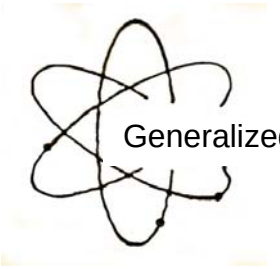
US Army Corps
of Engineers
Hydrologic Engineering Center



HECLIB

Volume 1: HECLIB Subroutines

Programmer's Manual

August 1987

CPD-58

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1987	Computer Program Document No. 58

4. TITLE AND SUBTITLE
HECLIB
Volume 1: HECLIB Subroutines
Programmer's Manual

5. FUNDING NUMBERS

6. AUTHOR(S)
CEWRC-IWR-HEC

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center
609 Second Street
Davis, CA 95616-4687

8. PERFORMING ORGANIZATION REPORT NUMBER
CPD-58

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
N/A

11. SUPPLEMENTARY NOTES
N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release. Distribution of this document is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document provides programmers information on the various subroutines in the library HECLIB. These subroutines are designed to be called by programs written in FORTRAN 77. The reader of this document should have a working knowledge of FORTRAN.

HECLIB has been fully implemented for HARRIS computers and MS-DOS microcomputers. The library is written in FORTRAN 77 and assembly language. The Microsoft® FORTRAN V4.0 compiler was used for the MS-DOS version of this library. HECLIB has been partially implemented for other computers and other compilers on the microcomputer. Several subroutines are written in assembly language to utilize computer capabilities not directly accessible by FORTRAN. These capabilities primarily include I/O for files and terminals.

14. SUBJECT TERMS	15. NUMBER OF PAGES
HECLIB	324
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

HECLIB

Volume 1: HECLIB Subroutines

Programmer's Manual

August 1987

US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center
609 Second Street
Davis, CA 95616

(530) 756-1104
(530) 756-8250 FAX
www.hec.usace.army.mil

CPD-58

HECLIB
Volume 1: HECLIB Subroutines
Software Distribution and Availability Statement

The HECLIB library and documentation are public domain software that was developed by the Hydrologic Engineering Center for the U.S. Army Corps of Engineers. The software was developed at the expense of the United States Federal Government, and is therefore in the public domain. HEC cannot provide technical support for this software to non-Corps users. See our software vendor list (www.hec.usace.army.mil) to locate organizations that provide the program, documentation, and support services for a fee. However, we will respond to all documented instances of program errors. Documented errors are bugs in the software due to programming mistakes not model problems due to user-entered data.

Table of Contents

Chapters

Table of Contents (continued)

Chapters

Table of Contents (continued)

Chapters

Table of Contents (continued)

Chapters

Table of Contents (continued)

Chapters

Table of Contents (continued)

Chapters

Table of Contents (continued)

Chapters

1 Introduction

This document provides programmers information on the various subroutines in the library HECLIB. These subroutines are designed to be called by programs written in FORTRAN 77. The reader of this document should have a working knowledge of FORTRAN.

HECLIB has been fully implemented for HARRIS computers and MS-DOS microcomputers. The library is written in FORTRAN 77 and assembly language. The Microsoft® FORTRAN V4.0 compiler was used for the MS-DOS version of this library. HECLIB has been partially implemented for other computers and other compilers on the microcomputer. Several subroutines are written in assembly language to utilize computer capabilities not directly accessible by FORTRAN. These capabilities primarily include I/O for files and terminals.

1.1 Machine Specifics

1.1.1 HARRIS Computers

HECLIB subroutines are accessed by linking in the FORTRAN 77 version of HECLIB. The location of the library may vary on different machines, but most often it can be found in either qualifier 2000SYSS (2000SYSS*HECLIB), or HLIB (HLIB*HLIB77). Note that there are FORTRAN 66 versions of HECLIB that will not work with programs using these FORTRAN 77 calls. A typical compilation and linking is as follows:

```
SAUF77 MYSOURCE
VU.R MYPROG
LIB 2000SYSS*HECLIB *LIBERY
BEGIN
```

1.1.2 Microcomputers Using MS-DOS

HECLIB has been fully implemented on microcomputers with Microsoft® FORTRAN Version 4.0. (HECLIB is incompatible with earlier versions of this compiler.) The library has been partially implemented for Lahey® and Ryan-McFarland® (Professional FORTRAN) compilers. The subroutines which have been implemented for Lahey® and Ryan-McFarland® compilers are the general ones found at the beginning of each section (they do not include those listed as MS-DOS specific).

All subroutines are compiled with a word length of INTEGER*2, except for a few specific subroutines. Programs accessing subroutines in HECLIB should either be compiled with a 2 byte integer word default (MS FORTRAN option /4I2), or with integer and logical variable declared as INTEGER*2 (except where noted otherwise). An exception to this are the Julian dates and the time interval used in several of the time and date routines, and disk

positioning variables used in several of the disk I/O subroutines. These variables must be passed as INTEGER*4.

The Microsoft® version of HECLIB is named HECLIBMS.LIB. The library assumes the large memory model and that the math co-processor is optional (option /Fpi). A typical compilation and linking of program using this library is as follows:

```
FL /c /4I2 /Gt /Od /Fpi myfile.for
LINK myfile,,,HECLIBMS
```

2 File Input/Output and Handling Subroutines

The following section describes the HECLIB subroutines that are generally used in the Input/Output (I/O) and handling of files. This includes subroutines for connecting files to programs, renaming, creating, deleting files, as well as direct access to assembly I/O. Some subroutines (e.g., ATTACH, GIOP) are applicable to terminal I/O as well as for files.

For HARRIS computers, the GIOP (General I/O Processing) provides access to all the low level I/O functions. On MS-DOS microcomputers, the disk-buffer I/O subroutines use low level I/O to read or write single lines, considerably faster than what may be obtained through FORTRAN I/O. Some of the file positioning used for the MS-DOS subroutines use INTEGER*4 words.

2.1 ATTACH – Attach Files to Units via Execution Line Parameters

Purpose:

Subroutine ATTACH uses information on the program execution line to open files, or pass execution line information to the program. This allows the program user to either connect their own files with the program, or to use the program's default files. Filenames and information are passed on the execution line by a keyword followed by an equal sign (=), then the file name or information. For example:

MYPROG INPUT=MYDATA OUTPUT=MYOUT

If the user enters a question mark (?) directly after the program name, ATTACH will print all keywords and default file names then stop.

If the computer system cannot provide the execution line to the ATTACH subroutine, the files names will be prompted for.

ATTACH is designed to be called at the beginning of the program. A call to subroutine ATTEND must follow the last call to ATTACH. Subroutine ATTSET may be called prior to the first call to ATTACH to have the program version or other information printed when the user enters a question mark on the execution line.

Calling Sequence:

CALL ATTACH (IUNIT, CKEYWD, CDEFLT, CONTRL, CNAME, IOSTAT)

Declarations:

```
INTEGER IUNIT, IOSTAT
CHARACTER CKEYWD, CDEFLT, CONTRL, CNAME
```

Argument Description:

IUNIT	Input	The unit number to open the specified file with. If execution line information only is to be passed to the program, this argument is ignored.
CKEYWD	Input	The keyword that identifies the file to open, or the information to pass. The keyword is given on the execution line (or is used in the prompt) to identify the file to open. In the above example "INPUT", and "OUTPUT" are keywords. A keyword must not contain blanks, but may be abbreviated (as long as the abbreviation is unique).
CDEFLT	Input	The default file to open, or information to pass, if the user does not specify the keyword on the execution line. The default

name may be a special reserved name to connect certain files. The default 'STDIN' will connect to the standard input, and 'STDOUT' will connect to the standard output. A list of the reserved names follows (under Notes).

CONTRL Input This character string defines the file parameters that are generally used in a OPEN statement. Parameters are separated by either a comma or a blank. To use all default values for CONTRL, provide a blank string (' ') (this is the same as CONTRL='A=S,F=F,P=N,S=U'). Refer to the OPEN statement in your FORTRAN manual for further information on the following parameters. The following control parameters are recognized by ATTACH:

Parameter	Description
A	Access. The file access (either Sequential or Direct) is specified by either A=S, or A=D. If a file is specified as direct, the record length must follow the "D", separated by a forward slash (/). For example, to open a direct access file with a record length of 512 bytes, CONTRL would be 'A=D/512'. If no Access parameter is specified, the default is sequential. (See the ACCESS and RECL parameters in the FORTRAN OPEN documentation.)
F	Form. To indicate whether the file is being opened for formatted or unformatted I/O specify either F=U or F=F. If no form is specified, the default is formatted.
NOP	No operation. No files are to be opened; information only is to be passed to the program from the execution line.
P	Prompt. Where a filename is required and was omitted on the execution line, it may be prompted for during the execution of the program. This is controlled by either a 'P=Y' for yes, or a 'P=N' for no. If no is used, the default file (CDEFLT) will automatically be opened. The default is no.
S	Status. The status of the file (New, Old, Scratch, or Unknown) is specified by either S=N, S=O, S=S, or S=U. If the status is new, and the file exists, then the user will be prompted for a decision of overwriting the file. The default status is Unknown. (See the STATUS parameter in the FORTRAN OPEN documentation.)

CNAME	Output	CNAME is returned with the name of the file opened (either the specified or the default name), or the information that was obtained from the execution line. CNAME must be declared long enough to hold the longest name that might be used.
IOSTAT	Output	A status parameter indicating the successfulness of the OPEN. If IOSTAT is less than or equal to zero, then the OPEN was successful. If IOSTAT is greater than zero, an error occurred, and the value of IOSTAT corresponds to the IOSTAT values given in the OPEN statement of the FORTRAN manual. The successful IOSTAT values are:

IOSTAT	**Description**
0	Open performed successfully. The default file name was used.
-1	Open performed successfully. This file was specified on the execution line.
-2	Open performed successfully. All default files were used (no keywords were given on the execution line).
-10	The user entered a question mark (?) on the execution line to determine the keywords and default file names. No files are opened, and no information is passed. The program will stop when the call to ATTEND is reached.

Remarks:

ATTACH will WIND a file when the file name specified is preceded by a plus sign (+). This will cause any information to be written to the file to be appended at the end of the file.

The subroutine ATTEND must be called after the last call to ATTACH. This indicates the stopping point when the user enters a question mark on the execution line (to obtain the keywords and default file names), or an unrecognized keyword is encountered. ATTSET may be called prior to the first call to ATTACH to pass information (such as the program version) to be printed when a question mark is entered on the execution line.

Example:

```
CHARACTER CNAME*64, CDSSFI*64, CYEAR*4

CALL ATTSET ('MYPROG:  December 31, 1980 Version')
CALL ATTACH (5, 'INPUT', 'STDIN', 'S=O', CNAME, ISTAT)
CALL ATTACH (6, 'OUTPUT', 'STDOUT', ' ', CNAME, ISTAT)
CALL ATTACH (8, 'TABLE1', '+MYTABLE', 'S=U/PR/OW', CNAME, ISTAT)
CALL ATTACH (9, 'SCRATCH1', 'SCRATCH1', ' ', CNAME, ISTAT)
CALL ATTACH (10, 'SCRATCH2', 'SCRATCH30', 'F=U', CNAME, ISTAT)
CALL ATTACH (0, 'YEAR', ' ', 'NOP', CYEAR, ISTAT)
```

```
CALL ATTACH (0, 'DSSFILE', ' ', 'NOP', CDSSFI, ISTAT)
CALL ATTEND
```

HARRIS Notes:

CONTROL Parameters

The file access may be specified in the "S" (status) CONTROL parameter to use when a file is created by ATTACH. This is accomplished by placing a slash following the U or N parameter, then the file access (those given in a HARRIS Map command). For example, to have the file created with public read, owner write, and owner delete access, enter the CONTRL parameter as:

'S=U/PR/OW/OD'

The default access level used is public read, public write and public delete.

A file may be created as unblocked or random access with the "A" (access) parameter for direct access files. To accomplish this, follow the record length with a slash (/) then a "U" (for unblocked) or a "R" (for random). For example, 'A=D/512/R'. The default is a unblocked file.

A file may be assigned in a exclusive or shared mode by use of the mode (M) parameter. An exclusive assignment is made with a 'M=E' control parameter. A shared assignment is made with a 'M=S' control parameter. The default is a normal assignment.

The above control parameters will be ignored on other systems.

HARRIS Reserved Filenames (for CDEFLT)

STDIN is attached to unit 0.
STDOUT is attached to unit 3.

SCRATCH1 through SCRATCH10 are attached to blocked work files W1 through W0 (W1, W2, W3, W4, W5, W6, W7, W8, W9, W0).

SCRATCH11 through SCRATCH20 are attached to blocked work files T1 through T0.

SCRATCH21 through SCRATCH30 are attached to blocked work files S1 through S0. (Caution: The S work files may not be accessible at some sites.)

SCRATCH31 through SCRATCH40 are attached to unblocked work files U1 through U0.

SCRATCH41 through SCRATCH50 are attached to unblocked work files H1 through H0. (Caution: The H work files may not be accessible at some sites.)

MS-DOS Notes:

STDIN is the keyboard and STDOUT is the screen, unless redirected (using > or <).

SCRATCH1 through SCRATCH999 will create files named SCRATCH with extensions of .001 through .999 in the default directory. These files are not eliminated at the end of the program execution unless they are declared scratch in the CONTRL parameter (S=S), or explicitly deleted in the CLOSE statement (e.g., CLOSE (UNIT=18,STATUS=DELETE)).

2.2 ATTEND – End of ATTACH Calls

Purpose:

ATTEND must be called after the last ATTACH call. This indicates to ATTACH where to stop the program execution when the user enters a question mark on the execution line to print the program's keywords and default file names. Unrecognized keywords are also identified at this point.

Calling Sequence:

CALL ATTEND

2.3 ATTSET – Set ATTACH Information

Purpose:

ATTSET provides a means of printing one line of information when the user enters a question mark on the execution line. This information is often the version date of the program. ATTSET must be called prior to the first call to ATTACH.

Calling Sequence:

CALL ATTSET (CLINE)

Declarations:

CHARACTER CLINE

Argument Description:

CLINE	Input	The line of information to be printed out when a question mark is entered on the execution line. Up to 132 characters may be printed.

Example:

```
CALL ATTSET ('MYPROG:  July 4, 1976;  5 reservoir limit')
CALL ATTACH (...
```

2.4 WIND – Position to the End of File

Purpose:

WIND positions a unit to the end of the file so that any writing to that unit will append to the file instead of replacing information in the file. WIND is the opposite of REWIND. WIND only operates on files (not on terminals).

Calling Sequence:

```
CALL WIND (IUNIT)
```

Declarations:

```
INTEGER IUNIT
```

Argument Description:

IUNIT	Input	The unit number of the opened file to position to the end of file.

2.5 RECMAX – Determine the Number of Records (Lines) in a File

Purpose:

Subroutine RECMAX determines the number of lines in a blocked file (or the number of sectors in an unblocked file for HARRIS computers).

Calling Sequence:

CALL RECMAX (IUNIT, NRECS)

Declarations:

INTEGER IUNIT, NRECS

Argument Description:

IUNIT	Input	The unit number connected to the file to determine the number of records (lines). The file must have been opened.
NRECS	Output	The number of records (lines) in the file for a blocked file, or the number of sectors in the file for an unblocked file.

2.6 NUMLN – Determine the Number of Lines in a File

Purpose:

Integer function NUMLIN determines the number of lines in a file, given the file name. Use subroutine RECMAX to determine the number of lines in a file that has already been opened.

Calling Sequence:

INUMB = NUMLIN (CNAME)

Declarations:

```
CHARACTER CNAME
INTEGER NUMLIN
```

Argument Description:

CNAME	Input	The name of the file to find the number of lines.
NUMLN	Output	The number of lines in file CNAME.

2.7 LISFIL – Determine if a Name is a Valid Filename

Purpose:

Logical function LISFIL determines if a given name is a valid file name. LISFIL does not indicate if the file exists or not, just whether the name given meets the specifications for a file name.

Calling Sequence:

LNAME = LISFIL (CNAME)

Declarations:

```
CHARACTER CNAME
LOGICAL LISFIL
```

Argument Description:

CNAME	Input	The name to be checked.
LISFIL	Output	A logical flag returned .TRUE. if CNAME met the specifications for a file name.

2.8 GETNAM – Get the Name of an Opened File

Purpose:

GETNAM returns the name of a file attached to a specified unit. This is identical to the FORTRAN INQUIRE statement for "NAME", except that on HARRIS computers the file name is returned in a usable form (see remarks).

Calling Sequence:

```
CALL GETNAM (IUNIT, CNAME, IERR)
```

Declarations:

```
CHARACTER CNAME
INTEGER IUNIT, IERR
```

Argument Description:

IUNIT	Input	The unit number the file is attached to. The file must be assigned, but does not have to be opened.
CNAME	Output	The name of the file attached to IUNIT.
IERR	Output	A status parameter indicating the successfulness of the call. If IERR is returned as zero (0), CNAME contains the file name. On HARRIS computers, if IERR is returned as negative one (-1), the unit number is not attached to a file. If IERR is returned greater than one, the unit is attached to a physical device and IERR is the PDN (physical device number).

Remarks:

The INQUIRE statement on the HARRIS does not return the name of a file in a way that is directly usable. GETNAM returns the name in a form that can be used in OPENS, ASSIGNS, etc. For example, GETNAM will return a file name such as '0000SYS*MYFILE'. On HARRIS computers, GETNAM calls LFNAME then rearranges the file name. On non-HARRIS computers, GETNAM does a direct INQUIRE.

Example:

```
CALL GETNAM (9, CNAME, IERR)
IF (IERR.NE.0) GO TO 100
CLOSE (UNIT=9)
OPEN (UNIT=12, FILE=CNAME, IOSTAT=ERR)
```

2.9 CCREAT – Create a File

Purpose:

CCREAT creates a file. On HARRIS computers, the granule size and pack may be specified.

Calling Sequence:

CALL CCREAT (CNAME, IGRAN, IPACK, ITYPE, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IGRAN, IPACK, ITYPE, IERR
```

Argument Description:

CNAME	Input	A character string containing the name of the file to create
IGRAN	Input	The granule size of the file to be created. If zero, the default size will be used.
IPACK	Input	The pack number of where to generate the file. If zero, the default pack will be used.
ITYPE	Input	A flag indicating the type of file to create. If ITYPE is zero, a blocked file will be created. If ITYPE is -1, an unblocked file will be created. If ITYPE is -2, a random access unblocked file will be created.
IERR	Output	A status parameter indicating the successfulness of the call. If IERR is returned as zero, the file was created successfully.

Remarks:

CCREAT is the same subroutine as the HARRIS CREATE subroutine, except that the file name is specified as a character string instead of a Hollerith array. CCREAT converts the file name to Hollerith, then calls the HARRIS CREATE subroutine. See the CREATE subroutine documentation in the HARRIS FORTRAN manual for more information.

2.10 CDELET – Delete a File

Purpose:

CDELET eliminates a file. The user of the calling program must have delete access for the file.

Calling Sequence:

CALL CDELET (CNAME, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IERR
```

Argument Description:

CNAME	Input	The name of the file to delete.
IERR	Output	A status parameter indicating the successfulness of the delete. If IERR is returned as zero, the file was deleted.

Remarks:

The file must not be opened or otherwise in use to delete it. Refer to the FORTRAN manual for error codes other than zero.

2.11 CRENAM – Rename a File

Purpose:

CRENAM renames a file. The user of the calling program must have delete access for the file.

Calling Sequence:

CALL CRENAM (COLDN, CNEWN, IERR)

Declarations:

```
CHARACTER COLDN, CNEWN
INTEGER IERR
```

Argument Description:

COLDN	Input	The current name of the file to be renamed.
CNEWN	Input	The new name to be given to the file.
IERR	Output	A status parameter indicating the successfulness of the rename. If IERR is returned as zero, the file was renamed successfully.

Remarks:

The file must not be opened or otherwise in use to rename it. Refer to the FORTRAN manual for error codes other than zero.

2.12 HARRIS Specific Subroutines

2.12.1 GIOP – General Input/Output Processing

Purpose:

Subroutine GIOP provides direct FORTRAN access to HARRIS assembly I/O functions. These functions include all read-write operations, and special terminal operations. They are described in the VOS I/O Services Reference Manual. This manual should be referred to when using GIOP.

Four versions of GIOP exist. The first, called GIOP, initiates a function that makes use of an input-output buffer. The second, named GIOPLW, does the same as the first, but then does a normal status call (which is often required to complete the function). GIOPLW will not return until the function has completed (or an error occurred). The third, called GIOPS, initiates a function that does not use an input-output buffer. The fourth, named GIOPSW, initiates the function as in GIOPS, but then does a status call. Example uses follow.

Calling Sequence:

GIOP Long Call
CALL GIOP (IUNIT, IFUN, IBUFF, NBUFF, ISTAT

GIOP Long Call with Wait (status)
CALL GIOPLW (IUNIT, IFUN, IBUFF, NBUFF, ISTAT)

GIOP Short Call
CALL GIOPS (IUNIT, IFUN, ISTAT)

GIOP Short Call with Wait (status)
CALL GIOPSW (IUNIT, IFUN, ISTAT)

Declarations:

INTEGER IUNIT, IFUN, IBUFF(NBUFF), ISTAT

Argument Description:

IUNIT	Input	The unit to perform the function on. The unit must be assigned prior to calling GIOP.
IFUN	Input	The function to perform. The functions are the octal numbers given in the VOS I/O Services Manual.

IBUFF	Input/ Output	The buffer containing the information to be written, or the buffer in which to place the data read. IBUFF must always be an integer array, regardless of the type of data to be transferred.
NBUFF	Input	The number of words of IBUFF to transfer.
ISTAT	Output	A status parameter containing information regarding the success of the call. This is the information returned in the A register. ISTAT is rarely returned with zero, as several pieces of information are returned indicated by what bits are set. Subroutine CSTAT may be used to decode the status parameter.

Remarks:

The unit must always be assigned prior to calling GIOP. This is normally done through an ASSIGN service (not a FORTRAN OPEN).

Do not mix different I/O modes; If you call GIOP for I/O with a file, do not use any FORTRAN I/O until that file has been closed and reopened with a FORTRAN OPEN statement. An exception to this is terminal I/O, where usually both modes of I/O can be performed.

IBUFF must always be an integer array, regardless of the type of data being transferred. If another type of data is to be written, it must first be converted into an integer array. For example, if character data is to be written or read, that character variable can be equivalent to IBUFF.

Refer to the VOS I/O Services Reference Manual for function codes.

Example Calls:

The following list provides sample calls for the commonly used I/O services. The following calls assume that IBUFF has been dimensioned to NBUFF integer words, and NBUFF words are to be transferred

Symbolic Read: CALL GIOPLW (IUNIT, '01, IBUFF, NBUFF, ISTAT)
Symbolic Write: CALL GIOPLW (IUNIT, '02, IBUFF, NBUFF, ISTAT)
Binary Read: CALL GIOPLW (IUNIT, '03, IBUFF, NBUFF, ISTAT)
Binary Write: CALL GIOPLW (IUNIT, '04, IBUFF, NBUFF, ISTAT)
Open (requires ASSIGN first): CALL GIOPSW (IUNIT, '13, ISTAT)
Close: CALL GIOPSW (IUNIT, '14, ISTAT)
Advance File: CALL GIOPSW (IUNIT, '16, ISTAT)
Rewind File: CALL GIOPSW (IUNIT, '22, ISTAT)
Move to Sector: CALL GIOPLW (IUNIT, '23, IDUM, NSECT, ISTAT)
Dump Buffer: CALL GIOPSW (IUNIT, '24, ISTAT)
Terminal Backstore: CALL GIOPLW (IUNIT, '27, IBUFF, NBUFF, ISTAT)
Flush Buffer: CALL GIOPS (IUNIT, '37, ISTAT)

Transmit Break: CALL GIOPSW (IUNIT, '50, ISTAT)
Enable Hot Read: CALL GIOPLW (IUNIT, '51, IBUFF, NBUFF, ISTAT)
Hot Read with Wait: CALL GIOPSW (IUNIT, '51, ISTAT)
Hot Write: CALL GIOPLW (IUNIT, '52, IBUFF, NBUFF, ISTAT)
Hot Read No Wait: CALL GIOPSW (IUNIT, '53, ISTAT)

2.12.2 CSTAT – Pick Apart an I/O Service Status

Purpose:

CSTAT is used to pick apart the status word returned from a system service I-O. (This is the status value returned by the subroutine GIOP.)

Calling Sequence:

CALL CSTAT (ISTAT, IOK, LOK, LEOF, LOPEN, LXDISC, IWC, LWCNC)

Declarations:

```
INTEGER ISTAT, IOK, IWC
LOGICAL LOK, LEOF, LOPEN, LXDISC, LWCNC
```

Argument Description:

ISTAT	Input	The status word returned from the I/O call.
IOK	Output	An integer flag indicating if the operation was successful. IOK is returned with zero (0) if the operation was completed, otherwise IOK is returned as one (1).
LOK	Output	A logical flag indicating if the operation was successful. LOK is returned as .TRUE. if the operation was completed, otherwise LOK is returned .FALSE.. (Similar to IOK, except a logical flag).
LEOF	Output	A logical flag that indicates if the I/O call reached the end of file. LEOF is returned .TRUE. if the end of file condition was met.
LOPEN	Output	A logical flag indicating if the file is open or not. LOPEN is returned .TRUE. if the file is open.
LXDISC	Output	A logical flag indicating if the last operation exceed a disc space bounds (either users, pack, or system disc space). LXDISC is returned .TRUE. if the disc space limit was reached.
IWC	Output	Word Count. IWC is an integer variable indicating the number of words transferred on the I/O operation.
LWCNC	Output	Word Count Not Complete. LWCNC is a logical flag that is returned .TRUE. if the number of words transferred in the I/O operation is incomplete.

2.12.3 CRETYP – Retype the Attributes of a File

Purpose:

CRETYP is the same subroutine as the HARRIS RETYPE subroutine, except that the file name is specified as a character instead of a Hollerith array.

Calling Sequence:

```
CALL CRETYP (CNAME, IBITS, ILEVEL, IERR)
```

Declarations:

```
CHARACTER CNAME
INTEGER IBITS, ILEVEL, IERR
```

Argument Description:

CNAME	Input	A character string containing the name of the file to retype.
IBITS	Input	The access bits to set. These bits contain information on the read, write, execute and delete access. See the system service $RTYPE for information.
ILEVEL	Input	The access level to set for the file.
IERR	Output	A status parameter indicating the successfulness of the retype. If IERR is returned as zero, the file was retyped successfully.

Remarks:

Converts the file name to Hollerith, then calls the HARRIS RETYPE subroutine. See the RETYPE subroutine documentation in the HARRIS FORTRAN manual for more information.

Example:

Retype a file to public read, write, delete access:

```
CALL CRETYP ('RES*MYFILE', 116, 0 ,IERR)
```

2.12.4 IFTYPE – Determine the Type of File Assigned

Purpose:

Function IFYTPE returns the type of file assigned to a unit. The different file types are blocked, unblocked, and random access. The file must be assigned, but does not have to be opened.

Calling Sequence:

ITYPE = IFTYPE (IUNIT)

Declarations:

INTEGER IFTYPE, IUNIT

Argument Description:

IUNIT	Input	The unit number that the file is assigned to.
IFTYPE	Output	A flag indicating the type of file assigned. IFTYPE is returned with five possible values:

Value	File Type
0	Blocked
1	Unblocked
2	Random
-1	Unassigned
-2	Physical Device

2.12.5 CASSIG – Assign a Unit to a File

Purpose:

CASSIG is the same subroutine as the HARRIS ASSIGN subroutine, except that the file name is specified as a character instead of a Hollerith array.

Calling Sequence:

CALL CASSIG (IUNIT, CNAME, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IUNIT, IERR
```

Argument Description:

IUNIT	Input	The unit number to assign the file to.
CNAME	Input	A character string containing the name of the file to assign.
IERR	Output	A status parameter indicating the successfulness of the assign. If IERR is returned as zero, the file was assigned successfully.

Remarks:

Converts the file name to Hollerith, then calls the HARRIS ASSIGN subroutine. See the ASSIGN subroutine documentation in the HARRIS FORTRAN manual for more information.

2.12.6 ASSIGX – Assign a File in an Exclusive Mode

Purpose:

ASSIGX assigns a file in an exclusive mode on HARRIS computers. In this mode, no other users (or other units) may connect to the file until the assignment is broken. If the file is already assigned (any type of assignment) by another user, the exclusive assign will fail. The file may be a sequential access or direct access file.

Calling Sequence:

CALL ASSIGX (IUNIT, CNAME, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IUNIT, IERR
```

Argument Description:

IUNIT	Input	The unit number to assign the file to.
CNAME	Input	A character string containing the name of the file to assign.
IERR	Output	A status parameter indicating the successfulness of the shared assign. If IERR is returned as zero, the file was assigned successfully. Error code 10 is returned if the file is assigned by some other user.

Remarks:

If the file is already assigned (e.g., by another user), the assign will fail and return an error of ten. See the $ASSIGN documentation in the VOS System Service's Manual for more information.

2.12.7 ASSIGS – Assign a File in a Shared Mode

Purpose:

ASSIGS assigns a direct access file for shared file operations on HARRIS computers. In this mode, two or more users may write to the file at the same time using record and file locks (see subroutine FLLKON). A file may be connected in this mode only if it is a HARRIS random file, and all other assignments are in the shared mode also.

Calling Sequence:

CALL ASSIGS (IUNIT, CNAME, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IUNIT, IERR
```

Argument Description:

IUNIT	Input	The unit number to assign the file to.
CNAME	Input	A character string containing the name of the file to assign.
IERR	Output	A status parameter indicating the successfulness of the shared assign. If IERR is returned as zero, the file was assigned successfully. Error code 23 is returned if the file is assigned by some other user in a non-shared mode, or the file is not a direct access file.

Remarks:

The system GEN file must specify the 'SHARED-FILES' capability. See the VOS Site Manager's Manual for more information.

If the file is assigned with a non-shared assignment (e.g., by another user), the assign will fail and return an error of 23. See the $ASSIGN documentation in the VOS System Service's Manual for more information. See the FLLKON subroutine documentation for information on shared-assign use.

2.12.8 FLLKON – Lock a Shared Access File

Purpose:

FLLKON "locks" a file that has been assigned to a program in a shared access mode (see subroutine ASSIGS). This lock prevents any other program (who also has a shared assignment to that file) from reading or writing to the file until the lock is removed (using subroutine FLLKOF).

Calling Sequence:

CALL FLLKON (IUNIT, IWAIT, ISTAT)

Declarations:

INTEGER IUNIT, IWAIT, ISTAT

Argument Description:

IUNIT	Input	The unit number connected to the file. The file must be a random access file and must have been assigned in a shared access mode.
IAWIT	Input	A flag indicating whether the subroutine should wait until the file is unlocked if it has already been locked by another user. If IWAIT is one (1), the subroutine will wait until the file has been unlocked. If IWAIT is zero (0), it will return immediately without locking the file (if unavailable).
ISTAT	Output	A status parameter. If ISTAT is returned zero, the file was successfully locked, otherwise not.

Remarks:

The file must be a unblocked or random access file, in a shared access mode. Refer to the VOS I/O Services manual, unblocked/random disc area I/O section (function code '25) for more information about file locking and return status codes.

2.12.9 FLLKOF – Unlock a Locked File

Purpose:

FLLKOF "unlocks" a file that has been locked by subroutine FLLKON, allowing other users to read and write to the file. Refer to subroutine FLLKON for more information.

Calling Sequence:

CALL FLLKOF (IUNIT, ISTAT)

Declarations:

INTEGER IUNIT, ISTAT

Argument Description:

IUNIT	Input	The unit number connected to the file. The file must be a random access file and must have been assigned in a shared access mode.
ISTAT	Output	A status parameter. If ISTAT is returned zero, the file was successfully unlocked.

2.13 MS-DOS Specific Subroutines

Purpose:

The disk-buffer I/O subroutines provide fast I/O on files for MS-DOS microcomputers. These subroutines use an integer buffer to read or write a large amount of data at one time. The disk-buffer subroutines are on the order of five times faster than most FORTRAN I/O.

Subroutine Summary:

DKBFOP - Open a file (must exist)
DKBFCR - Create (or truncate) a file and open
DKBFCL - Close the file
DKBFRD - Read from the file
DKBFWT - Write to the file
DKBFPS - Position to a byte within the file

2.13.1 DKBFOP – Disk-Buffer Open

Purpose:

Open a file for disk-buffer I-O. The file must exist.

Calling Sequence:

CALL DKBFOP (IHANDL, CNAME, IBUFF, NBUFF, ISTAT)

Declarations:

```
INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT
CHARACTER CNAME
```

Argument Description:

IHANDL	Output	The handle number given to the file. This is similar to a FORTRAN unit number, and must be used for all DKBF calls for that file. (Use a different handle variable for a different file).
CNAME	Input	The name of the file to perform I/O on.
IBUFF	Input/ Output	An array used for buffering I/O, dimensioned to NBUFF. Typically, IBUFF is dimensioned to 2058, but may range from 74 to 8192. A larger buffer size generally gives faster I/O. This same array should be passed to the other DKBF subroutines for this file.
NBUFF	Input	The dimension of IBUFF, in INTEGER*2 words.
ISTAT	Output	A status parameter, set to zero if the call was successful. Non-zero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.2 DKBFCR – Disk-Buffer Create (or Truncate) File and Open

Purpose:

Creates, and then opens a new file for disk-buffer I/O. If the file already exists, any information in the file will be eliminated.

Calling Sequence:

CALL DKBFCR (IHANDL, CNAME, IBUFF, NBUFF, ISTAT)

Declarations:

```
INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT
CHARACTER CNAME
```

Argument Description:

IHANDL	Output	The handle number given to the file. This is similar to a FORTRAN unit number, and must be used for all DKBF calls for that file. (Use a different handle variable for a different file).
CNAME	Input	The name of the file to perform I/O on.
IBUFF	Input/ Output	An array used for buffering I-O, dimensioned to NBUFF. Typically, IBUFF is dimensioned to 2058, but may range from 74 to 8192. A larger buffer size generally gives faster I/O. This same array should be passed to the other DKBF subroutines for this file.
NBUFF	Input	The dimension of IBUFF, in INTEGER*2 words.
ISTAT	Output	A status parameter, set to zero if the call was successful. Non-zero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.3 DKBFCL – Disk-Buffer Close

Purpose:

Closes the file (opened by DKBFOP or DKBFCR), dumping the buffer if necessary.

Calling Sequence:

CALL DKBFCL (IHANDL, IBUFF, ISTAT)

Declarations:

INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT

Argument Description:

IHANDL	Output	The handle number from DKBFOP or DKBFCR.
IBUFF	Input	The buffer array from DKBFOP or DKBFCR.
ISTAT	Output	A status parameter, set to zero if the call was successful. If the buffer was not initialized (with DKBFOP or DKBFCR), ISTAT is returned with a -3. Positive error codes may be found on page 6-42 of the DOS Technical Reference Manual.

Remarks:

A file opened by DKBFOP or DKBFCR should always be closed by this subroutine.

2.13.4 DKBFRD – Disk-Buffer Read

Purpose:

Reads a single line from a file. The file must have been opened with either DKBFOP or DKBFCR.

Calling Sequence:

CALL DKBFRD (IHANDL, CLINE, NLINE, IBUFF, ISTAT)

Declarations:

```
INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT, NLINE
CHARACTER CLINE
```

Argument Description:

IHANDL	Output	The handle number from DKBFOP or DKBFCR.
CLINE	Output	The line read from the file. The number of characters returned will not be greater than the declared length of CLINE.
NLINE	Output	A status parameter, set to zero if the call was successful. If the buffer was not initialized (with DKBFOP or DKBFCR), ISTAT is returned with a -3. Positive error codes may be found on page 6-42 of the DOS Technical Reference Manual.
IBUFF	Input/ Output	The buffer array from DKBFOP or DKBFCR.
ISTAT	Output	A status parameter, set to zero if the call was successful. If at the end of the file, ISTAT is returned with a -1. If the buffer was not initialized (with DKBFOP or DKBFCR), ISTAT is returned with a -3. Positive error codes may be found on page 6-42 of the DOS Technical Reference Manual.

Remarks:

DKBFRD reads blocks of information from the file, NBUFF words at a time. The line returned is from this block (or buffer). Physical reads are done only when the line requested is outside of the current block.

2.13.5 DKBFWT – Disk-Buffer Write

Purpose:

Writes a single line to a file. The file must have been opened with either DKBFOP or DKBFCR.

Calling Sequence:

CALL DKBFWT (IHANDL, CLINE, IBUFF, ISTAT

Declarations:

INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT
CHARACTER CLINE

Argument Description:

IHANDL	Input	The handle number from DKBFOP or DKBFCR.
CLINE	Input	The line to write to the file. The number of characters to write is implied by the length of CLINE (e.g., CLINE(1:20).
IBUFF	Input/ Output	The buffer array from DKBFOP or DKBFCR. Do not use the same buffer to read and write with.
ISTAT	Output	A status parameter, set to zero if the call was successful. If the buffer was not initialized (with DKBFOP or DKBFCR), ISTAT is returned with -3. Other error codes may be found on page 6-42 of the DOS Technical Reference Manual.

Remarks:

DKBFWT writes blocks of information to the file, NBUFF words at a time. The line passed to DKBFWT is stored in the buffer. The buffer is not dumped to disk until a reference outside the block is requested, or the file is closed. It is important to close the file with DKBFCL to insure the buffer has been dumped.

Do not use the same buffer to read and write with.

2.13.6 DKBFPS – Disk-Buffer Position

Purpose:

Positions to a specified byte in the file. DKBFPS will wind to the end of the file by setting IBYTE to -1, or return the current by position by setting IBYTE to zero. The file must have been opened with either DKBFOP or DKBFCR.

Calling Sequence:

CALL DKBFPS (IHANDL, IBYTE, IPOS, IBUFF, ISTAT)

Declarations:

INTEGER*2 IHANDL, IBUFF(NBUFF), ISTAT
INTEGER*4 IBYTE, IPOS

Argument Description:

IHANDL	Input	The handle number from DKBFOP or DKBFCR.
IBYTE	Input	The byte number to position to (where 1 is the first byte in the file). To position to the end of the file, set IBYTE to -1. To get the current position, set IBYTE to 0. IBYTE must be INTEGER*4.
IPOS	Output	The resulting byte position (usually equal to IBYTE unless an error occurred or the position was requested. IPOS must be INTEGER*4.
IBUFF	Input/ Output	The buffer array from DKBFOP or DKBFCR.
ISTAT	Output	A status parameter, set to zero if the call was successful. If the buffer was not initialized (with DKBFOP or DKBFCR), ISTAT is returned with -3. Positive error codes may be found on page 6-42 of the DOS Technical Reference Manual.

Remarks:

If reposition to a different block, the block (buffer) will be dumped to the disk.

2.13.7 OPENF – Open a File

Purpose:

OPENF is a low-level subroutine that opens an old file. Refer to the Open function (3DH) in the DOS Technical Reference Manual for more information (page 6-126).

Calling Sequence:

CALL OPENF (CNAME, IACESS, IHANDL, ISTAT)

Declarations:

```
CHARACTER CNAME
INTEGER*2 IACESS, IHANDL, ISTAT
```

Argument Description:

CNAME	Input	The name of the file to open. This file name must be terminated by a zero value byte (e.g., CNAME//CHAR(0)).
IACESS	Input	The file access. The accesses are: 0 Requires read access only. 1 Requires write access only. 2 Requires both read and write access.
IHANDL	Output	The file handle. This is similar to the FORTRAN unit number, but the number is assigned by the open function.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.8 CREAF – Create a File

Purpose:

CREAF is a low-level subroutine that creates and opens a new file or truncates an old file to zero length for preparation for writing. Refer to the CREAT function (3CH) in the DOS Technical Reference Manual for more information (page 6-122).

Calling Sequence:

CALL CREAF (CNAME, IFATT, IHANDL, ISTAT)

Declarations:

```
CHARACTER CNAME
INTEGER*2 IFATT, IHANDL, ISTAT
```

Argument Description:

CNAME	Input	The name of the file to create (or truncate). This file name must be terminated by a zero value byte (e.g., CNAME//CHAR(0)).
IFATT	Input	The file attributes, as described on page 5-11 of the DOS Technical Reference Manual. This should be set to zero for normal files.:
IHANDL	Output	The file handle. This is similar to the FORTRAN unit number, but the number is assigned by the create function.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.9 CLOSF – Close a File

Purpose:

CLOSF is a low-level subroutine that closes a file opened by OPENF or CREAF. Refer to the Close function (3EH) in the DOS Technical Reference Manual for more information (page 6-136).

Calling Sequence:

```
CALL CLOSF (IHANDL, ISTAT)
```

Declarations:

```
INTEGER*2 IHANDL, ISTAT
```

Argument Description:

IHANDL	Input	The file handle from OPENF or CREAF.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.10 READF – Read From a File

Purpose:

READF is a low-level subroutine that reads an integer buffer from a file opened by OPENF or CREAF. READF does not read individual lines, but a specified number of bytes. Refer to the Read function (3FH) in the DOS Technical Reference Manual for more information (page 6-137).

Calling Sequence:

CALL READF (IHANDL, IBUFF, NBYTES, ISTAT, NTRANS)

Declarations:

INTEGER*2 IHANDL, IBUFF, NBYTES, ISTAT, NTRANS

Argument Description:

IHANDL	Input	The file handle from OPENF or CREAF.
IBUFF	Output	An integer buffer to contain the information read.
NBYTES	Input	The number of bytes to read.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.
NTRANS	Output	The number of bytes actually read. NTRANS will be less than NBYTES if the end of file position was reached. NTRANS will be zero if the file position was at the end of file.

2.13.11 WRITF – Write to a File

Purpose:

WRITF is a low-level subroutine that writes an integer buffer to a file opened by OPENF or CREAF. WRITF does not write individual lines, but a specified number of bytes. Refer to the Write function (40H) in the DOS Technical Reference Manual for more information (page 6-139).

Calling Sequence:

CALL WRITF (IHANDL, IBUFF, NBYTES, ISTAT, NTRANS)

Declarations:

INTEGER*2 IHANDL, IBUFF, NBYTES, ISTAT, NTRANS

Argument Description:

IHANDL	Input	The file handle from OPENF or CREAF.
IBUFF	Input	The integer buffer to be written.
NBYTES	Input	The number of bytes to write.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.
NTRANS	Output	The number of bytes actually written.

2.13.12 SEEKF – Move the File Pointer

Purpose:

SEEKF is a low-level subroutine that moves the file pointer to a specified location for files opened by OPENF or CREAF. Refer to the LSEEK function (42H) in the DOS Technical Reference Manual for more information (page 6-143).

Calling Sequence:

CALL SEEKF (IHANDL, IMODE, IOFSET, IPOS, ISTAT)

Declarations:

INTEGER*2 IHANDL, IMODE, ISTAT
INTEGER*4 IOFSET, IPOS

Argument Description:

IHANDL	Input	The file handle from OPENF or CREAF.
IMODE	Input	The mode of the offset. IMODE has three possible values: 0 The offset is from the beginning of the file. 1 The offset is from the current location. 2 The offset is from the end of the file.
IOFSET	Input	The number of bytes to move. **This must be an INTEGER*4 number.**
IPOS	Output	The resulting file byte position after the move. **This must be an INTEGER*4 variable.**
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.13 ERASF – Erase a File

Purpose:

ERASF erases the specified file(s). The name specified may contain wild characters to erase all files that match the parts specified. Refer to the Delete function (13H) in the DOS Technical Reference Manual for more information (page 6-74).

Calling Sequence:

```
CALL ERASF (CNAME, ISTAT)
```

Declarations:

```
CHARACTER CNAME
INTEGER*2 ISTAT
```

Argument Description:

CNAME	Input	The name of the file to delete (or name with wild characters). This name must be terminated by a zero value byte (e.g., CNAME//CHAR(0)).
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.14 RNAMF – Rename a File

Purpose:

RNAMF renames a file(s). The file names may contain the wild characters used by DOS to rename several files with matching parts. Refer to the Rename function (17H) in the DOS Technical Reference Manual for more information (page 6-79).

Calling Sequence:

CALL RNAMF (COLDN, CNEWN, ISTAT)

Declarations:

```
CHARACTER COLDN, CNEWN
INTEGER*2 ISTAT
```

Argument Description:

COLDN	Input	The current name of the file (or name with wild characters). This name must be terminated by a zero value byte (e.g., COLDN//CHAR(0)).
CNEWN	Input	The new name to give the file (or name with wild characters). This name must be terminated by a zero value byte (e.g., CNEWN//CHAR(0)).
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

2.13.15 CHMOD – Change a File Mode

Purpose:

CHMOD is a low-level subroutine that changes a files mode. Refer to the CHMOD function (43H) in the DOS Technical Reference Manual for more information (page 6-145).

Calling Sequence:

CALL CHMOD (CNAME, IFATT, IFUN, ISTAT)

Declarations:

```
CHARACTER CNAME
INTEGER*2 IFATT, IFUN, ISTAT
```

Argument Description:

CNAME	Input	The name of the file whose mode is to be changed. This file name must be terminated by a zero value byte (e.g., CNAME//CHAR(0)).
IFATT	Input/ Output	The file attributes, as described on page 5-11 of the DOS Technical Reference Manual.
IFUN	Input	If IFUN is set to zero, the file's attribute is returned in IFATT. If IFUN is set to one, the file attributes will be set according to IFATT.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

3 Terminal Input/Output and Control Subroutines

The following chapter describes the subroutines that are designed for terminal input-output and control of the terminal screen. This includes reading and writing single characters (as opposed to complete lines as required by FORTRAN), and full screen control of the terminal. These items are available for both HARRIS and MS-DOS computers.

On HARRIS computers, terminal I/O is accomplished with the CHRIO (Hot-Read) subroutines. Screen control is provided for terminals that meet ANSI standards with subroutine ASCTRL. At this time there are no provisions for non-ANSI terminals.

On MS-DOS microcomputers, terminal I/O is accomplished with the subroutines STDIN and STDOUT. (The CHRIO subroutines have been implemented for the microcomputer using calls to these routines.) The screen is controlled with either the Video routines (those that begin with the letter "V"), or the PUF routines. The video subroutines perform functions such as scrolling the screen, clearing the screen, positioning the cursor, etc. The PUF subroutines provide a means of creating windows and changing colors or attributes of specific portions of the screen.

3.1 ANREAD – Perform a Prompted Read

Purpose:

ANREAD writes a prompt to the terminal screen, then reads from the terminal without an intervening carriage return and line feed. ANREAD will read from the different port types or from a file input.

Calling Sequence:

CALL ANREAD (IUNIT, CPROMPT, NPROMPT, CLINE, NLINE)

Declarations:

```
CHARACTER CPROMPT, CLINE
INTEGER IUNIT, NPROMPT, NLINE
```

Argument Description:

IUNIT	Input	The unit number to prompt and read from. This unit may be connected to either a file or a terminal (or console).
CPROMPT	Input	A character string containing the prompt to write out.
NPROMPT	Input	The number of characters in CPROMPT to write.
CLINE	Output	CLINE will contain the line read. The length of CLINE is implicit (e.g., CLINE(1:60)).
NLINE	Output	The number of characters read (in CLINE). If ANREAD detected an end-of-file condition, NLINE is returned as -1. If the declared length of CLINE is less then the length of the expanded line, NLINE is returned as -2 (and the line is truncated).

Remarks:

If an escape character is pressed, or the user backspaces into the prompt, ANREAD will print an exclamation mark (!), then write a new prompt. Currently, ANREAD does not correctly read from redirected input on the microcomputer.

Example:

```
CALL ANREAD (5, 'Do you want to continue? ', 25, CLINE, NLINE)
IF (CLINE(1:1).EQ.'Y') THEN ...
```

3.2 RBELL – Ring the Terminal Bell

Purpose:

RBELL rings the terminal bell. No action is taken if the program is running in a batch mode.

Calling Sequence:

CALL RBELL

3.3 HARRIS Specific Subroutines

3.3.1 Character (Hot Read) I/O Subroutines

Purpose:

This describes the CHRIO set of subroutines that perform Hot Read Input/Output functions with Character type data. These subroutines utilize HARRIS assembly code for specialized terminal I/O. Never call any of these subroutines when I-O is being performed on a file, as an error will occur (with a message of INVALID FUNCTION CODE). Refer to the HARRIS Asynchronous Device Handler or I/O System Services Reference Manual for more information on Hot Read I/O.

Before performing any reads with Hot Read, the Hot Mode must be initialized with a call to CHRIT1. This initialization sets up the I-O buffer and sets the port in Hot Mode. When in Hot Mode, only Hot Reads may occur; no FORTRAN reads to the terminal can take place during this time. When complete, the Hot Mode is terminated by a call to CHRFN1. (After this call, FORTRAN reads may be used.)

The CHRIO subroutines are divided into two sets for I/O on two different ports (simultaneously). Usually, terminal I/O will be conducted at one port (unit) only, so the CHRIO subroutines utilizing channel 1 will normally be used. The following subroutines end with the number 1, indicating they are for I/O on channel 1. The same set of subroutines can be used for another unit by replacing the number 1 with the number 2 (for channel 2).

Hot Writes to a terminal may occur at any time. The Hot Mode does not need to be initialized. FORTRAN writes can also occur at any time on ASYNC ports; regardless if the port is in Hot Mode (this was not true with previous implementations of Hot Read).

Subroutine Summary:

CHRIT1 - Initialize character I/O
CHRFN1 - Finish character I/O
CHRWT1 - Write character(s)
CHRRD1 - Read character(s), waiting for at least one character
CHRRI1 - Read character(s) without waiting for a character to arrive
CHRBK1 - Backstore characters
CHRFL1 - Flush characters in type-ahead buffer
CHRWI1 - Write without waiting for completion
CHRST1 - Request status on last operation (waiting for completion)
CHRSI1 - Request status on last operation (without waiting for completion)

3.3.1.1 CHRWT – Write Individual Character(s) to a Terminal

Purpose:

CHRWT writes individual characters to a terminal, or similar device. CHRWT writes out exactly what is specified: Implicit line feeds and carriage returns are not written at the end of the character sequence. No initialization subroutine needs to be called.

Calling Sequence:

CALL CHRWT (IUNIT, CSTR, NSTR)

Declarations:

```
CHARACTER CSTR
INTEGER NSTR, IUNIT
```

Argument Description:

IUNIT	Input	The unit to write the character to. This unit must be attached to a terminal (or physical device), not a file.
CSTR	Input	The character string to be written. No implicit carriage returns or line feeds will be added to the string (they must be explicitly written).
NSTR	Input	The number of characters (in CSTR) to write.

3.3.1.2 CHRIT1 – Initialize Characters I/O

Purpose:

CHRIT1 initializes a unit for reading characters from a terminal. CHRIT1 must be called prior to any character reads (but is not necessary for character writes). Immediately after this call, any characters entered at the terminal will be stored in a type-ahead buffer (until a character read routine is called). The character I/O mode will remain effective until CHRFN1 is called.

Calling Sequence:

CALL CHRIT1 (IUNIT, IBUFF, NUBFF)

Declarations:

INTEGER IUNIT, IBUFF(NBUFF)

Argument Description:

IUNIT	Input	The unit (channel #1) to perform the character I/O on. This unit must be attached to a terminal (or the physical device), not a file.
IBUFF	Input	An integer array where the characters read will be temporarily stored. The dimension of IBUFF determines the size of the type-ahead buffer. The maximum type-ahead buffer size is 86 words (for 256 characters).
NBUFF	Input	The dimension of IBUFF (in integer words).

Remarks:

To initiate the character I/O mode for a second terminal, call subroutine CHRIT2, with identical arguments.

3.3.1.3 CHRFN1 – Finish Characters I/O

Purpose:

CHRFN1 terminates the character I/O mode for channel #1, after a CHRIT1 call has been made. This call should be made prior to exiting a program, and must be made before a FORTRAN read may be accomplished on that unit.

Calling Sequence:

```
CALL CHRFN1
```

3.3.1.4 CHRWT1 – Write Character(s)

Purpose:

CHRWT1 writes characters to the unit specified in the CHRIT1 call. CHRWT1 writes out exactly what is specified: Implicit line feeds and carriage returns are not written at the end of the character sequence. Characters may be written to a terminal, without calling CHRIT1, by calling subroutine CHRWT instead.

Calling Sequence:

CALL CHRWT1 (CSTR, NSTR)

Declarations:

```
CHARACTER CSTR
INTEGER NSTR
```

Argument Description:

CSTR	Input	The character string to be written. No implicit carriage returns or line feeds will be added to the string (they must be explicitly written).
NSTR	Input	The number of characters (in CSTR) to write.

3.3.1.5 CHRRD1 – Read Character(s), Waiting for at Least One Character

Purpose:

CHRRD1 reads characters from channel 1. If no characters are in the type ahead buffer, CHRRD1 will wait until at least one character arrives. CHRIT1 must have been called prior to CHRRD1.

CHRRD1 removes the parity (8th) bit from all characters read.

Calling Sequence:

```
CALL CHRRD1 (CSTR, NSTR)
```

Declarations:

```
CHARACTER CSTR
INTEGER NSTR
```

Argument Description:

CSTR	Output	A character variable that will contain the characters read. This variable should an equivalent size as the IBUFF array passed to CHRIT1. For example, if IBUFF is dimensioned to 86, CSTR should be 258 characters long (although only one character might be returned).
NSTR	Output	The number of characters read and contained in CSTR. This will not be larger than that provided for by IBUFF in CHRIT1 (those entered beyond that limit will be lost).

3.3.1.6 CHRRI1 – Read Characters Without Waiting for a Character to Arrive

Purpose:

CHRRI1 reads characters from channel 1 similarly to CHRRD1, except CHRRI1 does not wait for any characters to arrive: CHRRI1 returns immediately, regardless if any characters have been read or not. CHRRI1 is typically used when another operation is occurring simultaneously.

CHRRI1 removes the parity (eighth) bit from all characters read.

Calling Sequence:

CALL CHRRI1 (CSTR, NSTR)

Declarations:

```
CHARACTER CSTR
INTEGER NSTR
```

Argument Description:

CSTR	Output	A character variable that will contain any characters read. This variable should an equivalent size as the IBUFF array passed to CHRIT1. For example, if IBUFF is dimensioned to 86, CSTR should be 258 characters long (although no characters may be returned).
NSTR	Output	The number of characters read and contained in CSTR. This will not be larger than that provided for by IBUFF in CHRIT1 (those entered beyond that limit will be lost), and may be zero.

3.3.1.7 CHRBK1 – Backstore Characters

Purpose:

CHRBK1 backstores the specified character string, returning it to the type-ahead buffer.

Calling Sequence:

```
CALL CHRBK1 (CSTR, NSTR)
```

Declarations:

```
CHARACTER CSTR
INTEGER NSTR
```

Argument Description:

CSTR	Input	The character string to backstore.
NSTR	Input	The number of characters (in CSTR) to backstore.

3.3.1.8 CHRFL1 – Flush Characters in Type-Ahead Buffer

Purpose:

Flushes (removes) all characters in the type-ahead buffer.

Calling Sequence:

```
CALL CHRFL1
```

3.3.1.9 CHRWI1 – Write Without Waiting for Completion

Purpose:

CHRWI1 writes characters to channel 1, but does not wait for the operation to complete. This subroutine is usually used when two simultaneous operations are occurring, and time is of the essence. CHRSI1 may be called to determine the status of the write.

Calling Sequence:

```
CALL CHRWI1 (CSTR, NSTR)
```

Declarations:

```
CHARACTER CSTR
INTEGER NSTR
```

Argument Description:

CSTR	Input	The handle number given to the file. This is similar to a FORTRAN unit number, and must be used for all DKBF calls for that file. (Use a different handle variable for a different file).
NSTR	Input	The name of the file to perform I/O on.

3.3.1.10 CHRST1 - Request Status on Last Operation

Purpose:

CHRST1 returns the status of the most recent operation. CHRST1 will wait until the operation has completed, or an error occurs. If it is necessary to check the status without being placed in a wait mode, call CHRSI1 instead of CHRST1.

Calling Sequence:

```
CALL CHRST1 (ISTAT, JSTAT)
```

Declarations:

```
INTEGER ISTAT, JSTAT
```

Argument Description:

ISTAT	Output	Returned as zero if the operation was completed without any errors. If an error occurred, ISTAT is returned as one.
JSTAT	Output	The status returned in a coded form, with bits set indicating information about the operation. Call subroutine CSTAT to decode this status. A description of this word may be found in the I-O Services Reference Manual for function code '00.

3.3.1.11 CHRSI1 – Request Status on Last Operation, Without Wait

Purpose:

CHRSI1 returns the status of the most recent operation, without waiting for the operation to complete. This is normally called to check the status of a call to CHRWI1.

Calling Sequence:

CALL CHRSI1 (ISTAT, JSTAT

Declarations:

INTEGER ISTAT, JSTAT

Argument Description:

ISTAT	Output	Returned as zero if the operation was completed without any errors. If the operation is still in progress, ISTAT is returned set to -1. If an error occurred, ISTAT is returned as one.
JSTAT	Output	The status returned in a coded form, with bits set indicating information about the operation. Call subroutine CSTAT to decode this status. A description of this word may be found in the I-O Services Reference Manual for function code '00.

3.3.1.12 CHRIO Examples

```
      SUBROUTINE PROMRD (CPROM, CREAD)
C
C     Perform a simple prompted read
C     (Note:  This is an incomplete example.  See the
C     source code to ANREAD for a complete example).
C
      CHARACTER CPROM*(*), CREAD*(*), CSTR*256
      INTEGER IBUFF(86)
C
C     Initialize Character Read (on unit 0)
      CALL CHRIT1 (0, IBUFF, 86)
C
C     Write Prompt, adding a line feed at the beginning
      NSTR = LEN(CPROM)
      CALL CHRWT1 (CHAR(10)//CPROM, NSTR+1)
C     Blank fill CREAD
      CALL CHRBLK (CREAD)
C
C     Read characters until a carriage return is found.
      IMAX = LEN(CREAD)
      NLEN = 0
   20 CONTINUE
      CALL CHRRD1 (CSTR, NSTR)
C
C     Process characters
      DO 40 I=1,NSTR
C
C     Check for backspace
      IF (CSTR(I:I).EQ.CHAR(8)) THEN
      ...
C
C     Echo character
      CALL CHRWT1 (CSTR(I:I), 1)
C
C     Check for carriage return
      IF (CSTR(I:I).EQ.CHAR(13)) THEN
      CALL CHRFN1
C     If more characters remaining, backstore them
      IF (I.NE.NSTR) CALL CHRBK1 (CSTR(I+1:NSTR), NSTR-I)
      RETURN
      ENDIF
C
C     Check for max length.  If OK, place character in CREAD
      NLEN = NLEN + 1
      IF (NLEN.GT.IMAX) GO TO 40
```

```
      CREAD(NLEN:NLEN) = CSTR(I:I)
C
  40  CONTINUE
C
C     Go back and read more
      GO TO 20
      END

C     This example illustrates direct 2 way communication between
C     two terminals.  (This is a complete operational program).
C
      INTEGER IBUFF1(86), IBUFF2(86)
      CHARACTER CSTR1*256, CSTR2*256
C
C     Resource the other terminal
C     CALL RSCPDN (...
C
C     OPEN and initialize the terminals
      OPEN (UNIT=8)
      CALL CHRIT1 (0, IBUFF1, 86)
      CALL CHRIT2 (8, IBUFF2, 86)
C
C     Now loop, reading and writing to each terminal
      LOOP
C
C     Don't wait for a character - Do an immediate read
      CALL CHRRI1 (CSTR1, NSTR1)
      CALL CHRRI2 (CSTR2, NSTR2)
C
      IF (NSTR1.GT.0) THEN
C     Exit if a control-A was entered
      EXIT LOOP IF (INDEX(CSTR1(1:NSTR1),CHAR(1)).GT.0)
C     Send the character(s) to the other terminal
      CALL CHRWT2 (CSTR1, NSTR1)
C     Echo the character(s) on this terminal
      CALL CHRWT1 (CSTR1, NSTR1)
      ENDIF
C
      IF (NSTR2.GT.0) THEN
C     Send the character(s) to the other terminal
      CALL CHRWT1 (CSTR2, NSTR2)
C     Echo the character(s) on this terminal
      CALL CHRWT2 (CSTR2, NSTR2)
      ENDIF
C
C     If no characters transferred, wait for a short amount of
C     time, so we don't burn CPU.
```

```
      IF ((NSTR1.EQ.0).AND.(NSTR2.EQ.0)) CALL WAITS (0.1)
C
      ENDLOOP
C
C     Terminate Character I-O
      CALL CHRFN1
      CALL CHRFN2
      CLOSE (UNIT=8)
C
      STOP
      END
```

3.3.2 TRMTYP – Determine the Terminal Port Type

Purpose:

TRMTYP returns the type of device handler being used. There are three types: Asynchronous, CRT, and TTY.

Calling Sequence:

CALL TRMTYP (IUNIT, CTYPE)

Declarations:

CHARACTER CTYPE*3
INTEGER IUNIT

Argument Description:

IUNIT	Input	The unit number connected to the port for which the type is desired. This typically is unit 0, but may be a resourced physical device (this is the unit number, not the PDN).
CTYPE	Output	(Output) A three character variable containing the port type. Four responses are possible: 'ASY' - Async Handler 'TTY' - TTY Handler 'CRT' - CRT Handler 'UNK' - Unknown or error

Remarks:

Call TRMTYP only for units connected to a physical device, not a file.

3.3.3 CLINES – Get the Number of Lines of a Terminal Screen

Purpose:

CLINES returns the number of lines the terminal screen is designated to hold. On HARRIS computers, CLINES obtains this information from the system gen file. On the MS-DOS microcomputer, the number of lines is set to twenty-five.

Calling Sequence:

CALL CLINES (NLINES)

Declarations:

INTEGER NLINES

Argument Description:

NLINES	Output	The number of lines the terminal can hold.

Remarks:

CLINES is useful when displaying several screens of information on the terminal.

Example:

```
      CALL CLINES(NLINES)
   10 DO 40 I=1,NLINES-1
      READ (9, 20, END=100) CLINE
   20 FORMAT (A)
      WRITE (6, 25) CLINE
   25 FORMAT (1X,A)
   40 CONTINUE
C
      CALL ANREAD ( 5,' -- Press Carriage Return to Continue -- ', 41,
   *  CLINE, NLINE)
      GO TO 10
```

3.3.4 CKANSI – Check if Terminal is ANSI

Purpose:

CKANSI is used to determine if the terminal being accessed meets the ANSI standards for terminals (or is in ANSI mode). If it is, ANSI commands may be sent to control the terminal (e.g., clear the screen, move the cursor, etc.).

Calling Sequence:

CALL CKANSI (IUNIT, LANSI)

Declarations:

```
INTEGER IUNIT
LOGICAL LANSI
```

Argument Description:

IUNIT	Input	The unit number of the terminal to check. The unit must have been opened (if other than zero or three).
LANSI	Output	A logical flag set to .TRUE. if the terminal responds to ANSI commands.

Remarks:

CKANSI sends a request for a cursor position report. If a valid response is received within three seconds, LASNI is returned true. If it is not an ANSI terminal, three characters may appear on the screen.

CKANSI will flush any type-ahead buffer. Subroutine CHRRD1 is used to read the report from the terminal. Therefore CHRIT1 should not be called before CKANIS, unless CHRFN1 is called to terminate the hot-read state (it may be re-initiated after CKANSI).

3.3.5 ASCTRL – ANSI Screen Control

Purpose:

ASCTRL controls certain terminal functions for ANSI terminals. This includes clearing the screen, moving the cursor, highlighting characters, etc.. The terminal accessed must be an ANSI terminal (or in ANSI mode). CKANSI should be called prior to ASCTRL to be certain that the terminal is ANSI.

Calling Sequence:

CALL ASCTRL (IUNIT, CFUN, IARG1, IARG2)

Declarations:

```
CHARACTER CFUN*2
INTEGER IUNIT, IARG1, IARG2
```

Argument Description:

IUNIT	Input	The unit number connected to the terminal to access (usually 3). This unit must have been opened.
CFUN	Input	A two character description of the function to be performed. The functions are listed in the table on the next page.
IARG1	Input	An integer argument used for those functions that provide for a variable number of occurrences. For example, if five characters are to be deleted, IARG1 should be five. For a single occurrence, IARG1 may be set to zero.
IARG2	Input	A second argument used only for positioning the cursor.

Functions:

CFUN	IARG1	IARG2	Description
'MC'	Row	Col	Move the cursor to row IARG1, column IARG2.
'CR'	Nchs	-	Move the cursor IARG1 spaces right.
'CL'	Nchs	-	Move the cursor IARG1 spaces left.
'CU'	Nlines	-	Move the cursor IARG1 lines up.
'CD'	Nlines	-	Move the cursor IARG1 lines down.
'DC'	Nchs	-	Delete IARG1 characters right from the cursor.
'DL'	Nlines	-	Delete IARG1 lines down from the current line.
'IL'	Nlines	-	Insert IARG1 lines below the current line.
'IC'	-	-	Go into Insert Character Mode.
'TO'	-	-	Go into Type-Over Mode.
'CS'	-	-	Clear Screen.

'EL'	-	-	Erase Line (fill with blanks).
'KA'	-	-	Put Keypad in Application Mode.
'KN'	-	-	Put Keypad in Numeric Mode.
'BO'	-	-	Bold characters.
'BL'	-	-	Blinking characters.
'UL'	-	-	Underline characters.
'RV'	-	-	Reverse Video characters.
'NO'	-	-	Normal character attributes.

Remarks:

The character attributes are additive; to change from one attribute to another, reset the attributes to normal then set the new attribute (otherwise both attributes will be set). These attributes only affect characters that are printed after the attribute has been issued.

On ANSI terminals, the upper left hand corner is referred to as row 1, column 1. On most terminals there are 25 rows and 80 columns. To go to the home position, use move cursor ('MC') to row 1, column 1.

Be sure that the terminal is an ANSI terminal (or in ANSI mode). Subroutine CKANSI may be called to check this.

Example:

```
      CALL CLINES(NLINES)
C     Form fill-in program
C
      CHARACTER CGAGE*10, CPRE*10, CSTAG*10
      LOGICAL LANSI, LVALID
C
C     Check that this is an ANSI terminal
      CALL CKANSI (3, LANSI)
      IF (.NOT.LANSI) GO TO 900
C
C     Clear the screen
      CALL ASCTRL (3, 'CS', 0, 0)
C     Write out the form (A short form is given here)
C     Position Cursor and write info.
      CALL ASCTRL (3, 'MC', 5, 14)
      CALL CHRWT (3, 'Gage Name:', 10)
C
      CALL ASCTRL (3, 'MC', 8, 10)
      CALL CHRWT (3, 'Precipitation:', 14)
      CALL ASCTRL (3, 'MC', 9, 18)
      CALL CHRWT (3, 'Stage:', 6)
C
C     Now read in info.  Use ANREAD, but move it a line above
```

```
C       position to read from because it issues a line feed at beg.
  20    CONTINUE
        CALL ASCTRL (3, 'MC', 4, 25)
        CALL ANREAD (3, ' ', 0, CGAGE, NGAGE)
C
C       Make sure that this is a valid gage
  30    CALL CKGAGE (LVALID, CGAGE, NGAGE)
C       If not a valid file, print an error message, bold and underline
        IF (.NOT.LVALID) THEN
        CALL ASCTRL (3, 'MC', 20, 20)
        CALL ASCTRL (3, 'BO', 0, 0)
        CALL ASCTRL (3, 'UL', 0, 0)
        CALL CHRWT (3, 'Unrecognized gage!', 18)
C       Re-read gage name (with normal attributes)
        CALL ASCTRL (3, 'NO', 0, 0)
        CALL ASCTRL (3, 'MC', 4, 25)
        CALL ANREAD (3, ' ', 0, CGAGE, NGAGE)
C       New name - erase error message
        CALL ASCTRL (3, 'MC', 20, 0)
        CALL ASCTRL (3, 'EL', 0, 0)
        GO TO 30
        ENDIF
C
C       Read other parameters
        CALL ASCTRL (3, 'MC', 7, 25)
        CALL ANREAD (3, ' ', 0, CPRE, NPRE)
        CALL ASCTRL (3, 'MC', 8, 25)
        CALL ANREAD (3, ' ', 0, CSTAG, NSTAG)
C
C       Process the data
        CALL PRDATA (CPRE, NPRE, CSTAG, NSTAG)
C
C       Erase the old information
        CALL ASCTRL (3, 'MC', 5, 25)
        CALL ASCTRL (3, 'DC', NGAGE, 0)
        CALL ASCTRL (3, 'MC', 8, 25)
        CALL ASCTRL (3, 'DC', NPRE, 0)
        CALL ASCTRL (3, 'MC', 9, 25)
        CALL ASCTRL (3, 'DC', NSTAG, 0)
C
C       Go back and read more data
        GO TO 20
```

3.3.6 STTY – Set Terminal Port Parameters for an ASYNC Port

Purpose:

Subroutine STTY provides a means of getting or altering port parameters on an ASYNC port, similar to program STTY. A complete list of the parameters that may be obtained or changed is provided in the Device Configuration Block listing of Table 3.1 in the HARRIS Asynchronous Device Handler Manual. STTY must only be called for a unit connected to an ASYNC port. The port type can be determined by subroutine TRMTYP.

Calling Sequence:

```
CALL STTY (IUNIT, CDIR, CITEM, CSTR, ISTAT)
```

Declarations:

```
CHARACTER CDIR*3, CITEM, CSTR
INTEGER IUNIT, ISTAT
```

Argument Description:

IUNIT	Input	The unit number of the port to set. This may be either the terminal the program is running at, or a resourced PDN. This unit should have been already opened.
CDIR	Input	The direction. Must either be 'SET' or 'GET'.
CITEM	Input	The item to set or get. This item must be one of the keywords specified in Table 3.1 of the ASYNC manual, or 'BAUD' to set the baud rate.
CSTR	Input/ Output	What that item is set to (or what to set it to). For items that indicate the setting of a single bit in table 3.1 (e.g., IXON), CSTR is either 'ON' or 'OFF'. For items that occupy one byte, CSTR is set or returned as a single character. For example, to set the abort character to control-B, CSTR would be set equal to CHAR(2). If the prompt is to be set or retrieved, CSTR will contain the character prompt, up to nine characters long. If the baud rate is to be set or retrieved, CSTR will contain the baud rate (for example, '2400'). CSTR should contain a number for keywords PADHI, PADMD, PADLO and COL. For example, if getting the number of columns for a terminal, do an internal read after calling STTY (e.g., READ(CSTR,'(I3)') NCOL).
ISTAT	Output	A status parameter, set to one of the following:

IOSTAT	Description
0	Call completed successfully
1	No Device Configuration Block available
2	Item not in list
3	CDIR not 'SET' or 'GET'
4	CSTR not 'ON' or 'OFF' for single bit items
5	Read only parameter - cannot set
6	Invalid parameter for CSTR

Remarks:

The HARRIS Asynchronous Device Handler Reference Manual should be referenced to when using this subroutine. The item must appear exactly as shown in Table 3.1 (e.g., 'rCTS').

When the unit is closed (or the user signs off), the device settings are reset to their default values. Thus, it would not be useful to reset the logon character via STTY.

When the prompt string is requested, it is returned in the variable CSTR, null filled.

Example 1:

```
C       Temporarily reset the prompt to 'Input>'
        CHARACTER CPROMP*9
C
C       First, get the current prompt
        CALL STTY (0, 'GET', 'PROMPT', CPROMP, ISTAT)
        IF (ISTAT.NE.0) GO TO 900
C
C       Now reset it
        CALL STTY (0, 'SET', 'PROMPT', 'Input>', ISTAT)
C
C       Do any input and output . . .
C
C       Finished, set it back
        CALL STTY (0, 'SET', 'PROMPT', CPROMP, ISTAT)
```

Example 2:

```
C       Set the baud rate of a unit resourced to a modem port
C
C       Resource the port to unit 9
        CALL RSCPDN ( ...
        OPEN (UNIT=9)
C
        CALL STTY (9, 'SET', 'BAUD', '2400', ISTAT)
```

Example 3:

```
C     Determine the delete to end of line character, and
C     change the delete word character to control-D.
      CHARACTER CDEND*1
C
C     Get Delete to end of line char
      CALL STTY (0, 'GET', 'DEND', CDEND, ISTAT)
      IDEND = ICHAR(CDEND)
C
C     Set delete word character to control-D (ASCII 4)
      CALL STTY (0, 'SET', 'DWORD', CHAR(4), ISTAT
```

3.3.7 BRKOFF – Turn the Break Key Off

Purpose:

BRKOFF disables the terminal break key. The break key will be disabled until it is turned back on using subroutine BRKON, or the session is ended.

Calling Sequence:

CALL BRKOFF

Remarks:

BRKOFF will work on all terminal types (i.e., ASYNC, CRT, TTY). A program may be aborted from the OPCOM if the break key has been disabled.

3.3.8 BRKON – Turn the Break Key On

Purpose:

BRKON re-enables the break key after it has been disabled by subroutine BRKOFF.

Calling Sequence:

```
CALL BRKON
```

3.4 MS-DOS Specific Subroutines

3.4.1 STDINC – Read a Character from the Keyboard (Standard In)

Purpose:

STDINC reads a character from the keyboard (or standard input) under strict control of the program.

Calling Sequence:

CALL STDINC (CWAIT, CECHO, CBREAK, CFLUSH, IASCII, ICODE)

Declarations:

```
CHARACTER CWAIT*1, CECHO*1, CBREAK*1, CFLUSH*1
INTEGER*2 IASCII, ICODE
```

Argument Description:

CWAIT	Input	If CWAIT is 'Y', then STDINC will wait for a key to be pressed. If CWAIT is 'N', then STDINC will return immediately, returning a character from the type-ahead buffer or with no character and IASCII set to -1.
CECHO	Input	If CECHO is 'Y', then STDINC will echo the character on the screen. If CWAIT is 'N', no echo will occur.
CBREAK	Input	If CBREAK is 'Y', then STDINC will check if the break key has been pressed (and abort the program). If CWAIT is 'N', the break will not be checked.
CFLUSH	Output	If CFLUSH is 'Y', then STDINC will flush any characters in the type-ahead buffer. If CWAIT is 'N', the type-ahead buffer will not be flushed.
IASCII	Output	The ASCII decimal equivalent value of the character, if the key pressed was a normal ASCII key. If an extended key was pressed (e.g., function keys), IASCII will be set to zero, and the extended code will be returned in ICODE.
ICODE	Output	The extended key code, if a non-ASCII character key was pressed. The extended key codes may be found in the IBM Technical Reference Manual (under System BIOS).

Remarks:

Not all of the above options are independent: Only certain combinations work for CWAIT, CECHO and CBREAK. They are:

CWAIT	CECHO	CBREAK
'Y'	'Y'	'Y'
'Y'	'N'	'Y'
'Y'	'N'	'N'
'N'	'N'	'N'

3.4.2 STDOUT – Write a Single Character to the Monitor (Standard Out)

Purpose:

STDOUT writes a single character to the monitor (or standard output). The character must be given in its ASCII decimal equivalent value.

Calling Sequence:

CALL STDOUT (CBREAK, IASCII)

Declarations:

```
CHARACTER CBREAK*1
INTEGER*2 IASCII
```

Argument Description:

CBREAK	Input	If CBREAK is 'Y', then STDOUT will check if the break key has been pressed (and abort the program). If CWAIT is 'N', the break will not be checked.
IASACII	Input	The ASCII decimal equivalent of the character to write. For example, to write a "J", pass ICHAR('J').

3.4.3 TXTCOL – Set the Screen Color for Text

Purpose:

TXTCOL sets the screen color so that future writes will be written with the specified color and attributes. TXTCOL currently requires ANSI.SYS to be installed.

Calling Sequence:

CALL TXTCOL (COLRFG, COLRBG, CATT)

Declarations:

CHARACTER COLRFG, COLRBG, CATT

Argument Description:

COLRFG	Input	The foreground (character) color. This should be one of the colors listed below.
COLRBG	Input	The background color. This should be one of the colors listed below.
CATT	Output	The attribute of the characters (foreground). This should be either a blank (' ') for normal characters, or 'BOLD' to highlight the characters, or 'BLINK' to make the characters blink. BOLD and BLINK may be combined using a dash (-).

Colors:

The recognized colors are:

BLACK
RED
YELLOW
GREEN
BLUE
CYAN
MAGENTA
WHITE

Example:

```
CALL TXTCOL ('YELLOW', 'BLUE', ' ')
CALL TXTCOL ('RED', 'BLACK', 'BOLD-BLINK')
```

3.4.4 VSTAT – Video Status

Purpose:

VSTAT returns the status of the video screen including the mode, active page, and the number of columns on the screen.

Calling Sequence:

CALL VSTAT (IMODE, ICOL, IPAGE)

Declarations:

INTEGER*2 IMODE, ICOL, IPAGE

Argument Description:

IMODE Output The mode the screen is set to. Possible values include:

ValueT	Description
0	40 X 25 Blank and White
1	40 X 25 Color
2	80 X 25 Blank and White
3	80 X 25 Color
4	320 X 200 Color Graphics
5	320 X 200 Black and White Graphics
6	640 X 200 Black and White Graphics
10	640 X 200 4 Color EGA Graphics
13	320 X 200 16 Color EGA Graphics
14	640 X 200 16 Color EGA Graphics
16	640 X 350 4 or 16 Color EGA Graphics

ICOL Output The number of columns allocated for the screen.

IPAGE Output The number of the current page.

3.4.5 VNEWPG – Clear Screen

Purpose:

VNEWPG clears the screen, moves the cursor to the home position and sets the screen to the specified attribute (color).

Calling Sequence:

CALL VNEWPG (IATT)

Declarations:

INTEGER*2 IATT

Argument Description:

IATT	Input	The attribute (color) to set the screen to.

Remarks:

The attribute is a combination of numbers defining the color and intensity of the foreground and background. To obtain an attribute, add a number from each of the following colors together:

Color	Foreground	Background
Black	0	0
Blue	1	16
Green	2	32
Light Blue	3	48
Red	4	64
Violet	5	80
Orange	6	96
White	7	112

To intensify the foreground color, add eight (8). To cause the foreground to blink, add 128. For example:

IATT = 7 + 0 = 7	gives white characters, black background
IATT = 7 + 16 = 23	gives white characters, blue background
IATT = 6 + 8 + 16 = 30	gives yellow (bright orange) characters, blue background
IATT = 3 + 64 + 128 = 195	gives blinking light blue characters on a red background.

3.4.6 VSCROL – Scroll Screen Window

Purpose:

VSCROL scrolls a window on the screen. This may include scrolling the entire screen up or down (leaving blank line(s) at the bottom or top), or scrolling a window of the screen. VSCROL may be used to clear the screen.

Calling Sequence:

CALL VSCROL (CDIR, NLINES, IUROW, IUCOL, ILROW, ILCOL, IATT)

Declarations:

CHARACTER CDIR*1
INTEGER*2 NLINES, IUROW, IUCOL, ILROW, ILCOL, IATT

Argument Description:

CDIR	Input	The direction to scroll. This may be either a 'U' to scroll the screen up (and place blank lines on the bottom), or a 'D' to scroll down (and place blank lines on the top).
NLINES	Input	The number of lines to scroll. To blank the entire screen, set NLINES equal to zero.
IUROW	Input	The upper row number defining the window to scroll. If the entire screen is to be scrolled, set IUROW to zero.
IUCOL	Input	The upper column number defining the window to scroll. If the entire screen is to be scrolled, set IUCOL to zero.
ILROW	Input	The lower row number defining the window to scroll. If the entire screen is to be scrolled, set ILROW to twenty-four.
ILCOL	Input	The lower column number defining the window to scroll. If the entire screen is to be scrolled, ILCOL should be seventy-nine (for eighty column screens).
IATT	Input	The attribute for the blank lines added to the screen. Refer to the description of attributes in the VNEWPG subroutine description.

Example:

To clear the screen:

```
CALL VSCROL ('U', 0, 0, 0, 24, 79, IATT)
```

To scroll up one line:

```
CALL VSCROL ('U', 1, 0, 0, 24, 79, IATT)
```

To scroll down five lines of a forty column by ten row window in the middle of the screen:

```
CALL VSCROL ('D', 5, 11, 20, 21, 59, IATT)
```

3.4.7 VTTYWT – Write a Line to the Screen

Purpose:

VTTYWT writes a character string to the screen, emulating a FORTRAN write. If the cursor is at the bottom of the screen, the screen will be scrolled up one line.

Calling Sequence:

CALL VTTYWT (CNEWL, CLINE, NLINE)

Declarations:

```
CHARACTER CNEWL*1, CLINE
INTEGER*2 NLINE
```

Argument Description:

CNEWL	Input	A flag indicating if the line should be written on a new line. If CNEWL is '+', the line will be started at the current cursor position (no line feed). If CNEWL is a blank (' '), the line will be written on a new line.
CLINE	Input	The line to write out.
NLINE	Input	The number of characters in CLINE to write.

3.4.8 VGETCR – Get Cursor Position and Size

Purpose:

VGETCR get the current cursor position and its size for a specified video page.

Calling Sequence:

CALL VGETCR (IPAGE, IROW, ICOL, ITOP, IBOTTM)

Declarations:

INTEGER*2 IPAGE, IROW, ICOL, ITOP, IBOTTM

Argument Description:

IPAGE	Input	The page.
IROW	Output	The current cursor row position.
ICOL	Output	The current cursor column position.
ITOP	Output	The starting scan line (pixel) (top) of the cursor, where zero is the top and seven is the bottom of the cursor block.
IBOTTM	Output	The ending scan line (bottom) of the cursor, where zero is the top and seven is the bottom of the cursor block.

Remarks:

The values returned may depend on the monitor adapter card being used.

The cursor has eight scan lines that may be turned on. (It is always the same size in width.) The top (starting) scan line is defined as line zero, and the bottom line seven. ITOP and IBOTTM will always be between zero and seven. If ITOP is greater than IBOTTM, the cursor will be a two part cursor.

3.4.9 VPOSCR – Position of Cursor

Purpose:

VPOSCR positions the cursor on the page specified.

Calling Sequence:

CALL VPOSCR (IPAGE, IROW, ICOL)

Declarations:

INTEGER*2 IPAGE, IROW, ICOL

Argument Description:

IPAGE	Input	The page to position the cursor on.
IROW	Input	The row to position the cursor on.
ICOL	Input	The column to position the cursor on.

3.4.10 VSETCR – Set the Cursor Size

Purpose:

VSETCR sets the size of the cursor. This size is based upon the starting and ending location of the eight scan lines that make up the cursor block (the width is always the same). The top (starting) scan line is defined as line zero, and the bottom line seven.

Calling Sequence:

```
CALL VSETCR (ITOP, IBOTTM)
```

Declarations:

```
INTEGER*2 ITOP, IBOTTM
```

Argument Description:

ITOP	Input	The starting scan line (top) of the cursor.
IBOTTM	Input	The ending scan line (bottom) of the cursor.

Remarks:

ITOP and IBOTTM must always be between zero and seven. If ITOP is greater than IBOTTOM, a two part cursor will be generated.

Note: For EGA or monochrome mode, the numbers range from zero to thirteen.

3.4.11 VRDAC – Get Character and Attribute at Cursor

Purpose:

VRDAC reads the character and attribute at the current cursor position on the page specified.

Calling Sequence:

CALL VRDAC (IPAGE, ICHAR, IATT)

Declarations:

INTEGER*2 IPAGE, ICHAR, IATT

Argument Description:

IPAGE	Input	The page number to read from.
ICHAR	Output	The ASCII decimal equivalent of the character read.
IATT	Output	The attributes of that position.

3.4.12 VSETPG – Set the Video Page

Purpose:

VSETPG changes the active page number. This will flash the new page on the screen.

Calling Sequence:

```
CALL VSETPG (IPAGE)
```

Declarations:

```
INTEGER*2 IPAGE
```

Argument Description:

IPAGE	Input	The number of the page to change to.

3.4.13 VMODE – Set the Video Mode

Purpose:

VMODE sets the video mode relative to color and screen size.

Calling Sequence:

CALL VMODE (IMODE)

Declarations:

INTEGER*2 IMODE

Argument Description:

IMODE Output The mode to set the screen to. Valid values include:

ValueT	Description
0	40 X 25 Blank and White
1	40 X 25 Color
2	80 X 25 Blank and White
3	80 X 25 Color
4	320 X 200 Color Graphics
5	320 X 200 Black and White Graphics
6	640 X 200 Black and White Graphics
10	640 X 200 4 Color EGA Graphics
13	320 X 200 16 Color EGA Graphics
14	640 X 200 16 Color EGA Graphics
16	640 X 350 4 or 16 Color EGA Graphics

3.4.14 PUF Subroutines

Purpose:

The PUF subroutines provide a means of quickly changing text or attributes (colors) on the microcomputer screen. The PUF Subroutines allows the programmer to cut windows onto the screen, and then restore the original screen when complete.

Subroutine Summary:

PUFA - Set a Single Attribute for a Line
PUFAS - Set Attributes for Characters on a Line
PUFC - Set a Single Character on a Line)
PUFCA - Set a Single Character and Attribute on a Line
PUFCAS - Set a Single Character and an Array of Attributes on a Line
PUFL - Write a Line of Characters
PUFLA - Write a Line of Characters with a Single Attribute
PUFLAS - Write a Line of Characters with Different Attributes
PUFWA - Set a Window to a Single Attribute
PUFWC - Set a Window to a Single Character
PUFWCA - Set a Window to a Single Character and a Single Attribute
PUFBFR - Save a Screen Window
PUFBFW - Restore a Screen Window

Attributes:

A common argument in the PUF subroutines is the attribute (IATT). The attribute controls the foreground color and intensity, and the background color. To obtain an attribute, add a number from each of the following columns:

Color	Foreground	Background
Black	0	0
Blue	1	16
Green	2	32
Light Blue	3	48
Red	4	64
Violet	5	80
Orange	6	96
White	7	112

To intensify the foreground color, add eight (8). To cause the foreground to blink, add 128. For example:

IATT = 7 + 0 = 7	gives white characters, black background
IATT = 7 + 16 = 23	gives white characters, blue background

IATT = 6 + 8 + 16 = 30	gives yellow (bright orange) characters, blue background
IATT = 3 + 64 + 128 = 195	gives blinking light blue characters on a red background.

3.4.14.1 PUFA – Set a Single Attribute for a Line

Purpose:

PUFA will set a single attribute for a specified number of characters on a line. The primary purpose of PUFA is to highlight a line (or a portion of a line) without changing any of the characters on the line.

Calling Sequence:

CALL PUFA (IATT, NCHS, IROW, ICOL)

Declarations:

INTEGER*2 IATT, NCHS, IROW, ICOL

Argument Description:

IATT	Input	The attribute to be set. (See the introduction to this section for information on attributes.)
NCHS	Input	The number of character locations from ICOL (inclusive) to set the attribute.
IROW	Input	The row number of the line to set the attributes (the first line on the screen is row zero).
ICOL	Input	The starting column number at which to set the attributes (the left-most column is column zero).

Example:

The word "ERROR!" appears on the screen at row ten, with the "E" in column forty. Highlight it, so that it is bright blinking red with a blue background.

First determine the attribute:

To scroll up one line:

```
IATT  =  red foreground  +  intensify  +  blinking  +  blue background
IATT  =          4       +      8      +     128    +          16
IATT  =  156
```

PUFA call:

CALL PUFA (156, 6, 10, 40)

3.4.14.2 PUFAS – Set an Array of Attributes for Characters on a Line

Purpose:

PUFAS sets an array of attributes for a specified number of characters on a line. PUFAS allows different attributes to be set for each character on the line (or portion of a line) without changing any of the characters on the line. Use subroutine PUFA if the same attribute is to be set.

Calling Sequence:

CALL PUFAS (IATTS, NCHS, IROW, ICOL)

Declarations:

INTEGER*2 IATTS(NCHS), NCHS, IROW, ICOL

Argument Description:

IATTS	Input	The attributes to be set. This must be an INTEGER*2 array with a one to one correspondence with the characters whose attributes are to be changed.
NCHS	Input	The number of character locations from ICOL (inclusive) to set the attributes.
IROW	Input	The row number of the line to set the attributes (the first line on the screen is row zero).
ICOL	Input	The starting column number at which to change attributes (the left-most column is column zero).

Example:

The words "Enter Location and Value:" appear on the screen at row ten, with the "L" in column twenty. Highlight it, so that "Location " has white characters on a blue background, and "and Value:" has blue characters on a white background (with "Enter " unchanged).

```
      DO 10 I=1,9
   10 IATTS(I) = 23     (white on blue)
C
      DO 20 I=10,19
   20 IATTS(I) = 113    (blue on white)
C
      CALL PUFAS (IATTS, 19, 10, 20)
```

3.4.14.3 PUFC – Set a Single Character on a Line

Purpose:

PUFC will set a single character for a specified number of times on a line. This call is used to blank or set any number of characters of a line to the same value without changing the attributes.

Calling Sequence:

CALL PUFC (CCHAR, NCHS, IROW, ICOL)

Declarations:

CHARACTER CCHAR*1
INTEGER*2 NCHS, IROW, ICOL

Argument Description:

CCHAR	Input	The single character to set the line (or portion of the line) to.
NCHS	Input	The number of character locations from ICOL (inclusive) to set.
IROW	Input	The row number of the line to set (the first line of the screen is row zero).
ICOL	Input	The starting column number (the left-most column is column zero).

Example:

Blank row five, starting in column ten, and ending in column seventy:

```
CALL PUFC ( ' ', 61, 5, 10)
```

3.4.14.4 PUFCA – Set a Single Character and Attribute on a Line

Purpose:

PUFCA will set a single character and a single attribute for a specified number of times on a line. This call is used to set both the character and the attribute for any number of characters on a line to the same value.

Calling Sequence:

CALL PUFCA (CCHAR, IATT, NCHS, IROW, ICOL)

Declarations:

```
CHARACTER CCHAR*1
INTEGER*2 IATT, NCHS, IROW, ICOL
```

Argument Description:

CCHAR	Input	The single character to set the line (or portion of the line) to.
IATT	Input	The attribute of the character to be set.
NCHS	Input	The number of character locations from ICOL (inclusive) to set.
IROW	Input	The row number of the line to set (the first line of the screen is row zero).
ICOL	Input	The starting column number (the left-most column is column zero).

Example:

In constructing a box around a table, set the top of the box with yellow (bright orange) on a blue background (attribute 30). CHAR(196) is a horizontal bar, CHAR(218) is the left-top corner of a box, and CHAR(191) is the right-top corner of a box.

```
CALL PUFCA (CHAR(218), 30, 1, 2, 5)
CALL PUFCA (CHAR(196), 30, 68, 2, 6)
CALL PUFCA (CHAR(191), 30, 1, 2, 74)
```

3.4.14.5 PUFCAS – Set a Single Character and an Array of Attributes

Purpose:

PUFCAS will set a single character and an array of attributes on a line (or portion of a line). This call allows the characters of a line to be set to the same value with each attribute set to a different value.

Calling Sequence:

CALL PUFCAS (CCHAR, IATTS, NCHS, IROW, ICOL)

Declarations:

```
CHARACTER CCHAR*1
INTEGER*2 IATTS(NCHS), NCHS, IROW, ICOL
```

Argument Description:

CCHAR	Input	The single character to set the line (or portion of the line) to.
IATTS	Input	The attributes to be set. This must be an INTEGER*2 array with a one to one correspondence with the character locations to be set.
NCHS	Input	The number of character locations from ICOL (inclusive) to set.
IROW	Input	The row number of the line to set (the first line of the screen is row zero).
ICOL	Input	The starting column number (the left-most column is column zero).

3.4.14.6 PUFL – Write a Line of Characters

Purpose:

PUFL writes a character string to the screen without changing the attributes.

Calling Sequence:

CALL PUFL (CLINE, NLINE, IROW, ICOL)

Declarations:

```
CHARACTER CLINE*NLINE
INTEGER*2 NLINE, IROW, ICOL
```

Argument Description:

CLINE	Input	The character string to write.
NLINE	Input	The number of characters in CLINE to write.
IROW	Input	The row number of the line to write.
ICOL	Input	The starting column number at which to begin the line.

3.4.14.7 PUFLA – Write a Line of Characters with a Single Attribute

Purpose:

PUFLA writes a character string, with a single attribute, to the screen.

Calling Sequence:

CALL PUFLA (CLINE, IATT, NLINE, IROW, ICOL)

Declarations:

```
CHARACTER CLINE*NLINE
INTEGER*2 IATT, NLINE, IROW, ICOL
```

Argument Description:

CLINE	Input	The character string to write.
IATT	Input	The attribute to set the characters to.
NLINE	Input	The number of characters in CLINE to write.
IROW	Input	The row number of the line to write.
ICOL	Input	The starting column number at which to begin the line.

Example:

Write a line to the screen with yellow characters on a blue background:

```
CALL PUFLA ('Enter Location Name:', 30, 20, 2, 0)
```

3.4.14.8 PUFLAS – Write a Line of Characters with Different Attributes

Purpose:

PUFLAS writes a character string to the screen, with each character having a different attribute.

Calling Sequence:

CALL PUFLAS (CLINE, IATTS, NLINE, IROW, ICOL)

Declarations:

```
CHARACTER CLINE*NLINE
INTEGER*2 IATTS(NLINE), NLINE, IROW, ICOL
```

Argument Description:

CLINE	Input	The character string to write.
IATTS	Input	The attributes to be set. This must be an INTEGER*2 array with a one to one correspondence with the characters in CLINE.
NLINE	Input	The number of characters in CLINE to write.
IROW	Input	The row number of the line to write.
ICOL	Input	The starting column number at which to begin the line.

Example:

Write the string "Enter Location and Value:" on the screen at row ten, with the "E" in column 0. Highlight it, so that "Enter Location " has white characters on a blue background, and "and Value:" has blue characters on a white background.

```
      DO 10 I=1,15
   10 IATTS(I) = 23      (white on blue)
C
      DO 20 I=16,25
   20 IATTS(I) = 113     (blue on white)
C
      CALL PUFLAS ('Enter Location and Value:', IATTS, 25, 10, 0)
```

3.4.14.9 PUFWA – Set a Window to a Single Attribute

Purpose:

PUFWA will set all of the attributes of a rectangular window on the screen to the same value. The characters in that window are not changed.

Calling Sequence:

CALL PUFWA (IATT, IROW, ICOL, NCOLS, NROWS)

Declarations:

INTEGER*2 IATT, IROW, ICOL, NCOLS, NROWS

Argument Description:

IATT	Input	The attribute to be set.
IROW	Input	The beginning row number of the window (the first line on the screen is row zero)
ICOL	Input	The starting column number of the window (the left-most column is column zero).
NCOLS	Input	The number of columns in the window.
NROWS	Input	The number of rows in the window.

Example:

Set a rectangular window to a yellow foreground and blue background. The window has five rows by forty columns and begins on row two, column ten:

```
CALL PUFWA (30, 2, 10, 5, 40)
```

3.4.14.10 PUFWC – Set a Window to a Single Character

Purpose:

PUFWC will set all of the characters of a rectangular window on the screen to the same value. The attributes in that window are not changed.

Calling Sequence:

CALL PUFWC (CCHAR, IROW, ICOL, NCOLS, NROWS)

Declarations:

CHARACTER CCHAR*1
INTEGER*2 IROW, ICOL, NCOLS, NROWS

Argument Description:

CCHAR	Input	The character to be set.
IROW	Input	The beginning row number of the window.
ICOL	Input	The starting column number of the window.
NCOLS	Input	The number of columns in the window.
NROWS	Input	The number of rows in the window.

Example:

Blank a rectangular window with five rows by forty columns beginning on row two, column ten:

```
CALL PUFWC (' ', 2, 10, 5, 40)
```

3.4.14.11 PUFWCA – Set a Window to a Single Character and Attribute

Purpose:

PUFWCA will set all of the characters and all of the attributes of a rectangular window on the screen to the same value.

Calling Sequence:

CALL PUFWCA (CCHAR, IATT, IROW, ICOL, NCOLS, NROWS)

Declarations:

```
CHARACTER CCHAR*1
INTEGER*2 IATT, IROW, ICOL, NCOLS, NROWS
```

Argument Description:

CCHAR	Input	The character to be set.
IATT	Input	The attribute to be set.
IROW	Input	The beginning row number of the window.
ICOL	Input	The starting column number of the window.
NCOLS	Input	The number of columns in the window.
NROWS	Input	The number of rows in the window.

Example:

Set a rectangular window of five rows by forty columns to blanks with a yellow foreground and a blue background. The window starts on row two, column ten:

```
CALL PUFWCA (' ', 30, 2, 10, 5, 40)
```

3.4.14.12 PUFBFR – Read a Screen Window From the Display

Purpose:

PUFBFR reads a screen window (or the entire screen), and stores the characters and attributes in an array so the screen can later be restore to its original state. PUFBFW restores the screen from this array.

Calling Sequence:

CALL PUFBFR (IBUFF, IROW, ICOL, NCOLS, NROWS)

Declarations:

INTEGER*2 IBUFF(NCOLS,NROWS), IROW, ICOL, NCOLS, NROWS

Argument Description:

IBUFF	Output	An array to contain the characters and attributes of the defined window. This should be dimensioned to NCOLS by NROWS.
IROW	Input	The beginning row number of the window (the first line on the screen is row zero).
ICOL	Input	The starting column number of the window (the left-most column is column zero).
NCOLS	Input	The number of columns in the window.
NROWS	Input	The number of rows in the window.

Example:

Save the entire screen:

```
      INTEGER*2 IBUFF(80,25)
C
      CALL PUFBFR (IBUFF, 0, 0, 80, 25)
```

3.4.14.13 PUFBFW – Write a Screen Window to the Display

Purpose:

PUFBFW restores a screen window (or the entire screen) from the array read by subroutine PUFBFR. PUFBFW restores the screen very quickly.

Calling Sequence:

CALL PUFBFW (IBUFF, IROW, ICOL, NCOLS, NROWS)

Declarations:

INTEGER*2 IBUFF(NCOLS,NROWS), IROW, ICOL, NCOLS, NROWS

Argument Description:

IBUFF	Output	The array read by PUFBFR.
IROW	Input	The beginning row number of the window (the first line on the screen is row zero).
ICOL	Input	The starting column number of the window (the left-most column is column zero).
NCOLS	Input	The number of columns in the window.
NROWS	Input	The number of rows in the window.

4 Date and Time Subroutines

The following section describes subroutines that deal with dates and times. This includes obtaining the current system date and time, and changing the date and time to different formats.

Several of the subroutines use Julian dates, in days since 31DEC1899 (not days since the beginning of the year). This form of date provides an exact and relative easy means of dealing with time-date information (for example, to increment the date by one day, one is added to the Julian date, whereas a more complex algorithm would be required for a military style date such as 28FEB1972). Julian dates can be negative, allowing for times in the 1800's or earlier. A Julian date can be converted to another style date (of which many forms are available) using the subroutine JULDAT. Conversely, different styles of dates can be converted to Julian using the subroutine DATJUL.

Several of the subroutines pass time information in minutes past midnight. The time in minutes can be converted to a twenty-four hour military style time (e.g., 1430 is 2:30 p.m.) by the subroutine M2IHM, and back to minutes with subroutine IHM2M. The time interval is given in minutes.

On MS-DOS microcomputers, the Julian dates and the time interval must always be declared as INTEGER*4.

4.1 DATYMD – Convert a Character Date to Integer-Year-Month-Day

Purpose:

DATYMD takes a character date, in a variety of styles, and converts it into an integer year-month-day style date. If no year is provided, the current year is returned. If no day is provided, the first of the month is returned. DATYMD will convert any of the dates produced by subroutine YMDDAT (or JULDAT). An example list of date styles is given.

Calling Sequence:

CALL DATYMD (CDATE, IYEAR, IMONTH, IDAY, IERROR)

Declarations:

```
INTEGER IYEAR, IMONTH, IDAY, IERROR
CHARACTER CDATE*20
```

Argument Description:

CDATE	Input	A character string containing the date to be converted. If no year is provided, the current year is returned. If no day is provided, the first of the month is returned (a month must be given).
IYEAR	Output	The year of the date. This will be a four digit year (e.g., 1982 instead of 82). If no year is given, the current year will be returned.
IMONTH	Output	The month number of the date provided (January is 1, February is 2, etc.).
IDAY	Output	The day of the date. If no day is given, the first of the month is returned.
IERROR	Output	A status parameter indicating the successfulness of the conversion. If IERROR is returned as zero, the date was converted. If IERROR is returned as -1, an invalid date was given.

Remarks:

DATYMD will convert the date successfully as long it can recognize the first three characters of the month (unless a style of 3/21/82 is passed), which may be either in lower or upper case. DATYMD assumes that the year (if given) is at the end of the character string.

If a two digit year is given, it is assumed to be for the 1900's. DATYMD will recognize dates for the 1800's (and earlier), as long as a four digit year is specified.

Example dates that are recognized by DATYMD are:

March 21, 1982	
Mar 21, 82	
21MAR82	
21 Mar 1882	
3/21/82	
3-21-82	
March 82	(The date for March 1, 1982 is returned)
21 March	(The date for March 21, of the current year is returned)
March 21	(The date for March 1, 1921 is returned, not the date for the 21st of March as the year is always assumed to be at the end of the date)
3-21	(The date for March 21, of the current year is returned)

4.2 DATJUL – Convert a Character Date to Julian

Purpose:

DATJUL takes a character date, in a variety of styles, and converts it into a Julian date in days since December 31, 1899. If no year is provided, the current year is assumed. If no day is provided, the first of the month is assumed. DATJUL will convert any of the dates produced by subroutine JULDAT (or YMDDAT). An example list of date styles is given.

Calling Sequence:

CALL DATJUL (CDATE, JULIAN, IERROR)

Declarations:

```
INTEGER JULIAN, IERROR
CHARACTER CDATE*20
```

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

```
INTEGER*4 JULIAN
```

Argument Description:

CDATE	Input	A character string containing the date to be converted. If no year is provided, the current year is assumed. If no day is provided, the first of the month is assumed (a month must be given).
JULIAN	Output	The Julian date of CDATE, in days since December 31, 1899.
IERROR	Output	A status parameter indicating the successfulness of the conversion. If IERROR is returned as zero, the date was converted. If IERROR is returned as -1, an invalid date was given, and JULIAN will be returned as -777777.

Remarks:

DATJUL will convert the date successfully as long it can recognize the first three characters of the month (unless a style of 3/21/82 is passed), which may be either in lower or upper case. DATJUL assumes that the year (if given) is at the end of the character string.

If a two digit year is provided, it is assumed to be for the 1900's. DATJUL will recognize dates for the 1800's (and earlier), as long as a four digit year is specified.

Example dates that are recognized by DATJUL are:

```
Mar 21, 82
21MAR82
```

21 Mar 1882	
3/21/82	
3-21-82	
March 82	(The date for March 1, 1982 is returned)
21 March	(The date for March 21, of the current year is returned)
March 21	(The date for March 1, 1921 is returned, not the date for the 21st of March as the year is always assumed to be at the end of the date)
3-21	(The date for March 21, of the current year is returned)

The subroutine DATYMD is used in the conversion.

4.3 YMDDAT - Convert an Integer Year-Month-Day Date into a Character Date

Purpose:

YMDDAT takes an integer date in the form of year-month-day, and converts it into a character date in one of a variety of styles. A list of the styles follows.

Calling Sequence:

CALL YMDDAT (IYEAR, IMONTH, IDAY, ISTYLE, CDATE, NDATE, IERROR)

Declarations:

```
INTEGER IYEAR, IMONTH, IDAY, ISTYLE, NDATE, IERROR
CHARACTER CDATE*20
```

Argument Description:

IYEAR	Input	The year portion of the date. This can either be a two digit or four digit number. For dates prior to 1900, a four digit number is required.
IMONTH	Input	The integer month portion of the date (e.g., 1 corresponds to January, 2 to February, etc.). This must be a number between one and twelve.
IDAY	Input	The integer day portion of the date. This must be a number between one and thirty-one.
ISTYLE	Input	The style of date to return. A complete list of the styles follows.
CDATE	Output	The returned character date.
NDATE	Output	The number of characters in the date. Characters beyond NDATE are not changed. (If you print or pass CDATE, imply the length by printing or passing CDATE(1:NDATE)).
IERROR	Output	A status flag indicating if an error occurred. If the date was converted properly, IERROR will be returned as zero, otherwise IERROR will be returned as -1.

Styles:

There are eleven basic style of dates, and four versions of each style. The differences are whether the month should be upper or lower case, and whether a two or four digit year should be used. The lower case styles of dates are:

ISTYLE	Form	ISTYLE	Form
0	June 2, 1985	10	June 2, 85
1	Jun 2, 1985	11	Jun 2, 85
2	2 June 1985	12	2 June 85
3	June 1985	13	June 85
4	02Jun1985	14	02Jun85
5	2Jun1985	15	2Jun85
6	Jun1985	16	Jun85
7	02 Jun 1985	17	02 Jun 85
8	2 Jun 1985	18	2 Jun 85
9	Jun 1985	19	Jun 85

The upper case styles of dates are:

ISTYLE	Form	ISTYLE	Form
100	JUNE 2, 1985	110	JUNE 2, 85
101	JUN 2, 1985	111	JUN 2, 85
02	2 JUNE 1985	112	2 JUNE 85
103	JUNE 1985	113	JUNE 85
104	02JUN1985	114	02JUN85
105	2JUN1985	115	2JUN85
106	JUN1985	116	JUN85
107	02 JUN 1985	117	02 JUN 85
108	2 JUN 1985	118	2 JUN 85
109	JUN 1985	119	JUN 85

The month-day-year style of dates are:

ISTYLE	Form
-1	6/2/85
-2	6-2-85
-11	06/02/85
-12	06-02-85

Remarks:

If CDATE is not declared large enough, the date will be truncated to fit in CDATE. It is prudent to use CDATE(1:NDATE), or to pre-blank CDATE prior to calling YMDDAT, as characters beyond NDATE are unchanged.

4.4 JULDAT – Convert a Julian Date into a Character Date

Purpose:

JULDAT takes a Julian date, in days since December 31, 1899, and converts it into a character date in one of a variety of styles. A list of the styles follows.

Calling Sequence:

CALL JULDAT (JULIAN, ISTYLE, CDATE, NDATE)

Declarations:

```
INTEGER JULIAN, ISTYLE, NDATE
CHARACTER CDATE*20
```

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

```
INTEGER*4 JULIAN
```

Argument Description:

JULIAN	Input	The Julian date, in days since December 31, 1899. JULIAN can be a negative number for dates in the 1800's, but a style that includes the full four digit year should be selected.
ISTYLE	Input	The style of date to return. A complete list of the styles follows.
CDATE	Input	The returned character date.
NDATE	Output	The number of characters in the date. Characters beyond NDATE are not changed. (If you print or pass CDATE, imply the length by printing or passing CDATE(1:NDATE)).

Remarks:

If CDATE is not declared large enough, the date will be truncated to fit in CDATE. It is prudent to use CDATE(1:NDATE), or to pre-blank CDATE prior to calling YMDDAT, as characters beyond NDATE are unchanged.

Styles:

There are eleven basic style of dates, and four versions of each style. The differences are whether the month should be upper or lower case, and whether a two or four digit year should be used. The lower case styles of dates are:

ISTYLE	Form	ISTYLE	Form
0	June 2, 1985	10	June 2, 85
1	Jun 2, 1985	11	Jun 2, 85
2	2 June 1985	12	2 June 85
3	June 1985	13	June 85
4	02Jun1985	14	02Jun85
5	2Jun1985	15	2Jun85
6	Jun1985	16	Jun85
7	02 Jun 1985	17	02 Jun 85
8	2 Jun 1985	18	2 Jun 85
9	Jun 1985	19	Jun 85

The upper case styles of dates are:

ISTYLE	Form	ISTYLE	Form
100	JUNE 2, 1985	110	JUNE 2, 85
101	JUN 2, 1985	111	JUN 2, 85
02	2 JUNE 1985	112	2 JUNE 85
103	JUNE 1985	113	JUNE 85
104	02JUN1985	114	02JUN85
105	2JUN1985	115	2JUN85
106	JUN1985	116	JUN85
107	02 JUN 1985	117	02 JUN 85
108	2 JUN 1985	118	2 JUN 85
109	JUN 1985	119	JUN 85

The month-day-year style of dates is:

ISTYLE	Form
-1	6/2/85
-2	6-2-85
-11	06/02/85
-12	06-02-85

4.5 IYMDJL – Convert an Integer Year-Month-Day Date to Julian

Purpose:

The integer function IYMDJL takes an integer date in the form of year, month, day, and converts it into a Julian date in days since December 31, 1899.

Calling Sequence:

JULIAN = IYMDJL (IYEAR, IMONTH, IDAY)

Declarations:

INTEGER JULIAN, IYEAR, IMONTH, IDAY

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

INTEGER*4 JULIAN

Argument Description:

IYEAR	Input	The year portion of the date. This can either be a two-digit or four-digit number. For years prior to 1900, a four-digit number is required.
IMONTH	Input	The integer month portion of the date (e.g., 1 corresponds to January, 2 to February, etc.). This must be a number between one and twelve.
IDAY	Input	The day of the date. This must be a number between one and thirty-one.
JULIAN	Output	The Julian date, in days since December 31, 1899. If an illegal date was passed, JULIAN will be returned as -777777.

Remarks:

Dates prior to December 31, 1899 can be obtained, as long as a 4 digit year is used. This would result in a negative Julian date.

4.6 JLIYMD – Convert a Julian Date into an Integer Year-Month-Day Date

Purpose:

The integer function JLIYMD takes a Julian date in days since December 31, 1899, and converts into an integer year-month-day style date.

Calling Sequence:

IDUMMY = JLIYMD (JULIAN, IYEAR, IMONTH, IDAY)

Declarations:

INTEGER JULIAN, IYEAR, IMONTH, IDAY, IDUMMY

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

INTEGER*4 JULIAN

Argument Description:

JULIAN	Input	The Julian date, in days since December 31, 1899. JULIAN can be a negative number for dates in the 1800's.
IYEAR	Output	The integer year of the Julian date provided. This will be returned as a 4 digit year (e.g., 1979 instead of 79).
IMONTH	Output	The month number of the Julian date (January is 1, February is 2, etc.).
IDAY	Output	The integer day of the Julian date.
IDUMMY	Output	A dummy integer variable.

Remarks:

JLIYMD is a function rather than a subroutine for compatibility reasons. Thus IDUMMY is a dummy variable.

4.7 IDAYWK – Get the Day of the Week from a Julian Date

Purpose:

Given a Julian day, in days since December 31, 1899, the integer function IDAYWK returns the day of the week for that date. IDAYWK is returned as a 1 for Sunday, a 2 for Monday, etc.

Calling Sequence:

NDAY = IDAYWK (JULIAN)

Declarations:

INTEGER JULIAN, NDAY

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

INTEGER*4 JULIAN

Argument Description:

JULIAN	Input	The Julian date, in days since December 31, 1899.
NDAY	Output	The day number of the week (ranging between one and seven). For a Sunday, NDAY is returned as one; for a Monday, NDAY is returned as two, etc.

4.8 IHM2M – Convert a Twenty-Four Hour Clock Time to Minutes

Purpose:

IHM2M takes a character string containing a twenty-four hour military style clock time (e.g., '1630'), and converts it into minutes past midnight.

Calling Sequence:

MINUTE = IHM2M (CTIME)

Declarations:

```
INTEGER MINUTE
CHARACTER CTIME*4
```

Argument Description:

CTIME	Input	A character string containing the twenty-four hour clock time (e.g., '1422').
MINUTE	Output	An integer number returned with CTIME converted to minutes past midnight. If an illegal time was passed, MINUTE is returned as -1.

Remarks:

The twenty-four hour clock time should always be four digits long. For example, the time ' 900' is valid, but '9' or '09' is not.

If you desire to convert an integer number containing a twenty-four hour clock time to minutes past midnight, the following code can be used:

```
IHOUR = ITIME/100
IMIN = ITIME - (IHOUR*100)
MINUTE = (IHOUR*60) + IMIN
```

4.9 M2IHM – Convert a Time in Minutes to Twenty-Four Hour Clock Time

Purpose:

M2IHM takes a time, in minutes past midnight, and converts it into a twenty-four hour four character military style clock time (e.g., 1630 or 0900). The clock time is returned as both an integer number and in a character string.

Calling Sequence:

ITIME = M2IHM (MINUTE, CTIME)

Declarations:

```
INTEGER MINUTE, M2IHM
CHARACTER CTIME*4
```

Argument Description:

MINUTE	Input	The time, in minutes past midnight.
CTIME	Output	The time returned in twenty-four hour military style clock time (e.g., '1630'). This must be a character variable.
M2IHM	Output	The time in twenty-four hour clock time returned as an integer number.

Remarks:

M2IHM returns both a character string and an integer number form of the time. CTIME must be a character variable with a length of at least four. If an invalid time is passed (MINUTE less than zero or greater than 1440, ITIME is returned as -1, and CTIME is filled with asterisks (*).

If an integer time only is desired, it is usually better to use the following code:

```
IHR = MINUTE/60
IMIN = MINUTE - (IHR*60)
ITIME = IHR*100 + IMIN
```

4.10 INCTIM – Increment a Date and Time

Purpose:

The integer function INCTIM increments a Julian date and time a specified number of periods, based on a given time interval. INCTIM handles leap years and the different number of days in the different months.

Calling Sequence:

IDUMMY = INCTIM (INTL, IFLAG, NPER, JULS, ISTIME, JULE, IETIME)

Declarations:

INTEGER INTL, IFLAG, NPER, JULS, ISTIME, JULE, IETIME, IDUMMY

On MS-DOS microcomputers, the Julian dates and time interval must be INTEGER*4:
INTEGER*4 JULS, JULE, INTL

Argument Description:

INTL	Input	The time interval corresponding to the number of periods to increment the date and time by. This is usually given in minutes (IFLAG=0), but may be specified in days (IFLAG=1) for larger intervals.
IFLAG	Input	IFLAG indicates the units of INTL. If INTL is given in minutes (the typical case), set IFLAG to zero. If INTL is given in days, set IFLAG to one.
NPER	Input	The number of periods to increment the date and time by. NPER may be a negative number to decrement the date and time.
JULS	Input	The starting Julian date, in days since December 31, 1899.
ISTIME	Input	The starting time, in minutes past midnight.
JULE	Output	The incremented Julian date, in days since December 31, 1899.
IETIME	Output	The incremented time, in minutes past midnight.
IDUMMY	Output	A dummy variable allowing INCTIM to be a integer function. This is for compatibility reasons

Remarks:

A time interval of one year is the only valid interval greater than one month.

If a monthly interval is used with a Julian date corresponding to the end of the month, the resultant incremented date will be for the end of the month. For example, if Jan. 31 is incremented one month, the result would be Feb. 28 (depending on leap year). If Feb. 28 is incremented one month, the result would be March 31.

INCTIM has been tested on HARRIS computers for periods of between -10,000 and 10,000 for intervals of one month and less, and for periods of between -1,000 and 1,000 for intervals of one year. Be cautious of round off errors for larger increments.

Examples:

```
      C     Increment a time/date 500 periods for an interval of 1 hour
            READ (5,*) JULS, ISTIME
            IDUM = INCTIM (60, 0, 500, JULS, ISTIME, JULE, IETIME)

      C     Obtain a time window for the last week.
      C     Get the current time, then decrement it one week
            CALL CURTIM (JULE, IETIME)
            IDUM = INCTIM (1440, 0, -7, JULE, IETIME, JULS, ISTIME)

      C     Obtain a time window spanning 50 years (ignoring time of day)
            READ (5,*) JULS
            IDUM = INCTIM (365, 1, 50, JULS, 1440, JULE, JDUM)
or, alternatively:
            IDUM = INCTIM (525600, 0, 50, JULS, 1440, JULE, JDUM)

      C     Get a time 7 months ago.
            CALL CURTIM (JULE, IETIME)
            IDUM = INCTIM (30, 1, -7, JULE, IETIME, JULS, ISTIME)
or, alternatively:
            IDUM = INCTIM (43200, 0, -7, JULE, IETIME, JULS, ISTIME)
```

4.11 NOPERS – Determine the Number of Periods between Two Times

Purpose:

Given two dates and times, and a time interval, the integer function NOPERS will determine the number of periods between them. This is the inverse function of routine INCTIM.

Calling Sequence:

NPER = NOPERS (INTL, IFLAG, JULS, ISTIME, JULE, IETIME)

Declarations:

INTEGER INTL, IFLAG, NPER, JULS, ISTIME, JULE, IETIME

On MS-DOS microcomputers, the Julian date and time interval must be INTEGER*4:
INTEGER*4 JULS, JULE, INTL

Argument Description:

INTL	Input	The time interval corresponding to the number of periods to determine. This is usually given in minutes (IFLAG=0), but may be specified in days (IFLAG=1) for larger intervals.
IFLAG	Input	IFLAG indicates the units of INTL. If INTL is given in minutes (the typical case), set IFLAG to zero. If INTL is given in days, set IFLAG to one.
JULS	Input	The Julian date of the start of the time window, in days since December 31, 1899.
ISTIME	Input	The starting time, in minutes past midnight.
JULE	Input	The Julian date of the end of the time window, in days since December 31, 1899.
IETIME	Input	The ending time, in minutes past midnight.
NPER	Output	The number of time periods. NPER may be negative if the ending date/time is prior to the starting date/time.

Remarks:

A time interval of one year is the only valid interval greater than one month.

NOPERS has been tested on HARRIS computers for periods of between -10,000 and 10,000 for intervals of one month and less, and for periods of between -1,000 and 1,000 for intervals of one year. Be cautious of round off errors for larger increments.

4.12 CURTIM – Get the Current Julian Date and Time

Purpose:

CURTIM returns the current date in Julian days since December 31, 1899, and the current time in minutes past midnight. This style of date and time can be used with most of the other HECLIB time and date subroutines.

Calling Sequence:

CALL CURTIM (JULIAN, MINUTE)

Declarations:

INTEGER JULIAN, MINUTE

On MS-DOS microcomputers, the Julian date must be INTEGER*4:

INTEGER*4 JULIAN

Argument Description:

JULIAN	Output	The current Julian date, in days since December 31, 1899 (according to the system clock).
MINUTE	Output	The current time in minutes past midnight.

Remarks:

The subroutine WHEN may be used to obtain the current date and time in character form.

4.13 DATIME – Get Current Date and Time

Purpose:

DATIME returns the current system date and time. This is in a format of the year, Julian day of the year (from January 1, not HEC's Julian date), and time in tenths of a second past midnight.

Calling Sequence:

CALL DATIME (IYEAR, JDAY, ITENTH)

Declarations:

INTEGER IYEAR, JDAY, ITENTH

Argument Description:

IYEAR	Output	The current four digit year (e.g., 1987, not 87).
JDAY	Output	The current Julian day from the first of the year. (This is not the Julian day referenced in the other date routines.)
ITENTH	Output	The current system time, in tenths of a second past midnight.

Remarks:

Other time/date subroutines are usually called instead of DATIME. Refer to the subroutines CURTIM, CDATE, CTIME, and WHEN.

4.14 WHEN – Get the Current Date and Time in Character Form

Purpose:

WHEN returns the current date and time in a character format. The date is given in a seven character military style date (e.g., 07JAN83), and the time is returned in an eight character hours, minutes, seconds style format (e.g., 08:32:45).

Calling Sequence:

```
CALL WHEN (CDATE, CTIME)
```

Declarations:

```
CHARACTER CDATE*7, CTIME*8
```

Argument Description:

CDATE	Output	The current date, returned in a seven character military style date (e.g., 07JAN83).
CTIME	Output	The current time, returned in an eight character hours, minutes, seconds style format (e.g., 08:32:45).

Remarks:

The date and time are returned according to the computer's clock.

If a different style date is desired, call HECLIB subroutine CURTIM then HECLIB subroutine JULDAT with the selected style. A four character military style time may be obtained by calling CURTIM then M2IHM.

4.15 CDATE – Get the Current Date

Purpose:

CDATE returns the current system date in a nine character military style format. An example date is '08 MAR 82'.

Calling Sequence:

CALL CDATE (CCDATE)

Declarations:

CHARACTER CCDATE*9

Argument Description:

CCDATE	Output	The current system date, in a nine character style format.

4.16 CTIME – Get the Current Time

Purpose:

CTIME returns the current system time in an eight character hour, minute, second format. An example of this format is '08:30:15'.

Calling Sequence:

CALL CTIME (CCTIME)

Declarations:

CHARACTER CCTIME*8

Argument Description:

CCTIME	Output	The current system time.

4.17 WAITS – Wait for a Specified Amount of Time

Purpose:

WAITS will cause the calling program to pause for the specified amount of time. The time specified is given in seconds and fractions of a second. The smallest amount of time WAITS can pause is generally about 0.01 second.

Calling Sequence:

CALL WAITS (SECS)

Declarations:

REAL SECS

Argument Description:

SECS	Input	A real number containing the time to wait, in seconds.

Examples:

Wait for three-quarters of a second:

```
CALL WAITS (0.75)
```

Wait for thirty seconds:

```
CALL WAITS (30.0)
```

Wait for one-twentieth of a second:

```
CALL WAITS (0.05)
```

4.18 XTIME – Get the Current CPU Time for the Session

Purpose:

XTIME returns the amount of CPU time used for the current session. On MS-DOS microcomputers, this is the number of seconds past midnight.

Calling Sequence:

CALL XTIME (SECS)

Declarations:

REAL SECS

Argument Description:

SECS	Output	The elapsed CPU time since the beginning of the session. This is returned in seconds and fractions of a second.

4.19 GETIME – Get Time Window from a Program Command Line

Purpose:

Subroutine GETIME obtains a time window from a program command line input. This is the subroutine called by programs DSPLAY and DSSUTL to specify the time window.

Calling Sequence:

CALL GETIME (CLINE, IBEG, ILEN, JULS, ISTIME, JULE, IETIME, ISTAT)

Declarations:

```
CHARACTER CLINE
INTEGER IBEG, ILEN, JULS, ISTIME, JULE, IETIME, ISTAT
```

On MS-DOS microcomputers, the Julian dates must be INTEGER*4:

```
INTEGER*4 JULS, JULE
```

Argument Description:

CLINE	Input	The program command line containing the users time window input.
IBEG	Input	The beginning character position in CLINE to process.
ILEN	Input	The number of characters in CLINE to process.
JULS	Input/ Output	The starting Julian date of the time window, in days since December 31, 1899. This is changed (or not changed) according to the input on CLINE. If the time window is cleared, JULS is set to -777777.
ISTIME	Input/ Output	The starting time of the time window, in minutes past midnight (for midnight ISTIME is 1440, not zero). If the time window is cleared, ISTIME is set to -1.
JULE	Input/ Output	The ending Julian date of the time window, in days since December 31, 1899. If the time window is cleared, JULE is set to -777777.
IETIME	Input/ Output	The ending time of the time window, in minutes past midnight. If the time window is cleared, IETIME is set to -1.

ISTAT	Output	A status parameter. If ISTAT is returned as zero, the time window was set without error. If ISTAT is returned as one, the time window was cleared. If ISTAT is returned as negative one, some error occurred and the time window was cleared.

Remarks:

A time window is specified by entering the starting date and time followed by the ending date and time. The time or date may be in either order (as long as the starting time and date precedes the ending time and date).

A time must be a four digit number, given in twenty-four hour clock time. A date can be one of several styles, but must not contain any spaces within it. (A seven or nine character military style date is the style typically used.) A time window may also be set relative to the system time by using the single character "T" or "D".

The ending date/time may be changed without affecting the beginning date/time by leaving empty fields (specified by commas). A date/time offset may be given by specifying the number of hours (H), days (D) or years (Y) to add or subtract from the previous date/time settings, or with the current date/time reference "T". The time registers may be cleared by sending a blank line to GETIME (or setting ILEN to zero).

Valid examples include:

01MAR72, 2400 18SEP72, 1200	
2400 01MAR72 1200,18SEP1972	
T-4H, T	(current date/time - four hours, current date/time)
D, 1600	(today at 4 p.m.)
-2D +8H	(subtract two days from the starting date/time; add eight hours to the ending date/time)
T-5Y, T-31D	(today - five years, today – thirty-one days)
,,,+12H	(add twelve hours to the ending date/time)

5 Character Manipulation Subroutines

The following section describes subroutines that operate on character strings. This includes scanning for specified characters (or delimiters), moving character strings, and converting character data into Hollerith (and visa-versa).

Some of the heavily used subroutines (such as CHRLNB and CHRBLK) are written in assembly language.

5.1 CHRBLK – Fill a Character String with Blanks

Purpose:

CHRBLK places the space character (' ') throughout a character string.

Calling Sequence:

CALL CHRBLK (CSTR)

Declarations:

CHARACTER CSTR

Argument Description:

CSTR	Output	The character string to blank fill. The beginning and ending position are implicit (e.g., CSTR(5:50)).

Remarks:

On HARRIS computers and MS-DOS microcomputers, CHRLNB uses assembly code for increased efficiency. A FORTRAN substitute is available for other computers.

CHRBLK replaces subroutines STRBLK and CHABLK.

5.2 CHRFIL – Fill a Character String with a Specified Character

Purpose:

CHRFIL fills a given character string with a given character.

Calling Sequence:

CALL CHRFIL (CSTR, CHR)

Declarations:

CHARACTER CSTR, CHR*1

Argument Description:

CSTR	Output	The character variable for which each character will be replaced with character CHR. The beginning and ending locations are implicit.
CHR	Input	The character to fill CSTR with.

Example:

CHRFIL may be used in generating the outlines of a table, for example:

```
CALL CHRFIL (CSTR(1:5), ' ')
CALL CHRFIL (CSTR(6:70), '-')
WRITE (6,'(A)') CSTR(1:70)
```

will output a line of five blanks followed by sixty-five dashes (-).

5.3 CHRLNB – Locate the Last Non-Blank Character

Purpose:

CHRLNB determines the position of the last non-blank character in a character string.

Calling Sequence:

```
CALL CHRLNB (CSTR, ILAST)
```

Declarations:

```
CHARACTER CSTR
INTEGER ILAST
```

Argument Description:

CSTR	Input	The character string in which to locate the position of the last non-blank character. The length of the string is implicit (e.g., CSTR(5:50)).
ILAST	Output	The position of the last non-blank character in CSTR. If CSTR is completely blank filled, ILAST is returned as zero.

Remarks:

On HARRIS computers and MS-DOS microcomputers, CHRLNB uses assembly code for increased efficiency. A FORTRAN substitute is available for other computers.

CHRLNB replaces LASTCH.

5.4 LFLNB – Locate the First and Last Non-Blank

Purpose:

LFLNB determines the position of the first and last non-blank character in a character string. If only the position of the last non-blank character is desired, use subroutine CHRLNB instead.

Calling Sequence:

CALL LFLNB (CSTR, IBEG, ILEN, IFNB, NLEN)

Declarations:

```
CHARACTER CSTR
INTEGER IBEG, ILEN, IFNB, NLEN
```

Argument Description:

CSTR	Input	The character string in which to determine the positions of the first and last non-blank characters.
IBEG	Input	The beginning character position in which to start searching.
ILEN	Input	The length (number of characters) from IBEG in which to search.
IFNB	Output	The position of the first non-blank character, relative to the beginning of CSTR (not to IBEG). If the entire string is blank filled, IFNB is returned as zero.
NLEN	Output	The number of characters from IFNB to the position of the last non-blank character (Note: this is the length, not the ending position). If the entire string is blank filled, NLEN is returned as zero.

5.5 REMBLK – Remove Blanks from a String

Purpose:

REMBLK removes all blanks from a string (while left justifying the string).

Calling Sequence:

CALL REMBLK (CIN, COUT, NOUT)

Declarations:

```
CHARACTER CIN, COUT
INTEGER NOUT
```

Argument Description:

CIN	Input	The character string in which to remove the blanks from.
COUT	Output	A character variable that will contain the compressed string. If the length of COUT is less than the number of characters to be placed in it, it will be truncated.
NOUT	Output	The number of characters placed in COUT. If CIN contains all blanks, NOUT will be returned as zero.

Example:

If:

```
CALL REMBLK (' THIS IS A TEST LINE.  ', COUT, NOUT)
```

Then:

```
NOUT = 16
COUT(1:NOUT) = 'THISISATESTLINE.'
```

5.6 UPCASE – Convert a Character String to Upper Case

Purpose:

UPCASE converts all characters in a character string to upper case. This provides a means for programs to read input in both lower and upper case, and treat it the same.

Calling Sequence:

```
CALL UPCASE (CLINE)
```

Declarations:

```
CHARACTER CLINE
```

Argument Description:

CLINE	Input/ Output	The character variable containing the string to be converted to upper case. If the string is already in upper case, no processing will be done.

Example:

```
CALL ANREAD (5, 'Enter Yes or No >', 17, CLINE, NLINE)
CALL UPCASE (CLINE)
IF (CLINE(1:1).EQ.'Y') THEN
. . .
```

5.7 MATCH – Search a List for a Character String

Purpose:

MATCH searches a character list (array) for the occurrence of a character string. The number of the element matching that string is returned. If no matches were found, zero is returned. MATCH was designed to determine what command from a program has been entered.

Calling Sequence:

CALL MATCH (CSTR, IBEG, ILEN, CLIST, NLIST, NLEN, IMATCH)

Declarations:

```
CHARACTER CSTR, CLIST(NLIST)
INTEGER IBEG, ILEN, NLIST, NLEN, IMATCH
```

Argument Description:

CSTR	Input	The character string to search for. Typically this is the command read from the input.
IBEG	Output	The beginning position in CSTR to compare.
ILEN	Input	The number of characters in CSTR, relative to IBEG, to compare.
CLIST	Input	A character array containing the strings to be searched. Typically this is an array containing command names.
NLIST	Input	The number of entries in CLIST. This may also be the dimension of CLIST.
NLEN	Input	The length of the character elements in CLIST.
IMATCH	Output	IMATCH is returned with the number of the element in CLIST that matched CSTR. Typically this will be the number of the command. If no match is found, IMATCH will be returned as zero.

Remarks:

MATCH will only scan for a match of ILEN characters, and will return on the first match found. If, for example, CSTR is one character long, then only the first character of each element in CLIST will be compared until a match is found. Thus, if a user abbreviates a command, it should be long enough to be unique.

Example:

```
    5   CONTINUE
        READ (5,10,END=100) CLINE
   10   CONTINUE
        J = ISCAN (CLINE, 1, 20, ', ', 1, 2, K)
        IF (J.EQ.0) GO TO 900
        CALL MATCH (CLINE, 1, J, CLIST, 34, 4, IMATCH)
C
        IF (IMATCH.EQ.0) THEN
           WRITE (6,*)'UNRECOGNIZED COMMAND;  REENTER'
           GO TO 5
        ENDIF
```

5.8 INDEXR - Reverse Index

Purpose:

The function INDEXR is similar to the FORTRAN INDEX function, except INDEXR searches in a reverse direction. INDEXR searches character string CSTR1 for the last occurrence of character string CSTR2 (i.e., searching from right to left). INDEXR will be returned as the position of the left most character in the match of the last occurrence of CSTR2. If CSTR2 is not found, INDEXR will be returned as zero.

Calling Sequence:

```
I = INDEXR (CSTR1,CSTR2)
```

Declarations:

```
INTEGER INDEXR
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to search. The beginning location and ending location are implied (e.g., CSTR1(8:35)).
CSTR2	Input	The character string to searched for.
INDEXR	Output	The position of the left-most character in the match of the last occurrence of CSTR2 in CSTR1, relative to the start of the search. If CSTR2 is not found, INDEXR will be returned as zero.

Remarks:

NDEXR searches for a string of characters, not individual characters. Refer to function ISCAN in order to search for individual characters.

Examples:

If:

```
                        1234567890123456789012345678901
   CSTR1(1:31) = '   THIS IS A TEST LINE, ABC ABC. '
```

Then:

```
   INDEXR(CSTR1,'.') = 30
   INDEXR(CSTR1,'ABC') = 27
   INDEXR(CSTR1,'ABC ') = 23
```

```
INDEXR(CSTR1,'A ') = 10
INDEXR(CSTR1(10:31),'A ') = 1
INDEXR(CSTR1(10:),'ABC') = 18
INDEXR(CSTR1,'BCA') = 0
```

5.9 NINDX – Search for the Non-Occurrence of a String

Purpose:

The function NINDX is similar to the FORTRAN INDEX function, except NINDX searches for the non-occurrence of a string. NINDX searches character string CSTR1 for the first non-occurrence of string CSTR2. NINDX is returned as the position of the left most character of the first non-match in CSTR1 of CSTR2. If CSTR2 matches all characters in CSTR1, NINDX is returned as zero. A typical use of NINDX is to search for the first non-blank in a string.

Calling Sequence:

I = NINDX(CSTR1,CSTR2)

Declarations:

```
INTEGER NINDX
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to search. The beginning location and ending location are implied (e.g., CSTR1(8:35)).
CSTR2	Input	The character string searched for its non-occurrence.
NINDX	Output	The position of the left most character in the first non-match of CSTR2. If CSTR2 matches all characters in CSTR1, NINDX will be returned as zero.

Remarks:

NINDX searches for the non-occurrence of a string of characters or a single character, not individual characters. Refer to function NSCAN in order to search for the non-occurrence of individual characters.

Usually NINDX is used to find the first non-blank in a string. Almost always, CSTR2 will only be one character long.

Examples:

If:

```
                          123456789012345678901234567
   CSTR1(1:27) = '      THIS IS A TEST LINE.   "
```

Then:
NINDX(CSTR1,' ') = 5
NINDX(CSTR1(1:4),' ') = 0
NINDX(CSTR1,'T') = 1

If:
123456789
CSTR1(1:9) = 'AAAABAAAA'

Then:
NINDX(CSTR1,'A') = 5
NINDX(CSTR1(3:),'A') = 3
NINDX(CSTR1(1:4),'A') = 0
NINDX(CSTR1,'B') = 1
NINDX(CSTR1,'AB') = 1

5.10 NINDXR – Search for the Last Non-Occurrence of a String

Purpose:

NINDXR provides a function similar to the FORTRAN function INDEX, except NINDXR searches for the last non-occurrence of a string. NINDXR searches character string CSTR1 (from right to left) for the last non-occurrence of string CSTR2. NINDXR is returned as the position of the left most character of the last non-match in CSTR1 of CSTR2. If CSTR2 matches all characters in CSTR1, NINDXR is returned as zero. NINDXR is usually used to search for the last non-blank character in a string.

Calling Sequence:

I = NINDXR (CSTR1,CSTR2)

Declarations:

```
INTEGER NINDXR
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to search. The beginning location and ending location are implied (e.g., CSTR1(8:35)).
CSTR2	Input	The character string searched for its non-occurrence.
NINDXR	Output	The position of the left most character in the last non-match of CSTR2. If CSTR2 matches all characters in CSTR1, NINDXR will be returned as zero.

Remarks:

NINDXR searches for the non-occurrence of a string of characters or a single character, not individual characters. Refer to function NSCAN in order to search for the non-occurrence of individual characters.

To search for the last non-blank of a character string, use subroutine CHRLNB.

Examples:

If:

```
                          123456789012345678901234567
   CSTR1(1:27) = '      THIS IS A TEST LINE.   "
```

Then:
NINDXR(CSTR1,' ') = 5
NINDXR(CSTR1(1:4),' ') = 0
NINDXR(CSTR1,'T') = 1

If:
123456789
CSTR1(1:9) = 'AAAABAAAA'

Then:
NINDXR(CSTR1,'A') = 5
NINDXR(CSTR1(3:),'A') = 3
NINDXR(CSTR1(1:4),'A') = 0
NINDXR(CSTR1,'B') = 9
NINDXR(CSTR1,'AB') = 9

5.11 ISCAN – Search a String for Individual Character(s)

Purpose:

ISCAN searches character string CSTR1 for the first or last occurrence of any characters in CSTR2. ISCAN is returned with the position of the first character in CSTR1 that matched any character in CSTR2. If no characters matched, ISCAN is returned zero. To make ISCAN search in a reverse direction (for the last occurrence), set the number of characters to scan to negative.

Calling Sequence:

I = ISCAN (CSTR1, NBEG1, NLEN1, CSTR2, NBEG2, NLEN2, IPOS2)

Declarations:

```
INTEGER ISCAN, NBEG1, NLEN1, NBEG2, NLEN2, IPOS2
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to search.
NBEG1	Input	The beginning position in CSTR1 to start searching.
NLEN1	Input	The number of characters in CSTR1 to search (from NBEG1). To cause ISCAN to search in a reverse direction (the last occurrence), set NLEN1 to negative. In this context, NBEG1 will actually be the ending position, as ISCAN will search in the order of NBEG1, NBEG1-1, NBEG1-2, etc.
CSTR2	Input	A character string with the individual characters to search for.
NBEG2	Input	The beginning character position in CSTR2 to be used.
NLEN2	Input	The number of characters in CSTR2 to be searched for (relative to NBEG2).
ISCAN	Output	The position of the first (or last) character in CSTR1 that matched a character in CSTR2, relative to the beginning of the string. If no matches were found, ISCAN is returned zero.
IPOS2	Output	The position of the character in CSTR2 for which there was a match in CSTR1.

Remarks:

The beginning and lengths of the character strings are explicitly given in the arguments NBEG1, NLEN1, etc. (This is a result of ISCAN originally being written in FORTRAN 66, where Holleriths were used instead of characters.) Note that ISCAN is returned as a position relative to the beginning of the string, not NBEG1.

ISCAN searches for individual characters, not a string of characters. Refer to the FORTRAN function INDEX or function INDEXR in order to search for a continuous string.

Examples:

If:

```
                    123456789012345678901
   CSTR1(1:21) = 'THIS, IS A TEST LINE.'
```

Then:

```
   ISCAN (CSTR1, 1, 21, ' ,', 1, 2, IPOS2) = 5;  IPOS2 = 2
   ISCAN (CSTR1, 1, 21, ' ,', 1, 1, IPOS2) = 6;  IPOS2 = 1
   ISCAN (CSTR1, 7, 15, ' ,', 1, 2, IPOS2) = 9;  IPOS2 = 1
   ISCAN (CSTR1, 1,  4, ' ,', 1, 2, IPOS2) = 0;  IPOS2 = 0
   ISCAN (CSTR1, 21, -21, ' ,', 1, 2, IPOS2) = 16;  IPOS2 = 1
   ISCAN (CSTR1, 1, 21, CSTR1, 17, 5, IPOS2) = 3;  IPOS2 = 18
   ISCAN (CSTR1, 1, 21, 'LINE.', 1, 5, IPOS2) = 3;  IPOS2 = 3
```

Note that if NLEN1 is -21, NBEG1 must be greater than or equal to twenty-one. If NBEG1 were, for example, one, then ISCAN would try to search from one through -19 (1, 0, -1). This would result in an illegal string bounds error.

5.12 NSCAN – Search a String for the Non-Occurrence of Individual Character(s)

Purpose:

NSCAN searches character string CSTR1 for the first or last non-occurrence of any characters in CSTR2. NSCAN is returned with the position of the first character in CSTR1 that did not match any character in CSTR2, relative to the beginning of CSTR1. If all characters matched, NSCAN is returned zero. To cause NSCAN to search in a reverse direction (for the last non-occurrence), make the number of characters to scan negative.

Calling Sequence:

I = NSCAN (CSTR1, NBEG1, NLEN1, CSTR2, NBEG2, NLEN2)

Declarations:

```
INTEGER NSCAN, NBEG1, NLEN1, NBEG2, NLEN2
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to search.
NBEG1	Input	The beginning position in CSTR1 to start searching.
NLEN1	Input	The number of characters in CSTR1 to search (from NBEG1). Note that this is not the ending position. To cause NSCAN to search in a reverse direction (the last occurrence), set NLEN1 to negative. In this context, NBEG1 will actually be the ending position, as NSCAN will search in the order of NBEG1, NBEG1-1, NBEG1-2, etc.
CSTR2	Input	A character string containing the characters searched for non-occurrence.
NBEG2	Input	The beginning character position in CSTR2.
NLEN2	Input	The number of characters in CSTR2 to use (relative to NBEG2).
NSCAN	Output	The position of the first (or last) character in CSTR1 that did not match a character in CSTR2, relative to the beginning of the string. If all matches were found, NSCAN is returned zero.

Remarks:

The beginning and lengths of the character strings are explicitly given in the arguments NBEG1, NLEN1, etc.. (This is a result of NSCAN originally being written in FORTRAN 66, where Holleriths were used instead of characters.) Note that NSCAN is returned as a position relative to the beginning of the string, not NBEG1.

NSCAN searches for individual characters, not a string of characters. Refer to the function NINDX or NINDXR for a continuous string.

Examples:

If:

```
                    1234567890123
CSTR1(1:13) = ' 13FEB1987   '
```

Then:

```
NSCAN (CSTR1, 1, 13, ' ,', 1, 2) = 2
NSCAN (CSTR1, 2, 12, '1234567890', 1, 10) = 4
NSCAN (CSTR1, 13, -13, ' ', 1, 1) = 10
NSCAN (CSTR1, 10, -10, '1234567890', 1, 10) = 6
```

Note that if NLEN1 is -13, NBEG1 must be greater than or equal to thirteen. If NBEG1 were, for example, one, then NSCAN would try to search from one through -10 (1, 0, -1). This would result in an illegal string bounds error.

5.13 FINDLM – Find Delimiters within a Character String

Purpose:

FINDLM locates the positions and lengths of fields (separated by delimiters) within a character string. FINDLM essentially provides the capability of reading from a character string in a "free format" form. The delimiters defining the fields may be set by calling subroutine SETDLM, otherwise default values will be used.

Calling Sequence:

```
      CALL FINDLM (CSTRNG, NBEG, NLEN, NFIELD, IBEGF, ILENF,
     * IDELMT, IDELMP, ITBL)
```

Declarations:

```
      CHARACTER CSTRNG
      INTEGER NBEG, NLEN, NFIELD, IBEGF(MAXF), ILENF(MAXF)
      INTEGER IDELMT(MAXF), IDELMP(MAXF), ITBL(128)
```

Typically

```
      PARAMETER (MAXF=20)
```

Argument Description:

CSTRNG	Output	The character string to search for delimiters.
NBEG	Output	The position in CSTRNG to begin the search.
NLEN	Output	The number of characters to search.
NFIELD	Input/ Output	If NFIELD is a negative number on input, then FINDLM will stop its search after the absolute value of NFIELD fields has been found. (This indicates the dimensions of arguments IBEGF, ILENF, IDELMT, and IDELMP.) On output, NFIELD is returned with the number of fields that were found in the string.
IBEGF	Output	An array returned with the beginning position of each field. IBEGF(1) corresponds to the beginning position of the first field, IBEGF(2) to the second field, up to IBEGF(NFIELD).
ILENF	Output	An array returned with the length (number of characters) in each field, with respect to array IBEGF. ILENF(1) corresponds to the length of the first field, ILENF(2) to the

second field, up to ILENF(NFIELD). The ending position may be computed by:

IENDP = IBEGF(1) + ILENF(1) - 1

IDLEMT Output An array returned with the type of delimiter ending each field. The values range from one to five.

Type	Description
1	Normal delimiter, such as a comma.
2	A "blank" delimiter.
3	A string type delimiter.
4	The last character scanned (as defined by NLEN) was a non-delimiter.
5	The last character scanned (as defined by NLEN) was in the middle of a string.

IDELMP Output An array containing the position in the delimiter string of the delimiter found (set by SETDLM). IDELMP and IDELMT identify the ending delimiter of the field.

ITBL Input/ Output An integer array, dimension to 128 words, that contains information on the delimiters set. (Information in this array is automatically set by FINDLM and SETDLM.) On EBCDIC computers ITBL must be dimensioned to 260 words.

Remarks:

FINDLM scans for three basic types of delimiters. The first type is usually identified by a comma (,). The second type is normally associated with a blank ( ). Two commas delimit two fields, whereas two blanks delimit only one field. A field terminated by a comma, then a blank (or several blanks), is identified as a type two delimiter (and will only be one field). The third type is a string delimiter. When a string delimiter is found, FINDLM will scan until the next occurrence of the same (exact) delimiter is found. Delimiters inside this string are ignored.

The dimensions of arrays IBEGF, ILENF, IDELMT, and IDELMP may be passed to FINDLM by setting NFIELD as the negative value of the dimension. (This will prevent the arrays from being overwritten.)

If FINDLM is called without calling SETDLM first, default delimiters are used. If SETDLM is called prior to FINDLM, default values will be not used (unless explicitly set). Refer to subroutine SETDLM for more information. The default delimiter types are:

Type	Delimiters	Notes
1	,=/>()	
2		A blank character.
3	'"	Single or double quotes.

If a slash (/) ended a field, then IDELMT would be one, and IDELMP would be three for that field.

Example 1:

If:

```
                    123456789012345678901
CSTRNG =  32.3 45, 0.2,72.3 85.1

      CALL FINDLM (CSTRNG, 1, 22, NFIELD, IBEGF, ILENF,
     *  IDELMT, IDELMP, ITBL)
```

Then NFIELD would be returned with five, and the arrays would contain:

Field	IBEGF	ILENF	IDELMT	IDELMP
1	1	4	2	1
2	6	2	1	1
3	10	3	1	1
4	14	4	2	1
5	19	4	4	1

Example 2:

If:

```
                    12345678901234567890123456789012345678901234567890123456789012345678901234567890
  CSTRNG =  This is 'a "test line"' (showing delimiter types/positions).

  CALL FINDLM (CSTRNG, 1, 60, NFIELD, ILENF, IDELMT, IDELMP, ITBL)
```

Then NFIELD would be returned with eight, and the arrays would contain:

Field	IBEGF	ILENF	IDELMT	IDELMP
1	1	4	2	1
2	6	2	3	1
3	10	13	1	5
4	26	7	2	1
5	34	9	2	1
6	44	5	1	3
7	50	9	1	6
8	60	1	4	1

Example 3:

In the following code, FINDLM is used to read data in free format, with error handling.

```
PARAMETER (MAXF=20), (MAXDAT=1000)
CHARACTER CLINE*80
NTEGER IBEGF(MAXF), ILENF(MAXF), IDELMT(MAXF), IDELMP(MAXF)
INTEGER ITBL(128)
```

```
      REAL VALUES(MAXDAT)

      . . .
            NVALS = 0
10          CONTINUE
            READ (5, 20, END=100, ERR=900) CLINE
20          FORMAT (A)
            NFIELD = -MAXF
            CALL FINDLM (CLINE, 1, 80, NFIELD, IBEGF, ILENF, IDELMT,
*           IDELMP, ITBL)
            DO 30 I=1,NFIELD
              IF (NVALS.GE.MAXDAT) GO TO 800
              NVALS = NVALS + 1
              VALUES(NVALS) = XREAL (CLINE, IBEGF(I), ILENF(I), IERR)
              IF (IERR.NE.0) GO TO 920
30          CONTINUE
            GO TO 10
```

5.14 SETDLM – Set Delimiters for FINDLM

Purpose:

SETDLM sets the delimiters to be searched for by FINDLM (if other than the default ones are desired). If SETDLM is call prior to the first call to FINDLM, the default delimiters will not be set (unless explicitly set).

Calling Sequence:

CALL SETLDM (ITYPE, CSTRNG, IBEG, NUMB, ITBL)

Declarations:

```
INTEGER ITYPE, IBEG, NUMB, ITBL(128)
CHARACTER CSTRNG
```

Argument Description:

ITYPE — Input — The type of delimiter to set:

ITYPE	Description
0	Resets all delimiters to be the default ones
1	Normal delimiters, such as a comma (,).
2	Blank type delimiters.
3	String type delimiters, such as quotes (").

CSTRNG — Input — A character string containing the delimiters to be set. If delimiters are set to default values, this argument is ignored.

IBEG — Input — The beginning position in CSTRNG.

NUMB — Input — The number of characters in CSTRNG. To set this type of delimiter to the default values, set NUMB to -1. To erase all delimiters from a type (so FINDLM will not search for this type), set NUMB to zero (0).

ITBL — Input/Output — The integer array used by FINDLM to store delimiter information.

Remarks:

SETDLM usually does not need to be called, unless delimiters other than the default are to be set. If SETDLM is called prior to FINDLM, the default delimiters will not be set, unless explicitly requested. Thus, if you desired to set type one delimiter to a comma and an equal sign only, leaving types two and three as the default, you would need to set the default delimiters for types two and three. (This could be accomplished by calling SETDLM with a type of zero (0) first.)

All delimiters will be reset to the default ones by setting ITYPE to zero. Each type of delimiter can be reset to the default values by setting NUMB to a negative one (-1). To erase all delimiters from a type (so FINDLM will not search for this type), set NUMB to zero (0).

The default delimiters are:

Type	Delimiters	Notes
1	,=/>()	
2		A blank character.
3	'"	Single or double quotes.

Examples:

To set all delimiters to the default values:
```
CALL SETDLM ( 0, ' ', IDUM, IDUM, ITBL)
```

To set type one delimiter to the default values:
```
CALL SETDLM ( 1, ' ', 1, -1, ITBL)
```

To set type one delimiter to a comma and an equal sign (only):
```
CALL SETDLM ( 1, ',=', 1, 2, ITBL)
```

To set type three delimiters to double quotes only, with others as their defaults:
```
CALL SETDLM ( 0, ' ', IDUM, IDUM, ITBL)
CALL SETDLM ( 3, '"', 1, 1, ITBL)
```

or, alternatively:
```
CALL SETDLM ( 3, '"', 1, 1, ITBL)
CALL SETDLM ( 1, ' ', 1, -1, ITBL)
CALL SETDLM ( 2, ' ', 1, -1, ITBL)
```

To remove all string delimiters (after other delimiters have been set):
```
CALL SETDLM ( 3, ' ', 1, 0, ITBL)
```

5.15 LISNUM – Determine if a Character String Contains a Number

Purpose:

LISNUM is a logical function that determines if a character string contains a number, or contains alpha characters.

Calling Sequence:

LNUMB = LISNUM (CSTRNG)

Declarations:

```
CHARACTER CSTRNG*(*)
LOGICAL LISNUM
```

Argument Description:

CSTRNG	Input	The character string to test.
LISNUM	Output	A logical flag that is returned .TRUE. if only numerical characters are found (-+.0123456789), or .FALSE. if some other characters are found. If any non-numeric characters are found, LISNUM is returned as .FALSE.

5.16 INTGR – Read an Integer Number from a Character String

Purpose:

Function INTGR converts a number in a character string into an integer number.

Calling Sequence:

NUMBER = INTGR (CSTR, NBEG, NLEN, IERR)

Declarations:

```
CHARACTER CSTR
INTEGER INTGR, NBEG, NLEN, IERR
```

Argument Description:

CSTR	Input	The character string containing the number to be read.
NBEG	Input	The beginning position in CSTR of the number to convert. This may include leading blanks.
NLEN	Input	The number of characters in CSTR to read for the number. This should not include trailing blanks.
IERR	Output	A status parameter indicating the successfulness of the conversion. If the number was converted correctly, IERR is returned as zero (0). If the string specified contained an illegal character, or some other error occurred, IERR is returned as -1.
INTGR	Output	The integer value of the converted string. INTGR is set to -1 if an error occurred.

Remarks:

If the length of the string is known, the number may be read directly using a FORTRAN READ. If the length of the string may vary from call to call, INTGR will create the proper format to read the number.

5.17 INTGRC – Write an Integer Number to a Character String

Purpose:

INTGRC writes an integer number into a character string.

Calling Sequence:

CALL INTGRC (NUMBER, CSTR, NBEG, NLEN)

Declarations:

```
CHARACTER CSTR
INTEGER NUMBER, NBEG, NLEN
```

Argument Description:

NUMBER	Input	The integer number to be written to the character string.
CSTR	Output	The character string to contain the integer number. The results will be right justified and blank filled. If the number overflows the space provided, the field will be set to asterisks (*).
NBEG	Input	The beginning position in CSTR in which to place the converted number.
NLEN	Input	The number of characters in CSTR available to write the number.

Remarks:

Generally, a FORTRAN write statement may be used instead of INTGRC. INTGRC may be used when the size of the number may vary considerably, since INTGRC forms a format based upon the size of the number.

5.18 XREAL – Convert a Real Number from a Character String

Purpose:

Function XREAL converts a number in a character string. A typical use for this occurs when data is read using a character format and then must be converted to a real number.

Calling Sequence:

XNUMB = XREAL (CSTR, NBEG, NLEN, IERR)

Declarations:

```
CHARACTER CSTR
INTEGER NBEG, NLEN, IERR
REAL XREAL
```

Argument Description:

CSTR	Input	The character string containing the number to be converted.
NBEG	Input	The beginning position in CSTR of the number to convert. This may include leading blanks.
NLEN	Input	The number of characters in CSTR to convert.
IERR	Output	A status parameter indicating the successfulness of the conversion. If the number was read correctly, IERR is returned as zero (0). If the string specified contained an illegal character, or some other error occurred, IERR is returned as -1.
XREAL	Output	The real number of the converted string. XREAL is set to -1.0 if an error occurred.

Remarks:

Exponential numbers may be converted with XREAL. The same rules apply to XREAL as to the 'F' descriptor in the FORTRAN format statement.

If the length of the string is known, the number may be read directly using a FORTRAN READ. If the length of the string may vary from call to call, XREAL will create the proper format to read the number.

5.19 XREALC – Convert a Real Number to a Character String

Purpose:

XREALC converts a real number to a character string.

Calling Sequence:

CALL XREALC (XNUMB, CSTR, NBEG, NLEN, NDEC)

Declarations:

```
CHARACTER CSTR
INTEGER NBEG, NLEN, NDEC
REAL XNUMB
```

Argument Description:

XNUMB	Input	The real number to be converted into character form.
CSTR	Output	The character string to contain the number. The results will be right justified and blank filled. If the number overflows the space provided, the field will be set to asterisks (*).
NBEG	Input	The beginning position in CSTR in which to place the number.
NLEN	Input	The number of characters in CSTR to write the number.
NDEC	Input	The number of digits after the decimal place to write.

Remarks:

NLEN and NDEC together form an equivalent 'F' format descriptor, in the form 'Fnlen.dec'. For example, if NLEN is ten and NDEC is three, the equivalent 'F' descriptor would be F10.3.

5.20 LJSTR – Left Justify a Character String

Purpose:

Subroutine LJSTR takes a character string and shifts it so that the string (the non-blank characters) is left justified.

Calling Sequence:

CALL LJSTR (CSTR1, NBEG1, NLEN1, CSTR2, NBEG2)

Declarations:

```
INTEGER NBEG1, NLEN1, NBEG2
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to be left justified.
NBEG1	Input	The position in CSTR1 defining the beginning of the string.
NLEN1	Input	The length of CSTR1, relative to NBEG1.
CSTR2	Output	The character variable to contain the left justified string. CSTR2 may be variable CSTR1.
NBEG2	Input	The beginning position in CSTR2 in which to place the left justified string.

Remarks:

CSTR1 and CSTR2 may be the same arguments.

Examples:

```
CALL LJSTR (CSTRNG, 1, 80, CSTRNG, 1)
CALL LJSTR ('      xyz                ', 1, 30, CSTRNG, 1)
```

5.21 RJSTR – Right Justify a Character String

Purpose:

Subroutine RJSTR takes a character string and shifts it so that the string (the non-blank characters) is right justified.

Calling Sequence:

CALL RJSTR (CSTR1, NBEG1, NLEN1, CSTR2, NBEG2)

Declarations:

```
INTEGER NBEG1, NLEN1, NBEG2
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to be right justified.
NBEG1	Input	The position in CSTR1 defining the beginning of the string.
NLEN1	Input	The length of CSTR1, relative to NBEG1.
CSTR2	Output	The character variable to contain the right justified string. CSTR2 may be variable CSTR1.
NBEG2	Input	The beginning position in CSTR2 in which to place the right justified string.

Remarks:

CSTR1 and CSTR2 may be the same arguments.

Examples:

```
CALL RJSTR (CSTRNG, 1, 80, CSTRNG, 1)
CALL RJSTR ('      xyz                 ', 1, 30, CSTRNG, 1)
```

5.22 CJSTR – Center Justify a Character String

Purpose:

Subroutine CJSTR takes a character string and shifts it so that the string (the non-blank characters) is in the center.

Calling Sequence:

CALL CJSTR (CSTR1, NBEG1, NLEN1, CSTR2, NBEG2)

Declarations:

```
INTEGER NBEG1, NLEN1, NBEG2
CHARACTER CSTR1, CSTR2
```

Argument Description:

CSTR1	Input	The character string to be centered.
NBEG1	Input	The position in CSTR1 defining the beginning of the string.
NLEN1	Input	The length of CSTR1, relative to NBEG1.
CSTR2	Output	The character variable to contain the centered string. CSTR2 may be variable CSTR1.
NBEG2	Input	The beginning position in CSTR2 in which to place the centered string.

Remarks:

CSTR1 and CSTR2 may be the same arguments.

Examples:

```
CALL CJSTR (CSTRNG, 1, 80, CSTRNG, 1)

CALL CJSTR ('   Title                ', 1, 30, CSTRNG, 1)

CALL CHRBLK (CSTRNG)
CSTRNG(1:) = 'APPENDIX A'
CALL CJSTR (CSTRNG, 1, 100, CSTRNG, 1)
```

5.23 CHRHOL – Convert a Character String to Hollerith (on Byte Boundaries)

Purpose:

CHRHOL converts a character string to Hollerith (an integer array). This is necessary where both alphanumeric and integer or real data must be stored in the same array. CHRHOL converts on byte boundaries. A similar routine, CH2HOL, is faster but converts complete machine words.

Calling Sequence:

```
CALL CHRHOL (CSTR, IBEG, ILEN, IHOL, NBEG)
```

Declarations:

```
CHARACTER CSTR
INTEGER IHOL(*), IBEG, ILEN, NBEG
```

Argument Description:

CSTR	Input	The character string to be converted into Hollerith.
IBEG	Input	The beginning position in CSTR defining where to start converting.
ILEN	Input	The number of characters in CSTR to convert.
IHOL	Output	An integer array to contain the characters in Hollerith form.
NBEG	Input	The beginning byte position in IHOL in which to place the converted characters.

Remarks:

The bytes in IHOL that are not replaced by the converted characters are unaltered.

5.24 HOLCHR – Convert a Hollerith Array to Character (on Byte Boundaries)

Purpose:

HOLCHR converts an integer array containing Hollerith characters to a character variable. This is necessary where both alphanumeric and integer or real data are stored in the same array. HOLCHR operates on byte boundaries. A similar routine, HOL2CH, is faster but converts complete machine words.

Calling Sequence:

```
CALL HOLCHR (IHOL, IBEG, ILEN, CSTR, NBEG)
```

Declarations:

```
CHARACTER CSTR
INTEGER IHOL(*), IBEG, ILEN, NBEG
```

Argument Description:

IHOL	Input	An integer array containing the characters in Hollerith form.
IBEG	Input	The beginning byte position in IHOL defining where to start converting.
ILEN	Input	The number of bytes in IHOL, from IBEG, to convert.
CSTR	Output	The character variable to contain the converted characters.
NBEG	Input	The beginning character position in CSTR in which to place the converted characters.

5.25 CH2HOL – Convert a Character String to Hollerith (on Word Boundaries)

Purpose:

CH2HOL converts a character string to Hollerith (an integer array) on word boundaries. This is necessary where both alphanumeric and integer or real data must be stored in the same array. A similar routine, CHRHOL, operates on byte boundaries but is slower.

Calling Sequence:

```
CALL CH2HOL (CSTR, IHOL, NWORDS)
```

Declarations:

```
CHARACTER CSTR
INTEGER IHOL(*), NWORDS
```

Argument Description:

CSTR	Input	The character string to be converted into Hollerith.
IHOL	Output	An integer array to contain the characters in Hollerith form.
NWORDS	Input	The number of words to convert from character into Hollerith. Complete words are converted.

5.26 HOL2CH – Convert a Hollerith Array to Character (on Word Boundaries)

Purpose:

HOL2CH converts an integer array containing Hollerith characters to a character variable on word boundaries. This is necessary where both alphanumeric and integer or real data are stored in the same array. HOLCHR operates on byte boundaries. A similar routine, HOLCHR, operates on byte boundaries but is slower.

Calling Sequence:

```
CALL HOL2CH (IHOL, CSTR, NWORDS)
```

Declarations:

```
CHARACTER CSTR
INTEGER IHOL(*), NWORDS
```

Argument Description:

IHOL	Input	The integer array containing the Hollerith characters to be converted.
CSTR	Output	The character variable to contain the converted characters
NWORDS	Input	The number of words to convert from Hollerith into character. Complete words are converted.

6 PREAD Subroutines

The PREAD preprocessor subroutines provide a means of enhancing the user friendliness of an interactive program. PREAD will operate in either an interactive or batch environment. A complete description of PREAD and the use of PREAD may be found in the Water Control Software Implementation and Management Guide. This section describes how to add the PREAD software to a program.

There are two ways in which to call PREAD. In the first method (the preferred method) the input unit number is passed to subroutine PREADC, which in turn returns the line read in a character variable. For the second method, PREAD is called prior to each FORTRAN READ. In this method the unit number is passed to PREAD in a variable, and this variable is used for the unit number in the FORTRAN READ. The second method is provided for compatibility of older programs.

The steps to add PREAD to a program are as follows:

1. Add calls to PTTACH. Calls to PTTACH should be made at the beginning of the program, similar to the example following.
2. Add calls to PREAD:

   Method 1

   a. Replace every READ (from the standard input) with a call to PREADC. PREADC will return the line read in character variable.

   Method 2

   a. Set an integer variable equal to the standard input unit.
   b. Call PREAD with this variable just before each FORTRAN READ.
   c. Use this variable for the unit number in each FORTRAN READ.
3. Call PEND at the end of the program to close the PREAD files.

If PREAD menus will never be accessed by the program, dummy menu subroutines may be loaded in order to reduce the program size and prevent references to graphics subroutines. This is accomplished by either loading the file PMDUM during linking, or compiling file PMDUMS along with the program. Both of these files should be located in the same area as the library. Also, do not call PTTACH with the menu file keyword (MENFILE). If menus will be accessed, the program must link in the Tektronix graphics library "AG2LIB".

Similarly, if PREAD screens will never be accessed by the program, dummy screen subroutines may be loaded from file PSDUM or compiled from file PSDUMS. Do not call PTTACH with the screen file keyword (SCNFILE).

Example:

```
      CHARACTER CNAME*64
C
      CALL ATTACH( 5, 'INPUT', 'STDIN', ' ', CNAME, ISTAT )
      CALL ATTACH( 6, 'OUTPUT', 'STDOUT', ' ', CNAME, ISTAT )
C
```

```
CALL PTTACH( 30, 'SCRATCH', 'SCRATCH1', ' ', CNAME, ISTAT )
CALL PTTACH( 31, 'FUNFILE', 'GENFUN', ' ', CNAME, ISTAT )
CALL PTTACH( 32, 'MACFILE', 'GENMAC', ' ', CNAME, ISTAT )
CALL PTTACH( 33, 'MENFILE', 'GENMEN', ' ', CNAME, ISTAT )
CALL PTTACH( 34, 'SCNFILE', 'GENSCN', ' ', CNAME, ISTAT )
CALL PTTACH( 35, 'LOGFILE', 'PGLOG',  ' ', CNAME, ISTAT )
CALL ATTEND
```

Method 1

```
      CALL PREADC (5, CLINE, ISTAT, *800)
      ...

800   CONTINUE

      CALL PREADC (5, CLINE, ISTAT, *800)
      READ (CLINE,40) X, Y, Z
      ...

800   CONTINUE
```

Method 2

```
      INPUT = 5
      ...

      CALL PREAD (INPUT)
      READ (INPUT,10,END=800) CLINE
      ...

      CALL PREAD (INPUT)
      READ (INPUT,40,END=700) X, Y, Z
      ...

      CALL PEND
      CLOSE (UNIT=5)
      CLOSE (UNIT=6)
      STOP
```

6.1 PTTACH – Attach PREAD Files

Purpose:

PTTACH is used to attach files accessed by PREAD (e.g., the macro file, function file, etc.). PTTACH has the same arguments as subroutine ATTACH, except that the files are not opened or accessed until a reference is made to them. (For example, the macro file is not opened, or created, until a !RUN or similar command is issued.)

All the files to be referenced by PREAD must have an associated PTTACH call. PREAD files not specified in a PTTACH call will have that capability disabled. For example, if a macro file is not provided, the macro capability will not be enabled. The PREAD scratch file must be specified in a PTTACH call (all other files are optional).

The subroutine ATTEND should be called after the last call to PTTACH or ATTACH. See the ATTACH subroutine documentation for further information.

Calling Sequence:

```
CALL PTTACH (IUNIT, CKEYWD, CDEFLT, CDUMMY, CNAME, IOSTAT)
```

Declarations:

```
INTEGER IUNIT, IOSTAT
CHARACTER CKEYWD, CDEFLT, CDUMMY, CNAME
```

Argument Description:

IUNIT	Input	The unit number to be associated with that file.
CKEYWD	Input	The keyword that identifies the file to be accessed. The valid keywords are: 'SCRATCH' 'FUNFILE' 'MACFILE' 'MENFILE' 'LOGFILE' 'SCNFILE'
CDEFLT	Input	The default file to access, if the user does not enter a file name on the execution line for this keyword.
CDUMMY	Input	A dummy character argument. This may be a blank character (' ').

CNAME	Output	CNAME is returned with the name of the file specified on the execution line, or the default name if none was specified. CNAME must be declared long enough to hold the longest name that might be used.
IOSTAT	Output	A status parameter indicating the successfulness of the call. Because the files are not opened until accessed, this argument is returned with zero unless the program was executed with a question mark on the execution line (see the ATTACH status codes).

Remarks:

PTTACH calls ATTACH with a CONTRL of 'NOP', and then remembers the file name. To disable a PREAD capability, do not call PTTACH with the associated keyword.

A PREAD scratch file must always be specified. This file can be any blocked scratch file (see the ATTACH documentation for a valid list).

Example:

```
      CHARACTER CNAME*64
C
      CALL ATTACH( 5, 'INPUT', 'STDIN', ' ', CNAME, ISTAT )
      CALL ATTACH( 6, 'OUTPUT', 'STDOUT', ' ', CNAME, ISTAT )
C
      CALL PTTACH( 30, 'SCRATCH', 'SCRATCH1',  ' ', CNAME, ISTAT )
      CALL PTTACH( 31, 'FUNFILE', 'GENFUN', ' ', CNAME, ISTAT )
      CALL PTTACH( 32, 'MACFILE', 'GENMAC', ' ', CNAME, ISTAT )
      CALL PTTACH( 33, 'MENFILE', 'GENMEN', ' ', CNAME, ISTAT )
      CALL PTTACH( 34, 'SCNFILE', 'GENSCN',  ' ', CNAME, ISTAT )
      CALL PTTACH( 35, 'LOGFILE', 'PGLOG',  ' ', CNAME, ISTAT )
      CALL ATTEND
```

6.2 PEND – Close PREAD Files

Purpose:

PEND closes all PREAD files accessed. PEND should be called at the end of a program, along with any other CLOSE statements.

Calling Sequence:

CALL PEND

6.3 PREADC – PREAD Processor (Method 1)

Purpose:

PREADC preprocess lines read from the terminal (or other input). PREADC returns a character variable containing the line read, and has an alternative return for End-Of-File conditions.

Calling Sequence:

CALL PREADC (IUNIT, CLINE, ISTAT, *EOF-statement)

Declarations:

```
CHARACTER CLINE
INTEGER IUNIT, ISTAT
```

Argument Description:

IUNIT	Input	The unit number attached to the standard input.
CLINE	Output	The expanded line read from IUNIT. CLINE is blanked by PREAD prior to reading.
ISTAT	Output	A status parameter. If ISTAT is returned as zero or positive, it reflects the number of characters read (in CLINE). If PREAD detected an end-of-file condition, ISTAT is returned as -1. If the declared length of CLINE is less then the length of the expanded line, ISTAT is returned as -2 (and the line is truncated).
*EOF-statement	Input	The statement number to jump to if an end-of-file condition was met (an alternative return). This is the same as an "END=" parameter in a FORTRAN READ.

Example:

```
      CHARACTER CLINE*132
C
      CALL ATTACH( 5, 'INPUT', 'STDIN', ' ', CNAME, ISTAT )
      …
C
      CALL PREADC (5, CLINE, ISTAT, *800)
      READ (CLINE,20) X, Y, Z
      …
C
```

```
C         EOF DETECTED
      800 CONTINUE
          …
```

6.4 PREAD – PREAD Processor (Method 2)

Purpose:

PREAD is the main subroutine that preprocesses lines read from the terminal or other input. PREAD should be called prior to each FORTRAN READ from the standard input.

Calling Sequence:

CALL PREAD (IUNIT)

Declarations:

INTEGER IUNIT

Argument Description:

IUNIT	Input/ Output	A variable containing the unit number attached to the standard input. This must be a variable, and the same variable for all calls to PREAD (if PREAD is called in another subroutine, that variable must be passed to that subroutine). The unit number should be set in a DATA or similar statement (only once).

Remarks:

PREAD should be called just before each READ from the standard input. The unit variable should be used for the unit number in the FORTRAN READ following the call to PREAD (do not use a literal value).

Example:

```
DATA INPUT /5/
...

CALL PREAD (INPUT)
READ (INPUT,10,END=900) CLINE
..
```

6.5 PREAD1 – Execute a PREAD Command from the Program

Purpose:

PREAD1 will execute a PREAD command directly from the calling program. PREAD1 is not intended to be called prior to a READ statement.

Calling Sequence:

CALL PREAD1 (CLINE)

Declarations:

CHARACTER CLINE

Argument Description:

CLINE	Input	CLINE (Input) The PREAD command to execute. This should be the complete command, including the PREAD command character (!).

Example:

```
CALL PREAD1 ('!RUN MAC1')
CALL PREAD1 ('!TEACH & /SOUTH BEND/FLOW/')
CALL PREAD1 ('!/SS')
```

6.6 PSET – Set PREAD Parameters

Purpose:

PSET is used to set PREAD parameters. This currently includes the prompt string, input echo, and the logging capability.

Calling Sequence:

CALL PSET (CFLAG, CPARM, NPARM)

Declarations:

```
CHARACTER CFLAG*4, CPARM
INTEGER NPARM
```

Argument Description:

CFLAG	Input/	A flag, indicating which parameter to set. A list of the valid flags follows.
CPARM	Input	The character to set the parameter to.
NPARM	Input	The integer number to set the parameter to.

Valid Parameters:

CFLAG	CPARM	NPARM	Description
'PROM'	prompt	# chars	Sets the input prompt to CPARM, with NPARM characters long.
'ECHO'	'ON' or 'OFF'	-	Turns the echo on or off.
'LOGF'	'ON' or 'OFF'	-	Turns the log on or off.
'LOGN'	-	unit	Changes the log file unit number.

6.7 PINQIR – Inquire About PREAD Parameters

Purpose:

PINQIR returns the current setting of several PREAD parameters.

Calling Sequence:

```
CALL PINQIR (CFLAG, CPARM, NPARM)
```

Declarations:

```
CHARACTER CFLAG*4, CPARM
INTEGER NPARM
```

Argument Description:

CFLAG	Input	A flag, indicating which parameter to inquire about. A list of the valid flags follows.
CPARM	Output	A character variable containing the setting of the parameter.
NPARM	Output	An integer number containing the setting of the parameter.

Valid Parameters:

CFLAG	CPARM	NPARM	Description
'PROM'	prompt	nchs	Returns the prompt in CPARM, and its length in NPARM.
'ECHO'	'ON' or 'OFF'	-	Indicates if the echo is on or off.
'LOGF'	'ON' or 'OFF'	-	Indicates if the log is on or off.
'LOGN'	-	unit	Returns the log file unit number.
'TERM'	term type	-	Returns the port (terminal) type (i.e., 'ASY', 'TTY', or 'CRT').
'FUNC'	'ON' or 'OFF'	-	Indicates if the function mode is on or off.
'MACR'	'ON' or 'OFF'	-	Indicates if a macro is currently running.
'MENU'	'ON' or 'OFF'	-	Indicates if input is from the menu.
'LEAR'	'ON' or 'OFF'	-	Indicates if the learn is on or off.
'SCRE'	'ON' or 'OFF'	-	Indicates if input is from screens.

6.8 PSETFN – Set PREAD Function

Purpose:

PSETFN sets a character to a function in the PREAD function file.

Calling Sequence:

CALL PSETFN (CKEY, CFUN, NFUN)

Declarations:

```
CHARACTER CKEY*1, CFUN
INTEGER NFUN 2
```

Argument Description:

CKEY	Input	The single character to set as the function character.
CFUN	Input	The character string to set as the function.
NFUN	Input	The number of characters in CFUN.

6.9 PFNKEY – Get the String Assigned to a Function Key

Purpose:

PFNKEY returns an expanded function string, given the function character.

Calling Sequence:

CALL PFNKEY (CKEY, CFUN, NFUN)

Declarations:

```
CHARACTER CKEY*1, CFUN
INTEGER NFUN
```

Argument Description:

CKEY	Input	The single function character of which to get the expanded string.
CFUN	Output	The expanded function character string.
NFUN	Output	The number of characters in CFUN.

7 Miscellaneous Subroutines

The following chapter describes general purpose miscellaneous subroutines. This includes a set of subroutines that test real numbers within a specified tolerance (accounting for real number round-off errors), bit manipulation subroutines, and name-list subroutines (to get a desired name from several synonyms.

This section also includes a variety of subroutines that are specific to either HARRIS computers or MS-DOS microcomputers.

7.1 LEQNER – Test for One Number Nearly Equal to Another

Purpose:

LEQNER is a logical function that tests two real numbers within a specified tolerance to determine if they are nearly equal to each other. LEQNER was designed to account for possible round-off errors of real numbers.

Calling Sequence:

LTEST = LEQNER (X, Y, TOL)

Declarations:

```
LOGICAL LEQNER
REAL X, Y, TOL
```

Argument Description:

X	Input	The number to compare against Y.
Y	Input	The number to be compared to X.
TOL	Input	The tolerance to check the numbers with. TOL is usually a small number, such as 0.0001.
LEQNER	Output	A logical flag that is returned .TRUE. if X is equal to Y within the tolerance specified.

Example:

Instead of:

```
IF (X.EQ.Y) THEN
```

use:

```
IF (LEQNER(X,Y,0.0001)) THEN
```

Make sure that LEQNER is declared as a logical variable. LEQNER works for both positive and negative numbers.

7.2 LGENER – Test for One Number Greater Than or Nearly Equal to Another

Purpose:

LGENER is a logical function that determines if one number is greater than or nearly equal to another within a specified tolerance. LGENER was designed to account for possible round-off errors of real numbers.

Calling Sequence:

LTEST = LGENER (X, Y, TOL)

Declarations:

```
LOGICAL LGENER
REAL X, Y, TOL
```

Argument Description:

X	Input	The number to test if it is greater than or nearly equal to Y.
Y	Input	The number to be compared against X.
TOL	Input	The tolerance to check the numbers with. TOL is usually a small number, such as 0.0001.
LGENER	Output	A logical flag that is returned .TRUE. if X is greater than or equal to Y within the tolerance specified.

Example:

Instead of:

```
IF (X.EQ.Y) THEN
```

use:

```
IF (LGENER(X,Y,0.0001)) THEN
```

Make sure that LGENER is declared as a logical variable. LGENER works for both positive and negative numbers.

7.3 LGTNER – Test for One Number Greater Than Another With a Tolerance

Purpose:

LGTNER is a logical function that determines if one number is greater than another, within a specified tolerance. LGTNER was designed to account for possible round-off errors of real numbers.

Calling Sequence:

LTEST = LGTNER (X, Y, TOL)

Declarations:

```
LOGICAL LGTNER
REAL X, Y, TOL
```

Argument Description:

X	Input	The number to test if it is greater than Y.
Y	Input	The number to be compared against X.
TOL	Input	The tolerance to check the numbers with. TOL is usually a small number, such as 0.0001.
LGTNER	Output	A logical flag that is returned .TRUE. if X is greater than Y within the tolerance specified. This is equivalent to (X.GT.Y+TOL), except LGTNER accounts for both positive and negative numbers.

Example:

Instead of:

```
IF (X.GT.Y+0.0001) THEN
```

use:

```
IF (LGTNER(X,Y,0.0001)) THEN
```

Make sure that LGTNER is declared as a logical variable. LGTNER works for both positive and negative numbers.

7.4 LLTNER – Test for One Number Less Than Another Within a Tolerance

Purpose:

LLTNER is a logical function determines if one number is less than another, within a specified tolerance. LLTNER was designed to account for possible round-off errors of real numbers.

Calling Sequence:

LTEST = LLTNER (X, Y, TOL

Declarations:

```
LOGICAL LLTNER
REAL X, Y, TOL
```

Argument Description:

X	Input/	The number to test if it is less than Y.
Y	Input	The number to be compared against X.
TOL	Input	The tolerance to check the numbers with. TOL is usually a small number, such as 0.0001.
LLTNER	Output	A logical flag that is returned .TRUE. if X is less than Y within the tolerance specified. This is the same as (X.LT.Y-TOL), except LLTNER accounts for both positive and negative numbers.

Example:

Instead of:

```
IF (X.LT.Y-0.0001) THEN
```

use:

```
IF (LLTNER(X,Y,0.0001)) THEN
```

Make sure that LLTNER is declared as a logical variable. LLTNER works for both positive and negative numbers.

7.5 LLENER – Test for One Number Less Than or Nearly Equal to Another

Purpose:

LLENER is a logical function that determines if one number is less than another, within a specified tolerance. LLENER was designed to account for possible round-off errors of real numbers.

Calling Sequence:

LTEST = LLENER (X, Y, TOL)

Declarations:

```
LOGICAL LLENER
REAL X, Y, TOL
```

Argument Description:

X	Input	The number to test if it is less than or nearly equal to Y.
Y	Input	The number to be compared against X.
TOL	Input	The tolerance to check the numbers with. TOL is usually a small number, such as 0.0001.
LLENER	Output	A logical flag that is returned .TRUE. if X is less than or equal to Y within the tolerance specified.

Example:

Instead of:

```
IF (X.LE.Y) THEN
```

use:

```
IF (LLENER(X,Y,0.0001)) THEN
```

Make sure that LLENER is declared as a logical variable. LLENER works for both positive and negative numbers.

7.6 LBTEST – Test to Determine if a Bit is Set

Purpose:

Logical Function LBTEST is used to determine if a specified bit is set. LBTEST is the same function as the MIL-STD-1753 BTEST function.

Calling Sequence:

LTEST = LBTEST (IWORD, NBIT)

Declarations:

```
LOGICAL LBTEST
INTEGER IWORD, NBIT
```

Argument Description:

IWORD	Input	The integer word containing the bits to be tested.
NBIT	Input	The bit number to test for. NBIT may range from zero to twenty-three on HARRIS computers, zero to fifteen on MS-DOS microcomputers.
LBTEST	Output	LBTEST is returned .TRUE. if the specified bit is set, otherwise .FALSE.

Example:

```
      LOGICAL LBTEST
C
C     AFTER A CALL TO GIOP, BIT 17 IS SET IF AN ERROR OCCURRED
      CALL GIOP (..., ISTAT)
      IF (LBTEST(ISTAT,17)) GO TO 900
```

7.7 IBSET – Set a Bit

Purpose:

IBSET sets a specified bit on in an integer word.

Calling Sequence:

JWORD = IBSET (IWORD, NBIT)

Declarations:

INTEGER IBSET, IWORD, NBIT

Argument Description:

IWORD	Input	The word in which to set the bit. Note that this is an input parameter and is not changed. The result is returned in IBSET.
NBIT	Input	The bit to set.
IBSET	Output	IBSET is returned with bit NBIT set in IWORD. (The other bits remain unchanged.)

7.8 IBCLR – Clear a Bit

Purpose:

PSETFN IBCLR sets a specified bit off from an integer word.

Calling Sequence:

JWORD = IBCLR (IWORD, NBIT)

Declarations:

INTEGER IBSET, IWORD, NBIT

Argument Description:

IWORD	Input	The word in which to set the bit off. Note that this is an input parameter and is not changed. The result is returned in IBSET.
NBIT	Input	The bit to reset.
IBSET	Output	IBSET is returned with bit NBIT reset in IWORD. (The other bits remain unchanged.)

7.9 MVBITS – Move Bits from One Word into Another

Purpose:

MVBITS moves bits from one integer word into another.

Calling Sequence:

CALL MVBITS (IWORD, IPOS, NBITS, JWORD, JPOS)

Declarations:

INTEGER IWORD, IPOS, NBITS, JWORD, JPOS

Argument Description:

IWORD	Input	The word containing the bits to be moved (copied).
IPOS	Input	The beginning bit position of IWORD to move.
NBITS	Input	The number of bits to move.
JWORD	Input/ Output	The word in which to move the bits into. The bits in JWORD which are outside the specified range are unchanged.
JPOS	Input	The beginning bit position in JWORD in which to move the bits.

Remarks:

JWORD and IWORD may be the same variable. If the range specified exceeds the word boundaries, those bits are truncated (MVBITS operates only on one integer word).

Example:

If:

I = 11111111	11111111	11010111
J = 00000000	00000000	00000000

Then:

CALL MVBITS (I, 0, 8, J, 0)

J = 00000000	00000000	11010111

CALL MVBITS (I, 1, 2, J, 7)

J = 00000000	00000001	10000000

7.10 IBITS – Extract a Field of Bits

Purpose:

IBITS extracts a field of bits from an integer word.

Calling Sequence:

JWORD = IBITS (IWORD, ISTART, NBITS)

Declarations:

INTEGER IBITS, IWORD, ISTART, NBITS

Argument Description:

IWORD	Input	The integer word from which to extract the bits.
ISTART	Input	The right-most bit number of the starting position of the bits to extract.
NBITS	Input	The number of bits to extract. NBITS extends left from ISTART.
IBITS	Output	The extracted field. The result is right justified and the remaining bits set to zero.

Remarks:

IBITS extracts a subfield of NBITS bits in length from IWORD starting with bit position ISTART and extending NBITS left.

7.11 GETBIN – Get the Binary Representation of a Word

Purpose:

GETBIN takes bytes from (a) word(s), and creates a binary representation in a character array for display purposes. The binary representation of a byte is a character variable (8 characters long), with each character being either a zero (bit off), or a one (bit on). The programmer may print this character array to see which bits are on, and which bits are off.

Calling Sequence:

CALL GETBIN (IWORDS, NBYTES, CREPR)

Declarations:

```
INTEGER IWORDS(*), NBYTES
CHARACTER CREPR(NBYTES)*8
```

Argument Description:

IWORDS	Input	The integer word (or words) to get the binary representation of.
NBYTES	Input	The number of bytes to process. The first byte is the leftmost byte in IWORDS.
CREPR	Output	The binary representation of IWORDS. This must be a character array eight characters long, and dimensioned to (at least) NBYTES.

Example:

```
INTEGER IWORDS(2)
CHARACTER CREPR(4)*8
```

If:

```
IWORDS(1) = 1234567
IWORDS(2) = 1234567
CALL GETBIN (IWORDS, 4, CREPR)
```

Then:

```
CREPR(1) = '00010010'
CREPR(2) = '11010110'
CREPR(3) = '10000111'
CREPR(4) = '00010010'
```

7.12 DIBIN – Display a Number as Binary

Purpose:

DIBIN displays a number in its binary representation.

Calling Sequence:

CALL DIBIN (IUNIT, NUMBER)

Declarations:

INTEGER IUNIT, NUMBER

Argument Description:

IUNIT	Input	The The unit number to write the binary representation to.
NUMBER	Input	The number to write the binary representation of. This must be a regular integer number (on the HARRIS, three bytes long, on MS-DOS microcomputers, two bytes long).

Example:

```
NUMBER = 123456
CALL DIBIN (6, NUMBER)
```

DIBIN writes to unit 6:

```
" VALUE = 00000001  11100010  01000000"
```

7.13 NAME-LIST Processing

Purpose:

The NAME-LIST subroutines provides a means of allowing several alternative items or names to be recognized as a single item or name. An example of this might be the name of a gaging station, where different agencies may give different names (or codes) for the same station. The name-list routines will take any of these names and obtain the primary name that is to be used. For example, the USGS may refer to a station by the code '08928231', the NWS may use the code 'STBO5', and the common (or desired) name might be 'South Bend'. If any of these names are given to the subroutine, it would return 'South Bend'.

Typically, names for the NAME-LIST subroutines are read from a file that contains the substitute (or pseudo) name followed by a comma, then the desired (or true) name. Such a file might appear as follows:

```
08928231,South Bend
STBO5,South Bend
S BEND,South Bend
08928422,Crescent City
CRCO6,Crescent City
CRES,Crescent City
C.C.,Crescent City
```

Subroutine Summary:

NAMFIL - Read a File of Pseudo and True Names
NAMLST - List all the Pseudo and True Names
TRUNAM - Obtain a True Name from a Pseudo Name
SETNAM - Set or Remove a Name

7.13.1 NAMFIL – Read a File of Pseudo and True Names

Purpose:

NAMFIL creates a name list from a file containing a list of alternative (pseudo) names and desired (true) names. This list is then used by the other name-list subroutines to find true names from pseudo names. NAMFIL is usually called once, at the beginning of the program. The length of a name is defined by the calling program, but may not exceed eighty characters.

The file that NAMFIL reads should contain the pseudo name followed by a comma then the true name. An example of such a list follows.

Calling Sequence:

CALL NAMFIL (IUNIT, CNAMES, INAMES, MAXNAM, ISTAT)

Declarations:

```
PARAMETER (MAXNAM=?)
CHARACTER CNAMES(MAXNAM)*(*)
INTEGER INAMES(MAXNAM+5), ISTAT
```

Argument Description:

IUNIT	Input	The unit number of the file containing the pseudo and true names. This file must have been opened by the calling program.
CNAMES	Output	The name-list. This must be a character array, dimensioned to MAXNAM. The maximum length for any name is implied in the CHARACTER statement.
INAMES	Output	INAMES is a pointer array, which must be dimensioned to MAXNAM + 5.
MAXNAM	Input	The dimension of CNAMES. This is the maximum number of names that can be in the list (including true names).
ISTAT	Output	A status parameter indicating the successfulness of the call. The following values are possible:

ISTAT	Description
0	NAMFIL completed successfully
-1	Unrecognizable line in file
-2	Illegal name in file
-3	Reached maximum number of names (prior to the end of the file)

Remarks:

Any error messages are written to unit 6. Unit 6 should be attached to the standard output by the calling program.

See the example use of the name-list subroutines given at the end of this section.

Example:

The following is an example of a name-list input file for use by NAMFIL. Such a file would be created by a standard editor.

```
08928231,South Bend
STBO5,South Bend
S BEND,South Bend
08928422,Crescent City
CRCO6,Crescent City
CRES,Crescent City
C.C.,Crescent City

etc.
```

7.13.2 NAMLST – List All the Pseudo and True Names

Purpose:

NAMLST prints all the pseudo and true names in a name-list to unit 6.

Calling Sequence:

CALL NAMLST (CNAMES, INAMES)

Declarations:

```
CHARACTER CNAMES(MAXNAM)*(*)
INTEGER INAMES(MAXNAM+5)
```

Argument Description:

CNAMES	Input	The name-list (read in by subroutine NAMFIL).
INAMES	Input	INAMES is the pointer array, which must be dimensioned to MAXNAM+5.

7.13.3 TRUNAM – Obtain a True Name from a Pseudo Name

Purpose:

TRUNAM obtains a desired or true name from a name-list, given a alternative or pseudo name. If the input pseudo name is not found, the returned true name will contain all blanks.

Calling Sequence:

CALL TRUNAM (CPSUDO, CTRUE, CNAMES, INAMES)

Declarations:

```
CHARACTER CPSUDO, CTRUE, CNAMES(MAXNAM)
INTEGER INAMES(MAXNAM+5)
```

Argument Description:

CPSUDO	Input	The name to be matched. The length of CPSUDO should be the same as CTRUE and CNAMES.
CTRUE	Output	The desired or true name matching CPSUDO. If a match could not be found, CTRUE is returned blank filled.
CNAMES	Input	The name-list (read in by subroutine NAMFIL).
INAMES	Input	INAMES is the pointer array, which must be dimensioned to MAXNAM+5.

Remarks:

See the example at the end of the section for an example use of TRUNAM.

7.13.4 SETNAM – Set or Remove a Name in the Name List

Purpose:

SETNAM allows editing of the name-list by a program. (SETNAM edits the in-core list, not the name-list file.) A pseudo or true name can be added or removed by SETNAM.

Calling Sequence:

CALL SETNAM (CPSUDO, CTRUE, MAXNAM, CNAMES, INAMES, NNAMES, ISTAT)

Declarations:

```
CHARACTER CPSUDO, CTRUE, CNAMES(MAXNAM)
INTEGER INAMES(MAXNAM+5), NNAMES, ISTAT
```

Argument Description:

CPSUDO	Input	The pseudo (alternative) name to set or remove. See use below on how to set or remove names.
CTRUE	Input	The true (desired) name to set or remove.
MAXNAM	Input	The dimension of CNAMES. This is the maximum number of names that can be in the list (including true names).
CNAMES	Input/ Output	The name-list to be edited. This list is typically created by NAMFIL.
INAMES	Input/ Output	A pointer array for the name-list. This array must be dimensioned to MAXNAM+5.
NNAMES	Output	The current number of names in the name-list (including true names).
ISTAT	Output	A status parameter indicating the successfulness of the call. The following values are possible:

ISTAT	Description
0	SETNAM completed successfully
1	Could not find true name in list
2	Could not find pseudo name in list
3	Name given is already in list
4	Reached maximum number of names

Use:

The name-list is edited by:

(1) Set a new true name in the list: Set CPSUDO equal to CTRUE.
(2) Set a new pseudo name in the name list: Set CPSUDO equal to the new pseudo name and CTRUE equal to the true name corresponding to it. The true name must have already been entered.
(3) Remove a pseudo name from the name list: Set CPSUDO equal to the pseudo name, and CTRUE blank filled.
(4) Remove a true name and all the associated pseudo names: Blank fill CPSUDO, and set CTRUE to the true name to remove.

Remarks:

Typically, editing of the list is not done by a program. However, SETNAM does provide this capability if it is desired.

Example Use o f Name-List Subroutines:

```
C    Allow a maximum of 200 names in the name list, with
C    a maximum of 32 characters in each name.
     PARAMETER (MAXNAM=200)
     CHARACTER CNAMES(MAXNAM)*32, CPSUDO*32, CLOC*32
     INTEGER INAMES(MAXNAM+5)

     ...

C    Open the name file
     OPEN (UNIT=10, FILE='NAMLST', IOSTAT=IERR)
     IF (IERR.NE.0) GO TO 900
C
C    Read in the name list
     CALL NAMFIL (10, CNAMES, INAMES, MAXNAM, ISTAT)
     CLOSE (UNIT=10)
     IF (ISTAT.NE.0) GO TO 900

     ...

10   CONTINUE
     WRITE (6,*)'Enter location name'
READ (5,20,END=920) CPSUDO
20   FORMAT (A)
C    Get the desired name, if the user entered a alternative one
     CALL TRUNAM (CPSUDO, CLOC, CNAMES, INAMES)
C    See if a valid name was entered
     IF (CLOC(1:3).EQ.'   ') THEN
```

```
      WRITE (6,*)'Unknown location - reenter location name'
      GO TO 10
      ENDIF
C
C     Process name ...
```

7.14 ABORT – Issue a Program Abort

Purpose:

Subroutine ABORT initiates an abort procedure (which causes certain error processes to occur), then stops the program. On HARRIS computers, this includes printing the program address when the abort occurred, and setting certain error registers (this will cause a batch job to terminate). If walkback is set, the program will print the location and subroutines called to this location. ABORT should be called only when a significant error occurs.

Calling Sequence:

```
CALL ABORT
```

Remarks:

On HARRIS computers, a special abort instruction is issued. On non-HARRIS computers, an error message is printed, then an illegal instruction is attempted (the square root of a negative number).

7.15 IEB2AS – Convert EBCDIC to ASCII

Purpose:

Subroutine IEB2AS converts an EBCDIC character decimal representation to ASCII for IBM mainframes and similar computers. This is need only where the decimal representation of characters on EBCDIC are used. IEB2AS operates on single characters represented as integer values. On ASCII computers, the character representation is returned unaltered.

Calling Sequence:

```
CALL IEB2AS (ICH)
```

Declarations:

```
INTEGER ICH
```

Argument Description:

ICH	Input/ Output	The EBCIDIC decimal representation of the character to be converted. ICH is returned with the ASCII representation of that character.

Example:

```
C    COUNT THE NUMBER OF EACH CHARACTER IN A FILE
C    THIS WILL WORK ON AN ASCII OR EBCIDIC COMPUTER
C
     INTEGER ICOUNT(128)
C
5    READ (5, 10, END=100) CLINE
10   FORMAT (A80)
     DO 20 I=1,80
        ICH = ICHAR(CLINE(I:I))
        CALL IEB2AS(ICH)
        ICOUNT(ICH) = ICOUNT(ICH) + 1
20   CONTINUE

     GO TO 5
```

7.16 HARRIS Specific Subroutines

7.16.1 LPOPT – Get Program Options

Purpose:

LPOPT is a logical function which indicates if a single character program option has been set from the execution line for HARRIS computers. An example of program options for the compiler is: "SAUF77.IL". LPOPT would be used to determine whether the "I" and the "L" options have been set.

Calling Sequence:

```
LTEST = LPOPT (C)
```

Declarations:

```
CHARACTER C*1
LOGICAL LPOPT
```

Argument Description:

C	Input	A single character letter to test if this option has been set.
LPOPT	Output	Returns .TRUE. if that letter option has been set, otherwise LPOPT is returned as .FALSE..

Remarks:

HARRIS options are valid for the letters A through X only. LPOPT must be declared as logical in the calling routine.

Example:

If the letter 'D' is chosen to indicate a debug run, then an execution of:

```
MYPROG.D
```

will cause LPOPT to return the following:

```
IF (LPOPT('D')) THEN  ...  (LPOPT returns .TRUE.)
IF (LPOPT('F')) THEN  ...  (LPOPT returns .FALSE.)
```

7.16.2 CIJOBE – Initiate a Batch Job

Purpose:

CIJOBE is the same subroutine as the HARRIS IJOBE subroutine, except that the job file name and optional password are specified as a character string instead of Hollerith arrays.

Calling Sequence:

```
CALL CIJOBE (CNAME, CPASS, IERR)
```

Declarations:

```
CHARACTER CNAME, CPASS*6
INTEGER IERR
```

Argument Description:

CNAME	Input	A character string containing the name of the job file to be initiated.
CPASS	Input	If a password is required to initiate the job, this character string must contain that password, and otherwise it should be blank.
IERR	Output	A status parameter indicating the successfulness of the initiation. If IERR is returned as zero, the job was successfully initiated.

Remarks:

Converts the file name and password to Hollerith, then calls the HARRIS IJOBE subroutine. See the IJOBE subroutine documentation in the HARRIS FORTRAN manual for more information.

7.16.3 CSPOOL – Spool a File to a Physical Device

Purpose:

CSPOOL is the same subroutine as the HARRIS SPOOL subroutine, except that the file name is specified as a character string instead of a Hollerith array.

Calling Sequence:

CALL CSPOOL (CNAME, IPDN, IERR)

Declarations:

```
CHARACTER CNAME
INTEGER IPDN, IERR
```

Argument Description:

CNAME	Input	A character string containing the name of the file to spool.
IPDN	Input	The physical device number to spool the file to. If this is the system printer, IPDN should be set to six.
IERR	Output	A status parameter indicating the successfulness of the call. If IERR is returned as zero, the file was spooled successfully.

Remarks:

Converts the file name to Hollerith, and then calls the HARRIS SPOOL subroutine. See the SPOOL subroutine documentation in the HARRIS FORTRAN manual for more information.

7.16.4 COPCOM – Execute an OPCOM Command

Purpose:

COPCOM is the same subroutine as the HARRIS OPCOM subroutine, except that the command is specified as a character string instead of a Hollerith array. OPCOM commands are those system commands that can be executed from a terminal (e.g., /SS), but must not be preceded by the slash (/).

Calling Sequence:

```
CALL COPCOM (COMAND, IERR)
```

Declarations:

```
CHARACTER COMAND
INTEGER IERR
```

Argument Description:

COMAND	Input	A character string containing the OPCOM command to execute.
IERR	Output	A status parameter indicating the successfulness of the execution. If IERR is returned as zero, the command was successfully executed.

Remarks:

Converts the command to Hollerith, and then calls the HARRIS OPCOM subroutine. See the OPCOM subroutine documentation in the HARRIS FORTRAN manual for more information.

7.16.5 CNTRLX – Interrupt a Program by Pressing CTRL X

Purpose:

CNTRLX provides a means for a user to interrupt the execution of a program by pressing a **CTRL X**. When this key is pressed, control of the program will jump to a predefined location. This is often used to allow a quick exit from a task that is displaying a substantial amount of data.

Calling Sequence:

CALL CNTRLX ($statement)

Declarations:

The argument may either be a literal or variable, depending on its use.

Argument Description:

$statement

Normal Use:

(1) Input — The statement number (preceded by a dollar sign) of where to jump to when a **CTRL X** is pressed. For example, if the program should go to statement 100 (100 CONTINUE), then this argument would be $100. The statement must be in the same subroutine where CNTRLX is called.

Other Use:

(2) Input — To disable the control-x key (after it has been set), pass zero to the subroutine (i.e., CALL CNTRLX (0)).

(3) Input/Output — To determine the address that control will be passed to, set an integer variable to -1, then pass that variable. The variable will be returned with the address that would be jumped to. Be sure that a variable is passed, not a literal -1. For example:

```
IVAR = -1
CALL CNTRLX (IVAR)
```

(4) Input — If an argument of -2 is passed to CNTRLX (either a literal or a variable), the program will jump to the previously specified location, just as if a **CTRL X** had been entered at the keyboard.

7.16.6 CRTN – Contingency (Error) Return

Purpose:

CRTN is a subroutine that provides a means of performing error processing after a program abort has occurred. With CRTN enabled, a specified subroutine will be called when an abort occurs. This subroutine may do processing such as closing files, writing an error message to a log file, etc.

Calling Sequence:

CALL CRTN (subroutine-name, IENABL)

Declarations:

```
EXTERNAL subroutine-name
INTEGER IENABL
```

Argument Description:

subroutine-name	Input	The name of the subroutine to call when an abort occurs. This name must be a literal and must be declared as EXTERNAL in the subroutine that calls it. See the Example section.
IENABL	Input	A argument that enables or disables the error return. Set IENABL to one to enable the error return, zero to remove the subroutine from error processing.

Remarks:

Several subroutines can be added to the contingency return list. They are executed in a last in, first out order. The subroutine(s) must execute a normal return (do not allow the subroutine to STOP). A second abort will cause the program to bypass any remaining contingency returns.

Example:

```
EXTERNAL ABTPRO
CALL CRTN (ABTPRO, 1)
 . . .

SUBROUTINE ABTPRO
. . .
RETURN
END
```

7.16.7 RSCPDN – Resource a Physical Device

Purpose:

RSCPDN resources a physical device (e.g., a terminal), based upon its PDN. RSCPDN may be called either from a interactive program, a real-time program, or a batch job, but different parameters must be passed for different program types. After a PDN has been resourced and opened, I-O can take place with that PDN.

Calling Sequence:

CALL RSCPDN (IUNIT, IPDN, IFUN, ISTAT)

Declarations:

INTEGER IUNIT, IPDN, IFUN, ISTAT

Argument Description:

IUNIT	Input	The unit number to attach to the PDN.
IPDN	Input	The physical device number to resource.
IFUN	Input	A number, indicating the function to perform (based on the terminal type): 1 Request resource for interactive programs. This function will not wait for the resource to occur. 2 Test for resource allocation. Returns value in ISTAT indicating if the allocation has been made or not. 3 Wait for allocation. RSCPDN will not return until the PDN has been resourced (could be a long time!). 5 Request resource for real-time programs. 6 Request resource for control point programs.
ISTAT	Output	A status parameter indicating the successfulness of the call. ISTAT is returned as zero if the call was successful, nonzero if the call failed.

Remarks:

Resourcing a physical device is a two step process. First, a request for the device is made. Then a test for resource allocation is made after waiting for an appropriate amount of time. Because a wait for allocation will cause a wait until that device is available, usually only the test for resource allocation is made. Because the resource takes some of time, a program

should pause between the resource request and the test. An example is shown in the Example section.

The status parameter should be checked after each call to RSCPDN. If ISTAT is not zero after the first call, then the PDN requested probably does not exist (or some other error occurred). On the second, and any subsequent calls, the status parameter indicates whether the resource has been accomplished. If it has not, and you do not want to wait for the allocation, be sure to CLOSE that unit, as the allocation may occur (unexpectedly) some time in the future.

Refer to the $RESORC section in the VOS System Services Manual for more information.

Example:

```
C     Resource unit 9 to PDN 80 from an interactive program
C     Wait up to 5 seconds for the resource to occur.
C
      IPDN = 80
      IUNIT = 9
C
      CALL RSCPDN (IUNIT, IPDN, 1, ISTAT)
C     Does this PDN exist?  (If not, error out)
      IF (ISTAT.NE.0) GO TO 900
      C
      C     Loop, waiting for 1/2 second for allocation to occur.
      DO 20 I=1,10
         CALL WAITS (0.5)
         CALL RSCPDN (IUNIT, IPDN, 2, ISTAT)
         IF (ISTAT.EQ.0) GO TO 40
      20    CONTINUE
C
C     Unable to resource PDN.  Close IUNIT
      CLOSE (UNIT=IUNIT)
      GO TO 920
C
C     Successfully resourced IUNIT to IPDN.  Open IUNIT
40    OPEN (UNIT=IUNIT,IOSTAT=ISTAT)
      . . .
```

Note: If the above code were for a real-time program, the third argument in the first call to RSCPDN would be five instead of one. If this were for a control-point (batch) program, the third argument would be six.

7.16.8 XQTLINE – Get the Program's Execution Line

Purpose:

XQTLNE obtains the execution line that was used to execute the program. This line may contain parameters to be passed to the program.

Calling Sequence:

CALL XQTLNE (CLINE, NLINE)

Declarations:

```
CHARACTER CLINE
INTEGER NLINE
```

Argument Description:

CLINE	Output	A character variable to contain the execution line.
NLINE	Output	The number of characters in the execution line. If the number of characters on the execution line is more than the length of CLINE, the line will be truncated and NLINE will indicate the length of CLINE.

Remarks:

XQTLNE will obtain up to 132 characters from the execution line. XQTLNE operates for batch and interactive environments.

7.16.9 XQTJCL – Execute One Job Control Command

Purpose:

XQTJCL executes one job control command from within an interactive program. XQTJCL does not work in a batch environment.

Calling Sequence:

```
CALL XQTJCL (ISUNIT, CLINE, NLINE)
```

Declarations:

```
CHARACTER CLINE
INTEGER ISUNIT, NLINE
```

Argument Description:

ISUNIT	Input	The unit number of a blocked scratch file (which must be assigned). The contents of that file will not be preserved.
CLINE	Input	The job control command to execute.
NLINE	Input	The number of characters in CLINE.

Remarks:

XQTJCL works by writing CLINE to the scratch file, then temporarily reassigning unit 0 to that file. Job control is chained to, where it reads from the scratch file (unit 0). After executing the command, unit 0 is reassigned to the terminal and control is returned to the program.

Example:

```
C    CONNECT UNIT 9 TO WORK FILE W3
     CALL CASSIG (9, 'W9', IERR)
     IF (IERR.NE.0) GO TO 900
      . .
C    DO A CONTROL POINT LISTING
     CALL XQTJCL (9, '/CL', 3)
      . .
C    EXECUTE A MAP COMMAND
     CALL XQTJCL (9, 'MAP.UL', 6)
     CLOSE (UNIT=9)
```

7.16.10 CHAIN3 – Chain From One Program Into Another

Purpose:

CHAIN3 provides a means of chaining (or transferring control) from one program into another (or into job control). When the program chained into executes an exit (e.g., a STOP), control will be returned to the calling program. CHAIN3 attempts to re-establish those file assignments that have been freed or closed before returning to the calling program, but the files are not repositioned (if they have been reassigned).

Calling Sequence:

CALL CHAIN3 (CFROM, NFROM, CTO, NTO)

Declarations:

```
CHARACTER CFROM, CTO
INTEGER NFROM, NTO
```

Argument Description:

CFROM	Input	The name of the program calling CHAIN3. This is used for informative purposes only.
NFROM	Input	The number of characters in NFROM.
CTO	Input	The name of the program to chain to. CTO should contain the entire execution line (whatever is normally passed to the program to be executed). If chaining into job control, CTO should be '*JOBCNTRL'. It is wise to include the qualifier in the program name.
NTO	Input	The number of characters in CTO.

Remarks:

CHAIN3 operates well when chaining into another program in an interactive environment. CHAIN3 cannot chain into a HARRIS Macro, it must chain directly into a program. Program letter options can be passed via CHAIN3 (e.g., MYPROG.D). The file positions are not guaranteed (and may be at the beginning of the file upon return to the calling program). CHAIN3 does not work well in a batch environment, as the input and output files may be repositioned. Subroutine EXPROG may be better suited for batch jobs (it does not reassign any units).

Abnormal assignment types (e.g., PDN resources or exclusive assignments) will not be correctly reassigned if those units were closed or freed by the program chained into. The calling program should re-establish these types of assignments.

Example:

Chain into job control:

```
CALL CHAIN3 ('MYPROG', 6, '*JOBCNTRL', 9)
```

Chain into COED:

```
CALL CHAIN3 ('MYPROG', 6, 'SYST*COED WRKFIL', 16)
```

Chain into YOURPRG:

```
CALL CHAIN3 ('MYPROG', 6, '2002RES*YOURPRG.D INPUT=MYIN', 28)
```

7.16.11 EXPROG – Execute One Program from Another

Purpose:

EXPROG provides a means of chaining (or transferring control) from one program into another. EXPROG is similar to CHAIN3, except that no units are re-established. EXPROG may be used in a batch environment where the file assignments used are known, and any re-establishing of assignments will be completed by the calling program. CHAIN3 should be used in an interactive environment. When the program chained into executes an exit (e.g., a STOP), control will be returned to the calling program.

Calling Sequence:

```
CALL EXPROG (CPROG)
```

Declarations:

```
CHARACTER CPROG
```

Argument Description:

CPROG	Input	The name of the program to chain to. CPROG should contain the entire execution line (whatever is normally passed to the program to be executed). It is wise to include the qualifier in the program name.

Remarks:

EXPROG cannot chain into a HARRIS Macro, it must chain directly into a program. Program letter options cannot be passed via EXPROG (at this time).

7.16.12 GSTRRG – Get String Register

Purpose:

GSTRRG obtains the contents of a HARRIS string (text) register.

Calling Sequence:

CALL GSTRRG (CNAME, CSTR, NSTR, ISTAT)

Declarations:

```
CHARACTER CNAME*3, CSTR
INTEGER NSTR, ISTAT
```

Argument Description:

CNAME	Input	The three character name of the string register to obtain.
CSTR	Output	A character variable that will contain the contents of the register. If the length of CSTR is less than the register contents, the string will be truncated.
NSTR	Output	The number of characters in the string register. (NSTR is the actual number of characters in the register, regardless of the length of CSTR).
ISTAT	Output	A status parameter. If the call was successful, ISTAT is returned as zero.

Remarks:

GSTRRG will retrieve up to 255 bytes from a register. Refer to the HARRIS System Services Manual for more information on registers.

7.16.13 GNUMRG – Get Numeric Register

Purpose:

GNUMRG obtains the contents of a HARRIS numeric register.

Calling Sequence:

CALL GNUMRG (CNAME, NRANGE, NVALUE, ISTAT)

Declarations:

```
CHARACTER CNAME*3
INTEGER NRANGE, NVALUE, ISTAT
```

Argument Description:

CNAME	Input	The three character name of the numeric register to obtain.
NRANGE	Output	If the register contains a numeric range, this is the increment value. That is, NVALUE is the low value of the range and NVALUE + NRANGE is the high value. If no range is set, NRANGE is returned as zero.
NVALUE	Output	The numeric value of the register.
ISTAT	Output	A status parameter. If the call was successful, ISTAT is returned as zero.

Remarks:

Refer to the HARRIS System Services Manual for more information on registers.

7.16.14 SSTRRG – Set String Register

Purpose:

SSTRRG sets a HARRIS string (text) register.

Calling Sequence:

CALL SSTRRG (CNAME, CSTR, NSTR, ISTAT)

Declarations:

```
CHARACTER CNAME*3, CSTR
INTEGER NSTR, ISTAT
```

Argument Description:

CNAME	Input	The three character name of the string register to set.
CSTR	Input	The string to be set.
NSTR	Input	The number of characters in CSTR.
ISTAT	Output	A status parameter. If the call was successful, ISTAT is returned as zero.

Remarks:

SSTRRG will set up to 255 characters in a register. Refer to the HARRIS System Services Manual for more information on registers.

7.16.15 SNUMRG – Set Numeric Register

Purpose:

SNUMRG sets a HARRIS numeric register.

Calling Sequence:

CALL SNUMRG (CNAME, NRANGE, NVALUE, ISTAT)

Declarations:

```
CHARACTER CNAME*3
INTEGER NRANGE, NVALUE, ISTAT
```

Argument Description:

CNAME	Input	The three character name of the numeric register to set.
NRANGE	Input	If the register is to contain a numeric range, this is the increment value. That is, NVALUE is the low value of the range and NVALUE + NRANGE is the high value. If no range is to be set, NRANGE should be zero.
NVALUE	Input	The numeric value to set.
ISTAT	Output	A status parameter. If the call was successful, ISTAT is returned as zero.

Remarks:

Refer to the HARRIS System Services Manual for more information on registers.

7.16.16 TRKSET – Set Parameters for Program Tracking

Purpose:

TRKSET provides a means of setting a program name and program version for tracking HEC programs. Program tracking indicates how often programs were executed and the amount of time they took. Contact the HEC for further information.

Calling Sequence:

CALL TRKSET (CITEM, CPARM)

Declarations:

CHARACTER CITEM, CPARM

Argument Description:

CITEM	Input	The item to set. This should either be 'PROGRAM' or 'DATE'
CPARM	Input	The corresponding parameter. If the item to set is 'PROGRAM', this should be program name (up to six characters). If the item to be set is the 'DATE' this should be the version date of the program. The date to be set may be a variety of styles (see remarks).

Remarks:

TRKSET should be called at the beginning of the program, prior to any calls to ATTACH.

The date may be the month and year, or the day month and year. It may be either upper or lower case, and of several styles as long as the year is the last part of the date. See the YMDDAT subroutine documentation for the date styles that are recognized.

Example:

```
      CALL TRKSET ('PROGRAM', 'MYPROG')
      CALL TRKSET ('DATE', 'June 1986')
C
      CALL ATTSET ('MYPROG:  June 1986')
      CALL ATTACH ( ...
```

7.17 MS-DOS Specific Subroutines

7.17.1 CPARMS – Get Command Line Parameters

Purpose:

Subroutine CPARMS returns the parameters entered on the command line following the program name (used to execute the current program).

Calling Sequence:

CALL CPARMS (CLINE, NLINE)

Declarations:

```
CHARACTER CLINE*80
INTEGER*2 NLINE
```

Argument Description:

CLINE	Input	CLINE is returned with the command line beginning just after the last character of the program name. CLINE must be a long enough variable to hold the longest possible command line.
NLINE	Output	The number of characters in CLINE.

Example:

If a program is executed as follows:

```
MYPROG,File1 /r /X
```

then:

```
CLINE = ',File1 /r /X'
NLINE = 12
```

7.17.2 PRNCHR – Send a Single Character to the Printer

Purpose:

PRNCHR sends a single character to the printer.

Calling Sequence:

```
CALL PRNCHR (CCHAR)
```

Declarations:

```
CHARACTER CCHAR*1
```

Argument Description:

CCHAR	Input	The character to send to the printer.

7.17.3 PRNLN – Send a Line to the Printer

Purpose:

PRNLN sends a line to the printer with the appropriate carriage return and line feed characters added.

Calling Sequence:

CALL PRNLIN (CLINE)

Declarations:

CHARACTER CLINE

Argument Description:

CLINE	Input	The line to send to the printer.

7.17.4 DSKSPC – Determine the Amount of Disk Space Left

Purpose:

Subroutine DSKSPC returns the amount of available disk space for a specified drive.

Calling Sequence:

```
CALL DSKSPC (CDRIVE, ISPACE, ISTAT)
```

Declarations:

```
CHARACTER CDRIVE*1
INTEGER*2 ISTAT
```

Argument Description:

CDRIVE	Input	The letter of the drive to check.
ISPACE	Output	An INTEGER*4 variable returned with the number of bytes of disk space available on CDRIVE.
ISTAT	Output	A status parameter set to 0 if the call was successful. If the drive is not available, ISTAT is returned with -1.

7.17.5 WHRFRM – Get the Path of the Program Executing

Purpose:

Subroutine WHRFRM obtains the drive and path of the currently executing program.

Calling Sequence:

CALL WHRFRM (CPATH)

Declarations:

CHARACTER CPATH*68

Argument Description:

CPATH	Output	The drive and path of the program. Unused characters are blanked.

Example:

CPATH = 'C:\PROGDIR\MYPROG.EXE'

7.17.6 CPLOCK – Control the Caps Lock Key

Purpose:

CPLOCK allows manipulation of the Caps Lock key.

Calling Sequence:

```
CALL CPLOCK (CFLAG, LSTATE)
```

Declarations:

```
CHARACTER CFLAG
LOGICAL LSTATE
```

Argument Description:

CFLAG	Input	A flag indicating what to set the Caps Lock key to. The following flags are valid: 'SET' Turn the Caps Lock key on. 'RESET' Turn the Caps Lock key off. 'TOG' Toggle the Caps Lock key. 'CURR' Just return the current state of the Caps Lock key.
LSTATE	Output	The state of the Caps Lock key after the desired action. LSTAT is returned .TRUE. if Caps Lock is on.

7.17.7 NMLOCK – Control the Num Lock Key

Purpose:

NMLOCK allows manipulation of the Num Lock key.

Calling Sequence:

CALL NMLOCK (CFLAG, LSTATE)

Declarations:

```
CHARACTER CFLAG
LOGICAL LSTATE
```

Argument Description:

CFLAG	Input	A flag indicating what to set the Num Lock key to. The following flags are valid: 'SET' Turn the Num Lock key on. 'RESET' Turn the Num Lock key off. 'TOG' Toggle the Num Lock key. 'CURR' Just return the current state of the Num Lock key.
LSTATE	Output	The state of the Num Lock key after the desired action. LSTAT is returned .TRUE. if Num Lock is on.

7.17.8 PRESED – Which (Special) Keys are Pressed

Purpose:

CALL PRESED (LALT, LCTRL, LLSHFT, LRSHFT)

Calling Sequence:

CALL PRESED (LALT, LCTRL, LLSHFT, LRSHFT)

Declarations:

LOGICAL LALT, LCTRL, LLSHFT, LRSHFT

Argument Description:

LALT	Output	Returns .TRUE. if the Alt key is pressed, .FALSE. if it is not.
LCTRL	Output	Returns .TRUE. if the Control key is pressed.
LLSHFT	Output	Returns .TRUE. if the left shift key is pressed.
LRSHFT	Output	Returns .TRUE. if the right shift key is pressed.

7.17.9 FILEN – Get File Names for a Directory

Purpose:

Subroutine FILEN obtains the names and attributes of the files in a directory, based upon a file name mask. FILEN is called once for each file, until a status flag indicates that all files (matching the mask) have been found.

Calling Sequence:

```
      CALL FILEN (CMASK, IFATT, CMODE, CFNAME, IFSIZE, CFDATE,
     * CFTIME, IATT, ISTAT)
```

Declarations:

```
CHARACTER CMASK, CMODE*1, CFNAME*12, CFDATE*6, CFTIME*8
INTEGER*2 IFATT, IATT
INTEGER*4 IFSIZE
```

Argument Description:

CMASK	Input	The DOS mask used for defining file names, ending with a zero byte (e.g., CMASK//CHAR(0)). To search for all files (and subdirectories) in the current directory, use a mask of '*.*'//CHAR(0). To search for all ".OUT" files, use a mask of '*.OUT'//CHAR(0).
IFATT	Input	The attributes of the files to search for, as defined in chapter 5 of the DOS Technical Reference Manual (page 5-11). To search for all normal files, set IFATT = 0.
CMODE	Input	A flag to indicate if this is the first call, or a "next" call. If the first, set CMODE = 'F'. If a next call (after the first call), set CMODE = 'N'.
CFNAME	Output	The name of the file found.
IFSIZE	Output	The size of the file found, in bytes.
CFDATE	Output	The last written date of the file, returned in a format of YYMMDD.
CFTIME	Output	The last written time of the file, returned in a format of HH:MM:SS.
IATT	Output	The attributes of the file, as described in chapter 5 of the DOS Technical Reference Manual (page 5-11).

ISTAT	Output	A status flag returned as zero if a file was found (and FILEN should be called again), or returned as one if no more files were found (the search should be ended).

Remarks:

FILEN uses the DOS FIND FIRST (4EH) and FIND NEXT (4FH) functions for searching. Refer to the documentation of those functions in the Dos Technical Reference Manual for more information (Chapter 6).

Example:

Find all files in the current directory with an extension of '.DAT'. Place these names in the character array CNAMES.

```
      CHARACTER CFNAME*12, CFDATE*6, CFTIME*8, CNAMES(100)*12
      INTEGER*2 IFATT, IATT
      INTEGER*4 IFIZE
C
      NNAMES = 0
      CALL FILEN ('*.DAT'//CHAR(0), 0, 'F', CFNAME,
*     IFSIZE, CFDATE, CFTIME, IATT, ISTAT)
      IF (ISTAT.NE.0) GO TO 100
C
      NNAMES = NNAMES + 1
      CNAMES(NNAMES) = CFNAME
C
10    CONTINUE
      CALL FILEN ('*.DAT'//CHAR(0), 0, 'N', CFNAME,
*     IFSIZE, CFDATE, CFTIME, IATT, ISTAT)
      IF (ISTAT.NE.0) GO TO 100
      NNAMES = NNAMES + 1
      IF (NNAMES.GT.100) GO TO 900
      CNAMES(NNAMES) = CFNAME
      GO TO 10
C
100   CONTINUE

      ...
```

7.17.10 GETPTH – Get the Current Path

Purpose:

GETPTH obtains the current path (directory), given the letter of the drive for the path desired.

Calling Sequence:

CALL GETPTH (CDRIVE, CPATH)

Declarations:

CHARACTER CDRIVE*1, CPATH*68

Argument Description:

CDRIVE	Input	The letter of the drive for the path desired.
CPATH	Output	The current path of that drive (including the drive letter). If no drive letter is specified in CDRIVE, or the drive letter is invalid, the default drive is used.

Remarks:

If CDRIVE is not a valid drive, or a blank, CPATH is returned with the default drive and path.

Example:

If:

```
CALL GETPTH ('A', CPATH)
```

then:

```
CPATH = 'A:\MYDIR'
```

7.17.11 GETDRV – Get the Default Drive

Purpose:

GETDRV obtains the default drive letter.

Calling Sequence:

```
CALL GETDRV (CDRIVE)
```

Declarations:

```
CHARACTER CDRIVE*1
```

Argument Description:

CDRIVE	Output	The letter of the current default drive.

7.17.12 SETDRV – Set the Default Drive

Purpose:

SETDRV sets the default drive to the letter specified.

Calling Sequence:

CALL SETDRV (CDRIVE, IDRIVE)

Declarations:

```
CHARACTER CDRIVE*1
INTEGER*2 IDRIVE
```

Argument Description:

CDRIVE	Input	The letter to set the default drive to.
IDRIVE	Output	The number of drives in the system.

7.17.13 CHDIR – Change Directory

Purpose:

CHDIR changes the current directory to the directory specified. Refer to the CHDIR function (3BH) in the DOS Technical Reference Manual for more information (page 6-121).

Calling Sequence:

CALL CHDIR (CDIR, ISTAT)

Declarations:

```
CHARACTER CDIR
INTEGER*2 ISTAT
```

Argument Description:

CDIR	Input	The name of the directory to change to. This name must be terminated by a zero value byte (e.g., CDIR//CHAR(0)).
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

7.17.14 MKDIR – Make Directory

Purpose:

MKDIR creates the specified subdirectory. Refer to the MKDIR function (39H) in the DOS Technical Reference Manual for more information (page 6-119).

Calling Sequence:

```
CALL MKDIR (CDIR, ISTAT)
```

Declarations:

```
CHARACTER CDIR
INTEGER*2 ISTAT
```

Argument Description:

CDIR	Input	The name of the directory to create. This name must be terminated by a zero value byte (e.g., CDIR//CHAR(0)).
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

7.17.15 RMDIR – Remove Directory

Purpose:

RMDIR removes the specified subdirectory. Refer to the RMDIR function (3AH) in the DOS Technical Reference Manual for more information (page 6-120).

Calling Sequence:

```
CALL RMDIR (CDIR, ISTAT)
```

Declarations:

```
CHARACTER CDIR
INTEGER*2 ISTAT
```

Argument Description:

CDIR	Input	The name of the directory to remove. This name must be terminated by a zero value byte (e.g., CDIR//CHAR(0)).
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

7.17.16 CRDIR – Create Directories

Purpose:

CRDIR creates all the subdirectories of the specified path. This is different than MKDIR in that MKDIR creates only one directory, while CRDIR creates all the directories specified in the path (that do not already exist).

Calling Sequence:

CALL CRDIR (CPATH, ISTAT)

Declarations:

```
CHARACTER CPATH
INTEGER*2 ISTAT
```

Argument Description:

CPATH	Input	The path containing the directories to create. If the drive is not included, the default drive will be used.
ISTAT	Output	A status parameter, set to zero if the call was successful. Nonzero error codes may be found on page 6-42 of the DOS Technical Reference Manual.

Example:

```
CALL CRDIR ('D:\DIRA\DIRB\DIRC')
```

This is equivalent to:

```
MKDIR D:\DIRA
MKDIR D:\DIRA\DIRB
MKDIR D:\DIRA\DIRB\DIRC
```

7.17.17 GETSUP – Get Path of a Supplemental File

Purpose:

GETSUP obtains the path for the specified file. Such a file would be one used by the calling program (e.g., the program help file), but whose exact location is unknown (depending on how the program was installed).

Calling Sequence:

CALL GETSUP (CNAME, CPATH, NPATH)

Declarations:

```
CHARACTER CNAME, CPATH
INTEGER*2 NPATH
```

Argument Description:

CNAME	Input	The name of the file to search for. If CNAME is blanked, GETSUP will look for the first existing path in the search order listed below.
CPATH	Output	The complete path of the file, including the drive.
NPATH	Output	The number of characters in CPATH. NPATH is returned with 0 if the file could not be found. If the number of actual characters in the path was more than the length of CPATH, NPATH is returned as -1.

Remarks:

The search order of GETSUP is as follows:

1. Searches the Environment Table for HECSUP. If found, that directory is searched.
2. If the program resides in directory \HECEXE, the directory \HECEXE\SUP is searched.
3. The directory where the program resides.
4. The default directory is searched.

7.17.18 FSTENV/NXTENV – Get the Environment Table

Purpose:

Subroutines FSTENV and NXTENV returns items in the environment table. FSTENV returns the first item in the table, and NXTENV returns subsequent items. Items in the environment table may include the PATH, and the prompt. A list of the item set will be displayed by executing the DOS 'SET' command.

Calling Sequence:

```
CALL FSTENV (CITEM, NITEM)
CALL NXTENV (CITEM, NITEM)
```

Declarations:

```
CHARACTER CITEM
INTEGER*2 NITEM
```

Argument Description:

CITEM	Output	The item from the environment table.
NITEM	Output	The number of characters in CITEM. NITEM is returned with 0 if there are no more items in the table. If the number of actual characters for the item was more than the length of CITEM, NITEM is returned as -1.

Example:

```
      C  Print out all items in the Environment Table
         CHARACTER CITEM*80
      C
         CALL FSTENV (CITEM, NITEM)
         IF (NITEM.EQ.0) GO TO 100
         IF (NITEM.EQ.-1) GO TO 900
         WRITE (6, 10) CITEM(1:NITEM)
   10    FORMAT (A)
      C
   20    CALL NXTENV (CITEM, NITEM)
         IF (NITEM.EQ.0) GO TO 100
         IF (NITEM.EQ.-1) GO TO 900
         WRITE (6, 10) CITEM(1:NITEM)
         GO TO 20
```

7.17.19 ICAT – Concatenate Two Bytes into One Word

Purpose:

Integer function ICAT takes the lower bytes of two integer words and concatenates them into one integer word. The higher bytes of each word are ignored.

Calling Sequence:

IWORD = ICAT (IHIGH, ILOW)

Declarations:

INTEGER*2 ICAT, IHIGH, ILOW

Argument Description:

IHIGH	Input	The lower byte of this integer word will be placed in the higher byte of ICAT.
ILOW	Input	The lower byte of this integer word will be placed in the lower byte of ICAT.
ICAT	Output	The concatenated word.

Example:

```
I = ICAT (1,0)
      ICAT returns I = 256

I = ICAT (0,1)
      ICAT returns I = 1
```

7.17.20 DCAT – De-Concatenate One Word into Two Bytes

Purpose:

Subroutine DCAT takes an integer word and de-concatenates it into the lower bytes of two integer words.

Calling Sequence:

CALL DCAT (IWORD, IHIGH, ILOW)

Declarations:

INTEGER*2 IWORD, IHIGH, ILOW

Argument Description:

IWORD	Input	The integer word to be de-concatenated.
IHIGH	Output	The higher byte of IWORD (placed in the lower byte position of IHIGH with the higher byte zeroed).
ILOW	Output	The lower byte of IWORD (placed in the lower byte position of ILOW with the higher byte zeroed).

Example:

CALL DCAT (256, IHIGH, ILOW)
returns IHIGH = 1, ILOW = 0

CALL DCAT (255, IHIGH, ILOW)
returns IHIGH = 0, ILOW = 255

7.17.21 DBITS – Determine Which Bits of a Byte are Set

Purpose:

DBITS takes the lower byte of an integer words and determines which bits are set. The status of each bit is indicated by a 1 if that bit is on, or a 0 if that bit is off.

Calling Sequence:

CALL DBITS (IBYTE, IB7, IB6, IB5, IB4, IB3, IB2, IB1, IB0)

Declarations:

INTEGER*2 IBYTE, IB7, IB6, IB5, IB4, IB3, IB2, IB1, IB0

Argument Description:

IBYTE	Input	The integer word of which the lower byte is to be used in determining which bits are set.
IB7	Output	The status of the 7th (highest) bit. This integer is set to one (1) if the bit is on, zero (0) if the bit is off.
IB6	Output	The status of the 6th bit.
IB5	Output	The status of the 5th bit.
IB4	Output	The status of the 4th bit.
IB3	Output	The status of the 3th bit.
IB2	Output	The status of the 2th bit.
IB1	Output	The status of the 1st bit.
IB0	Output	The status of the 0^{th} (lowest) bit.

8 Special Purpose Subroutines

This chapter describes those subroutines that are used for special purposes on HARRIS machines only. These subroutines are low-level subroutines that are not generally called by typical programs. Computer manual are needed to use a considerable number of these subroutines.

8.1 HARRIS Specific Subroutines

8.1.1 INFO2 – Get Information About This Session

Purpose:

INFO2 returns several pieces of information about the current session. This includes such items as the PDN, the name of the program running, the priority of the program, the qualifier, the user's name, and the starting time of the session.

Calling Sequence:

```
      CALL INFO2 (CPTYPE, CPDN, IPDN, IPRIOR, CPROG,
•     * CDQUAL, CSQUAL, CUNAME, CUNMUB, CSTIME)
```

Declarations:

```
CHARACTER CPTYPE*3, CPDN*3, CPROG*17, CDQUAL*8
CHARACTER CSQUAL*8, CUNAME*12, CUNUMB*6, CSTIME*16
INTEGER IPDN, IPRIOR
```

Argument Description:

CPTYPE	Output	The program type. CPTYPE will be returned as one of the following: 'INT' Interactive 'CP' Control Point 'RT' Real Time
CPDN	Output	The physical device number that the program is running under. This is a three character string containing the PDN or the control point letter (e.g., '52', or 'F'). If the program is a real-time program, CPDN will contain the PDN that it was initiated from.
IPDN	Output	The physical device number in an integer representation. If the program is running at a control point, IPDN is returned as zero. If the program is running interactively, IPDN will be returned positive containing the PDN of the terminal.
IPRIOR	Output	The current priority of the session (e.g., 15).
CPROG	Output	The file name of the program executing. This is the complete name of the file of the program, including the qualifier. For example, '1000SYSS*MYPROGX'.

CDQUAL	Output	The default qualifier set.
CSQUAL	Output	The qualifier the user signed-on under.
CUNAME	Output	The twelve character user's name associated with this session.
CUNUMB	Output	The six character user's number.
CSTIME	Output	The starting date and time of this session (or when this program was initiated, if a real time program). CSTIME is returned in the format: 'DD MMM YY HH:MM:SS' For example: '16 JAN 86 14:30:15'

Remarks:

Make sure the character variables declared are of the correct length, as shown in the declarations.

8.1.2 GRNSIZ – Get the Granule Size of a File

Purpose:

GRNSIZ returns the granule size of an assigned file.

Calling Sequence:

CALL GRNSIZ (UNIT, IGSIZE)

Declarations:

INTEGER IUNIT, IGSIZE

Argument Description:

IUNIT	Input	The unit number assigned to the file to determine the granule size of.
IGSIZE	Output	The granule size of the file.

8.1.3 FOPEN – Fast Open

Purpose:

FOPEN opens a file without changing the last accessed or written date and time. This reduces the time in connecting to a file. FOPEN should only be called when time is of the essence. The file must have been previously assigned.

Calling Sequence:

CALL FOPEN (UNIT, ISTAT)

Declarations:

INTEGER IUNIT, ISTAT

Argument Description:

IUNIT	Input	The unit number assigned to the file to open.
ISTAT	Output	A status parameter set to zero if the call was successful.

Remarks:

See the VOS I/O Services Manual section on $I/O (function code ’13) for more information and return status codes.

8.1.4 GETQDD – Get the Qualifier Disc Directory of a File

Purpose:

GETQDD returns a file's QDD (Qualifier Disc Directory) and similar information. This information includes the file name, size, last referenced date/time, etc. Refer to the $DASAVE command in the VOS System Services Manual for a complete list.

Calling Sequence:

```
CALL GETQDD (IAREA, IQUAL, IQDD)
```

Declarations:

```
INTEGER IAREA(2), IQUAL(2), IQDD(28)
```

Argument Description:

IAREA	Input	The file name, in truncated ASCII.
IQUAL	Input	The qualifier, in truncated ASCII.
IQDD	Output	The QDD and other information. This is the list parameters that follows the $DASAVE command. Note that PARLIST +0 corresponds to element one of array IQDD. Also note that the date and times are coded in a special format.

8.1.5 SYSLV – Get Current Operating System Level

Purpose:

SYSLV returns the current VOS operation system version. Only the first 3 characters of the level are returned.

Calling Sequence:

```
CALL SYSLV (ILEVEL)
```

Declarations:

```
INTEGER ILEVEL
```

Argument Description:

ILEVEL	Output	The current operation system level returned in a Hollerith format (A3).

Example:

```
   CALL SYSLV (ILEVEL)
WRITE (3,' (1X,A3)')ILEVEL
```

This is printed out as:

```
5.1
```

8.1.6 NXTLFN – Determine Units of All Files Assigned

Purpose:

NXTLFN will determine the unit numbers and open status of all files assigned. This is accomplished by calling NXTLFN several times, one for each unit, until NXTLFN indicates no more units are opened. Unit numbers are returned in a sequential order (beginning with unit 0).

Calling Sequence:

CALL NXTLFN (IUNIT, IOPEN)

Declarations:

INTEGER IUNIT, IOPEN

Argument Description:

IUNIT	Input/ Output	To begin the search for files assigned, set IUNIT to negative one (-1). NXTLFN will return the unit number of each file assigned in IUNIT. When all files assigned have been determined, IUNIT will be returned as -1.
IOPEN	Output	A flag indicating whether this file is opened or not. IOPEN is returned as 1 for an opened file, 0 if the file is not open.

Example:

```
C    PRINT THE UNIT NUMBER OF ALL ASSIGNED FILES
     IUNIT = -1
10   CONTINUE
     CALL NXTLFN (IUNIT, IOPEN)
     IF (IUNIT.EQ.-1) GO TO 30
     WRITE (6,20) IUNIT, IOPEN
20   FORMAT (‘ UNIT ASSIGNED:’I4,’ OPEN STATUS: ‘,I2)
     GOT TO 10
C
30   CONTINUE
```

8.1.7 TRNSBK – Transmit a Break

Purpose:

TRNSBK sends a break to a physical device on an Async port. TRNSBK is typically used to send a break to a resourced modem.

Calling Sequence:

```
CALL TRNSBK (IUNIT, ISTAT)
```

Declarations:

```
INTEGER IUNIT, ISTAT
```

Argument Description:

IUNIT	Input	The unit number connected to the PDN to send the break to.
ISTAT	Output	A status parameter, returned zero if no error occurred.

Remarks:

The unit must be connected to an Async port. A break will not be sent on a TTY or CRT port.

8.1.8 SPINT – Send a Special Interrupt to a Program

Purpose:

Special Interrupt provides a means for two or more programs to communicate with each other. One program "interrupts" the other program (regardless of what it is doing), optionally passing a buffer of information.

Special interrupt is usually used with real-time programs. Refer to the $SPINT documentation in the VOS System Services (Chapter 12) for information on interrupts. It is intended that the $SPINT documentation be the primary source of information, and should be referenced along with this documentation.

Subroutine Summary:

SPINIT	-	Initialize special interrupts
SPINFO	-	Get the information buffer
SPDID	-	Define program identification
SPIP	-	Initiate a sub-system program with special interrupts
SPTRIG	-	Trigger a special interrupt
SPHINT	-	Hold interrupts
SPRINT	-	Release interrupts
SPWAIT	-	Wait for interrupt
SPDLAY	-	Wait a specified amount of time for an interrupt
IRETRN	-	Return from an interrupt subroutine

8.1.8.1 SPINIT – Initialize Special Interrupts

Purpose:

SPINIT enables special interrupts to be sent or received. SPINIT must be called prior to any other special interrupt routines, and before any interrupts take place. This is the $SPINT function.

Calling Sequence:

CALL SPINIT (ILEVEL, IBUFF, sub-name, ISTAT)

Declarations:

```
INTEGER ILEVEL, IBUFF(9), ISTAT
EXTERNAL sub-name
```

Argument Description:

ILEVEL	Input	The group/level specification as defined in $SPINT documentation. Bit 23 enables or disable the special interrupt.
IBUFF	Input/ Output	A nine word integer buffer, returned with. The information defined in chapter 12.2.1 of the VOS System Services Manual.
sub-name	Input	The name of the subroutine to execute when the program receives a special interrupt. This must be a literal, and must be define as an external. Returns from this subroutine should be made by a call to IRETRN, instead of a FORTRAN RETURN.
ISTAT	Output	A status parameter set to zero if the call was successful.

8.1.8.2 SPINFO – Get the Information Buffer Passed

Purpose:

SPINFO retrieves the information buffer that was optionally passed to it via a SPTRIG call. This is the $SPINFO function.

Calling Sequence:

CALL SPINFO (INFOB, NINFO, ISTAT)

Declarations:

INTEGER NINFO, INFOB(NINFO), ISTAT

Argument Description:

INFOB	Output	The information buffer. This is integer array, NINFO words long.
NINFO	Input	The dimension of INFOB.
ISTAT	Output	A status parameter set to zero if the call was successful.

8.1.8.3 SPDID – Define Program Identification

Purpose:

SPDID defines the identifier (name) of the program. This is the $DEFID function.

Calling Sequence:

CALL SPDID (INAME, ISTAT)

Declarations:

INTEGER INAME(2), ISTAT

Argument Description:

INAME	Input	A six character Hollerith string containing the name or ID of the program for other programs to refer to.
ISTAT	Output	A status parameter set to zero if the call was successful.

8.1.8.4 SPIP – Initiate a Sub-System Program with Special Interrupts

Purpose:

SPIP initiates a sub-system real-time program with special interrupts. It does not need to be called for program not using a sub-system. This is the $INITSS function.

Calling Sequence:

CALL SPIP (ITAREA, ITQUAL, IPRI, IPAR, IPID, ILEVEL, IBUFF, ISTAT)

Declarations:

```
INTEGER ITAREA(2), ITQUAL(2), IPRI, IPAR, IPID(2)
INTEGER ILEVEL, IBUFF(9), ISTAT
```

Argument Description:

ITAREA	Input	The file (area) name of the program to initiated, in truncated ASCII.
ITQUAL	Input	The qualifier of the program's file name, in truncated ASCII.
IPRI	Input	The priority to initiate it at.
IPAR	Input	A parameter to be sent to the program via the "K" register.
IPID	Input	A six character program identifier, as defined in the SPDID subroutine.
ILEVEL	Input	The group/level word, as defined in the SPINIT subroutine.
IBUFF	Input/ Output	The nine word buffer define in the SPINIT subroutine.
ISTAT	Output	A status parameter, set to zero if the call was successful.

8.1.8.5 SPTRIG – Trigger a Special Interrupt

Purpose:

SPTRIG initiates a sub-system real-time program with special interrupts. It does not need to be called for programs not using a sub-system. This is the $INITSS function.

Calling Sequence:

CALL SPTRIG (IPID, INWORD, ILEVEL, INFOB, NINFO, ISTAT)

Declarations:

INTEGER IPID(2), INWORD, ILEVEL, INFOB(NINFO), ISTAT

Argument Description:

IPID	Input	The identifier of the program to send the interrupt to. This is the same identifier used in the SPDID call (which the receiving program must have called).
INWORD	Input	A single information word that is passed to the receiving program.
ILEVEL	Input	The group/level word defined in the SPINIT call.
INFOB	Input	When more than one word of information needs to be passed, the information is passed via this array. The receiving program obtains.
NINFO	Input	The number of words in INFOB to pass.
ISTAT	Output	A status parameter, set to zero if the call was successful.

8.1.8.6 SPHINT – Hold Interrupts

Purpose:

SPHINT holds interrupts so that they queue instead of interrupting the program. This is usually called after a program has been interrupted, so that another interrupt does not interrupt the current process. Interrupts are released by calling SPRINT. Returns from and interrupt processing subroutine should be made my call to IRETRN, not a FORTRAN RETURN. This is the $HPINT function.

Calling Sequence:

```
CALL SPHINT
```

Declarations:

```
CHARACTER CNAMES(MAXNAM)*(*)
INTEGER INAMES(MAXNAM+5)
```

8.1.8.7 SPRINT – Release Interrupts

Purpose:

SPRINT releases interrupts after a call to SPHINT is made. SPRINT should be called just prior to IRETRN. This is he $RPINT function.

Calling Sequence:

```
CALL SPRINT
```

8.1.8.8 SPWAIT – Wait for Interrupts

Purpose:

SPDLAY causes the program to wait a specified amount of time for an interrupt. If the time specified expires, a normal return will occurs. If an interrupt is received, that interrupt will be processed, and any remaining time to wait will be canceled. This is the $IDELAY function.

Calling Sequence:

```
CALL SPWAIT
```

8.1.8.9 SPDLAY – Wait a Specified Amount of Time for an Interrupt

Purpose:

Subroutine ABORT initiates an abort procedure (which causes certain error processes to occur), then stops the program. On HARRIS computers, this includes printing the program address when the abort occurred, and setting certain error registers (this will cause a batch job to terminate). If walkback is set, the program will print the location and subroutines called to this location. ABORT should be called only when a significant error occurs.

Calling Sequence:

CALL SPDLAY (NTICKS, ISTAT)

Declarations:

INTEGER NTICKS, ISTAT

Argument Description:

NTICKS	Input	The number of clock ticks to wait. Clock ticks are given in 1/120th of a second.
ISTAT	Output	A status parameter set to zero if the call was successful.

8.1.8.10 IRETRN – Return from an Interrupt Subroutine

Purpose:

IRETRN returns an interrupt processing subroutine to the calling program. IRETRN should always be called instead of the FORTRAN RETURN statement. This is the $IRETRN function.

Calling Sequence:

```
CALL IRETRN
```

8.1.8.11 Special Interrupt Example

The use of special interrupts can be quiet involved. The following example briefly shows the order that the subroutines should be called in.

```
      PROGRAM MYPROG
C
C     This is Real-time program that looks at a resourced port.
C     When at least 10 characters arrive, it stores the data in a
C     buffer and sends it to real-time program PRDATA
C     via an interrupt.
C     PRDATA may send info back to MYPROG.
C
      EXTERNAL PROSUB
      COMMON/INTBLK/ INTBUF(9)
      INTEGER ITAREA, ITQUAL
C
      DATA ILEVEL/'02000100/
      DATA PID /6HMYPROG/
      DATA PIDPR /6HPRDATA/
C
C     Initialize interrupts (if a interrupt is received,
C     PROSUB is called)
      CALL SPINIT (ILEVEL, INTUBUF, PROSUB, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
      CALL SPDID (PID, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C
C     Initialize processing program "PRDATA"
      CALL ATOTA (8H0000SYST, ITQUAL, 8)
      CALL ATOTA (8HPRDATA, ITAREA, 8)
      CALL SPIP (ITAREA, ITQUAL, 30, 0, PIDPR, ILEVEL, INTBUF, ISTAT)
      IF (ISTAT.NE.0) GOT TO 920
C
C     Now resource the port, etc.
      . . .
C
C     Assume that NCHS characters have arrived.  Send them to PRDATA
C     via an interrupt.  (Send the number of characters
C     as the information word)
      NBUFF = (N-1)/3+1
      CALL SPTRIG (PIDPR, N, ILEVEL, IBUFF, NBUFF, ISTAT)
C
      . . .
```

```
      SUBROUTINE PROSUB
C
C     This subroutine is called if MYPROG is interrupted
C
      INTEGER INFOB(100)
C
C     Hold any interrupts
      CALL SPHINT
C
C     Get any information passed
      NWORD = 100
      CALL SPINFO (INFOB, NWORD, ISTAT)
C     Process that information
      . . .

C     All done – go back to main program
C     First, release interrupts
      CALL SPRINT
C     Now return, using IRETRN
      CALL IRETRN
      END
```

Program PRDATA would have similar calls and a similar PROSUB subroutine. Because PRDATA was initiated by SPIP, PRDATA does not need a SPINIT or SPDID call. If PRDATA was initiated via some other means (e.g., it was initiated via a call to SPTRIG.

PRDATA's main loop may be a call to SPWAIT:

```
10    CONTINUE
      CALL SPWAIT
      GO TO 10
```

8.1.9 GETA – Get the A Register

Purpose:

GETA returns the current value of the "A" software register.

Calling Sequence:

```
CALL GETA (IA)
```

Declarations:

```
INTEGER IA
```

Argument Description:

IA	Output	IA is returned with the current value of the "A" register.

8.1.10 GETE – Get the E Register

Purpose:

GETE returns the current value of the "E" software register.

Calling Sequence:

CALL GETE (IE)

Declarations:

INTEGER IE

Argument Description:

IE	Output	IE is returned with the current value of the "E" register.

8.1.11 GETK – Get the K Register

Purpose:

GETK returns the current value of the K software register. The K register holds the program parameter passed to a real-time program. GETK will retrieve this parameter if it is the very first executable statement in the program.

Calling Sequence:

```
CALL GETK (K)
```

Declarations:

```
INTEGER K
```

Argument Description:

K	Output	K is returned with the current value of the "K" register.

8.1.12 CHRLOC – Get the Address of a Character Variable

Purpose:

CHRLOC is a low level routine that retunes the equivalent integer address and length of a character variable. The subroutine is usually used for machine specific low level operations.

Calling Sequence:

CALL CHRLOC (CHR, IWADD, IPOS, ILEN)

Declarations:

```
CHARACTER CHR
INTEGER IWADD, IPOS, ILEN
```

Argument Description:

CHR	Input	The character variable.
IWADD	Output	The equivalent integer word address of the character variable.
IPOS	Output	The byte position within IWADD of the variable. IPOS may range from 1 to 3, with 1 being the left-most byte.
ILEN	Output	The length of the character variable, in bytes.

8.1.13 OPTSET – Set Program Options

Purpose:

OPSET sets the option word for a program. This is normally called during chaining.

Calling Sequence:

```
CALL OPSET (IBITS)
```

Declarations:

```
INTEGER IBITS
```

Argument Description:

IBITS	Input	The option word to be set, with bits indicating which letter option is set. Bit 0 corresponds to option A, bit 23 corresponds to option X.

8.2 MS-DOS Specific Subroutines

8.2.1 MEMSIZ – Memory Size

Purpose:

MEMSIZ returns the amount of RAM installed in the computer. This is not the amount of free memory available.

Calling Sequence:

CALL MEMSIZ (IMEM)

Declarations:

INTEGER*2 IMEM

Argument Description:

IMEM	Output	The amount of RAM, in kilobytes (e.g., 640).

8.2.2 KEYBRD – Keyboard Interrupt

Purpose:

The subroutine KEYBRD provides direct access to the BIOS keyboard control. Refer to section 5 of the IBM Technical Reference Manual fore information regarding this (page 5-46).

Calling Sequence:

CALL KEYBRD (IAX, IBX, ICX, IDX, IFLAGS)

Declarations:

INTEGER*2 IAX, IBX, ICX, IDX, IFLAGS

Argument Description:

IAX	Input/ Output	The AX register.
IBX	Input/ Output	The BX register.
ICX	Input/ Output	The CX register.
IDX	Input/ Output	The DX register.
IFLAGS	Input/ Output	The FLAGS register.

8.2.3 VIDEO – Video Interrupt

Purpose:

The subroutine VIDEO provides a direct access to the BIOS display control. Refer to section 5 (I/O Support: Display) of the IBM Technical Reference Manual for information.

Calling Sequence:

CALL VIDEO (IAX, IBX, ICX, IDX)

Declarations:

INTEGER*2 IAX, IBX, ICX, IDX

Argument Description:

IAX	Input/ Output	The AX register.
IBX	Input/ Output	The BX register.
ICX	Input/ Output	The CX register.
IDX	Input/ Output	The DX register.

8.2.4 GETPSP – Get Program Segment Prefix

Purpose:

GETPSP returns the segment address of the program segment prefix (PSP). Refer to chapter 7 of the DOS Technical Reference Manual for more information.

Calling Sequence:

```
CALL GETPSP (ISEG)
```

Declarations:

```
INTEGER*2 ISEG
```

Argument Description:

ISEG	Output	The segment address of the PSP.

8.2.5 PEEKB – Get Byte from PSP

Purpose:

PEEKB returns the value of the byte from the memory location specified.

Calling Sequence:

CALL PEEKB (ISEG, IOFF, IVAL)

Declarations:

INTEGER*2 ISEG, IOFF, IVAL

Argument Description:

ISEG	Input	The segment address of the PSP.
IOFF	Input	The offset from ISEG, in bytes.
IVAL	Output	The value of the byte in memory at the location (given in the lower byte of IVAL).

8.2.6 PEEKW – Get Word from PSP

Purpose:

PEEKW returns the value of the word from the memory location specified.

Calling Sequence:

CALL PEEKW (ISEG, IOFF, IVAL)

Declarations:

INTEGER*2 ISEG, IOFF, IVAL

Argument Description:

ISEG	Input	The segment address of the PSP.
IOFF	Input	The offset from ISEG, in bytes.
IVAL	Output	The value of the word in memory at that location.

8.2.7 POKEB – Set Byte in PSP

Purpose:

POKEB sets a byte in the memory location specified.

Calling Sequence:

CALL POKEB (ISEG, IOFF, IVAL)

Declarations:

INTEGER*2 ISEG, IOFF, IVAL

Argument Description:

ISEG	Input	The segment address of the PSP.
IOFF	Input	The offset from ISEG, in bytes.
IVAL	Input	The value of the byte to set (in the lower byte of IVAL).

8.2.8 POKEW – Set Word in PSP

Purpose:

POKEW sets a word in the memory location specified.

Calling Sequence:

CALL POKEW (ISEG, IOFF, IVAL)

Declarations:

INTEGER*2 ISEG, IOFF, IVAL

Argument Description:

ISEG	Input	The segment address of the PSP.
IOFF	Input	The offset from ISEG, in bytes.
IVAL	Input	The value of the word to set.

8.2.9 INPB – Read a Byte from a Port

Purpose:

INPB reads a byte (inports) from the specified hardware port.

Calling Sequence:

CALL INPB (IPORT, IVAL)

Declarations:

INTEGER*2 IPORT, IVAL

Argument Description:

IPORT	Input	The port to read the byte from.
IVAL	Output	The value of the byte read (in lower byte of IVAL).

8.2.10 INPW – Read a Word from a Port

Purpose:

INPW reads a word (inports) from the specified hardware port.

Calling Sequence:

CALL INPW (IPORT, IVAL)

Declarations:

INTEGER*2 IPORT, IVAL

Argument Description:

IPORT	Input	The port to read from.
IVAL	Output	The value of the word read.

8.2.11 OUTPB – Write a Byte to a Port

Purpose:

OUTPB writes a byte (outports) to the specified hardware port.

Calling Sequence:

CALL OUTPB (IPORT, IVAL)

Declarations:

INTEGER*2 IPORT, IVAL

Argument Description:

IPORT	Input	The port to write the byte to.
IVAL	Output	The byte to write (in the lower byte of IVAL).

8.2.12 OUTPW – Write a Word to a Port

Purpose:

OUTPW writes a word (outports) to the specified hardware port.

Calling Sequence:

CALL OUTPW (IPORT, IVAL)

Declarations:

INTEGER*2 IPORT, IVAL

Argument Description:

IPORT	Input	The port to write the word to.
IVAL	Output	The word to write.

Appendix A Obsolete Subroutines

The following subroutines are no longer supported. However, they currently remain in HECLIB for compatibility. Programs accessing these subroutines should be updated. These subroutines will be removed from the library in the future.

Subroutine	Replaced By
A4TOCH	none
CHABLK	CHRBLK
CHAFIL	CHRFIL
CHARFL	CHRFIL
CHTOA4	none
CIJOB	CIJOBE
CLOSE	FORTRAN CLOSE (or GIOP)
COMPAR	FORTRAN .EQ.
CREAT	CREAF
DEFDRV	GETDRV
DISECT	none
ILSTR	FORTRAN INDEX
IYMDML	YMDDAT
JULMIL	JULDAT
JNLML9	JULDAT
LASTCH	CHRLNB
LOCKST	CPLOCK
LSAME	FORTRAN .EQ.
MILJUL	DATJUL
MLIYMD	DATYMD
MOVSTR	FORTRAN =
OPENFL	FORTRAN OPEN (or GIOP)
PDN	TRMTYP
STDINS	none
STRBLK	CHRBLK
STRMOV	FORTRAN =
TOUPC	UPCASE
XPARMS	ATTACH

Appendix B Summary of Subroutine Calling Sequences

Subroutine Index