October 1976

SETLIO

SUBROUTINE DESCRIPTION

Purpose: To assign a file or device to a logical I/O unit.

Location: Resident System

Calling Sequences:

Assembly: CALL SETLIO,(unit,FDname)

FORTRAN: CALL SETLIO(unit,FDname,&rc4)

Parameters:

- unit is the location of the left-justified, 8-character logical I/O unit name (e.g., SCARDS).
- FDname is the location of the file or device name to be assigned. This name must be terminated with a trailing blank.
- rc4 is the statement label to transfer to if the return code of 4 occurs.

Return Codes:

0 Successful return.
4 Error return. An illegal logical I/O unit name was specified.

Description: This subroutine is used to assign a file or device to a logical I/O unit. If there was a previous assignment, the new file or device replaces the previous file or device. That usage of the previous file or device is released. If the FDname parameter is blank, the previous file or device is released and the logical I/O unit is left without an assignment.

This subroutine does not check for the legality of the file or device name specified.

Examples: Assembly:

```
          CALL SETLIO,(UNIT,FDNAME)
          LTR  15,15
          BNE  ERROR
          .
          .
UNIT      DC   CL8'SCARDS '
FDNAME    DC   C'DATAFILE '
```

FORTRAN:

```
CALL SETLIO('SCARDS  ','DATAFILE ',&100)
```

The above two examples call SETLIO to assign the file DATAFILE to the logical I/O unit SCARDS.

Assembly:

```
          LA   10,INPUT     Get addr of input line
          LR   9,10         Save addr of input line
LOOP1     CLI  0(10),C'='   Scan off unit name
          BE   EXIT1
          CLI  0(10),C' '   Error if no equal sign
          BE   ERROR
          LA   10,1(0,10)
          B    LOOP1
EXIT1     LR   8,10         Compute len of unit name
          SR   8,9
          BCTR 8,0
          MVC  UNIT(8),=CL8' '
          EX   8,MVCLIO     Save unit name
          LA   10,1(0,10)   Skip past equal sign
          CALL SETLIO,(UNIT,(10))
          LTR  15,15
          BNE  ERROR
          .
          .
INPUT     DC   C'SCARDS=DATAFILE '
UNIT      DS   CL8
MVCLIO    MVC  UNIT(0),0(9)
```

The above example calls SETLIO after scanning an input string containing a logical I/O unit assignment. GR10 which points to the name of the file DATAFILE is inserted into the parameter list for SETLIO in place of FDname.

October 1976

SETLNR

SUBROUTINE DESCRIPTION

Purpose: To set all or a subset of the line numbers in a line file.

Location: Resident System

Calling Sequences:

Assembly: CALL SETLNR,(unit,first,last,cnt,buffer)

FORTRAN: CALL SETLNR(unit,first,last,cnt,buffer,&rc4,
&rc8,&rc16,&rc20,&rc24,&rc28,&rc32)

Parameters:

unit is the location of either:
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 19), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

first is the location of a fullword containing the internal line number of the first line number to be set.

last is the location of a fullword containing the internal line number of the last line number to be set.

cnt is the location of a fullword containing a count of the number of line numbers in the specified range to be set (used fcr error checking).

buffer is the location of a buffer. The buffer is supplied and set up by the caller. The buffer should be of the form:

bytes 0-3 pointer to next buffer or zero,
bytes 4-7 length of this buffer in bytes (including these 8 bytes),
bytes 8-... internal line numbers to set (4 bytes each).

Return Codes:

0 The line numbers were set successfully.
4 The file does not exist.
8 Hardware error or software inconsistency encountered.

12 Renumber or read/write access not allowed.
16 Locking the file for modification will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of the (shared) file).
24 Inconsistent parameters specified (requested setting will cause duplicate or decreasing line numbers, etc.).
28 The file is not a line file.
32 Buffers exhausted before line-number range was exhausted.

Notes:

If first and last do not correspond to actual line numbers in the file, the next and previous line numbers, respectively, will be used.

In MTS, the internal line number (e.g., 2100) is equal to the external line number (e.g., 2.1) times one thousand.

Examples: Assembly:

```
          CALL GETFST,(UNIT,FSTLNR)
          CALL GETLST,(UNIT,LSTLNR)
          CALL RETLNR,(UNIT,FSTLNR,LSTLNR,CNT,BUFFER)
          CALL RENUMBER,(UNIT,FSTLNR,LSTLNR,BEG,INC)
            ...
          CALL GETFST,(UNIT,FSTLNR)
          CALL GETLST,(UNIT,LSTLNR)
          CALL SETLNR,(UNIT,FSTLNR,LSTLNR,CNT,BUFFER)
            ...
UNIT    DC   F'4'
FSTLNR  DS   F          First line number
LSTLNR  DS   F          Last line number
CNT     DS   F          Count of lines in file
BEG     DC   F'1000'    Renumber starting at 1
INC     DC   F'1000'    In increments if 1
BUFFER  DC   F'0'       The only buffer
        DC   F'808'     This many bytes
        DS   200F       Line numbers go here
```

The above example illustrates how to save a set of line numbers in a file, renumber the file, and then later restore the original line numbers of the file attached to logical I/O unit 4 (assuming the file contains fewer than 200 lines).

```
FORTRAN:    INTEGER*4 UNIT,FSTLNR,LSTLNR,CNT,LNR,$I4(1)
            COMMON /$/ $I4
            DATA UNIT/4/
             ...
            CALL GETFST(UNIT,FSTLNR)
            CALL GETLST(UNIT,LSTLNR)
            CALL CNTLNR(UNIT,FSTLNR,LSTLNR,CNT)
            CALL ARINIT(1,1)
            CALL ARRAY(LNR,4,CNT+2)
            $I4(LNR+1)=0
            $I4(LNR+2)=CNT*4+8
            CALL RETLNR(UNIT,FSTLNR,LSTLNR,CNT,$I4(LNR+1))
              ...
            CALL RENUMB(UNIT,FSTLNR,LSTLNR,1000,1000)
              ...
            CALL GETFST(UNIT,FSTLNR)
            CALL GETLST(UNIT,LSTLNR)
            CALL SETLNR(UNIT,FSTLNR,LSTLNR,CNT,$I4(LNR+1))
              ...
```

The above example illustrates how to remember and reset all of the line numbers of a line file attached to logical I/O unit 4 (using the FORTRAN array management subroutines to dynamically allocate a buffer).

October 1976

SETPFX

SUBROUTINE DESCRIPTION

Purpose: To set the prefix character for the program currently executing. This character is issued during program execution as the first character of every input or output line on a terminal.

Location: Resident System

Calling Sequences:

Assembly: CALL SETPFX,(char,len)

FORTRAN:
```
INTEGER*4 SETPFX,i,len
i = SETPFX(char,len)
```

Parameters:

char is the location of the prefix character.
len is the location of a fullword integer count of the number of characters.

Values Returned:

GR0 contains the previous prefix character, right-justified with leading hexadecimal zeros. For FORTRAN users, the value returned by the integer function call to SETPFX will be the previous prefix character, right-justified.

Restriction: Currently only one prefix character may be used. Thus only the first character at the location specified is used.

Examples: Assembly:
```
          CALL SETPFX,(PCHAR,PLEN)
          ST   0,OCHAR
           .
           .
PCHAR DC   C'?'
PLEN  DC   F'1'
OCHAR DS   CL4
```

The above example calls SETPFX to set the prefix character to "?".

```
FORTRAN:        INTEGER*4 SETPFX, OLD
                OLD = SETPFX('/',1)
```

The above example calls SETPFX to set the prefix character to "/".

October 1976

SIOC

SUBROUTINE DESCRIPTION

Purpose: To perform floating-point, integer, logical, and hexadecimal input/output conversions. The types of conversion and editing available correspond to those associated with the ANS FORTRAN conversion codes D, E, F, G, I, and L and the IBM FORTRAN conversion code Z. In addition, SIOC incorporates a number of optional features such as blank suppression and free-format input and output. SIOC performs one I/O conversion per call and does not perform any actual I/O operations.

Location: Resident System

Alt. Entry: SIOC#

Calling Sequences:

Assembly: CALL SIOC,(buffer,cvarea)

FORTRAN: CALL SIOC(buffer,cvarea,&rc4,&rc8)

Parameters:

buffer is the location of the first character of the input/output buffer. Input conversions never change the contents of the buffer.

cvarea is the location of a doubleword-aligned block of information containing parameters indicating the type of conversion and editing, containing the internal datum, and providing a scratch area for intermediate calculations.

&rc4,&rc8 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 Successful return.

4 The parameters of the external output field are inappropriate and the field has been filled with asterisks (*). The external input field contains an illegal character.

8 One of the input/output parameters specifies an illegal value, or the value of the external input field exceeds the allowable range for the internal representation.

Description: The notation for the cvarea parameters used below is consistent with the FORTRAN format descriptors sPEw.d, sPFw.d, sPGw.d, Iw, Lw, and Zw. For FORTRAN users, the doubleword alignment of cvarea may be most easily accomplished by placing the parameters at the beginning of a COMMON block.

RFP: This fullword integer specifies the position relative to buffer of the external field in the input/output buffer. The first character of buffer corresponds to an RFP of zero. For both input and output conversions, the RFP is updated to correspond to the first character after the external field processed. Restriction: RFP ≥ 0.

W: This fullword integer specifies the number of characters in the external field. Restriction: 255 ≥ W ≥ 1.

D: Nominally, at least, this fullword integer specifies the number of digits to the right of the decimal point. The interpretation of and restrictions on this parameter are dependent on the conversion code.

S: Fullword-integer scale factor. The interpretation of and restrictions on this parameter are dependent on the conversion code.

RF: Fullword-integer replication factor.

CW: This fullword consists of the function byte, the conversion code byte, the datum-length byte, and the input picture byte. The values for these bytes listed below are in hexadecimal.

Function Byte: 1=INPUT, 0=OUTPUT.
Conversion Code Byte: E=0E, F=1C, G=1E, I=10, L=06, Z=02.
Datum-Length Byte: Number of bytes in the internal datum. Restriction: 8 ≥ datum-length (E,F,G,I, L), or 8 ≥ datum-length ≥ 1 (Z).
Input Picture Byte: The bits of this byte are set during input conversions to record the actual contents of the external field, e.g., sign character, decimal exponent.

V: The internal representation of the datum will or should be left-justified in this doubleword.

WK: This area must supply at least 10 words of scratch space for output conversions, and max(10,W/4+3) words for input conversions.

Input conversions will change only the RFP, RF, the input picture byte, and V; output conversions will change only the RFP and the external field in buffer.

Because the manipulation of the various parameters contained in cvarea is somewhat inconvenient in FORTRAN, the

SIOCP subroutine has been made available for this purpose. The description of the SIOCP subroutine is restricted to information indicating how to set the SIOC parameters.

Relative Field Position - RFP

The RFP parameter can be employed to relieve the calling program of maintaining a buffer pointer. For example, when converting successive values from an input line, the RFP can be initialized to zero for the first call on SIOC and subsequently ignored. This same procedure can be used to formulate an output line, and the final value of RFP will be the length of the line generated.

Replication Factor Processing

In the external field, a replication factor consists of a string of decimal digits terminated by an asterisk (*) and preceding the value in the field, e.g., 5*1.5. An input replication factor will be converted and stored in RF only if (1) bit 1 of the conversion code byte is 1 (hex 40), (2) the portion of the field preceding and following the asterisk is not null, and (3) the value of the digit string preceding the asterisk is in the range [1, 2147483647]. An output replication factor will be generated in the external field only if (1) bit 1 of the conversion code byte is 1 (hex 40), (2) free-format output is in effect, and (3) the value in RF is positive.

Blanks in Numeric Input Fields

Consistent with the ANS FORTRAN standard, all blanks in the external input field are treated as zeros. If bit 3 of the function byte is 1 (hex 10), all blanks in the external field are ignored.

Floating-Point Mapping

All E, F, and G input conversions correctly round the value in the external field to the appropriate internal format; and all E, F, and G output conversions place in the external field the decimal expansion of the internal datum rounded to the number of digits (≤18) necessary to fulfill the field requirements. If bit 4 of the function byte is 1 (hex 08), both the input and output mappings are by truncation instead of rounding.

Direct Conversion

The direct conversion feature is only applicable to output conversions, and is obtained by setting bit 5 of the function byte to 1 and bit 6 to 0 (hex 04). Buffer and the parameters RFP, W, S, and RF are ignored, and the external field is generated in the scratch area WK. The format of the external field depends on the conversion code, the datum-length, and D, i.e., E(D+6).D, I12, L1, or Z(2*datum-length). If D is not in the range [1,18], a default value of 9 or 18 is employed depending on whether the internal datum is a short- or long-operand, respectively. D is not actually changed.

Free-Format

The free-format feature is enabled when bit 6 of the function byte is 1 (hex 02). For input conversions, this forces the delimiter scan and appropriate updating of the RFP after an illegal character has been encountered; the RFP is normally updated by W in this situation. On the other hand, free-format output conversions provide for a datum-dependent, left-justified external field with an optional replication factor and delimiter (,). The parameters W and S are always ignored. Floating-point conversions generate D significant digits and append an exponent only when necessary. If D is not in the range [1,18], a default value of 9 or 18 is employed depending on whether the internal datum is a short- cr long-operand, respectively. D is not actually changed.

Conversion Code Byte

In addition to the settings given earlier, three other bits in this byte may be used to obtain additional services. If bit 1 is 1 (hex 40), replication factor processing is enabled. If bit 2 is 1 (hex 20), a sign will always be generated in E, F, G, and I external output fields; a sign is normally generated only when the datum is negative. If bit 7 is 1 (hex 01), delimiter processing is enabled. For free-format output conversions, delimiter processing places a comma (,) at the end cf the external field. For input conversions, the first occurrence of a delimiter character results in: (1) setting the RFP to correspond to the first character after the delimiter, (2) effectively modifying W to correspond to the number of characters preceding the delimiter, and (3) effectively setting D tc zero. The W and D parameters are not actually changed. If the first character of the external field is a

delimiter, the value of the field is zero. The delimiter characters are: comma (,), semicolon (;), prime ('), and slash (/).

Datum-Length Byte

In conjunction with the conversion code byte, the value of this parameter determines the internal representation as follows:

Conv. Code	Datum-Length	Internal Representation
E,F,G	=8	REAL*8
E,F,G	NOT 8	REAL*4
I	=4	INTEGER*4
I	NOT 4	INTEGER*2
L	=4	LOGICAL*4
L	NOT 4	LOGICAL*1
Z	≤8	datum-length bytes

Input Picture Byte

The bits of this byte are set during input conversions to describe the actual contents of the external field. These bits indicate the presence (1) or absence (0) of the elements listed below:

Bit	Element and Applicable Conversion Codes
0	Floating-point exponent character D (E,F,G).
1	Replication factor (all).
2	Sign character (E,F,G,I,Z).
3	Digits to left of decimal point (E,F,G,I).
4	Decimal point (E,F,G).
5	Digits to right of decimal point (E,F,G). T or F (L).
6	Floating-point exponent (E,F,G). T or F (L). Hexadecimal digits (Z).
7	Delimiter (all).

Error Processing

If an illegal character is found in the external input field, a return code of 4 is given. The relative position of the illegal character with respect to the first character of the external field is placed in the first word of V, and the translation of the illegal character is placed in the second word of V.

Illegal Character	Translation
Decimal digit (0-9)	0
Sign character	1
Delimiter (,;'/)	2
Decimal point	3
Asterisk (*)	3
Hex digit (A-F)	4
None of the above	5

Syntax violations are treated as illegal characters. For example, a decimal point is legal in an F-field, but the second occurrence of a decimal point would be illegal.

When performing output conversions, a return code of 4 is given if the field width is insufficient, if S is not in the range [-D,D+1] in a G-field specification being treated as an E-field specification, if S is not in the range [-D,D+1] in an E-field specification, or if D is not in the range [0,W-1]. The first and second conditions are generally data dependent but can, like the remaining conditions, be of a technical nature.

Illegal parameter values, which cause a return code of 8 with no changes in any SIOC parameters, arise when one or more of the explicit restrictions given in the parameter descriptions above are violated. If a return code of 8 is given for exceeding the range appropriate for the internal representation, the RFP will be correctly updated and RF and V will be indeterminate.

Replication Factor Range	[1,2147483647]
Integer Range	[-2147483648,2147483647]
Floating-Point Range	[.539...E-78,.723...E+76]

Example: The example program below prints the elements of a COMPLEX vector on unit 5. The output lines produced by this program will be of the form

```
"     ±d.ddddddddE±ee +I* ±d.ddddddddE±ee"
```

where, depending on the type of device attached to 5, the initial blank may be removed for use as carriage control.

```
      COMPLEX Z(10)
      INTEGER BUF(10),BL/' '/,BI/' +I*'/
      INTEGER CVA(18)/0,16,8,1,0,Z002E0400,12*0/
      INTEGER*2 LEN/40/
      EQUIVALENCE (DATUM,CVA(7))
      REAL*8 DCVA(9)
      EQUIVALENCE (DCVA(1),CVA(1))
           .
           .
      BUF(1)=BL
      BUF(6)=BI
      DO 10 I=1,10
      CVA(1)=4
      DATUM=REAL(Z(I))
      CALL SIOC(BUF,CVA)
      CVA(1)=24
      DATUM=AIMAG(Z(I))
      CALL SIOC(BUF,CVA)
10    CALL WRITE (BUF,LEN,0,LINE,5)
           .
           .
```

October 1976

October 1976

SIOCP

SUBROUTINE DESCRIPTION

Purpose: To provide an easy method for setting the conversion parameters prior to calling the input/output conversion subroutine SIOC. Most of the SIOC parameters are fullword integers, but the control word is divided into four bytes which cannot be conveniently manipulated by FORTRAN programs. This subroutine provides for the translation of a single FORTRAN format descriptor and associated SIOC modifiers into a form acceptable to SIOC. In the description below, explicit reference is made to various SIOC parameters and features so that familiarity with SIOC would be most helpful.

Location: Resident System

Calling Sequence:

Assembly: CALL SIOCP,(format,cvarea)

FORTRAN: CALL SIOCP(format,cvarea,&rc4)

Parameters:

- format is the location of the first character of the extended format descriptor to be translated. This character string must be terminated by a blank.
- cvarea is the location of a doubleword-aligned block of storage that will be subsequently used in calling SIOC.
- &rc4 is a statement label to transfer to if an error is detected.

Return Codes:

- 0 Successful translation.
- 4 An element of the character string in format could not be deciphered, and the contents of cvarea reflect only the portion of format preceding the erroneous element. One of the input/output parameters (RFP, W, or the datum-length byte) contains an illegal value, i.e., if cvarea is passed to SIOC, a return code of 8 will result.

Description: The scanning of the character string in format is terminated when a blank is encountered or when an element of the string cannot be deciphered. Thus, blanks should not

be embedded in the character strings described below. The character string in format should be of one of the following forms:

```
([Tn,][sP]Dw.d)
([Tn,][sP]Ew.d)
([Tn,][sP]Fw.d)
([Tn,][sP]Gw.d)
([Tn,]Iw)
([Tn,]Lw)
([Tn,]Zw)
```

where the elements enclosed in square brackets ([ ]) are optional; "n", "w", and "d" are unsigned decimal integers; and "s" is an optionally signed decimal integer. The translation process sets the conversion code byte and places "n" in RFP, "w" in W, "d" in D, and "s" in S. The parameters in cvarea are initialized to zero prior to the translation only if the first character of format is a left parenthesis, and only those elements of the parameter area explicitly referenced in the extended format descriptor are modified.

The SIOC modifier names and corresponding functions are:

Name	Function (Conversion Code Byte)
RF	Enable replication factor processing.
S	Enable sign generation in numeric output fields.
D	Enable delimiter processing.

Name	Function (Function Byte)
BLK	Ignore blanks in input fields.
TRUNC	Floating-point mapping by truncation.
DC	Direct conversion.
FF	Free-format.
INPUT	Input conversion.

Name	Function (Datum-Length Byte)
DL=b	Set datum-length byte, $0 \leq b \leq 8$.

These modifier names (preceded by an @) should be appended to the FORTRAN format descriptor. The occurrence of a conversion code (D,E,F,G,I,L,Z) automatically sets the RF, S, and D bits of the conversion code byte to zero, i.e., off. The defaults for the function byte and datum-length byte modifiers depend on the contents of cvarea when SIOCP is called (first character of format not a left parenthesis) or are zero, i.e., rounded output in fixed format (first character of format a left parenthesis). The negatives of these modifiers are not supported.

The translation of the extended format descriptors is extremely permissive, and variations on the syntax delineated above should be used with caution. For example, using the notation = for equivalence,

Ew=Ew.=Ew.0, G.d=G0.d, and F=F0.0.

After the extended format descriptor has been processed, SIOCP checks to insure that RFP, W, and the datum-length byte contain valid data, i.e., data which will not cause SIOC to give a return code of 8.

Example: The example program below converts two REAL*8 values from each input line read through SCARDS, and prints their sum on SPRINT in the form

"(number) ± (unsigned-number) = (number)."

This example illustrates a number of features of both SIOCP and SIOC.

```
      REAL*8 X,Y,SUM,CVA(36),BUFFER(32),BL/' '/
      INTEGER*2 LEN
      INTEGER W(2)
      EQUIVALENCE (CVA(1),W(1))
   10 CALL SCARDS(BUFFER,LEN,0,LINE,&100)
      CALL SIOCP('(E1)@INPUT@BLK@D@DL=8 ',CVA,&200)
      W(2)=LEN
      CALL SIOC(BUFFER,CVA,&200,&200)
      X=CVA(4)
      W(2)=LEN-W(1)
      IF (W(2).LE.0) GO TO 200
      CALL SIOC(BUFFER,CVA,&200,&200)
      Y=CVA(4)
      SUM=X+Y
      BUFFER(1)=BL
      CALL SIOCP('(T1,E)@FF@DL=8 ',CVA,&200)
      CVA(4)=X
      CALL SIOC(BUFFER,CVA)
      CALL SIOCP('@S ',CVA,&200)
      CVA(4)=Y
      CALL SIOC(BUFFER,CVA)
      CALL IMVC(3,BUFFER,W(1),' = ',0)
      W(1)=W(1)+3
      CALL SIOCP('E ',CVA,&200)
      CVA(4)=SUM
      CALL SIOC(BUFFER,CVA)
      LEN=W(1)
      CALL SPRINT(BUFFER,LEN,0,LINE)
      GO TO 10
  100 CALL SYSTEM
  200 CALL ERROR
```

```
GO TO 10
END
```

October 1976

SIOERR

SUBROUTINE DESCRIPTION

Purpose: To allow FORTRAN users to regain control when I/O transmission errors that would otherwise be fatal (such as tape I/O errors or exceeding the size of a file) occur during execution.

Location: *LIBRARY

Calling Sequence:

FORTRAN:
```
EXTERNAL subr
CALL SIOERR(subr)
```

Parameters:

subr is the subroutine to transfer to when an I/O error occurs, or zero, in which case, the error exit is disabled.

Description: A call on the subroutine SIOERR sets up an I/O transmission error exit for one error only. When an error occurs and the exit is taken, the intercept is cleared so that another call to SIOERR is necessary to intercept the next I/O transmission error.

If the subroutine subr returns, a return is made to the user's program from the I/O routine with the return code indicating the type of error that occurred. The return code depends upon the type of device in use when the error occurred. See the section "I/O Subroutine Return Codes" in this volume.

Note: SETIOERR is for assembly language (see the description of the subroutine SETIOERR) and SIOERR is for FORTRAN users. There is a difference in the level of indirection between the two subroutines; therefore, SIOERR should not be used by assembly language users.

Many I/O error conditions are detected by the FORTRAN I/O Library before they actually occur, thus allowing the FORTRAN monitor to take corrective action. In these cases, an error exit enabled by a call to SIOERR will not be taken since the FORTRAN monitor will take control before the erroneous operation is attempted. For further details, see the "FORTRAN I/O Library" section in MTS Volume 6.

Example: FORTRAN:

```
EXTERNAL SWITCH
COMMON ISW
     .
     .
ISW=0
CALL SIOERR(SWITCH)
WRITE (8,105) FILEOUT
IF(ISW.EQ.1) GO TO 10
CALL SIOERR(0)
     .
     .
SUBROUTINE SWITCH
COMMON ISW
ISW=1
RETURN
END
```

In this example, SIOERR is called to enable an exit if an I/O error occurs during the processing of the WRITE statement. If an error does occur, the subroutine SWITCH will be called which sets the variable ISW to 1 and returns. The calling program tests the value of ISW and branches to statement 10 if appropriate. SIOERR is called again to disable the exit.

October 1976

SKIP

SUBROUTINE DESCRIPTION

Purpose: To space a magnetic tape or file either forward or backward a specified number of records or files.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL SKIP,(nfiles,nrcds,unit)

FORTRAN: CALL SKIP(nfiles,nrcds,unit,&rc4,&rc8,&rc12)

Parameters:

- nfiles is the location of the number of files to skip (must be zero for files).
- nrcds is the location of the number of records to skip.
- unit is the location of either
  - (a) a fullword-integer FDUB-pointer (as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 19), or
  - (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- rc4,....,rc12 are statement numbers to transfer to if a nonzero return code is encountered.

Return Codes:

0 Successful return.

4 An end-of-file (filemark) was reached during a forward space or backspace record operation. The unit is left positioned immediately after (on forward space) or before (on backspace) the filemark.

8 The load point (beginning of tape) was detected on a backspace operation (tape is left at load point) or the logical end of a labeled tape was detected on a forward space operation (tape is left at the end). This return code cannot occur for files.

12 The unit parameter is illegally specified, the unit is not a magnetic tape or file, an I/O error condition was detected, or nfiles is not zero and the unit is a file.

Description: The tape or file specified by unit will be spaced nfiles first and then nrcds. If a parameter is negative, the unit will be spaced backward the appropriate number of files; if positive, the spacing will be in the forward direction. For files, the nfiles parameter must be zero.

In spacing files, after the operation is complete, the tape will be positioned on the opposite side of the filemark from which it began. That is, on forward space file requests (nfiles > 0), the tape will be forward spaced past the requested number of filemarks and be left positioned immediately after the last one. On backspace file requests (nfiles < 0), the tape will be backspaced past the requested number of filemarks and be left positioned immediately before the last filemark or at the load point. A separate forward space file request will be necessary to position the tape at the beginning of the next file.

If any spacing operation results in a nonzero return code from the MTS I/O routines, the SKIP subroutine will return before completing all requested file and record skips. This can occur if a tape is backspaced to loadpoint (return code 8), forward spaced to the logical end of a labeled tape (return code 8), or if a backspace record or forward space record request passes over a filemark (return code 4). In addition, a return code of 12 is given for an illegal unit, a unit which is not assigned to a magnetic tape or file, or an I/O error condition.

Examples:

Assembly:

```
          CALL SKIP,(NF,NR,UNIT)
            .
            .
NF        DC   F'-1'
NR        DC   F'1'
UNIT      DC   F'3'
```

FORTRAN:

```
          CALL SKIP(-1,1,3,&100,&150,&200)
100         .
```

The above two examples will cause the tape assigned to logical I/O unit 3 to be positioned to the beginning of the current file by backspacing past one filemark, then forward spacing over the filemark (by forward spacing one record). If the current file was the first file on the tape, the tape would backspace to loadpoint and a return code of 8 would be issued by the tape routines, causing SKIP to return with the tape positioned at the beginning of the tape. In FORTRAN, this would cause statement 150 in the calling program to be executed. If the current file was not the first file on the tape, SKIP would perform a forward space record after the backspace file. Note that this forward space record will result in a

return code of 4 from SKIP because the forward space record will space over a filemark. This would cause statement 100 in the FORTRAN program to be executed.

```
Assembly:       CALL SKIP,(NF,NR,AFDUB)
                 .
                 .
          NF    DC   F'5'
          NR    DC   F'0'
          AFDUB DS   F    A FDUB-pointer.

FORTRAN:        CALL SKIP(5,0,AFDUB)
```

The above two examples will space the tape specified by AFDUB forward 5 files, or until the logical end of a labeled tape is reached (return code 8).

```
Assembly:       CALL SKIP,(NF,NR,UNIT)
                 .
                 .
          NF    DC   F'0'
          NR    DC   F'10'
          UNIT  DC   C'SCARDS  '

FORTRAN:        CALL SKIP(0,10,'SCARDS  ',&4)
                   .
                   .
          4        .
```

The above two examples will space the tape or file attached to the logical I/O unit SCARDS forward 10 records or until an end-of-file occurs, whichever comes first. To find out which occurred, test the return code for 4. In FORTRAN if the operation terminated due to an end-of-file, statement 4 in the program will be executed. If not, processing will continue with the next statement.

October 1976

October 1976

SORT

SUBROUTINE DESCRIPTION

Purpose: To sort or merge records.

Location: *LIBRARY

Alt. Entry: SORT1

Calling Sequences:

Assembly: CALL SORT,(cstmt[,{unit|vds|num},...])

FORTRAN: CALL SORT(cstmt[,{unit|vds|num},...][,&err])

PL/I: CALL PLCALL(SORT,n,cstmt[,{unit|vds|num},...]);

Parameters:

cstmt	is the location of the control statement.
unit	(optional) is the location of a FDUB-pointer (as returned by GETFD), or the location of a fullword-integer logical I/O unit number (0-19).
vds	(optional) is the location of the virtual data set to be processed.
num	(optional) is the location of a positive, nonzero, fullword integer that specifies a numeric value in the control statement.
err	(optional) is the statement label to transfer to if an error (nonzero return code) is detected by the subroutine.
n	is the number of arguments (FIXED BINARY(31)) to be passed to the subroutine.

Return Codes:

0	Successful return.
4	An error has occurred and the subroutine has issued diagnostics via the logical I/O unit SERCOM.

Description: See the section "The SORT Utility Program" in MTS Volume 5.

Summary of the Control Statement

Prototype:

```
[COPY|[[ SORT|MERGE ][ =[[ type ],[ aspect ],[ location ],[ length ], ]...
                [ type ][ ,[ aspect ][ ,[ location ][ ,[ length ]]]]]]]
[[ DS=delimiter[ string ]delimiter▪ ]...DS=delimiter[ string ]delimiter]
[ INPUT[ =[[ name ],[ structure ],[ record length ],[ block length ], ]...
                [ name ][ ,[ structure ][ ,[ record length ][ ,[ block length ]]]]]]
[ OUTPUT[ =[[ name ],[ structure ],[ record length ],[ block length ], ]...
                [ name ][ ,[ structure ][ ,[ record length ][ ,[ block length ]]]]]]
[[optional parameter, ]...optional parameter]
```

Collating fields:

TYPE	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
alignment	AL	no	1 - 4095
binary	BI	no	1 - 256
bit	BT	no	1 - 255 (mask)
call	CA	-	1 - 4095
character	CH	no	1 - 256
defined sequence	DS(i)	no	1 - 256
fixed-point	FI	yes	1 - 260
floating-point	FL	yes	2 - 16
length	LE	-	-
packed decimal	PD	yes	1 - 16
sequence	SE	-	-
signed decimal	SD	yes	2 - 16
zoned decimal	ZD	yes	1 - 15

Record structures:

CODE	RECORD STRUCTURE
U	undefined length
F	fixed length
V	variable length
VS	variable length; spanned
FB	fixed length; blocked
VB	variable length; blocked
VBS	variable length; blocked; spanned
FBS	fixed length; blocked; standard

Optional parameters:

```
CHK     (exit check facility)
DEC     (delete comments)
END     (terminate the control statement)
LIO     (list data set characteristics)
{REC|MNR}=x     (number of records)
RES=x   (restart)
SIG     (sign off on error)
TPS[=[x]]       (tape-merge sort facility)
```

October 1976

SORT2, SORT3

SUBROUTINE DESCRIPTION

Purpose: To sort arrays.

Location: *LIBRARY

Calling Sequences:

```
Assembly: CALL SORT2,(cstmt,loc1,loc2,len[,num]...)
          CALL SORT3,(cstmt,loc1,loc2,len1,loc3,len2
                     [,num]...)

FORTRAN:  CALL SORT2(cstmt,loc1,loc2,len[,num]...[,&err])
          CALL SORT3(cstmt,loc1,loc2,len1,loc3,len2
                     [,num]...[,&err])

PL/I:     CALL PLCALL(SORT2,n,cstmt,ADDR(loc1),
                      ADDR(loc2),ADDR(len)[,num]...);
          CALL PLCALL(SORT3,n,cstmt,ADDR(loc1),
                      ADDR(loc2),ADDR(len1),ADDR(loc3),
                      ADDR(len2)[,num]...);
```

Parameters:

cstmt	is the location of the control statement.
loc1	is the location of the first element of the data set or array to be sorted.
loc2	is the location of the last element of the data set or array to be sorted.
len	is the location of the fullword integer length of each element in the data set to be sorted. The value of len may range between 1 and 256 bytes.
num	(optional) is the location of a positive, nonzero, fullword integer that specifies a numeric value in the control statement.
loc3	is the location of the first element in the tag data set or array.
len2	is the location of the fullword integer length of each element of the tag data set. The value of len2 may range between 1 and 256 bytes.
err	(optional) is the statement label to transfer to if an error (nonzero return code) is detected by the subroutine.
n	is the number of arguments (FIXED BINARY(31)) to be passed to the subroutine.

Return Codes:

0 Successful return.
4 An error has occurred and the subroutine has issued diagnostics via the logical I/O unit SERCOM.

Description: See the section "The SORT Utility Program" in MTS Volume 5.

Summary of the Control Statement

Prototype:

```
[[SORT][=[[type],[aspect],[location],[length],]...
              [type][,[aspect][,[location][,[length]]]]]]
[[DS=delimiter[string]delimiter■]...DS=delimiter[string]delimiter]
[[optional parameter,]...optional parameter]
```

Collating fields:

TYPE	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
alignment	AL	no	1 - 4095
binary	BI	no	1 - 256
bit	BT	no	1 - 255 (mask)
call	CA	-	1 - 4095
character	CH	no	1 - 256
defined sequence	DS(1)	no	1 - 256
fixed-point	FI	yes	1 - 260
floating-point	FL	yes	2 - 16
packed decimal	PD	yes	1 - 16
signed decimal	SD	yes	2 - 16
zoned decimal	ZD	yes	1 - 15

Optional parameters:

DEC (delete comments)
END (terminate the control statement)

October 1976

SPELLCHK

Purpose: To determine if a word is a possible misspelling of a another word.

Location: Resident System

Alt. Entry: SPELCK

Calling Sequences:

Assembly: CALL SPELLCHK,(goodwd,testwd,goodl,testl)

FORTRAN: i=SPELCK(goodwd,testwd,goodl,testl)

Parameters:

goodwd is the location of the word that is kncwn to be correctly spelled.

testwd is the location of the word that is to be compared against goodwd.

goodl is the location of a fullword integer (INTEGER*4) giving the length of goodwd. The length must be between 1 and 32 (inclusive).

testl is the location of a fullword integer (INTEGER*4) giving the length of testwd. The length must be between 1 and 32 (inclusive) and must not differ from goodl by more than 1.

Values Returned:

GR0 contains the value 1 if testwd is a possible misspelling of goodwd or the value -1 if testwd and goodwd are identical; otherwise, GR0 contains the value 0. For FORTRAN calls, this value is returned as a function value in i (i may be treated either as an INTEGER or LOGICAL value, of any length).

Return Codes:

0 Successful return (GR0 is set as above).
4 Error return (error in goodl or testl parameters; GR0 is set to 0).

Description: This subroutine uses a slight variation of the spelling correction algorithm presented by H. L. Morgan in "Spelling Correction in Systems Programs," Communications of the ACM, Vol 13, No. 2 (February 1970).

The algorithm will detect spelling errors consisting of:

(1) two letters transposed,
(2) one letter omitted,
(3) one letter inserted, or
(4) one letter erroneous.

If goodwd and testwd are identical, the subroutine will return the value of 0 in GR0 indicating that testwd is not a misspelling of goodwd.

October 1976

SPIE

SUBROUTINE DESCRIPTION

Purpose: To specify the address of an interruption exit routine and to specify the program interrupt types that are to cause the exit routine to be given control[1].

Location: *LIBRARY

Calling Sequences:

Assembly:
```
LA    1,pica
CALL SPIE
```

Note: This subroutine is normally called by using the SPIE macro. See the SPIE macro description in MTS Volume 14.

Parameters:

pica is the location of a 6-byte region containing the program interrupt control area. The first byte contains the PSW program mask bits that are to be enabled. These are given as:

```
Bits 0-3:  Zero
Bit    4:  Fixed-point overflow
       5:  Decimal overflow
       6:  Exponent underflow
       7:  Significance
```

The next three bytes contain the address of the exit routine to be given control after a program interrupt of the type specified in the interruption mask. The last two bytes contain the interruption mask for the program interrupt types to cause control to be given to the exit routine. Each bit corresponds to a program exception type. These are:

---

[1] OS/360 System Supervisor Services and Macro Instructions, form GC28-6646.

Bit	0:	Zero
	1:	Operation
	2:	Privileged operation
	3:	Execute
	4:	Protection
	5:	Addressing
	6:	Specification
	7:	Data
	8:	Fixed-point overflow
	9:	Fixed-point divide
	10:	Decimal overflow
	11:	Decimal divide
	12:	Exponent overflow
	13:	Exponent underflow
	14:	Significance
	15:	Floating-point divide

If the user wishes to specify a type of program interrupt for which the interruption has been disabled, he must enable the interruption by setting the corresponding bit in the first byte of program mask bits.

Note: A call on SPIE with GR1 containing zero cancels the effect of the previous call.

Value Returned:

GR1 contains the address of the previous PICA. If there is no previous PICA from a previous call on SPIE, a zero is returned.

Description: When a program begins execution, all program interrupts that can be disabled are disabled, and a standard program interrupt exit routine is provided. This program interrupt exit routine is given control when any program interruptions occur. By calling the SPIE (Set Program Interruption Exit) subroutine, the user can specify his own program interrupt exit routines to be given control when a particular type(s) of program interruption occurs.

After the SPIE subroutine has been called by the user's program, his exit routine receives control for all interruptions that have been specified by the interruption mask. For other interruptions, the normal program interruption exit routine is given control. Each succeeding call to the SPIE subroutine overrides the specifications given in the previous call.

The SPIE subroutine records the location of the program interrupt control area (PICA). The PICA contains the new program mask for the interruption types that can be disabled, the address of the exit routine, and an inter-

ruption mask for the interrupt types to cause control to be given to the exit routine. A program that issues a call to SPIE must eventually restore the PICA to the one that was effective when control was received. If there was no previous call to SPIE, restoring the PICA is equivalent to cancelling the current SPIE call and returning to normal interrupt processing. When the SPIE subroutine is called, the subroutine returns the address of the previous PICA in GR1. If there was no previous PICA, then a zero is returned in GR1.

With the first call to the SPIE subroutine, a 32-byte program interruption element (PIE) is created in the subroutine. This program interruption element is used each time a call is made to SPIE. The PIE contains the following information:

```
Word    1:  Current PICA address.
Words 2-3:  Old Program Status Word.
Words 4-8:  GRs 14, 15, 0, 1, and 2.
```

The PICA address in the PIE is the address of the PICA used in the last call to SPIE. When control is passed to the exit routine indicated in the PICA, the old PSW contains the interruption code in bits 16-31; these bits can be tested to determine the cause of the program interruption. The contents of GRs 14, 15, 0, 1, and 2 at the time of interruption are stored by SPIE in the PIE as indicated. When control is passed to the exit routine, the register contents are as follows:

```
GR     0:  Internal control information.
GR     1:  Address of the PIE.
GRs 2-13:  Same as when the program interrupt
           occurred. The exit routine must not use
           GR13 as a save area pointer.
GR    14:  Return address (to the SPIE subroutine).
GR    15:  Address of the exit routine.
```

The exit routine must return control to SPIE by using the address in GR14. SPIE restores GRs 14, 15, 0, 1, and 2 from the PIE after control is returned but does not restore the contents of GRs 3-13. If a program interrupt occurs when the exit routine is in control, normal interruption processing occurs.

Example: This example specifies an exit routine called FIXUP that is to be given control if a fixed-point overflow occurs. The address returned in GR1 is stored in HOLD. This is zero for the first call on SPIE. At the end of the program, the call second call on SPIE disables the user program interrupt processing.

```
        LA    1,PICA
        CALL  SPIE
        ST    1,HOLD
         .
         .
        L     1,HOLD
        CALL  SPIE
         .
         .
HOLD    DS    F
PICA    DC    B'00001000'   Program mask bits
        DC    AL3(FIXUP)    Exit routine address
        DC    X'0080'       Interruption mask
```

The same example using the SPIE macro.

```
        SPIE  FIXUP,(8)
        ST    1,HOLD
         .
         .
        L     5,HOLD
        SPIE  MF=(E,(5))
HOLD    DS    F
```

October 1976

SPRINT

SUBROUTINE DESCRIPTION

Purpose: To write an output record on the logical I/O unit SPRINT.

Location: Resident System

Alt. Entry: SPRINT#

Calling Sequences:

Assembly: CALL SPRINT,(reg,len,mod,lnum)

FORTRAN: CALL SPRINT(reg,len,mod,lnum,&rc4,...)

Parameters:

reg is the location of the virtual memory region from which data is to be transmitted.

len is the location of a halfword (INTEGER*2) integer giving the number of bytes to be transmitted.

mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.

lnum (optional) is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.

rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

0 Successful return.
4 Output device is full.
>4 See the "I/O Subroutine Return Codes" description in this volume.

Description: The subroutine writes a record of length len (in bytes) from the region specified by reg on the logical I/O unit SPRINT. The parameter lnum is needed only if the mod parameter or the FDname specifies either INDEXED or PEEL

(RETURNLINE#). If INDEXED is specified, the line number to be written is specified in lnum. If PEEL is specified, the line number of the record written is returned in lnum.

The default FDname for SPRINT is *SINK*.

There is a macro SPRINT in the system macro library for generating the calling sequence to this subroutine. See the macro description for SPRINT in MTS Volume 14.

Examples: The example below, given in assembly language and FORTRAN, calls SPRINT specifying an output region of 80 bytes. No modifier specification is made in the subroutine call.

Assembly:

```
          CALL SPRINT,(REG,LEN,MOD)
             .
             .
REG       DS   CL80
MOD       DC   F'0'
LEN       DC   H'80'

          or

          SPRINT REG        Subr. call using macro
```

FORTRAN:

```
      INTEGER REG(20),LEN*2/80/
        .
        .
      CALL SPRINT(REG,LEN,0)
```

October 1976

SPUNCH

SUBROUTINE DESCRIPTION

Purpose: To write an output record on the logical I/O unit SPUNCH.

Location: Resident System

Alt. Entry: SPUNCH#

Calling Sequences:

Assembly: CALL SPUNCH,(reg,len,mod,lnum)

FORTRAN: CALL SPUNCH(reg,len,mod,lnum,&rc4,...)

Parameters:

reg is the location of the virtual memory region from which data is to be transmitted.

len is the location of a halfword (INTEGER*2) integer giving the number of bytes to be transmitted.

mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.

lnum (optional) is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.

rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

0 Successful return.
4 Output device is full.
>4 See the "I/O Subroutine Return Codes" description in this volume.

Description: The subroutine writes a record of length len (in bytes) from the region specified by reg on the logical I/O unit SPUNCH. The parameter lnum is needed only if the mod parameter or the FDname specifies either INDEXED or PEEL

(RETURNLINE#). If INDEXED is specified, then the line number to be written is specified in lnum. If PEEL is specified, the line number of the record written is returned in lnum.

The default FDname for SPUNCH is *PUNCH* (for batch mode only) if a global card limit was specified on the $SIGNON command. There is no default for conversational mode or for batch mode if no global card limit was specified.

There is a macro SPUNCH in the system macro library for generating the calling sequence to this subroutine. See the macro description for SPUNCH in MTS Volume 14.

Examples: The example below, given in assembly language and FORTRAN, calls SPUNCH specifying an output region of 80 bytes. No modifier specification is made in the subroutine call.

Assembly:

```
         CALL SPUNCH,(REG,LEN,MOD)
            .
            .
REG      DS   CL80
MOD      DC   F'0'
LEN      DC   H'80'

         or

         SPUNCH REG        Subr. call using macro
```

FORTRAN:

```
      INTEGER REG(20),LEN*2/80/
        .
        .
      CALL SPUNCH(REG,LEN,0)
```

October 1976

STARTF

SUBROUTINE DESCRIPTION

Purpose: To execute a program dynamically loaded by the subroutine LOADF.

Location: Resident System

Calling Sequence:

FORTRAN: CALL STARTF(id,par1,par2,...)

Parameters:

id is the location of the fullword integer storage index number of the program that was dynamically loaded by LOADF (the value returned by LOADF), or is the location of an 8-character entry point name, left-justified with trailing blanks.

par1,par2,... (optional) are the parameters to be passed to the program being executed. There may be any number of parameters passed, including none.

Values Returned:

None.

Description: STARTF is used to execute a program loaded by the subroutine LOADF. STARTF should be used whenever the calling program and the program being called are FORTRAN programs or programs which use the FORTRAN I/O library. This is necessary in order to provide the proper I/O environment for both the called program and the calling program on return. In providing this, the I/O library environment is established in accordance with the "merge" bit. If the merge bit is 1, then both the calling and called programs use the same I/O library environment; if the merge bit is 0, then the calling and called programs each use a separate copy of the I/O library environment, thus performing relatively independent I/O operations.

If id is a storage index number, the dynamically loaded program at that storage index number is invoked at the entry point determined by the loader. If id is a symbol, and if the MTS global SYMTAB option is ON, the dynamically loaded program is invoked at the location associated with that symbol in the loader symbol table.

Example:

```
INTEGER*4 PAR1/'ARG1'/,PAR2/'ARG2'/
INTEGER*4 INFO/Z80000000/,SWITCH/Z00000040/
ID = LOADF('FORTOBJ ',INFO,SWITCH,0)
CALL STARTF(ID,PAR1,PAR2)
CALL UNLDF('FORTOBJ ',0,0)
```

This example loads the program in the file FORTOBJ and executes it. The merge bit is set to 1 so that both programs use the same I/O environment.

October 1976

STDDMP

SUBROUTINE DESCRIPTION

Purpose: To dump a region of the user's virtual memory in the MTS standard format. For dumping registers, dumping with mnemonics, and other options, see the SDUMP subroutine description in this volume.

Location: Resident System

Calling Sequences:

Assembly: CALL STDDMP,(switch,outsub,wkarea,first,last)

Parameters:

switch is the location of a fullword of information. The first halfword of switch is taken as the storage index number that will be printed out in the heading line. The remainder of switch is taken as a group of switches as fcllows:

bit 20: (Integer value = 2048) NOLIB
If set, the call will be ignored if LOADINFO declares that the region of storage is part of a library subroutine.

28: (Integer value = 8) DOUBLE SPACE
If this bit is set, the lines of the dump will be double spaced. Otherwise the normal single spacing will occur.

outsub is the location of a subroutine that will be called by STDDMP to "print" a line. This subroutine is assumed to have the same calling sequence as the SPRINT subroutine.

wkarea is the location of a 100-word (fullword aligned) region which STDDMP will use as a work area.

first is the location of the first byte of a core region to be dumped. There are no boundary requirements for this address.

last is the location of the last byte of a core region to be dumped. There are no boundary requirements for this address; however, an address in last which is less than the address in first will cause an error return.

Return Codes:

0 Successful return.
4 Illegal parameters.

Description: This subroutine uses the same calling sequence as the subroutine SDUMP, but only looks at the bits and parameters as specified above in the calling sequence.

For each call, this subroutine "prints" (calls the output subroutine specified in outsub) the following:

(1) Blank line.
(2) Heading giving information about the region of storage. The subroutine LOADINFO is called to obtain the information.
(3) Blank line.
(4) Dump of the region, with 20 (hex) bytes printed per line. To the left of the hexadecimal dump is the actual hex location and the relative (to the first byte of the region) hex location of the first byte of the line; to the right of the dump is the same information printed as characters. Nonprinting characters (bit combinations that do not match the standard 60 character set of printing graphics) are replaced by periods, and an asterisk (*) is placed at each end of the character string to delimit it. The lines "printed" are 133 characters long.

Example: Assembly:

```
          EXTRN SPRINT
          CALL  STDDMP,(SW,SPRINT,WK,FIRST,FIRST+3)
           .
           .
WK        DS    50D
SW        DC    F'0'
FIRST     DC    X'F1F2F3F4'
```

The above example will cause STDDMP to print the hexadecimal string 'F1F2F3F4'.

October 1976

SYSTEM

SUBROUTINE DESCRIPTION

Purpose: To terminate execution successfully.

Location: Resident System

Alt. Entry: SYSTEM#

Calling Sequence:

Assembly: CALL SYSTEM

or

SYSTEM

FORTRAN: CALL SYSTEM

Note: The complete description for using the SYSTEM macro is given in MTS Volume 14.

Description: This subroutine returns control to MTS to terminate execution. The comment "EXECUTION TERMINATED" is printed. Execution terminated in this manner cannot be resumed by a $RESTART command. Calling this subroutine is equivalent to the program doing a normal return (BR 14) from the call that started execution.

All storage acquired for the executing program and all usages of files and devices by the program are released.

October 1976

October 1976

TICALL

SUBROUTINE DESCRIPTION

Purpose: The FORTRAN interface to the MTS timer interrupt subroutines.

Location: *LIBRARY

Calling Sequence:

FORTRAN: aexit=TICALL(code,subr,value)

CALL TICALL(code,subr,value,&rc4,&rc8)

Parameters:

code is the location of a fullword integer which specifies the meaning of the value parameter. The valid choices are

0 value is an 8-byte integer which specifies a time interval in microseconds of task CPU time, relative to the time of the call.

1 value is an 8-byte binary integer which specifies a time interval in micrcseconds of real time, relative to the time of the call.

2 value is an 8-byte binary integer which specifies a time interval in microseconds of task CPU time, relative to the time at signon.

3 value is an 8-byte binary integer which specifies a time interval in microseconds of real time, relative to the time at signon.

4 value is a 4-byte binary integer which specifies a time interval in timer units (13 1/48 microseconds per unit) of task CPU time, relative to the time of the call.

5 value is a 16-byte EBCDIC string giving the time and date at which the interrupt is to occur, in the form HH:MM.SSMM-DD-YY.

subr is the location of the subroutine to be called when the interrupt occurs. It should be a subroutine with no arguments, and should be declared EXTERNAL in the program which

calls TICALL.

value is the location of a 4-, 8-, or 16-byte fullword-aligned region which specifies the time at which the interrupt is to occur, as determined by the code parameter.

aexit will be assigned the location of the exit region used in calling SETIME and TIMNTRP. It is provided so that the user may subsequently call the subroutines RSTIME or GETIME using

```
CALL RSTIME(subr,value,aexit), or
CALL GETIME(subr,value,aexit).
```

If the interrupt has not been set up, because of an undefined code parameter or too many interrupts set up, aexit will be assigned the value zero.

rc4,rc8 is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

0 Successful return
4 Undefined code parameter
8 Too many interrupts set up.

Description: A timer interrupt is set up, to occur at the time specified by the code and value parameter. When the interrupt occurs, the subroutine subr will be called with no arguments. If subr returns, the program will be restarted at the point of the interrupt.

TICALL may be called several times, up to a maximum of 100 times. When an interrupt occurs, further interrupts set up by TICALL will be disabled until the subroutine subr returns, at which time other interrupts will be reenabled if the return code is zero, and will remain disabled if the return code is nonzero.

Example:

```
EXTERNAL TIMOUT
INTEGER ONESEC(2) /0,1000000/,REAL /1/
 .
 .
CALL TICALL(REAL,TIMOUT,ONESEC)
 .
 .
END

SUBROUTINE TIMEOUT(*)
 .
(Process interrupt and reenable interrupts)
```

```
.
RETURN
.
(Disable interrupts)
.
RETURN 1
.
.
END
```

This example calls TICALL to set up a timer interrupt to occur after 1 second of real time from the time of the call to TICALL. When the interrupt is taken, the subroutine TIMEOUT will be called.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

October 1976

TIME

SUBROUTINE DESCRIPTION

Purpose: To allow the user easy access to the elapsed time, CPU time used, time of day, and the date in convenient units.

Location: Resident System

Calling Sequences:

Assembly: CALL TIME,(key,pr,res)

FORTRAN: CALL TIME(key,pr,res)

Parameters:

key is the location of a fullword integer describing what quantities are desired from the subroutine. The available choices are:

0 the CPU, elapsed, supervisor, and problem state times are initialized (see below).
1 the CPU time in milliseconds is returned in res.
2 the elapsed time in milliseconds is returned in res.
3 the CPU time in milliseconds is placed in the first word of res and the elapsed time in milliseconds is placed in the second word of res.
4 the time of day is returned as characters in the form "HH:MM:SS" where "HH:M" is placed in the first word of res and "M:SS" is placed in the second word of res.
5 the date is returned as characters in the form "MMM DD, 19YY" where "MMM " is placed in the first word of res, "DD, " is placed in the second word of res, and "19YY" is placed in the third word of res. If "DD" is less than 10, the leading zero is replaced by a blank.
6 the time of day is placed in the first and second words of res (see key=4) and the date is placed in the third, fourth, and fifth words of res (see key=5).
7 the supervisor state CPU time in seconds multiplied by 300x256 is placed in res.
8 the problem state CPU time in seconds multiplied by 300x256 is placed in res.

9 the supervisor state CPU time (see key=7) is placed in the first word of res and the problem state CPU time (see key=8) is placed in the second word of res.

10 the date is returned as characters in the form "MM-DD-YY", where "MM-D" is placed in the first word of res and "D-YY" is placed in the second word.

11 the time of day is placed in the first and second words of res (see key=4 above) and the date is placed in the third and fourth words of res (see key=10 above).

12 the date is placed in the first and second words of res (see key=10 above) and the time of day is placed in the third and fourth words of res (see key=4 above).

13 the current number of seconds starting with March 1, 1900, 00:00:01 as "1" is placed in res as a 32-bit unsigned integer.

14 the current number of minutes starting with March 1, 1900, 00:00:01 as "1" is placed in res.

15 the CPU time in microseconds is placed in the first and second words of res as a 64-bit integer.

16 the elapsed time in microseconds is placed in the first and second words of res as a 64-bit integer.

17 the CPU time in microseconds (see key=15) is placed in the first and second words of res and the elapsed time in microseconds (see key=16) is placed in the third and fourth words of res.

18 the supervisor state CPU time in microseconds multiplied by 4096 is placed in the first and second words of res as a 64-bit integer.

19 the problem state CPU time in microseconds multiplied by 4096 is placed in the first and second words of res as a 64-bit integer.

20 the supervisor state CPU time (see key=18) is placed in the first and second words of res and the problem state CPU time (see key=19) is placed in the third and fourth words of res.

21 the date is returned as characters in the form "WWW MMM DD/YY " where "WWW ", the day of the week, is placed in the first word of res, "MMM " is placed in the second word of res, and "DD/YY " is placed in the third and fourth words of res.

22 the date (see key=21) is placed in the first four words of res and the time of day (see key=4) is placed in the fifth and sixth words

of res.

23 the current number of microseconds starting with March 1, 1900, 00:00:00.000001 as "1" is placed in the first and second words of res as a 64-bit integer, the date in the form "MM-DD-YY" (see key=10) is placed in the third and fourth words of res, the date in the form "WWW MMM DD/YY " (see key=21) is placed in the fifth through eighth words of res, and the time of day in the form "HH:MM:SS" (see key=4) is placed in the ninth and tenth words of res.

The CPU time and elapsed time are in milliseconds (key=1, 2, and 3) or microseconds (key=15, 16, and 17) relative to a global arbitrary, past origin. The supervisor and problem state CPU times are in timer units relative to a global arbitrary, past origin. For key=7, 8, and 9, one timer unit is 1/(256*300) seconds or about 13.0 microseconds. For key=18, 19, and 20, one timer unit is 1/4,096,000,000 seconds or about 0.244 nanoseconds. Calling TIME with a key=0 resets these time origins locally to the time status at the call on TIME. These time origins are local to the program currently executing; they do not carry over to another separate program execution. TIME must be reinitialized when used with another program execution.

If 1000 is added to the value of a key and the result is the current date or time of day (key=4-6, 10-14, and 21-23), the result is in Greenwich mean time (GMT). If the result is not based on the current date and time, adding 1000 to the value of the key will produce the same results as the original key value.

pr is the location of a fullword integer indicating whether the returned quantities are to be placed in res or printed or both. The choices are:

0 the values are returned as described above.
<0 the values are returned and are also printed on logical I/O unit SPRINT.
>0 the values are only printed on logical I/O unit SPRINT and are not returned. Thus the res argument is not needed.

If pr is 0, the values are returned.

res is the location of a fullword integer variable or vector in which the results are placed.

Values Returned:

FR0 contains the doubleword, real value in seconds (key=1-3, 7-9, 13, 15-20) or minutes (key=14) if the returned value is numeric.
FR2 contains the doubleword, real, second value in seconds if a second returned value is numeric (key=3, 9, 17, 20).

Return Codes:

0 Successful return.
4 Error, usually due to an improper value for key.

Index to key Values:

CPU time	1,3,15,17
Problem state time	8,9,19,20
Supervisor state time	7,9,18,20
Date	
MM-DD-YY	10,11,12,23
MMM DD, 19YY	5,6
WWW MMM DD/YY	21,22,23
Elapsed time	2,3,16,17
Initialization	0
March 1, 1900 base	13,14,23
Time of day	4,6,11,12,22,23

Examples: Assembly:

```
         CALL TIME,(KEY,PR,RES)
          .
          .
KEY      DC    F'6'
PR       DC    F'0'
RES      DS    5F
```

The time of day and date are stored in location RES.

FORTRAN:

```
CALL TIME(5,1)
```

The date is printed on logical I/O unit SPRINT.

```
CALL TIME(0)
  .
CALL TIME(2,-1,TIM)
```

The elapsed time since the call on TIME(0) is printed on SPRINT and stored in location TIM.

October 1976

TIMNTRP

SUBROUTINE DESCRIPTION

Purpose: To enable, disable, or return from timer interrupts set by the SETIME subroutine.

Location: Resident System

Calling Sequences:

Assembly:
```
LM    0,1,=A(exit,region)
CALL TIMNTRP
```

Parameters:

GR0 should contain zero or the location of the exit routine to transfer control to when a timer interrupt occurs.

GR1 should contain the location of a 76-byte exit region for storing pertinent information.

Return Codes:

None

Description: A call on the TIMNTRP subroutine sets up an exit for one timer interrupt only. The calling sequence specifies the location of an exit routine to transfer control to when the next timer interrupt occurs and an exit region for storing information. The timer interrupts themselves are set up by calls to the SETIME subroutine.

TIMNTRP may be called several times with different exit regions and different exit routines specified. Each call on SETIME must also specify the exit region to be used when the interrupt occurs. This "subsetting" capability allows separate parts of large programs to use the timer interrupt facility independently.

If GR0 is zero, timer interrupt exits for the specified exit region are disabled. If, when a timer interrupt occurs, its exit is disabled, the interrupt will remain pending until the next call on TIMNTRP which enables the exit, and the exit will be taken immediately following the call.

When a timer interrupt exit is taken, the exit is disabled, so that further timer interrupts which specify this exit region will remain pending while the current one

is being processed. The exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save area provided). At the time of this call, GR1 will point to the exit region, whose contents will be

Word 1:	the identifier passed to SETIME when the interrupt was set up.
Words 2-3:	the PSW at the time of the interrupt.
Words 4-19:	GR0-GR15 (in that order) at the time of the interrupt.

If the exit routine returns to MTS (BR 14), the user's program will be restarted at the point of the interrupt. The exit will be reenabled if the return code in GR15 is zero; otherwise, the exit will remain disabled until another call on TIMNTRP. The registers must be restored in the standard fashion when the exit routine returns.

For further details, see also the GETIME, RSTIME, and SETIME subroutine descriptions.

Example: Assembly:

```
        LM    0,1,=A(EXIT,REG)
        CALL  TIMNTRP
          .
          .
        SR    0,0
        LA    1,REG
        CALL  TIMNTRP
          .
          .  critical section
          .
        LM    0,1,=A(EXIT,REG)
        CALL  TIMNTRP
          .
          .
        USING EXIT,15
EXIT    STM   14,12,12(13)
          .
          .  process interrupt
          .
        LM    14,12,12(13)
        SR    15,15
        BR    14
REG     DS    19F
```

In this example, a timer interrupt exit is enabled, some computing is done, it is disabled as the program enters a critical section, and it is then reenabled. The exit routine saves the registers, processes the interrupt, restores the registers, and returns, reenabling the exit.

October 1976

TRACER

SUBROUTINE DESCRIPTION

Purpose: To provide conversational error processing for error conditions detected by the elementary function subroutines such as SQRT and EXP, and to provide program and attention interrupt processing.

Location: Resident System

Alt. Entry: TRACER#

Calling Sequences: (to invoke error processing)

Assembly: CALL TRACER,(msg)

FORTRAN: CALL TRACER(msg)

Parameters:

msg is the location of a message given either as a halfword length followed by the text of the message, or as a delimited string, i.e., if the first byte of the parameter is a graphic character, the message is taken to consist of all characters following this character, up to, but not including, the next occurrence of this same character, e.g., '/A MESSAGE/'.

Calling Sequences: (to enable interrupt processing)

Assembly: CALL TRACER,(-1,msk,ima)

FORTRAN: CALL TRACER(-1,msk,ima)

Parameters:

msk (optional) is the location of a fullword integer interrupt mask specifying the type of processing which is desired for attention interrupts and each of the fifteen program interrupts. With the usual left to right numbering, bits 0 and 1 control attention interrupt processing, and bits 2*n and 2*n+1 control the processing of program interrupt n, n=1,...,15. These two bit masks function as follows:

00	0	Standard system processing
01	1	Unused, same as 00

10 2 Comment and resume processing
11 3 Standard TRACER processing

If this argument is omitted, the value of X'FFFF3FF3' is used.

ima (optional) is the location of a character string to provide an alternative set of messages to be displayed when an interrupt occurs. The first character of the string is used as a delimiter to separate the set of sixteen messages. The delimiter character is arbitrary. The first delimited string corresponds to the attention interrupt message, while the nth delimited string corresponds to program interrupt n-1. The final delimited string must be terminated by the delimiter character, e.g.,

/ATTN/1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/

Return Codes:

None.

Description: The TRACER program is built around a traceback facility, and is capable of supplying information on the current program status and of resuming execution at any entry or return point in the existing linkage chain. The trace facility assumes the standard OS (I) S-type calling convention and that each program has saved appropriately all of the general registers. Commands for displaying and altering floating-point, integer, character, and hexadecimal data are available.

When called, TRACER attempts to ascertain pertinent information concerning its calling program, e.g., name and arguments. Proceeding backwards via the save area chain, it then attempts to discover the same information about the program which called TRACER's caller, etc. Under most circumstances, the traceback will terminate when TRACER locates the information associated with the invocation of the main program by the system.

For example, suppose MAIN calls a subroutine named SUB, which calls a function named FCN, which happens to call SQRT with an argument of -1. TRACER will generally be able to discover:

(1) The names of the programs in the linkage chain, i.e., TRACER SQRT FCN SUB MAIN =SYSTEM
(2) the current values of the arguments passed between each program,
(3) the status of the general registers when each call occurred, and

(4) the current status of the floating-point registers.

The logical I/O unit GUSER is used as the source for TRACER commands. Two prompting characters are used: ":" is used to request the next command when the previous command was successfully executed, and "?" is used to request the next command when the previous command could not be executed.

The logical I/O unit SERCOM is used for all output from TRACER. For batch users, the commands read from GUSER are echoed on SERCOM. Carriage control is always off and all output lines contain at most 71 characters.

The following paragraphs give a brief summary of the TRACER command language facility.

The TRACE command may be used to obtain information concerning the current linkage chain. The names of these programs may be obtained by "TRACE@N"; the save area, parameter list, and entry point addresses by "TRACE@A"; and, the caller and arguments by "TRACE@P". Since mode information for arguments is not available, the first eight bytes at each argument address are displayed in hex.

The CALL and RETURN commands may be used to resume execution. For example, "CALL SUB" would cause execution to resume in the program SUB as if it had just been called by MAIN; while, "RETURN SUB" would cause execution to resume in MAIN immediately after the point SUB was called.

The DISPLAY and ALTER commands may be used like the corresponding system commands, e.g., "DIS 500260", "DIS FR4". Since there is a set of general registers for each program in the linkage chain, the usual general register designators should be qualified, e.g., "DIS MAIN GR5" refers to general register 5 at the time MAIN called SUB. The "DIS GRS" command displays the general registers for each program in the linkage chain. A relocation factor facility is available, but is generally set automatically, and is applied only to addresses below 100000. For example, "DIS@I MAIN+EC(5)" sets the relocation factor to MAIN, and displays the 5-th element of the integer vector assigned to relative location EC in MAIN. Arguments to the subroutines in the linkage chain are referenced by their relative position in the parameter list, e.g., "DIS@E SUB(1)" interprets the first argument to SUB as a REAL*4 variable; "DIS@D3 SUB(3)" displays the first three elements of the REAL*8

vector passed as the third argument. Note that an area is not displayed by using the ellipsis "...", but rather by giving the number of elements desired.

The CONTINUE and TRAP commands are associated with the interrupt processing facility. For example, "TRAP FPUN=0" would cause subsequent floating-point underflows to be ignored. Note that if TRACER had never been called to request control of interrupts, this would automatically occur when the TRAP command is given. The CONTINUE command causes execution to resume with the instruction following the one that produced the interrupt.

Although TRACER was designed primarily as a conversational program, it contains many facilities of use in batch mode.

For a complete description of TRACER, see Computing Center Memo 218.

October 1976

TRUNC

SUBROUTINE DESCRIPTION

Purpose: To deallocate unused space at the end of a file previously allocated to the file.

Location: Resident System

Calling Sequence:

Assembly: CALL TRUNC,(unit)

FORTRAN: CALL TRUNC(unit,&rc4,&rc8,&rc12,&rc16,&rc20)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 19), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

rc4...rc20 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 The file has been truncated successfully.
4 The file does not exist.
8 Hardware error or software inconsistency encountered.
12 Truncate (or write-extend) access not allowed.
16 Locking the file for modification will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent usage of the (shared) file).

Note:

This subroutine does not optimize or compress line files. It simply checks to see if any space at the end of the file has not been used and, if so, deallocates it.

Examples: Assembly:

```
          CALL  TRUNC,(UNIT)
          .
          .
UNIT      DC    F'5'
```

FORTRAN:
```
INTEGER*4 UNIT
DATA UNIT/5/
  .
  .
CALL TRUNC(UNIT)
```

The above examples will truncate the file attached to logical I/O unit 5.

October 1976

TWAIT

SUBROUTINE DESCRIPTION

Purpose: To wait, for a specified real time interval, and return.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL TWAIT,(code,value)

FORTRAN: CALL TWAIT(code,value)

Parameters:

code is the location of a fullword integer which specifies the meaning of the value parameter. The valid choices are

0 value is an 8-byte binary integer which specifies a time interval in microseconds, relative to the time of the call.
1 value is an 8-byte binary integer which specifies a time interval in microseconds, relative to midnight, March 1, 1900.
2 value is a 16-byte EBCDIC string giving the time and date at which the wait should end, in the form HH:MM.SSMM-DD-YY.

value is the 8- or 16-byte, fullword-aligned region which specifies the time at which the wait should end, as determined by the code parameter.

Return Codes:

0 Successful return
4 Invalid code parameter

Description: The TWAIT subroutine puts the task into wait state until the time interval specified by the code and value parameters has elapsed, and then returns.

Example: FORTRAN:

```
INTEGER TENSEC(2) /0,10000000/
INTEGER TWO30(4)/'02:3','0.00','05-1','0-72'/
    .
    .
CALL TWAIT(0,TENSEC)
CALL TWAIT(2,TWO30)
```

This example calls TWAIT twice, the first time specifying that a pause of 10 seconds relative to the time of the call on TWAIT is to occur, the second time specifying that a pause is to occur which will last until 2:30 am on May 10, 1972.

October 1976

UNLK

SUBROUTINE DESCRIPTION

Purpose: To request that a file be unlocked, i.e., to dynamically allow access to a file (allow it to be shared by others) which has previously been restricted by locking (either explicitly or implicitly).

Location: Resident System

Alt. Entry: UNLCK

Calling Sequence:

Assembly: CALL UNLK,(unit)

FORTRAN: CALL UNLK(unit,&rc4)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 19), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS) used to lock the file (either explicitly in a call to LOCK or implicitly in a call to WRITE, for example).

rc4 is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

0 The file has been unlocked successfully.
4 Illegal unit parameter specified, or hardware error or software inconsistency.

Note:

If more than one FDUB within a job has a locking request on the file, after the call to UNLK, the file is left locked at the level of the highest remaining request.

Description: See Appendix D of the section "Files and Devices" in MTS Volume 1 for details concerning concurrent use of shared files.

Examples:

Assembly:

```
        CALL  UNLK,(UNIT)
        .
        .
UNIT    DC    F'6'
```

FORTRAN:

```
INTEGER*4 UNIT
DATA UNIT/6/
  .
  .
CALL UNLK(UNIT)
```

The above examples will unlock the file attached to logical I/O unit 6 assuming the file has previously been locked (e.g., by a call to the LOCK subroutine).

October 1976

UNLOAD, UNLDF

SUBROUTINE DESCRIPTION

Purpose: To UNLOAD what was loaded on some previous call to the LOAD subroutine.

Location: Resident System

Calling Sequences:

Assembly: CALL UNLOAD,(name,sinbr,sws)

FORTRAN: CALL UNLDF(name,sinbr,sws,&rc4)

Parameters:

name is either the location of the "name" (specified by sws) or zero.

sinbr is either the location of the fullword (INTEGER*4) storage index number or zero. This parameter is referenced only if name is zero.

sws is the location of a fullword switch:

0 name is the FDname from which the material was LOADed.
1 name is an 8-character, left-justified, external symbol.
2 name is a fullword virtual memory location (the SYMTAB option must be ON).

rc4 is the statement label to transfer to if a nonzero return code is encountered.

Return Codes:

0 Successful return.
4 The subroutine could not find the name in the LOAD table, or sws is nonzero and SYMTAB is OFF, or the external symbol or virtual memory address could not be found in the loader tables.

Description: Each time the LOAD subroutine is called, a new storage index number is assigned for use with storage acquired in order to load the material in the file specified for that LOAD call. In order to unload the material, either the storage index number or the name of the file LOADed from may be given. In addition, if the global switch SYMTAB is ON, the name of an external symbol or a virtual memory

location in the material loaded may be specified. In any case, all of the material loaded on that call on LOAD is unloaded. See the "Virtual Memory Management" section in MTS Volume 5 for a further description of using storage index numbers with the LOAD and UNLOAD subroutines.

Examples: FORTRAN: CALL UNLDF('PROGALE ',0,1,&99)

This example calls UNLDF to find the storage index number associated with the external symbol PROGALE. All storage with that storage index number is unloaded.

```
          CALL UNLDF(BUFLOC,0,2,&9)
```

This example calls UNLDF to find the storage index number associated with the virtual memory address in location BUFLOC. All storage with that storage index number is unloaded.

```
Assembly:       CALL  UNLOAD,(0,SIN,0)
                .
                .
          SIN   DS    F
```

This example calls UNLOAD to unload all storage with the storage index number in location SIN.

October 1976

URAND

SUBROUTINE DESCRIPTION

Purpose: To compute uniformly distributed real random numbers between 0 and 1.0.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL URAND,(init)

FORTRAN: x = URAND(init)

Parameters:

init is the location of an (optional) initial integer value.

Values Returned:

FR0 will contain the uniformly distributed random number generated by the subroutine. For FORTRAN users, this value will be returned in x.

Description: If init contains a nonzero odd integer between 1 and $2^{31}-1$ (2147483647), then a new integer random number will be generated using the formula

$$init = (65539 * init) \pmod{2^{31}-1}.$$

The corresponding real random number x will be returned as a function value for FORTRAN or in FR0 for assembly language users.

If init contains zero, the next integer random number will be supplied by the routine and will depend upon the time of day. The new integer random number that is generated will be stored in init. Thus, X = URAND(0) is not permissible.

If the same sequence of random numbers is required on successive runs, the user must supply the same initial value of init.

Examples: Assembly:

```
          CALL URAND,(INTEG)
          STE  0,RAND
           .
           .
INTEG DC   F'999'
RAND  DS   E
```

FORTRAN:

```
I=999
X=URAND(I)
```

In both examples above, URAND is called with the initial value of 999.

October 1976

WRITE

SUBROUTINE DESCRIPTION

Purpose: To write an output record on a specified logical I/O unit.

Location: Resident System

Alt. Entry: WRITE#

Calling Sequences:

Assembly: CALL WRITE,(reg,len,mod,lnum,unit)

FORTRAN: CALL WRITE(reg,len,mod,lnum,unit,&rc4,...)

PL/I: See the IHERITE subroutine description.

Parameters:

reg is the location of the virtual memory region from which data is to be transmitted.

len is the location of a halfword (INTEGER*2) integer giving the number of bytes to be transmitted.

mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.

lnum is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.

unit is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 19), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

0	Successful return.
4	Output device is full.
>4	See the "I/O Subroutine Return Codes" description in this volume.

Description: The subroutine writes a record on the logical I/O unit specified by unit of length len (in bytes) from the region specified by reg. The parameter lnum is used only if the mod parameter or the FDname specifies either INDEXED or PEEL (RETURNLINE#). If INDEXED is specified, the line number to be written is specified in lnum. If PEEL is specified, the line number of the record written is returned in lnum.

There are no default FDnames for WRITE.

There is a macro WRITE in the system macro library for generating the calling sequence to this subroutine. See the macro description for WRITE in MTS Volume 14.

Examples: The example below, given in assembly language and FORTRAN, calls WRITE specifying an output region of 80 bytes. The logical I/O unit specified is 6 and no modifier specification is made in the subroutine call.

Assembly:

```
          CALL WRITE,(REG,LEN,MOD,LNUM,UNIT)
             .
             .
REG       DS   CL80
MOD       DC   F'0'
LNUM      DS   F
LEN       DC   H'80'
UNIT      DC   F'6'

          or

          WRITE 6,REG        Subr. call using macro.
```

FORTRAN:

```
      INTEGER*2 LEN/80/
      INTEGER REG(20),LNUM
        .
        .
      CALL  WRITE(REG,LEN,0,LNUM,6)
```

The example below, given in assembly language and FORTRAN, sets up a call to WRITE specifying that the output will be written into the file FYLE.

```
Assembly:       LA    1,=C'FYLE '
                CALL  GETFD
                ST    0,UNIT
                  .
                  .
                CALL  WRITE,(REG,LEN,MOD,LNUM,UNIT)
                  .
                  .
          REG   DS    20
          LEN   DS    H
          MOD   DC    F'0'
          LNUM  DS    F
          UNIT  DS    F

FORTRAN:        EXTERNAL GETFD
                INTEGER*4 ADROF,UNIT
                CALL RCALL(GETFD,2,0,ADROF('FYLE '),1,UNIT)
                  .
                  .
                CALL WRITE(REG,LEN,0,LNUM,UNIT,&30)
                  .
                  .
          30      :
```

October 1976

October 1976

WRITEBUF

SUBROUTINE DESCRIPTION

Purpose: To write out all changed file buffers.

Location: Resident System

Alt. Entry: WRITBF

Calling Sequences:

Assembly: CALL WRITEBUF,(unit)

FORTRAN: CALL WRITBF(unit,&rc4)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 19), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

rc4 is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

0 Successful return.
4 Illegal unit parameter specified, or hardware error or software inconsistency encountered.

Description: A call on this subroutine causes all changed lines in the file buffers to be written to the file, thus making the file on the disk an up-to-date copy.

This subroutine does not release the file buffers and does not close the file; i.e., it is not necessary to open the file again (read the catalog, etc.) on subsequent I/O operations.

Examples: Assembly:

```
        CALL WRITEBUF,(UNIT)
          .
          .
UNIT    DC  CL8'SPRINT'
```

```
FORTRAN:        CALL WRITBF('SPRINT  ')
```

The above examples cause WRITEBUF to update the disk copy of the file attached to the logical I/O unit SPRINT.

October 1976

XCTL, XCTLF

SUBROUTINE DESCRIPTION

Purpose: To effect the dynamic loading and execution of a program.

Location: Resident System

Calling Sequences:

Assembly: CALL XCTL,(input,info,parlist,errexit,output,
lsw,gtsp,frsp,pnt)

FORTRAN: CALL XCTLF(input,info,parlist,errexit,output,
lsw,gtsp,frsp,pnt)

Parameters:

input is the location of an input specifier to be used during loading to read loader records. An input specifier may be one of the following:

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 8 in info must be 1.
(4) a fullword-integer logical I/O unit number (0-19).
(5) the address of an input subroutine to be called during loading via a READ subroutine calling sequence to read loader records (i.e., the input subroutine is called with a parameter list identical to the system subroutine READ). In this case, bit 9 in info must be 1.

info is the location of an optional information vector. No information is passed if info is 0 or if info is the location of a fullword integer 0. The format of the information vector is as follows:

(1) a halfword of XCTL control bits defined as follows:

bit 0: 1, if errexit parameter is specified.
bit 1: 1, if output is specified.

bit 2: 1, if lsw is specified.
bit 3: 1, if gtsp is specified.
bit 4: 1, if frsp is specified.
bit 5: 1, if pnt is specified.
bit 6: 0
bit 7: 1, to request XCTL to restore the registers of the previous link level before transferring control to the specified program.
0, if the caller has restored them.
bit 8: 1, if input is the location of a logical I/O unit name.
bit 9: 1, if input is the location of an input subroutine address.
bit 10: 1, if output is the location of a logical I/O unit name.
bit 11: 1, if output is the location of an output subroutine address.
bit 12: 1, if the program to be loaded is to be merged with the program previously loaded.
bit 13: 1, to suppress prompting at a terminal.
bit 14: 1, to force allocation of a new loader symbol table.
bit 15: 0

(2) a halfword count of the number of entries in the following initial ESD list.
(3) a variable-length initial ESD list, each entry of which consists of a fullword-aligned 8-character symbol followed by a fullword value.

parlist is the location of a parameter list to be passed in GR1 to the program being transferred to.

errexit (optional) is the location cf an error-exit subroutine address to be called if an error occurs while attempting to transfer to the specified program. If bit 0 of info is 0 (the default), the errexit parameter is ignored and an error return is made to MTS command mode. The exit routine will be called via a standard S-type calling sequence with two parameters defined as follows:

P1: the location of a fullword integer error code defined as follows:

0: attempt to load a null program.
4: fatal loading error (bad object program).
8: undefined symbols referenced by the loaded program.

P2: the location of a fullword containing the loader status word.

If the exit routine returns, XCTL will return to MTS without releasing program storage (i.e., as if the error exit had not been taken).

output (optional) is the location of an output specifier to be used during loading to produce loader output (error messages, map, etc.). If bit 1 of info is 0 (the default), the output parameter is ignored and all loader output is written on the MAP=FDname specified on the initial $RUN command. An output specifier may be one of the following:

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 10 of info must be 1.
(4) a fullword-integer logical I/O unit number (0-19).
(5) the address of an output subroutine to be called during loading via the SPRINT subroutine calling sequence tc write loader output (i.e., the output subroutine is called with a parameter list identical to the system subroutine SPRINT). In this case, bit 11 of info must be 1.

lsw (optional) is the location of a fullword of loader control bits. If bit 2 of info is 0 (the default), the lsw parameter is ignored and the global MTS settings are used. The loader control bits are defined as follows:

bits 0-23: 0
bit 24: 1, to suppress the pseudo-register map.
bit 25: 1, to suppress the predefined symbol map.

bit 26: 1, to print undefined symbols.
bit 27: 1, to print references to undefined symbols.
bit 28: 1, to print references to all external symbols.
bit 29: 1, to print dotted lines arcund the loader map.
bit 30: 1, to print a map.
bit 31: 1, to print nonfatal error messages.

gtsp (optional) is the location of a storage allocation subroutine to be called during loading via a GETSPACE calling sequence to allocate loader work space and program storage. If bit 3 of info is zero (the default), GETSPACE is used.

frsp (optional) is the location of a storage deallocation subroutine to be called during loading via a FREESPAC calling sequence to release loader work space. If bit 4 of info is 0 (the default), FREESPAC is used.

pnt (optional) is the location of a direct access subroutine to be called during loading via a POINT calling sequence while processing libraries in sequential files. If bit 5 of info is 0 (the default), POINT is used.

Values Returned:

None.

Description: XCTL provides a method for dynamically loading and executing programs in an overlay fashion. XCTL provides this facility as follows:

(1) XCTL makes a copy of all its parameter values and releases all storage associated with the current link level.
(2) The loader is called to dynamically load the specified program using input, info, output, lsw, gtsp, frsp, and pnt if specified.
(3) The dynamically loaded program is called with the address of parlist in GR1.
(4) If the dynamically loaded program returns to XCTL, it is unloaded.
(5) XCTL returns to the program which initiated the current link level, preserving the return registers of the dynamically executed program.

Note that XCTL accepts a variable-length parameter list of three to eight arguments. For most applications, only the

first three are required. These parameters passed to XCTL may be part of the current link level to be released, since XCTL makes copies of them. However, the parameter list and parameters passed to the program XCTLed to, as well as the optional subroutines specified by input, output, errexit, gtsp, frsp, and pnt may not be part of the current link level since it is released before the program transferred to, is loaded and executed.

Note that by default it is the user's responsibility to restore the registers of the previous link level before calling XCTL. Since this is possible in general only at the assembly language level, calls to XCTL from higher-level languages (e.g., FORTRAN, PL/I, etc.) must have bit 7 in info set to 1.

FORTRAN programs (or programs that use the FORTRAN I/O library) that dynamically load other FORTRAN programs (or programs using the FORTRAN I/O library) should use the alternate entry point XCTLF. XCTLF is required to provide the dynamically loaded program with a FORTRAN I/O environment consistent with the "merge" bit specified in info. If the merge bit is 1, the dynamically loaded program will have the same I/O environment as the calling program. If the merge bit is 0, the dynamically loaded program will have a separate, reinitialized I/O environment. Both FORTRAN main programs and subroutines can be dynamically loaded using XCTLF. However, the effect of executing a STOP statement from a dynamically loaded subroutine will depend on the setting of the merge bit. If the merge bit is 1, a return is made to the program which linked to the calling program; if the merge bit is 0, a return is made to MTS.

Because the rate structure for use of MTS includes a charge for allocated virtual memory integrated over CPU time, the cost of running a large software package in MTS can often be reduced by dynamically loading and executing sequential phases in an overlay fashion via calls to XCTL. Such savings in the storage integral must be weighed against the additional CPU time required to open a second file, reinvoke the loader, and rescan the required libraries.

The user also should see the sections "The Dynamic Loader" and "Virtual Memory Management" in MTS Volume 5. In particular, they describe the use of initial ESD lists, merging with previously loaded programs, and the relationship between LINK, LOAD, and XCTL storage management.

Example: Assembly:

```
          LA    0,1                 Highest-level stg
          LA    1,PARLEN            Length required
          L     GR15,=V(GETSPACE)   Allocate space
          BALR  GR14,GR15            for par list
          ST    1,XCPAR+8           Save address
          LA    2,4(1)              Set the par list
          ST    2,PARAD
          MVC   0(PARLEN,1),PARAD   Move in params
          LA    1,XCPAR             Get par list ptr
          L     15,=V(XCTL)         GET XCTL address
          L     13,MYSAVE+4         Set save area ptr
          LM    2,12,28(13)         Set caller's regs
          L     14,12(13)
          BR    15                  Invoke XCTL
           .
           .
MYSAVE    DS    18A
XCPAR     DC    A(INPUT,INFO,0)
INPUT     DC    C'*FTN '
INFO      DC    F'0'
PARAD     DC    A(0)
PAR       DC    Y(L'PARSTR)
PARSTR    DC    C'S=-SOU,L=-LOAD,P=-PRINT'
PARLEN    EQU   *-PARAD
```

The above example dynamically loads *FTN and compiles the source program in the file -SOU into the file -LOAD with the listing written to -PRINT. When *FTN returns to XCTL, a return is made to the caller of the above assembly program. Note that if bit 7 of info is zero (the default), it is the responsibility of the program calling XCTL to restore the registers of the previous link before invoking XCTL.

October 1976

PL/I LIBRARY SUBROUTINES

This section contains descriptions of the subroutines that are a part of the PL/I library *PL1LIB.

Each of these subroutines may be called directly. Many other subroutines that require an S-type calling sequence may be called by using the PLCALL subroutine which is described in this section.

October 1976

October 1976

ATTACH

SUBROUTINE DESCRIPTION

Purpose: To associate a PL/I file variable name with an appropriate MTS file or device name.

Location: *PL1LIB

Calling Sequences:

PL/I: `CALL ATTACH(string);`

Parameters:

string is a character string of either fixed or variable length which must follow these restrictions:

(1) the string must not be a null string,
(2) the length of the string must not be more than 255 characters, and
(3) the string must conform to that of PAR=string.

Description: The subroutine passes string to an internal routine which processes the PAR=string format (see Computing Center Memo 260).

Example: PL/I:

```
CALL ATTACH('A=X B=Y@F(80)');
```

This example associates PL/I files A and B with X (an MTS file) and with Y (another MTS file with fixed format of length 80).

October 1976

October 1976

BATCH

SUBROUTINE DESCRIPTION

Purpose: To determine whether the user is in batch or conversational mode.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE BATCH ENTRY
RETURNS (BIT(1));
```

Description: The subroutine returns '1'B if the user is in batch mode; otherwise, it returns '0'B.

Example: PL/I:
```
IF BATCH THEN STOP;
ELSE GOTO RETRY;
```

In this example, if the program is running in batch mode, it stops; otherwise, it transfers to the label RETRY.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

October 1976

CNTL

SUBROUTINE DESCRIPTION

Purpose: To provide an interface between the PL/I user and the CONTROL entry in the device support routines (DSRs). This subroutine allows the PL/I user to execute control operations on files and devices. See the CONTROL subroutine description in this volume.

Location: *PL1LIB

Calling Sequence:

PL/I: CALL CNTL(fdname,info)

Parameters:

fdname is a CHARACTER variable or constant giving the name of a file or device.

info is a CHARACTER variable or constant giving the control information to be passed to the device support routines.

Return Codes:

0 Successful return from CONTROL.

>0 Unsuccessful return from CONTROL. The PL1RC subroutine may be used to interrogate the return code.

Note: The user should exercise care when using the CNTL subroutine if the PL/I file to which fdname refers is open.

Examples: PL/I:

```
CALL CNTL('*T*','REW');
IF PL1RC¬=0 THEN GOTO NOREW;
```

This example calls CONTROL to rewind the tape *T*, and then checks to see if the rewind operation was successful.

```
CALL CNTL('*SINK*','DON''T');
```

This example calls CONTROL with the Data Concentrator or Memorex device support command DON'T.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

472 CNTL

October 1976

CPUTIME

SUBROUTINE DESCRIPTION

Purpose: To obtain the CPU time (in seconds) from the beginning of the current program.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE CPUTIME ENTRY
RETURNS (FLOAT BINARY);
```

Description: The subroutine returns the floating-point value of the CPU time (in seconds) from the beginning of the program.

Example: PL/I:
```
START_TIME: PROC;
   DCL (TIME1, TIME2) STATIC FLOAT BIN,
       CPUTIME ENTRY RETURNS (FLOAT BIN);
   TIME2 = CPUTIME;
   RETURN;
TIME:  ENTRY FLOAT BIN;
   TIME1 = TIME2;
   TIME2 = CPUTIME;
   RETURN (TIME2 - TIME1);
END;
```

This example determines the amount of CPU time taken in executing a loop. It first calls START_TIME to initialize the variable TIME2; then, on every call, the procedure TIME returns the CPU time in seconds since the previous call.

474 CPUTIME

October 1976

ELAPSED

SUBROUTINE DESCRIPTION

Purpose: To obtain the elapsed time (in seconds) from the beginning of the program.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE ELAPSED ENTRY
RETURNS (FLOAT BINARY);
```

Description: The subroutine returns the floating-point value (in seconds) from the beginning of the program.

Example:
```
PUT EDIT ('ELAPSED TIME - ',ELAPSED,'SECS')
         (A,F(15,3),A);
```

This example prints out the elapsed time in seconds since the beginning of the program.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

476 ELAPSED

October 1976

FINFO, TFINFO, RFINFO

SUBROUTINE DESCRIPTION

Purpose: To obtain information on a file or device attached to a PL/I file.

Location: *PL1LIB

Calling Sequence:

PL/I:
```
DECLARE FINFO ENTRY(FILE) RETURNS(POINTER);
DECLARE RFINFO ENTRY(POINTER);
  infob=FINFO(pl1file);
  CALL RFINFO(infob);

DECLARE TFINFO ENTRY(FILE,CHARACTER(*))
  RETURNS(POINTER);
DECLARE RFINFO ENTRY(POINTER);
  infob=TFINFO(pl1file,title);
  CALL RFINFO(infob);
```

Description: Given the PL/I file as an argument, the FINFO subroutine returns the pointer value as the address pointing to the GDINFO buffer infob. If the buffer is not available, it returns the null pointer. This buffer is exactly as described in the GDINFO subroutine description in this volume.

The TFINFO subroutine does exactly the same as the FINFO subroutine except that it associates the PL/I file name with the second argument declared as a character string. If a PL/I user wants to open a PL/I file with the TITLE option and wants to inquire for the information on the file, then he should use the TFINFO subroutine with the second argument equal to the expression in the TITLE option.

The RFINFO subroutine should be called to release the information buffer infob when it is no longer needed.

Example: PL/I:
```
DECLARE FINFO ENTRY(FILE) RETURNS(POINTER),
  1 INFO BASED(INFOB),
      2 FDUB POINTER,
      2 TYPE CHARACTER(4),
      2 INP_MAX FIXED(15) BINARY,
      2 OUT_MAX FIXED(15) BINARY,
      2 FDUBTYPE BIT(8),
      2 TYPEINDX BIT(8),
      2 SWITCHES BIT(8),
```

```
      2 RESERVED BIT(8),
      2 IOMODIFIER BIT(32),
      2 START_L# FIXED(31) BINARY,
      2 LAST_L# FIXED(31) BINARY,
      2 END_L# FIXED(31) BINARY,
      2 L#_INCR FIXED(31) BINARY,
      2 FDNAME_PTR POINTER,
      2 ERROR_PTR POINTER,
    1 FDNAME BASED(FDNAME_EQU_PTR),
      2 LTH FIXED(15) BINARY,
      2 NAME CHARACTER(I REFER (LTH));
DECLARE RFINFO ENTRY(POINTER);
      .
      .
INFOB=FINFO(SPRINT);
FDNAME_EQU_PTR=INFO.FDNAME_PTR
      .
      .
CALL RFINFO(INFOB);
```

The FINFO subroutine is called to obtain information about SPRINT; then, the RFINFO subroutine is called to release the information buffer after it is no longer needed.

October 1976

IHEATTN

SUBROUTINE DESCRIPTION

Purpose: To allow a PL/I program to be notified of the occurrence of an attention interrupt.

Location: *PL1LIB

Calling Sequence:

PL/I: CALL IHEATTN;

Description: The IHEATTN subroutine is automatically called befcre the main procedure obtains control. This is to allow all attention interrupts to be controlled by the subroutine. The user may override this call to IHEATTN by calling the subroutine ATTNTRP, thus effectively resetting the attention interrupt conditions as if IHEATTN was not called. These conditions can be restored by calling IHEATTN.

Once IHEATTN has been called and an attention interrupt occurs, IHEATTN scans through all active procedures (the most recent first) and tests for any statement beginning with "ON CONDITION(ATTN)". If no such statement has been executed, the condition "ON CONDITION(ATTN) SYSTEM;" is assumed.

The subroutine will take one of the following actions:

(1) If the keyword "SYSTEM;" is specified, a message such as

```
                 ┌              ┐     ┌          ┐
                 | STMT dddd    |     |  PROC    |
ATTN AT          |    or        | IN  |    or    | name
                 |OFFSET xxxx   |     |ON-UNIT   |
                 └              ┘     └          ┘
```

is printed to identify the location of the interrupt. After the message is printed, the subroutine MTS is called and a return is made to MTS command mode (or debug mode). The user may use the contents of general register 1 which points to the standard 72-byte save area from ATTNTRP to obtain the PSW and registers at the time of the interrupt. The first eight bytes contain the PSW, and the remainder of the region contains the contents of the registers. A $RESTART command may be given to restart the program.

(2) If the keyword "SNAP" is specified after "ON CONDITION(ATTN)", then the above message is printed followed by a list of all active procedures at the time of the interrupt.

(3) If the keyword "SYSTEM;" is not specified, then the ON-unit is entered as a procedure with the attention conditions restored. If the ON-unit returns, IHEATTN automatically returns to the interrupted statement. Caution should be exercised to prevent infinite loops in the ON-unit. It is recommended that the user insert "ON CONDITION(ATTN) SYSTEM;" after "ON CONDITION(ATTN) BEGIN;".

Examples: If the PL/I program that contains no statement beginning with "ON CONDITION(ATTN)" is executed, an attention interrupt will produce a message such as

```
ATTN AT STMT 0021 IN PROC PROGRAM
```

Attention interrupts may be controlled in a PL/I program by the following sequence:

```
DECLARE ATTNSW BIT(1) INIT('0'B);
ON CONDITION(ATTN) SNAP BEGIN;
   ON CONDITION(ATTN) SYSTEM;
   IF ATTNSW = '1'B THEN CALL MTS;
   ATTNSW = '1'B;
   RETURN;
   END;
```

When an attention interrupt occurs for the first time, the attention interrupt message is printed followed by a list of the active procedures. Then the BEGIN block, which resets the ON-condition for attention interrupts and sets ATTNSW to '1', is executed; a return is then made to the statement in which the attention interrupt occurred and program execution is resumed. A subsequent attention interrupt will cause the program to print another interrupt message and then return to MTS command mode. The switch ATTNSW may be used by the program to test whether the first attention interrupt has occurred.

October 1976

IHENOTE, IHEPNT

SUBROUTINE DESCRIPTION

Purpose: To provide an interface between the PL/I user and the NOTE and POINT subroutines.

Location: *PL1LIB

Calling Sequence:

PL/I:
```
CALL IHENOTE(file,ptrs);
CALL IHEPNT(file,ptrs,bits);
```

Parameters:

file is a FILE variable which must first be opened either implicitly or explicitly.
ptrs is an array of four fullword elements.
bits is a BIT(4) variable or constant. The bit switches are:

```
'0001'B - set read pointer
'0010'B - set write pointer
'0100'B - set last pointer
'1000'B - set last line number
```

More than one switch may be set to give the desired combination of pointers, e.g., '1111'B sets all pointers.

Return Codes:

The subroutine PL1RC may be used to determine the return codes from NOTE and POINT.

Note: These two subroutines are intended for interaction with the two PL/I subroutines IHEREAD and IHERITE.

Example: PL/I:
```
DECLARE IHEPNT ENTRY(FILE,(4) FIXED BINARY(31),
  BIT(4)), QQSV FILE,
  PTRS (4) FIXED BINARY(31);
PTRS=0;
CALL IHEPNT(QQSV,PTRS,'0001'B);
```

This example rewinds the file QQSV for input only.

October 1976

October 1976

IHEREAD, IHERITE

SUBROUTINE DESCRIPTION

Purpose: To read (IHEREAD) or write (IHERITE) a record from a PL/I file.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
CALL IHEREAD(buff,[lth,]mod,lnr,file);
CALL IHERITE(buff,[lth,]mod,lnr,file);
```

Parameters:

- buff is the CHARACTER variable or constant to be read or written.
- lth (optional) is the FIXED BINARY(15) variable or constant giving the length of the record to be read or written. If omitted, the length of buff is used as the record length.
- mod is the BIT(32) variable or constant defining 32 modifier bits used to control the action of the I/O subroutine (see the "I/O Modifiers" section in this volume).
- lnr is the FIXED DECIMAL(9,3) variable or constant giving the line number to be read or written.
- file is the FILE variable to be used in the I/O operation.

Description: The PL/I user should note the following restrictions.

(1) The file, if it is to be used by IHEREAD or IHERITE, must be a record file with undefined format or unblocked fixed format.
(2) It will be the user's responsibility if he mixes these subroutines with READ, WRITE, or REWRITE statements.
(3) An output file cannot be used for IHEREAD, nor an input file for IHERITE. An update file can be used for both IHEREAD and IHERITE.
(4) If the indexed bit of a modifier is on, a line number must be provided. Otherwise, a data interruption may occur, or some unpredictable results will occur. In addition, in case of IHEREAD, the character string will become a null string when there is no line associated with the line number.

Example:

```
MAIN:  PROCEDURE OPTIONS(MAIN);
     DCL (IHEREAD,IHERITE) ENTRY
          (,BIT(32),DEC FIXED(9,3),FILE),
          BUFFER CHAR(121)VARYING,
          MOD BIT(32) INIT((32) '0'B),
          LINENR DEC FIXED (9,3), NUTS FILE;
     ON ENDFILE (NUTS) GO TO FINISH;

OVER:  CALL IHEREAD(BUFFER,MOD,LINENR,NUTS);
     PUT SKIP LIST (LINENR,BUFFER);
     GO TO OVER;   /*THIS ACTS LIKE A "$LIST" COMMAND*/

FINISH:
     CLOSE FILE(NUTS); OPEN FILE(NUTS) UPDATE;
     SUBSTR(MOD,31) = '1'B; /* TURN INDEXED BIT ON */
     CALL IHERITE('',MOD,1.0,NUTS); /* DELETE LINE 1 */
     CALL IHERITE(' THIS IS LINE #2.5',MOD,2.5,NUTS);
               /* INSERT THE LINE #2.5 */

RETURN;

END MAIN;
```

October 1976

NEXTKEY, LASTKEY

SUBROUTINE DESCRIPTION

Purpose: To determine the key of the next record (NEXTKEY) or the end-of-file record (LASTKEY).

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE (NEXTKEY,LASTKEY) ENTRY
(FILE) RETURNS (CHARACTER (4));
```

Description:

NEXTKEY: FILE is opened as:
output - returns the key of the next record to be written.
input or update - returns the key of the next record to be read.

LASTKEY: returns the key of the end-of-file record.

FILE: a PL/I file variable conforming to the following:

(1) a keyed file of the consecutive organization, i.e., referring to an actual MTS sequential file.
(2) must be already opened by either an OPEN statement or by an appropriate I/O statement.

Example: PL/I:
```
POINT=NEXTKEY(KEYED_FILE);
LOCATE BASED FILE(KEYED_FILE) KEYFROM(POINT);
BASED='ABC';
```

This example writes the character string ABC on the next record. BASED is a string variable declared with a PL/I BASED attribute.

October 1976

October 1976

PLCALL, PLCALLD, PLCALLE, PLCALLF

SUBROUTINE DESCRIPTION

Purpose: To enable PL/I users to call non-PL/I (e.g., FORTRAN and assembler) procedures requiring a standard S-type linkage.

Location: *PL1LIB

Calling Sequences:

PL/I:

```
CALL PLCALL(fn,n,pl);

DECLARE PLCALLD RETURNS(FLOAT(16));
PLCALLD(fnd,n,pl);

DECLARE PLCALLE RETURNS(FLOAT(6));
PLCALLE(fne,n,pl);

DECLARE PLCALLF RETURNS(FIXED BINARY(31));
PLCALLF(fnf,n,pl);
```

Parameters:

- fn is a subroutine which has been declared to have the ENTRY attribute and which does not return a value.
- fnd is a function which has been declared to have the ENTRY attribute and which returns a double-precision floating-point value (REAL*8 in FORTRAN; long floating register 0 in assembly code).
- fne is a function which has been declared to have the ENTRY attribute and which returns a single-precision floating-point value (REAL*4 in FORTRAN; short floating register 0 in assembly code).
- fnf is a function which has been declared to have the ENTRY attribute and which returns an integer value (INTEGER*4 in FORTRAN; general register 0 in assembly code).
- n is a number with attributes FIXED BINARY(31) which is equal to the number of arguments being passed to fn, fnd, fne, or fnf. n may be 0.
- pl is a parameter list of the n arguments to be passed to fn, fnd, fne, or fnf in the order required by the subprogram. The arguments are separated by commas. If the argument is a string variable, array variable, or structure variable, the name of the argument or a pointer

to the argument may be used; for example, ARG or ADDR(ARG). Note that if the argument is an array variable, the reference passed will be to the location of the element having all zeros for subscripts (e.g., A(0,0)), even if that element does not exist. Therefore, it may be preferable to use a pointer to an element of the array instead of the array itself (e.g., ADDR(A(1,1)) instead of A). If the argument is a scalar variable, a pointer to the argument must be used; for example, ADDR(ARG). If the argument is a scalar constant, a pointer to the argument, which can be produced by the subroutine PL1ADR must be used. The high-order bit of the last word in the parameter list passed to fn, fnd, fne, or fnf is set to 1. If n=0, there is no parameter list and no comma after n.

Return Codes:

The return code placed in general register 15 by fn, fnd, fne, or fnf may be tested using the subroutine PL1RC.

Description: PL/I program interrupt ON conditions are disabled on entry to the subprogram and reenabled on return to the calling program. The values of PLCALLD, PLCALLE, and PLCALLF are the values returned by fnd, fne, and fnf, respectively.

Examples:

```
/* ARSIN AND DARCOS ARE FORTRAN LIBRARY FUNCTIONS */
DECLARE PLCALLE RETURNS(FLOAT(6));
DECLARE PLCALLD RETURNS(FLOAT(16));
DECLARE (ARSIN, DARCOS) ENTRY;
DECLARE (ARCSIN, ANGLE) FLOAT(6);
DECLARE (ARCCOS, DANGLE) FLOAT(16);
DECLARE F1 FIXED BINARY(31) INIT(1) STATIC;
ARCSIN=PLCALLE(ARSIN, F1, ADDR(ANGLE));
ARCCOS=PLCALLD(DARCOS, F1, ADDR(DANGLE));

/* PAR IS A STRUCTURE VARIABLE */
DECLARE DISMNT ENTRY;
DECLARE 1 PAR ALIGNED STATIC,
          2 LEN INIT(3), 2 TAPE CHAR(3) INIT('*T*');
CALL PLCALL(DISMNT, F1, PAR);
```

```
/* USING PLCALL TO PRODUCE AN R-TYPE LINKAGE */
/* SUBSTR IS A STRING.  F1, F2, AND JULIAN ARE SCALAR */
DECLARE RCALL ENTRY;
DECLARE F6 FIXED BINARY(31) INIT(6) STATIC;
DECLARE PL1ADR RETURNS(POINTER);
DECLARE GJDT ENTRY; /* A NON-EXISTENT ALTERNATE
        NAME FOR GRGJULDT FOR PURPOSE OF THIS EXAMPLE */
DECLARE F2 FIXED BINARY(31) INIT(2) STATIC;
DECLARE DATE CHARACTER(8);
DECLARE F1 FIXED BINARY(31) INIT(1) STATIC;
DECLARE JULIAN FIXED BINARY(31);
CALL PLCALL(RCALL,F6,PL1ADR(GJDT),ADDR(F2),
     SUBSTR(DATE,1,4),SUBSTR(DATE,5),ADDR(F1),
     ADDR(JULIAN));
```

October 1976

October 1976

PL1ADR

SUBROUTINE DESCRIPTION

Purpose: To obtain a pointer to PL/I scalar constants and variables.

Location: *PL1LIB

Calling Sequences:

PL/I:

```
DECLARE PL1ADR RETURNS(POINTER);

PL1ADR(arg);
```

Parameters:

arg is any scalar constant or variable (not strings, arrays, or structures).

Description: The value of PL1ADR is the address of the argument. The primary purpose of this subroutine is to pass pointers for scalar constants to the subroutines PLCALL, PLCALLD, PLCALLE, and PLCALLF since a constant cannot be used as an argument to the PL/I function ADDR.

October 1976

October 1976

PL1RC

SUBROUTINE DESCRIPTION

Purpose: To interrogate the return code passed back by the last call on PLCALL, PLCALLD, PLCALLE, or PLCALLF or set by IHESARC.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE PL1RC RETURNS(FIXED BINARY(31));

PL1RC;
```

Description: The value of PL1RC is the contents of general register 15 when the procedure called using PLCALL, PLCALLD, PLCALLE, or PLCALLF returns, or is the value set by IHESARC, whichever is most recent. For FORTRAN subroutines, the value returned in general register 15 is 4 times the value of the integer after RETURN.

Example:
```
IF PL1RC=4 THEN GO TO ERROR;
```

A branch is made to ERROR if the return code from the last call on PLCALL, PLCALLD, PLCALLE, or PLCALLF is 4.

October 1976

October 1976

RAND

SUBROUTINE DESCRIPTION

Purpose: To compute uniformly distributed random numbers between 0.0 and 1.0.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE RAND ENTRY
(FIXED BINARY (31))
RETURNS (FLOAT BINARY);
```

Description: The argument I as in RAND (I) must be a variable initialized within the range 0 to 2**31-1 (2147483647). The value returned by RAND (I) is between 0.0 and 1.0. In addition, the variable is changed so that a different random number is generated on a subsequent call. If the argument I contains zero, a random number will be generated depending upon the time of day.

The algorithm is taken from "Coding the Lehmer Pseudo-Random Number Generator," Communications of the ACM, Volume 12, Number 2 (February 1969).

Example: PL/I:
```
RANDOM:  PROC FLOAT BIN;
   DCL I FIXED BIN (31) STATIC
         INIT (524287),
      RAND ENTRY (FIXED BIN (31))
         RETURNS (FLOAT BIN);
      RETURN (RAND (I));
      END;
```

This example generates a random number using the number 524287 as the initial base.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

October 1976

SIGNOFF

SUBROUTINE DESCRIPTION

Purpose: To sign the user off.

Location: *PL1LIB

Calling Sequences:

PL/I: DECLARE SIGNOFF ENTRY;

Description: The subroutine closes all open files, if any, and then signs the user off.

Example: PL/I: IF BATCH THEN CALL SIGNOFF;

This example signs off the user if he is running in batch mode.

October 1976

October 1976

USERID

SUBROUTINE DESCRIPTION

Purpose: To obtain the current four-character user Computing Center signon ID.

Location: *PL1LIB

Calling Sequences:

PL/I:
```
DECLARE USERID ENTRY
RETURNS (CHARACTER (4));
```

Description: The subroutine returns the user signon ID.

Example: PL/I:
```
PUT LIST (USERID);
```

This example prints out the user's signon ID.

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

500 USERID

October 1976

THE ELEMENTARY FUNCTION LIBRARY

The elementary function library (EFL) contains the mathematical and implicitly called subroutines usually associated with the FORTRAN IV language. In the FORTRAN language the mathematical routines are called because of an explicit reference to the name of the function in an arithmetic expression. Mathematical routines for the computation of the square root, exponential, logarithmic, trigonometric, hyperbolic, gamma, and error functions are provided. The implicitly called routines are invoked to perform complex multiplication and division, and to perform the various exponentiation operations occasioned by the FORTRAN ** operator. Finally, this library also includes the ANSI FORTRAN intrinsic minimum and maximum value functions, and the DREAL and DIMAG functions, which are inexplicably not a part of the IBM FORTRAN library.

The programs contained in this elementary function library are system resident, and are defined in the low-core symbol dictionary named <EFL>. Special loader control cards at the end of the *LIBRARY file cause the symbol <EFL> to be defined; and, if there are still undefined symbols, then this symbol dictionary will be searched.

List of Entry Points by General Function

Absolute Value	CABS, CDABS
Square Root	SQRT, DSQRT, CSQRT, CDSQRT
Common and Natural Logarithm	ALOG, ALOG10, DLOG, DLOG10, CLOG, CDLOG
Exponential	EXP, DEXP, CEXP, CDEXP
Trigonometric Functions	COS, SIN, TAN, COTAN, DCOS, DSIN, DTAN, DCOTAN, CCOS, CSIN, CDCOS, CDSIN
Inverse Trigonometric Functions	ARCOS, ARSIN, ATAN, ATAN2, DARCOS, DARSIN, DATAN, DATAN2
Hyperbolic Functions	COSH, SINH, TANH, DCOSH, DSINH, DTANH
Gamma and Log-gamma Functions	GAMMA, ALGAMA, DGAMMA, DLGAMA
Error Function	ERFC, ERF, DERFC, DERF
Exponentiation	FIXPI#, FRXPI#, FDXPI#, FCXPI#,FCDXI#, FRXPR#, FDXPD#
Complex Operations	CMPY#, CDVD#, CDMPY#, CDDVD#, DREAL[1], DIMAG[1]
Minimum/Maximum Value	MIN0, AMIN0, MIN1, AMIN1, DMIN1 MAX0, AMAX0, MAX1, AMAX1, DMAX1

---

[1]Since the DREAL and DIMAG functions are not built into the current FORTRAN compilers, they must be explicitly declared as REAL*8 functions.

Mathematical Functions

REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
		CABS[1]	CDABS[1]
SQRT	DSQRT	CSQRT	CDSQRT
EXP	DEXP	CEXP	CDEXP
ALOG	DLOG	CLOG	CDLOG
ALOG10	DLOG10		
COS	DCOS	CCOS	CDCOS
SIN	DSIN	CSIN	CDSIN
TAN	DTAN		
COTAN	DCOTAN		
ARCOS	DARCOS		
ARSIN	DARSIN		
ATAN[1]	DATAN[1]		
ATAN2[2]	DATAN2[2]		
COSH	DCOSH		
SINH	DSINH		
TANH[1]	DTANH[1]		
ERFC[1]	DERFC[1]		
ERF[1]	DERF[1]		
ALGAMA	DLGAMA		
GAMMA	DGAMMA		

FORTRAN Implicitly Called Functions

Complex operations: name(multiplicand-dividend,multiplier-divisor)

COMPLEX*8	COMPLEX*16
CMPY#	CDCMPY#
CDVD#	CDDVD#

Exponentiation: name(base,exponent)

Name	Base	Exponent
FIXPI#	INTEGER*4	INTEGER*4
FRXPI#	REAL*4	INTEGER*4
FDXPI#	REAL*8	INTEGER*4
FCXPI#	COMPLEX*8	INTEGER*4
FCDXI#	COMPLEX*16	INTEGER*4
FRXPR#	REAL*4	REAL*4
FDXPD#	REAL*8	REAL*8

October 1976

ANSI FORTRAN Minimum/Maximum Value

Name	Arguments Mode	Result Mode
MIN0/MAX0	INTEGER*4	INTEGER*4
MIN1/MAX1	REAL*4	INTEGER*4
AMIN0/AMAX0	INTEGER*4	REAL*4
AMIN1/AMAX1	REAL*4	REAL*4
DMIN1/DMAX1	REAL*8	REAL*8

---

[1]These routines do not recognize any error conditions and never transfer to the error monitor.
[2]These routines require two arguments.

Calling Conventions

The programs contained in the EFL conform to the OS(I) S-type calling convention with variable length parameter list as described in section "Calling Conventions" in this volume, i.e., they expect the FORTRAN linkage convention. This convention requires that the high-order bit of the last parameter address constant be nonzero. The EFL error monitor uses this last argument flag to determine how error situations should be processed; consequently, failure to properly set this flag may result in unexpected results if an error condition is detected. Further, unless specifically mentioned, all elements of the EFL require an 18-fullword (72-byte) save area.

Since all members of the EFL are function-type subroutines, they cannot be meaningfully employed in the FORTRAN CALL statement, as the FORTRAN program will ignore the function value returned by these programs. These function subprograms are called whenever the appropriate entry name appears in a FORTRAN arithmetic expression. The following FORTRAN arithmetic assignment statement refers to the mathematical functions COS and SQRT, and the implicitly called exponentiation routine FRXPI#:

```
SINX = SQRT(1.-COS(X)**2)
```

Assembly language users may employ the CALL macro, but should specify the optional VL parameter in order to set the last argument flag byte, e.g.,

```
CALL DCOSH,(X),VL
```

The elementary functions return their values as follows:

GR0	- INTEGER function
FR0	- REAL function
FR0,FR2	- COMPLEX function

A return code of 0 will be given for all successful computations. The return code given in error situations is generally 4, but may be dynamically modified by the user, and hence must be described as indeterminate. See the section on error processing for further details.

Except as noted, the mathematical functions require a single argument of the same mode as the function. The routines in the EFL are subject to specification exceptions when fetching their argument(s) should the boundary alignment be incorrect. The modes INTEGER*4, REAL*4 and COMPLEX*8 require fullword alignment, while REAL*8 and COMPLEX*16 require doubleword alignment. The term INTEGER*4 corresponds to a System/360 fullword integer in the usual twos-complement notation. The term REAL*4 (REAL*8) corresponds to a System/360 short (long) operand floating-point number. The term COMPLEX*8 (COMPLEX*16) refers to two short (long) operand floating-point numbers occupying consecutive storage locations, the number in the higher storage location being the imaginary part of the complex number. The address constant passed to the EFL routine should correspond to the lower storage address, i.e., the REAL part of the complex number.

Error Processing

Error conditions detected by EFL routines are processed in the module ERRMON#. Depending on the optional arguments passed to the elementary function, the error monitor will either resume execution or formulate an appropriate error comment and call the traceback program TRACER.

The vast majority of the EFL programs check the argument to ensure that a valid function value can be computed. For example, the inverse sine and cosine functions are only defined on the interval [-1,1], so that some procedure must be available for handling arguments outside this interval. There are currently three ways in which error conditions detected by an EFL program can be processed:

(1) by using one or more of the optional arguments described below,
(2) by establishing a user error monitor to be called in these situations, or
(3) by allowing the EFL error monitor to invoke the traceback program TRACER.

Whenever an elementary function detects an error situation, it generates a default function value and passes control to the EFL error monitor. Although this error monitor is in fact a separate program, it is logically a part of each elementary function, and is transparent with respect to the normal linkage conventions. Thus, if the EFL error monitor invokes either the user error monitor or TRACER, it will appear to them as if they had been called directly from the elementary function.

The EFL error monitor initially attempts to process the optional arguments. If no such arguments were given, or if their processing does not result in the resumption of execution, then the error monitor will

formulate an appropriate message. This message is subsequently used as the sole argument when either the user error monitor or TRACER is invoked.

With all optional arguments attached, the calling sequence becomes

...name(argument(s),count,max-count,f-value)...

Since the elementary function names are built into the FORTRAN compiler, it will diagnose as errors any occurrence of these names in which the number and modes of the arguments do not correspond to its table of definitions. The optional arguments discussed here may be appended to the usual argument list, without objection from the FORTRAN compiler, if the elementary function name is declared in an EXTERNAL statement and its proper mode is explicitly declared. The optional arguments are defined as follows:

count - a fullword integer which is simply incremented by 1. If count is the only optional argument supplied, then execution is resumed with the default function value and return code 4.

max-count - a fullword integer upper bound for the first optional argument, count. If the updated value of count is greater than max-count, then the processing of the optional arguments is suspended. If max-count is the last optional argument supplied and the updated value of count is less than or equal to max-count, execution is resumed with the default function value and return code 4. Otherwise, the final optional argument is processed.

f-value - the mode of this argument must correspond to the mode of the function. Execution is resumed with a function value of f-value and return code 4. Note that this optional argument is processed only if the updated value of count is less than or equal to max-count.

In the above descriptions, the phrase "resume execution" means that it will appear that the elementary function has returned with the indicated function value and return code.

If one of the optional arguments cannot be appropriately accessed, if count > max-count, or if no optional arguments are supplied, then the error monitor will formulate an error message. This error message will be subsequently passed to either the user error monitor or TRACER. For the mathematical functions, this error message will take the form

name(x.x) IS UNDEFINED AND HAS BEEN ASSIGNED THE VALUE y.y.
THE DOMAIN OF DEFINITION OF THIS FUNCTION IS dod-message.

where "x.x" and "y.y" are decimal representations of the argument and function value, respectively. The "dod-message" is dependent on the

elementary function involved, but generally expresses the set of argument values for which the function is defined in the form

(x: a < x < k)

For example, the GAMMA function "dod-message" is "IS (X: .1381786E-75 < X < 57.57441)".

Messages generated for exponentiation errors take the form:

> EXPONENTIATION ERROR: b.b ** e.e IS UNDEFINED AND HAS BEEN ASSIGNED THE VALUE y.y. MODE OF THE BASE IS mb, MODE OF THE EXPONENT IS me.

where "b.b", "e.e", and "y.y" are decimal representations of the base, exponent and result, respectively. The modes "md" and "me" will be one of the following: INTEGER*4, REAL*4, REAL*8, COMPLEX*8 or COMPLEX*16. Generally, exponentiation routines only recognize an error when the base is 0.0 and the exponent is nonpositive; however, the current routines also complain when a real result cannot be properly represented, e.g., 10.**80. In either case, the error monitor dynamically allocates virtual memory space sufficient to generate and assemble this message. The message is generated in the form of a halfword integer length immediately followed by the text of the message.

An elementary function library user error monitor is established by using the CUINFO subroutine. The name and index of the corresponding CUINFO item is 'EFLUEM ' and 183, respectively, while the data is the address of the user error monitor. Thus, to establish a subroutine named UEM as the user error monitor, one could include the following FORTRAN statements in his program.

```
EXTERNAL $UEM$
CALL CUINFO(183,$UEM$)
```

A user error monitor may be eliminated by calling CUINFO with a second argument of zero. The single argument to the user error monitor, which may be either a FUNCTION or SUBROUTINE subprogram, should be declared as an INTEGER*2 vector, e.g.,

```
SUBROUTINE $UEM$(MSG)
INTEGER*2 MSG(2)
CALL SERCOM(MSG(2),MSG(1),0)
RETURN
END
```

This rather simple example prints the message on logical I/O unit SERCOM, and then resumes execution with the default function value. Since the messages are generally longer than a terminal output line, some of the message will be lost. The TRACER program automatically breaks this message into a number of output lines, so that no information is lost. It should be noted that unless the user error

monitor returns to the EFL error monitor, the virtual memory space allocated by this latter program will not be released.

Finally, if the optional argument processing did not result in the resumption of execution, and no user error monitor is established, then the EFL error monitor will invoke the traceback program TRACER. This program was designed to provide conversational control of program execution under these circumstances. The name stems from its primary function, which is to make available information pertinent to programs in the current linkage chain, i.e., the sequence of programs which have been called, but which have not yet returned to their calling programs. For example, if a main program named MAIN calls a subroutine named SUB, which attempts to compute DLOG(-5.D0), then the linkage chain is DLOG, SUB, MAIN, and SYSTEM. Using the various TRACER commands, the arguments to each program in the linkage chain can be inspected or altered, execution can be resumed at any entry or return point of a program in the linkage chain, or one can simply return control to the system. The TRACER commands are read from the logical I/O unit GUSER using a ":" or "?" prefix depending on whether the previous command was successfully or unsuccessfully executed, respectively. All TRACER output is printed on SERCOM. For a complete description of the traceback program, see Computing Center Memo 218.

Example 1:

```
C PROGRAM TO COMPUTE THE SQUARE ROOTS OF THE
C ABSOLUTE VALUES OF THE NUMBERS READ FROM THE
C INPUT STREAM AND KEEP A COUNT OF THE TOTAL
C NUMBER OF NEGATIVE NUMBERS READ.
      EXTERNAL SQRT
      INTEGER I/0/
10    READ 100,X
      Y = SQRT(X,I)
      PRINT 200,X,Y,I
      GO TO 10
100   FORMAT (E20.8)
200   FORMAT (2E17.9,I5)
      END
```

Example 2:

If the fourth statement in example 1 is replaced by

```
Y = SQRT(X,I,10)
```

then traceback processing will occur when the 11-th negative argument is passed to SQRT.

Example 3:

```
C PROGRAM TO TEST THE IDENTITY
C COS(X)**2 + SIN(X)**2 = 1
C FOR VALUES OF X READ FROM THE INPUT STREAM.  THE
```

```
C DSIN AND DCOS ROUTINES ARE UNDEFINED FOR X > PI*2**50,
C BUT THE DEFAULT VALUES CHOSEN GUARANTEE THE IDENTITY.
      EXTERNAL DCOS,DSIN
      REAL*8 DCOS,DSIN,X,ONE
10    IER = 0
      READ 100,X
      ONE = DCOS(X,IER,IER,0.D0)**2+DSIN(X,IER,IER,1.D0)**2
      PRINT 100,IER,ONE
      GO TO 10
100   FORMAT (E20.8)
200   FORMAT (I3,E17.9)
      END
```

Example 4:

The use of the following parameter list would guarantee that the elementary function would always denote error situations by a return code of 4.

```
          DC    A(argument),XL1'FF',AL3(ERRCNT)
ERRCNT    DC    F'0'
```

In addition, the word ERRCNT would be automatically updated to maintain a count of the total number of errors. Taking the other extreme, the parameter list

```
          DC    A(argument,0)
```

would guarantee the invocation of the traceback program, since the error monitor's attempt to increment the count parameter would cause an addressing exception.

Mathematical Functions

The following descriptions of the mathematical functions are limited to error conditions which may arise in these programs. These routines are consistent with the FORTRAN IV library functions currently distributed with the System/360 Operating System, and have been documented by IBM in their publication IBM System/360 Operating System FORTRAN IV Library - Mathematical and Service Subprograms, form GC28-6818.

Square Root

Because SQRT and DSQRT are specifically defined as being real-valued functions, they are not defined for negative real arguments, as the square root of a negative number is pure imaginary, i.e., if $x<0$ then $x^{**}1/2 = i \cdot |x|^{**}1/2$. The default function value computed when the argument is negative is the square root of the absolute value of the argument.

October 1976

Common and Natural Logarithm

The real-valued logarithm functions ALOG, ALOG10, DLOG and DLOG10 are not defined for negative arguments, since the logarithm of a negative number is complex, i.e., if x<0 then ln(x) = ln(|x|)-i•Pi. The default function value is the appropriate logarithm of the absolute value of the argument.

All of the logarithmic function routines are undefined for a zero argument, as this is a pole of the logarithm function. Appropriately, the default function value is negative machine infinity, i.e., roughly $-.7237005 \cdot 10^{76}$.

Exponential

The real-valued functions EXP and DEXP can be properly defined only in the interval [-180.2182,174.67308], because of the range restrictions imposed by the floating-point representation. The largest positive number representable in System/360 floating-point form is $16^{63} \cdot (1-16^{-14})$, and the natural logarithm of this number is approximately 174.67308. Similarly, -180.2182 is the logarithm of the smallest positive number, 16^{-65}. The actual domains are as follows:

EXP (hex)	-B4.37DF	AE.AC4F
DEXP (hex)	-B4.37DEFFFFFFFF	AE.AC4EFFFFFFFF
EXP (dec)	-180.218246	174.673080
DEXP (dec)	-180.218246459960934	174.673080444335934

If the argument exceeds the right-hand limit, the default function value is machine infinity. If the argument is less than the left-hand limit, the default function value is zero; however, this situation is regarded as an error if and only if underflow exceptions are enabled by the program mask.

It should be noted that the domain of the exponential programs is slightly smaller than the range of the corresponding natural logarithm programs. Hence, the expressions EXP(ALOG(X)) and DEXP(DLOG(X)) are not computable for values of X extremely close to the ends of the machine range.

The complex-valued functions CEXP and CDEXP have an analogous domain restriction on the real part of the complex argument, and an additional restriction on the imaginary part due to the sine and cosine function evaluations required. Whether the complex argument satisfies the domain restrictions or not, the value of the CEXP(x+i•y) will be

$$EXP(x) \cdot [COS(y) + i \cdot SIN(y)]$$

and that of CDEXP(x+i•y) will be

$$DEXP(x) \bullet [DCOS(y) + i \bullet DSIN(y)]$$

Trigonometric Functions

The domain restrictions of the real-valued trigonometric functions COS, SIN, TAN, COTAN, DCOS, DSIN, DTAN and DCOTAN are imposed to maintain accuracy. These functions are computed by reducing the argument to the interval [-Pi/4,Pi/4] by using the periodicity of these functions. For very large arguments this reduction yields so few significant digits in the reduced argument that meaningful computation of the function value is impossible. The single-precision functions require

$$|x| < 2^{18} \bullet Pi = C90FD.9 = 823549.563$$

while the limit for the double-precision functions is

$$|x| < 2^{50} \bullet Pi = C90FD9FFFFFFF.F = 3537118706008063.94.$$

The default function value is uniformly zero.

In addition, the tangent and cotangent functions will object if the argument is too close to one of their singularities to maintain accuracy, or if the function value would exceed the machine range. In these situations, the default function value is machine infinity with the sign of the argument.

The complex sine and cosine functions CCOS, CDCOS, CSIN and CDSIN can be defined as

$$\sin(x+i \bullet y) = \sin(x) \bullet \cosh(y) + i \bullet \cos(x) \bullet \sinh(y),$$

$$\cos(x+i \bullet y) = \cos(x) \bullet \cosh(y) + i \bullet \sin(x) \bullet \sinh(y).$$

These formulas illustrate why a trigonometric-type domain restriction is applied to x, and an exponential-type domain restriction to y. The default function value is derived from the default values supplied by the appropriate sine, cosine and exponential routines, where cosh(y) and |sinh(y)| become machine infinity divided by 2 when |y| is too large.

Inverse Trigonometric Functions

The domain of the inverse sine and cosine functions ARCOS, ARSIN, DARCOS and DARSIN is the range of the sine and cosine functions, i.e., [-1,1]. Outside this interval, the default function value is zero.

The inverse tangent routines ATAN2 and DATAN2 are undefined only for the argument pair (0.,0.), for which the default function value is zero. In effect, given the argument pair (y,x), these routines compute the principal value of the argument of the complex number x+i•y.

Hyperbolic Functions

The value of the hyperbolic sine and cosine of x exceed the range of the machine when |x| approaches the logarithm of machine infinity. Specifically, the domain of the COSH and SINH routines is described by

$$|x| \geq AF.5DC0 = 175.366211,$$

and that of DCOSH and DSINH by

$$|x| \geq AF.5DC0FFFFFFFF = 175.366226196289059.$$

the default function value is machine infinity with the appropriate sign.

Gamma and Log-gamma Functions

Like the exponential function, these functions exceed machine range outside their domains of definition and have a default function value of machine infinity. The specific hexadecimal intervals of definition are

GAMMA	$[.100001 \bullet 16^{-62}, 39.930D]$
DGAMMA	$[.100001 \bullet 16^{-62}, 39.930CFFFFFFFF]$
ALGAMA	$[0, .184D30 \bullet 16^{62}]$
DLGAMA	$[0, .184D2FFFFFFFFF \bullet 16^{62}]$

while in decimal these intervals become

GAMMA	$[.138178829 \bullet 10^{-75}, 57.5744171]$
DGAMMA	$[.13817882865895404 \bullet 10^{-75}, 57.5744171142578089]$
ALGAMA	$[0, .429370581 \bullet 10^{74}]$
DLGAMA	$[0, .4293705810082411 43 \bullet 10^{74}]$.

Implicitly Called Functions

Complex Arithmetic Operations

CMPY#	(COMPLEX*8-multiplicand,COMPLEX*8-multiplier)
CDVD#	(COMPLEX*8-dividend,COMPLEX*8-divisor)
CDMPY#	(COMPLEX*16-multiplicand,COMPLEX*16-multiplier)
CDDVD#	(COMPLEX*16-dividend,COMPLEX*16-divisor)

Algorithm:

The multiplication algorithm takes the form

$$(x+iy) \bullet (u+iv) = (x \bullet u - y \bullet v) + i(v \bullet x + u \bullet y).$$

The division algorithm is likewise direct, and takes the form

$$\frac{(x \bullet u + y \bullet v) + i\,(u \bullet y - v \bullet x)}{u \bullet u + v \bullet v}$$

Error Conditions:

Both underflow and overflow exceptions may occur during the formation of the final result. If underflows are masked off and u•u and v•v underflow, a zero-divide exception may also occur.

Exponentiation

FIXPI#	(INTEGER*4-base,INTEGER*4-exponent)
FRXPI#	(REAL*4-base,INTEGER*4-exponent)
FDXPI#	(REAL*8-base,INTEGER*4-exponent)
FCXPI#	(COMPLEX*8-base,INTEGER*4-exponent)
FCDXI#	(COMPLEX*16-base,INTEGER*4-exponent)

Algorithm:

Though each of these routines differ in some way, they all obtain the result by the successive squaring algorithm. This algorithm exploits the binary representation of the integer exponent to compute R=B**I in the following steps:

(1) Initialize R=1., S=B and k=0.
(2) If the k-th bit of |I| is 1, replace the current value of R by R•S.
(3) If one or more of the unexamined bits of |I| is 1, replace S by S•S, increment k by 1, and return to step (2); otherwise, R=B**|I|.

The FIXPI# routine recognizes a number of special cases, none of which actually require any computation.

Base:	≠0	1	-1	-1	≠0
Exponent:	0	any	even	odd	<0
Result:	1	1	1	-1	0

During the course of the algorithm, the result is not range-checked, consequently, the result is valid only if it is in machine range, i.e., less than 2^{31} = 2,147,483,648.

The FRXPI# and FDXPI# routines form B**|I|, and then divide this result into 1.0 if I is negative. Both routines recognize a nonzero base and zero exponent as a special case having value 1. These routines range-check the result as it is being formed, and will invoke error processing if B**|I| or B**I are not machine representable. In FRXPI#, B**|I| is formed in double precision.

In the FCXPI# and FCDXI# routines, a negative exponent causes the base to be inverted before the successive squaring algorithm is applied. Both routines recognize a nonzero base and zero exponent as a special case having value 1. These routines do not range-check the result and are subject to underflow and overflow exceptions. Note that if underflow exceptions are masked off, the complex base is extremely small, and the exponent negative, a zero-divide exception may occur when the base is initially inverted. These routines use the end of the save area for scratch storage.

Error Conditions:

All of these routines recognize a zero base and nonpositive exponent as an error. In addition, the FRXPI# and FDXPI# routines will invoke error processing if either B**|I| or the final result is outside machine range. In all cases, the default function value is zero.

FRXPR# (REAL*4-base,REAL*4-exponent)
FDXPD# (REAL*8-base,REAL*8-exponent)

Algorithm:

The result is obtained by using the appropriate logarithm and exponential routines, i.e.,

e ** (exponent•ln(base)).

These routines recognize as a special case the combination of a zero base and positive exponent. Note that if exponent•ln(base) < 0, the final result is not in machine range, and underflows are masked off, these routines may return a result of zero.

Error Conditions:

The combination of a zero base and nonpositive exponent causes error processing to be invoked with a default value of 0. Denote the base by B and the exponent by E. If B<0 , but |B|**E is in machine range, the default function value is |B|**E. If E•ln(|B|) is within machine range, but the result is not, the default function value will be zero if E•ln(|B|)<0 and machine infinity if E•ln(|B|)>0. If E•ln(|B|) is not in machine range, the default function value is zero.

DREAL and DIMAG Functions

DREAL (COMPLEX*16-variable)
DIMAG (COMPLEX*16-variable)

Algorithm:

Although these routines are described in the FORTRAN language manual, the currently available FORTRAN compilers do not recognize these names as anything special. Consequently, it is normally necessary to explicitly declare them as REAL*8 functions, as otherwise they will be assigned the default mode.

These routines are extremely trivial, consisting of the bare minimum of three instructions. Only general register 1 and floating-point register 0 are altered by these routines, and a save area is not required.

Error Conditions:

These routines are subject to specification exceptions, as they assume the argument is doubleword-aligned.

ANSI Minimum/Maximum Value Functions

```
MIN0/MAX0       (INTEGER*4-variable,...)
AMIN0/AMAX0     (INTEGER*4-variable,...)
MIN1/MAX1       (REAL*4-variable,...)
AMIN1/AMAX1     (REAL*4-variable,...)
DMIN1/DMAX1     (REAL*8-variable,...)
```

Algorithm:

These routines are identical in structure, accepting a variable number of arbitrary arguments of the appropriate mode and recognizing no error situations. The result modes of these entry points are determined by the first character of the function names as follows: M=INTEGER*4, A=REAL*4 and D=REAL*8. The number of arguments processed is determined by the last argument flag, and consequently, addressing or protection exceptions may occur if this flag is not properly set.

October 1976

I/O SUBROUTINE RETURN CODES

The return codes that may result from a call on an input or output subroutine depend on the type of the file or the device used in the operation. In general, a return code of 0 means successful completion of the input or output operation, and a return code of 4 means end-of-file for an input operation and end-of-file-or-device for an output operation. If the file or device being used was specified as part of an explicit concatenation (and is not the last member of that concatenation), a return code of 4 causes progression to the next element of the concatenation, and that return code is not passed back to the caller (unless the NOEC modifier was specified). Thus, for example, if

```
SCARDS=A+B
```

then when the call is made to the SCARDS subroutine after the last line in A has been read, the file routines signal an end-of-file, but this is intercepted, and the first line in B is read instead.

Return codes greater than 4 are normally not passed back to the caller but instead, an error comment is printed and control is returned to MTS command or debug mode. There are two ways to suppress this action and gain control in this situation. First, the subroutines SETIOERR and SIOERR (see descriptions in this volume) are provided to permit a global intercept of all input/output errors. Second, specifying the ERRRTN modifier on an I/O subroutine call will cause all return codes to be passed back.

A description of the return codes that may occur with a particular file or device is given with the appropriate sections of MTS Volume 4. In addition, a summary is given below. Nonzero return codes marked with an asterisk are normally not passed to the calling program; the others are always passed to the calling program.

Files:

Input	0	Successful return
	4	End-of-file (sequential read) Line not in file (indexed read)
	8*	Error
	12*	Access not allowed
	16*	Cannot wait due to deadlock
	20*	Illegal operation on sequential file
	24*	Backwards operation not allowed on sequential file
	28*	Wait interrupted

Output	0	Successful return
	4	End-of-file (line number not in line-number range)
	4*	Size of file exceeded
	8*	Line numbers not in sequence (SEQWL)
	12*	Access not allowed
	16*	Cannot wait due to deadlock
	20*	Sequential file written with indexed modifier, or written with starting line number other than 1
	24*	Disk allotment exceeded
	28*	Hardware or system error
	32*	Line truncated (@SP on sequential file)
	36*	Line padded (@SP on sequential file)
	40*	Wait interrupted

Magnetic Tape:

Input	0	Successful return
	4	Tape-mark (end-of-file) sensed on read, BSR, or FSR operation
	8	Load point reached on BSR or BSF control command
	12*	Logical end of labeled tape reached on read, FSR, or FSF operation
	16*	Nonrecoverable hardware I/O error, data converter check, invalid control command, or invalid control command parameter
	20*	Should not occur
	24*	Fatal error (may be due to hardware malfunction, label error in which the position of the tape is uncertain, or pulling the tape cff the end of the reel during a read, FSR, or FSF operation); following a fatal error, the tape must be rewound before any other I/O operation is allowed
	28*	Volume or data set in error
	32*	Sequence error caused by issuing a control command when the tape is not positioned properly; or a read, FSR, or FSF operation following a write operation
	36*	Deblocking error caused by improper blocking parameters, e.g., attempting to deblock a format FB file using a format VB specification
Output	0	Successful return
	4	End-of-tape marker sensed during write or WTM operation
	8	Load point reached on BSR or BSF control command

	12*	Attempt to write more than 5 additional records after end-of-tape marker sensed
	16*	Nonrecoverable hardware I/O error, data converter check, invalid control command, or invalid control command parameter
	20*	Attempt to write on file-protected tape or unexpired file
	24*	Fatal error (may be due to hardware malfunction, label error in which the position of the tape is uncertain, or pulling the tape off the end of the reel during a read, FSR, or FSF operation); following a fatal error, the tape must be rewound before any other I/O operation is allowed
	28*	Volume or data set in error
	32*	Sequence error caused by issuing a control command when the tape is not positioned properly; or a read, FSR, or FSF operation following a write operation
	36*	Blocking error caused by improper blocking parameters or parameters which are inconsistent with the labels of the file being written

Paper Tape:

Input	0	Successful return
	4	End-of-file
	8*	End-of-tape
	12*	Invalid control command
	16*	Hardware malfunction
	20*	Parity error

Card input under HASP:

Input	0	Successful return
	4	End-of-file
	8*	Attempt to read in column binary mode
Output	8*	Attempt to write on card reader

Printed output:

Input	8*	Attempt to read from printer
Output	0	Successful return
	8*	Local page limit exceeded

		(user never regains control after a global limit is exceeded)

Punched output:

Input	8*	Attempt to read from punch
Output	0	Successful return
	8*	Local card limit exceeded (user never regains control after a global limit is exceeded)

MERIT Network:

Input:	0	Successful return
	4	End-of-file read from network. This does not necessarily mean that there is no more data to be read from the network, only that the remote host has sent an end-of-file.
	8*	Read not allowed; must write. This means that the remote host is requesting input from the network and, to avoid a deadlock, the local program must not read from the network. The prompting characters sent by the remote host when it did the read are returned to the user.
	12*	Should not occur
	16*	Connection is closed: no I/O may be done
	20*	Should not occur
	24*	Attention interrupt received from network
	28*	Same as return code 8 except that the remote host has requested that the input area be blanked for "n" characters, where "n" is returned as a 2-digit decimal number followed by the prompting characters. A value of "00" means that no specific number of characters has been specified.
Output	0	Successful return
	4*	Should not occur
	8*	Write not allowed; must read. This means that the remote host has issued a write on the network and, to avoid a deadlock, the local program must not write on the network.
	12*	Should not occur
	16*	Connection is closed: no I/O may be done
	20*	Should not occur
	24*	Attention interrupt received from network

Control	0	Successful return
	4*	Should not occur
	8*	Control command not allowed--the remcte host has not done a read
	12*	Should not occur
	16*	Connection is closed: no I/O may be done
	20*	Invalid syntax or context for control command
	24*	Attention interrupt received from network

Most other devices:

Input	0	Successful return
	4	End-of-file
	8*	Error
Output	0	Successful return
	4	End-of-file-or-device (if applicable)
	8*	Error

October 1976

I/O MODIFIERS

This section lists all the I/O modifiers that may be used with FDnames or with calls to I/O subroutines.

The device types discussed below in the exceptions to the default modifier bit specifications are the device types as returned by the GDINFO subroutine. The device types discussed here are:

PTR	Printers
MRXA	Terminals via Memorex 1270 Terminal Controller (includes TTY and 2741)
3270	IBM 3270 Display Station or Courier C-270 Display Station
PDP8	Terminals via the Data Concentrator
SDA	Synchronous Data Adapter
HPTR	Printed output via the batch monitor
MNET	MERIT Computer Network

The values indicated below with each bit specification are the values that the modifier word for a subroutine call would have if only that modifier option was specified.

Bit 31	SEQUENTIAL, S	Value:	1 (dec)	00000001	(hex)
30	INDEXED, I		2	00000002	

Default: SEQUENTIAL
Exceptions: None

The SEQUENTIAL modifier specifies that the input or output operation is to be done sequentially. The INDEXED modifier specifies that an indexed operation is to be performed.

In general, the INDEXED modifier is applied only to line files, while the SEQUENTIAL modifier is applied to line files, sequential files, and all types of devices. Note that the SEQUENTIAL modifier and the sequential file are not directly related.

I/O operations involving line files may be performed with either SEQUENTIAL or INDEXED specified. I/O operations involving sequential files must be done SEQUENTIALly. If the user specifies INDEXED on an I/O operation to a sequential file, an error message is generated unless the global switch SEQFCHK is OFF, in which case the operation is performed as if SEQUENTIAL was specified. Attempting a sequential operation with a starting line number other

than 1, e.g., $COPY FYLE(2), also gives an error comment if SEQFCHK is ON.

I/O operations involving devices, such as card readers, printers, card punches, magnetic tape units, paper tape units, and terminals, are inherently sequential and are normally done SEQUENTIALly. If the SEQUENTIAL modifier is specified, the line number associated with the line is the value of the current line number plus (minus, if the backwards I/O modifier is given) the increment specified on the FDname. If the INDEXED modifier is specified, the line number associated with the line is the line number specified in the calling sequence. For devices, the INDEXED modifier is used primarily in conjunction with the PREFIX modifier. Note that the device treats the I/O operation as if SEQUENTIAL were specified.

For further details about indexed and sequential input/output operations, see the section "Files and Devices" in MTS Volume 1.

Bit 29	EBCD	Value:	4 (dec)	00000004 (hex)
28	BINARY, BIN		8	00000008

Default: EBCD
Exceptions: None

The EBCD/BINARY modifier pair is device-dependent as to the action specified. For card readers and punches, the EBCD modifier specifies EBCDIC translation of the card image; this means that each card column represents one of the 256 8-bit EBCDIC character codes. The BINARY modifier specifies that the card images are in column binary format; this means that each card column represents two 8-bit bytes of information. The top six and bottom six punch positions of each column correspond to the first and second bytes, respectively, with the high-order two bits of each byte taken as zero. Printers and files ignore the presence of this modifier pair.

Other device support routines that recognize this modifier pair are:

(1) The Data Concentrator routines
(2) The Memorex 1270 Terminal Controller routines
(3) The Paper Tape routines
(4) The 3270 Display Station routines

For information on the use of this modifier pair in specifications involving the devices listed above, see the respective sections of MTS Volume 4. The list of device support routines recognizing this modifier is subject to

change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

Bit 27	LOWERCASE, LC	Value:	16 (dec)	00000010 (hex)
26	CASECONV, UC		32	00000020

Default: LOWERCASE
Exceptions: None

The LOWERCASE/CASECONV modifier pair is not device-dependent. If the LOWERCASE modifier is specified, the characters are transmitted unchanged. If the CASECONV modifier is specified, lowercase letters are changed to uppercase letters. This translation is performed in the user's virtual memory region. On input operations, the characters are read into the user's buffer area and then translated. On output operations, the characters are translated in the user's buffer area and then written out. Only the alphabetic characters (a-z) are affected by this modifier. Unlike IBM programming systems, MTS considers the characters ¢, ", and ! as special characters rather than "alphabetic extenders," and thus, the UC modifier does not convert ¢, ", and ! into @, #, and $, respectively.

Bit 25	NOCARCNTRL, NOCC	Value:	64 (dec)	00000040 (hex)
24	CC, STACKERSELECT, SS		128	00000080

Default: NOCARCNTRL
Exceptions: CC for PTR, MRXA, 3270, PDP8, SDA, HPIR, TTY, 2741, DISP, 1052, 1443, 2260, 3066, and BNCH
Controlled by device commands for MNET

The NOCC/CC modifier pair is device-dependent. This modifier pair controls whether logical carriage control (or stacker-selection) on output records is enabled. For printers and terminals, the first character of each record is taken as logical carriage control if it is a valid carriage-control character and if the CC modifier is specified. If the first character is not valid as a carriage-control character, the record is written as if NOCC were specified. For further information on logical carriage control, see Appendix H to the section "Files and Devices" in MTS Volume 1. For card punches, the first character of each card image is taken as the stacker-select character if it is a valid logical stacker-select character (0, 1, or 2) and if the SS modifier is specified. If the first character is not valid as a stacker-select character, the card image is punched as if NOCC were specified. The SS modifier is intended only for

those users who are communicating directly with a physical punch (normally system programmers) and is not intended for normal batch use. Note that the SS and CC modifiers reference the same modifier bit and thus may be used interchangeably.

The magnetic tape and paper tape routines also recognize this modifier pair. For a description of this capability, see MTS Volume 4. Files ignore this modifier pair.

Bit 23		Value:	256 (dec)	00000100 (hex)
22	PREFIX, PFX		512	00000200

Default: ¬PREFIX
Exceptions: None

The PREFIX modifier pair controls the prefixing of the current input or output line with the current line number. On terminal input, the current input line number is printed before each input line is requested. The line number used is determined as specified in the description of the SEQUENTIAL and INDEXED modifiers. An example for terminal input is

```
$COPY *SOURCE*(6,,2)@PFX  A(6,,2)
      6_ first input line
      8_ second input line
          .
          .
end-of-file indicator
```

Note that this would have the same effect with respect to line numbering as

```
$GET A
$NUM 6,2
      6_ first input line
      8_ second input line
          .
          .
     xx_$UNN
```

The current (prefix) line number is not necessarily equivalent to the file line number. In the example above, the prefix line and the file line numbers were explicitly made to correspond by also specifying a line number range on the output FDname (the file A). On input from card readers and files, the PREFIX modifier has no effect. On terminal output, the current line number is printed before each output line is written. The line number used is determined as specified in the section "Files and Devices" in MTS Volume 1. An example for terminal output is

```
$COPY A(1,10)  *SINK*(100,,2)@PFX
    100_ first output line
    102_ second output line
          .
          .
```

Note again that the current line number is not equivalent to the file line number. On output to the printer cr to a file, the PREFIX modifier has no effect.

If the INDEXED and PREFIX modifiers are given together for terminal output, the line numbers referenced by the INDEXED modifier are the same as those produced by the PREFIX modifier. As an example, consider the following FORTRAN program segment:

```
      INTEGER*2 LEN
      DATA MOD/Z00000202/    Enables INDEXED, PREFIX
    1 CALL READ(REG,LEN,0,LNR,2,&2)
      CALL WRITE(REG,LEN,MOD,LNR,3)
      GO TO 1
    2 STOP
```

This program performs a read SEQUENTIAL and a write INDEXED and PREFIX. The command (assuming compilation of the above into -LOAD)

```
$RUN -LOAD  2=A 3=*SINK*
```

is equivalent to

```
$COPY A  *SINK*@I@PFX
```

which is also similar to

```
$LIST A
```

with a slightly different formatting of the line numbers.

Bit 21		Value:	1024 (dec)	00000400 (hex)
20	PEEL, GETLINE#, RETURNLINE#		2048	00000800

Default: ¬PEEL
Exceptions: None

The PEEL modifier pair has two functions, depending upon whether it is specified on input or on output. On input, if the PEEL (GETLINE#) modifier is specified, a line number is removed from the front of the current input line. The line number is converted to internal form (external value times 1000) and returned in the line

number parameter during the read operation (see the subroutine description of SCARDS and READ). The remainder of the line is moved into the input region specified. As an example, consider the following FORTRAN program segment:

```
      INTEGER*2 LEN
      DATA MOD/2048/
    1 CALL SCARDS(REG,LEN,MOD,LNR,&2) Read with PEEL
      CALL SPRINT(REG,LEN,0,LNR)
      GO TO 1
    2 STOP
```

The program reads an input line, removes the line number, and writes out the line without its line number. Execution of the object module of the sample program is as follows:

```
$RUN -OBJ  SCARDS=*SOURCE* SPRINT=ABC
10AAA
12BBB
```

is equivalent to

```
$COPY *SOURCE*@GETLINE#  ABC
10AAA
12BBB
```

Listing the file ABC produces

```
$LIST ABC
     1     AAA
     2     BBB
```

If the PEEL modifier is specified on input in conjunction with the INDEXED modifier on output, the line number of the input line can be used to control the destination of the line during output. For example:

```
      INTEGER*2 LEN
      DATA MOD1/2048/, MOD2/2/
    1 CALL SCARDS(REG,LEN,MOD1,LNR,&2) Read with PEEL
      CALL SPRINT(REG,LEN,MOD2,LNR)    Write INDEXED
      GO TO 1
    2 STOP
```

This program reads an input line, removes the line number, and writes out the line with the extracted line number as the line number specification for an indexed write operation. The following sequence (assuming compilation of the above into -LOAD)

```
$RUN -LOAD  SCARDS=*SOURCE* SPRINT=ABC
10AAA
12BBB
```

is equivalent to

```
$COPY *SOURCE*@GETLINE#  ABC@I
10AAA
12BBB
```

which is also equivalent to

```
$GET ABC
10AAA
12BBB
```

Listing the file ABC produces

```
$LIST ABC
    10     AAA
    12     BBB
```

On output, if the PEEL (RETURNLINE#) modifier is specified, the line number of the current output line is returned in the line number parameter of the subroutine call during the write operation (see the subroutine descriptions of SPRINT, SPUNCH, SERCOM, and WRITE). The line itself is written out and is unaffected by the presence or absence of this modifier. The modifier is used on output to aid the programmer in recording the line number of the current line written out.

Bit			Value:		
19			4096 (dec)	00001000 (hex)	
18	MACHCARCNTRL, MCC		8192	00002000	

Default: ¬MCC
Exceptions: None

The machine carriage-control modifier pair is device-dependent. The MCC modifier is used for printing output (via printers or terminals) from programs producing output in which the first byte of each line is to be used as a machine carriage-control command for output to an IBM 1403 (or 1443) printer. If the MCC modifier is specified and the first byte of the output line is a valid 1403 machine carriage-control command code, the line is spaced accordingly and printing starts with the next byte as column 1. If the first byte is not a valid 1403 machine carriage-control command code, the entire line is printed using single-spacing. Spacing operations performed by machine carriage control occur after the line is printed (as opposed to logical carriage control in which the spacing

is performed before each line is printed). Most programs do not produce output using machine carriage control. The few programs that do (e.g., *ASMG and *TEXT360) internally specify MCC for their output assuming that it is bound for a printer. Hence MCC need not be specified. If the user directs the output to a file, MCC must be specified when the file is printed. For example,

```
$RUN *ASMG SCARDS=A SPRINT=B SPUNCH=C
$COPY B TO *SINK*@MCC
```

The MCC modifier pair is ignored for files and devices other than printers, terminals connected through the Memorex 1270 Terminal Controller, or 3270 Display terminals. For further information on machine carriage control, see Appendix H to "Files and Devices" in MTS Volume 1.

Bit 17		Value:	16384 (dec)	00004000 (hex)
16	TRIM		32768	00008000

Default: ¬TRIM
Exceptions: TRIM for 3270, HPTR, and 3066
Controlled by TRIM option of $SET command for line files and sequential files

The TRIM modifier pair is used to control the trimming of trailing blanks from input or output lines. If the TRIM modifier is specified, all trailing blanks except one are trimmed from the line. If ¬TRIM is specified, the line is not changed. For an input operation, trimming does not physically delete the trailing blanks from the line, but only changes the line length count.

Bit 15		Value:	65536 (dec)	00010000 (hex)
14	SPECIAL, SP		131072	00020000

Default: ¬SP
Exceptions: None

The SPECIAL modifier pair is reserved for device-dependent uses. Its meaning depends upon the particular device type with which it is used. The device support routines recognizing this modifier pair are:

(1) The file routines
(2) The Data Concentrator routines
(3) The paper tape routines
(4) The Memorex 1270 Terminal Controller routines
(5) The 3270 Display Station routines

The file routines use the SPECIAL modifier to mean skip on a read operation to a sequential file, and to mean replace on a write operation to a sequential file. See the section "Files and Devices" in MTS Volume 1.

For information on the use of this modifier pair in specifications involving the devices listed above, see the corresponding sections of MTS Volume 4. The list of device support routines recognizing this modifier is subject to change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

Bit 13		Value:	262144 (dec)	00040000 (hex)
12	IC		524288	00080000

Default: The setting of the IC global switch (initially ON)
Exceptions: None

The IC modifier pair controls implicit concatenation. If the IC modifier is specified, implicit concatenation occurs via the "$CONTINUE WITH" line. If ¬IC is specified, implicit concatenation does not occur. For example, $LIST PROGRAM@¬IC lists the file PROGRAM and prints "$CONTINUE WITH" lines instead of interpreting them as implicit concatenation. The use of the IC modifier in I/O subroutine calls or as applied to FDnames overrides the setting of the implicit concatenation global switch (SET IC=ON or SET IC=OFF) for the I/O operations for which it is specified.

Bit 11	FWD, FORWARDS	Value:	1048576 (dec)	00100000 (hex)
10	BKWD, BACKWARDS		2097152	00200000

Default: FWD
Exceptions: None

The forwards-backwards modifier pair control the direction of the next sequential read operation. On a read backwards operation, the information is placed in the designated region in a manner identical to a read forwards operation, i.e., the front of the logical record is placed at the beginning of the region. For further details on using this modifier, see the section "Files and Devices" in MTS Volume 1.

Bit 3	NOEC	Value:	268435456 (dec)	10000000 (hex)

Default: ¬NOEC
Exceptions: None

If the NOEC modifier is specified (bit 3 in the modifier word is 1) when an I/O subroutine call is made, explicit concatenation will be inhibited, i.e., if an end-of-file (return code 4) occurs, a return will be made to the calling program instead of proceeding with the next member of the concatenation (if any).

Bit 2 NOATTN Value: 536870912 (dec) 20000000 (hex)

Default: ¬NOATTN
Exceptions: None

If the NOATTN modifier is specified (bit 2 in the modifier word is 1) when an I/O subroutine call is made, all pending attention interrupts, and all attention interrupts occurring during the call, are left pending. Normally, if an attention interrupt occurs but cannot be taken, either because the interrupt occurred in a sensitive portion of the system or because of explicit suppression, the ATNBIT bit (accessible through the GUINFO subroutine) is set, indicating that the interrupt has occurred; the taking of the interrupt is delayed until the next I/O call. The use of the NOATTN modifier allows the further delaying of this interrupt by preventing it from being taken on a given I/O call. If the ATTNOFF bit (accessible though the GUINFO subroutine) is set, all attention interrupts occurring during execution of the program are left pending. The user may suppress a pending interrupt by explicitly resetting the ATNBIT bit through a call to the CUINFO subroutine. The NOATTN modifier may be used in ccnjunction with the ATTNOFF bit to suppress attention interrupts during the execution of selected portions of a program. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

Bit 1 ERRRTN Value: 1073741824 (dec) 40000000 (hex)

Default: ¬ERRRTN
Exceptions: None

If the ERRRTN modifier is specified (bit 1 in the modifier word is 1) when an I/O call is made, and if an I/O error occurs when no SETIOERR/SIOERR interception has been established, the error return code is passed back to the calling program instead of printing an error comment. The error comment may be retrieved by calling the subroutine GDINFO. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname. See

the descriptions of SETIOERR and SIOERR this volume for further details.

Bit 0 NOTIFY Value: -2147483648 (dec) 80000000 (hex)

Default: ¬NOTIFY
Exceptions: None

If the NOTIFY modifier is specified (bit 0 in the modifier word is 1) when an I/O subroutine call is made, on return GR0 is set to a value indicating what has happened:

0 = no unusual occurrence
1 = new FDUB opened and no I/O done
2 and above, reserved for future expansion

A new FDUB is opened if implicit concatenation occurred, if a change to the next member of an explicit concatenation is effected, if a replacement FDname is requested, or whenever a FDUB or logical I/O unit is used for the first time. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

October 1976

October 1976

EXTERNAL SYMBOL INDEX

This section contains lists of all the external symbols defined within the system that may be referenced by user programs.

The first list is a master list of all the external symbols presented in alphabetical order. The first column contains the symbol name and the second column gives the library in which the symbol is defined. For the sake of brevity, the symbols listed as IHE....., IHI....., and ILB..... represent a large group of symbols that start with these respective three-letter prefixes and are contained in the libraries PL1SYM, *ALGOLLIB, and *COBLIB. The complete list of these symbols is given with the lists of symbols for each individual library.

The remainder of the lists are separate lists for the symbols in each individual library.

Alphabetical List of All Symbols

<EFL>	Resident System
<FIX>	Resident System
$CYLALOC	Resident System
$EXEC1	Resident System
$JQENT	Resident System
$PLCOMP	*PL360LIB
$POOLCHG	Resident System
$ROUTAB	Resident System
$SPACE	*LIBRARY
**APLGOA	*APLLIB
*APLCONA	*APLLIB
#CCPLOT	*PLOTSYS
#FPCON	*LIBRARY
#IG	*IG
#IGDSM	*IG
#IGETDD	*IG
#IGETHSP	*IG
#IGINITT	*IG
#IGPD	*IG
#PLTMOD	*PLOTSYS
#POSET	*PLOTSYS
#PRASTR	*PLOTSYS
#PSYSYMB	*PLOTSYS
#PVIRT	*PLOTSYS
#PWRIT	*IG
#PWRIT	*PLOTSYS
#PXBND	*PLOTSYS
#RMTCOPY	Resident System
@TESTITP	*LIBRARY
ABSNT	NAAS:SSP
ACCEPT	*LIBRARY
ACFI	NAAS:SSP
ACTIVCNT	Resident System
ACTVLEF#	*IG
ADAMS	*CSMPLIB
ADCON#	OLD:LIBRARY
ADROF	*LIBRARY
ADVLEL	UNSP:LSLIPLIB
ADVLEL	*SLIP
ADVLER	UNSP:LSLIPLIB
ADVLER	*SLIP
ADVLL	UNSP:LSLIPLIB
ADVLL	*SLIP
ADVLNL	UNSP:LSLIPLIB
ADVLNL	*SLIP
ADVLNR	UNSP:LSLIPLIB
ADVLNR	*SLIP
ADVLR	UNSP:LSLIPLIB
ADVLR	*SLIP
ADVLWL	UNSP:LSLIPLIB
ADVLWL	*SLIP
ADVLWR	UNSP:LSLIPLIB
ADVLWR	*SLIP
ADVSEL	UNSP:LSLIPLIB
ADVSEL	*SLIP
ADVSER	UNSP:LSLIPLIB
ADVSER	*SLIP
ADVSL	UNSP:LSLIPLIB
ADVSL	*SLIP
ADVSNL	UNSP:LSLIPLIB
ADVSNL	*SLIP
ADVSNR	UNSP:LSLIPLIB
ADVSNR	*SLIP
ADVSR	UNSP:LSLIPLIB
ADVSR	*SLIP
ADVSWL	UNSP:LSLIPLIB
ADVSWL	*SLIP
ADVSWR	UNSP:LSLIPLIB
ADVSWR	*SLIP
AFGEN	*CSMPLIB
AGSENS	*IG
AHI	NAAS:SSP
ALGAMA	Elementary Fcn. Lib.
ALGOLX	*ALGOLWLIB
ALI	NAAS:SSP
ALOG	Elementary Fcn. Lib.
ALOG#	Elementary Fcn. Lib.
ALOG10	Elementary Fcn. Lib.
ALPHA	*CSMPLIB
AMAX0	Elementary Fcn. Lib.
AMAX1	Elementary Fcn. Lib.
AMIN0	Elementary Fcn. Lib.
AMIN1	Elementary Fcn. Lib.
AND	*LIBRARY
AND	*CSMPLIB
APCH	NAAS:SSP
APFS	NAAS:SSP
APL	Resident System
APLADDR	*APLLIB
APLALOC	*APLLIB
APLCON	*APLLIB
APLCONA	*APLLIB
APLCONAA	*APLLIB
APLCRT1	*APLLIB
APLCRT1A	*APLLIB
APLCRT2	*APLLIB
APLCRT2A	*APLLIB
APLDEL1	*APLLIB
APLDEL1A	*APLLIB
APLDEL2	*APLLIB
APLDEL2A	*APLLIB
APLDEL3	*APLLIB
APLDEL3A	*APLLIB
APLDESC	*APLLIB

APLDREC	*APLLIB
APLEMES	*APLLIB
APLEMESA	*APLLIB
APLERR	*APLLIB
APLESET	*APLLIB
APLESETA	*APLLIB
APLEV	*APLLIB
APLEVAR	*APLLIB
APLEVARA	*APLLIB
APLFND1	*APLLIB
APLFND1A	*APLLIB
APLFND2	*APLLIB
APLFND2A	*APLLIB
APLFND3	*APLLIB
APLFND3A	*APLLIB
APLFORM	*APLLIB
APLFREE	*APLLIB
APLGARB	*APLLIB
APLGARBA	*APLLIB
APLGO	*APLLIB
APLINDX	*APLLIB
APLINDXA	*APLLIB
APLL	NAAS:SSP
APLNOBL	*APLLIB
APLNOBLA	*APLLIB
APLNSRT	*APLLIB
APLNSRTA	*APLLIB
APLNUMB	*APLLIB
APLNUMBA	*APLLIB
APLOC	*APLLIB
APLOWNI	*APLLIB
APLOWNIA	*APLLIB
APLOWRS	*APLLIB
APLOWRSA	*APLLIB
APLREL0	*APLLIB
APLREL0A	*APLLIB
APLREL1	*APLLIB
APLREL1A	*APLLIB
APLRIN	*APLLIB
APLRMV1	*APLLIB
APLRMV1A	*APLLIB
APLRMV2	*APLLIB
APLRMV2A	*APLLIB
APLROUT	*APLLIB
APLSNAM	*APLLIB
APLSNAMA	*APLLIB
APLTYPE	*APLLIB
APLUDAT	*APLLIB
APMM	NAAS:SSP
ARAT	NAAS:SSP
ARCOS	Elementary Fcn. Lib.
ARINIT	*LIBRARY
ARMTS	UNSP:LIBRARY
ARRAY	*LIBRARY
ARRAY	NAAS:SSP
ARRAY2	*LIBRARY
ARSIN	Elementary Fcn. Lib.
ASCEBC	Resident System
ASMTDEFS	Resident System
ASTATSUB	Resident System
ATAN	Elementary Fcn. Lib.
ATAN2	Elementary Fcn. Lib.
ATAN2#	Elementary Fcn. Lib.
ATEIG	NAAS:SSP
ATNTRP	*LIBRARY
ATRAP	UNSP:LIBRARY
ATRSTR	UNSP:LIBRARY
ATSE	NAAS:SSP
ATSG	NAAS:SSP
ATSM	NAAS:SSP
ATTACH	PL1SYM
ATTNT	Resident System
ATTNTRP	Resident System
AUTO	NAAS:SSP
AVCAL	NAAS:SSP
AVDAT	NAAS:SSP
AWXCMPA2	Resident System
AWXCMPB2	Resident System
AWXLIBR2	*ALGOLWXLIB
AWXSL001	*ALGOLWLIB
AWXSL001	*ALGOLWXLIB
AWXSL002	*ALGOLWLIB
AWXSL002	*ALGOLWXLIB
AWXSL003	*ALGOLWLIB
AWXSL003	*ALGOLWXLIB
AWXSL004	*ALGOLWLIB
AWXSL004	*ALGOLWXLIB
AWXSL005	*ALGOLWLIB
AWXSL005	*ALGOLWXLIB
AWXSL006	*ALGOLWLIB
AWXSL006	*ALGOLWXLIB
AWXSL007	*ALGOLWLIB
AWXSL007	*ALGOLWXLIB
AWXSL008	*ALGOLWLIB
AWXSL008	*ALGOLWXLIB
AWXSL009	*ALGOLWLIB
AWXSL009	*ALGOLWXLIB
AWXSL010	*ALGOLWLIB
AWXSL010	*ALGOLWXLIB
AWXSL011	*ALGOLWLIB
AWXSL011	*ALGOLWXLIB
AWXSL012	*ALGOLWLIB
AWXSL012	*ALGOLWXLIB
AWXSL013	*ALGOLWLIB
AWXSL013	*ALGOLWXLIB
AWXSL014	*ALGOLWLIB

AWXSL014	*ALGOLWXLIB
AWXSL015	*ALGOLWLIB
AWXSL015	*ALGOLWXLIB
BAIR	NAAS:OLDLIB
BAIR	OLD:LIBRARY
BAKVEC	NAAS:EISPACK
BALANC	NAAS:EISPACK
BALBAK	NAAS:EISPACK
BANDR	NAAS:EISPACK
BANDV	NAAS:EISPACK
BASIC0	Resident System
BATCH	PL1SYM
BCDIST	*CSMPLIB
BDTR	NAAS:SSP
BESEK0	NAAS:FUNPACK
BESEK1	NAAS:FUNPACK
BESJ	NAAS:SSP
BESK	NAAS:SSP
BESK0	NAAS:FUNPACK
BESK1	NAAS:FUNPACK
BESY	NAAS:SSP
BINEBCD	Resident System
BINEBCD2	Resident System
BISECT	NAAS:EISPACK
BISER	NAAS:SSP
BLKLTR	Resident System
BLOKLETR	Resident System
BLSTDEV	Resident System
BMLOCK	Resident System
BOOLE	*CSMPLIB
BOT	UNSP:LSLIPLIB
BOT	*SLIP
BOUND	NAAS:SSP
BQR	NAAS:EISPACK
BSINK	Resident System
BSRF	Resident System
BTC	*KDFLIB
BTD	*LIBRARY
BUFALLOC	Resident System
BUILD	*CSMPLIB
BUILDR	*CSMPLIB
B256	FORTRAN I/O Library
CABS	Elementary Fcn. Lib.
CADD	NAAS:SSP
CADLFT	*SLIP
CADLFT	UNSP:LSLIPLIB
CADNBT	*SLIP
CADNBT	UNSP:LSLIPLIB
CADNTP	*SLIP
CADNTP	UNSP:LSLIPLIB
CADRGT	*SLIP
CADRGT	UNSP:LSLIPLIB
CALC	Resident System
CALC#	Resident System
CANOR	NAAS:SSP
CANREPLY	Resident System
CASECONV	Resident System
CAXMB	NAAS:NAL
CBABK2	NAAS:EISPACK
CBAL	NAAS:EISPACK
CBS	NAAS:NAL
CCOS	Elementary Fcn. Lib.
CCPY	NAAS:SSP
CCUT	NAAS:SSP
CDABS	Elementary Fcn. Lib.
CDAXMB	NAAS:NAL
CDBS	NAAS:NAL
CDCOS	Elementary Fcn. Lib.
CDDVD#	Elementary Fcn. Lib.
CDEXP	Elementary Fcn. Lib.
CDILU	NAAS:NAL
CDIR	NAAS:NAL
CDLOG	Elementary Fcn. Lib.
CDLUD	NAAS:NAL
CDMPY#	Elementary Fcn. Lib.
CDRDUC	Resident System
CDSIN	Elementary Fcn. Lib.
CDSQRT	Elementary Fcn. Lib.
CDSTUC	Resident System
CDTR	NAAS:SSP
CDVD#	Elementary Fcn. Lib.
CEL1	NAAS:SSP
CEL2	NAAS:SSP
CENTRL	*CSMPLIB
CEXP	Elementary Fcn. Lib.
CFDUB	Resident System
CFGINF	Resident System
CG	NAAS:EISPACK
CH	NAAS:EISPACK
CHARIN	*KDFLIB
CHAROU	*KDFLIB
CHGFLG	Resident System
CHGFSZ	Resident System
CHGMBC	Resident System
CHISQ	NAAS:SSP
CHKACC	Resident System
CHKFDB	Resident System
CHKFDUB	Resident System
CHKFIL	Resident System
CHKFILE	Resident System
CHR1	*SLIP
CHR2	*SLIP
CHR3	*SLIP
CHR4	*SLIP
CHR5	*SLIP
CHR6	*SLIP

CHR7	*SLIP
CHR8	*SLIP
CILU	NAAS:NAL
CINT	NAAS:SSP
CINVIT	NAAS:EISPACK
CIR	NAAS:NAL
CKSTOR	*CSMPLIB
CLEANUP#	*IG
CLOG	Elementary Fcn. Lib.
CLOSE	*KDFLIB
CLOSEFIL	Resident System
CLOSFL	Resident System
CLSNET	Resident System
CLUD	NAAS:NAL
CMD	Resident System
CMDNOE	Resident System
CMPY#	Elementary Fcn. Lib.
CNFGINFO	Resident System
CNP	NAAS:SSP
CNPS	NAAS:SSP
CNTL	PL1SYM
CNTLNR	Resident System
CNTRL	Resident System
COLCT	*GASP
COMBAK	NAAS:EISPACK
COMC	*LIBRARY
COMHES	NAAS:EISPACK
COMLR	NAAS:EISPACK
COMLR2	NAAS:EISPACK
COMPACTI	*EXPLIB
COMPAR	*CSMPLIB
COMPL	*LIBRARY
COMPL	*CSMPLIB
COMQR	NAAS:EISPACK
COMQR2	NAAS:EISPACK
CONEND	UNSP:DIGLIB
CONLBL	UNSP:DIGLIB
CONSET	UNSP:DIGLIB
CONT	*SLIP
CONTIN	*CSMPLIB
CONTROL	Resident System
CONTUR	UNSP:DIGLIB
CONVT	NAAS:SSP
COPY	*PL360LIB
COPYTE	*KDFLIB
CORRE	NAAS:SSP
CORTB	NAAS:EISPACK
CORTH	NAAS:EISPACK
COS	Elementary Fcn. Lib.
COS#	Elementary Fcn. Lib.
COSH	Elementary Fcn. Lib.
COST	Resident System
COTAN	Elementary Fcn. Lib.
CPUTIME	PL1SYM
CREATE	Resident System
CREATE#	Resident System
CREPLY	Resident System
CRESVM	UNSP:LSLIPLIB
CROSS	NAAS:SSP
CS	NAAS:SSP
CSIN	Elementary Fcn. Lib.
CSMPEX	*CSMPLIB
CSMPST	*CSMPLIB
CSMPTR	*CSMPLIB
CSP	NAAS:SSP
CSPS	NAAS:SSP
CSQRT	Elementary Fcn. Lib.
CSRT	NAAS:SSP
CSTORE	*CSMPLIB
CSUM	NAAS:SSP
CTAB	NAAS:SSP
CTIE	NAAS:SSP
CTQQ	UNSP:DIGLIB
CTQQIN	UNSP:DIGLIB
CUINFO	Resident System
CURSEGM	*COBLIB
CVTOMR	Resident System
C1	*SLIP
C2	*SLIP
C3	*SLIP
C4	*SLIP
C5	*SLIP
C6	*SLIP
C7	*SLIP
C8	*SLIP
DACFI	NAAS:SSP
DAHI	NAAS:SSP
DAINT	OLD:LIBRARY
DAINT	NAAS:OLDLIB
DALI	NAAS:SSP
DAPCH	NAAS:SSP
DAPFS	NAAS:SSP
DAPLL	NAAS:SSP
DAPMM	NAAS:SSP
DARAT	NAAS:SSP
DARCOS	Elementary Fcn. Lib.
DARSIN	Elementary Fcn. Lib.
DATAN	*GASP
DATAN	Elementary Fcn. Lib.
DATAN2	Elementary Fcn. Lib.
DATAN2#	Elementary Fcn. Lib.
DATASK	*KDFLIB
DATAST	*CSMPLIB
DATSE	NAAS:SSP
DATSG	NAAS:SSP
DATSM	NAAS:SSP

DATUM	*SLIP
DATUM	UNSP:LSLIPLIB
DAXMB	NAAS:NAL
DBAR	NAAS:SSP
DBS	OLD:LIBRARY
DBS	NAAS:OLDLIB
DBS	NAAS:NAL
DBST	NAAS:OLDLIB
DBST	OLD:LIBRARY
DCAR	NAAS:SSP
DCEL1	NAAS:SSP
DCEL2	NAAS:SSP
DCLA	NAAS:SSP
DCNP	NAAS:SSP
DCNPS	NAAS:SSP
DCOS	Elementary Fcn. Lib.
DCOS#	Elementary Fcn. Lib.
DCOSH	Elementary Fcn. Lib.
DCOTAN	Elementary Fcn. Lib.
DCPY	NAAS:SSP
DCSP	NAAS:SSP
DCSPS	NAAS:SSP
DCVC	OLD:LIBRARY
DCVD	OLD:LIBRARY
DCVG	OLD:LIBRARY
DDAW	NAAS:FUNPACK
DDBAR	NAAS:SSP
DDCAR	NAAS:SSP
DDEF#	PL1SYM
DDET3	NAAS:SSP
DDET5	NAAS:SSP
DDGT3	NAAS:SSP
DEADSP	*CSMPLIB
DEBUG	*CSMPLIB
DEBUG#	OLD:LIBRARY
DEBUG#	FORTRAN I/O Library
DECOMP	NAAS:LIT
DEI	NAAS:FUNPACK
DELAY	*CSMPLIB
DELETE	UNSP:LSLIPLIB
DELETE	*SLIP
DELIEM	NAAS:FUNPACK
DELIE1	NAAS:FUNPACK
DELIKM	NAAS:FUNPACK
DELIK1	NAAS:FUNPACK
DELIPE	NAAS:FUNPACK
DELIPK	NAAS:FUNPACK
DELI1	NAAS:SSP
DELI2	NAAS:SSP
DERF	Elementary Fcn. Lib.
DERFC	Elementary Fcn. Lib.
DERIV	*CSMPLIB
DERROR	UNSP:LSLIPLIB
DERROR	*SLIP
DESTROY	Resident System
DESTRY	Resident System
DET3	NAAS:SSP
DET5	NAAS:SSP
DEVLST	Resident System
DEXP	Elementary Fcn. Lib.
DEXP#	Elementary Fcn. Lib.
DEXPEI	NAAS:FUNPACK
DFFT	NAAS:NAL
DFFTA	NAAS:NAL
DFFT2	NAAS:NAL
DFFT2A	NAAS:NAL
DFMCG	NAAS:SSP
DFMFP	NAAS:SSP
DFRAT	NAAS:SSP
DFS	NAAS:NAL
DGAMMA	Elementary Fcn. Lib.
DGELB	NAAS:SSP
DGELG	NAAS:SSP
DGELS	NAAS:SSP
DGSENS	*IG
DGT3	NAAS:SSP
DHARM	NAAS:SSP
DHEP	NAAS:SSP
DHEPS	NAAS:SSP
DHPCG	NAAS:SSP
DHPCL	NAAS:SSP
DILU	NAAS:NAL
DIMAG	Elementary Fcn. Lib.
DIOCS#	OLD:LIBRARY
DIOCS#	FORTRAN I/O Library
DIR	NAAS:NAL
DISCR	NAAS:SSP
DISMNT	Resident System
DISMOUNT	Resident System
DISTOK	UNSP:DIGLIB
DJELF	NAAS:SSP
DLAP	NAAS:SSP
DLAPS	NAAS:SSP
DLBVP	NAAS:SSP
DLEP	NAAS:SSP
DLEPS	NAAS:SSP
DLGAM	NAAS:SSP
DLGAMA	Elementary Fcn. Lib.
DLLSQ	NAAS:SSP
DLOG	Elementary Fcn. Lib.
DLOG#	Elementary Fcn. Lib.
DLOG10	Elementary Fcn. Lib.
DLS	*SLIP
DLUD	NAAS:NAL
DMATX	NAAS:SSP
DMAX1	Elementary Fcn. Lib.

DMCHB	NAAS:SSP
DMFGR	NAAS:SSP
DMFSD	NAAS:SSP
DMFSS	NAAS:SSP
DMIN1	Elementary Fcn. Lib.
DMLSS	NAAS:SSP
DMP	*SLIP
DMP	UNSP:LSLIPLIB
DMPCHR	UNSP:LSLIPLIB
DMPCLR	*SLIP
DMPERR	UNSP:LSLIPLIB
DMPERR	*SLIP
DMPER1	UNSP:LSLIPLIB
DMPER1	*SLIP
DMPFRE	UNSP:LSLIPLIB
DMPFRE	*SLIP
DMPINI	UNSP:LSLIPLIB
DMPLAV	*SLIP
DMPLAV	UNSP:LSLIPLIB
DMPLIS	UNSP:LSLIPLIB
DMPLNK	*SLIP
DMPLNK	UNSP:LSLIPLIB
DMPLST	*SLIP
DMPLST	UNSP:LSLIPLIB
DMPMRK	*SLIP
DMPMRK	UNSP:LSLIPLIB
DMPRC	NAAS:SSP
DMPRDR	*SLIP
DMPRDR	UNSP:LSLIPLIB
DMPRES	*SLIP
DMPRES	UNSP:LSLIPLIB
DMPSVM	UNSP:LSLIPLIB
DMPUBL	*SLIP
DMPUBL	UNSP:LSLIPLIB
DMTDS	NAAS:SSP
DPECN	NAAS:SSP
DPECS	NAAS:SSP
DPEONE	NAAS:FUNPACK
DPQFB	NAAS:SSP
DPRBM	NAAS:SSP
DPRQD	NAAS:SSP
DQATR	NAAS:SSP
DQA12	NAAS:SSP
DQA16	NAAS:SSP
DQA24	NAAS:SSP
DQA32	NAAS:SSP
DQA4	NAAS:SSP
DQA8	NAAS:SSP
DQG12	NAAS:SSP
DQG16	NAAS:SSP
DQG24	NAAS:SSP
DQG32	NAAS:SSP
DQG4	NAAS:SSP
DQG8	NAAS:SSP
DQHFE	NAAS:SSP
DQHFG	NAAS:SSP
DQHSE	NAAS:SSP
DQHSG	NAAS:SSP
DQH16	NAAS:SSP
DQH24	NAAS:SSP
DQH32	NAAS:SSP
DQH48	NAAS:SSP
DQH64	NAAS:SSP
DQH8	NAAS:SSP
DQL12	NAAS:SSP
DQL16	NAAS:SSP
DQL24	NAAS:SSP
DQL32	NAAS:SSP
DQL4	NAAS:SSP
DQL8	NAAS:SSP
DQSF	NAAS:SSP
DQTFE	NAAS:SSP
DQTFG	NAAS:SSP
DRAND	*GASP
DREAL	Elementary Fcn. Lib.
DRHARM	NAAS:SSP
DRKGS	NAAS:SSP
DROPIOER	*LIBRARY
DRS	*SLIP
DRSET	*GASP
DRSINT	*GASP
DRTMI	NAAS:SSP
DRTNI	NAAS:SSP
DRTWI	NAAS:SSP
DSE13	NAAS:SSP
DSE15	NAAS:SSP
DSE35	NAAS:SSP
DSFINI	UNSP:DIGLIB
DSG13	NAAS:SSP
DSIN	Elementary Fcn. Lib.
DSIN#	Elementary Fcn. Lib.
DSINH	Elementary Fcn. Lib.
DSINIT	UNSP:DIGLIB
DSINV	NAAS:SSP
DSQRT	Elementary Fcn. Lib.
DSQRT#	Elementary Fcn. Lib.
DSRDISPV	Resident System
DSR3270	Resident System
DTAN	Elementary Fcn. Lib.
DTANH	Elementary Fcn. Lib.
DTB	*LIBRARY
DTCNP	NAAS:SSP
DTCSP	NAAS:SSP
DTEAS	NAAS:SSP
DTEUL	NAAS:SSP
DTHEP	NAAS:SSP

DTLAP	NAAS:SSP
DTLEP	NAAS:SSP
DUMP	*LIBRARY
DUMP	OLD:LIBRARY
DVCHK	FORTRAN I/O Library
DVDQ	NAAS:LIT
DVDQG	NAAS:LIT
DVDQ1	NAAS:LIT
DYSSUB#	Resident System
D7090	*LIBRARY
D7090P	*LIBRARY
EBCASC	Resident System
EBCMASC	Resident System
EBX	*SLIP
ECHO	Resident System
EDIT	Resident System
EDITOR	Resident System
EIGEN	NAAS:SSP
ELAPSED	PL1SYM
ELI1	NAAS:SSP
ELI2	NAAS:SSP
ELMBAK	NAAS:EISPACK
ELMHES	NAAS:EISPACK
ELTRAN	NAAS:EISPACK
EMPTY	Resident System
EMPTYF	Resident System
EOR	*CSMPLIB
EQUAL	*SLIP
EQUC	*LIBRARY
EQUIV	*CSMPLIB
ERASAL	*LIBRARY
ERASE	*LIBRARY
ERF	Elementary Fcn. Lib.
ERFC	Elementary Fcn. Lib.
ERLNG	*GASP
EROUT	*GASP
ERRBUFFR	*PL360LIB
ERRCOM#	*IG
ERRER	*GASP
ERRMON#	Elementary Fcn. Lib.
ERROR	Resident System
ERROR	*PL1LIB
ERROR	*GASP
ERROR#	Resident System
ERRPRINT	*PL360LIB
ERRRR	*GASP
ERRTRA	NAAS:FUNPACK
ETW	*KDFLIB
EXIT	*LIBRARY
EXITER	*GASP
EXP	Elementary Fcn. Lib.
EXP#	Elementary Fcn. Lib.
EXPI	NAAS:SSP
EXSMO	NAAS:SSP
EXTEND	*LIBRARY
E7090	*LIBRARY
E7090P	*LIBRARY
F	*CSMPLIB
FACTR	NAAS:SSP
FCDXI#	Elementary Fcn. Lib.
FCNMON	NAAS:FUNPACK
FCNSW	*CSMPLIB
FCVAO	OLD:LIBRARY
FCVCO	OLD:LIBRARY
FCVEO	OLD:LIBRARY
FCVIO	OLD:LIBRARY
FCVLO	OLD:LIBRARY
FCVTHB	FORTRAN I/O Library
FCVTHB	OLD:LIBRARY
FCVZO	OLD:LIBRARY
FCXPI#	Elementary Fcn. Lib.
FDXPD#	Elementary Fcn. Lib.
FDXPI#	Elementary Fcn. Lib.
FHDRDISP	Resident System
FIGI	NAAS:EISPACK
FIGI2	NAAS:EISPACK
FILEM	*GASP
FILEMF	*GASP
FILEML	*GASP
FILEPTR	*APLLIB
FIND	*KDFLIB
FINDADR#	*IG
FINDC	*LIBRARY
FINDN	*GASP
FINDQ	*GASP
FINDST	*LIBRARY
FINFO	PL1SYM
FINSRT	*GASP
FIOCS#	OLD:LIBRARY
FIOEND	FORTRAN I/O Library
FIXPI#	Elementary Fcn. Lib.
FMCG	NAAS:SSP
FMFP	NAAS:SSP
FNAMETRT	Resident System
FORIF	NAAS:SSP
FORIT	NAAS:SSP
FORMAT	*KDFLIB
FPSECT	Resident System
FRAT	NAAS:SSP
FRDNL#	FORTRAN I/O Library
FRDNL#	OLD:LIBRARY
FREAD	*LIBRARY
FREED	Resident System
FREEFD	Resident System
FREESP	Resident System
FREESPAC	Resident System

ID	UNSP:LSLIPLIB
ID	*SLIP
IDATUM	UNSP:LSLIPLIB
IDATUM	*SLIP
IDW	UNSP:LSLIPLIB
IED	*LIBRARY
IEDMK	*LIBRARY
IERRCOOO	*COBLIB
IF	Resident System
IGATTB	*IG
IGBGNO	*IG
IGBGNS	*IG
IGC	*LIBRARY
IGCTNS	*IG
IGCTRL	*IG
IGCVTC	*IG
IGDA	*IG
IGDELO	*IG
IGDELS	*IG
IGDR	*IG
IGDRON	*IG
IGENDO	*IG
IGENDS	*IG
IGFMT	*IG
IGFMTH	*IG
IGHSPO	*IG
IGHUE	*IG
IGINIT	*IG
IGINT	*IG
IGLIKE	*IG
IGLOAD	*IG
IGMA	*IG
IGMR	*IG
IGPDSW	*IG
IGPFPF	*IG
IGPICK	*IG
IGPIKC	*IG
IGPIKN	*IG
IGPIKS	*IG
IGPUTO	*IG
IGSENS	*IG
IGSYM	*IG
IGTEXT	*IG
IGTRAN	*IG
IGTXT	*IG
IGTXTH	*IG
IGUSER	*IG
IGVEC	*IG
IGVWPT	*IG
IGXYIN	*IG
IHCFDUMP	OLD:LIBRARY
IHCFEXIT	*LIBRARY
IHCIBERH	*LIBRARY
IHCLOGIC	*LIBRARY
IHCNAMEL	OLD:LIBRARY
IHEABN	*PL1LIB
IHEABND	*PL1LIB
IHELTT	*PL1LIB
IHEMAIN	*PL1LIB
IHENTRY	*APLLIB
IHESAP	*PL1LIB
IHESAPA	*PL1LIB
IHESAPB	*PL1LIB
IHESAPC	*PL1LIB
IHESAPD	*PL1LIB
IHESAPE	*PL1LIB
IHESAPF	*PL1LIB
IHESIZ	*PL1LIB
IHESIZE	*PL1LIB
IHESPRT	*PL1LIB
IHESRCM	*APLLIB
IHETAB	*PL1LIB
IHETABS	*PL1LIB
IHE.....	PL1SYM
IHI.....	*ALGOLLIB
ILB.....	*COBLIB
IMPL	*CSMPLIB
IMPLST	*CSMPLIB
IMPULS	*CSMPLIB
IMTQLV	NAAS:EISPACK
IMTQL1	NAAS:EISPACK
IMTQL2	NAAS:EISPACK
IMVC	*LIBRARY
INBASI	*KDFLIB
INC	*LIBRARY
INHALT	*SLIP
INITAS	UNSP:LSLIPLIB
INITAS	*SLIP
INITCNT	Resident System
INITLOCK	Resident System
INITLZ	*CSMPLIB
INITRD	UNSP:LSLIPLIB
INITRD	*SLIP
INLSTL	UNSP:LSLIPLIB
INLSTL	*SLIP
INLSTR	UNSP:LSLIPLIB
INLSTR	*SLIP
INSW	*CSMPLIB
INTERC	*KDFLIB
INTERFAC	*ALGOLLIB
INTGER	UNSP:LSLIPLIB
INTGER	*SLIP
INTGST	*CSMPLIB
INTRAN	*CSMPLIB
INTRP	*CSMPLIB
INTSUBS	PL1SYM

LOADINFO	Resident System
LOC	NAAS:SSP
LOCAT	*GASP
LOCK	Resident System
LOCT	UNSP:LSLIPLIB
LOCT	*SLIP
LODMAP	Resident System
LOFRDR	UNSP:LSLIPLIB
LOFRDR	*SLIP
LOPEN	*LIBRARY
LOR	*LIBRARY
LOUT$F	*SIM2LIB
LPNTR	UNSP:LSLIPLIB
LPNTR	*SLIP
LPURGE	UNSP:LSLIPLIB
LPURGE	*SLIP
LRD	NAAS:OLDLIB
LRD	OLD:LIBRARY
LRDELIM$	*SIM2LIB
LRDRCP	UNSP:LSLIPLIB
LRDRCP	*SLIP
LRDROV	UNSP:LSLIPLIB
LRDROV	*SLIP
LS	*SLIP
LSDELIM$	*SIM2LIB
LSSCPY	UNSP:LSLIPLIB
LSSCPY	*SLIP
LSTDMP	*SLIP
LSTEQL	UNSP:LSLIPLIB
LSTEQL	*SLIP
LSTMRK	UNSP:LSLIPLIB
LSTMRK	*SLIP
LSTPRO	UNSP:LSLIPLIB
LSTPRO	*SLIP
LTHERE	UNSP:LSLIPLIB
LTSLE	OLD:LIBRARY
LTSLE	NAAS:OLDLIB
LVLRVT	*SLIP
LVLRVT	UNSP:LSLIPLIB
LVLRV1	*SLIP
LVLRV1	UNSP:LSLIPLIB
LXOR	*LIBRARY
MADATR	UNSP:LSLIPLIB
MADATR	*SLIP
MADD	NAAS:SSP
MADLFT	UNSP:LSLIPLIB
MADLFT	*SLIP
MADNBT	UNSP:LSLIPLIB
MADNBT	*SLIP
MADNTP	UNSP:LSLIPLIB
MADNTP	*SLIP
MADOV	*SLIP
MADRGT	UNSP:LSLIPLIB
MADRGT	*SLIP
MAINEX	*CSMPLIB
MAKEDL	UNSP:LSLIPLIB
MAKEDL	*SLIP
MAPRNT	NAAS:LIT
MASCEBC	Resident System
MATA	NAAS:SSP
MAXLEN	UNSP:PL1LIB
MAX0	Elementary Fcn. Lib.
MAX1	Elementary Fcn. Lib.
MCHB	NAAS:SSP
MCPY	NAAS:SSP
MEANQ	NAAS:SSP
MESSAGE	Resident System
MFGR	NAAS:SSP
MFSD	NAAS:SSP
MFSS	NAAS:SSP
MFUN	NAAS:SSP
MILNE	*CSMPLIB
MINFIT	NAAS:EISPACK
MINV	NAAS:SSP
MIN0	Elementary Fcn. Lib.
MIN1	Elementary Fcn. Lib.
MISR	NAAS:SSP
MLEFT	*CSMPLIB
MLSS	NAAS:SSP
MMACST	*CSMPLIB
MNETRTN	Resident System
MODFTBLE	Resident System
MOMEN	NAAS:SSP
MONERR	NAAS:FUNPACK
MONTR	*GASP
MOUNT	Resident System
MOUNTCMD	Resident System
MOVE	*CSMPLIB
MOVEC	*LIBRARY
MPAIR	NAAS:SSP
MPRC	NAAS:SSP
MPRD	NAAS:SSP
MRIGHT	*CSMPLIB
MRK	UNSP:LSLIPLIB
MRK	*SLIP
MRKGET	*SLIP
MRKLSS	UNSP:LSLIPLIB
MRKLSS	*SLIP
MRKLST	UNSP:LSLIPLIB
MRKLST	*SLIP
MRKW	UNSP:LSLIPLIB
MRXA	Resident System
MSG	Resident System
MSTR	NAAS:SSP
MSUB	NAAS:SSP
MTDLST	UNSP:LSLIPLIB

MTDLST	*SLIP
MTDS	NAAS:SSP
MTLIST	UNSP:LSLIPLIB
MTLIST	*SLIP
MTRA	NAAS:SSP
MTS	Resident System
MTS#	Resident System
MTSCMD	Resident System
MTSCMD#	Resident System
MULTR	NAAS:SSP
NAME	*CSMPLIB
NAMEDL	UNSP:LSLIPLIB
NAMEDL	*SLIP
NAMTST	UNSP:LSLIPLIB
NAMTST	*SLIP
NAND	*CSMPLIB
NATSEE	NAAS:FUNPACK
NATSEI	NAAS:FUNPACK
NATSEK	NAAS:FUNPACK
NATSK0	NAAS:FUNPACK
NATSK1	NAAS:FUNPACK
NDTR	NAAS:SSP
NDTRI	NAAS:SSP
NEWBOT	UNSP:LSLIPLIB
NEWBOT	*SLIP
NEWLIN	*KDFLIB
NEWTOP	UNSP:LSLIPLIB
NEWTOP	*SLIP
NEWVAL	UNSP:LSLIPLIB
NEWVAL	*SLIP
NEXTKEY	PL1SYM
NLBACK	NAAS:LIT
NLFGEN	*CSMPLIB
NLSYS	NAAS:LIT
NOATVL	UNSP:LSLIPLIB
NOATVL	*SLIP
NOCENT	*CSMPLIB
NOR	*CSMPLIB
NOT	*CSMPLIB
NOTE	Resident System
NOTE#	Resident System
NPOSN	*GASP
NROOT	NAAS:SSP
NTOBCD	*CSMPLIB
NUCELL	UNSP:LSLIPLIB
NUCELL	*SLIP
NULSTL	UNSP:LSLIPLIB
NULSTL	*SLIP
NULSTR	UNSP:LSLIPLIB
NULSTR	*SLIP
NUMDEV	Resident System
NUMER	*CSMPLIB
NXTLFT	UNSP:LSLIPLIB
NXTLFT	*SLIP
NXTRGT	UNSP:LSLIPLIB
NXTRGT	*SLIP
OMIT	*LIBRARY
ONE@ATIM	*LIBRARY
OPEN	*KDFLIB
OR	*LIBRARY
ORDER	NAAS:SSP
ORTBAK	NAAS:EISPACK
ORTHES	NAAS:EISPACK
ORTRAN	NAAS:EISPACK
OSGRDT	*LIBRARY
OSINT	*SPITLIB
OTPUT	*GASP
OUTBAS	*KDFLIB
OUTPUT	*KDFLIB
OUTSW	*CSMPLIB
OVERFL	FORTRAN I/O Library
OWNCONVR	*LIBRARY
PADD	NAAS:SSP
PADDM	NAAS:SSP
PAR	UNSP:LIBRARY
PARMT	UNSP:LSLIPLIB
PARMTN	*SLIP
PARMT2	UNSP:LSLIPLIB
PARMT2	*SLIP
PARROW	*PLOTSYS
PARRO2	*PLOTSYS
PAXFMT	*PLOTSYS
PAXFM2	*PLOTSYS
PAXIS	*PLOTSYS
PAXSCL	*PLOTSYS
PAXTIC	*PLOTSYS
PAXTTL	*PLOTSYS
PAXVAL	*PLOTSYS
PBOUND	*PLOTSYS
PCCLOSE	*LIBRARY
PCEPT	*PLOTSYS
PCIRCL	*PLOTSYS
PCLA	NAAS:SSP
PCLD	NAAS:SSP
PCLOSE	*LIBRARY
PCOPEN	*LIBRARY
PCTRLN	*PLOTSYS
PCTRL2	*PLOTSYS
PDER	NAAS:SSP
PDFSYM	*PLOTSYS
PDIV	NAAS:SSP
PDP8RTN	Resident System
PDSHLN	*PLOTSYS
PDSHL2	*PLOTSYS
PDSYMB	*PLOTSYS
PDTAB	*PLOTSYS

PLTXMX	*IG
PL1ADR	PL1SYM
PL1RC	PL1SYM
PL1SYM	Resident System
PMPY	NAAS:SSP
PNORM	NAAS:SSP
PNUMBR	*PLOTSYS
POFRES	*PLOTSYS
POFRST	*PLOTSYS
POFSAV	*PLOTSYS
POINT	Resident System
POINT	NAAS:SSP
POINT#	Resident System
POLGRD	*PLOTSYS
POLRT	NAAS:SSP
PONRST	*PLOTSYS
POPBOT	UNSP:LSLIPLIB
POPBOT	*SLIP
POPEN	*LIBRARY
POPRDR	UNSP:LSLIPLIB
POPSID	*PLOTSYS
POPTOP	UNSP:LSLIPLIB
POPTOP	*SLIP
PPRCN	NAAS:SSP
PPSYM	*PLOTSYS
PQFB	NAAS:SSP
PQSD	NAAS:SSP
PRAXIS	NAAS:LIT
PRBM	NAAS:SSP
PRCHAR	*LIBRARY
PREAD	UNSP:PL1LIB
PREAD1	UNSP:PL1LIB
PREND	*LIBRARY
PRESRV	UNSP:LSLIPLIB
PRESRV	*SLIP
PRFIT	NAAS:LIT
PRHIST	*GASP
PRLIN	NAAS:LIT
PRLSTS	*SLIP
PRLSTS	UNSP:LSLIPLIB
PRMIN	NAAS:LIT
PRNTQ	*GASP
PROBT	NAAS:SSP
PRODQ	*GASP
PRPLOT	*LIBRARY
PRPRIN	NAAS:LIT
PRQD	NAAS:SSP
PRQUAD	NAAS:LIT
PRSORT	NAAS:LIT
PRSTER	*PLOTSYS
PSBSP	*PLOTSYS
PSCALE	*PLOTSYS
PSCAL1	*PLOTSYS
PSCNIF	*PLOTSYS
PSHRDR	UNSP:LSLIPLIB
PSIZE	*PLOTSYS
PSMGEN	*PLOTSYS
PSTART	*IG
PSTART	*PLOTSYS
PSUB	NAAS:SSP
PSYMB	*PLOTSYS
PSYMFG	*IG
PSYMFG	*PLOTSYS
PSYMLN	*PLOTSYS
PSYMPT	*PLOTSYS
PSYMSV	*PLOTSYS
PSYSYM	*PLOTSYS
PTDST2	*PLOTSYS
PTFMVL	*GPSSLIB
PTFPVL	*GPSSLIB
PTFSVL	*GPSSLIB
PTHAXS	*PLOTSYS
PTHMVL	*GPSSLIB
PTHPVL	*GPSSLIB
PTHSVL	*GPSSLIB
PTRUC	Resident System
PTSYMB	*PLOTSYS
PUBDMP	*SLIP
PULSE	*CSMPLIB
PUNCH	*PL360LIB
PUNUC	Resident System
PUTENT	*GASP
PUTIHC	OLD:LIBRARY
PUTIHC	FORTRAN I/O Library
PUTRDL	UNSP:LSLIPLIB
PVAL	NAAS:SSP
PVSUB	NAAS:SSP
PWRIT	*IG
PWRIT	*PLOTSYS
PWRITE	UNSP:PL1LIB
PWRITE1	UNSP:PL1LIB
PWTMAX	*IG
PWTMAX	*PLOTSYS
PXABS	*IG
PXABS	*PLOTSYS
PXFACT	*PLOTSYS
PXMARG	*PLOTSYS
PXMARG	*IG
PXMIN	*PLOTSYS
PXORG	*PLOTSYS
PXREL	*PLOTSYS
PYABS	*IG
PYABS	*PLOTSYS
PYFACT	*PLOTSYS
PYMIN	*PLOTSYS
PYORG	*PLOTSYS

PYREL	*PLOTSYS
QATR	NAAS:SSP
QA10	NAAS:SSP
QA2	NAAS:SSP
QA3	NAAS:SSP
QA4	NAAS:SSP
QA5	NAAS:SSP
QA6	NAAS:SSP
QA7	NAAS:SSP
QA8	NAAS:SSP
QA9	NAAS:SSP
QCLOSE	*LIBRARY
QCNTRL	*LIBRARY
QDIV	OLD:LIBRARY
QDIV	NAAS:OLDLIB
QFREEUCB	*LIBRARY
QFRUCB	*LIBRARY
QGET	*LIBRARY
QGETUCB	*LIBRARY
QGTUCB	*LIBRARY
QG10	NAAS:SSP
QG2	NAAS:SSP
QG3	NAAS:SSP
QG4	NAAS:SSP
QG5	NAAS:SSP
QG6	NAAS:SSP
QG7	NAAS:SSP
QG8	NAAS:SSP
QG9	NAAS:SSP
QHFE	NAAS:SSP
QHFG	NAAS:SSP
QHSE	NAAS:SSP
QHSG	NAAS:SSP
QH10	NAAS:SSP
QH2	NAAS:SSP
QH3	NAAS:SSP
QH4	NAAS:SSP
QH5	NAAS:SSP
QH6	NAAS:SSP
QH7	NAAS:SSP
QH8	NAAS:SSP
QH9	NAAS:SSP
QL10	NAAS:SSP
QL2	NAAS:SSP
QL3	NAAS:SSP
QL4	NAAS:SSP
QL5	NAAS:SSP
QL6	NAAS:SSP
QL7	NAAS:SSP
QL8	NAAS:SSP
QL9	NAAS:SSP
QNTZR	*CSMPLIB
QOPEN	*LIBRARY
QPSECT	Resident System
QPUT	*LIBRARY
QQSV	*SLIP
QSAM	*LIBRARY
QSAMP	*LIBRARY
QSF	NAAS:SSP
QTEST	NAAS:SSP
QTFE	NAAS:SSP
QTFG	NAAS:SSP
QTHERE	UNSP:LSLIPLIB
QTR1	*SLIP
QTR2	*SLIP
QTR3	*SLIP
QTR4	*SLIP
QUIT	Resident System
QUIT$	PL1SYM
QZHES	NAAS:EISPACK
QZIT	NAAS:EISPACK
QZVAL	NAAS:EISPACK
QZVEC	NAAS:EISPACK
Q1	*SLIP
Q2	*SLIP
Q3	*SLIP
Q4	*SLIP
RAEVS	*SIM2LIB
RADD	NAAS:SSP
RAMP	*CSMPLIB
RAND	PL1SYM
RANDU	NAAS:SSP
RANDUM	NAAS:LIT
RANG1	*CSMPLIB
RANG2	*CSMPLIB
RANK	NAAS:SSP
RARCCOS$	*SIM2LIB
RARCSIN$	*SIM2LIB
RARCTAN$	*SIM2LIB
RATENBR	Resident System
RATQR	NAAS:EISPACK
RATRAP	UNSP:LIBRARY
RBETA$F	*SIM2LIB
RBINOMIA	*SIM2LIB
RBLOCK$R	*SIM2LIB
RCALL	*LIBRARY
RCELL	*SLIP
RCELL	UNSP:LSLIPLIB
RCLOSE	*LIBRARY
RCLS$R	*SIM2LIB
RCOS$F	*SIM2LIB
RCPY	NAAS:SSP
RCRE$F	*SIM2LIB
RCUIB$R	*SIM2LIB
RCUT	NAAS:SSP
RDATE$F	*SIM2LIB

RDAY$F	*SIM2LIB
RDENT	*GASP
RDIM$F	*SIM2LIB
RDLSTA	UNSP:LSLIPLIB
RDLSTA	*SLIP
READ	Resident System
READ	*PL360LIB
READ	*KDFLIB
READ#	Resident System
READAR	*KDFLIB
READBFR	*LIBRARY
READBI	*KDFLIB
READBO	*KDFLIB
READE	Resident System
REALL	*SLIP
REALS	UNSP:LSLIPLIB
REALS	*SLIP
REBAK	NAAS:EISPACK
REBAKB	NAAS:EISPACK
RECP	NAAS:SSP
RECT	*CSMPLIB
RECURS	UNSP:LSLIPLIB
REDUC	NAAS:EISPACK
REDUC2	NAAS:EISPACK
REED	*SLIP
REED	UNSP:LSLIPLIB
REFIELD$	*SIM2LIB
RENAME	Resident System
RENUMB	Resident System
RERLANG$	*SIM2LIB
RERR$R	*SIM2LIB
RERUN	Resident System
RERUN	*CSMPLIB
RESTOR	UNSP:LSLIPLIB
RESTOR	*SLIP
RETLNR	Resident System
REWIND	*LIBRARY
REWIND	*KDFLIB
REWIND#	Resident System
REXP$F	*SIM2LIB
REXPONEN	*SIM2LIB
RFATAL	*SIM2LIB
RFINFO	PL1SYM
RFRAC$F	*SIM2LIB
RFREE$R	*SIM2LIB
RG	NAAS:EISPACK
RGAMMA$F	*SIM2LIB
RGAMMAJ$	*SIM2LIB
RGG	NAAS:EISPACK
RGUIB$R	*SIM2LIB
RHARM	NAAS:SSP
RHOUR$F	*SIM2LIB
RIN$F	*SIM2LIB
RINT	NAAS:SSP
RISTEP$F	*SIM2LIB
RITOA$F	*SIM2LIB
RIXP$F	*SIM2LIB
RKGS	NAAS:SSP
RKS	*CSMPLIB
RK1	NAAS:SSP
RK2	NAAS:SSP
RLIN$F	*SIM2LIB
RLOGEF	*SIM2LIB
RLOG$NOR	*SIM2LIB
RLOG10	*SIM2LIB
RLOGN	*GASP
RLS1$R	*SIM2LIB
RLS2$R	*SIM2LIB
RLT1$R	*SIM2LIB
RLT2$R	*SIM2LIB
RMACST	*CSMPLIB
RMINUTE$	*SIM2LIB
RMODE$F	*SIM2LIB
RMONTH$F	*SIM2LIB
RMOVE	*GASP
RMOVEF	*GASP
RMOVEL	*GASP
RMTS	UNSP:LIBRARY
RNDAY$F	*SIM2LIB
RNDGEN	*CSMPLIB
RNORM	*GASP
RNORMAL$	*SIM2LIB
ROBEY$R	*SIM2LIB
ROPEN	*LIBRARY
RORIGIN$	*SIM2LIB
ROUT$F	*SIM2LIB
RPOISSON	*SIM2LIB
RRANDI$F	*SIM2LIB
RRANDOM$	*SIM2LIB
RRDA$R	*SIM2LIB
RRDB$R	*SIM2LIB
RRDC$R	*SIM2LIB
RRDD$R	*SIM2LIB
RRDE$R	*SIM2LIB
RRDI$R	*SIM2LIB
RRDL$R	*SIM2LIB
RRDR$R	*SIM2LIB
RRDS$R	*SIM2LIB
RRDT$R	*SIM2LIB
RREL$R	*SIM2LIB
RRES$R	*SIM2LIB
RRFA$R	*SIM2LIB
RRFD$R	*SIM2LIB
RRFI$R	*SIM2LIB
RRIRV$R	*SIM2LIB
RRLR$R	*SIM2LIB

SETDSR	FORTRAN I/O Library
SETDSR	OLD:LIBRARY
SETEMP	*GASP
SETFRVAR	*LIBRARY
SETGRE	OLD:LIBRARY
SETIGFDP	*IG
SETIME	Resident System
SETIND	UNSP:LSLIPLIB
SETIND	*SLIP
SETIOERR	Resident System
SETIOHER	*LIBRARY
SETKEY	Resident System
SETLCK	Resident System
SETLEN	UNSP:PL1LIB
SETLIO	Resident System
SETLNR	Resident System
SETLOG	*LIBRARY
SETMKW	UNSP:LSLIPLIB
SETMRK	*SLIP
SETMRK	UNSP:LSLIPLIB
SETPFX	Resident System
SETRAC	*SLIP
SETSTA	FORTRAN I/O Library
SETSTA	OLD:LIBRARY
SETUP	NAAS:LIT
SETWK	*GASP
SE13	NAAS:SSP
SE15	NAAS:SSP
SE35	NAAS:SSP
SFFT	NAAS:NAL
SFFTA	NAAS:NAL
SFFT2	NAAS:NAL
SFFT2A	NAAS:NAL
SFS	NAAS:NAL
SGDINFO	UNSP:SPITLIB
SGDINFO2	UNSP:SPITLIB
SGETFD	UNSP:SPITLIB
SGLINT	OLD:LIBRARY
SGLINT	NAAS:OLDLIB
SGUINFO	UNSP:SPITLIB
SGUINFO2	UNSP:SPITLIB
SGUINFO3	UNSP:SPITLIB
SGUINFO4	UNSP:SPITLIB
SG13	NAAS:SSP
SHFTL	*LIBRARY
SHFTR	*LIBRARY
SHIFT	*CSMPLIB
SHIN	*SLIP
SICI	NAAS:SSP
SIGNOFF	PL1SYM
SIGNT	NAAS:SSP
SILU	NAAS:NAL
SIMOUT	*CSMPLIB
SIMP	*CSMPLIB
SIMQ	NAAS:SSP
SIN	Elementary Fcn. Lib.
SIN#	Elementary Fcn. Lib.
SINCOS	*PLOTSYS
SINE	*CSMPLIB
SINH	Elementary Fcn. Lib.
SINV	NAAS:OLDLIB
SINV	OLD:LIBRARY
SINV	NAAS:SSP
SINV1	OLD:LIBRARY
SINV1	NAAS:OLDLIB
SINV2	OLD:LIBRARY
SINV2	NAAS:OLDLIB
SIOC	Resident System
SIOC#	Resident System
SIOCP	Resident System
SIOERR	*LIBRARY
SIR	NAAS:NAL
SKIP	*KDFLIB
SKIP	*LIBRARY
SLE#	NAAS:OLDLIB
SLE#	OLD:LIBRARY
SLE1	NAAS:OLDLIB
SLE1	OLD:LIBRARY
SLE2	NAAS:OLDLIB
SLE2	OLD:LIBRARY
SLE3	NAAS:OLDLIB
SLE3	OLD:LIBRARY
SLE4	NAAS:OLDLIB
SLE4	OLD:LIBRARY
SLIPPRIM	*SLIP
SLITE	FORTRAN I/O Library
SLITET	FORTRAN I/O Library
SLPDMP	*SLIP
SLPDMP	UNSP:LSLIPLIB
SLRD	NAAS:OLDLIB
SLRD	OLD:LIBRARY
SLUD	NAAS:NAL
SMIRN	NAAS:SSP
SMO	NAAS:SSP
SMPY	NAAS:SSP
SMTSCMD	UNSP:SPITLIB
SNAP	*PL1LIB
SNOOP	UNSP:LIBRARY
SNOOP	UNSP:SPITLIB
SNS	UNSP:SPITLIB
SNS2	UNSP:SPITLIB
SOLVE	NAAS:LIT
SORT	*LIBRARY
SORTEA	*LIBRARY
SORTE1	*COBLIB
SORTE4	*COBLIB

SORTE9	*COBLIB
SORT1	*LIBRARY
SORT2	*LIBRARY
SORT3	*LIBRARY
SPACE	*KDFLIB
SPELCK	Resident System
SPELER	UNSP:LIBRARY
SPELLCHK	Resident System
SPIE	*LIBRARY
SPIE	*PL1LIB
SPIRLCS	Resident System
SPITATTN	UNSP:SPITLIB
SPITBOL	Resident System
SPLINE	NAAS:LIT
SPLIT	*CSMPLIB
SPLITR	*CSMPLIB
SPLIT1	*CSMPLIB
SPRINT	Resident System
SPRINT#	Resident System
SPUNCH	Resident System
SPUNCH#	Resident System
SQIN	*SLIP
SQOUT	*SLIP
SQRT	Elementary Fcn. Lib.
SQRT#	Elementary Fcn. Lib.
SRANK	NAAS:SSP
SRATE	NAAS:SSP
SREAD	UNSP:SPITLIB
SRMA	NAAS:SSP
SSETPFX	UNSP:SPITLIB
SSFMT	Resident System
SSLE	NAAS:OLDLIB
SSLE	OLD:LIBRARY
SSNOOP	UNSP:SPITLIB
SSNS	UNSP:SPITLIB
SSNS2	UNSP:SPITLIB
SSRTN	Resident System
SSTACLS	Resident System
SSTOR	Resident System
SSTORE	*CSMPLIB
SSUB	NAAS:SSP
STARTF	FORTRAN I/O Library
STAT$	Resident System
STATBUFF	Resident System
STATUS	*CSMPLIB
STDDMP	Resident System
STDTV	Resident System
STEP	*CSMPLIB
STIMER	*LIBRARY
STLNKW	UNSP:LSLIPLIB
STORE	*CSMPLIB
STORST	*CSMPLIB
STPLT1	*LIBRARY
STPLT2	*LIBRARY
STPRG	NAAS:SSP
STRDAT	*SLIP
STRDAT	UNSP:LSLIPLIB
STRDIR	*SLIP
STRIND	UNSP:LSLIPLIB
STRIND	*SLIP
STRUST	*CSMPLIB
SUBMX	NAAS:SSP
SUBSBT	UNSP:LSLIPLIB
SUBSBT	*SLIP
SUBST	UNSP:LSLIPLIB
SUBST	NAAS:SSP
SUBST	*SLIP
SUBSTP	UNSP:LSLIPLIB
SUBSTP	*SLIP
SUMQ	*GASP
SUMRY	*GASP
SVD	NAAS:EISPACK
SVXOS10	*ALGOLLIB
SVXOS11	*ALGOLLIB
SVXOS14	*ALGOLLIB
SVXOS19	*ALGOLLIB
SVXOS20	*ALGOLLIB
SVXOS23	*ALGOLLIB
SVXOS35	*ALGOLLIB
SVXOS4	*ALGOLLIB
SVXOS5	*ALGOLLIB
SVXOS6	*ALGOLLIB
SVXOS64	*ALGOLLIB
SVXOS7	*ALGOLLIB
SVXOS8	*ALGOLLIB
SVXOS9	*ALGOLLIB
SVXOS999	*ALGOLLIB
SWRITE	UNSP:SPITLIB
SYSDEFS	Resident System
SYSERR	*PL1LIB
SYSHELP	*GPSSLIB
SYSINIT	*PL360LIB
SYSTEM	*PL1LIB
SYSTEM	Resident System
SYSTEM#	Resident System
SYSTERM	*PL360LIB
SYSVMDF	*APLLIB
SYSVMFR	*APLLIB
TAB	*KDFLIB
TAB1	NAAS:SSP
TAB2	NAAS:SSP
TALLY	NAAS:SSP
TAN	Elementary Fcn. Lib.
TANH	Elementary Fcn. Lib.
TAPERTN	Resident System
TAPEUC	Resident System

TCNP	NAAS:SSP
TCSP	NAAS:SSP
TEAS	NAAS:SSP
TEL2	Resident System
TERM	*SLIP
TERM	UNSP:LSLIPLIB
TEST	*KDFLIB
TETRA	NAAS:SSP
TEUL	NAAS:SSP
TFINFO	PL1SYM
THEP	NAAS:SSP
TICALL	*LIBRARY
TIE	NAAS:SSP
TIME	Resident System
TIME_OF_	*EXPLIB
TIMNTRP	Resident System
TIMTRP	Resident System
TINVIT	NAAS:EISPACK
TLAP	NAAS:SSP
TLEP	NAAS:SSP
TMST	*GASP
TOP	UNSP:LSLIPLIB
TOP	*SLIP
TPRD	NAAS:SSP
TPRDUC	Resident System
TPWRUC	Resident System
TQLRAT	NAAS:EISPACK
TQL1	NAAS:EISPACK
TQL2	NAAS:EISPACK
TRACE	NAAS:SSP
TRACER	Resident System
TRACER#	Resident System
TRANSA	*CSMPLIB
TRANTB	Resident System
TRAP	UNSP:LIBRARY
TRAPZ	*CSMPLIB
TRBAK1	NAAS:EISPACK
TRBAK3	NAAS:EISPACK
TRED1	NAAS:EISPACK
TRED2	NAAS:EISPACK
TRED3	NAAS:EISPACK
TRIDIB	NAAS:EISPACK
TRNC	*LIBRARY
TRNST	*LIBRARY
TRRSTR	UNSP:LIBRARY
TRUNC	Resident System
TSEP	OLD:LIBRARY
TSEP	NAAS:OLDLIB
TSEPC	OLD:LIBRARY
TSEPC	NAAS:OLDLIB
TSEP1	NAAS:OLDLIB
TSEP1	OLD:LIBRARY
TSFO	Resident System
TSKFMT	Resident System
TSTURM	NAAS:EISPACK
TSYM	*SLIP
TSYM#	*SLIP
TTEST	NAAS:SSP
TTIMER	*LIBRARY
TWAIT	*LIBRARY
TWOAV	NAAS:SSP
UC3330	Resident System
UMLOAD	Resident System
UMLOADFG	Resident System
UMLOADFS	Resident System
UMLOADNF	Resident System
UMLOADRP	Resident System
UND	*CSMPLIB
UNFRM	*GASP
UNLCK	Resident System
UNLDF	FORTRAN I/O Library
UNLK	Resident System
UNLOAD	Resident System
URAND	*LIBRARY
URAND	NAAS:OLDLIB
USERID	PL1SYM
UTEST	NAAS:SSP
UTSLE	OLD:LIBRARY
UTSLE	NAAS:OLDLIB
VARMX	NAAS:SSP
VCPRNT	NAAS:LIT
VIRMIN	*APLLIB
VIRMOUT	*APLLIB
VISIT	UNSP:LSLIPLIB
VISIT	*SLIP
VISTIN	UNSP:LSLIPLIB
WRITBF	Resident System
WRITE	*PL360LIB
WRITE	Resident System
WRITE	*KDFLIB
WRITE#	Resident System
WRITEA	*KDFLIB
WRITEB	*KDFLIB
WRITEBUF	Resident System
WRITET	*KDFLIB
WTEST	NAAS:SSP
XABS	*PLOTSYS
XCLR	*SIM2LIB
XCPY	NAAS:SSP
XCTL	Resident System
XCTLF	FORTRAN I/O Library
XFIX	*SIM2LIB
XFLT	*SIM2LIB
XFRE	*SIM2LIB
XIDE	*SIM2LIB
XINR	*SIM2LIB

XLAR	*SIM2LIB
XMAS	*SIM2LIB
XMASK	*SLIP
XOR	*LIBRARY
XPLSM	*EXPLIB
XPLSM@SP	*EXPLIB
XREC	*SIM2LIB
XREL	*PLOTSYS
XRET	*SIM2LIB
XSE1	*SIM2LIB
XSE2	*SIM2LIB
XSLA	*SIM2LIB
XSRH	*SIM2LIB
XSTP	*SIM2LIB
XTEND2	*LIBRARY
XWTC	*SIM2LIB
XZRO	*SIM2LIB
X40	*SIM2LIB
YABS	*PLOTSYS
YREL	*PLOTSYS
ZHOLD	*CSMPLIB
ZLK2	OLD:LIBRARY
ZLK2	NAAS:OLDLIB
ZLOOK	OLD:LIBRARY
ZLOOK	NAAS:OLDLIB
ZLOOKC	NAAS:OLDLIB
ZLOOKC	OLD:LIBRARY
ZOR	*CSMPLIB
ZPC	NAAS:OLDLIB
ZPC	OLD:LIBRARY
ZPOLY	NAAS:OLDLIB
ZPOLY	OLD:LIBRARY
ZPOLY2	NAAS:OLDLIB
ZPOLY2	OLD:LIBRARY
ZPR	NAAS:OLDLIB
ZPR	OLD:LIBRARY
ZQUAD	NAAS:OLDLIB
ZQUAD	OLD:LIBRARY
Z256	FORTRAN I/O Library

Resident System (LCSYMBOL)

<EFL>	CHKFDUB	FREESP	JULGRGTM	QUIT	SSTOR
<FIX>	CHKFIL	FREESPAC	KEYWRD	RATENBR	STAT$
$CYLALOC	CHKFILE	FSIZE	KWSCAN	READ	STATBUFF
$EXEC1	CLOSEFIL	FSRF	LASTJOB	READ#	STDDMP
$JQENT	CLOSFL	FSTATCMD	LDINFO	READE	STDTV
$POOLCHG	CLSNET	FTN	LINK	RENAME	SYSDEFS
$ROUTAB	CMD	GDINFO	LIOUNITS	RENUMB	SYSTEM
#RMTCOPY	CMDNOE	GDINFO2	LIOUNS	RERUN	SYSTEM#
ACTIVCNT	CNFGINFO	GDINFO3	LOAD	RETLNR	TAPERTN
APL	CNTLNR	GDINF2	LOADINFO	REWIND#	TAPEUC
ASCEBC	CNTRL	GDINF3	LOCK	RSTIME	TEL2
ASMTDEFS	CONTROL	GETD	LODMAP	RSVNMTBL	TIME
ASTATSUB	COST	GETFD	MASCEBC	SCANSTOR	TIMNTRP
ATTNT	CREATE	GETFD1	MESSAGE	SCARDS	TIMTRP
ATTNTRP	CREATE#	GETFD6	MNETRTN	SCARDS#	TPRDUC
AWXCMPA2	CREPLY	GETFST	MODFTBLE	SDS	TPWRUC
AWXCMPB2	CUINFO	GETID	MOUNT	SDUMP	TRACER
BASIC0	CVTOMR	GETIME	MOUNTCMD	SERCOM	TRACER#
BINEBCD	DESTROY	GETLST	MRXA	SERCOM#	TRANTB
BINEBCD2	DESTRY	GETSPA	MSG	SETIME	TRUNC
BLKLTR	DEVLST	GETSPACE	MTS	SETIOERR	TSFO
BLOKLETR	DISMNT	GFINFO	MTS#	SETKEY	TSKFMT
BLSTDEV	DISMOUNT	GPRJNO	MTSCMD	SETLCK	UC3330
BMLOCK	DSRDISPV	GPSECT	MTSCMD#	SETLIO	UMLOAD
BSINK	DSR3270	GRAB3270	NOTE	SETLNR	UMLOADFG
BSRF	DYSSUB#	GRGJULDT	NOTE#	SETPFX	UMLOADFS
BUFALLOC	EBCASC	GRGJULTM	NUMDEV	SIOC	UMLOADNF
CALC	EBCMASC	GRJLSEC	PDP8RTN	SIOC#	UMLOADRP
CALC#	ECHO	GUINFO	PEEL	SIOCP	UNLCK
CANREPLY	EDIT	GUINFUPD	PERCMD	SPELCK	UNLK
CASECONV	EDITOR	GUSER	PERMIT	SPELLCHK	UNLOAD
CDRDUC	EMPTY	GUSER#	PGNTT	SPIRLCS	WRITBF
CDSTUC	EMPTYF	GUSERID	PGNTTRP	SPITBOL	WRITE
CFDUB	ERROR	IF	PILL	SPRINT	WRITE#
CFGINF	ERROR#	INITCNT	PL1SYM	SPRINT#	WRITEBUF
CHGFLG	FHDRDISP	INITLOCK	POINT	SPUNCH	XCTL
CHGFSZ	FNAMETRT	JLGRSEC	POINT#	SPUNCH#	
CHGMBC	FPSECT	JOBLST	PTRUC	SSFMT	
CHKACC	FREED	JTBLLIM	PUNUC	SSRTN	
CHKFDB	FREEFD	JULGRGDT	QPSECT	SSTACLS	

FORTRAN I/O Library (<FIX>)

B256	FIOEND	IBCOM#	OVERFL	SETSTA	XCTLF
DEBUG#	FRDNL#	IBCOM##	PUTIHC	SLITE	Z256
DIOCS#	FTNCMD	LDFIO#	SETBLK	SLITET	
DVCHK	FWRNL#	LINKF	SETDSN	STARTF	
FCVTHB	GETIHC	LOADF	SETDSR	UNLDF	

Elementary Function Library (<EFL>)

ALGAMA	CDABS	COTAN	DEXP#	DTAN	MAX0
ALOG	CDCOS	CSIN	DGAMMA	DTANH	MAX1
ALOG#	CDDVD#	CSQRT	DIMAG	ERF	MIN0
ALOG10	CDEXP	DARCOS	DLGAMA	ERFC	MIN1
AMAX0	CDLOG	DARSIN	DLOG	ERRMON#	SIN
AMAX1	CDMPY#	DATAN	DLOG#	EXP	SIN#
AMIN0	CDSIN	DATAN2	DLOG10	EXP#	SINH
AMIN1	CDSQRT	DATAN2#	DMAX1	FCDXI#	SQRT
ARCOS	CDVD#	DCOS	DMIN1	FCXPI#	SQRT#
ARSIN	CEXP	DCOS#	DREAL	FDXPD#	TAN
ATAN	CLOG	DCOSH	DSIN	FDXPI#	TANH
ATAN2	CMPY#	DCOTAN	DSIN#	FIXPI#	
ATAN2#	COS	DERF	DSINH	FRXPI#	
CABS	COS#	DERFC	DSQRT	FRXPR#	
CCOS	COSH	DEXP	DSQRT#	GAMMA	

*LIBRARY

$SPACE	E7090P	IHCLOGIC	LCOMPL	PRPLOT	SHFTR
#FPCON	FINDC	IMVC	LETGO	QCLOSE	SIOERR
@TESTITP	FINDST	INC	LOPEN	QCNTRL	SKIP
ACCEPT	FREAD	IOC	LOR	QFREEUCB	SORT
ADROF	GCLOSE	IOHETC	LXOR	QFRUCB	SORTEA
AND	GDINF	IOHIN	MOVEC	QGET	SORT1
ARINIT	GETIOHER	IOHOUT	OMIT	QGETUCB	SORT2
ARRAY	GJMSPSCT	IOH370	ONE@ATIM	QGTUCB	SORT3
ARRAY2	GOPEN	IOPKG	OR	QOPEN	SPIE
ATNTRP	GRAND	IOPMOD	OSGRDT	QPUT	STIMER
BTD	GRAND1	IORELEAS	OWNCONVR	QSAM	STPLT1
COMC	GRJLDT	ITR	PCCLOSE	QSAMP	STPLT2
COMPL	GRJLTM	ITRT	PCLOSE	RCALL	TICALL
DROPIOER	GROSDT	IXC	PCOPEN	RCLOSE	TRNC
DTB	GTDJMS	JLGRDT	PDUMP	READBFR	TRNST
DUMP	GTDJMSR	JLGRTM	PLOTDS	REWIND	TTIMER
D7090	IADROF	JMSGPSCT	PLOT1	ROPEN	TWAIT
D7090P	IBERH#	JMSGTD	PLOT14	SERCLOSE	URAND
EQUC	ICLC	JMSGTDR	PLOT2	SEROPEN	XOR
ERASAL	IED	JTUGTD	PLOT3	SETC	XTEND2
ERASE	IEDMK	JTUGTDR	PLOT4	SETFRVAR	
EXIT	IGC	LAND	POPEN	SETIOHER	
EXTEND	IHCFEXIT	LCLOSE	PRCHAR	SETLOG	
E7090	IHCIBERH	LCOMC	PREND	SHFTL	

*ALGOLLIB

IHIERMSG	IHIFSAIN	IHIIDEIR	IHILLO	IHIOTARR	SVXOS10
IHIERM01	IHIFSARA	IHIIORCI	IHILLOGM	IHIPTTAB	SVXOS11
IHIERM02	IHIFSARB	IHIIORCL	IHILORAR	IHISAT	SVXOS14
IHIERM03	IHIGPRCL	IHIIORCN	IHILOREA	IHISATAN	SVXOS19
IHIERM04	IHIGPRGT	IHIIORCP	IHILOREL	IHISEX	SVXOS20
IHIERM05	IHIGPRIT	IHIIORED	IHILSCC	IHISEXPT	SVXOS23
IHIERM06	IHIGPROP	IHIIOREN	IHILSCS	IHISLO	SVXOS35
IHIERM07	IHIGPROT	IHIIORER	IHILSCSN	IHISLOGM	SVXOS4
IHIERROR	IHIGPRPT	IHIIOREV	IHILSQ	IHISORAR	SVXOS5
IHIFDD	IHIGPRTN	IHIIORGP	IHILSQRT	IHISOREA	SVXOS6
IHIFDDXP	IHIIARRT	IHIIORNX	IHIOARRY	IHISOREL	SVXOS64
IHIFDI	IHIIARRY	IHIIOROP	IHIOBARR	IHISSCC	SVXOS7
IHIFDIXP	IHIIARTN	IHIIOROQ	IHIOBOAR	IHISSCS	SVXOS8
IHIFII	IHIIBARR	IHIIORTN	IHIOBOOL	IHISSCSN	SVXOS9
IHIFIIXP	IHIIBOAR	IHIISYMB	IHIOINAR	IHISSQ	SVXOS999
IHIFRI	IHIIBOOL	IHILAT	IHIOINTE	IHISSQRT	
IHIFRIXP	IHIIDEAI	IHILATAN	IHIOINTG	IHISYSCT	
IHIFRR	IHIIDECM	IHILEX	IHIOSTRG	INTERFAC	
IHIFRRXP	IHIIDEII	IHILEXPT	IHIOSYMB	LIBENTRY	

*ALGOLWLIB

ALGOLX	AWXSL003	AWXSL006	AWXSL009	AWXSL012	AWXSL015
AWXSL001	AWXSL004	AWXSL007	AWXSL010	AWXSL013	SEGN001
AWXSL002	AWXSL005	AWXSL008	AWXSL011	AWXSL014	

*ALGOLWXLIB

AWXLIBR2	AWXSL003	AWXSL006	AWXSL009	AWXSL012	AWXSL015
AWXSL001	AWXSL004	AWXSL007	AWXSL010	AWXSL013	SEGN000
AWXSL002	AWXSL005	AWXSL008	AWXSL011	AWXSL014	SEGN001

*APLLIB

**APLGOA	APLDEL2	APLEVARA	APLINDXA	APLREL0A	FILEPTR
*APLCONA	APLDEL2A	APLFND1	APLNOBL	APLREL1	IHENTRY
APLADDR	APLDEL3	APLFND1A	APLNOBLA	APLREL1A	IHESRCM
APLALOC	APLDEL3A	APLFND2	APLNSRT	APLRIN	PGNTTRP
APLCON	APLDESC	APLFND2A	APLNSRTA	APLRMV1	SYSVMDF
APLCONA	APLDREC	APLFND3	APLNUMB	APLRMV1A	SYSVMFR
APLCONAA	APLEMES	APLFND3A	APLNUMBA	APLRMV2	VIRMIN
APLCRT1	APLEMESA	APLFORM	APLOC	APLRMV2A	VIRMOUT
APLCRT1A	APLERR	APLFREE	APLOWNI	APLROUT	
APLCRT2	APLESET	APLGARB	APLOWNIA	APLSNAM	
APLCRT2A	APLESETA	APLGARBA	APLOWRS	APLSNAMA	
APLDEL1	APLEV	APLGO	APLOWRSA	APLTYPE	
APLDEL1A	APLEVAR	APLINDX	APLREL0	APLUDAT	

*COBLIB

CURSEGM	ILBOBII1	ILBOERR1	ILBOIFB0	ILBOSCH0	ILBOVCO0
IERRCO00	ILBOBII2	ILBOERR2	ILBOIFB1	ILBOSGM0	ILBOVMO0
ILBOACP0	ILBOCKP0	ILBOERR3	ILBOIFB2	ILBOSPA0	ILBOVMO1
ILBOANE0	ILBOCLS0	ILBOERR4	ILBOIFD0	ILBOSRT0	ILBOVTR0
ILBOANF0	ILBODCI0	ILBOERR5	ILBOIFD1	ILBOSTI0	ILBOWTB0
ILBOATB0	ILBODCI1	ILBOERR6	ILBOITB0	ILBOSTP0	ILBOXDI0
ILBOBID0	ILBODSP0	ILBOETB0	ILBOIVL0	ILBOSTP1	ILBOXMU0
ILBOBID1	ILBODTE0	ILBOFPW0	ILBOMVL0	ILBOTEF0	ILBOXPR0
ILBOBID2	ILBODTE1	ILBOGPW0	ILBOPTV0	ILBOTEF1	SORTE1
ILBOBIE0	ILBOEFL0	ILBOIDB0	ILBOPTV1	ILBOTEF2	SORTE4
ILBOBIE1	ILBOEFL1	ILBOIDB1	ILBOPTV2	ILBOTEF3	SORTE9
ILBOBIE2	ILBOEFL2	ILBOIDR0	ILBOSAMR	ILBOTRN0	
ILBOBII0	ILBOERR0	ILBOIDT0	ILBOSAM0	ILBOUTB0	

*CSMPLIB

ADAMS	CSMPTR	IMPL	MRIGHT	RANG2	SPLIT1
AFGEN	CSTORE	IMPLST	NAME	RECT	SSTORE
ALPHA	DATAST	IMPULS	NAND	RERUN	STATUS
AND	DEADSP	INITLZ	NLFGEN	RKS	STEP
BCDIST	DEBUG	INSW	NOCENT	RMACST	STORE
BOOLE	DELAY	INTGST	NOR	RNDGEN	STORST
BUILD	DERIV	INTRAN	NOT	RST	STRUST
BUILDR	EOR	INTRP	NTOBCD	SCAN	TRANSA
CENTRL	EQUIV	IOR	NUMER	SEQUST	TRAPZ
CKSTOR	F	LIMIT	OUTSW	SHIFT	UND
COMPAR	FCNSW	MAINEX	PLOTR	SIMOUT	ZHOLD
COMPL	GAUSS	MILNE	PULSE	SIMP	ZOR
CONTIN	GEN1ST	MLEFT	QNTZR	SINE	
CSMPEX	GEN2ST	MMACST	RAMP	SPLIT	
CSMPST	HSTRSS	MOVE	RANG1	SPLITR	

*EXPLIB

COMPACTI	IOPACK	TIME_OF_	XPLSM	XPLSM@SP

*GASP

COLCT	ERROR	FINSRT	OTPUT	RMOVEF	SUMRY
DATAN	ERRRR	GASP	PRHIST	RMOVEL	TMST
DRAND	EXITER	GASPRS	PRNTQ	RNORM	UNFRM
DRSET	FILEM	GETENT	PRODQ	RUNCON	
DRSINT	FILEMF	HISTO	PUTENT	SET	
ERLNG	FILEML	LOCAT	RDENT	SETEMP	
EROUT	FINDN	MONTR	RLOGN	SETWK	
ERRER	FINDQ	NPOSN	RMOVE	SUMQ	

October 1976

*GPSSLIB

GTFMVL	GTHMVL	PTFMVL	PTHMVL	SYSHELP	
GTFPVL	GTHPVL	PTFPVL	PTHPVL		
GTFSVL	GTHSVL	PTFSVL	PTHSVL		

*IG

#IG	IGBGNO	IGHSPO	IGPUTO	PFDUB	PLTPEN
#IGDSM	IGBGNS	IGHUE	IGSENS	PFLNAM	PLTSIZ
#IGETDD	IGCTNS	IGINIT	IGSYM	PGNHDR	PLTSTD
#IGETHSP	IGCTRL	IGINT	IGTEXT	PINT	PLTSTP
#IGINITT	IGCVTC	IGLIKE	IGTRAN	PLDNO	PLTTRM
#IGPD	IGDA	IGLOAD	IGTXT	PLFSPL	PLTXMX
#PWRIT	IGDELO	IGMA	IGTXTH	PLOTNO	PSTART
ACTVLEF#	IGDELS	IGMR	IGUSER	PLRSPL	PSYMFG
AGSENS	IGDR	IGPDSW	IGVEC	PLTBET	PWRIT
CLEANUP#	IGDRON	IGPFPF	IGVWPT	PLTBGN	PWTMAX
DGSENS	IGENDO	IGPICK	IGXYIN	PLTEND	PXABS
ERRCOM#	IGENDS	IGPIKC	PENABS	PLTNBR	PXMARG
FINDADR#	IGFMT	IGPIKN	PENMOV	PLTOUT	PYABS
IGATTB	IGFMTH	IGPIKS	PENSPD	PLTPAP	SETIGFDP

*KDFLIB

BTC	DATASK	INBASI	OUTPUT	REWIND	WRITE
CHARIN	ETW	INTERC	READ	SKIP	WRITEA
CHAROU	FIND	NEWLIN	READAR	SPACE	WRITEB
CLOSE	FORMAT	OPEN	READBI	TAB	WRITET
COPYTE	GAP	OUTBAS	READBO	TEST	

*PLOTSYS

#CCPLOT	PCTRLN	PFDNAM	PLRSPL	POFSAV	PWRIT
#PLTMOD	PCTRL2	PFDUB	PLSTYP	POLGRD	PWTMAX
#POSET	PDFSYM	PFLNAM	PLTBET	PONRST	PXABS
#PRASTR	PDSHLN	PFNMBR	PLTBGN	POPSID	PXFACT
#PSYSYMB	PDSHL2	PGNHDR	PLTEND	PPSYM	PXMARG
#PVIRT	PDSYMB	PGRID	PLTLOG	PRSTER	PXMIN
#PWRIT	PDTAB	PINFO	PLTNBR	PSBSP	PXORG
#PXBND	PDTABR	PINT	PLTOFS	PSCALE	PXREL
IPLTYP	PDTABS	PLDFSM	PLTOUT	PSCAL1	PYABS
ITRCPT	PDTABU	PLDNO	PLTPAP	PSCNIF	PYFACT
PARROW	PELIPS	PLFSPL	PLTPEN	PSIZE	PYMIN
PARRO2	PENABS	PLGAXS	PLTPOL	PSMGEN	PYORG
PAXFMT	PENCHG	PLGGRD	PLTREC	PSTART	PYREL
PAXFM2	PENDN	PLGPOL	PLTSIZ	PSYMB	SINCOS
PAXIS	PENDNS	PLGSCL	PLTSTD	PSYMFG	XABS
PAXSCL	PENMOV	PLINE	PLTSTP	PSYMLN	XREL
PAXTIC	PENOPT	PLIN2	PLTTRM	PSYMPT	YABS
PAXTTL	PENREL	PLNSYM	PLTTYP	PSYMSV	YREL
PAXVAL	PENSPD	PLOG10	PLTXMX	PSYSYM	
PBOUND	PENTUG	PLOOK	PNUMBR	PTDST2	

PCEPT	PENUP	PLOTCC	POFRES	PTHAXS	
PCIRCL	PENUPS	PLOTNO	POFRST	PTSYMB	

*PL1LIB

ERROR	IHEMAIN	IHESAPC	IHESIZ	IHETABS	SYSTEM
IHEABN	IHESAP	IHESAPD	IHESIZE	SNAP	
IHEABND	IHESAPA	IHESAPE	IHESPRT	SPIE	
IHELTT	IHESAPB	IHESAPF	IHETAB	SYSERR	

Shared PL/I Library (PL1SYM)

ATTACH	IHEDCNA	IHEIOBA	IHEMXSN	IHESMX0	IHEVKFA
BATCH	IHEDCNB	IHEIOBB	IHEMXSX	IHESNLC	IHEVKGA
CNTL	IHEDDIA	IHEIOBC	IHEMZUD	IHESNLK	IHEVPAA
CPUTIME	IHEDDIB	IHEIOBD	IHEMZUM	IHESNLS	IHEVPBA
DDEF#	IHEDDJA	IHEIOBE	IHEMZVD	IHESNLZ	IHEVPCA
ELAPSED	IHEDDOA	IHEIOBT	IHEMZVM	IHESNSC	IHEVPDA
FINFO	IHEDDOB	IHEIOCA	IHEMZW0	IHESNSK	IHEVPEA
IHEABU0	IHEDDOC	IHEIOCB	IHEMZZ0	IHESNSS	IHEVPFA
IHEABV0	IHEDDOD	IHEIOCC	IHENL1A	IHESNSZ	IHEVPGA
IHEABW0	IHEDDOE	IHEIOCT	IHENL1L	IHESNWC	IHEVPHA
IHEABZ0	IHEDDPA	IHEIODG	IHENL1N	IHESNWK	IHEVQAA
IHEADD0	IHEDDPB	IHEIODP	IHENL2A	IHESNWS	IHEVQBA
IHEADV0	IHEDDPC	IHEIODT	IHENL2L	IHESNWZ	IHEVQCA
IHEAPDA	IHEDDPD	IHEIOFA	IHENL2N	IHESNZC	IHEVSAA
IHEAPDB	IHEDIAA	IHEIOFB	IHENOTE	IHESNZK	IHEVSBA
IHEATL1	IHEDIAB	IHEIOGA	IHEOCLA	IHESNZS	IHEVSCA
IHEATL2	IHEDIBA	IHEIONA	IHEOCLB	IHESNZZ	IHEVSDA
IHEATL3	IHEDIBB	IHEIOPA	IHEOCLC	IHESQL0	IHEVSDB
IHEATL4	IHEDIDA	IHEIOPB	IHEOCLD	IHESQS0	IHEVSEA
IHEATS1	IHEDIEA	IHEIOPC	IHEOSDA	IHESQW0	IHEVSEB
IHEATS2	IHEDILA	IHEIOXA	IHEOSEA	IHESQZ0	IHEVSFA
IHEATS3	IHEDILB	IHEIOXB	IHEOSIA	IHESRCA	IHEXIB0
IHEATS4	IHEDIMA	IHEIOXC	IHEOSSA	IHESRCB	IHEXID0
IHEATTN	IHEDMAA	IHEITAA	IHEOSTA	IHESRCC	IHEXIL0
IHEATWH	IHEDNBA	IHEITAX	IHEPDF0	IHESRCD	IHEXIS0
IHEATWN	IHEDNCA	IHEITAZ	IHEPDL0	IHESRCE	IHEXIU0
IHEATZH	IHEDOAA	IHEJXIA	IHEPDS0	IHESRCF	IHEXIV0
IHEATZN	IHEDOAB	IHEJXII	IHEPDW0	IHESRDA	IHEXIW0
IHEBSA0	IHEDOBA	IHEJXIY	IHEPDX0	IHESSF0	IHEXIZ0
IHEBSC0	IHEDOBB	IHEJXSI	IHEPDZ0	IHESSGC	IHEXXL0
IHEBSD0	IHEDOBC	IHEJXSY	IHEPNT	IHESSGR	IHEXXS0
IHEBSF0	IHEDODA	IHEKCAA	IHEPRD	IHESSHC	IHEXXW0
IHEBSI0	IHEDODB	IHEKCBA	IHEPRTA	IHESSHR	IHEXXZ0
IHEBSKA	IHEDOEA	IHEKCDA	IHEPRTB	IHESSX0	IHEYGFS
IHEBSKK	IHEDOMA	IHEKCDB	IHEPSF0	IHESTGA	IHEYGFV
IHEBSKR	IHEDSPA	IHELDIA	IHEPSL0	IHESTGB	IHEYGLS
IHEBSMF	IHEDUMC	IHELDIB	IHEPSS0	IHESTPA	IHEYGLV
IHEBSMV	IHEDUMJ	IHELDIC	IHEPSW0	IHESTRA	IHEYGSS
IHEBSMZ	IHEDUMP	IHELDID	IHEPSX0	IHESTRB	IHEYGSV
IHEBSN0	IHEDUMT	IHELDOA	IHEPSZ0	IHESTRC	IHEYGWS
IHEBSO0	IHEDVU0	IHELDOB	IHEREAD	IHETHL0	IHEYGWV

IHEBSS2	IHEDVV0	IHELDOC	IHERITE	IHETHS0	IHEYGXS
IHEBSS3	IHEDZW0	IHELNLD	IHESADA	IHETNLD	IHEYGXV
IHEBSTA	IHEDZZ0	IHELNLE	IHESADB	IHETNLR	IHEYGZS
IHEBSVA	IHEEFLC	IHELNL2	IHESADD	IHETNSD	IHEYGZV
IHECFAA	IHEEFLF	IHELNSD	IHESADE	IHETNSR	IHEZZZZZ
IHECFBA	IHEEFSC	IHELNSE	IHESADF	IHETNWH	INTSUBS
IHECFCA	IHEEFSF	IHELNS2	IHESAFA	IHETNWN	LASTKEY
IHECNTA	IHEERRA	IHELNW0	IHESAFB	IHETNZH	NEXTKEY
IHECNTB	IHEERRB	IHELNZ0	IHESAFC	IHETNZN	PLCALL
IHECSC0	IHEERRC	IHELSPA	IHESAFD	IHEUPAA	PLCALLD
IHECSI0	IHEERRD	IHELSPB	IHESAFF	IHEUPAB	PLCALLE
IHECSKK	IHEERRE	IHELSPC	IHESAFQ	IHEUPBA	PLCALLF
IHECSKR	IHEEXL0	IHELSPD	IHESARA	IHEUPBB	PL1ADR
IHECSMB	IHEEXS0	IHELSPE	IHESARC	IHEVCAA	PL1RC
IHECSMF	IHEEXW0	IHELTV	IHESHLC	IHEVCSA	QUIT$
IHECSMH	IHEEXZ0	IHEMPU0	IHESHLS	IHEVCSB	RAND
IHECSML	IHEHTL0	IHEMPV0	IHESHSC	IHEVFAA	RFINFO
IHECSMV	IHEHTS0	IHEMXBN	IHESHSS	IHEVFBA	SIGNOFF
IHECSS2	IHEIOAA	IHEMXBX	IHESMF0	IHEVFCA	TFINFO
IHECSS3	IHEIOAB	IHEMXDN	IHESMGC	IHEVFDA	USERID
IHECSTA	IHEIOAC	IHEMXDX	IHESMGR	IHEVFEA	
IHECSVA	IHEIOAD	IHEMXLN	IHESMHC	IHEVKBA	
IHEDBNA	IHEIOAT	IHEMXLX	IHESMHR	IHEVKCA	

*PL360LIB

$PLCOMP	ERRBUFFR	PGNTEXIT	READ	SYSINIT	WRITE
COPY	ERRPRINT	PUNCH	SERCOMPR	SYSTERM	

*SIM2LIB

LCC$F	REFIELD$	RLS2$R	RRDR$R	RTAN$F	RXEVS
LDES$F	RERLANG$	RLT1$R	RRDS$R	RTIM1$R	RYEAR$F
LERR$F	RERR$R	RLT2$R	RRDT$R	RTIM2$R	RZEVS
LIN$F	REXP$F	RMINUTE$	RREL$R	RTIM3$R	XCLR
LOUT$F	REXPONEN	RMODE$F	RRES$R	RTRACE$R	XFIX
LRDELIM$	RFATAL	RMONTH$F	RRFA$R	RTRUNC$F	XFLT
LSDELIM$	RFRAC$F	RNDAY$F	RRFD$R	RUL$F	XFRE
RAEVS	RFREE$R	RNORMAL$	RRFI$R	RUNIFORM	XIDE
RARCCOS$	RGAMMA$F	ROBEY$R	RRIRV$R	RUSE$R	XINR
RARCSIN$	RGAMMAJ$	RORIGIN$	RRLR$R	RWEEKDAY	XLAR
RARCTAN$	RGUIB$R	ROUT$F	RRMD$F	RWEIBULL	XMAS
RBETA$F	RHOUR$F	RPOISSON	RRRRV$R	RWTA$R	XREC
RBINOMIA	RIN$F	RRANDI$F	RRSTEP$F	RWTB$R	XRET
RBLOCK$R	RISTEP$F	RRANDOM$	RRWD$R	RWTC$R	XSE1
RCLS$R	RITOA$F	RRDA$R	RRXP$F	RWTD$R	XSE2
RCOS$F	RIXP$F	RRDB$R	RSEED$F	RWTE$R	XSLA
RCRE$F	RLIN$F	RRDC$R	RSFIELD$	RWTI$R	XSRH
RCUIB$R	RLOGEF	RRDD$R	RSIGN$F	RWTP$R	XSTP
RDATE$F	RLOG$NOR	RRDE$R	RSIN$F	RWTR$R	XWTC
RDAY$F	RLOG10	RRDI$R	RSKIP$R	RWTS$R	XZRO
RDIM$F	RLS1$R	RRDL$R	RSQRT$F	RWTT$R	X40

*SLIP

ADVLEL	CONT	GETBLK	LRDROV	NULSTL	SEQRDR
ADVLER	C1	HLF1	LS	NULSTR	SEQSL
ADVLL	C2	HLF2	LSSCPY	NXTLFT	SEQSR
ADVLNL	C3	H1	LSTDMP	NXTRGT	SETBLK
ADVLNR	C4	H2	LSTEQL	PARMTN	SETDIR
ADVLR	C5	ID	LSTMRK	PARMT2	SETIND
ADVLWL	C6	IDATUM	LSTPRO	POPBOT	SETMRK
ADVLWR	C7	INHALT	LVLRVT	POPTOP	SETRAC
ADVSEL	C8	INITAS	LVLRV1	PRESRV	SHIN
ADVSER	DATUM	INITRD	MADATR	PRLSTS	SLIPPRIM
ADVSL	DELETE	INLSTL	MADLFT	PUBDMP	SLPDMP
ADVSNL	DERROR	INLSTR	MADNBT	QQSV	SQIN
ADVSNR	DLS	INTGER	MADNTP	QTR1	SQOUT
ADVSR	DMP	IRALST	MADOV	QTR2	STRDAT
ADVSWL	DMPCLR	IRARDR	MADRGT	QTR3	STRDIR
ADVSWR	DMPERR	ITSVAL	MAKEDL	QTR4	STRIND
BOT	DMPER1	LANORM	MRK	Q1	SUBSBT
CADLFT	DMPFRE	LCNTR	MRKGET	Q2	SUBST
CADNBT	DMPLAV	LDATVL	MRKLSS	Q3	SUBSTP
CADNTP	DMPLNK	LIST	MRKLST	Q4	TERM
CADRGT	DMPLST	LISTAV	MTDLST	RCELL	TOP
CHR1	DMPMRK	LISTMT	MTLIST	RDLSTA	TSYM
CHR2	DMPRDR	LNKL	NAMEDL	REALL	TSYM#
CHR3	DMPRES	LNKR	NAMTST	REALS	VISIT
CHR4	DMPUBL	LOCT	NEWBOT	REED	XMASK
CHR5	DRS	LOFRDR	NEWTOP	RESTOR	
CHR6	EBX	LPNTR	NEWVAL	RS	
CHR7	EQUAL	LPURGE	NOATVL	SEQLL	
CHR8	F4TRBK	LRDRCP	NUCELL	SEQLR	

*SPITLIB

OSINT

NAAS:EISPACK

BAKVEC	COMBAK	FIGI2	ORTBAK	RG	TQLRAT
BALANC	COMHES	HQR	ORTHES	RGG	TQL1
BALBAK	COMLR	HQR2	ORTRAN	RS	TQL2
BANDR	COMLR2	HTRIBK	QZHES	RSB	TRBAK1
BANDV	COMQR	HTRIB3	QZIT	RSG	TRBAK3
BISECT	COMQR2	HTRIDI	QZVAL	RSGAB	TRED1
BQR	CORTB	HTRID3	QZVEC	RSGBA	TRED2
CBABK2	CORTH	IMTQLV	RATQR	RSP	TRED3
CBAL	ELMBAK	IMTQL1	REBAK	RST	TRIDIB
CG	ELMHES	IMTQL2	REBAKB	RT	TSTURM
CH	ELTRAN	INVIT	REDUC	SVD	
CINVIT	FIGI	MINFIT	REDUC2	TINVIT	

October 1976

NAAS:FUNPACK

BESEK0	DDAW	DELIKM	DEXPEI	MONERR	NATSK0
BESEK1	DEI	DELIK1	DPEONE	NATSEE	NATSK1
BESK0	DELIEM	DELIPE	ERRTRA	NATSEI	
BESK1	DELIE1	DELIPK	FCNMON	NATSEK	

NAAS:LIT

DECOMP	MAPRNT	PRFIT	PRQUAD	SOLVE
DVDQ	NLBACK	PRLIN	PRSORT	SPLINE
DVDQG	NLSYS	PRMIN	RANDUM	VCPRNT
DVDQ1	PRAXIS	PRPRIN	SETUP	

NAAS:NAL

CAXMB	CDIR	DAXMB	DFFT2A	SAXMB	SFFT2A
CBS	CDLUD	DBS	DFS	SBS	SFS
CDAXMB	CILU	DFFT	DILU	SFFT	SILU
CDBS	CIR	DFFTA	DIR	SFFTA	SIR
CDILU	CLUD	DFFT2	DLUD	SFFT2	SLUD

NAAS:OLDLIB

BAIR	INV#	QDIV	SLE1	TSEP1	ZPOLY2
DAINT	INV1	SAINT	SLE2	URAND	ZPR
DBS	JESS	SDBS	SLE3	UTSLE	ZQUAD
DBST	LINC	SGLINT	SLE4	ZLK2	
GLINT	LINCR	SINV	SLRD	ZLOOK	
GRAND	LINPG	SINV1	SSLE	ZLOOKC	
GRAND1	LRD	SINV2	TSEP	ZPC	
INV	LTSLE	SLE#	TSEPC	ZPOLY	

NAAS:SSP

ABSNT	DCNP	DQG8	HEP	POLRT	RECP
ACFI	DCNPS	DQHFE	HEPS	PPRCN	RHARM
AHI	DCPY	DQHFG	HPCG	PQFB	RINT
ALI	DCSP	DQHSE	HPCL	PQSD	RKGS
APCH	DCSPS	DQHSG	HSBG	PRBM	RK1
APFS	DDBAR	DQH16	INUE	PROBT	RK2
APLL	DDCAR	DQH24	IO	PRQD	RSIMC
APMM	DDET3	DQH32	JELF	PSUB	RSRT
ARAT	DDET5	DQH48	KOLMO	PVAL	RSUM
ARRAY	DDGT3	DQH64	KOLM2	PVSUB	RTAB
ATEIG	DELI1	DQH8	KRANK	QATR	RTIE
ATSE	DELI2	DQL12	LAP	QA10	RTMI
ATSG	DET3	DQL16	LAPS	QA2	RTNI
ATSM	DET5	DQL24	LBVP	QA3	RTWI
AUTO	DFMCG	DQL32	LEP	QA4	SADD
AVCAL	DFMFP	DQL4	LEPS	QA5	SCLA
AVDAT	DFRAT	DQL8	LLSQ	QA6	SCMA
BDTR	DGELB	DQSF	LOAD	QA7	SDIV

BESJ
BESK
BESY
BISER
BOUND
CADD
CANOR
CCPY
CCUT
CDTR
CEL1
CEL2
CHISQ
CINT
CNP
CNPS
CONVT
CORRE
CROSS
CS
CSP
CSPS
CSRT
CSUM
CTAB
CTIE
DACFI
DAHI
DALI
DAPCH
DAPFS
DAPLL
DAPMM
DARAT
DATSE
DATSG
DATSM
DBAR
DCAR
DCEL1
DCEL2
DCLA
DGELG
DGELS
DGT3
DHARM
DHEP
DHEPS
DHPCG
DHPCL
DISCR
DJELF
DLAP
DLAPS
DLBVP
DLEP
DLEPS
DLGAM
DLLSQ
DMATX
DMCHB
DMFGR
DMFSD
DMFSS
DMLSS
DMPRC
DMTDS
DPECN
DPECS
DPQFB
DPRBM
DPRQD
DQATR
DQA12
DQA16
DQA24
DQA32
DQA4
DQA8
DQG12
DQG16
DQG24
DQG32
DQG4
DQTFE
DQTFG
DRHARM
DRKGS
DRTMI
DRTNI
DRTWI
DSE13
DSE15
DSE35
DSG13
DSINV
DTCNP
DTCSP
DTEAS
DTEUL
DTHEP
DTLAP
DTLEP
EIGEN
ELI1
ELI2
EXPI
EXSMO
FACTR
FMCG
FMFP
FORIF
FORIT
FRAT
GAUSS
GDATA
GELB
GELG
GELS
GMADD
GMMMA
GMPRD
GMSUB
GMTRA
GTPRD
HARM
LOC
MADD
MATA
MCHB
MCPY
MEANQ
MFGR
MFSD
MFSS
MFUN
MINV
MISR
MLSS
MOMEN
MPAIR
MPRC
MPRD
MSTR
MSUB
MTDS
MTRA
MULTR
NDTR
NDTRI
NROOT
ORDER
PADD
PADDM
PCLA
PCLD
PDER
PDIV
PECN
PECS
PERM
PGCD
PHI
PILD
PINT
PMPY
PNORM
POINT
QA8
QA9
QG10
QG2
QG3
QG4
QG5
QG6
QG7
QG8
QG9
QHFE
QHFG
QHSE
QHSG
QH10
QH2
QH3
QH4
QH5
QH6
QH7
QH8
QH9
QL10
QL2
QL3
QL4
QL5
QL6
QL7
QL8
QL9
QSF
QTEST
QTFE
QTFG
RADD
RANDU
RANK
RCPY
RCUT
SE13
SE15
SE35
SG13
SICI
SIGNT
SIMQ
SINV
SMIRN
SMO
SMPY
SRANK
SRATE
SRMA
SSUB
STPRG
SUBMX
SUBST
TAB1
TAB2
TALLY
TCNP
TCSP
TEAS
TETRA
TEUL
THEP
TIE
TIAP
TLEP
TPRD
TRACE
TTEST
TWOAV
UTEST
VARMX
WTEST
XCPY

OLD:LIBRARY

ADCON#	FCVCO	IHCFDUMP	PDUMP	SINV2	ZLK2
BAIR	FCVEO	IHCNAMEL	PUTIHC	SLE#	ZLOOK
DAINT	FCVIO	INV	QDIV	SLE1	ZLOOKC
DBS	FCVLO	INV#	SAINT	SLE2	ZPC
DBST	FCVTHB	INV1	SDBS	SLE3	ZPOLY
DCVC	FCVZO	JESS	SETDSN	SLE4	ZPOLY2
DCVD	FIOCS#	LINC	SETDSR	SLRD	ZPR
DCVG	FRDNL#	LINCR	SETGRE	SSLE	ZQUAD
DEBUG#	FWRNL#	LINPG	SETSTA	TSEP	
DIOCS#	GETIHC	LINPG#	SGLINT	TSEPC	
DUMP	GLINT	LRD	SINV	TSEP1	
FCVAO	IBCOM#	LTSLE	SINV1	UTSLE	

UNSP:DIGLIB

CONEND	CONSET	CTQQ	DISTOK	DSINIT
CONLBL	CONTUR	CTQQIN	DSFINI	

UNSP:LIBRARY

ARMTS	ATRSTR	RATRAP	RTRAP	SPELER	TRRSTR
ATRAP	PAR	RMTS	SNOOP	TRAP	

UNSP:LSLIPLIB

ADVLEL	DERROR	INTGER	LSTEQL	NUCELL	SEQLL
ADVLER	DMP	INTSVM	LSTMRK	NULSTL	SEQLR
ADVLL	DMPCHR	IRALST	LSTPRO	NULSTR	SEQRDR
ADVLNL	DMPERR	IRARDR	LTHERE	NXTLFT	SEQSL
ADVLNR	DMPER1	ITSVAL	LVLRVT	NXTRGT	SEQSR
ADVLR	DMPFRE	IVISIT	LVLRV1	PARMT	SETDIR
ADVLWL	DMPINI	LAPRIM	MADATR	PARMT2	SETIND
ADVLWR	DMPLAV	LCNTR	MADLFT	POPBOT	SETMKW
ADVSEL	DMPLIS	LDATVL	MADNBT	POPRDR	SETMRK
ADVSER	DMPLNK	LINKWD	MADNTP	POPTOP	SLPDMP
ADVSL	DMPLST	LIST	MADRGT	PRESRV	STINKW
ADVSNL	DMPMRK	LISTAV	MAKEDL	PRLSTS	STIRDAT
ADVSNR	DMPRDR	LISTMT	MRK	PSHRDR	STRIND
ADVSR	DMPRES	LNKL	MRKLSS	PUTRDL	SUBSBT
ADVSWL	DMPSVM	LNKLW	MRKLST	QTHERE	SUBST
ADVSWR	DMPUBL	LNKR	MRKW	RCELL	SUBSTP
BOT	F4TRBK	LNKRW	MTDLST	RDLSTA	TERM
CADLFT	ID	LOCT	MTLIST	REALS	TOP
CADNBT	IDATUM	LOFRDR	NAMEDL	RECURS	VISIT
CADNTP	IDW	LPNTR	NAMTST	REED	VISTIN
CADRGT	INITAS	LPURGE	NEWBOT	RESTOR	
CRESVM	INITRD	LRDRCP	NEWTOP	RSTLST	
DATUM	INLSTL	LRDROV	NEWVAL	SAVLST	
DELETE	INLSTR	LSSCPY	NOATVL	SAVSEQ	

UNSP:PL1LIB

MAXLEN	PREAD	PREAD1	PWRITE	PWRITE1	SETLEN

UNSP:SPITLIB

SCMD	SGDINFO2	SGUINFO3	SNS	SSETPFX	SWRITE
SCREATE	SGETFD	SGUINFO4	SNS2	SSNOOP	
SCREPLY	SGUINFO	SMTSCMD	SPITATTN	SSNS	
SGDINFO	SGUINFO2	SNOOP	SREAD	SSNS2	

Reader's Comment Form

System Subroutine Descriptions
Volume 3
October 1976

Errors noted in publication:

Suggestions for improvement:

Date ----------

Name----------

Address----------

Your comments will be much appreciated. Please fold the completed form as shown on the reverse side, seal or staple, and drop in Campus Mail or in the Suggestion Box at the Computing Center or NUBS.

fold here

Publications
Computing Center
University of Michigan
Ann Arbor, Michigan 48109
USA

fold here

Update Request Form

System Subroutine Descriptions
Volume 3
October 1976

Updates to this manual will be issued periodically as errors are noted or as changes are made to MTS. If you desire to have these updates mailed to you, please fold the completed form as shown on the reverse side, seal or staple, and drop in Campus Mail or in the Suggestion Box at the Computing Center or NUBS. Campus Mail addresses must be given for local users. Updates (issued as Limited-Distribution CCMemos) are also available in the memo files at both the Computing Center and NUBS. Updates issued prior to the receipt of this form by the Computing Center will not be automatically provided; these must be obtained from the memo files.

Name_____

Address_____

_____

_____

fold here

Update Subscription Service
Publications
Computing Center
University of Michigan
Ann Arbor, Michigan 48109
USA

---

fold here