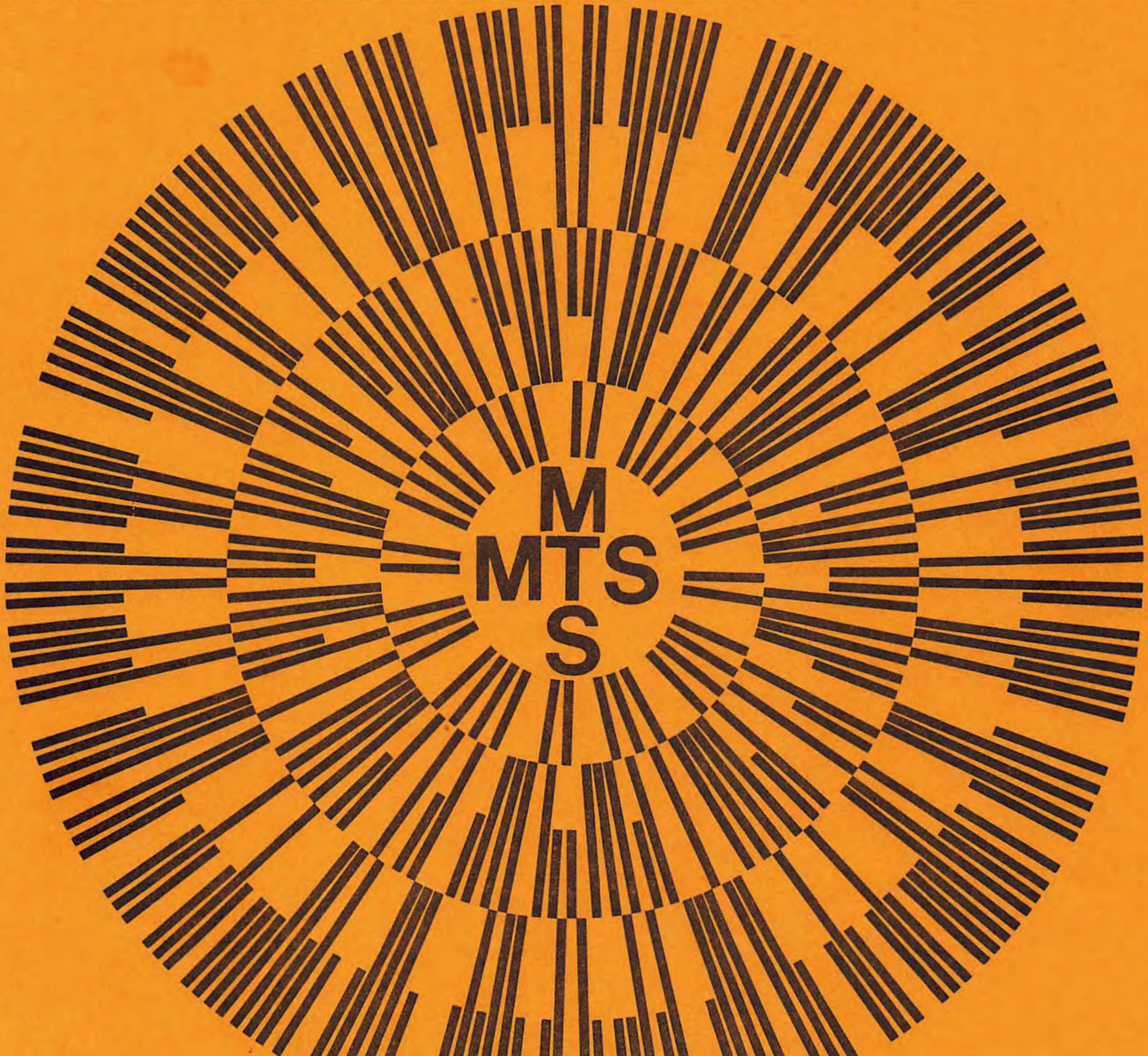
THE UNIVERSITY OF ALBERTA
COMPUTING CENTER
PUBLICATION



PL/1

ACKNOWLEDGEMENTS

This manual was largely compiled from material prepared by the staff of the University of Michigan Computing Center. Their documentation was invaluable and we are indebted to them for allowing us to use it. In particular, the following were most useful:

MTS Users' Manual, Second Edition, Volumes I and II

MTS Users' Manual, Third Edition, Volume 2

Introduction to MTS and the Computing Center (Flanigan)

Computing Center News Items

Computing Center Memos

The Computing Center wishes to personally acknowledge the assistance of Mike Alexander and Don Boettner who helped us to establish MTS at the University of Alberta.

Acknowledgement should also be made to the Computing Centre, University of British Columbia, for information obtained from some of their documentation and to I.B.M., whose manuals provided certain sections for our manuals.

DISCLAIMER

This MTS manual is a combination of earlier manuals, update notices, memos and limited experience with the system itself. Because of this, certain discrepancies are bound to occur and the Computing Center would appreciate being notified of all differences between what this manual says and what the system actually does.

This publication is intended to represent the current state-of-the-system. However, it should not be construed as an obligation to maintain the system as so stated. The MTS system, like most good systems, is continually being improved. As a result, additions, extentions, changes and deletions will occur. Notice of such changes will be made and provision for a manual updating service has been planned.

Errors, comments and suggestions should be sent to:

Information Coordinator
Computing Center
University of Alberta

PL/I
MAY 1970

PL/I

TABLE OF CONTENTS

SUMMARY

*PL1

```
SCARDS = source input
SPRINT = listing & diagnostics
SPUNCH = macro and/or object decks
0      = object module
```

Parameters

PAR=[I/O keywords],[options],....

I/O keyword file modifier parameters:

```
@U - undefined length          @B - blocked
@V - variable length           @A - attach CC bit
@F - fixed length              @M - attach MCC bit
```

Options:

[ATR NOATR]	[,CHAR60 CHAR48]	[,COMP NOCOMP]	[,DECK NODECK]
[EBCDIC TTY]	[, EXTREF NOEXTREF]	[,~~FLAGW~~ ~~FLAGE~~ FLAGS]	[,LIST NOLIST]
[,LOAD NOLOAD]	[, MACDCK NOMACDCK]	[,MACRO NOMACRO]	[,NEST NONEST]
[,OPLIST NOOPLIST]	[,SOURCE NOSOURCE]	[,SOURCE2 NOSOURCE2]	[,STMT NOSTMT]
[, XREF NOXREF]	[,LINECNT= n]	[,OPT = nn]	[,SIZE = yyyP]
	[,SORMGIN=(m,n,c) FREE]		

Options Abbreviated:

[A NA]	[,C60 C48]	[,C NC]	[,D ND]	[,EB T]	[,E NE]	[,FW ~~FE~~ FS]
[,L NL]	[,LD NLD]	[,MD NMD]	[,M NM]	[,NT NMT]	[,OL NOL]	
[0=n]	[,LC=n]	[,SIZE=yyyP]	[,SM=(m,n,c) F]			

Default Values:

```
SCARDS = *MSOURCE*@U80
SPRINT = *MSINK*@VA(129,125)
SPUNCH = *PUNCH*@F80
0      = GUSER
LC     = 60
OPT    = 01
SIZE   = 10P
SM     = (1,72)
```

LANGUAGE DESCRIPTION

References

The following IBM manuals are recommended as reference texts.

1. A PL/1 Primer (C28-6808)
2. A Guide to PL/1 for Commercial Programmers (C20-1651)
3. A Guide to PL/1 for FORTRAN Users (C20-1637)
4. PL/1 Reference Manual (C28-8201)
5. PL/1 (F) Programmer's Guide (C28-6504)

The MTS Compiler is derived from IBM's OS/360 fourth version F-level PL/1 Compiler with modifications for MTS features.

MTS Conventions and Restrictions

1. INCLUDE convention

   The following is the sole support: INCLUDE identifier ,...,identifier ; where each identifier refers to an MTS file. The spelling rules for a PL/1 identifier must apply. Hence, only permanent files can be used for INCLUDE.

2. Unspecified file names

   Where filenames are not specified, the PL/1 compiler will default to SPRINT/SCARDS instead of SYSPRINT/SYSIN.

3. TIME function

   Returns the current time of day as a character string of length eight in the form hh:mm:ss where hh is hours, mm is minutes and ss is seconds.

4. DATE function

   Returns the current data as a character string of length eight in the form mm-dd-yy where mm is month, dd is day, yy is year.

5. DELAY statement

   If executed, will generally raise an error comment that delaying is not permitted in MTS.

6. WAIT statement

   Is not supported in MTS.

7. Multi-tasking

   Is unsupported in MTS since a single MTS task cannot have subtasks.

8. Environment options in MTS.

CONSECUTIVE — dataset consists only of sequential records. This organization is assumed if none is specified.

INDEXED — dataset is an MTS line file whose records can be processed by the indexed file subroutines.

RECORD FORMAT — F(blocksize ,recordsize) V(max-blocksize ,max-recordsize) U(max-block size)

POSITIONING — LEAVE and REWIND options are used to position a magnetic tape volume when the corresponding PL/1 file is closed or when a volume-switch occurs.

9. COBOL Option

Specifies that files with this attribute will contain instructions mapped according to the COBOL algorithm. This type of file may be used only for READ INTO and WRITE FROM statements. The following figures illustrate the equivalent PL/1 data types for COBOL data types.

COBOL	PL/1
DISPLAY	PICTURE with A and/or X picture character
COMPUTATIONAL	
decimal length (=no. of 9s in picture) is	
1 to 4	no equivalent
5 to 9	FIXED BINARY (integers only)
10 to 18	no equivalent

COMPUTATIONAL-1	FLOAT(n)BINARY (for n ≤ 21)
	FLOAT(n)DECIMAL (for n ≤ 6)

COMPUTATIONAL-2	FLOAT(n)BINARY (for n < 21)
	FLOAT(n)DECIMAL (for n < 6)

COMPUTATIONAL-3	FIXED DECIMAL (precision and scale as in COBOL picture).

Printer/punch Control Characters

Two options are available for RECORD CONSECUTIVE OUTPUT files only: CC and MCC. They have the following meaning:

CC - carriage control
MCC - machine carriage control is used. These options allow spacing, skipping in record I/O files. It is the user's responsibility to insure that the first byte of each record contains a valid control character. These options are ignored in stream I/O, and the general default (except for print files) is NOCC.

Data Interchange

I. The ALIGNED and UNALIGNED attributes allow PL/1 programs to use FORTRAN unformatted records. The FORTRAN unformatted set consists of records which are a concatenation of internal data items without regard to alignment stringency.

For example:

```
INTEGER*4  A
LOGICAL*1  B
REAL*8 C
WRITE(5)A,B,C
```

This record can be declared as PL/1 structure of 3 data items:

```
DECLARE 1 R UNALIGNED,
        2 S FIXED BINARY,
        2 T CHAR(1)
        2 U FLOAT(16);
```

There are two exceptions:

1. PL/1 (F version 4) compiler does not support halfword binary data.
2. FORTRAN IOCS can split its logical records into several physical records, a technique called spanning. Only unspanned FORTRAN records can be read by PL/1 using UNALIGNED structures.

II. For COBOL data interchange, see COBOL options on the previous page.

Combination of PL/1 with other languages

Possible combinations of modules written in other languages with those in PL/1 is considerably limited since most other languages are not structured to support advanced features of the PL/1 language. Assembler users must have an intimate knowledge on the object code structure and requirements of PL/1 programs. Appendix C of PL/1 (F) Programmer's Guide (form C28-6594-3) describes the PL/1 structure in detail.

PROCESSOR DESCRIPTION

Name: *PL1

Contents: The initial object module of the MTS PL/1 Compiler.

Purpose: Compilation of MTS PL/1 source programs.

Usage: The F-level PL/1 compiler is invoked by an appropriate $RUN command specifying *PL1 as the file where the object module is to be found.

Logical I/O Units Referenced:

SCARDS - source input followed by an implied end-of-file or $ENDFILE command.
SPRINT - listing output, including diagnostics.
SPUNCH - macro deck and object deck output.
0 - object module output.

Examples:

```
$RUN  *PL1  0=QUADRAT
     [(where SCARDS and SPRINT default to
       *SOURCE* and *SINK*, respectively)]

$RUN  *PL1  SCARDS=PRIME SPUNCH=-LOAD PAR=DECK,NLD,NT

$RUN  *PL1  SPUNCH=*PUNCH* PAR=M,MD,NOCOMP,SOURCE2
```

Description: The MTS Compiler is derived from IBM's OS/360 fourth version F-level PL/1 Compiler with modifications for MTS features.

Compiler Options

Compiler options may be passed by the parameter list or in a control record. These options can be written in any order and must be separated by commas. If conflicting options are specified, the last specification in the option list will be used.

Option	Abbreviated name	Default
ATR/NOATR	A/NA	NOATR
CHAR60/CHAR48	C60/C48	CHAR60
COMP/NOCOMP	C/NC	COMP
DECK/NODECK	D/ND	NODECK
EBCDIC/TTY	EB/T	EBCDIC
EXTREF/NOEXTREF	E/NE	NOEXTREF
FLAGW/FLAGE/FLAGS	FW/FE/FS	FLAGW
LINECNT	LC	LC=60
LIST/NOLIST	L/NL	NOLIST
LOAD/NOLOAD	LD/NLD	LOAD
MACDCK/NOMACDCK	MD/NMD	NOMACDCK
MACRO/NOMACRO	M/NM	NOMACRO
NEST/NONEST	NT/NNT	NEST
OPLIST/NOOPLIST	OL/NOL	OPLIST
OPT=nn	O	OPT=01
SIZE=yyy^{P}	SIZE	SIZE=10P
SORMGIN=(mmm,nnn ,ccc)/FREE	SM/F	SM=(1,72)
SOURCE/NOSOURCE	S/NS	SOURCE
SOURCE2/NOSOURCE2	S2/NS2	SOURCE2
STMT/NOSTMT	ST/NST	STMT
XREF/NOXREF	X/NX	NOXREF

Description of Options

ATR
attributes of each identifier are to be listed. In addition, an aggregate length table is produced for arrays and structures.

CHAR60/CHAR48
allows the source language to be written in one of two character sets (60 or 48 characters).

DECK
specifies that the object module is to be written on SPUNCH.

EBCDIC/TTY
these options allow the programmer to state in which character code the source program is typed. The BCD option is not supported.

The following table gives the correspondence between teletype graphics and PL/1 symbols.

TTY Graphics	Key Combination	PL/I Symbol
← or _	shift-O	_
	shift-L	
↑ or	shift-N	|
others		same

EXTREF
causes a listing of the external symbol dictionary (ESD) of the object module to be produced.

FLAGW/FLAGE/FLAGS
specifies the severity of diagnostic messages to be listed. The following types of messages are indicated:

type of message	specification
warning	FLAGW
error	FLAGE
severe error	FLAGS
terminal error	none

FREE/SORMGIN=
(mmm,nnn ,ccc)
where 1 ≤ mmm ≤ nnn ≤ 100. The range (mmm,nnn) represents the margins for scanning source statements. ccc, if specified, must be outside the range (mmm,nnn) and indicates the position

in the input line to be used as a carriage control character in the SPRINT buffer. Valid control characters are 1, -, 0, +, and blank. The FREE option is equivalent to SM=(1,100).

LINECNT=XXX specifies the number of lines to be written per page.

LIST specifies that the generated machine instructions, constants, etc. are to be printed.

LOAD specifies that the object module is to be written on logical unit 0.

MACDCK causes the output records of the compile-time processor to be written on SPUNCH. If this deck is to be reprocessed, the source margins should be specified SM=(2,72).

MACRO specifies that compile-time processing is required.

NEST If specified, causes the PL/I compiler to indicate in the source listing the PROCEDURE and BEGIN block level count as well as DO level count.

OPLIST controls the listing of the options currently accepted by the compiler.

OPT=nn two levels of optimization are now available:

OPT=00 the object time storage requirements are minimized.

OPT=01 the execution speed is slightly improved at the expense of object-time storage space.

SIZE=nnP the compiler attempts to obtain nn pages of virtual memory. The following table shows the text and dictionary blocks to be used.

core size (in pages)	block size (in pages)
≥8	1
≥50	2
≥100	4

SOURCE specifies that the source program is to be listed on SPRINT.

SOURCE2 specifies that the input to the compile-time processor is to be listed.

STMT specifies that extra code is to be produced to allow diagnostic messages printed during the execution time to contain statement numbers.

XREF specifies that cross references of each identifier are to be listed.

The logical I/O units used by the PL/1-F compiler are as follows:

Unit	Default record format	Principal Functions
SCARDS	U(100)	source input
SPRINT	U(129, 121)	listing output
SPUNCH	F(80)	object deck and macro deck
0	F(80)	object module output
-SYSUT4		auxiliary scratch file for compile-time processor

Batch compilation

The MTS PL/1 compiler can process more than one external procedure. This is achieved by preceding the second (and each subsequent) compilation by a control record of the form:

```
    % PROCESS('option list');
```

The first position must have a percent sign (%). There may be blanks between the percent sign and the keyword PROCESS. However, there should be no blanks embedded between the P of PROCESS and the semicolon. There is no carry-over of the options used previously. If the user does not wish to specify any options, he should place the semicolon right after the keyword PROCESS. The following shows how batch compilation is achieved:

```
$RUN  *PL1 0=A PAR=LOAD,NODECK
    [source program one ]
%PROCESS('LOAD,NOEXTREF,ATR');
    [source program two ]
%PROCESS;
    [source program three]
$ENDFILE
$RUN  A+*PL1LIB
```

EXECUTION TIME CONSIDERATIONS

Running a PL/1 program

It is an absolute necessity that the main program be written in PL/1. Otherwise, one may be greeted with the message "IHE0061 NO MAIN PROCEDURE". Running nothing with *PL1LIB will also produce the same message.

A simple program may be run by specifying:

```
$RUN MYPROGRAM+*PL1LIB
```

Since stream I/O routines in PL/1 process character by character (rather than record by record), extra information describing record format and length may be needed. The following record formats are supported:

$U\left[\begin{smallmatrix}A\\M\end{smallmatrix}\right]$ (max-blocksize)

$V[B]\left[\begin{smallmatrix}A\\M\end{smallmatrix}\right]$ (max-blocksize, max-recordsize)

$F[B]\left[\begin{smallmatrix}A\\M\end{smallmatrix}\right]$ (blocksize,recordsize)

where

U stands for undefined format records. The maximum blocksize is the largest length that a record can have.

V stands for variable length records. The following figure illustrates V-format.

C_1 C_2 data (unblocked V-format)

C_1 C_2 data C_2 data ... C_2 data

(blocked V-format)

where

data consists of data bytes, not less than 10 in number

C_2 record control field (4 bytes long) describes the length of the record <u>including</u> the record control field.

C_1 block control field (4 bytes) specifies the overall length of the block including the block control field.

The blocksize must be greater than or equal to record length plus four.

F. stands for fixed records. These records can be blocked together. The blocksize must be an integral multiple of the record size.

B says that records may be blocked. U-format records <u>cannot</u> be blocked.

A sets the CC bit on. It is used primarily for printer and punch control characters which are placed in the first position of data bytes.

M sets the MCC bit on.

If A or M is not explicitly or implicitly specified a NOCC bit will be set. Standard PL/1 files have the following defaults:

PL/1 filename[1]	Record Format
Print files such as SPRINT	VA(129,125)
All others	U(80)

Record formats, can be specified by inserting these in PAR=.

Examples:

```
1. $RUN COPY+*PL1LIB       PAR=SCARDS=*SOURCE*@F

2. $RUN FMAINT+*PL1LIB     PAR=INTER=*TAPE*@F(2400,600)      -
                           MASTER=*OLDMAS*@U(132)            -
                           EXCEPTIONFILE=-E@U(300)           -
                           SCARDS=*SOURCE*@F(80)

3. conversational           $RUN    BLDLIN
                            #EXECUTION   BEGINS
                            SYSPRINT - SPECIFY FDNAME OR SEND
                             END-OF-FILE
                            *SINK*@VA(141,137)
                            INTER - SPECIFY FDNAME OR SEND
                             END-OF-FILE
                            *INTER*@FB(2400,600)
                            LAYFILE - SPECIFY FDNAME OR SEND
                             END-OF-FILE
                            -LAYFILE@F(176)
                           #EXECUTION TERMINATED
```

[1] Filenames are automatically truncated to 8-characters. Also environment options can be extracted to default record format.

4. Passing parameter strings

```
$RUN  PAR+*PL1LIB  PAR=SPRINT=     -
  *SINK*;ABC
PAR:  PROC(STR) OPTIONS(MAIN);
  DCL(STR,PARSTR)  CHAR(255)
VAR;
I=INDEX(STR,' ;');
IF I=0|I=LENGTH(STR)
 THEN PARSTR='';
 ELSE
 PAR=SUBSTR(STR,I+1);
PUT DATA  (PARSTR);
END PAR;
```

How to get a dump

In most cases, a single diagnostic message suffices and a dump is rather useless and expensive. Hence PL/1 library routines do not abend (except in catastrophic cases such as "IHE0041-INTERRUPT IN ERROR HANDLER") but return MTS with the code set in the register 15. If a PL/1 user prefers to have a dump, he may insert a simple statement (or the like) in his program:

```
ON ERROR CALL IHEDUMP;
```

This routine IHEDUMP prints useful information. The output is on the file PL1DUMP with the default "PL1DUMP=*SINK*". The information consists of:

1) SPRINT buffer and intermediate buffers, if any
2) Files currently open
3) Current file in use
4) Save areas
5) On-units, interrupts and other details

The routine IHEDUMP then calls on ERROR to dump everything if any $ERRORDUMP was previously encountered.

Name: *PL1LIB

Contents: The subroutine library for PL/1-compiled programs.

Usage: *PL1LIB is concatenated to the file containing the object modules resulting from a PL/1 compilation in a $RUN command.

Example:

```
$RUN   OBJECT+*PL1LIB
```

Description: *PL1LIB contains the object modules of subroutines which may be invoked by PL/1 compiled code. It is formatted as a library and hence only those object modules which are referenced are loaded.

PL/I SUBROUTINES

The following subroutines are intended to be called directly from PL/I programs.

Name: BATCH

Purpose: To determine whether in batch or conversational mode.

Declaration:

```
DECLARE BATCH ENTRY
        RETURNS(BIT(1));
```

Description: Returns '1'B if in batch mode; otherwise '0'B

Example:

```
IF BATCH THEN STOP;
ELSE GOTO RETRY;
```

Name: CPUTIME

Purpose: To obtain in seconds the CPU time since the start of the current program.

Declaration:

```
DECLARE CPUTIME ENTRY
RETURNS (FLOAT BINARY);
```

Description: Returns the floating-point value in seconds since the start of the program.

Example:

```
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;
```

Name: ELAPSED

Purpose: To obtain in seconds the elapsed time since the start of the program.

Declaration:

```
DECLARE ELAPSED ENTRY
RETURNS (FLOAT BINARY);
```

Description: Returns the floating-point value in seconds since the start of the program.

Example:

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

Name: IHEREAD, IHERITE

Purpose: Allow PL/I user to gain nearly complete control over I/O, especially line files.

Declaration:

```
DCL (IHEREAD,IHERITE) ENTRY
     ( /* character string preferably varying */,
     BIT (32),   /* a 32-bit modifier */
     DEC FIXED (9,3), /* a line number in the
               range from -99999.999 to
               99999.999*/
     FILE /* file name */   );
```

Description:

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 both for IHEREAD and IHERITE.
4. If the indexed bit of a modifier is on, a line number must be provided. Otherwise, a data interruption may likely occur or some unpredictable results will come up. In addition, in case of IHEREAD, the character string will become a null string when there is no line associated with the line number.

Examples:

```
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 FINSH;

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

Name: PL1ADR

Purpose: To obtain a pointer to PL1 scalar constants and variables.

Declaration: DECLARE PL1ADR RETURNS(POINTER);

Calling Sequence:

```
PL1ADR(ARG)
```

Argument: ARG-any scalar constant or variable (not strings, arrays, or structures).

Result: The value of PL1ADR is the address of the argument.

Names: PLCALL, PLCALLD, PLCALLE, PLCALLF

Purpose: To enable the calling of non-PL1 (FORTRAN and assembler) procedures which require a standard S type linkage from PL1 programs.

Declarations:

```
DECLARE PLCALLD RETURNS(FLOAT, 16);
DECLARE PLCALLE RETURNS(FLOAT, 6);
DECLARE PLCALLF RETURNS(FIXED BINARY(31));
```

Calling Sequence:

```
CALL PLCALL(FN, N, PL);
CALL PLCALLD(FND, N, PL);
CALL PLCALLE(FNE, N, PL);
CALL PLCALLF(FNF, N, PL);
```

Arguments:

FN a subroutine which has been declared to have the ENTRY attribute and which does not return a value.

FND 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 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 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 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 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). 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 must be used which can be produced by PL1ADR, which see. If N=0, there is no parameter list and no comma after N.

Results: The values of PLCALLD, PLCALLE, and PLCALLF are the values returned by FND, FNE, and FNF respectively.

Return Code: The return code placed in general register 15 by FN, FND, FNE, or FNF may be interrogated using PL1RC, which see.

Examples:

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

DECLARE DISMNT ENTRY;
DECLARE 1 PAR ALIGNED STATIC,
          2 LEN BIT(16),
          2 TAPE CHAR(3) INIT('*T*');
LEN=BIN(3, 16, 0);
CALL PLCALL(DISMNT, F1, PAR);
```

Name: PL1RC

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

Declarations: DECLARE PL1RC RETURNS(FIXED BINARY(31));

Calling Sequence:

```
PL1RC
```

Result: The value of PL1RC is the contents of general register 15 when the procedure called using PLCALL, PLCALLD, PLCALLE, or PLCALLF returns: or 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;
```

Name: RAND

Purpose: To compute random numbers between 0.0 and 1.0.

Declaration:

```
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 $(1,2^{31}-2)$. 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.

Example:

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

Name: SIGNOFF

Purpose: To sign off

Declaration: DECLARE SIGNOFF ENTRY;

Description: Closes all open files, if any, and then signs the user off

Example: IF BATCH CALL SIGNOFF;

Name: USERID

Purpose: To obtain the current four character userid

Declaration:
```
DECLARE USERID ENTRY
 RETURNS (CHARACTER (4));
```

Description: Return the userid

Example: PUT LIST (USERID);