITOR USE</td>
<td></td>
<td>The request made is one of those which may be executed only by the monitor.</td>
</tr>
<tr>
<td>ILLEGAL CORE REDUCTION REQUEST</td>
<td></td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td>ILLEGAL DD80 OPERATION REQUESTED</td>
<td></td>
<td>The dd80 may not be referenced for this I/O.</td>
</tr>
<tr>
<td>ILLEGAL OPERATION REQUESTED OF THE PRINTER</td>
<td></td>
<td>An illegal request, such as a read, may have been issued to the printer.</td>
</tr>
<tr>
<td>ILLEGAL OR UNRECOGNIZED LOGICAL UNIT</td>
<td></td>
<td>The logical unit number may have been destroyed by the program.</td>
</tr>
<tr>
<td>ILLEGAL PUNCH OPERATION REQUESTED</td>
<td></td>
<td>The punch may not be referenced for this I/O.</td>
</tr>
<tr>
<td>ILLEGAL USE OF 100033B OP-CODE</td>
<td></td>
<td>This is a privileged code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td>IMPROPER OR MISSING FILE NAME</td>
<td></td>
<td>This file is not on the disk.</td>
</tr>
<tr>
<td>INDEFINITE</td>
<td></td>
<td>There is an indefinite floating-point result.</td>
</tr>
<tr>
<td>INVALID CONTROL FUNCTION CODE</td>
<td></td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td>I/O CALLING SEQUENCE CHAIN COUNT .GT. 1</td>
<td></td>
<td>CNT field in a RJ CM must be 1 or 0.</td>
</tr>
<tr>
<td>I/O CALLING SEQUENCE ENTRY OUT OF BOUNDS</td>
<td></td>
<td>FWA or LWA is outside the bounds of the program.</td>
</tr>
<tr>
<td>I/O CALLING SEQUENCE FWA .GE. LWA+1</td>
<td></td>
<td>LWA must be greater than FWA.</td>
</tr>
<tr>
<td>I/O CALLING SEQUENCE OUT OF BOUNDS</td>
<td></td>
<td>FWA or LWA has exceeded the program's field length.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONTROL (continued)</td>
<td>LAYOUT TABLE AREA EXCEEDS CORE</td>
<td>Table area exceeds 20000. In some cases, this is a systems error--see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>LAYOUT TABLE REQUEST TOO LARGE FOR CORE</td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>LCM BLOCK RANGE</td>
<td>An LCM block copy exceeds the fixed length of large core for your control point.</td>
</tr>
<tr>
<td></td>
<td>LCM DIRECT RANGE</td>
<td>A read or write LCM references an address greater than or equal to large core memory field length.</td>
</tr>
<tr>
<td></td>
<td>LCM PARITY</td>
<td>Submit job again.</td>
</tr>
<tr>
<td></td>
<td>NO STATUS CELL AVAILABLE</td>
<td>Check status cell in CM call.</td>
</tr>
<tr>
<td></td>
<td>OPERATOR DROPPED THIS JOB</td>
<td>The computer operator dropped the program at the console. A note accompanies the output explaining why he did this. Often a tight infinite loop is the reason. Check the control statement logic (IFs and DOs). Examine the $P counter on the panel printout to determine where in the program the drop occurred. This may help find an infinite loop.</td>
</tr>
<tr>
<td>OUTPUT FILE REWIND RQST WHEN NOT IN OUTPUT PHASE</td>
<td>A system error. See a systems programmer.</td>
<td></td>
</tr>
<tr>
<td>OVERFLOW</td>
<td></td>
<td>There is an overflow floating-point result.</td>
</tr>
<tr>
<td>PAGE LIMIT EXCEEDED</td>
<td></td>
<td>The page count exceeds pages specified on the $LIMIT card. If the $LIMIT card is not in the deck, then the page count is assumed to be 10. If 6 or PR has been assigned to the dd80, a frame (not page) count will be monitored.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PARTIAL CARD PUNCH REQUEST</td>
<td>Sixteen words must be punched on a card when punching binary.</td>
<td></td>
</tr>
<tr>
<td>PERIPHERAL TIME LIMIT EXCEEDED</td>
<td>PP time exceeds time specified on the *LIMIT card.</td>
<td></td>
</tr>
<tr>
<td>PLIB NAME FILE TOO BIG (OVER 5000 OCTAL)</td>
<td>PLIB names exceed the number of words in the table. No more names may be added.</td>
<td></td>
</tr>
<tr>
<td>PLIB OR SLIB RECORD REQUESTED NOT ON DISK</td>
<td>The record specified is not recorded on the disk.</td>
<td></td>
</tr>
<tr>
<td>PP BUFFER REQUEST CORE OVERFLOW</td>
<td>See a systems programmer.</td>
<td></td>
</tr>
<tr>
<td>PROGRAM MAY NOT SHRINK LAYOUT TABLES</td>
<td>See a systems programmer.</td>
<td></td>
</tr>
<tr>
<td>PROGRAM RANGE</td>
<td>This is a PROGRAM STOP. The P register equals 0 or an error exit instruction was issued.</td>
<td></td>
</tr>
<tr>
<td>PROGRAM STOPPED</td>
<td>A program stop instruction was encountered by the peripheral monitor in testing the first 6 bits (operations code) of the word. In a Fortran program look for arrays not dimensioned for enough storage or for faulty indexing. The program may be trying to execute data. There may be no external statement for a routine in a parameter list. There may be too few arguments in a calling sequence. If the program has overlap with dimensions and no entry point is provided, the program stops.</td>
<td></td>
</tr>
<tr>
<td>PUNCH REQUEST FOR MORE THAN 4000 CARDS</td>
<td>The punch request on *ASSIGN card is too long.</td>
<td></td>
</tr>
<tr>
<td>REQUEST FOR AN INVALID BUFFER CHANGE</td>
<td>Check CM call for correct arguments.</td>
<td></td>
</tr>
</tbody>
</table>
## Utility Routines

(Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL (continued)</td>
<td>REQUESTED CORE EXCEEDS MACHINE SIZE</td>
<td>The program and subroutines have used more core than is available. Economize on array size by decreasing dimensions or using equivalence or common more efficiently. The loader may be overlapped with a blank common block at least 4000 words long specified in the first program, which can save between 3000 and 4000 locations.</td>
</tr>
<tr>
<td>SCM BLOCK RANGE</td>
<td>An SCM block copy request exceeds small core field length.</td>
<td></td>
</tr>
<tr>
<td>SCM DIRECT RANGE</td>
<td>An address in SCM is outside the bounds of the program field length.</td>
<td></td>
</tr>
<tr>
<td>SCM PARITY</td>
<td>Submit job again.</td>
<td></td>
</tr>
<tr>
<td>STATUS CELL FOR WAIT REQUEST NOT FOUND</td>
<td>Check CM call for correct arguments.</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>This program is running in step mode, and an error exit (EEA) occurs for each instruction.</td>
<td></td>
</tr>
<tr>
<td>SYSTEM LOST WHILE TRYING TO INSERT A MODPACK INTO THE INPUT STREAM</td>
<td>There may be an overlapping instruction. Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>INSERT 10</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>REPLACE 11,15</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>See a systems programmer.</td>
<td></td>
</tr>
<tr>
<td>THE SECOND NUMBER ON AN ALTER CARD MUST BE GE THAN THE FIRST</td>
<td>Check alter card punching.</td>
<td></td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIME LIMIT EXCEEDED</td>
<td>The *LIMIT card is missing and the time is assumed to be 1 minute, or the program exceeds the time limit specified on the *LIMIT card.</td>
<td></td>
</tr>
<tr>
<td>TOO MUCH OUTPUT</td>
<td>There is too much dd80 output.</td>
<td></td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>There is an underflow floating-point result.</td>
<td></td>
</tr>
<tr>
<td>WAIT REQUEST HAS NO STATUS</td>
<td>Check CM call for correct arguments.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drum or Disk Errors</td>
<td>ATTEMPT TO MODIFY PLIB WHEN IT IS IN USE BY OTHER CONTROL POINTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO READ UNWRITTEN LIBRARY RECORD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO USE PLIB WHEN IT IS RESERVED FOR MODIFICATION BY ANOTHER CP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO WRITE 2 FILE MARKS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAN'T LOCATE NAME FOR SWAP (DD710)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAN'T UPDATE PROGRAM LIBRARY FILE NOT TERM OR SYS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRUM OR DISK PARITY ERROR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ILLEGAL OPERATION REQUESTED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ILLEGAL OPERATION SEQUENCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBRARY DIRECTORY TOO SHORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBRARY FILE UNIT NOT AVAILABLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBRARY RECORD LIST OVERFLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAXIMUM BLOCK RESERVATION CODE EXCEEDED</td>
<td></td>
</tr>
</tbody>
</table>
### Utility Routines (Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drum or Disk Errors</td>
<td>NO STORAGE SPACE LEFT FOR LIBRARY OR RANDOM FILE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NOT ENOUGH SPACE LEFT TO COMPLETE THE OP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROGRAM LIBRARY PROJECT NUMBER NOT IN DIRECTORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROJECT NUMBER NOT IN GENERAL DIRECTORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECORD LIST BUFFER NOT ASSIGNED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECORD LIST BUFFER TOO SHORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECORD NAME MISSING IN LIBRARY CALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECORD OUT OF BOUNDS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RL CONTINUATION LIST ERROR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEARCH FILE FORWARD IS ILLEGAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNIT CONTROL TABLE OVERFLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRITE, WEF, SFF PRECEDE A READ OP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRITE WITH 0 WORD COUNT</td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td>CENTRAL PROCESSOR DUMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The *TRAP monitor control card, required to get a dump in addition to the panel, is not in the deck.</td>
<td></td>
</tr>
<tr>
<td>EXIT MODE</td>
<td>FLOATING POINT OVERFLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A number greater than $10^{322}$ has been generated. Look for a divide by a very small number or zero, or for a function which is increasing exponentially and exceeds this number.</td>
<td></td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INDEFINITE OPERAND</td>
<td>A ± 1777 in the exponent field is stored in a variable to which the program</td>
<td>A ± 1777 in the exponent field is stored in a variable to which the program is applying a floating-point operation (/, *, +, -) code. This indefinite has been generated somewhere earlier in the program. Look for errors in mixed-mode references, or in mixed-mode equivalence logic, as well as a % operation. Some subroutines pack an indefinite for an error exit.</td>
</tr>
<tr>
<td>MEMORY BOUNDS ERROR</td>
<td>The program has referenced an operand or instruction which is outside the core</td>
<td>The program has referenced an operand or instruction which is outside the core area (FL) reserved for the program. The program may have a &quot;wild&quot; jump and be trying to execute the data. Check array dimensions and DO-loop indices. There may be too few arguments in a calling sequence.</td>
</tr>
<tr>
<td>INPUTB/OUTPTB</td>
<td>ATTEMPT TO WRITE TAPE WITH NO RING REQUESTED, LOGICAL UNIT NO.</td>
<td>Check the *ASSIGN card for an R. (INPUTB is used by a READ(i) or READ TAPE i instruction; OUTPTB is used by a WRITE(i) or WRITE TAPE i instruction.)</td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO WRITE 2 FILE MARKS, LOGICAL UNIT NO.</td>
<td>Do not write two file marks in succession.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO WRITE WITH ZERO WORD COUNT, LOGICAL UNIT NO.</td>
<td>Check this I/O request for an illegal list.</td>
</tr>
<tr>
<td></td>
<td>CHECKSUM ERROR, LOGICAL UNIT NO.</td>
<td>The read has calculated a checksum which does not agree with control word 2.</td>
</tr>
<tr>
<td></td>
<td>DISK/DRUM ERROR, LOGICAL UNIT NO.</td>
<td>Possibly a parity error.</td>
</tr>
<tr>
<td></td>
<td>DISK/DRUM SFF IS ILLEGAL, LOGICAL UNIT NO.</td>
<td>No file mark can be searched this time. SFF is search forward file.</td>
</tr>
<tr>
<td></td>
<td>DRUM/DISK ILLEGAL OP REQUESTED, LOGICAL UNIT NO.</td>
<td>An illegal request has been made at this unit number.</td>
</tr>
</tbody>
</table>
## Utility Routines

(Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTB/OUTPTB (continued)</td>
<td>DRUM/DISK ILLEGAL OP SEQUENCE, LOGICAL UNIT NO.</td>
<td>Check I/O statements for an illegal request.</td>
</tr>
<tr>
<td></td>
<td>END OF FILE READ ON LOGICAL UNIT NO.</td>
<td>An EOF was encountered.</td>
</tr>
<tr>
<td></td>
<td>END OF TAPE ENCOUNTERED ON LOGICAL UNIT NO.</td>
<td>The tape is not long enough, and the read or write has gone off the end.</td>
</tr>
<tr>
<td></td>
<td>LOGICAL UNIT NOT IN SYSIOU TABLE =</td>
<td>The unit has been changed or omitted.</td>
</tr>
<tr>
<td></td>
<td>NEGATIVE RECORD LENGTH, LOGICAL UNIT NO.</td>
<td>There is an error in the I/O statements.</td>
</tr>
<tr>
<td></td>
<td>NOT ENOUGH SPACE LEFT TO COMPLETE OP, LOGICAL UNIT NO.</td>
<td>The end of the tape has been reached.</td>
</tr>
<tr>
<td></td>
<td>PARITY ERROR ON SKIP BAD SPOT</td>
<td>A new tape may be needed.</td>
</tr>
<tr>
<td></td>
<td>R. L. CONTINUATION LIST ERROR, LOGICAL UNIT NO.</td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>RECORD LENGTH EXCEEDED, LOGICAL UNIT NO.</td>
<td>There is an error in the I/O statements.</td>
</tr>
<tr>
<td></td>
<td>TAPE ERROR, LOGICAL UNIT NO.</td>
<td>There is a parity error or an illegal request on that unit.</td>
</tr>
<tr>
<td></td>
<td>UNIT CONTROL TABLE OVERFLOW, LOGICAL UNIT NO.</td>
<td>Too many units are assigned.</td>
</tr>
<tr>
<td></td>
<td>UNRECOVERABLE TAPE READ ERROR, LOGICAL UNIT NO.</td>
<td>The tape may have been written with even parity.</td>
</tr>
<tr>
<td></td>
<td>UNRECOVERABLE TAPE WRITE ERROR, LOGICAL UNIT NO.</td>
<td>Make the tape again. If it still fails, ask for a new tape.</td>
</tr>
<tr>
<td></td>
<td>WRITE, WEF OR SFF PRECEDE READ OP, LOGICAL UNIT NO.</td>
<td>An EOF has been placed at the load point.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INPUTC/INPUTS</td>
<td>DATA OVERFLOW - EXPONENT</td>
<td>Check for the characters other than D, E, +, -, in an E field to see if the exponent is too large. It may also not be right-adjusted. (This routine is called whenever a READn,L, READ(i,n)L, or READ INPUT TAPE i,n,L is executed. The FORMAT statement INPUTS is called whenever DECODE(c,m,V)L is executed. It is scanned at execution time, not during compile time.)</td>
</tr>
<tr>
<td>END FILE ON INPUT</td>
<td>No panel is output for this error message.</td>
<td>This diagnostic occurs if the program tries to execute a READ statement when there is no more data on the input file. Check the data for card count on large arrays or for indexing errors in the READ statement. Many programmers terminate on an END OF FILE by branching back to a READ instead of counting data sets. In this case there is no programming error; the message is informative.</td>
</tr>
<tr>
<td>EXCEEDED RECORD SIZE</td>
<td>A READ statement may specify 80 card columns or 50 characters from a BCD tape for input. Examine format for too many repeated groups or an omitted slash.</td>
<td></td>
</tr>
<tr>
<td>EXPONENT TOO LARGE ON DATA INPUT</td>
<td>A data card has an exponent which is too large. Check the field width.</td>
<td></td>
</tr>
<tr>
<td>HOLLERITH FORMAT WITH LIST</td>
<td>No field specification for the list is contained in the FORMAT statement.</td>
<td></td>
</tr>
</tbody>
</table>
## Utility Routines (Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTC/INPUTS</td>
<td>ILLEGAL DATA ENCOUNTERED</td>
<td>Look for a decimal point in an I field, a decimal digit in an O field, or an alphabetic or illegal special character in an I, O, E, D, or F field. The indexing on the READ statement may be too large or the data card count is wrong, and the card read does not go with this FORMAT statement.</td>
</tr>
<tr>
<td></td>
<td>ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT</td>
<td>A number specification has been made using a character other than X, I, A, R, F, E, G, D, 0, H, *, L, P, +, or -. Look for keypunch errors. A wild index or too small a DIMENSION statement may have caused some of the core to be overwritten. Also check the count in an H specification.</td>
</tr>
<tr>
<td>INTEGER TOO LARGE ON DATA INPUT</td>
<td></td>
<td>An integer greater than $\pm 2^{53} - 1$ is specified on input.</td>
</tr>
<tr>
<td>MULTIPLE DECIMAL POINTS IN DATA</td>
<td></td>
<td>An illegal data field has two or more decimal points.</td>
</tr>
<tr>
<td>PAREN GROUP NOT CLOSED IN FORMAT STATEMENT</td>
<td></td>
<td>There are more left than right parentheses in the FORMAT statements.</td>
</tr>
<tr>
<td>TAPE ERROR DURING READ OPERATION</td>
<td></td>
<td>The program attempted to read a binary tape in BCD mode or there was a parity error.</td>
</tr>
<tr>
<td>TAPE UNIT NUMBER NOT IN 1-6761B RANGE GIVEN UNIT NUMBER =</td>
<td></td>
<td>The unit number assigned to a tape is outside the legal range.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>ZERO FIELD WIDTH IN FORMAT STATEMENT</td>
<td>There is a zero field width (such as F0.3, or IO). Look for commas omitted after the specification or an error in the Hollerith count. Also look for indexing errors or DIMENSION statement errors which may have disrupted the FORMAT statement.</td>
</tr>
<tr>
<td>IOCHEK</td>
<td>IF UNIT CALL BEFORE I/O CALL</td>
<td>The IF UNIT tests may not be used before the first BUFFER statement. (Called by IF UNIT tests in Fortran statements.)</td>
</tr>
<tr>
<td>IOPROC</td>
<td>ATTEMPT TO USE AN ILLEGAL MODE NUMBER</td>
<td>The mode is greater than three or is negative.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO USE AN UNASSIGNED UNIT NUMBER</td>
<td>The unit number used has not been assigned.</td>
</tr>
<tr>
<td></td>
<td>ATTEMPT TO USE ILLEGAL UNIT NUMBER</td>
<td>The unit number is illegal or is greater than 6761B.</td>
</tr>
<tr>
<td>LENGTHF</td>
<td>LENGTHF CALL BEFORE I/O CALL</td>
<td>The BUFFER statement on a given unit must be programmed before calling LENGTHF(i). Check the logical unit numbers.</td>
</tr>
<tr>
<td>OUTPTC/OUTPTS</td>
<td>FIELD WIDTH LESS THAN DECIMAL WIDTH</td>
<td>In a format, Fw.d, the width (w) is not large enough to accommodate d, the number of decimal digits. Example: F6.7 is a field width error. (OUTPTC is the routine that is called whenever you have a PRINT, PUNCH, WRITE OUTPUT TAPE i,n,L, or WRITE(i,n)L. OUTPTS is called whenever you have an ENCODE(c,m,V)L statement. The FORMAT statement is scanned at execution time, not during compile time.)</td>
</tr>
<tr>
<td></td>
<td>HOLLERITH FORMAT WITH LIST</td>
<td>The FORMAT statement does not contain a field specification for the list.</td>
</tr>
</tbody>
</table>
### Utility Routines

#### Execution Diagnostics

(Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPTC/OUTPTS (continued)</td>
<td>ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT</td>
<td>A number specification has been made using a character other than X, I, A, R, E, D, F, G, O, H, *, L, F, +, or -. Look for keypunch errors. A wild index or too small a DIMENSION statement may have caused some of the core to be overwritten. Also check the count in an H specification.</td>
</tr>
<tr>
<td></td>
<td>OUTPUT RECORD LENGTH EXCEEDED</td>
<td>The number of characters the program may write per line is 128. Examine format for too many repeated groups or an omitted slash.</td>
</tr>
<tr>
<td></td>
<td>PAREN GROUP NOT CLOSED IN FORMAT STATEMENT</td>
<td>Check parentheses count; there should be an equal number of left and right parentheses.</td>
</tr>
<tr>
<td></td>
<td>SINGLE ARRAY, D FORMAT</td>
<td>A double-precision format has been used for a single-precision array.</td>
</tr>
<tr>
<td></td>
<td>TAPE ERROR ON OUTPUT</td>
<td>There is a parity error in the tape unit, or a binary tape write has been attempted under a BCD write.</td>
</tr>
<tr>
<td></td>
<td>UNIT NO. NOT PUNCH UNIT, OR IN 1-6761B RANGE GIVEN UNIT NUMBER =</td>
<td>The punch unit is not correct.</td>
</tr>
<tr>
<td></td>
<td>ZERO FIELD WIDTH IN FORMAT STATEMENT</td>
<td>The w in the format specification is zero, specifying F0.6 or F0.10, etc. Look for a wild index or dimensions that are too small; the FORMAT statement may be disrupted. Look for keypunch error.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>OVERLAY LIST DOES NOT CONTAIN OVLY/SEG</td>
<td>The program has called for an overlay or a segment that has not been loaded with the program deck. A CALL OVERLAY(IO,IS) statement is incorrect. An OVERLAY card has not been placed in a deck. Check description of routine OVERLAY.</td>
</tr>
<tr>
<td>FDUMP</td>
<td>ARGUMENT EXCEEDS FIELD LENGTH</td>
<td>A dump limit is greater than the program field length.</td>
</tr>
<tr>
<td>PLOT</td>
<td>ARG 5-8 IN SET ZERO OR NEGATIVE WITH LOG SCALING</td>
<td>Check these arguments for negative or zero values.</td>
</tr>
<tr>
<td></td>
<td>FLASH BUFFER OVERFILL</td>
<td>The user's FLASH buffer has been exceeded.</td>
</tr>
<tr>
<td></td>
<td>OPERATOR DROP OF DD80 ONLINE JOB (DD80 FAULT)</td>
<td>The camera may be out of film. Set sense switch 1 ON to drop the job.</td>
</tr>
<tr>
<td></td>
<td>TRIED TO START FLASH WHILE IN FLASH</td>
<td>The programmer tried to start emptying the FLASH buffer while still collecting flash instructions.</td>
</tr>
<tr>
<td></td>
<td>X OR Y COOR ZERO OR NEGATIVE WITH LOG SCALING</td>
<td>Examine plotter coordinates to see that they are not zero or negative.</td>
</tr>
<tr>
<td>Q8QIOU</td>
<td>ATTEMPT TO READ/WRITE NONEXISTENT TAPE FORMAT</td>
<td>There is no FORMAT statement.</td>
</tr>
<tr>
<td></td>
<td>LOGICAL UNIT TABLE OVERFLOW</td>
<td>Too many units are assigned.</td>
</tr>
<tr>
<td></td>
<td>TAPE FORMAT CHANGED FROM LOGICAL TO PHYSICAL</td>
<td>A WRITE(i) LIST and a BUFFER OUT(i,p)(FM,EM) are not allowed on the same unit number.</td>
</tr>
<tr>
<td></td>
<td>TAPE FORMAT CHANGED FROM PHYSICAL TO LOGICAL</td>
<td>A tape instruction request on a unit has been changed. For example, WRITE(9)A might be followed by BUFFER IN(9,1) (A(1),A(100)).</td>
</tr>
</tbody>
</table>
### Utility Routines (Continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q8QI0U (continued)</td>
<td>TAPE MANIPULATION BEFORE FORMAT ESTABLISHED</td>
<td>See a systems programmer.</td>
</tr>
<tr>
<td>RANINT/RANWT/RANRD</td>
<td>UNABLE TO ESTABLISH RANFILE</td>
<td>The random file has not been created during initialization.</td>
</tr>
<tr>
<td></td>
<td>UNABLE TO RESET RANFILE</td>
<td>A programmed file has not been made.</td>
</tr>
<tr>
<td>SAVEF</td>
<td>CHECKSUM ERROR DURING RESTART PHASE</td>
<td>The tape made by the SAVEF routine cannot be read on restart. Check that the correct tape is assigned.</td>
</tr>
<tr>
<td></td>
<td>DRUM READ ERROR</td>
<td>The drum failed on a read operation.</td>
</tr>
<tr>
<td></td>
<td>DRUM WRITE ERROR</td>
<td>The drum failed on a write operation.</td>
</tr>
<tr>
<td></td>
<td>ERROR IN READING OVERLAYS FROM DRUM</td>
<td>A drum read error occurred on restart.</td>
</tr>
<tr>
<td></td>
<td>ERROR READING LCM DATA FROM SAVE TAPE</td>
<td>Submit job again, then see a systems programmer. It may be necessary to resave the job.</td>
</tr>
<tr>
<td></td>
<td>ERROR READING OVERLAYS FROM PROGRAMMER LIB. FILE</td>
<td>Try to save once more, then see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>ERROR WRITING OVERLAYS ON PROGRAMMER LIBRARY FILE</td>
<td>Try to save once more, then see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>FAILED TO READ RESIDENT OVERLAY FROM DRUM</td>
<td>A drum read error occurred on restart.</td>
</tr>
<tr>
<td></td>
<td>FAILURE TO WRITE RESIDENT OVERLAY TO DRUM</td>
<td>The program is using subroutine OVERLAY. In the process of moving overlays to the drum during the SAVEF procedure, an unrecoverable drum write error has occurred. Notify the operator.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TAPE READ ERROR</td>
<td>In the process of reading the save tape during a restart, an unrecoverable read error has occurred. Have the operator mount the tape on a different unit and try again.</td>
<td></td>
</tr>
<tr>
<td>TAPE WRITE ERROR</td>
<td>In the process of writing the save tape, an unrecoverable write error has occurred. Assign a new tape to be used as the save tape.</td>
<td></td>
</tr>
<tr>
<td>SW6 TERM BY SAVE</td>
<td>No panel is output for this error message. The program is using the SAVF routine to implement restart procedures on a long job. The operator has set sense switch 6 to force the SAVF routine to terminate the program for restart at a later time. (This message is printed by SAVF and not with a RJ Q8QERR.)</td>
<td></td>
</tr>
<tr>
<td>UNABLE TO ESTABLISH</td>
<td>An error occurred during execution of subroutine WTHIS. The third argument = 1.</td>
<td></td>
</tr>
<tr>
<td>PROGRAMMER FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNABLE TO ESTABLISH</td>
<td>An error in restarting. See a systems programmer.</td>
<td></td>
</tr>
<tr>
<td>RANDOM FILE BUFFER ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTART</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LOAD DIAGNOSTICS**

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADER</td>
<td>ATTEMPT TO REDEFINE COMMON BLOCK LENGTH</td>
<td>A common block length may not exceed the original block length specified.</td>
</tr>
<tr>
<td></td>
<td>BLANK COMMON MAY NOT BE PRESET</td>
<td>DATA statements are not legal for variables in blank common.</td>
</tr>
<tr>
<td></td>
<td>CARD MISSING OR DECK OUT OF ORDER</td>
<td>Check the control cards in the deck.</td>
</tr>
<tr>
<td></td>
<td>CHECKSUM ERROR</td>
<td>A card has a checksum. Submit deck again.</td>
</tr>
<tr>
<td></td>
<td>ENTRY PT TABLE OVERFLOW</td>
<td>There are too many routine names in the deck. They exceed the current table size.</td>
</tr>
<tr>
<td></td>
<td>ERROR READING SYSTEM LIBRARY</td>
<td>Submit deck again; then see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>ERROR WRITING ENTRY TABLE ONTO RANFILE</td>
<td>Submit deck again; then see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>ERROR WRITING OVERLAYS ONTO RANFILE</td>
<td>Submit deck again; then see a systems programmer.</td>
</tr>
<tr>
<td></td>
<td>EXTERNAL AND ENTRY TABLES DON'T MATCH</td>
<td>A routine referenced is not available to the LOADER.</td>
</tr>
<tr>
<td></td>
<td>EXTERNAL CHAIN TABLE OVERFLOW</td>
<td>There are too many subprograms called for loading entry points.</td>
</tr>
<tr>
<td></td>
<td>INCREMENT TABLE OVERFLOW</td>
<td>A LOADER table has overflowed.</td>
</tr>
<tr>
<td></td>
<td>LIBRARY DIRECTORY ERROR</td>
<td>There may be missing entry points in the library routines.</td>
</tr>
<tr>
<td></td>
<td>LIBRARY EXTERNAL REQUEST OVERFLOW</td>
<td>There are too many external library references. They exceed the table size.</td>
</tr>
<tr>
<td></td>
<td>LOADER FAILURE, CORE REQUIRED=XXXXXX, CORE AVAILABLE=XXXXXX</td>
<td>The program did not load because it required too much core. Use blank common if possible in your program.</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Error Message</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOCAL EXTERNAL TABLE OVERFLOW</td>
<td>The number of external entries exceeds the limit for a given routine.</td>
<td></td>
</tr>
<tr>
<td>NO ENTRY PTS IN AN OVERLAY OR SEGMENT</td>
<td>Make sure that overlays are subprograms except for (0,0).</td>
<td></td>
</tr>
<tr>
<td>NO MAIN PROGRAM</td>
<td>There must be a main program in order to execute the job.</td>
<td></td>
</tr>
<tr>
<td>OVERLAY TRANSFER POINT NOT IN ENTRY LIST</td>
<td>Check overlay numbers that are called from the driver; one may be missing.</td>
<td></td>
</tr>
<tr>
<td>SEGMENT WITH MISSING OR INCORRECT OVERLAY</td>
<td>Check overlay and segment cards.</td>
<td></td>
</tr>
<tr>
<td>TOO MANY COMMON BLOCKS</td>
<td>The number of common blocks exceeds the current limit, or there may be more than the current limit in one routine.</td>
<td></td>
</tr>
<tr>
<td>TOO MANY LIBRARY ROUTINES REQUESTED</td>
<td>There are too many library routines requested.</td>
<td></td>
</tr>
<tr>
<td>TOO MANY OVERLAYS AND SEGMENTS</td>
<td>The table limit for overlays and segments is exceeded.</td>
<td></td>
</tr>
<tr>
<td>2 ENTRY PTS WITH THE SAME NAME</td>
<td>Check for a nondimensioned subscripted variable with the same name as a subprogram, or two subprograms with the same name.</td>
<td></td>
</tr>
<tr>
<td>2 MAIN PROGRAMS</td>
<td>Only one main program is allowed.</td>
<td></td>
</tr>
</tbody>
</table>
8.26
Execution Diagnostics

<table>
<thead>
<tr>
<th>LOADER TABLE SIZE LIMITS&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Table Description</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entry point table</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>External chain pointer table</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Missing subroutines</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Library routines</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Library externals</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Common blocks</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Local common blocks (in one subprogram)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Local external entries</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Total of overlays and segments</td>
<td>16</td>
</tr>
</tbody>
</table>

<sup>1</sup>These limits are current as of the publication date of this manual. They are subject to change.
| **Absolute value (in-line functions)**, 1.2 | **BNDZRO** (ULIB), 5.3 |
| **Accessing data**, 6.1-10 | **BRANCK** (BRANRD entry point), 3.8 |
| **ACGOER**, 3.3 | **BRANRD**, 3.6-9 |
| diagnostic, 8.6 | diagnostics, 8.6 |
| **ACOS**, 2.9 | entry points |
| **Address, relative**, 8.1 | **BRANCK**, 3.8 |
| **ADI** (ULIB), 5.25 | **BRANRD** 3.7 |
| **ALGUER** (ULIB), 5.26 | **BRANST**, 3.9 |
| **ALOG**, 2.10-11 | **BRANWT**, 3.7 |
| **ALOG10**, 2.12 | **BRANST** (BRANRD entry point), 3.9 |
| **AMAX/AMIN**, 2.13-14 | **BRANWT** (BRANRD entry point), 3.7 |
| **ASA** (ULIB), 5.29 | **BSEARCH** (ULIB), 5.20 |
| **ASIN**, 2.15 | **BSJ** (ULIB), 5.9 |
| **ATAN**, 2.16-17 | **BSTAPE** (IOPROC entry point), 3.50 |
| **ATAN2**, 2.18-19 | **BUFCL** (BUFRD entry point), 3.20 |
| **AXES** (PLOT entry point), 4.33; see also 4.4 | **BUFFEI**, 3.10-13 |
| **Backgrounds** (PLOT), 4.29-40 | diagnostics, 8.6, 8.19 |
| **BACKSP** (backspace), 3.4-5 | entry points |
| diagnostic, 8.6 | **BUFFEI**, 3.10-11 |
| entry points | **BUFFEO**, 3.11 |
| **BACKSP**, 3.4 | **IOCHEK**, 3.12 |
| **IFENDF**, REWINM, 3.5 | **LENGTHF**, 3.13 |
| **BACKSPACE i** command, 3.4 | **BUFFEO** (BUFFEI entry point), 3.11 |
| **Backspace one record**, 3.4, 3.50 | diagnostics, 8.6 |
| **BAND3** (ULIB), 5.18 | **BUFFER IN/OUT**, 3.10-13; see also 5.20 |
| **BESLI** (ULIB), 5.9 | Buffer, input/output (CM), 7.12-13 |
| **BESLJ** (ULIB), 5.9 | **Buffering**, see BRANRD, BUFFEI, BUFRD, |
| **BESLK** (ULIB), 5.9 | PLOT buffer control, and RANRD |
| **BESLY** (ULIB), 5.9 | **BUFINT** (BUFRD entry point), 3.17-18 |
| Blocking, see Tape blocking, unblocking | **BUFRD**, 3.14-24 |
| **BMD02R** (ULIB), 5.29 | entry points |
| **BMD02S** (ULIB), 5.29 | **BUFCL**, 3.20 |
| **BMD02V** (ULIB), 5.29 | **BUFFINT**, 3.17-18 |
| **BMD04M** (ULIB), 5.29 | **BUFRD**, 3.19 |
| **BMD05R** (ULIB), 5.29 | **BUFWT**, 3.18-19 |
| **BMD06M** (ULIB), 5.29 | exchange units, 3.16 |
| **BNDSLV** (ULIB), 5.18 | input phase, 3.15 |
| | output phase, 3.16 |
| | rewind and exchange units, 3.16 |
| | sample program, 3.21-24 |
| | **BUFWT** (BUFRD entry point), 3.18-19 |
| Byte accessing, see GBYTES, 3.41, 6.9 |
CABS, 2.20
CALCNT, 4.48-50, 5.13
Call-by-name routines, 2.3-6
Call-by-value routines, 2.1-3
Card-to-card routine, see REPBIN, 6.10
Card-to-tape routines, see CDTAPE and CDTAPEB, 6.9
CBAIEX, 2.21
CCOS, 2.22
CDTAPE, 6.9
CDTAPEB, 6.9
Central monitor routines, 7.1-15
arguments for CM call, 7.7-11
CNT, 7.8
FWA, 7.11
LWA, 7.11
OP, 7.7-8
RETURN, 7.7
STATUS, 7.9-11
UNIT, 7.9
arguments for CM MACRO call, 7.12
Ascent calling sequence, 7.7
buffer, input and output, 7.12-13
communications area, 7.1-6
error messages, 7.1
hardware and software interrupts, 7.2-6
I/O file handling, 7.12-15
examples, 7.14-15
monitor communications area, 7.1-6
registers, 7.7
7600 cell 0, 7.5-6
hardware and software interrupts, 7.5-6
6600 cell 0, 7.2-4
hardware and software interrupts, 7.2-4
CEXP, 2.23
CHARAC (ULIB), 5.25
Character conversion in core, see CHCONV, 6.9
CHCONV, 6.9
CLOCKF/DATEN, 2.24
CLOG, 2.25
CM call, arguments, see Central monitor
routines
CM MACRO call, arguments, 7.12
CM routines, see Central monitor
routines, 7.1-15
CMPXQR (ULIB), 5.16
CNT (CM call argument), 7.8
COMBES (ULIB), 5.9
Communications area, central monitor, 7.1-6
CONTROL diagnostics, 8.7-13
Control (PLOT), 4.22-28
Conversion
ded, 5.20, 6.9
in-line functions, 1.2
CONVOL (ULIB), 5.26
CONV360 (ULIB), 5.20
Copy output, 6.7
Copy tapes, see Tape copy and edit, 6.1-8
CORFOR, 6.9
CORR (ULIB), 5.30
COS, 2.26
CPOLY (ULIB), 5.3
CRT address, 4.3, 4.6-9
CSIN, 2.27
CSQRT, 2.28
CUBIC (ULIB), 5.3
CUBRT, 2.29-30
CUBTRP (ULIB), 5.8
CURVE (PLOT entry point), 4.14;
see also 4.4
CX (ULIB), 5.10
CXERFC (ULIB), 5.10
DARIV (ULIB), 5.8
DASHLN (PLOT entry point), 4.26-27
Data accessing, 6.1-10
Data blocking, see Tape blocking, un-
blocking
Data handling routines, 6.1-10
Data processing, 6.1-10
DATAN, 2.31
DATAN2, 2.32-33
DATE (ULIB), 5.20
DATEF, 2.24
DBADEX, 2.34
DBAIEX, 2.35
DBAREX, 2.36
DCOS, 2.37
dd80, 4.1, see Plotting library routines
character set, 4.19
PLOT, 4.5
entry points, 4.2
DEBUG, 3.25-34
  sample program, 3.30-34
  snap cards, 3.27-29
Deck conversion, 5.20, 6.9
DECODE statement, 3.48-49
DECOMP (ULIB), 5.19
DERIV (ULIB), 5.8
DET (ULIB), 5.17
Determinants (ULIB), 5.17
Diagnostics, see Execution diagnostics, 8.1-23
Differential equations (ULIB), 5.24-25
Disk, user storage, 3.80
Disk errors, 8.13-14
DLOG, 2.40-41
DLOG10, 2.42
DMAX1/DMIN1, 2.43
DMOD, 2.44
Drum errors, 8.13-14
DSIGN, 2.45
DSIN, 2.46
DSQRT, 2.47
DUMP, 3.35
diagnostic, 8.14
Dump
  entire or part of program (PDUMP), 3.77-79
  trace, 8.3-4
DUP (ULIB), 5.8

EDIT (ULIB), 5.20
EI (ULIB), 5.10
Eigenvalues
  and eigenvectors of a matrix (ULIB), 5.16
  of ordinary differential equations (ULIB), 5.24
ELIP (ULIB), 5.10
 ENCODE statement, 3.60
END (EXIT entry point), 3.37
END FILE I command, 3.36
ENDFIL, 3.36
End-of-file check, 3.5
EOF, I, 3.5
EOT (ULIB), 5.20
EQSLV (ULIB), 5.18
ERF and MERFFAST (ULIB), 5.10
Error messages, see Execution diagnostics, 8.1-23
Examples, see Sample programs
Exchange jump, 7.1, 7.7-8
Execution diagnostics, 8.1-23
examples, 8.1-4
LOADER diagnostics, 8.24-25
mathematical library routines, 8.5
trace with *TRAP, 8.3-4
utility library routines
ACGOER, 8.6
BACKSPACE, 8.6
BRANRD, 8.6
BUFFEI/O, 8.6
CONTROL, 8.7-13
disk/drum errors, 8.13-14
DUMP, 8.14
EXIT MODE, 8.14-15
INPUTB/OUTPTB, 8.15-16
INPUTC/INPUTS, 8.17-19
IOPEN, 8.19
IOPPROC, 8.19
LENGTHF, 8.19
OUTPTB, 8.15-16
OUTPTC/OUTPTS, 8.19-20
OVERLAY, 8.21
PDUMP, 8.21
PLOT, 8.21
Q8QIOU, 8.21-22
RANINT/RANWT/RANRD, 8.22
SAVEF, 8.22-23
EXIT, 3.37
entry points
  END, EXIT, STOP, 3.37
EXIT MODE diagnostics, 8.14-15
EXP, 2.48
EXPON (ULIB), 5.11
EZND (ULIB), 5.24
FACT (ULIB), 5.30
FARG=SAVEF, 3.98-99
Fast Fourier transforms (ULIB), 5.12
FFA (ULIB), 5.12
FFS (ULIB), 5.12
File handling (CM), 7.12-15
FLASH1 (PLOT entry point), 4.41, 4.44-45
FLASH2 (PLOT entry point), 4.42, 4.44-45
FLASH3 (PLOT entry point), 4.42, 4.44-45; see also 4.23
FLASH4 (PLOT entry point), 4.43
Floating-point overflow, 7.2
FLUSH (PLOT entry point), 4.24;
see also 4.22
Fortran library functions, definition
and mode, 2.7-8
FOR2D (ULIB), 5.12
FOURT (ULIB), 5.12
FOUR2 (ULIB), 5.12
FRAME (PLOT entry point), 4.24;
see also 4.22
FRSTPT (PLOT entry point), 4.12;
see also 4.4
Functions
call-by-name, 2.3-6
call-by-value, 2.1-3
Fortran library, definition and
mode, 2.7-8
in-line, 1.1-3
LENGTHF, 3.13
mathematical, see Mathematical
library routines, 2.1-66
SAVEF, 3.98-104
FWA (CM call argument), 7.11
HALFAX (PLOT entry point), 4.33;
see also 4.4, 4.25, 4.35-36
Hardware interrupts
7600, 7.6
6600, 7.2, 7.4
HARMON (ULIB), 5.30
HASEL2 (ULIB), 5.27
HESREL (ULIB), 5.17
HESRSM (ULIB), 5.17
Hessenberg form (ULIB), 5.17
HEVAL (ULIB), 5.16
HIST (ULIB), 5.14
HOURS (ULIB), 5.21
Housesr (ULIB), 5.16
HPCL (ULIB), 5.24
IBAIX, 2.49
IBM Fortran extended character set
conversion, see CDS360, 6.9
IDINT, 2.50
IDIOX, 4.51-53, 5.14
IFENDF (BACKSP entry point), 3.5
IF(EQF,i), 3.5
IF(UNIT,i), 3.12
IMPRUV (ULIB), 5.19
INCGAM (ULIB), 5.11
Indefinite operand, 7.2
In-line functions, 1.1-3
absolute value, 1.2
conversion, 1.2
location, 1.3
modulus, 1.3
positive difference, 1.3
sign, 1.3
INPUTB, 3.43-45
diagnostics, 8.15-16
entry points
INPUTB, 3.43-44
OUTPTB, 3.45
INPUTC, 3.46-47
diagnostics, 8.17-19
Input/output
file handling (CM), 7.12-15
routines (ULIB), 5.15
INPUTS, 3.48-49
diagnostics, 8.17-19
Instruction manipulation (PLOT),
4.41-45
Graphics display (ULIB), 5.13-14;
see also Plotting library
routines, 4.1-54
GRID (PLOT entry point), 4.31;
see also 4.25
GRID terminal, 4.45
GRIDALL (PLOT entry point), 4.34;
see also 4.4, 4.25, 4.35-36
GRIDL (PLOT entry point), 4.31;
see also 4.25, 4.36
Interrupt
  driver, 7.1
  hardware, 7.2, 7.4, 7.6
  software, 7.2-3, 7.5
INTRAN (ULIB), 5.28
INVTX (ULIB), 5.17
IOCHEK (BUFFEI entry point), 3.12
diagnostic, 8.19
IOPROC, 3.50-52
diagnostics, 8.19
  entry points
    BSTAPE, 3.50
    IOWAIT, 3.52
    RDTAPE, 3.50-51
    WRTAPE, 3.52
IOWAIT (IOPROC entry point), 3.52

JOBID, 3.52a

KODER, 3.53; see also 3.57, 3.60
KOLMO (ULIB), 5.30
KRAKER, 3.54; see also 3.46, 3.48

LABMOD (PLOT entry point), 4.36-38
LAGUER (ULIB), 5.23
Layout of physical record made by
  RPTOUT, 3.97
LCKSUM (ULIB), 5.21
Least squares (ULIB), 5.5
LENGTHF (BUFFEI entry point), 3.13
diagnostic, 8.19
Lettering, 5.13; see also Plotting
  library routines, 4.1-54
Library routines, see In-line functions,
  Mathematical, Plotting, Process
  and access data, ULIB (special
  purpose), or Utility
Light pen, 4.45
Limits, LOADER table size, 8.26
LINE (PLOT entry point), 4.14;
  see also 4.4
LINLSQ (ULIB), 5.5
LNPRG (ULIB), 5.21
LOADER
diagnostics, 8.24-25
  table size limits, 8.26

Location (in-line functions), 1.3
LSTSQ (ULIB), 5.5
LTPEN (PLOT entry point), 4.45
LWA (CM call argument), 7.11

MACHINE, 3.55
MACRO call arguments (CM), 7.12
MAP (ULIB), 5.13
MAP80 (ULIB), 5.14
Mathematical library routines, 2.1-66
  ACOS, 2.9
  ALOG, 2.10-11
  ALOG10, 2.12
  AMAX/AMIN, 2.13-14
  ASIN, 2.15
  ATAN, 2.16-17
  ATAN2, 2.18-19
  CABS, 2.20
  call-by-name routines, 2.3-6
  call-by-value routines, 2.1-3
  CBAIEX, 2.21
  CCOS, 2.22
  CEXP, 2.23
  CLOG, 2.25
  COS, 2.26
  CSIN, 2.27
  CSQRT, 2.28
  CUBRT, 2.29-30
  DATAN, 2.31
  DATAN2, 2.32-33
  DATEF, 2.24
  DBADEX, 2.34
  DBAIEX, 2.35
  DBAREX, 2.36
  DCOS, 2.37
  definitions, mode and list of names,
    2.7-8
  DEXP, 2.38-39
diagnostics, 8.5
  DLOG, 2.40-41
  DLOG10, 2.42
  DMAI1/DMIN1, 2.43
  DMOD, 2.44
  DSIGN, 2.45
  DSIN, 2.46
  DSQRT, 2.47
  EXP, 2.48
  IBAIEX, 2.49
  IDINT, 2.50
modes, 2.7-8
names, definition, and mode, 2.7-8
RANF, 2.51-52; see also 5.28
entry points
RANF, 2.51
RANGET, 2.52
RANSET, 2.52
RBAIEX, 2.53
RBAREX, 2.54-57
SIN, 2.58-59
SINCL, 2.60
SQRRT, 2.61-62
table of, 2.7-8
TAN, 2.63-64
TANH, 2.65
TIMEF, 2.66
Matrix analysis (ULIB), 5.16-19
determinants, 5.17
eigenvalues and eigenvectors of a matrix, 5.16
Hessenberg form, reduction to, 5.17
matrix inversion, 5.17
matrix package, 5.19
orthogonalization, 5.17
reduction to Hessenberg form, 5.17
solution of system of equations, 5.18
Memory bounds, 7.2
MINSER (ULIB), 5.23
Mode options, 6.5
Modulus (in-line functions), 1.3
Monitor
central, see Central monitor routines, 7.1-15
communications area, 7.1-6
MOVE (ULIB), 5.21
MPAKGE (ULIB), 5.5
MRROMB (ULIB), 5.27
MULROOT (ULIB), 5.23
MXMY (PLOT entry point), 4.28
NDA (ULIB), 5.24
NDATE (ULIB), 5.21
NDTR (ULIB), 5.30
NEWLAB, 3.56
NLEQS (ULIB), 5.23
NOLLS (ULIB), 5.5
Nonalgebraic zeros (ULIB), 5.23
Nonlinear equations, roots of (ULIB), 5.23
NTYPE, 6.5
Numerical analysis routines, see ULIB
subroutines, 5.1-31
ODEIG (ULIB), 5.24
OP (CM call argument), 7.7-8
OPTION (PLOT entry point), 4.25;
see also 4.13, 4.17
Ordinary differential equations (ULIB), 5.24
ORTHO (ULIB), 5.17
Orthogonalization (ULIB), 5.17
OUTPTB (INPUTB entry point), 3.45
diagnostics, 8.15-16
OUTPTC, 3.57-59
diagnostics, 8.19-20
OUTPTS, 3.60
diagnostics, 8.19-20
OVERLAY, 3.61-75
call statements, 3.66
central memory allocation, 3.66
central memory allocation, 3.66
control program, 3.62
diagnostics, 8.21
identification card, 3.65
loader map, 3.74-75
overlay program, 3.62-63
program loading, 3.67-68
sample program, 3.69-73
segment program, 3.64
storage, 3.66
Parity errors, 6.2, 6.4-5, 6.7-8
Partial differential equations (ULIB), 5.25
PAUSE and PAUSE n, 3.76
PDUMP, 3.77-79
diagnostic, 8.21
PERIM (PLOT entry point), 4.32;
see also 4.25, 4.35
PERIML (PLOT entry point), 4.32;
see also 4.25, 4.35-36
Peripheral monitor, 7.1
Permanent storage library (PLIB), 3.80-84
PERMUT (ULIB), 5.22
Physical record layout (made by
RPTOUT), 3.97
PLIBCK (PLIBRD entry point), 3.83
PLIBRD, 3.80-84
  check read or write, 3.83
  Cosy deck, 3.84
  entry points
    PLIBCK, 3.83
    PLIBRD, 3.82
    PLIBREW, 3.81
    PLIBWT, 3.82
  file identification, 3.80
  notes, 3.84
  read a file, 3.82
  rewind, 3.81
  write a file, 3.82
PLIBREW (PLIBRD entry point), 3.81
PLIBWT (PLIBRD entry point), 3.82
PLOT, 4.5
  buffer control, 4.22-24
  diagnostics, 8.21
  entry points, 4.2
Plotting library routines, 4.1-54;
  see also Graphic display, 5.13-14
  advance frame, 4.24
  backgrounds, 4.29-40
    AXES, 4.33; see also 4.4
    GRID, 4.31; see also 4.25
    GRIDALL, 4.34; see also 4.4, 4.25,
    4.35-36
    sample program, 4.40
    GRIDL, 4.31; see also 4.25, 4.36
    HALFAX, 4.33; see also 4.4, 4.25,
    4.35-36
    LABMOD, 4.36-38
    PERIM, 4.32; see also 4.25, 4.35
    PERIML, 4.32; see also 4.25, 4.35-36
    sample programs, 4.39-40
    TICK4, 4.35
    TICKS, 4.35
  buffer control, 4.22-24
CALCNT, 4.48-50, 5.13
character(s)
  controls, 4.18
  flags for character options, 4.18,
  4.20, 4.25
  limit per line, 4.17
  set, 4.19
  sizes, 4.18
contour
  two-dimensional field, see CALCNT
  world and USA maps, see SUPMAP
Plotting library routines (continued)
  control, 4.22-28
  DASHLN, 4.26-27
    sample program, 4.27
  FLUSH, 4.24; see also 4.22
  FRAME, 4.24; see also 4.22
  MXMY, 4.28
  OPTION, 4.25; see also 4.13, 4.17
  CRT address, 4.3, 4.6-9
dd80 cathode ray tube plotter, 4.1
diagnostics, 8.21
  flags for character options, 4.18,
  4.20, 4.25
FLASH routines, 4.41-45
  format, instruction, 4.46-47
  graph generation, 4.36-40
  GRID terminal, 4.45
IDIOT, 4.51-53, 5.14
  instruction
    format, 4.46-47
  manipulation, 4.41-45
light pen, 4.45
LOC argument, notes on, 4.21
mapping diagrams, 4.6, 4.9
maximum characters per line, 4.17
miscellaneous, see CALCNT, IDIOT,
  SUPMAP, and Graphic display
notes on
  LOC argument, 4.21
  X,Y and MX,MY arguments, 4.3-4
other useful routines, 4.48-54;
  see also Graphic display
CALCNT, 4.48-50, 5.13
IDIOT, 4.51-53, 5.14
SUPMAP, 4.54, 5.14
PLOT, 4.5
  buffer control, 4.22-24
  diagnostics, 8.21
  entry points, 4.2
plot instruction manipulation, 4.41-45
FLASH1, 4.41
FLASH2, 4.42
FLASH3, 4.42; see also 4.23
FLASH4, 4.43
  instruction format, 4.46-47
LTPEN, 4.45
  sample program, 4.44-45
Plotting library routines (continued)
plotting, 4.12-15
CURVE, 4.14; see also 4.4
FRSTPT, 4.12; see also 4.4
LINE, 4.14; see also 4.4
POINT, 4.13; see also 4.4
PSYM, 4.13; see also 4.4, 4.17, 4.25
sample program, 4.15
VECTOR, 4.12; see also 4.4
scaling, 4.6-11
PORGN, 4.11; see also 4.3-4, 4.7
PSCALE, 4.10; see also 4.3, 4.7
SET, 4.7-9; see also 4.3, 4.31,
   4.33-34, 4.36-38
single subroutine call, see IDIOT
SUPMAP, 4.54, 5.14
tabular format, 4.47
tick mark(s), 4.32; see also 4.29
length, 4.35; see also 4.29
writing, 4.16-21
PWRT, 4.16-21; see also 4.4,
   4.25-26, 5.13
internal flags, 4.18, 4.20
notes on LOC argument, 4.21
X,Y and MX,MY arguments, notes on,
   4.3-4
POINT (PLOT entry point), 4.13;
   see also 4.4
PORGN (PLOT entry point), 4.11;
   see also 4.3-4, 4.7
Positive difference (in-line functions),
   1.3
POWERX (ULIB), 5.30
PRINT n, 3.57
Process and access data routines, 6.1-10
copy output, 6.7
data control cards, 6.4-6
control words, 6.6
mode and type options, 6.5
data handling routines, 6.9-10
tape copy and edit, 6.1-8
sample program, 6.7-8
USER subroutines, 6.2-3
Program interrupt, 7.2
floating-point overflow, 7.2
indefinite operand, 7.2
memory bounds, 7.2
Program samples, see Sample programs
PROOT (ULIB), 5.3
PSARITH (ULIB), 5.22
PSCALE (PLOT entry point), 4.10;
   see also 4.3, 4.7
PSPLIN (ULIB), 5.6
PSYM (PLOT entry point), 4.13;
   see also 4.4, 4.17, 4.25
PUNCH n, 3.57
PWRT (PLOT entry point), 4.16-21;
   see also 4.4, 4.25-26, 5.13
internal flags, 4.18, 4.20
PWRX (ULIB), 5.13
Q8QERR, 2.6, 8.2
Q8QIOU diagnostics, 8.21-22
Q8QRSD, 2.6
Quadrature (ULIB), 5.26-27
QUARTC (ULIB), 5.3
QUARTD (ULIB), 5.4
RANCK (RANRD entry point), 3.90
Random file
read, 3.6-7
status check, 3.8
write, 3.7
RANF, 2.51-52; see also 5.28
entry points
RANF, 2.51
RANGET, 2.52
RANSET, 2.52
RANF2 (ULIB), 5.28
RANGET (RANF entry point), 2.52
RANIN (RANRD entry point), 3.90
RANINT (RANRD entry point), 3.85-86
RANOUT (RANRD entry point), 3.89
RANRD, 3.85-92
diagnostics, 8.22
entry points
RANCK, 3.90
RANIN, 3.90
RANINT, 3.85-86
RANOUT, 3.89
RANRD, 3.88
RANWT, 3.87
rules, 3.88
sample program, 3.91-92
RANSET (RANF entry point), 2.52
RANWT (RANRD entry point), 3.87
RATION (ULIB), 5.6
Rational approximation (ULIB), 5.6
RBAIEX, 2.53
RBAREX, 2.54-57
RDTAPE (IOPROC entry point), 3.50-51, 6.10
READ INPUT TAPE, 3.46-47
READ TAPE, 3.43
READ(n,FMT)L, 3.46-47
Read-only library (ULIB), 3.108-110, 5.1-31
REALQR (ULIB), 5.16
RECIN/RECOT (ULIB), 5.15
Registers, central monitor, 2.1-5, 7.7
Relative address, 8.1
RENO (ULIB), 5.22
REPBIN, 6.10
Restart deck, 3.100-101
RETURN (CM call argument), 7.7
Return jump, 7.1
REWIND i command, 3.5
REWINDM (BACKSP entry point), 3.5
RFB (CM OP code), 7.12
RFC (CM OP code), 7.12
RICH (ULIB), 5.22
RNDEV (ULIB), 5.28
RNKUT (ULIB), 5.24
ROMB (ULIB), 5.26
Roots
of a polynomial (ULIB), 5.3-4
of nonlinear equations (ULIB), 5.23
RPOLY (ULIB), 5.4
RPTIN/RPTOUT, 3.93-97, 6.10
diagnostics, 3.95, 3.97
entry points
   RPTIN, 3.94-95
   RPTOUT, 3.96-97
physical record layout for RPTOUT, 3.97
RSYMQR (ULIB), 5.16
RTSRH (ULIB), 5.23

Sample programs
   backgrounds (PLOT), 4.39-40
   BUFRD, 3.21-24
control cards with USER subroutines, 6.7-8
   DASHLN (PLOT), 4.27
   DEBUG, 3.30-34
   execution diagnostics with *TRAP, 8.3-4
   FLASH calls, 4.44-45
   GRIDALL (PLOT), 4.40
   OVERLAY, 3.69-73
   plotting, 4.15
   RANRD, 3.91-92
   7600 hardware interrupt, 7.6
   7600 software interrupt, 7.5
   6600 hardware interrupt, 7.4
   6600 software interrupt, 7.3
tape copy and edit, 6.7-8
trace following trace dump, 8.3-4
USER subroutines, 6.7-8
SAVEF (SAVE function), 3.98-104;
   see also 3.88
diagnostics, 8.22-23
   returns, 3.104
   rules, 3.100-103
   sense switches, 3.98-99, 3.104
SBITS, 6.9
SBYTE (BYTE entry point), 3.40
SBYTES (BYTE entry point), 3.42
Scaling (PLOT), 4.6-11
SCHMIT (ULIB), 5.11
SEARCH (ULIB), 5.8
Sense light, 3.107
Sense switch (SSW), 3.107
   PAUSE, 3.76
   SAVEF, 3.98-99, 3.104
SET (PLOT entry point), 4.7-9; see
   also 4.3, 4.31, 4.33-34, 4.36-38
7600 cell 0, 7.5-6
Sign (in-line functions), 1.3
SIMPNE (ULIB), 5.26
SIMPSON (ULIB), 5.26
SIN, 2.58-59
SINGRT (ULIB), 5.6
SINTRP (ULIB), 5.6
SIPMIN (ULIB), 5.23
6600 cell 0, 7.2-4
SLT (SSW entry point), 3.107
SMIRN (ULIB), 5.31
Snap cards (DEBUG), 3.27-29
SNGL, 2.60
Software interrupt
   7600, 7.5
   6600, 7.2-3
SOLIDS (ULIB), 5.14
SOLVE (ULIB), 5.19
SORT (ascending order), 3.105-106
SORTTB, 6.10
SORTUU, 6.10
SPAL (ULIB), 5.31
Special purpose subroutines, see ULIB
   subroutines, 5.1-31
SPI (ULIB), 5.26
SPLINE (ULIB), 5.7
Spline fitting (ULIB), 5.6-7
Spline package (ULIB), 5.7
SPLINT (ULIB), 5.7
SPLVN (ULIB), 5.7
SQRT, 2.61-62
SSW, 3.107
entry points
SLT, SSW, 3.107
STATUS (CM call argument), 7.9-11
STDQR (ULIB), 5.16
STOP (EXIT entry point), 3.37
STRMLN (ULIB), 5.14
Subroutines, utility, see ULIB sub-
routines, 5.1-31, or Utility library subroutines, 3.1-111
SUPMAP, 4.54, 5.14

Table size limits, LOADER, 8.26
TAN, 2.63-64
TANH, 2.65
Tape(s)
BACKSP, 3.4-5
BACKSPACE i command, 3.4
backspace one record, 3.4, 3.50
blocking, unblocking, see RPTOUT, 3.96-97, UBLOCK and UZBLOCK, 6.10
check EOF, 3.5
control
cards, 6.4-6
words, 6.6
copy and edit routines, 6.1-8
copy output, 6.7
data control cards, 6.4-6
control words, 6.6
mode and type options, 6.5
sample program, 6.7-8
USER subroutines, 6.2-3
data
control cards, 6.4-6
sample program, 6.7-8
handling routines, 6.9-10
dumps, see TDUMP, 6.10
edit and copy routines, see Tape copy and edit, 6.1-8
EOF
check, 3.5
write, 3.36
Fortran read error recovery, see CORFOR, 6.9
IOPROC, 3.50-52
label, 3.56
mode options, 6.5
NEWLAB, 3.56
parity errors, 6.2, 6.4-8
read error recovery, see CORFOR, 6.9
read record, 3.50-51
reading non-NCAR records, see IOPROC, 3.50-52, UBLOCK and UZBLOCK, 6.10
RECDN/RECOUT, 5.15
REWIND i command, 3.5
RPTIN/RPTOUT, 3.93-97
diagnostics, 3.95
SAVE function, 3.98-104
sorting, see SORTBB and SORTUU, 6.10
tape-to-printer, see TDUMP, 6.10
TAPECY (ULIB), 5.15; see also
Tape copy and edit, 6.1-8
type options, 6.5
unblocking, blocking, see RPTOUT, 3.96-97, UBLOCK and UZBLOCK, 6.10
UNLOAD, 3.111; see also 3.56
USER, 6.2, 6.7-8
USERF, 6.3, 6.7-8
write EOF, 3.36
write record, 3.52
Tape copy and edit, 6.1-8
copy output, 6.7
data control cards, 6.4-6
control words, 6.6
mode and type options, 6.5
sample program, 6.7-8
USER subroutines, 6.2-3
TAPECY (ULIB), 5.15; see also Tape
 copy and edit, 6.1-8
TDUMP, 6.10
Terminate program, 3.37
TICK4 (PLOT entry point), 4.35
TICKS (PLOT entry point), 4.35
TIMEF, 2.66
Trace dump, 8.3-4
TRAP control card, 3.35, 4.17, 8.3
TRAUB (ULIB), 5.4
TRIDI (ULIB), 5.18
TRIDIP (ULIB), 5.18
Trouble on a write, 6.7-8
Type options, 6.5

Table size limits, LOADER, 8.26
TAN, 2.63-64
TANH, 2.65
Tape(s)
BACKSP, 3.4-5
BACKSPACE i command, 3.4
backspace one record, 3.4, 3.50
blocking, unblocking, see RPTOUT, 3.96-97, UBLOCK and UZBLOCK, 6.10
check EOF, 3.5
control
cards, 6.4-6
words, 6.6
copy and edit routines, 6.1-8
copy output, 6.7
data control cards, 6.4-6
control words, 6.6
mode and type options, 6.5
sample program, 6.7-8
USER subroutines, 6.2-3
data
control cards, 6.4-6
sample program, 6.7-8
handling routines, 6.9-10
dumps, see TDUMP, 6.10
eedit and copy routines, see Tape copy and edit, 6.1-8
EOF
check, 3.5
write, 3.36
Fortran read error recovery, see CORFOR, 6.9
UBLOK, 6.10
ULIB, 3.108-110; see also
ULIB subroutines, 5.1-31
check status of read, 3.110
entry points
ULIBCK, 3.110
ULIBRD, 3.110
ULIBREW, 3.109
read a file, 3.110
rewind, 3.109
ULIB subroutines (special purpose),
5.1-31; see also 3.108-110
algebraic zeros, 5.3-4
roots of a polynomial, 5.3-4
approximation and curve fitting, 5.5-8
least squares, 5.5
other, 5.8
rational approximation, 5.6
spline fitting, 5.6-7
classifications, 5.2
contouring, 5.13
determinants, 5.17
eigenvalues and eigenvectors of a
matrix, 5.16
eigenvalues of ordinary differential
equations, 5.24
elementary functions, 5.9-11
fast Fourier transforms, 5.12
multi-dimensional, 5.12
one-dimensional, 5.12
graphic display, 5.13-14; see also
Plotting library routines,
4.1-54
contouring, 5.13
lettering, 5.13
plotting, 5.14
Hessenberg form, reduction to, 5.17
input/output, 5.15
least squares, 5.5
lettering, 5.13
matrix analysis, 5.16-19
determinants, 5.17
eigenvalues and eigenvectors of a
matrix, 5.16
Hessenberg form, reduction to, 5.17
matrix inversion, 5.17
matrix package, 5.19
orthogonalization, 5.17
reduction to Hessenberg form, 5.17
solution of system of equations, 5.18
miscellaneous, 5.20-22
nonalgebraic zeros, 5.23
roots of nonlinear equations, 5.23
ordinary differential equations, 5.24
eigenvalues of ordinary differential
equations, 5.24
solution of system of ordinary
differential equations, 5.24
orthogonalization, 5.17
partial differential equations, 5.25
plotting, 5.14; see also Plotting
library routines, 4.1-54
polynomial, roots of, 5.3-4
quadrature, 5.26-27
multi-dimensional, 5.27
one-dimensional, 5.26
random number generators, 5.28
rational approximation, 5.6
reduction to Hessenberg form, 5.17
roots of
nonlinear equations, 5.23
polynomial, 5.3-4
solution of system of
equations (matrix analysis), 5.18
ordinary differential equations,
5.24
spline fitting, 5.6-7
statistics, 5.29-31
ULIBCK (ULIB entry point), 3.110
ULIBRD (ULIB entry point), 3.110
ULIBREW (ULIB entry point), 3.109
UNIT (CM call argument), 7.9
UNLOAD, 3.111; see also 3.56
USER, 6.2, 6.7-8
User
job identification (JOBID), 3.52a
storage library (PLIB), 3.80
USRF, 6.3, 6.7-8
Utility library subroutines, 3.1-111;
see also ULIB subroutines, 5.1-31
ACGOER, 3.3
diagnostic, 8.6
BACKSP, 3.4-5
diagnostic, 8.6
BRANRD, 3.6-9
diagnostics, 8.6
BUFFE, 3.10-13
diagnostics, 8.6, 8.19
Utility library subroutines (continued)

PLIBRD, 3.80-84
  check read or write, 3.83
  Cosy deck, 3.84
  file identification, 3.80
  notes, 3.84
  read a file, 3.82
  rewind, 3.81
  write a file, 3.82
RANRD, 3.85-92
  diagnostics, 8.22
  rules, 3.88
  sample program, 3.91-92
RPTIN/RPTOUT, 3.93-97
  diagnostics, 3.95, 3.97
  physical record layout for
  RPTOUT, 3.97
SAVEF, 3.98-104; see also 3.88
  diagnostics, 8.22-23
  returns, 3.104
  rules, 3.100-103
  sense switches, 3.98-99, 3.104
SORT, 3.105-106
SSW, 3.107
TAPECY, see Tape copy and edit, 6.1-8
ULIB, 3.108-110; see also ULIB
  subroutines, 5.1-31
  check status of read, 3.110
  read a file, 3.110
  rewind, 3.109
UNLOAD, 3.111; see also 3.56
UZBLOK, 6.10

VECTOR (PLOT entry point), 4.12;
  see also 4.4

WRB (CM OP code), 7.12
WRC (CM OP code), 7.12
WRITE OUTPUT TAPE n, 3.57
WRITE TAPE, 3.45
WRITE(6,n), 3.57
Writing (PLOT), 4.16-21
WRTAPE (IOPROC entry point), 3.52

XYPOIS (ULIB), 5.25
XYZSPLN (ULIB), 5.7; see also 5.6