M T S

The Michigan Terminal System

Volume 3: System Subroutine Descriptions

Reference R1003

April 1981

Updated March 1982 (Update 1)
Updated February 1983 (Update 2)
Updated January 1984 (Update 3)
Updated September 1984 (Update 4)
Updated April 1985 (Update 5)
Updated September 1985 (Update 6)
Updated July 1987 (Update 7)
Updated September 1989 (Update 8)

The University of Michigan Computing Center
Ann Arbor, Michigan

DISCLAIMER

This volume is intended to represent the current state of the Michigan Terminal System (MTS), but because the system is constantly being developed, extended, and refined, sections of this volume will become obsolete. The user should refer to the U-M Computing News, Computing Center Memos, and future updates to this volume for the latest information about changes to MTS.

PREFACE

The software developed by the Computing Center staff for the operation of the high-speed processor computer can be described as a multiprogramming supervisor that handles a number of resident, reentrant programs. Among them is a large subsystem, called MTS (Michigan Terminal System), for command interpretation, execution control, file management, and accounting maintenance. Most users interact with the computer's resources through MTS.

The MTS Manual is a series of volumes that describe in detail the facilities provided by the Michigan Terminal System. Administrative policies of the Computing Center and the physical facilities provided described in other publications.

The MTS volumes now in print are listed below. The date indicates the most recent edition of each volume; however, since volumes are periodically updated, users should check the file *CCPUBLICATIONS, or watch for announcements in the U-M Computing News, to ensure that their MTS volumes are up to date.

Volume 1: The Michigan Terminal System, November 1988
Volume 2: Public File Descriptions, January 1987
Volume 3: System Subroutine Descriptions, March 1989
Volume 4: Terminals and Networks in MTS, July 1988
Volume 5: System Services, May 1983
Volume 6: FORTRAN in MTS, October 1983
Volume 7: PL/I in MTS, September 1982
Volume 8: LISP and SLIP in MTS, June 1976
Volume 9: SNOBOL4 in MTS, September 1975
Volume 10: BASIC in MTS, December 1980
Volume 11: Plot Description System, August 1978
Volume 12: PIL/2 in MTS, December 1974
Volume 13: The Symbolic Debugging System, September 1985
Volume 14: 360/370 Assemblers in MTS, May 1983
Volume 15: FORMAT and TEXT360, April 1977
Volume 16: ALGOL W in MTS, September 1980
Volume 17: Integrated Graphics System, December 1980
Volume 18: The MTS File Editor, February 1988
Volume 19: Tapes and Floppy Disks, March 1989
Volume 20: Pascal in MTS, January 1989
Volume 21: MTS Command Extensions and Macros, April 1986
Volume 22: Utilisp in MTS, May 1988
Volume 23: Messaging and Conferencing in MTS, August 1988

The numerical order of the volumes does not necessarily reflect the chronological order of their appearance; however, in general, the higher the number, the more specialized the volume. Volume 1, for example, introduces the user to MTS and describes in general the MTS operating system, while Volume 10 deals exclusively with BASIC.

The attempt to make each volume complete in itself and reasonably independent of others in the series naturally results in a certain amount of repetition. Public file descriptions, for example, may appear in more than one volume. However, this arrangement permits the user to buy only those volumes that serve his or her immediate needs.

Richard A. Salisbury,

General Editor

PREFACE TO REVISED VOLUME 3

The April 1981 edition reflects the changes that have been made to MTS since October 1976. Some of these changes were described in Updates 1-5 and are incorporated into this revision.

The section "PL/I Library Subroutines" has been deleted from this edition as those subroutines are currently described in MTS Volume 7, PL/I in MTS.

The section "External Symbol Index" has been deleted. This information is now available through the program *SYMBOLS.

The following subroutine descriptions have been added to this edition since Update 5 (April 1980).

CHKPAR
COMMAND
GPRJNO
NPAR
PKEY
RSSAS
TRLCUC, TRUCLC
TRTLC, TRTUC, TRTNONAN

The following subroutines have been deleted from this edition as they are no longer actively supported by the Computing Center. Descriptions of these subroutines may be found in the October 1976 edition of MTS Volume 3, System Subroutine Descriptions, which is available in the Computing Center staff library.

CVTOMR
E7090, D7090, E7090P, D7090P
KEYWRD
TRACER

The following subroutine has been deleted from this edition as it is callable only from internal system programs.

SETFPRIV

The CASECONV subroutine description is now a part of the TRLCUC, TRUCLC subroutine description.

A special edition of this volume has been published for use by systems programmers. This edition contains descriptions of several internal system subroutines which are callable only from system mode or which contain parameters which are only of use to systems programs. A copy of this edition is available in the Computing Center Staff Library.

Contents

USING SUBROUTINE LIBRARIES

The Computing Center maintains a number of subroutine libraries in public files. In addition, the user can construct and use his own libraries.

The loader will selectively load subroutines from both user and system libraries as follows:

(1) All libraries explicitly specified on the $RUN command are processed.

(2) If, after all files explicitly specified on the $RUN command are processed, there remain unresolved subroutine calls, the loader will search implicitly specified libraries if the LIBR option is ON (the default) as follows:

   a. The loader will implicitly search any private libraries specified via the $SET LIBSRCH=FDname command. The default setting for the LIBRSRCH option is OFF, in which case no user libraries are implicitly searched.
   b. If, after implicitly searching all user libraries, there remain unresolved subroutine calls, the system will implicitly search *LIBRARY and the resident system library if the *LIBRARY option is ON (the default).

(3) If, after all implicitly specified libraries have been searched, there remain unresolved subroutine calls, a terminal user will be prompted for more input; a batch user will be given an error return from the loader.

The default settings for LIBR, LIBSRCH, and *LIBRARY are such that, for example, issuing the command

```
$RUN -LOAD+*PL1LIB
```

will cause the loader to go through the following steps:

(1) The object modules in the file -LOAD are loaded and linked together.

(2) Object modules are selectively loaded from *PL1LIB (since it is a library) to resolve external symbols (i.e., subroutine names) from -LOAD.

(3) Finally, if there are still unresolved external symbols, *LIBRARY and the resident system library are searched for the appropriate object modules.

Note that this concatenation can be implicit as well as explicit. Instead of specifying

```
$RUN OBJ+*PL1LIB
```

the user could specify

```
$CONTINUE WITH *PL1LIB
```

as the last line in the file OBJ and then specify

```
$RUN OBJ
```

to get the same effect.

The dynamic loader's library facility consists of four control records, namely LCS, LIB, RIP, and DIR records (named because the records have LCS, LIB, RIP, or DIR, respectively, in columns 2 to 4 of the record). The LCS record causes symbols which are referenced but not yet defined to be defined from a resident system table if they exist there. The LIB record loads selectively the object module which follows it or to which the LIB record points only if the module name has been referenced but not yet defined. The RIP record handles forward references and multiple entry point problems in the one-pass library scan. The DIR record is used to facilitate the loading of modules stored in a sequential file.

A library consists of the object modules the user desires in his library together with the library control records necessary to define the module names, entry points, and references for the selective loading feature of the loader. Although the user can construct such a library himself by inserting appropriate library control records in both his object modules, this task has proven formidable enough with large libraries that a program has been written to analyze the object modules for a library and generate the library complete with all library control records. A description of this program, *OBJUTIL, is given in the section "The Object-File Editor" in MTS Volume 5, <u>System Services</u>. A description of the format of library control records is given in the section "The Dynamic Loader" in MTS Volume 5.

SUBROUTINES LIBRARIES AVAILABLE IN MTS

The following is a list of the public files that contain subroutine libraries:

*LIBRARY

All subroutines that are contained in *LIBRARY are described in this volume except for the IOH subroutines which are described in the section "IOH" in MTS Volume 14, 360/370 Assemblers in MTS.

*PL1LIB
*PL1OPTLIB

These files contain subroutines needed to support PL/I programs. A few of these which were added or modified by the Computing Center are described in MTS Volume 7, PL/I in MTS. The remainder are described in the IBM publications IBM System/360 Operating System PL/I (F) Programmer's Guide, form number GC28-6594, and IBM System/360 Operating System, PL/I Subroutine Library, Computational Subroutines, form number GC28-6590.

*PL360LIB

This file contains subroutines to support the external procedures READ, WRITE, PUNCH, and PAGE for PL360 programs.

*SLIP

The SLIP (Symmetric List Processor) subroutine package is an implementation of Joseph Weizenbaum's IBM 7090 SLIP language. The description of SLIP is given in the section "SLIP" in MTS Volume 8, LISP and SLIP in MTS.

*WATLIB

This file contains WATFOR-coded functions and subroutines for use with WATFIV programs. The description of WATFIV is given in the section "WATFIV" in MTS Volume 6, FORTRAN in MTS.

*CSMPLIB
*GASP
*GPSSLIB
*SIM2LIB

These files contain library modules for use with the CSMP, GASP, GPSS, and SIMSCRIPT2 simulation languages.

*ALGOLLIB
*KDFLIB

These files contain subroutines for use with the ALGOL language.

*SPITLIB

This file contains the execution-time support routines for object programs produced by *SPITBOL.

*PLOTSYS

This file contains the subroutines for use with the Plot Description System (PDS). The description of the Plot Description System is given in MTS Volume 11, Plot Description System.

*IG

This file contains the subroutines for use with the Integrated Graphics (IG) system. The description of IG is given in MTS Volume 17, Integrated Graphics System.

*ALGOLWLIB

This file contains subroutines for use with the ALGOL W language. The description of ALGOL W is given in MTS Volume 16, ALGOL W in MTS.

*APLLIB

This file contains subroutines for use with the General Motors Associative Programming Language (APL).

*XPLIBRARY
*EXPLIB

These files contain subroutines for use with the XPL and extended XPL languages.

*COBLIB

This file contains subroutines for use with the COBOL language.

*PASCALJBLIB
*PASCALJBINCLUDE
*PASCALJBSYSLIB
*PASCALVSLIB
*PASCALVSINCLUDE
*PASCALVSSYSLIB

These files contain subroutines for use with the PASCAL/VS and PASCAL/JB languages.

One subroutine library is available under the Computing Center ID OLD.

OLD:LIBRARY

This file contains subroutines that were once contained in *LIBRARY. These subroutines are no longer supported by the Computing Center.

Several subroutine libraries are available under the Computing Center ID NAAS. These are used for numerical analysis applications. They are the following:

NAAS:NAL

This file contains a package of general numerical analysis subroutines.

NAAS:EISPACK

This file contains a package of eigensystem subroutines developed by the Argonne National Laboratory.

NAAS:FUNPACK

This file contains a package of special function subroutines developed by the Argonne National Laboratory.

NAAS:IMSL

This file contains a package of single-precision subroutines from International Mathematical and Statistical Libraries, Inc.

NAAS:IMSL/D

This file contains a package of double-precision subroutines from International Mathematical and Statistical Libraries, Inc.

NAAS:OLDLIB

This file contains the mathematical subroutines that were once contained in *LIBRARY.

The NAAS and IMSL subroutine packages are fully described in Computing Center Memos 407 and 442.

Several subroutine libraries are available under the Computing Center ID UNSP. They are the following:

UNSP:LIBRARY

This file contains a collection of FORTRAN-callable subroutines.

UNSP:PL1LIB

This file contains a collection of PL/I-callable subroutines.

UNSP:SPITLIB

This file contains a collection of functions callable from SNOBOL4 or SPITBOL programs.

UNSP:LSLIPLIB

This file contains the single-precision version of the SLIP subroutines.

UNSP:DIGLIB

This file contains a device-independent graphics system.

For more detailed information on these subroutine libraries, see the UNSP descriptions in the documentation racks at the Computing Center and NUBS.

The ID UNSP is part of an effort to gather a number of unsupported programs and subroutines into one location. This unsupported software is being made available under UNSP rather than in public files because the Computing Center does not have the resources (people, time, or money) to completely ensure its quality or to provide continuing maintenance. Many of these programs and subroutines represent interim solutions to particular problems which will be replaced with supported software as better solutions are developed.

As the name UNSP suggests, this software is not actively supported by the Computing Center Staff. This means that there are no guarantees to its reliability, performance, or continued availability, no counseling is available beyond that normally provided for user programs, and no rebates will be given for errors caused by the operation of unsupported software. (It should be noted, however, that before any software is made available under UNSP, a member of the Computing Center staff will have done minimal testing and determined that the programs does what it claims to do for the common cases.) The file UNSP:CATALOG may be copied to obtain a list of the programs and subroutines currently available together with a short description and directions for obtaining additional documentation.-

SUBJECT CATEGORIES OF SUBROUTINES

In an effort to aid users in finding subroutines that may be useful in their work, a number of subject categories have been defined. Each category consists of a type of activity a user might be doing. Under each category is listed the name of the appropriate subroutine description, the purpose of the subroutine, and whether the subroutine is callable by an S-type or R-type calling sequence.

Character and Numeric Conversion

ASCEBC,IASCEBC	USASCII to EBCDIC translation	Table
BINEBCD	Binary input to EBCDIC translation	R-type
BINEBCD2	Binary input to EBCDIC translation	R-type
EBCASC,IEBCASC	EBCDIC to USASCII translation	Table
IOH	Numeric input/output conversion	R-type
SIOC	Numeric input/output conversion	S-type
SIOCP	Numeric input/output conversion	S-type
TRLCUC,TRUCLC	Lowercase-uppercase conversion	Table
TRTLC,TRTUC,TRTNONAM	Lowercase-uppercase detection	Table
Translation Routines	Lowercase-uppercase conversion and USASCII-EBCDIC conversion	S-type

Date and Time Conversion

GRGJULDT	Gregorian to Julian date and time	R-type
GRGJULTM	Gregorian to Julian time	R-type
GRJLDT	Gregorian to Julian date and time	S-type
GRJLSEC	Gregorian to Julian time	R-type
GRJLTM	Gregorian to Julian time	S-type
GROSDT	Gregorian to OS date	S-type
GTDJMS	Gregorian to Julian date and time	S-type
GTDJMSR	Gregorian to Julian time	R-type
JLGRDT	Julian to Gregorian date and time	S-type
JLGRSEC	Julian to Gregorian time	R-type
JLGRTM	Julian to Gregorian time	S-type
JMSGTD	Julian to Gregorian date and time	S-type
JMSGTDR	Julian to Gregorian date and time	R-type
JTUGTDR	Julian to Gregorian date and time	R-type

JULGRGDT	Julian to Gregorian date and time	R-type
JULGRGTM	Julian to Gregorian time	R-type
OSGRDT	OS to Gregorian date	S-type
TIME	Get time of day, CPU and elapsed time	S-type
Time Routines	General time and date conversion	S-type

File and Device Usage

ANSI File Routines	File control for FORTRAN programs	S-type
CATSCAN	Scan the system catalog	S-type
CFDUB	Compare FDUB-pointers	S-type
CHGFSZ	Change file size	S-type
CHGMBC	Change number of file buffers	S-type
CHGXF	Change file expansion factor	S-type
CHKACC	Check access to file	S-type
CHKFDUB	Get a FDUB-pointer for a file	S-type
CHKFILE	Determine existence of a file	S-type
CLOSEFIL	Close a file	S-type
CNTLNR	Count number of lines in a file	S-type
CREATE	Create a file	S-type
DESTROY	Destroy a file	S-type
EDIT	Edit a file	S-type
EMPTY	Empty a file	R-type
EMPTYF	Empty a file	S-type
FILEINFO	Get file information	S-type
FNAMETRT	Check for legal file name	Table
FREEFD	Free a file or device	R-type
FSIZE	Determine size required for a file	S-type
FSRF,BSRF	Forward and backspace records in a file	S-type
GDINF	Get file information	S-type
GDINFO	Get file or device information	R-type
GDINFO2	Get file or device information	R-type
GDINFO3	Get file or device information	R-type
GETFD	Get a file or device	R-type
GETFST,GETLST	Get first and last line numbers of a line file	S-type
GFINFO	Get file and catalog information	S-type
LETGO	Periodically unlock and lock a file	S-type
LOCK	Lock a file	S-type
LSFILE	Get locking status information for file	S-type
LSTASK	Get locking status information for task	S-type
NOTE	Remember sequential file pointers	S-type
PERMIT	Permit a file	S-type
PKEY	Push or pop program key	S-type
POINT	Change sequential file pointers	S-type
RENAME	Rename a file	S-type
RENUMB	Renumber a file	S-type
RETLNR	Return line numbers of a file	S-type
REWIND	Rewind a logical I/O unit	S-type

REWIND#	Rewind a file or magnetic tape	R-type
RSSAS	Reset *SOURCE* and *SINK*	S-type
SETFSAVE	Enable or disable file saving	S-type
SETKEY	Set program key for a file	S-type
SETLNR	Set line numbers of a file	S-type
TOUCH	Update the last data-change time for a file	S-type
TRUNC	Truncate a file	S-type
UNLK	Unlock a file	S-type
WRITEBUF	Write file buffers	S-type

FORTRAN Usage

ADROF	Get address of a FORTRAN variable	S-type
ANSI Bit Routines	Bit manipulation for FORTRAN programs	S-type
ANSI File Routines	File control for FORTRAN programs	S-type
Array Management Routines	Array processing for FORTRAN	S-type
ATNTRP	Attention interrupt processing	S-type
Bitwise Logical Functions	FORTRAN bitwise logical functions	S-type
BMS Routines	Bit manipulation for FORTRAN programs	S-type
Character Manipulation Routines	Character processing for FORTRAN	S-type
CHKPAR	Check parameters to a subroutine	S-type
DUMP,PDUMP	Dump storage	S-type
FREAD,FWRITE	Free-format input/output	S-type
FTNCMD	Execute FORTRAN I/O library command	S-type
GDINF	Get file information	S-type
GRJLDT	Gregorian to Julian date and time	S-type
GRJLTM	Gregorian to Julian time	S-type
GTDJMS	Gregorian to Julian date and time	S-type
JLGRDT	Julian to Gregorian date and time	S-type
JLGRTM	Julian to Gregorian time	S-type
JMSGTD	Julian to Gregorian date and time	S-type
LINKF	Dynamic loading	S-type
LOADF	Dynamic loading	S-type
Logical Operators	FORTRAN logical machine operations	S-type
NPAR	Count parameters to a subroutine	S-type
RCALL	R-type call from FORTRAN	S-type
REWIND	Rewind a logical I/O unit	S-type
SIOERR	I/O error processing	S-type
STARTF	Dynamic loading	S-type
TICALL	Timer interrupt processing	S-type
UNLDF	Dynamic unloading	S-type

Input/Output Routines

Blocked I/O Routines		
	Read and write blocked records	S-type
FREAD,FWRITE		
	Free-format input/output	S-type
GUSER	Read from logical I/O unit GUSER	S-type
LIOUNITS	Table of valid logical I/O units	R-type
READ	Read a record	S-type
READBFR	Read without knowing length	R-type
REWIND	Rewind a logical I/O unit	S-type
REWIND#	Rewind a magnetic tape or file	R-type
SCARDS	Read from logical I/O unit SCARDS	S-type
SERCOM	Write on logical I/O unit SERCOM	S-type
SETIOERR	I/O error processing	R-type
SETLIO	Set logical I/O unit	S-type
SIOERR	I/O error processing	S-type
SPRINT	Write on logical I/O unit SPRINT	S-type
SPUNCH	Write on logical I/O unit SPUNCH	S-type
WRITE	Write a record	S-type

Interrupt Processing

ATNTRP	Attention interrupt processing	S-type
ATTNTRP	Attention interrupt processing	R-type
GETIME	Timer interrupt processing	S-type
PGNTTRP	Program interrupt processing	R-type
RSTIME	Timer interrupt processing	S-type
SETIME	Timer interrupt processing	S-type
SETLCL	To set a local time limit	S-type
SPIE	Program interrupt processing	R-type
TICALL	Timer interrupt processing	S-type
TIMNTRP	Timer interrupt processing	R-type
TWAIT	Timer interrupt processing	S-type

Status of User and System

CANREPLY	Terminal or batch status	S-type
CNFGINFO	Get system configuration information	Table
COST	Get cost of current signon	S-type
CUINFO	Change user status information	S-type
GPRJNO	Get user project number	R-type
GUINFO	Get user status information	S-type
GUINFUPD	Update user status information	R-type
GUSERID	Get user ccid	S-type
LOADINFO	Get symbol or address information	S-type

System Utilities

BLOKLETR	Produce block letters	S-type
CALC	Call $CALC routines	S-type
CHARGE	To compute charges for computer resources	S-type
CMD	Execute an MTS command	S-type
CMDNOE	Execute an MTS command without echoing	S-type
COMMAND	Execute an MTS command	S-type
CONTROL	Execute a device support operation	S-type
CRYPT	Encrypt or decrypt data	S-type
DISMOUNT	Dismount a tape	S-type
ERROR	Terminate execution with error	S-type
GRAND	Normally distributed random number	S-type
IBSCH	Binary searching	S-type
KWSCAN	Keyword processing	R-type
MOUNT	Mount a tape	S-type
MTS	Return to MTS command mode	S-type
MTSCMD	Return to MTS and execute a command	S-type
Printer Plot Routines	Produce plots	S-type
QUIT	Signoff user at next MTS command	S-type
SETLIO	Assign logical I/O units	S-type
SETPFX	Set prefix character	S-type
SKIP	Space a magnetic tape	S-type
SORT	Sort and merge records	S-type
SORT2	Sort vectors	S-type
SORT3	Sort vectors	S-type
SPELLCHK	Spelling check	S-type
SRCHI	Binary searching	S-type
SYSTEM	Terminate execution	S-type
URAND	Uniformly distributed random number	S-type
Xerox 9700 Font Routines	Get Xerox 9700 font information	S-type

Virtual Memory Management

DUMP,PDUMP	Dump storage	S-type
FREESPAC	Release storage	S-type
GETSPACE	Acquire storage	S-type
GPSECT, FPSECT, QPSECT	Psect storage management	R-type
LINK	Dynamic loading	R-type
LINKF	Dynamic loading	S-type
LOAD	Dynamic loading	R-type
LOADF	Dynamic loading	S-type
LOADINFO	Get loader table information	S-type
LODMAP	Produce loader map	S-type
SCANSTOR	Scan storage blocks	R-type
SDUMP	Dump storage and registers	R-type
STARTF	Dynamic loading	S-type
STDDMP	Dump storage	R-type

UNLDF	Dynamic unloading	S-type
UNLOAD	Dynamic unloading	R-type
XCTL	Dynamic loading	R-type
XCTLF	Dynamic loading	S-type

CALLING CONVENTIONS

INTRODUCTION

A calling convention is a very rigid specification of the sequence of instructions to be used by a program to transfer control to another program (usually referred to as a subroutine). It is very desirable, although not always practical, to set up only one set of conventions to be used by all programs no matter what language they are written in so that FORTRAN programs may call assembly language programs and so forth. In MTS, the OS type I calling conventions have been adopted as the standard. A complete specification of these standards can be found in the IBM publication, OS/360 System Supervisor Services and Macro Instructions, form number GC28-6646. This description will attempt to bring out the pertinent details of these calling conventions.

Throughout this discussion we will refer to the terms calling program, called program, save area, and calling sequence. The calling program is the program which is in control and wants to call another program (subroutine). The called program is the program (subroutine) which the calling program wants to call. The save area is an area belonging to the calling program which the called program uses to save and later restore general-purpose registers. The save area has a very rigid format and is discussed in more detail later on. A calling sequence is the actual sequence of machine instructions which perform the tasks as specified by the calling conventions.

The facilities that must be provided by the calling conventions are:

(1) Establish addressability and transfer to the entry point.
(2) Pass parameters on to the called program.
(3) Pass results back to the calling program.
(4) Save and restore general-purpose and floating-point registers.
(5) Reestablish addressability and return to the calling program.
(6) Pass a return code (error indication) back to the calling program so it knows how things went.

The remainder of this description will describe the OS type I calling conventions to show how they are used and how the facilities listed above are provided for.

REGISTER AND STORAGE VARIANTS OF CALLS

The OS type I calling conventions actually consist of two very similar calling conventions, referred to as S-type calling conventions and R-type calling conventions. The two differ only in the way parameters and results are passed between the _calling_ and _called_ programs. The _R_ refers to _register_ and the _S_ to _storage_.

The R-type calling conventions utilize the general-purpose registers 0 and 1 for passing parameters and results. This allows only two parameters or results and cannot be generated in higher-level languages such as FORTRAN. Its advantages are that calling sequences are shorter and take less time to set up. These are very popular in lower-level system subroutines such as GETSPACE or GETFD. FORTRAN users needing to call subroutines that utilize R-type calling conventions can use the RCALL subroutine described in this volume.

The S-type calling conventions require a pointer to a vector of address constants called a parameter list (in register 1). Since the parameter list can be of any required length, several parameters can be passed using S-type calling convention. These conventions are used by system subroutines such as SCARDS or LINK and are generated by all function or subprogram references in FORTRAN. Results can be passed back by giving variables in the parameter list new values or via register 0.

PARAMETER LISTS

As stated above, a parameter list is a vector of address constants. The parameter list must be on a fullword boundary and the entries are each four bytes long. The address of the first parameter is the first word of the list, the address of the second parameter the second word of the list, and so on. For example, the parameter list for the FORTRAN statement

```
CALL QQSV(X,Y,Z)
```

might be written in assembly code as:

```
PAR  DC   A(X)      address of X
     DC   A(Y)      address of Y
     DC   A(Z)      address of Z
```

Now this parameter list works well enough when the parameter list for the subroutine is of fixed length, but there is not enough information yet to allow a subroutine to determine the length of the parameter list and hence accept variable-length parameter lists. For this reason there are two types of parameter lists, _fixed-length parameter lists_ as described above, and an extended form of parameter list called a _variable-length parameter list_ which is described next.

Since a standard System/360/370 computer uses 24-bit storage addresses, the left-most byte of an address constant is usually zero. In a variable-length parameter list, bit zero of the left-most byte of the _last_ parameter address constant is set to 1 to show that it is the last item in the list. The example above then would be written as:

```
PAR  DC   A(X)      address of X
     DC   A(Y)      address of Y
     DC   XL1'80'   turn on bit zero
     DC   AL3(Z)    address of Z
```

if it generated a variable-length parameter list, as FORTRAN does. Note though that programs expecting a fixed-length parameter list will work with a variable-length parameter list, provided it is at least as long as the fixed-length list the program is expecting, since it extracts only the address part when it uses the parameters.

REGISTER ASSIGNMENTS

Of the sixteen general-purpose registers, five are assigned for use in the calling conventions. The use of the general registers differs slightly depending upon whether an R- or S-type call is being made. Table 1 specifies exactly what each register is used for during a call.

Notice that it is the called program's responsibility to save and restore registers 2-12 in the save area provided by the calling program. There are two reasons for this. First, only the called program knows how many of the registers from 2-12 it is going to use. Since a register need be saved and restored only if it is actually going to be changed, the called program may be able to save some time by saving and restoring only those registers which it will use. Secondly, the called program requires addressability over the area in which it will save registers upon entry, since any attempt to acquire the address of a save area would destroy some of the registers which are to be saved. Furthermore, the save area should not be a part of the called program since that would prevent it from being reentrant (shareable). This means the calling program should provide the save area in which registers are saved and restored. And so we have the called program saving and restoring registers 2-12 in a save area provided by the calling program.

The calling conventions are quite different with floating-point registers. Since a large percentage of programs do not leave items in floating-point registers across subroutine calls it seems rather wasteful to always save and restore the floating-point registers. So the convention has been established that the _calling_ program must save and restore those floating-point registers that contain items which are wanted. Also, programs that return a single floating-point result quite frequently do so via floating-point register 0.

Register Number	Contents
0	Parameter to be passed in R-type sequences. Result to be passed back in R- and S-type sequences.
1	Parameter to be passed in R-type sequences. Address of a parameter list in S-type sequences.
2-12	Not used as a part of the calling sequence. Must be saved and restored by the called program. The save area is usually used for this.
13	The address of the save area provided by the calling program to be used by the called program.
14	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Address of the entry point in the called program at the time of the call. A return code at the time of the return that indicates to the calling program whether or not an exceptional condition occurred during processing of the called program. The return code should be zero for a normal return or a multiple of four for various exceptional conditions.

Table 1: General-Purpose Register Conventions

RETURNING RESULTS

There are in the calling conventions four ways in which a subroutine can return a result. These are:

(1) Value of result in general-purpose register 0.
(2) Value of result in general-purpose register 1.
(3) Value of a result in floating-point registers (usually FR0).
(4) Value of a parameter from the parameter list changed.

The particular method used depends upon whether the R- or S-type convention is used and whether the called program can be used as a function in arithmetic statements.

The first three methods are used by R-type calling conventions for all returned results. The contents of each of the registers depends upon the particular called program and are described in the subroutine description for each subroutine using the R-type calling conventions.

The first, third, and fourth methods are used by S-type calling conventions for all returned results. The first and third methods are used by function subprograms whose calls can be embedded in FORTRAN statements. The choice of general register 0 or floating-point register 0 depends upon whether the result returned is integer or floating-point, respectively. An example would be a function subprogram called by the statement

```
SUM = ADD(A,B)
```

which adds the floating-point variables A and B and returns the floating-point result in floating-point register 0 which is then assigned to SUM. The fourth method can be used by a subroutine call. The above function subprogram ADD could be changed to a subroutine called by the statement

```
CALL ADD(A,B,SUM)
```

which adds A and B and returns the result in SUM by means of the parameter list instead of using floating-point register 0.

The return code cannot be checked by a FORTRAN program if the subprogram is called by the first or third method. Only the fourth method allows the return code to be checked. This is done by including statement labels in the parameter list indicating the statements to branch to if the corresponding return codes occur. For example, a return from the subroutine ADD when called by the statement

```
CALL ADD(X,Y,SUM,&10)
```

will be to statement number 10 if the return code in general register 15 is 4.

SAVE AREA FORMAT

The save area is an area belonging to the _calling_ program which the _called_ program uses to save and later restore general-purpose registers. The address of the save area is passed to the called program by the calling program via general-purpose register 13. The save area has a very rigid format and is described in Table 2.

Word	Displacement	Contents
1	0	Used by FORTRAN, PL/I, and other beasties for many devious purposes. Don't touch!
2	4	Address of the save area used by the calling program. Forms a backward chain of save areas. Stored by calling program.
3	8	Address of the save area provided by the called program for programs it calls. Forms a forward chain of save areas.
4	12	Return address. Contents of register 14 at time of call.
5	16	Entry point address. Contents of register 15 at time of call.
6	20	Register 0 contents.
7	24	Register 1 contents.
8	28	Register 2 contents.
9	32	Register 3 contents.
10	36	Register 4 contents.
11	40	Register 5 contents.
12	44	Register 6 contents.
13	48	Register 7 contents.
14	52	Register 8 contents.
15	56	Register 9 contents.
16	60	Register 10 contents.
17	64	Register 11 contents.
18	68	Register 12 contents.

Table 2: Save Area Format

There are two things to be noted about the save area format, namely, who sets what parts of the save area and how these areas might be set up. The _calling_ program is responsible for setting up the second word of the save area. This is to contain the address of the save area which was provided when the _calling_ program was called. Although this is technically set up by the calling program as a part of the call, most programs set up the save area they will provide to subroutines they call once and leave its address in general register 13. This process then does not need to be repeated for each call. The _called_ program is responsible for setting up the third through eighteenth words of the save area. The called program usually saves the general registers which it will use as a part of its initialization procedure and restores the registers as a part of the return procedure. Notice that the save area format is amenable to use of the store multiple and load multiple instructions for saving and restoring blocks of registers. All of this will be made clearer in the examples at the end of this section.

Some system subroutines (notably GETSPACE, FREESPAC, and a few others) do not require that a save area be provided for them. For these subroutines general register 13 need not be set up before a call; its contents are preserved by the called subroutine. The subroutines which need no save area are clearly marked as such in the MTS subroutine descriptions. Notice that it is all right to provide a save area to one of these subroutine; it will simply be ignored.

CALLING PROGRAM RESPONSIBILITIES AND CONSIDERATIONS

The calling program is responsible for the following:

(1) Loading register 13 with the address of the save area and setting up the second word of the save area.
(2) Loading register 14 with the return address.
(3) Loading register 15 with the entry point address.
(4) Loading registers 0 and 1 with the parameters in an R-type call or loading register 1 with the address of the parameter list in an S-type call.
(5) Saving floating-point registers, if necessary.
(6) Transferring to the entry point of the subroutine.
(7) Restoring floating-point registers, if necessary.
(8) Testing the return code in register 15, if desired.

After the return from a subroutine, the status of the program will be as follows:

(1) In general, the contents of the floating-point registers will be unpredictable unless saved and restored by the calling program.
(2) The contents of general registers 2 through 14 will be restored to their contents at the time the called program was entered.
(3) The program mask will be unchanged.
(4) The contents of general registers 0, 1, and 15 may be changed.
(5) The condition code may be changed.

Note that general registers 0 and 1 and floating-point register 0 may contain results in the case of R-type subroutine calls or a function subprogram. General register 15 will normally contain a return code, indicating whether or not an exceptional condition occurred during processing of the called program.

CALLED PROGRAM RESPONSIBILITIES AND CONSIDERATIONS

The called program is responsible for the following:

(1) Saving the contents of general registers 2 through 12 and 14 in the save area provided by the calling program. These registers need be saved only if the called program modifies these registers.
(2) Setting up the third word of the save area with the address of the save area, which will be provided to subroutines it will call.
(3) Restoring the contents of general registers 2 through 14 before returning to the calling program.
(4) Restoring the program mask if changed.
(5) Loading general registers 0 and 1 or floating-point register 0 with the result in the case of R-type subroutine calls or a function subprogram.
(6) Loading general register 15 with the return code.
(7) Transferring to the return location.
(8) Saving and restoring the program mask, if necessary.

EXAMPLE CALLING SEQUENCES

This section will describe and give the assembly language statements for the typical machine instructions necessary to implement the calling conventions.

A typical entry point might consist of the following statements:

```
        USING SUBRA,12       12 will be a base register
SUBRA   STM   14,12,12(13)   save registers
        LR    12,15          set up 12 as the base register
        LA    15,SAVE        this is save area provided for others
        ST    15,8(0,13)     set up forward pointer
        ST    13,4(0,15)     set up backward pointer
        LR    13,15          set up for any calls we issue
        LR    11,1           get parameter pointer into nonvolatile
                             register
        .
        .
        .
SAVE    DS    18F            save area we provide for others
```

Inside a subroutine that began with the entry sequence given above, the value of the second parameter in the parameter list could be put into general-purpose register 3 with the following sequence:

```
           .
           .
           .
           L     3,4(0,11)      pick up second adcon from par list
           L     3,0(0,3)       pick up value of parameter
           .
           .
           .
```

Inside a subroutine that began with the entry sequence given above, another subroutine, SUBRB, could be called using the following sequence. Remember that register 13 already points to the correct save area:

```
           .
           .
           .
           LA    1,PARLIST      set up parameter list address
           L     15,=V(SUBRB)   set up entry point address
           BALR  14,15          set  up  return  address and branch to
                                the subroutine
           B     *+4(15)        test return code via a transfer table
           B     AOK            RC=0
           B     BAD1           RC=4
           B     BAD2           RC=8
           .
           .
           .
   AOK     ...                  normal return to here
           .
           .
           .
   PARLIST DC    A(PAR1)        first parameter address
           DC    A(PAR2)        second parameter address
           DC    A(PAR3)        third parameter address
           .
```

Finally, a subroutine that began with the entry sequence given above could return to the program that called it with the following sequence:

```
           LE    0,RESULT       floating point result to FPR 0
           L     13,4(0,13)     use back pointer to get save area
           LM    14,12,12(13)   restore registers
           SR    15,15          indicate a zero return code--no errors
           BR    14             return to what called us
           .
           .
           .
```

It should be pointed out that although the above sequences are typical of the instructions used to implement the calling conventions, many variations are possible.

MACROS FOR CALLING SEQUENCES

There are two sets of macro definitions in the MTS macro library *SYSMAC which can be used to help generate calling sequences. These are the macros SAVE, CALL, and RETURN; and the macros ENTER and EXIT. The more useful of these macros are ENTER, CALL, and EXIT. Besides these there is a set of macros which generate the entire calling sequences for many of the system subroutines and IOH. For details, see the macro descriptions in MTS Volume 14, 360/370 Assemblers in MTS.

The example given above is repeated below using the ENTER, CALL, and EXIT macros.

```
     SUBRA   ENTER 12,SA=SAVE
             LR    11,1
             .
             .
             .
     SAVE    DS    18F
             .
             .
             .
             L     3,4(0,11)
             L     3,0(0,3)
             .
             .
             .
             CALL  SUBRB,(PAR1,PAR2,PAR3)
             B     *+4(15)
             B     AOK
             B     BAD1
             B     BAD2
             .
             .
     AOK     ...
             .
             .
             .
             LE    0,RESULT
             EXIT  0
```

The CALL macro generates its own parameter list, hence the parameter list specified by PARLIST in the original example need not appear in the macro example.

RESIDENT SYSTEM AND *LIBRARY SUBROUTINES

This section contains descriptions of the subroutines that are a part of the resident system or are contained in the public file *LIBRARY.

Each of these subroutines is called with either the standard S-type calling sequence (such as FORTRAN uses) or the R-type calling sequence. Both types of calling sequences are described in the section "Calling Conventions" in this volume.

ADROF

Subroutine Description

Purpose: To return the address of a FORTRAN variable.

Location: *LIBRARY

Alt. Entry: IADROF

Calling Sequences:

FORTRAN: x = ADROF(var)

Parameters:

var is the location of the variable name whose address is to be returned. If the variable name is a character string which is intended to be used as an FDname, it should be terminated with a trailing blank.

Values Returned:

GR0 will contain the address of the variable. In a FORTRAN call, this address will be returned in *x*.

Note: In FORTRAN, ADROF should be declared as an INTEGER*4 function. ADROF is intended for use with RCALL to compute addresses as necessary in calling R-type subroutines (see the RCALL subroutine description in this volume).

Example: FORTRAN:

```
INTEGER*4 RESULT,ADROF
  ...
RESULT = ADROF('FDname ')
```

This example returns the address of the character string "FDname" in the variable RESULT.

ANSI Standard Bit Manipulation Subroutines

Subroutine Description

This set of subroutines contains procedures for bit manipulation with integers and date/time functions as described in ANSI/ISA-S61.1, Industrial Computer System FORTRAN Procedures for Executive Functions, Process Input/Output, and Bit Manipulation, as well as additional bit manipulation functions as described in Military Standard 1753, FORTRAN, DOD Supplement to American National Standard X3.9-1978. Other subroutines described in ANSI/ISA-S61.1, the executive interface and the process input/output function interfaces, do not apply to the MTS environment and thus are not implemented.

These subroutines are intended to allow FORTRAN programs written for other systems that provide subroutines implementing the same standards to be run in MTS with little or no modification, and to facilitate the development in MTS of FORTRAN programs intended for use on such systems.

The following subroutines are available in *LIBRARY:

Subroutine	Function
IOR	Inclusive OR of the bits in two integers.
IAND	Logical AND of two integers.
IEOR	Exclusive OR of two integers.
NOT	Logical complement of an integer.
ISHFT	Shift bits right or left (noncircular).
BTEST	Test a specific bit.
IBSET	Set a bit to one.
IBCLR	Clear a bit to zero.
ISHFTC	Circular shift of some or all of the bits in an integer.
IBITS	Extract a bit substring.
MVBITS	Move bits from one integer to another.
DATE	Return current date.
ANSITM	Return current time.

The ANSITM subroutine is named TIME in the standard. However, since there is a different MTS subroutine named TIME, a different name had to be chosen for the ANSI subroutine. The object-file editor can be used to change calls to TIME to calls to ANSITM (see the ANSITM description for an example).

Although these subroutines were intended for FORTRAN programs in the standard, they may be called from any programming language that uses the standard IBM OS S-type linkage conventions.

The complete description of these subroutines is given in MTS Volume 6, FORTRAN in MTS.

38.2 ANSI Standard Bit Manipulation Subroutines

ANSI Standard File Control Subroutines

Subroutine Description

This set of subroutines contains procedures for file control as described in ANSI/ISA-S61.2, Industrial Computer System FORTRAN Procedures for File Access and the Control of File Contention.

These subroutines are intended to allow FORTRAN programs written for other systems, which provide subroutines implementing the same standards, to be run under MTS with little or no modification, and to facilitate the development under MTS of FORTRAN programs intended for use on such systems.

The following subroutines are available in *LIBRARY:

Subroutine	Function
CFILW	Create a file
DFILW	Destroy a file
OPENW	Open a file
CLOSEW	Close a file
MODAPW	Modify access privileges for an open file
RDRW	Read a record from a file
WRTRW	Write a record to a file

Note: These subroutines only provide for direct access to files.

The following list describes all extensions to and incompatibilities with the standard.

(1) The standards make no specific mention of the handling of calls with invalid parameters. In this implementation, the return code for each subroutine is set to indicate the type of error detected.

(2) File names are not covered by the standards, but left dependant on the processor. These subroutines expect file names to be standard MTS file names, terminated by a blank space. (This can be effected in full accord with the standard by using integer arrays initialized to contain the file names.)

(3) The standards permit concurrently executing programs to write to the same file and allow one program to read a file while a concurrent program is writing to it; under MTS such access is not possible. Therefore, a program requesting write access to a file will receive it only if no other program is accessing the file in any way.

(4) The standards specify that an open file is attached to a particular unit, and use the unit number to identify the file. These subroutines make use of the unit numbers as specified, but do not actually associate the units with the MTS logical I/O units. Thus, it would be possible to have a file open under the ANSI file subroutines, attached to unit 5, and to have a

different file open and attached to MTS unit 5. Note also that MTS logical I/O units run from 0 to 99, while the ANSI subroutines allow the unit number to be any integer.

(5) A file that is open may be destroyed. This might cause an error return if I/O is subsequently attempted to the file.

The complete description of these subroutines is given in MTS Volume 6, FORTRAN in MTS.

Array Management Subroutines

Subroutine Description

Purpose: The array management subroutine (AMS) package permits FORTRAN users to create, extend, and erase 1- and 2-dimensional arrays at execution time.

Location: *LIBRARY

Description: Any program or subroutine which references an array created by AMS must include an appropriate subset of the following statements:

```
LOGICAL*1 $L1(1)
LOGICAL*4 $L4(1)
INTEGER*2 $I2(1)
INTEGER*4 $I4(1)
REAL*4 $R4(1)
REAL*8 $R8(1)
COMPLEX*8 $C8(1)
EQUIVALENCE ($L1(1),$L4(1),$I2(1),$I4(1),$R4(1),
             $R8(1),$C8(1))
COMMON /$/ $I4
```

The above statements establish a set of names called base names, all of which reference the same address in memory.

An ordinary FORTRAN array element is addressed in the form:

array name(index)

An AMS array element is addressed in the form:

base name(array name + index)

where the base name should match the FORTRAN type of the array. For example, an INTEGER*4 FORTRAN array named ALPHA might be referenced as ALPHA(I). An AMS array of the same name and type should be referenced as $I4(ALPHA+I). If the array type is REAL*8, it should be referenced as $R8(ALPHA+I) and so on for the other array types.

Other base names may be used instead, but the above names are recommended as they serve to remind the user of the type of array being referenced. Starting the base names with a dollar sign ($) serves to make references to these arrays conspicuous in the program listing. Base names need not be defined for any array types not used by the program, except that an INTEGER*4 base must be named and passed in COMMON /$/ even if the user creates no INTEGER*4 arrays.

If the above declarations are properly made, then an AMS array may be passed to a subroutine merely by passing its array name, either as an argument or in COMMON.

The user-callable subroutines in AMS are:

Name	Purpose
ARINIT	to initialize AMS
ARRAY	to create a 1-dimensional array
ARRAY2	to create a 2-dimensional array
EXTEND	to extend a 1-dimensional array
XTEND2	to extend a 2-dimensional array
ERASE	to erase a single array
ERASAL	to erase all arrays

All arguments passed to and returned by these routines must be INTEGER*4 values.

AMS calls in turn the MTS subroutines GETSPACE, FREESPAC, IMVC and ADROF.

Note to users who are doing dynamic program loading via LINKF, LOADF, and XCTLF: the storage obtained by AMS will be associated with the highest level program and will not be released until execution is terminated. To release unwanted arrays call ERASE or ERASAL.

Warning: The subroutines will not work properly if called from *WATFIV or *IF.

ARINIT

Purpose: Before any arrays are created, the user must make one and only one call to subroutine ARINIT. This routine initializes AMS, mainly by creating an array called the master table, which is used by AMS to keep track of the user's arrays. The user does not have direct access to the master table.

Calling Sequence:

CALL ARINIT(noar,minc,&s1,&s2,&s3)

Parameters:

noar	an integer in the range 1 to 37449, which specifies the number of arrays the user expects to create during the job. This is an estimate and not an upper limit.
minc	a positive integer specifying the number of arrays that the master table should be extended to accommodate in case it overflows. It will be automatically extended by this amount an indefinite number of times, as needed.

Return Codes:

Normal	Initialization successful.
&s1	No space available to create master table.
&s2	Invalid argument passed (i.e., noar not in range or minc not positive).
&s3	ARINIT already has been called successfully.

Example: CALL ARINIT(100,50,&98,&99)

The master table is created with enough room to handle 100 arrays. Should more arrays be requested, the master table will be automatically extended to accommodate another 50 arrays. If any time during the run the master table should overflow again, it will be extended to accommodate yet another 50 arrays. Control will pass to statement 98 in the user's program if memory space is not available to create the master table. Control will pass to statement 99 if an invalid argument is passed.

ARRAY, ARRAY2

Purpose: To create a 1-dimensional array, ARRAY should be called. To create a 2-dimensional array, ARRAY2 should be called.

Calling Sequences:

```
CALL ARRAY(n,t,d1,&s1,&s2,&s3,&s4)
CALL ARRAY2(n,t,d1,d2,&s1,&s2,&s3,&s4)
```

Parameters:

- t — length in bytes of an array element (1, 2, 4 or 8).
- d1 — a positive integer specifying the number of elements in the 1st dimension of the array.
- d2 — a positive integer specifying the number of elements in the 2nd dimension of the array.

Note: The number of bytes in the array will be t*d1*d2, and this product must be in the range 1 to 1048576.

Values Returned:

n — name of array to be created. The integer value returned will be such that when n is used in the array reference "base name(n+i)", the "i"th element of the array will be referenced (base name = $L1, $L4, $I2, $I4, $R4, $R8 or $C8.)

When creating a 1-dimensional array, argument n may take the form of an undimensioned FORTRAN variable such as N, a FORTRAN array element such as N(J), or an AMS array element such as $I4(N+J). In any case, n must be of type INTEGER*4.

When creating a 2-dimensional array, argument n may not take the form of an undimensioned variable. It must be the first element of either a FORTRAN or an AMS INTEGER*4 array dimensioned at least d2 in length. This is the user’s responsibility.

Return Codes:

- Normal — Array created successfully.
- &s1 — Requested array size out of range.

&s2	No space available for requested array. No new arrays may be created unless some existing arrays are erased.
&s3	Request for extension of master table is greater than 1048576 bytes.
&s4	t is not equal to 1, 2, 4 or 8, or ARINIT was never called.

Examples: The following examples illustrate the creation of 1-dimensional arrays:

```
(1)  CALL ARRAY(N,1,100,&1,&2,&3,&4)

         To reference "i"th element:  $L1(N+I)

(2)  INTEGER*4 N(20)
       ...
     CALL ARRAY(N(J),8,250)

         To reference "i"th element:  $R8(N(J)+I)

(3)  CALL ARRAY(N,4,20)
       ...
     CALL ARRAY($I4(N+J),2,1500)

         To reference "i"th element:  $I2($I4(N+J)+I)
```

Note that by the method of the second and third examples, a series of independent arrays may be created, all referenced by the same name, but by different values of J. This is like having a 2-dimensional array where each column may be of a different type and length and may be created, extended, or erased independently. This is useful if the exact number of arrays required by a program is unknown until determined by execution-time data or calculation.

The following examples illustrate the creation of 2-dimensional arrays:

```
(4)  INTEGER*4 N(20)
       ...
     CALL ARRAY2(N(1),4,200,20)

         To reference element "i,j":  $R4(N(J)+I)

(5)  CALL ARRAY(N,4,20)
       ...
     CALL ARRAY2($I4(N+1),8,3000,20)

         To reference element "i,j":  $R8($I4(N+J)+I)
```

EXTEND, XTEND2

Purpose: To extend a 1-dimensional array, EXTEND should be called. To extend a 2-dimensional array, XTEND2 should be called. This routine allocates new space dimensioned according to the request, moves the contents of the old space to the new space, calculates new name values for the new space, and frees the old space.

Calling Sequences:

```
CALL EXTEND(n,inc1,&s1,&s2,&s3)
CALL XTEND2(n,inc1,inc2,&s1,&s2,&s3)
```

Parameters:

n	name of array to be extended.
inc1	a positive integer or zero specifying the number of array elements to be added to 1st dimension of array.
inc2	a positive integer or zero specifying the number of array elements to be added to 2nd dimension of array.

Note: inc1 and inc2 may not both be zero.

Values Returned:

n	new name value for new space obtained.

Return Codes:

Normal	Array extended successfully.
&s1	Size of extended array is greater than 1048576 bytes.
&s2	No space available for extension of array.
&s3	Invalid argument (i.e., array name not recognized, negative inc1 or inc2, or inc1 and inc2 both zero), or ARINIT was never called.

Examples:

```
CALL EXTEND(ALPHA,500,&9,&10,&11)
CALL EXTEND(BETA,M)
CALL XTEND2($I4(A+1),M,0)
CALL XTEND2($I4(A+1),M,N)
```

Note: When extending a two-dimensional array in the second dimension, the argument n (the array name) must be the first element of an array dimensioned at least d2 in length. If the array containing n is not as long as the new expected value of d2, the array containing n must be extended before the two-dimensional array to which it refers is extended. For example,

```
CALL ARRAY(N,4,20)
 ...
CALL ARRAY2($I4(N+1),8,3000,20)
 ...
CALL EXTEND(N,30)
CALL XTEND2($I4(N+1),0,30)
```

ERASE

Purpose: This routine may be called to erase an array.

Calling Sequence:

CALL ERASE(n,&s1)

Parameters:

n	name of array to be erased.

Values Returned:

n	A value of -1 is returned to enable both the user and AMS to check if an array has been erased.

Return Codes:

Normal	Array erased successfully.
&s1	Array name not recognized, or ARINIT was never called.

Examples:

```
CALL ERASE(X)
CALL ERASE(ABC,&99)
CALL ERASE($I4(XYZ+1),&100)
```

ERASAL

Purpose: This routine may be called to erase all arrays. New arrays may subsequently be created without recalling ARINIT. (In fact, ARINIT should never be called more than once in the same run.)

Calling Sequence:

CALL ERASAL

ASCEBC

Translate Table Description

Purpose: To translate 8-bit ISO ASCII characters into IBM EBCDIC characters. An inverse table (EBCASC) is also available.

Location: Resident System

Alt. Entries: IASCEBC, TRASCEBC, TRIAE

Calling Sequence:

Assembly:
```
L    r,=V(ASCEBC)
TR   d(l,b),0(r)
```

Parameters:

- r — is a general register that will contain the address of the ASCEBC translate table.
- d(l,b) — is the location of the region to be translated. d is the displacement, l is the length of the region in bytes, and b is the base register for the region. This parameter may be given also in an assembly language symbolic format.

Description: The ASCII/EBCDIC translation table is shown on the next several pages. This table is for translating ISO 8859/1 8-bit ASCII characters into IBM Code Page 37 EBCDIC characters used in MTS. This table is also given in the file DOC:ALLCHARTABLE.

See the EBCASC subroutine description for a table to translate from EBCDIC into ASCII.

Example:

Assembly:
```
             L    6,=V(ASCEBC)
             TR   REG(100),0(6)
              .
              .
       REG   DS   CL100
```

FORTRAN:
```
LOGICAL*1 REG(100),TRTAB(256)
COMMON /ASCEBC/TRTAB
 ...
CALL ITR(100,REG,0,TRTAB,0)
```

The above examples will translate the ASCII characters of the 100-byte region at location REG into EBCDIC characters.

The FORTRAN example uses the ITR entry point (see the description of the Logical Operators subroutines in this volume). In addition, a RIP loader record (RIP ASCEBC) must be inserted into the object file to force the loader to resolve the symbol ASCEBC from the low-core symbol table.

ATNTRP

Subroutine Description

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

Location: *LIBRARY

Calling Sequence:

FORTRAN: CALL ATNTRP(flag)

Parameter:

flag is a LOGICAL*4 variable which will be set to .TRUE. when an attention interrupt occurs.

Return Codes:

None.

Description: A call to the ATNTRP subroutine will set the value of _flag_ to .FALSE. and will enable the attention interrupt trap. When an attention interrupt occurs, _flag_ will be set to .TRUE., the trap will be disabled, and execution of the interrupted program will be resumed at the point of the interrupt. It is the responsibility of the FORTRAN program to detect a change in the value of _flag_ and to act accordingly.

One call to ATNTRP allows only one attention interrupt to be intercepted. If it is desired to intercept another attention interrupt, ATNTRP must be called again.

Example: FORTRAN:

```
      LOGICAL*4 FLAG
      CALL ATNTRP(FLAG)
        ...
   10 IF(FLAG) GO TO 20
        ...
      GO TO 10
   20 CONTINUE
```

This example calls ATNTRP to enable the intercept of one attention interrupt. Periodically, the program checks the value of FLAG to determine if an interrupt has occurred; if an interrupt has occurred, a branch is made to statement label 20.

ATTNTRP

Subroutine Description

Purpose: To allow control to be returned to the user on an attention interrupt from a terminal.

Location: Resident System

| Alt. Entries: ATTNT, ATTNTRPS, ATNTPS

Calling Sequences:

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

          CALL ATTNTRPS,(exit,region),VL

FORTRAN:  CALL ATNTPS(exit,region,&rc4)
```

Parameters:

- exit (GR0) should be zero or the location to transfer to if an attention interrupt occurs.
- region (GR1) should should contain the location of a 72-byte save region for storing pertinent information.
- &rc4 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Illegal parameter or no VL bit specified.

Description: A call on the subroutine ATTNTRP sets up an attention interrupt intercept for one interrupt only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of an attention interrupt. When an interrupt occurs and the exit is taken, the intercept is cleared so that another call to ATTNTRP is necessary to intercept the next attention interrupt. When an attention interrupt occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location previously specified. If the exit subroutine returns to MTS (BR 14), MTS will handle the interrupt as if ATTNTRP had not been called originally. This feature allows the user to take brief control of the interrupt before MTS takes complete control of the interrupt. When MTS takes control of the interrupt, execution of the program will be terminated and a message will be printed providing the location of the interrupt.

If GR0 is zero on a call to ATTNTRP, the attention interrupt intercept is disabled. GR1 should be zero, or it should point to a valid save region.

When the attention interrupt exit is taken, the first eight bytes of the save region contain the attention interrupt PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The PSW stored in the savearea is always in BC mode (bit 12 is zero). The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region. The contents of GR0 and GR2 to GR12 are unpredictable.

If on a call to ATTNTRP the first byte of the save region is X'FF', ATTNTRP does not return to the calling program; rather, the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on ATTNTRP, after the first attention interrupt exit is taken, allows the user to set a switch (for example) and to return to the point at which he was interrupted with the attention interrupt intercept again enabled.

Routines called from within an attention interrupt exit routine must be recursive if execution is to be resumed after interrupt processing. The MTS I/O subroutines READ, WRITE, SCARDS, SPRINT, SPUNCH, SERCOM, and GUSER are recursive; the FORTRAN I/O subroutines are not.

The ATTNTRP item of the GUINFO/CUINFO subroutine may be used to save a previous exit address and associated region so that it may be later restored.

A call on the ATTNTRPS or ATNTPS subroutines takes the S-type parameters and loads them into an R-type call on the ATTNTRP subroutine.

Example: In this example, the attention interrupt intercept is enabled for a specified portion of the program. When the interrupt occurs, a branch will be made to the label EXIT where a switch will be set marking the interrupt occurrence. The intercept will be reenabled by a second call to ATTNTRP with the FF flag set and a branch will be made back to the point where the interrupt occurred.

```
         LM   0,1,=A(EXIT,REGION)
         CALL ATTNTRP   The intercept is enabled.
          ...
         SR   0,0
         SR   1,1
         CALL ATTNTRP   The intercept is disabled.
          ...
         USING EXIT,15
EXIT     OI   SW,X'01'
         MVI  0(1),X'FF'
         LA   0,EXIT
         CALL ATTNTRP   The intercept is reenabled.
REGION   DS   18F
SW       DC   X'00'
```

BINEBCD

Subroutine Description

Purpose: To convert from binary card-image format into EBCDIC format.

Location: Resident System

Alt. Entries: BINEB, BINEBCDS, BINEBS

Calling Sequence:

```
Assembly: LA   1,input
          LA   2,output
          CALL BINEBCD

          CALL BINEBCDS,(input,output),VL

FORTRAN:  CALL BINEBS(input,output,&rc4)
```

Parameters:

- input (GR1) is the 160-byte region containing the input binary card image.
- output (GR2) is the 80-byte region to contain the converted EBCDIC form.
- &rc4 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return.
- 4 Illegal parameter or no VL bit specified.

Notes: Illegal characters are not detected and are translated unpredictably.

The binary card-image region is destroyed during the translation process. See the description of BINEBCD2 for a subroutine that does not destroy this region.

Description: A call on the BINEBCDS or BINEBS subroutines takes the S-type parameters and loads them into an R-type call on the BINEBCD subroutine.

Example: Assembly:

```
                LA   1,INPUT
                LA   2,OUTPUT
                CALL BINEBCD
                 .
                 .
        INPUT   DS   CL160           Binary card image
        OUTPUT  DS   CL80            EBCDIC form
```

The binary card image in the region INPUT is converted to EBCDIC format and placed in the region OUTPUT.

BINEBCD2

Subroutine Description

Purpose: To convert from binary card-image format into EBCDIC format.

Location: Resident System

Alt. Entries: BINEB2, BINEBCDS, BINEBS

Calling Sequence:

```
Assembly: LA   1,input
          LA   2,output
          LA   3,wkarea
          CALL BINEBCD2

          CALL BINEBCDS,(input,output,wkarea),VL

FORTRAN:  CALL BINEBS(input,output,wkarea,&rc4)
```

Parameters:

- input (GR1) is the 160-byte region containing the input binary card image.
- output (GR2) is the 80-byte region to contain the converted EBCDIC form.
- wkarea (optional) (GR3) is the location of an 80-byte work area.
- &rc4 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Illegal parameter or no VL bit specified.

Notes: Illegal characters are not detected and are translated unpredictably.

The binary card-image region is not destroyed during the translation process.

Description: A call on the BINEBCDS or BINEBS subroutines takes the S-type parameters and loads them into an R-type call on the BINEBCD2 subroutine.

Example: Assembly:

```
                 LA   1,INPUT
                 LA   2,OUTPUT
                 LA   3,WKAREA
                 CALL BINEBCD2
                  .
                  .
         INPUT   DS   CL160          Binary card image
         OUTPUT  DS   CL80           EBCDIC form
         WKAREA  DS   CL80           Work area
```

The binary card image in the region INPUT is converted to EBCDIC format and placed in the region OUTPUT.

BMS (Bit Manipulation Subroutines)

Subroutine Description

BMS is a subroutine package that enables the user to manipulate bit strings. It was written with the FORTRAN user in mind, so most examples are in FORTRAN. However, these subroutines may be called from any program that uses the standard OS type I (S-type) calling conventions that FORTRAN uses; a few examples are included to illustrate this.

A *bit string* is a region of contiguous bits in the user's storage. It need not begin or end on any of the recognized storage boundaries. To define a bit string to a BMS subroutine, the user passes three parameters: *baseadd*, *bitdisp*, and *bitlen*.

- *baseadd* is a valid address in the user's storage.
- *bitdisp* is a fullword integer containing a displacement in bits from *baseadd* (may be 0 or a positive integer).
- *bitlen* is a fullword integer containing the length of the string in bits (may be 0 or a positive integer).

baseadd and *bitdisp* together determine the beginning of the string in a manner analogous to a base address and a displacement in a 360/370 machine instruction, the difference being that *bitdisp* is a displacement in bits rather than bytes. For example,

baseadd = ALPHA, a fullword variable
bitdisp = 16
bitlen = 8

The bit string defined is the third byte of ALPHA.

```
 ALPHA
┌─────────┬─────────┬─────────┬─────────┐
│ byte 1  │ byte 2  │ byte 3  │ byte 4  │
└─────────┴─────────┴─────────┴─────────┘
 0      7 8      15 16     23 24     31
```

The subroutines are of two types: subroutines and integer-valued functions. The subroutines all have a normal return and an error return. Since they all work the same way, the return codes are summarized here:

Return Codes:

0 Operation successful.
4 Negative parameter passed or wrong number of parameters passed.

FORTRAN users can take advantage of the return code by coding an ampersand followed by a statement number after the last parameter of a subroutine; if the return code is 4, the subroutine will return to the specified statement, rather than to the point from which the subroutine was called.

The subroutines available in the BMS package in *LIBRARY are:

Subroutine	Function
BCLEAR	Clear a bit string to zeros
BSET	Set a bit string to ones
BFLIP	Complement a bit string (NOT)
BCOPY	Copy a bit string to another location in storage
BSWAP	Switch 2 bit strings in storage
BAND	Calculate the logical product (AND) of 2 bit strings
BOR	Calculate the logical sum (OR) of 2 bit strings
BXOR	Calculate the modulo-two sum (XOR) of 2 bit strings
BFETCH	Return a bit string as an integer value
BCOMP	Compare 2 bit strings (<, =, >)
BOOLE	Perform on 2 bit strings the boolean operation defined by a truth table passed as an argument
BINSRT	Insert a substring in a bit string
BDLETE	Delete a substring from a bit string
BSCAN	Find the location in a bit string of a substring
BCOUNT	Count the occurrences of a substring in a bit string

The complete description of these subroutines is given in MTS Volume 6, FORTRAN in MTS.

Bitwise Logical Functions

Subroutine Description

Purpose: These simple functions do the bitwise logical operations which are difficult to state in FORTRAN arithmetic formulas. If their names are prefixed with an "L", they are INTEGER; otherwise, they are declared REAL. The only exception to this rule is that SHFTR and SHFTL must be declared INTEGER or LOGICAL (to prevent unwanted conversions).

Location: *LIBRARY

Functions: AND, LAND, OR, LOR, XOR, LXOR, COMPL, LCOMPL, SHFTR, and SHFTL.

Calling Sequences:

```
 AND      C = AND(A,B)
LAND     IC = LAND(IA,IB)
```

The result has bits on only if the corresponding bits of the arguments are both on.

```
 OR       C = OR(A,B)
LOR      IC = LOR(IA,IB)
```

The result has bits on only if either or both arguments have the corresponding bits on.

```
 XOR      C = XOR(A,B)
LXOR     IC = LXOR(IA,IB)
```

The result has bits on only if the corresponding bits of the two arguments are not the same.

```
 COMPL    B = COMPL(A)
LCOMPL   IB = LCOMPL(IA)
```

The result has all the bits of the argument reversed.

```
SHFTR    IC = SHFTR(IA,IB)
SHFTL    IC = SHFTL(IA,IB)
```

The first argument is shifted right or left by the number of bits specified by the last 6 bits of the second integer argument (i.e., modulo 64). As logical shift functions, they are not equivalent to a division or to a multiplication by a power of two.

Unless otherwise stated, the arguments of the functions may be either REAL or INTEGER provided that they are fullwords (four bytes long).

All of the functions except for XOR can be generated as <u>in-line</u> code by the FORTRAN-H compiler by specifying the XL option (see the section "*FTN Interface" in MTS Volume 6, <u>FORTRAN in MTS</u>, for details). Caution should be exercised in their use. The functions AND, OR, and COMPL are <u>always</u> generated <u>in-line</u> by FORTRAN-H, but their arguments should not be LOGICAL*1 or INTEGER*2 (specification exceptions may occur on System/360s, or speed is drastically reduced on System/370s). The other functions, if generated <u>in-line</u> by FORTRAN-H by specifying the XL option, may take LOGICAL*1 or INTEGER*2 arguments.

Examples:

```
WORD = XOR(WORD,WORD)
```

This example zeros all the bits of the fullword WORD.

```
DATA MASK/Z00FF0000/
SCDBYT = AND(WORD,MASK)
```

This example examines the second byte of the fullword WORD by deleting the other bytes and storing the result into the fullword SCDBYT.

```
LOGICAL*4 SHFTR
IWORD = SHFTR(IWORD,24)
```

This example moves the first byte of the fullword IWORD into the fourth byte position and leaves the other bytes zero.

```
      DIMENSION CHAR(4)
      READ (5,4) (CHAR(I),I=1,4)
4     FORMAT(4A1)
      DATA MASK/ZFF000000/
      WORD = 0.
      DO 6 I=1,4
6     WORD = OR(WORD,SHFTR(AND(CHAR(I),MASK),(I-1)*8))
```

This example packs four characters into one word.

Blocked Input/Output Routines

Subroutine Description

Purpose: To read and write blocked records consisting of one or more fixed-length logical records.

Location: *LIBRARY

Entry Points: The blocked input/output routines have the following entry points: QGETUCB, QOPEN, QCLOSE, QGET, QPUT, QFREEUCB, and QCNTRL.

Description: These routines will read and write blocked input/output records consisting of one or more fixed-length logical records. All input/output requests are made for logical records; the routine handles record blocking and deblocking automatically. These routines are intended for use with magnetic tape records although they are not restricted to magnetic tapes. More than one input/output file or device may be handled at one time. The type of processing done by these routines is similar to that done by the Queued Sequential Access Method (QSAM) within OS, and for this reason they are sometimes referred to as the MTS QSAM routines. They should not be confused with the OS routines of the same name because the MTS routines provide only a subset of the features of the OS routines.

Several error messages can be generated. Each of these begins with the prefix:

QSAM ERROR: <FDname>

which will be abbreviated as "•••".

The error messages which can be generated by each routine will be listed with that routine in the descriptions which follow.

Some of the error messages will be followed by another message giving an error comment produced by a DSR (device support routine). These will be of the form

message

where "message" is the DSR message.

If the subroutine ERROR is called by these routines, a $RESTART command will cause an RC=4 return.

QGETUCB

Purpose: To acquire a file or device which will be used by the blocked input/output routines and generate a table of control information for that file or device. This table is referred to as the UCB (Unit Control Block).

Alt. Entry: QGTUCB

Calling Sequences:

Assembly: CALL QGETUCB,(name,ptr)

FORTRAN: CALL QGTUCB(name,ptr,&rc4)

Parameters:

<u>name</u> is the location of the name of the file or device which is to be used by the blocked input/output routines ending with a blank or a zero-level comma. The name may not be longer than 256 characters. If the name begins with the character X'00', it is assumed to be a four-byte FDUB-pointer or logical I/O unit number for the file or device.

<u>ptr</u> is the location of a word in which the pointer to the UCB will be placed.

<u>rc4</u> (optional) is a statement label to transfer to if a nonzero return code occurs. .

Return Codes:

0 Successful return. The file or device was acquired and can now be used by the other blocked input/output routines.

4 The file or device could not be acquired properly from MTS. The subroutine GETFD or GDINFO returned a nonzero return code.

Messages: ··· COULD NOT BE ACQUIRED FROM MTS.
··· ERROR FREEING GDINFO VECTOR.

Description: A chain of all UCBs acquired thus far is searched to see if this file or device has been set up before. If so, the UCB pointer is returned immediately. Otherwise, a UCB is built and added to the chain, a pointer to it is returned, GETFD and GDINFO are called for the file or device, and pertinent information is set up in the UCB. The comparison is performed on the full name given, that is, F and F(1,10) are considered different files or devices.

QOPEN

Purpose: To prepare a file or device which has been acquired by QGETUCB for blocked input/output operations.

Assembly: CALL QOPEN,(ptr,key,num,len)

FORTRAN: CALL QOPEN(ptr,key,num,len,&rc4)

Parameters:

- _ptr_ is the location of a word containing a UCB pointer as returned by QGETUCB.
- _key_ is the location of a fullword integer which indicates whether information is to be read or written:
  - 1 Information is to be written.
  - 2 Information is to be read.
  - 5 Information is to be written using previous _num_ and _len_ values.
  - 6 Information is to be read using previous _num_ and _len_ values.
- _num_ is the location of the fullword integer maximum number of logical records per physical record.
- _len_ is the location of the fullword integer length of each logical record (in bytes).
- _rc4_ (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return. The file or device can now be read via QGET (if _key_ is 2 or 6) or written via QPUT (if _key_ is 1 or 5).
- 4 The file or device is already open, or _key_ is not 1, 2, 5, or 6, messages 1, 2, 4, 5, or 7 have occurred, or the physical record length for output is larger than the maximum possible output record length returned by GDINFO.

ERROR:

The subroutine ERROR is called if messages 3 or 6 are printed.

Messages:

1 ••• IS ALREADY OPEN. IT CAN'T BE OPENED TWICE.
2 ••• READ/WRITE SPECIFICATION INCORRECT IN CALL TO OPEN.
3 ••• INCORRECT FORMAT ON LABELED TAPE.
4 ••• ATTEMPT TO CHANGE FORMAT WHILE OPEN.
5 ••• MAXIMUM RECORD LENGTH TOO LARGE.
6 ••• CONTROL COMMAND REJECTED.

The control command was rejected by the tape device support routines; this message may be followed by an error message from the tape device support routines.

7 ••• HAS NOT BEEN SUCCESSFULLY ACQUIRED BY QGETUCB.

Description: The parameters are checked for consistency. The information from the parameters is placed in the UCB. The largest possible physical record length is computed, and a buffer of that length is acquired. If the device is a magnetic tape, blocking will be turned on in the tape DSR and the format will be set to

FB(<u>num</u>*<u>len</u>,<u>len</u>)

unless this is a call to read a labeled tape, in which case, QOPEN will check that the format is F or FB with the logical record length equal to <u>len</u>. If it is, it will not be changed; if it is not, an error message will be printed. Otherwise, if this is a call to write to a device other than a tape, the maximum physical record length for output is checked against the maximum possible output record length as returned by GDINFO. The maximum physical record length is computed as the logical record length times the maximum number of logical records per physical record.

QGET

Purpose: To acquire the next logical record from a file or device which has been opened as an input file or device via QOPEN.

Calling Sequences:

Assembly: CALL QGET,(area,ptr)

FORTRAN: CALL QGET(area,ptr,&rc4)

Parameters:

- _area_ is the location of an area in which the next logical record will be stored (input area).
- _ptr_ is the location of a word containing a UCB-pointer as returned by QGETUCB.
- _rc4_ (optional) is a statement label to transfer to if a nonzero return code is encountered.

Return Codes:

- 0 Successful return. The next logical record has been placed in the input area.
- 4 End-of-file. The input area is sprayed with the character having FF as its hexadecimal representation. This corresponds to the 12-11-0-7-8-9 punched card code.

ERROR:

The subroutine ERROR is called if any of the messages below are printed.

Messages:

- ••• USED IN GET ALTHOUGH NOT OPENED AS AN INPUT FILE.
- ••• USED IN GET ALTHOUGH END-OF-FILE INDICATION GIVEN.
- ••• INPUT RECORD IS LONGER THAN MAXIMUM SPECIFIED.
- ••• RETURN CODE GREATER THAN 4 FROM READ IN GET.

  This message may be followed by an error message from the input device support routine.

- ••• TAPE INPUT LENGTH WRONG.

Description: Physical records are read from the file or device as required. Each physical record is broken into one or more logical records of the length specified in the call upon QOPEN. The last logical record in a physical record may actually be shorter than the length of a logical record. In that case it is padded out with blanks. If there are no more logical records, the input area is sprayed with the character having FF as its hexadecimal representation. All necessary indices are maintained in the UCB.

If the device is a magnetic tape, the data is moved directly into area by the magnetic-tape routines and no deblocking is done by QGET since QOPEN has turned blocking on in the magnetic-tape routines.

QPUT

Purpose: To write the next logical record to a file or device which has been opened as an output file or device via QOPEN.

Calling Sequences:

Assembly: CALL QPUT,(area,ptr)

FORTRAN: CALL QPUT(area,ptr,&rc4)

Parameters:

- area is the location of the area in which the next logical record is stored (output area).
- ptr is the location of a word containing a UCB-pointer as returned by QGETUCB.
- rc4 (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return. The next logical record has been placed to the current physical record.
- 4 File or device appears to be full (RC=4 from WRITE).

ERROR:

A message is printed and the subroutine ERROR is called if the file or device has not been opened for output via the subroutine QOPEN or if a return code greater than 4 was received from WRITE while writing out a physical record.

Messages:
- ··· USED IN QPUT ALTHOUGH NOT OPENED AS AN OUTPUT FILE.
- ··· APPEARS TO BE FULL. (RC=4 FROM WRITE)
- ··· ERROR WHILE WRITING.

This message may be followed by an error message from the output device support routine.

Description: Each logical record presented by a call upon QPUT is placed into a buffer. When the buffer becomes full, it is written out as one physical record. All buffers will contain the maximum number of logical records specified in the call to QOPEN except the last buffer, which will be truncated if it is only partially full when QCLOSE is called. All necessary indices are maintained in the UCB.

If the device is a magnetic tape, the data is written directly from _area_ and is blocked by the magnetic-tape routines.

QCLOSE

Purpose: To terminate blocked input/output operations on a file or device which has been opened via QOPEN. If the file or device was used for output and a partial buffer of logical records for it is present, it is written out as a part of the closing procedure.

Calling Sequences:

Assembly: CALL QCLOSE,(ptr)

FORTRAN: CALL QCLOSE(ptr)

Parameters:

ptr is the location of a word containing a UCB pointer as returned by QGETUCB for the file or device to be closed. The word should contain a zero if all the currently open files or devices are to be closed.

Return Codes:

0 All returns are successful even though some error messages may have been printed.

Messages:

··· APPEARS TO BE FULL. (RC>4 FROM WRITE)
··· FISHY RETURN FROM FREESPAC.
··· ERROR WHILE WRITING.

This message may be followed by an error message from the output device support routine.

Description: If the file or device was used for output and a partial buffer of logical records for it is present, it is written out. All information in the UCB is reset to the normal state of an unopened file or device. The file or device is available for use and can be reopened or positioned.

Note: No tape mark is written when an output file is closed. If the tape is repositioned (e.g., rewound), a tape mark will be written by the magnetic-tape routines.

QFREEUCB

Purpose: To free a file or device which has been acquired via a call to QGETUCB.

Alt. Entry: QFRUCB

Calling Sequences:

Assembly: CALL QFREEUCB,(ptr)

FORTRAN: CALL QFRUCB(ptr,&rc4)

Parameter:

ptr is the location of a fullword containing the UCB-pointer (such as returned by QGETUCB) for the file or device to be released.

rc4 (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return. The file or device was closed and the UCB was released.

4 The UCB-pointer was not found. The file was not closed.

Messages: ••• ERROR RETURN FROM "FREEFD".
••• ERROR RETURN FROM FREESPAC IN QFRUCB.

Description: The chain of all UCBs acquired is searched for the UCB specified by ptr. If it is found, QCLOSE is called using that UCB; then, the UCB is deleted from the chain and released. Any subsequent operations on this file or device must be preceded by a call to QGETUCB in order to reallocate its UCB.

QCNTRL

Purpose: To position or write tape marks on a magnetic tape which has been acquired for use by the blocked input/output routines. To rewind a file or device.

Calling Sequences:

Assembly: CALL QCNTRL,(ccon,ptr)

FORTRAN: CALL QCNTRL(ccon,ptr,&rc4)

Parameters:

- ccon is the location of the three-byte control command used to perform the function required, or a halfword length followed by a control command of that length (see the section "Magnetic Tapes" in MTS Volume 19, Tapes and Floppy Disks.
- ptr is the location of a word which contains a UCB-pointer as returned by QGETUCB.
- rc4 (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return. Operation was accepted by the tape device support routines.
- 4 Any error condition producing one of the error messages below (except the message ERROR RETURN FROM CONTROL OPERATION (RC>4)).

ERROR:

The subroutine ERROR is called if the message ERROR RETURN FROM CONTROL OPERATION (RC>4) is printed.

Messages:

- ••• CANNOT BE POSITIONED BECAUSE IT IS OPEN.
- ••• CANNOT BE POSITIONED BECAUSE IT IS NOT A TAPE.
- ••• DOES NOT HAVE A FDUB AND SO CAN'T BE POSITIONED.
- ••• RC=4 FROM CONTROL OPERATION. TAPE IS FULL.
- ••• ERROR RETURN FROM CONTROL OPERATION (RC>4).

  This message may be followed by an error message from the tape device support routine.

- ••• CANNOT BE POSITIONED BECAUSE NEVER ACQUIRED BY QGETUCB.
- ••• CANNOT BE REWOUND.
- ••• RC>0 FROM "REWIND#".

Description: If the request is "REW", the information returned by GDINFO is checked to be sure the file or device can be rewound. If it can, REWIND# is called to rewind the file or device. For all other requests, the device must be a tape, and the operation is performed by calling the magnetic-tape routines.

BLOKLETR

Subroutine Description

Purpose: To convert a character string into block letters.

Alt. Entry: BLKLTR

Location: Resident System

Calling Sequences:

Assembly: CALL BLOKLETR,(chars,linct,output,flen)

FORTRAN: CALL BLKLTR(chars,linct,output,flen)

Parameters:

- _chars_ is the location of the character string to be converted into block letters.
- _linct_ is the location of a fullword integer with a value between 1 and 12. This specifies which of the twelve lines of the block letter is to be produced on this call.
- _output_ is the location of the output region in which the subroutine will build the resultant output line. It must be of size equal to 14 times the length of _chars_.
- _flen_ is the location of a fullword integer specifying the length of _chars_.

Return Codes:

None.

Description: The characters generated are those of the 029 keypunch character set (PL/I character set plus ¢, !, and ") and the lowercase letters. Any other "characters" in the input string are converted into blanks. The block characters produced are 12 characters wide by 12 rows high and are spaced apart by 2 blank columns. The block characters are composed of the character in question--that is, in a block "ABC", the block A is made up of As, the B of Bs, and the C of Cs. This subroutine produces _one_ of the twelve output rows on each call (specified by the _linct_ parameter). It prints nothing--it only performs the conversion. In order to produce the complete block character string, the subroutine must be called twelve times.

Examples: Assembly:

```
                 SR    8,8
        LP       LA    8,1(,8)
                 ST    8,LINCT
                 CALL  BLOKLETR,(CHARS,LINCT,OUTPUT,FLEN)
                 SPRINT OUTA,OLEN
                 C     8,=F'12'
                 BL    LP
                  .
                  .
        CHARS    DC    C'ABC'
        FLEN     DC    F'3'
        LINCT    DS    F
        OLEN     DC    Y(3*14+1)
        OUTA     DC    C' '
        OUTPUT   DS    CL80
```

FORTRAN:

```
      DATA CHARS/'ABC'/
      LOGICAL*1 OUTPUT(42)
      DO 2 J=1,12
      CALL BLKLTR(CHARS,J,OUTPUT,3)
    2 WRITE (6,100) OUTPUT
  100 FORMAT(' ',42A1)
```

These examples convert the character string ABC into block letters. The output will appear as

```
 AAAAAAAAAA   BBBBBBBBBBB    CCCCCCCCCC
AAAAAAAAAAAA  BBBBBBBBBBBB  CCCCCCCCCCCC
AA        AA  BB        BB  CC        CC
AA        AA  BB        BB  CC
AA        AA  BB       BB   CC
AAAAAAAAAAAA  BBBBBBBBBB    CC
AAAAAAAAAAAA  BBBBBBBBBB    CC
AA        AA  BB       BB   CC
AA        AA  BB        BB  CC
AA        AA  BB        BB  CC        CC
AA        AA  BBBBBBBBBBBB  CCCCCCCCCCCC
AA        AA  BBBBBBBBBBB    CCCCCCCCCC
```

CALC

Subroutine Description

Purpose: To allow program access to the $CALC command routines.

Location: Resident System

Calling Sequences:

Assembly: CALL CALC,(sws,inparm,outparm),VL

FORTRAN: CALL CALC(sws,inparm,outparm,&rc4,&rc8,&rc12)

Parameters:

sws is the location of a fullword (INTEGER*4) of switches assigned as follows:

Bit 31: 0 - release CALC internal storage on return
1 - do not release internal storage, thus allowing reuse of the same invocation on subsequent calls

Bit 30: 0 - evaluate one expression and return
1 - remain in CALC mode until a RETURN, MTS, STOP command, or an end-of-file is encountered

Bit 29: 0 - _inparm_ is the location of a halfword (INTEGER*2) input length followed by the character string to be used as input
1 - _inparm_ is the location of an input routine

Bit 28: 0 - no output other than FR0 (floating register zero) is desired
1 - character output is desired

Bit 27: 0 - _outparm_ is the location of a halfword (INTEGER*2) output length followed by an output region
1 - _outparm_ is the location of an output routine

inparm (optional) is one of the following:
(a) the location of a halfword (INTEGER*2) length followed by a character input line,
(b) the location of an input routine which

will be called via the standard I/O subroutine call for input to CALC, or
(c) 0 or omitted, which means use SCARDS for input regardless of bit 29 setting.

<u>outparm</u> (optional) is one of the following:
(a) the location of a halfword (INTEGER*2) length followed by a character output region (the length must be the maximum length of the region and will be replaced by the actual length of the resulting character string output),
(b) the location of an output routine which will be called via the standard I/O subroutine call for output from CALC, or
(c) 0 or omitted, which means use SPRINT for output regardless of bit 27 setting.

<u>rc4,...,rc12</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Values Returned:

FR0 contains the value of the last successfully evaluated expression on return. This allows CALC to be used as a double-precision (REAL* 8) function-type FORTRAN subprogram.

Return Codes:

0 Successful return.
4 The last expression evaluated generated an error message.
8 The output field provided was of insufficient length for the output.
12 Internal CALC subroutine error--consult the Computing Center staff.

Description: The CALC subroutine allows the user to invoke the $CALC command routines to evaluate one or more character arithmetic expressions. The switch settings control the options available concerning input, output, and mode of operation.

The first two switches (bits 31 and 30) control the mode of operation, i.e., whether or not to allow reuse of this invocation of CALC and whether or not to stay in CALC mode. Note that it is necessary to retain the CALC internal storage if variable values are to be preserved on subsequent calls to the CALC subroutine.

The next switch (bit 29) controls the mode of input, whether the expression is obtained from a given string or is obtained by a subroutine call. If <u>inparm</u> is 0 or omitted, then the input is read from SCARDS. If <u>inparm</u> is omitted, then _outparm_ also must be omitted, forcing input to be read from SCARDS and output, if any, to be written on SPRINT. If _inparm_ specifies an input string (bit 29 is 0) and CALC is to remain in CALC mode (bit 30 is 1), then any additional input is read from SCARDS.

The next two switches (bits 28 and 27) control the mode of output. If no output is specified, the subroutine is assumed to be called as a function with its only output value returned in FR0. If _outparm_ is 0 or omitted, the value of the expression is written in character form on SPRINT. If _outparm_ is the location of an output string, the result is placed in character form in the specified location and the length is modified to the length of the resulting string. If _outparm_ is the location of an output string and CALC remains in CALC mode (bit 30 is 1), then all output will be written in the location provided.

For further information on the $CALC command, see the $CALC command description in MTS Volume 1, _The Michigan Terminal System_.

Examples: FORTRAN:

```
          REAL*8 X,CALC
            ...
          X=CALC(0)
          PRINT 100,X
    100   FORMAT(1X,'X=',E24.18)
```

In the above example, one expression will be evaluated. The expression will be read from SCARDS and there will be no output other than that produced by the PRINT statement.

```
          INTEGER*2 IN(5)/7,'SQ','RT','(2',')'/
          INTEGER*2 OUT(11)/20/
            ...
          CALL CALC(8,IN,OUT,&100,&200,&300)
            ...
    100   PRINT 1
    1     FORMAT(1X,'BAD EXPRESSION')
            ...
    200   PRINT 2
    2     FORMAT(1X,'INSUFFICIENT OUTPUT LENGTH')
            ...
    300   PRINT 3
    3     FORMAT(1X,'CALC SYSTEM ERROR')
```

In the above example, one expression will be evaluated and it will come from the array IN. The result will be produced in character form in the array OUT. The switch value of 8 specifies that bit 28 of the switch word is 1 and all other bits are 0.

```
EXTERNAL INRTE,OUTRTE
 ...
CALL CALC(30,INRTE,OUTRTE)
```

In the above example, expressions will be evaluated until the occurrence of RETURN, MTS, STOP, or an end-of-file as input. Input is returned from the subroutine INRTE and character output is written by calling the subroutine OUTRTE. The switch value of 30 specifies that bits 27, 28, 29, and 30 are 1 and all other bits are 0.

CANREPLY

Subroutine Description

Purpose: To determine whether a program can process interactive responses.

Location: Resident System

Alt. Entry: CREPLY

Calling Sequences:

Assembly: CALL CANREPLY

FORTRAN: CALL CREPLY(&rc4)

Parameters:

rc4 (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Yes
4 No

Description: The CANREPLY subroutine determines whether or not the program can process interactive responses. A program may process interactive responses if

(1) it is running directly in conversational mode, or
(2) it is a job server program (the GUINFO SERVER item is 1) and the GUINFO SRVREPLY item is 1.

A program may not process interactive responses if

(1) it is running in batch mode, or
(2) it is a job server program (the GUINFO SERVER item is 1) and the GUINFO SRVREPLY item is 0.

Example: Assembly:

```
CALL CANREPLY
LTR  15,15
BNE  BATCH
```

FORTRAN:

```
CALL CREPLY(&100)
```

The above two examples branch to the specified statement label if the user is running in batch mode.

CATSCAN

Subroutine Description

Purpose: To scan the file catalog.

Location: Resident system

Calling Sequences:

Assembly: CATSCAN,(catname,flags,type,name,workptr),VL

FORTRAN: CATSCAN(catname,flags,type,name,workptr,
&rc4,...,&rc16)

Parameters:

catname is the location of the catalog name to scan (if _flags_ bit 23 is set) or a pattern to scan for (if _flags_ bit 22 is set). The format is halfword length followed by the character string.

flags is the location of a fullword of flags as follows:

bit 22 - set if the _name_ parameter is a name pattern. The scan returns only those entities whose name matches the pattern (ignored if _workptr_ is not zero).

bit-23 - set if the _name_ parameter is the name of a catalog to be scanned (ignored if _workptr_ is not zero).

bit 30 - set if the scan was aborted; any storage acquired by CATSCAN is released (this is done automatically when the scan is completed as indicated by return code 4).

bit 31 - return information on the current entity. This allows for a rescan when the name of the entity is larger than the allocated region (see the _name_ parameter below).

All other bits are reserved and must be 0.

type is the location of the type of the entity as follows:

1 - File

Other values are reserved for future use.

name — is the location of the catalog entity name. This value is set by name to be the name of the entity found in the catalog. The format is a fullword maximum length (set by the caller), a fullword actual length of the name (set by CATSCAN), and the text comprising the entity name. If the maximum length specified is less than the actual length, the entity name is truncated and return code 8 is given. CATSCAN can then be called again with a new (larger) region and with flag bit 31 set in order to obtain the untruncated entity name.

workptr — is the location of a fullword used by CATSCAN to store a pointer to the CATSCAN private workarea. This workarea is not accessible to the user. This pointer is should be initialized to zero prior to the first call to CATSCAN. CATSCAN will zero this pointer when the work area is released either by user request (flags bit 30 set on call) or when the scan is completed (return code 4).

rc4,...,rc16 are statement labels to transfer to if a nonzero return code occurs.

Return codes:

- 0 Successful return.
- 4 Scan completed with no entity returned, workarea released.
- 8 The entity name was truncated.
- 12 workptr is invalid or other parameter error.
- 16 Internal error.

Description: The CATSCAN subroutine scans the system catalog for entities either in the specified catalog (if flag bit 23 is set) or for entities whose names match the specified pattern (if flag bit 22 is set). The first call to CATSCAN (with workptr set to zero) returns information about the first entity found and sets workptr for future calls. CATSCAN can then be called repeatedly with this workptr to return information for the next entity found. When no more entities are found, CATSCAN resets workptr to zero and returns with the return code set to 4.

The CATSCAN workptr can be used in call to the FILEINFO subroutine to obtain more information about the entity provided that entity is a file (currently the only possibility).

CFDUB

Subroutine Description

Purpose: To determine whether two FDUB-pointers, logical I/O unit numbers, or logical I/O unit names refer to the same file or device.

Location: Resident System

Calling Sequences:

Assembly: CALL CFDUB,(fdub1,fdub2)

FORTRAN: CALL CFDUB(fdub1,fdub2,&rc4,&rc8)

Parameters:

- _fdub1_ is the location of a fullword FDUB-pointer (such as returned by GETFD), a fullword-integer logical I/O unit number (0 through 19), or a left-justified 8-character logical I/O unit name.
- _fdub2_ is the location of a fullword FDUB-pointer (such as returned by GETFD), a fullword-integer logical I/O unit number (0 through 19), or a left-justified 8-character logical I/O unit name.
- _rc4,rc8_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

- 0 _fdub1_ and _fdub2_ refer to the same file or device (with possibly different modifiers or line number ranges).
- 4 _fdub1_ and _fdub2_ refer to different files or devices.
- 8 _fdub1_ and/or _fdub2_ is illegal.

Note: If either _fdub1_ or _fdub2_ (or both) is a member of an explicit or implicit concatenation of files and/or devices, the CFDUB subroutine will use the current member of the concatenation when making the comparison.

Example: Assembly:

```
                 CALL  CFDUB,(UNITA,UNITB)
                 LTR   15,15
                 BNE   ERROR
                  .
                  .
           UNITA DC    C'SPRINT  '
           UNITB DC    C'SPUNCH  '
```

This example checks whether the logical I/O units SPRINT and SPUNCH refer to the same file or device.

FORTRAN:

```
CALL CFDUB(5,6,&4,&8)
```

This example checks whether the logical I/O units 5 and 6 refer to the same file or device.

Character Manipulation Routines

Subroutine Description

Purpose: To provide character manipulation capability for FORTRAN programs.

Location: *LIBRARY

Entry Points: The character manipulation routines have the following entry points: BTD, COMC, DTB, EQUC, FINDC, FINDST, IGC, LCOMC, MOVEC, SETC, TRNC, TRNST.

Description: The subroutines described in this section make use of the character orientation of the System/360/370 and the fact that each character can be referenced in a LOGICAL*1 array in a FORTRAN program. Subroutines are available for searching for characters or character strings, ignoring characters, translating characters or character strings, moving characters, and comparing character strings. All of these subroutines are written in 360-assembler language. It is possible to write FORTRAN equivalents of each, but at the expense of both CPU time and virtual memory space.

Four of the routines, FINDC, FINDST, IGC, and TRNST, return a position in a LOGICAL*1 array as an argument. In order that this position be relative to the start of the array, these routines have a slightly more cumbersome calling sequence than the other routines. This approach was dictated by the fact that routines which return positions relative to the start of a search (which may not be the start of an array) result in many programming errors due to misunderstandings about the positions returned.

Three of the routines, FINDC, IGC, and TRNC, search for characters. In order for the search to be carried out, an initialization step, which may take more CPU time than the search itself, is made. Since the initialization is the same for any given set of characters or character string, these routines allow the user to indicate whether the same characters are to be used again. If the expression indicating the number of characters is set to zero, the same characters given on the last nonzero call will be used. This saves repeating the initialization step. Users should try to take advantage of this in their programs.

While the subroutines were designed with the use of LOGICAL*1 variables in mind, knowledgeable users can, in fact, use them to manipulate characters stored in any type of FORTRAN variable.

These routines typically require a fraction of a millisecond of CPU time. This depends a great deal on the number of characters involved, but timings greater than one-half millisecond are rare. The virtual memory required averages about 250 bytes per routine.

The following terms are used in the subroutine descriptions that follow:

array variable

The name of a dimensioned variable or element of a dimensioned variable.

INTEGER expression

Any valid INTEGER constant (e.g., 10), variable name (e.g., I), or arithmetic expression (e.g., I+3, 4*K+12).

LOGICAL*1 character array

A dimensioned LOGICAL*1 variable containing character information.

BTD

Purpose: To convert FORTRAN INTEGER numbers into numeric character strings.

Calling Sequence:

FORTRAN: CALL BTD(integer,to,cnumb,dnumb,fill,&err)

Parameters:

integer	is an INTEGER expression giving the number to be converted.
to	is a LOGICAL*1 array variable indicating the position at which the first character is to be stored.
cnumb	is an INTEGER expression giving the number of characters in the string. cnumb should be ≤ 12 and ≥ 0. If cnumb=0, then the number of characters will be the number of significant digits in integer plus one for the sign if integer is negative. If cnumb>12, the characters will be right-justified in the 12 positions starting with to and a RETURN 1 will be taken.
dnumb	is an INTEGER variable which will be set to the number of significant digits in integer (plus one if the sign is negative).
fill	is a LOGICAL*1 character variable, or a Hollerith literal, giving a character to be used to replace leading zeros in the string.
err	(optional) is the number of a FORTRAN statement to transfer to if cnumb>12.

Comments: After a call to BTD, dnumb>cnumb implies a loss of significant digits in the conversion.

If integer equals zero, then the entire field of cnumb characters, starting with the character specified by to, will consist of fill characters.

Example: The example below converts the integer I into a 7-character string with leading zeros replaced by percent signs (%).

```
LOGICAL*1 CHAR(10)
CALL BTD(I,CHAR(1),7,ND,'%')
```

If I=-84, the 7 characters stored in CHAR(1) to CHAR(7) will be %%%%-84. ND will be set to 3.

COMC

Purpose: To determine whether one character string is less than, equal to, or greater than, another string.

Calling Sequence:

FORTRAN: CALL COMC(numb,string1,string2,differ,&err1, &err2,&err3)

Parameters:

numb is an INTEGER expression giving the number of characters in each string.

string1,string2 are the character strings to be compared for equality and may be specified either by an array variable or by a Hollerith literal. Equality is interpreted in the sense of position within the 360 collating sequence.

differ is an INTEGER variable which is set to the position of the first character in *string1* which differs from the corresponding character in *string2*. If *string1* and *string2* are identical, *differ* is set to zero.

err1 (optional) is the number of a FORTRAN statement to transfer to if *string1*<*string2*, i.e., if *string1* precedes *string2* in the collating sequence.

err2 (optional) is the number of a FORTRAN statement to transfer to if *string1*>*string2*, i.e., if *string1* follows *string2* in the collating sequence.

err3 (optional) is the number of a FORTRAN statement to transfer to if *numb*≤0.

Comments: The first character that differs dictates whether *string1* is less than or greater than *string2*. If this character in *string1* appears in the collating sequence before the corresponding character in *string2*, then *string1*<*string2*; otherwise, *string1*>*string2*. A normal RETURN is made if *string1* is identical to *string2*. If *numb*≤0, no comparison is made.

Example: The example below compares the 9 characters starting at A(15) with the character string PAR FIELD and branches to statement number 12 on inequality.

```
LOGICAL*1 A(50)
CALL COMC(9,'PAR FIELD',A(15),IDIF,&12,&12)
```

DTB

Purpose: To convert a string of numeric characters into a FORTRAN INTEGER number.

Calling Sequence:

FORTRAN: CALL DTB(from,integer,cnumb,dnumb,fill,&err)

Parameters:

- _from_ is a LOGICAL*1 array variable, or a Hollerith literal, giving the numeric characters to be converted.
- _integer_ is an INTEGER variable which will be set to the integer resulting from the conversion.
- _cnumb_ is an INTEGER variable which, on entry to DTB, should contain the maximum number of characters to be scanned in the conversion. On exit from DTB, _cnumb_ is set to the actual number of characters scanned.
- _dnumb_ is an INTEGER variable which will be set to the number of significant digits in _integer_. The sign is not included in this number.
- _fill_ is a LOGICAL*1 character variable, or a Hollerith literal, specifying a character to be ignored if it precedes the numeric digits in the string.
- _err_ (optional) is the number of a FORTRAN statement to transfer to if invalid characters or multiple signs are encountered, if the converted number is too large to hold in a FORTRAN fullword INTEGER, or if on entry, _cnumb_≤0.

Comments: A single sign (+ or -) may be imbedded in the leading fill characters and will determine the sign of _integer_. If there is no sign, '+' is assumed.

DTB can be used to reverse any action of the BTD subroutine.

If the field _from_ is all fill characters, then _integer_ and _dnumb_ are set to zero. If the field _from_ is all zeros, then _integer_ is set to zero and _dnumb_ is set to _cnumb_, the actual number of zeros in the field.

If the error return to statement _err_ is taken because of invalid characters or adjacent multiple signs, then _integer_=_dnumb_=0 and _cnumb_ is set to the number of characters scanned before the error was encountered.

There will be no error return taken once a digit is encountered. After the first digit, any nondigit (even another sign or a fill character) terminates the number.

If the error return to statement err is taken because the converted number was too large to hold in the fullword integer, then integer=0, dnumb is set to the number of digits encountered, and cnumb is set to the total number of characters in the field (fill characters plus sign character plus numeric characters).

If the error return to statement err is taken because cnumb≤0, then integer=dnumb=0 and cnumb remains unchanged.

Example: The example below converts the character string

.....-139.....

stored starting in element 30 of array NUMB, into an integer number:

```
LOGICAL*1 NUMB(75)
NC=14
CALL DTB(NUMB(30),I,NC,ND,'.',&10)
```

On exit, I=-139, NC=9, and ND=3.

EQUC

Purpose: To compare two characters for equality.

Calling Sequence:

FORTRAN:

```
LOGICAL EQUC
IF (EQUC(char1,char2)) statement
```

Parameters:

char1,char2 are LOGICAL*1 variables or array elements, or single-character Hollerith literals, to be compared for equality.

statement is a FORTRAN statement to transfer to if char1 and char2 are equal.

Comment: If char1 is identical to char2, then EQUC(char1,char2) has the value .TRUE.; otherwise, it has the value .FALSE.

Example: The example below transfers to statement number 10 if the 7th element of ARRAY is the letter G.

```
LOGICAL EQUC
LOGICAL*1 ARRAY(25)
IF (EQUC('G',ARRAY(7))) GO TO 10
```

FINDC

Purpose: To search for any one of a set of characters.

Calling Sequence:

FORTRAN:
```
CALL FINDC(array,len,char,numb,start,finish,
           cfound,&err1,&err2)
```

Parameters:

- array — is the LOGICAL*1 character array to be searched.
- len — is an INTEGER expression giving the position in array of the last character to be searched.
- char — is either an array variable indicating the characters for which to search or a Hollerith literal specifying the characters.
- numb — is an INTEGER expression giving the number of characters in char. If numb=0, then the same characters as given in a preceding call with numb>0 will be used.
- start — is an INTEGER expression indicating the position in array at which the search is to start.
- finish — is an INTEGER variable which will contain the position in array at which a character in char is found. If none of the characters is found, finish is set to zero.
- cfound — is an INTEGER variable which will be set to the position in char of the character which is found. If none of the characters is found, cfound is set to zero.
- err1 — (optional) is the number of a FORTRAN statement to transfer to if none of the characters is found in the search.
- err2 — (optional) is the number of a FORTRAN statement to transfer to if start≤0, start>len, or numb<0.

Comment: If numb=0 on the first call to FINDC, no characters will be found. Control will be transferred to the statement numbered err2.

Example: The example below searches the array LARRAY for the first occurrence of the numeric characters 0,1,2,3,...,9.

```
LOGICAL*1 LARRAY(125)
CALL FINDC(LARRAY,125,'0123456789',10,1,IF,ICF,&10)
```

If LARRAY contains the character '7' in position 39, i.e., in LARRAY(39), with no numeric characters preceding it, then, upon exit from FINDC, IF will be 39 and ICF will be 8, indicating that the 8th character in the string '0123456789' was found in LARRAY(39). If there are no numeric characters in LARRAY, then control will transfer to statement 10 with IF=ICF=0.

If, on subsequent calls to FINDC, the same characters 0,1,2,3,...,9 are to be searched for, then the fourth parameter numb should be set to zero so that initialization need not be repeated.

FINDST

Purpose: To search an array for a specified character string.

Calling Sequence:

FORTRAN:
```
CALL FINDST(array,len,string,numb,start,finish,
            &err1,&err2)
```

Parameters:

- _array_ is the LOGICAL*1 character array to be searched.
- _len_ is an INTEGER expression giving the position in _array_ of the last character in the search.
- _string_ is an array variable, or a Hollerith literal, indicating the character string for which to search.
- _numb_ is an INTEGER expression giving the number of characters in _string_.
- _start_ is an INTEGER expression indicating the position in _array_ at which the search is to start.
- _finish_ is an INTEGER variable which will be set to the position of the character in _array_ at which _string_ starts. If _string_ is not found, _finish_ is set to zero.
- _err1_ (optional) is the number of a FORTRAN statement to transfer to if _string_ is not found.
- _err2_ (optional) is the number of a FORTRAN statement to transfer to if _start_≤0, _start_>_len_, or _numb_≤0.

Comment: The complete _string_ must be within the limits _start_ and _len_ of _array_.

Example: The example below searches the array AR for the string MODE with the search starting at the 10th character and continuing to the 40th character.

```
LOGICAL*1 AR(50)
CALL FINDST(AR,40,'MODE',4,10,IFINIS,&12)
```

IGC

Purpose: To ignore all of a set of characters, i.e., to find the first character which is not one of a specified set of characters.

Calling Sequence:

FORTRAN: CALL IGC(array,len,char,numb,start,finish,
&err1,&err2)

Parameters:

- array — is the LOGICAL*1 character array to be searched.
- len — is an INTEGER expression giving the position in array of the last character in the search.
- char — is either an array variable containing, or a Hollerith literal specifying, the characters to be ignored.
- numb — is an INTEGER expression giving the number of characters in char. If numb=0, the characters given in a preceding call with numb>0 will be used in the search.
- start — is an INTEGER expression giving the position in array of the character at which the search is to start.
- finish — is an INTEGER variable which will be set to the character position in array at which the first character different from those in char is found. If all characters are ignored, finish is set to zero.
- err1 — (optional) is the number of a FORTRAN statement to transfer to if all characters are ignored.
- err2 — (optional) is the number of a FORTRAN statement to transfer to if start≤0, start>len, or numb<0.

Comment: If numb=0 on the first call to IGC, no characters are ignored; finish is set equal to start.

Example: The example below searches for the first nonblank character in the array LARRAY.

```
LOGICAL*1 LARRAY(212)
CALL IGC(LARRAY,212,' ',1,1,IF,&10)
```

If the first nonblank character is in character position 132 of the array, IF will be set to 132. If all characters are blank, then IF will be set to zero and control will transfer to statement number 10.

LCOMC

Purpose: To determine whether one character string is less than, equal to, or greater than another string.

Calling Sequence:

FORTRAN: i=LCOMC(numb,string1,string2)

Parameters:

numb is an INTEGER expression giving the number of characters in each string.

string1,string2 are the character strings to be compared for equality. They may be specified either by an array variable or by a Hollerith literal. Equality is interpreted in the sense of position within the 360 collating sequence.

Values Returned:

LCOMC is a FUNCTION subprogram and will return an integer _i_ having a value of:

+1 if _string1_>_string2_, i.e., if _string1_ follows _string2_ in the collating sequence.

0 if _string1_=_string2_, i.e., if the character strings are identical.

-1 if _string1_<_string2_, i.e., if _string1_ precedes _string2_ in the collating sequence.

Comment: If _numb_≤0, no comparison is made and _i_ is set to zero.

Example: The example below compares 2 character strings of 20 characters starting at A(1) and B(19) and branches to statement 12 on equality.

```
LOGICAL*1 A(50),B(60)
IF(LCOMC(20,A(1),B(19)).EQ.0) GO TO 12
```

MOVEC

Purpose: To move character strings from one place to another.

Calling Sequence:

FORTRAN: CALL MOVEC(numb,from,to,&err)

Parameters:

- numb — is an INTEGER expression giving the number of characters to be moved. numb must be greater than zero.
- from — is either an array variable containing the character string to be moved or a Hollerith literal specifying the string.
- to — is an array variable indicating the start of the place to which the from characters are to be moved.
- err — (optional) is the number of a FORTRAN statement to transfer to if numb≤0 or numb>32767.

Comments: The from and to array variables can indicate portions of the same array. In fact, they can be overlapping portions. However, in the latter case, the user must ensure that characters to be moved are not replaced before being moved. The characters are moved one at a time from the first to the numbth position.

If numb≤0 or numb>32767, no transfer of characters will occur.

Example: The example below moves 7 characters, starting with the 10th character of array AR1, to AR2, starting with the 80th character.

```
LOGICAL*1 AR1(100),AR2(132)
CALL MOVEC(7,AR1(10),AR2(80))
```

The example below moves the character string ERROR MESSAGES into the array MSG.

```
LOGICAL*1 MSG(80)
CALL MOVEC(14,'ERROR MESSAGES',MSG)
```

The example below moves the 4 characters DATA into a simple INTEGER variable I.

```
DATA X/'DATA'/
CALL MOVEC(4,X,I)
```

SETC

Purpose: To set adjacent characters equal to a specified character.

Calling Sequence:

FORTRAN: CALL SETC(numb,array,char,&err)

Parameters:

- numb — is an INTEGER expression giving the number of characters to be set.
- array — is an array variable giving the starting position of the characters to be set.
- char — is either a variable containing the character to which the numb characters are to be set or a Hollerith literal specifying the character.
- err — (optional) is the number of a FORTRAN statement to transfer to if numb≤0.

Comment: If numb≤0, no characters are changed.

Example: The example below sets all of the characters in the array A to blanks.

```
LOGICAL*1 A(50)
CALL SETC(50,A,' ')
```

TRNC

Purpose: To translate specified characters in an array into other characters.

Calling Sequence:

FORTRAN: CALL TRNC(numb,array,oldchar,newchar,cnumb,&err)

Parameters:

- numb — is an INTEGER expression giving the number of characters for translation.
- array — is an array variable giving the starting position of the characters for translation.
- oldchar — is either an array variable containing a list of the characters to be translated, or a Hollerith literal specifying the characters.
- newchar — is either an array variable containing a list of the characters into which oldchar is to be translated, or a Hollerith literal specifying the characters. Any occurrence of the first character in oldchar will be translated into the first character of newchar, the second character of oldchar into the second of newchar, etc.
- cnumb — is an INTEGER expression giving the number of characters in oldchar and newchar. If cnumb= 0, then oldchar and newchar as given in a preceding call with cnumb>0 will be used.
- err — (optional) is the number of a FORTRAN statement to transfer to if numb≤0 or cnumb<0.

Comments: The routine does not check for duplication of characters in oldchar. The final appearance of a duplicated character will dictate its translation.

It is the user's responsibility to ensure that there are the same number of characters in oldchar and newchar. If there are not, unpredictable translations may occur.

If numb≤0 or cnumb<0 (or ≤0 on the first call), no translation will occur. All characters not mentioned in oldchar are left alone.

Example: The example below translates all As to 1s, Bs to 2s, and Cs to 3s in the array CHAR.

```
LOGICAL*1 CHAR(65)
CALL TRNC(65,CHAR,'ABC','123',3)
```

TRNST

Purpose: To search for a given character string and translate it into another string.

Calling Sequence:

```
FORTRAN:  CALL TRNST(array,len,oldst,newst,numb,start,
                     finish,&err1,&err2)
```

Parameters:

- array — is the LOGICAL*1 character array to be searched.
- len — is an INTEGER expression giving the character position in array at which searching is to terminate.
- oldst — is either an array variable containing the character string to be translated or a Hollerith literal specifying the character string.
- newst — is either an array variable containing the new character string or a Hollerith literal specifying the string.
- numb — is an INTEGER expression giving the number of characters in the strings.
- start — is an INTEGER expression giving the position in array at which searching is to start.
- finish — is an INTEGER variable which will be set to the starting position of the translated string. finish will be set to zero if the string is not found.
- err1 — (optional) is the number of a FORTRAN statement to transfer to if oldst is not found in the search.
- err2 — (optional) is the number of a FORTRAN statement to transfer to if start≤0, start>len, or numb≤0.

Comments: oldst and newst must be the same lengths. Only the first occurrence of oldst is translated. oldst must be completely within the limits start and len of array for translation to occur.

Example: The example below translates the string RECIEVE in the array A to RECEIVE.

```
LOGICAL*1 A(200)
CALL TRNST(A,200,'RECIEVE','RECEIVE',7,1,IF,&30)
```

If the string is found starting in character 29 of A, then IF will be set to 29. If the string is not found, then IF=0 and control is transferred to statement number 30.

CHARGE

Subroutine Description

Purpose: To compute the charge for the given quantities of resources using the current rates for the signed on ID.

Location: Resident System

Calling Sequences:

```
Assembly: (a) CALL CHARGE,(cnt,quantvec,zero)
          (b) CALL CHARGE,(cnt,quant,type)

FORTRAN:  (a) amount=CHARGE(cnt,quantvec,zeroval)
          (b) amount=CHARGE(cnt,quant,type)

PL/I(F):  (a) amount=plcallt(CHARGE,f3,ADDR(cnt),
              ADDR(quantvec),ADDR(zeroval));
          (b) amount=plcallt(CHARGE,f3,ADDR(cnt),
              ADDR(quant),ADDR(type));
```

Parameters:

<u>cnt</u> is the location of the fullword (INTEGER*4, FIXED BINARY(31)) or halfword (INTEGER*2, FIXED BINARY(15)), integer number of elements (0-14) in the array "quantvec" or "quant". (If the value is zero, it must be a fullword.) This value need be only as large as the minimum number of elements necessary to pass all of the relevant quantities.

<u>quantvec</u> is the location of the first element of a fullword integer array (INTEGER*4, FIXED BINARY(31)) containing "cnt" elements which have the following data:

<u>Element</u>	<u>Data</u>
1	CPU time in milliseconds
2	CPU virtual memory integral in page-milliseconds
3	line-printer lines printed
4	line-printer pages printed
5	elapsed time in seconds
6	cards read
7	cards punched
8	disk storage in page-minutes
9	reserved; should be zero

10	magnetic-tape drive time in seconds
11	magnetic-tape mounts
12	plotter time in seconds
13	plotter paper in millimeters
14	paper tape punched in millimeters
15	wait virtual memory in page-seconds
16	reserved (should be zero)
17	paper-tape reader time in seconds
18	paper-tape mounts
19	paper-tape punch time in seconds
20	paper-tape punch mounts
21	floppy-disk drive time in seconds
22	floppy-disk mounts
23	page-printer lines printed
24	page-printer pages printed
25	page-printer images printed
26	page-printer sheets printed
27	phototypesetter units
28	phototypesetter media (cm^2)

<u>zero</u> (optional) is a fullword integer or floating-point zero or the location of a fullword zero.

<u>zeroval</u> (optional) is the location of a fullword integer or floating-point (INTEGER*4, FIXED BINARY(31)) zero.

<u>quant</u> is the location of a fullword integer array (INTEGER*4, FIXED BINARY(31)) containing the values of the quantities for which the charge is wanted.

<u>type</u> is the location of the first element of a fullword (INTEGER*4, FIXED BINARY(31)) or halfword (INTEGER*2, FIXED BINARY(15)), integer array containing indexes to identify the corresponding values in "quant". The values of these indexes are the same as the element numbers for the relevant values in "quantvec".

<u>plcallt</u> is one of the procedures PLCALLF, PLCALLE, or PLCALLD.

<u>f3</u> is the location of a FIXED BINARY(31) constant or variable having the value three.

Values Returned:

GR0 contains the charge for the specified quantities of resources computed in centicents (ten-thousandths of a dollar) using the current rates for the signed on ID.

FR0 contains the doubleword charge for the specified quantities of resources computed in dollars using the current rates for the signed on ID.

Return Codes:

0 The value has been returned as described above.
4 Invalid value for "cnt".
8 Invalid value in "type"; the value returned is the index which is in error.
12 Error, probably due to values in "quantvec" or "quant" which are too large; the value returned is the number (subscript) of the element if (a) call, or the index for the element if (b) call, being processed at the time the error occurred.
16 Error caused by either an invalid parameter list pointer or an error return from a system subroutine (the latter should not occur).

Examples: FORTRAN:

```
INTEGER VMIVEC(2)/0, 60000/
CPU=CHARGE(1, 60000, 0)
VMI=CHARGE(2, VMIVEC, 0)
FACTOR=VMI/CPU

FACTOR=CHARGE(1, 60000, 2)/CHARGE(1, 60000, 1)
```

The above two examples compute the factor by which the CPU virtual memory integral (VMI) is multiplied to produce processing time.

CHGFSZ

Subroutine Description

Purpose: To change the size or maxsize of a file either absolutely or incrementally.

Location: Resident System

Calling Sequences:

Assembly: CALL CHGFSZ,(unit,size,flag)

FORTRAN: CALL CHGFSZ(unit,size,flag,&rc4,&rc8,&rc12,
&rc16,&rc20,&rc24,&rc28,&rc32,&rc36)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

size is the location of a fullword containing the desired size or maxsize (absolute or incremental) in pages.

flag is the location of a fullword integer giving more information about the size parameter as follows:

0 - size is the desired size, absolute
1 - size is the desired change in size (positive or negative)
2 - size is the desired maxsize, absolute
3 - size is the desired change in maxsize (positive or negative)

rc4,...,rc36 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return--size or maxsize changed.
4 File does not exist.
8 Hardware error or software inconsistency.
12 Access not allowed--write-expand access required to increase size; truncate or write-expand access required to decrease size.

16 Locking the file will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of a shared file).
24 Bad parameters (i.e., bad FDUB-pointer, not a file, etc.).
28 Inconsistent <u>size</u> parameter (see Note 1 below).
32 No disk space available for expansion.
36 The space allocated to this account has been exceeded.

Notes: The resultant absolute size must be positive, greater than, or equal to the truncated size, and less than or equal to the maxsize. The maxsize must be less than or equal to 32767 pages.

A request for an absolute size of zero is defined to mean truncate the file.

A request for an absolute maxsize of zero is defined to mean set the maxsize equal to the current size.

Example: Assembly:

```
              CALL CHGFSZ,(UNIT,SIZE,FLAG)
                .
                .
        UNIT  DC   F'5'
        SIZE  DC   F'150'
        FLAG  DC   F'0'
```

The above example sets the absolute size of the file associated with logical I/O unit 5 to 150 pages.

FORTRAN:

```
              INTEGER*4 UNIT
              DATA UNIT/4/
                ...
              CALL CHGFSZ(UNIT,-10,1)
```

The above example decrements the size of the file associated with logical I/O unit 4 by 10 pages.

CHGMBC

Subroutine Description

Purpose: To change dynamically the number of page-sized buffers used by the file system to read and write a particular file.

Location: Resident System

Calling Sequences:

Assembly: CALL CHGMBC,(unit,maxbuf)

FORTRAN: CALL CHGMBC(unit,maxbuf,&rc4,&rc8,&rc12,&rc16,
&rc20,&rc24)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

maxbuf is the location of a fullword integer specifying the maximum number of buffers to use.

$1 \leq$ maxbuf ≤ 100 for sequential files
$3 \leq$ maxbuf ≤ 100 for line files

rc4,...,rc24 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Maximum number of buffers changed as specified.
4 The file does not exist.
8 Hardware error or software inconsistency.
12 Access not allowed to file.
16 Locking the file will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of a shared file).
24 Bad parameters (i.e., bad FDUB-pointer, not a file, maxbuf out of legal range).

Description: In general, the file system will dynamically allocate as many page-sized buffers for use in reading and writing a particular file as there are pages in actual use by the file (i.e., the truncated size) up to the maximum number of buffers specified. The default maximum number of buffers for both line and sequential files is 5. In simple terms, the more buffers one allows, the less physical disk I/O required, but the greater the virtual memory required.

Notes: The maximum number of buffers set by CHGMBC is <u>not</u> a static quantity saved with the file and used each time the file is accessed. The default value is always used when the file is first referenced; it may be changed dynamically by a call to CHGMBC.

In general, large line files will benefit more than sequential files from an increase in the maximum number of buffers.

Examples: Assembly:

```
          CALL CHGMBC,(UNIT,MAXBUF)
           .
           .
UNIT      DC   F'3'
MAXBUF    DC   F'10'
```

FORTRAN:

```
      INTEGER*4 UNIT, MAXBUF
      DATA UNIT/3/, MAXBUF/10/
       ...
      CALL CHGMBC(UNIT,MAXBUF)
```

The above examples dynamically assign a maximum of 10 buffers to use during I/O operations on the file associated with logical I/O unit 3.

CHGXF

Subroutine Description

Purpose: To change the expansion factor of a file.

Location: Resident System

Calling Sequences:

```
Assembly: CALL CHGXF,(unit,expfac)

FORTRAN:  CALL CHGXF(unit,expfac,&rc4,&rc8,&rc12,&rc16,
                     &rc20,&rc24)
```

Parameters:

<u>unit</u> is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

<u>expfac</u> is the location of a fullword integer (of absolute value < 32768) specifying the expansion factor to use.
<u>expfac</u> = 0 designates the default expansion factor (which is 10% of the file size).
> 0 designates an absolute number of pages by which the file may be expanded.
< 0 designates a percentage of the file size by which the file may expand, e.g., -50 means 50%.

<u>rc4,...,rc24</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 The expansion factor was changed as specified.
4 The file does not exist.
8 Hardware error or software inconsistency.
12 Access not allowed to file.
16 Locking the file will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of a shared file).
24 Invalid call (i.e., bad FDUB-pointer, not a file, <u>expfac</u> out of legal range).

Description: The expansion factor of a file determines the amount by which the file may expand when it exceeds the size of its current disk allocation. This amount is added to the current allocation and the corresponding disk space is used to contain the new data that was being written into the file when the expansion occurred.

There is a certain amount of system overhead necessary each time a file is expanded which adds to the user's cost in writing to the file. By reducing the number of times a file must be expanded, this cost may be lowered. One method of reducing this is to increase the amount by which a file is expanded each time, i.e., increase the expansion factor.

The CHGXF subroutine may be used to increase the expansion factor. By setting the <u>expfac</u> parameter, the user may specify either an absolute number of pages or a percentage of the current (at the time of expansion) size to be used as the expansion amount when an expansion occurs. The default expansion factor is 10%.

For example, if the user has a file with a current size of 100 pages and wishes to write 150 pages of data into it, the file will have to be expanded 5 times in order to accommodate the data using the default expansion factor of 10% (the file is expanded to the sizes 110, 121, 133, 146, and 161 pages, respectively). If this expansion factor is changed to 50%, the file will be expanded only once to a size of 150 pages. If an expansion factor of 100% were used, the file would be expanded to 200 pages leaving 50 pages unused.

The expansion amount calculated using the expansion factor will not (except as noted below) result in an expansion of insufficient size to contain the new data, as adequate space is always acquired to ensure that the new data may be written into the file. However, an improper expansion factor may cause file space to be wasted as illustrated in the example above.

If an extensive allocation is requested which would cause the user's disk space allocation or the remaining free space on the disk volume to be exceeded, the expansion amount is decreased accordingly. This prevents an expansion factor from inhibiting an otherwise legitimate extension of a file.

Examples: Assembly:

```
                 CALL CHFXF,(UNIT,EXPFAC)
                   .
                   .
          UNIT   DC   F'3'
          EXPFAC DC   F'-20'
```

FORTRAN:

```
      INTEGER*4 UNIT, EXPFAC
      DATA UNIT/3/, EXPFAC/-20/
        ...
      CALL CHGXF(UNIT,EXPFAC)
```

The above examples set the expansion factor to 20% for the file assigned to logical I/O unit 3.

CHKACC

Subroutine Description

Purpose: To determine the access that a signon ID, project number, and program key "triple" has to a particular file.

Location: Resident System

Calling Sequences:

Assembly: CALL CHKACC,(name,triple)

FORTRAN: CALL CHKACC(name,triple,&rc4,&rc8,&rc12)

```
INTEGER*4 CHKACC,x
x=CHKACC(name,triple)
```

Parameters:

name is the location of the name (with trailing blank) of the file.

triple is the location of a 4-character signon ID, followed by a 4-character project number, followed by an external program key (with trailing blank), such as returned by GUINFO or GFINFO.

x is the fullword-integer value returned (i.e., the access) if the file exists (see values returned below).

rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return code occurs.

Values Returned:

If the return code from CHKACC is zero (or twelve), then GR0 contains the access that the "triple" has to the file as follows:

1 Read access allowed.
2 Write-expand access allowed.
4 Write-change/empty access allowed.
8 Truncate/renumber access allowed.
16 Destroy/rename access allowed.
32 Permit access allowed.

If more than one type of access is allowed, the value returned in GR0 is the sum of the different types of access, e.g., GR0=63 implies unlimited access.

Return Codes:

0 The file exists, access returned in GR0.
4 The file does not exist.
8 Hardware error or software inconsistency encountered.
12 Access not allowed, zero returned in GR0.

Note: FORTRAN users wishing to obtain both the return codes and the access types may use the RCALL subroutine to call CHKACC.

Examples: Assembly:

```
       CALL CHKACC,(FNAME,TRIPLE)
       LTR  15,15
       BNZ  NOREAD
       N    GR0,=F'1'
       C    GR0,=F'1'
       BE   READ
        .
        .
FNAME  DC   C'6AGA:DATAFILE '
TRIPLE DC   C'1KYZ'        Signon ID
       DC   C'W000'        Project number
       DC   C'*EXEC '      Program key
```

FORTRAN:

```
INTEGER*4 CHKACC,X
DATA MASK/Z00000001/
X=CHKACC('6AGA:DATAFILE ','1KYZW000*EXEC ')
X=LAND(X,MASK)
IF (X.EQ.1) GO TO 10
```

These examples call CHKACC to determine whether signon ID 1KYZ under project number W000 running a program with a program key of *EXEC (the default) has read access to file 6AGA:DATAFILE.

CHKFDUB

Subroutine Description

Purpose: To obtain a FDUB-pointer for a specified logical I/O unit; to verify that a given FDUB-pointer is legal.

Location: Resident System

Alt. Entry: CHKFDB

Calling Sequences:

Assembly: CALL CHKFDUB,(unit)

FORTRAN: INTEGER*4 CHKFDB,x
x = CHKFDB(unit)

Parameters:

unit is the location of either
(a) a FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

x is the fullword-integer FDUB-pointer obtained (see "Value Returned" below).

Value Returned:

GR0 contains the FDUB-pointer obtained for the specified logical I/O unit if a successful return is made.

Return Codes:

0 Successful return.
4 Illegal unit parameter specified, supplied pointer is not pointing to a FDUB, or logical I/O unit unassigned.

Description: If the unit parameter is the location of a FDUB-pointer, the subroutine will check the legality of the FDUB-pointer.

If the unit parameter is the location of a logical I/O unit name or number, the subroutine will obtain a FDUB-pointer for the file or device attached to that logical I/O unit. This is one way to obtain a FDUB-pointer for a file or device attached to a specific logical I/O unit, but in general it is better to use the logical I/O unit name or number rather than the FDUB-pointer. If the logical I/O unit is unassigned, no FDUB-pointer will be returned.

This subroutine does not check the legality of the file or device name attached to the logical I/O unit specified.

Examples: Assembly:

```
              CALL  CHKFDUB,(UNIT)
              LTR   15,15
              BNZ   ERROR
               .
               .
        UNIT  DC    F'6'
```

FORTRAN:

```
              INTEGER*4 UNIT
              DATA UNIT/6/
                ...
              CALL CHKFDB(UNIT,&99)
```

The above examples call CHKFDUB to get a FDUB-pointer for the file or device attached to logical I/O unit 6.

CHKFILE

Subroutine Description

Purpose: To determine whether a file exists, as well as what access the calling program has to the file. This is the easiest way to determine whether a scratch file exists without creating it.

Location: Resident System

Alt. Entry: CHKFIL

Calling Sequence:

Assembly: CALL CHKFILE,(name)

FORTRAN: CALL CHKFIL(name,&rc4,&rc8,&rc12)

or

```
INTEGER*4 CHKFIL,x
x = CHKFIL(name)
```

Parameters:

name is the location of the name of the file (with a trailing blank).
x is the fullword-integer value returned if the file exists (see "Values Returned" below).
rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return code occurs.

Values Returned:

If the return code from CHKFILE is zero (or twelve), then GR0 contains the access that the calling user has to the file as follows:

1 Read access allowed.
2 Write-expand access allowed.
4 Write-change/empty access allowed.
8 Truncate/renumber access allowed.
16 Destroy/rename access allowed.
32 Permit access allowed.

If more than one type of access is allowed, the value returned in GR0 is the sum of the different types of access, e.g., GR0=63 implies unlimited access.

Return Codes:

0 The file exists.
4 The file does not exist.
8 Unaddressable parameter or hardware/software inconsistency.
12 Access not allowed.

Note: FORTRAN users wishing to obtain both the return codes and access types may use the RCALL subroutine to call CHKFILE.

Examples: Assembly:

```
      CALL CHKFILE,(FNAME)
      LTR  15,15
      BNE  NOREAD
      SLL  0,31
      SRL  0,31
      C    GR0,=F'1'
      BE   READ
        .
        .
FNAME DC   C'2AGA:DATAFILE '
```

FORTRAN:

```
INTEGER*4 CHKFIL,X
DATA MASK/Z00000001/
X = CHKFIL('2AGA:DATAFILE ')
X = LAND(X,MASK)
IF(X.EQ.1) GO TO 10

EXTERNAL CHKFIL
INTEGER*4 ADROF,X
DATA MASK/Z00000001/
PAR = ADROF('2AGA:DATAFILE ')
CALL RCALL(CHKFIL,2,0,ADROF(PAR),1,X,&100)
X = LAND(X,MASK)
IF(X.EQ.1) GO TO 10
```

These examples call CHKFILE to determine whether the calling program has read access to the file 2AGA:DATAFILE. The second FORTRAN example uses the RCALL subroutine to obtain both the return code and the return value.

CHKPAR

Subroutine Description

Purpose: To check the number and data types of parameters passed to a subroutine.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL CHKPAR(icode,'string ',&rc4)

Parameters:

icode is a switch indicating the action to be taken if an error is found by CHKPAR. The legal switch values are:

0 A traceback of the subroutine calls is produced and then execution is suspended. Execution may be resumed by the $RESTART command.
1 A traceback of the subroutine calls is produced and then execution is resumed.
2 Execution is continued with an error message but without a traceback.
3 Execution is continued without an error message or a traceback.

In all cases, a return code 4 (RETURN 1) is produced if an error is detected.

string is a string of characters of the form I (integer), R (real), and X (other) which corresponds in data type to the dummy variables in the calling sequence of the subroutine being checked. CHKPAR checks only REAL*4 and REAL*8 variables, and INTEGER*4 variables of magnitude less than 1048575. All other variables must be indicated by an X and are ignored. The string must be enclosed in primes and terminated by a blank.

The letter O may be included in the string to indicate that the remaining parameters are optional. The letter S may be included to stop the checking of parameters before the end of the parameter list is encountered. The S option is useful if the caller is not required to set the variable length bit (the high-order bit in the last parameter address).

CHKPAR will not differentiate between REAL*4 and REAL*8 variables.

rc4 (optional) is the number of a FORTRAN statement to transfer to if the number of parameters or their data types are not correct. If omitted, control will return to the statement following the call to CHKPAR.

Note: Standard OS Type-I(S) calling conventions must be used in all subroutine calls. See the section "Calling Conventions" in this volume.

Description: CHKPAR tests the data types of the arguments in the subroutine from which CHKPAR was called against the data types specified in the string parameter. A value of zero is legal regardless of data type. If the value is nonzero, the absolute value of the variable is taken and the high-order byte is tested for zero. If this byte is nonzero, the corresponding data type must be R. If this byte is zero, the next 4 bits (20-23) must be zero for integer variables and nonzero for real variables.

CHKPAR must be called from the subroutine whose parameter list is being checked.

Examples: FORTRAN:

```
          X=10.
          Y=20.
          CALL SUBR(X,Y,Z)
          STOP
          END

          SUBROUTINE SUBR(I,Y,Z)
          CALL CHKPAR(1,'IRX ',&10)
          Z=FLOAT(I)+Y
          RETURN
    10    WRITE(6,100)
   100    FORMAT('0ERROR IN CALL TO SUBR')
          STOP
          END
```

In the above example, X is incorrect in the call to SUBR. The following type of message is subsequently printed:

```
Error in argument number n in call to subroutine SUBR.
Type should be (integer/real) is (real/integer).
Integer value is "xxxx", real "xxxx", hex "xxxx",
character "xxxx".
```

CHKPAR then produces a traceback and transfers control to statement number 10. The third parameter Z in the above example is not checked by CHKPAR because it is returned by the subroutine SUBR and therefore is not initialized when CHKPAR is called.

FORTRAN:

```
      I=10.
      Y=20.
      CALL SUBR(I,Y)
      STOP
      END

      SUBROUTINE SUBR(I,Y,Z)
      CALL CHKPAR(0,'IRX ',&10)
      Z=FLOAT(I)+Y
      RETURN
 10   WRITE(6,100)
100   FORMAT('0ERROR IN CALL TO SUBR')
      STOP
      END
```

In the above example, the following message is printed:

    Number of arguments wrong in call to SUBR.

CHKPAR then produces a traceback and suspends execution. The user may resume execution via the $RESTART command.

CLOSEFIL

Subroutine Description

Purpose: To close a file and release its file buffers.

Location: Resident System

Alt. Entry: CLOSFL

Calling Sequences:

Assembly: CALL CLOSEFIL,(unit)

FORTRAN: CALL CLOSFL(unit,&rc4)

Parameter:

unit is the location of either
(a) a FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

rc4 (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

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

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

The subroutine WRITEBUF may be called to write the changed lines without closing the file and releasing the buffers. WRITEBUF is more efficient and therefore is generally preferred. See the description of WRITEBUF in this volume.

Examples: Assembly:

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

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

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

CMD

Subroutine Description

Purpose: To execute an MTS command from a program and return to the program after the command has been executed.

Location: Resident System

Calling Sequences:

Assembly: CALL CMD,(char,len)

or

CMD char[,len]

FORTRAN: CALL CMD(char,len)

Parameters:

- char is the location of a character string containing an MTS command.
- len is the location of the length of the character string expressed as either a fullword (INTEGER*4) or a halfword (INTEGER*2). If the first two bytes of len are zero, it is assumed len specifies a fullword integer. Otherwise, len is assumed to be a halfword.

Note: The complete description for using the CMD macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

Description: This subroutine returns to MTS specifying a character string to be interpreted as an MTS command. After the command has been executed, a return is made to the program.

The command is echoed on *SINK* and/or *MSINK* if the $SET ECHO option is ON.

This subroutine cannot be used properly with character strings that specify the following commands:

DEBUG	LOAD
RUN	UNLOAD
START AT location	SIGNON
RESTART AT location	SIGNOFF
RERUN	

If any of these commands are used with CMD, the subroutine will not return to the calling program. This would be the same as if the MTSCMD subroutine were used instead.

The START and RESTART commands will work properly unless an explicit restart address is given.

See also the description of the COMMAND subroutine in this volume.

Examples: FORTRAN:

```
        CALL CMD('$SINK FYLEB ',12)
```

The above example calls CMD to reassign *SINK* to the file FYLEB.

Assembly:

```
              CALL CMD,(CHAR,LEN)
                .
                .
        CHAR  DC   C'$CREATE ALPHA '
        LEN   DC   F'14'

              CMD  '$CREATE ALPHA '
```

The above two examples call CMD to create the file ALPHA. The first uses the CALL macro and the second uses the CMD macro.

CMDNOE

Subroutine Description

Purpose: To execute an MTS command from a program and return to the program after the command has been executed.

Location: Resident System

Calling Sequences:

Assembly: CALL CMDNOE,(char,len)

FORTRAN: CALL CMDNOE(char,len)

Parameters:

char is the location of a character string containing an MTS command.

len is the location of the length of the character string expressed as either a fullword (INTEGER*4) or a halfword (INTEGER*2). If the first two bytes of *len* are zero, it is assumed *len* specifies a fullword integer. Otherwise, *len* is assumed to be a halfword.

Description: This subroutine returns to MTS specifying a character string to be interpreted as an MTS command. After the command has been executed, a return is made to the program.

The command is never echoed on *SINK* and/or *MSINK*, regardless of the setting of the $SET ECHO option.

This subroutine cannot be used properly with character strings that specify the following commands:

DEBUG	LOAD
RUN	UNLOAD
START AT location	SIGNON
RESTART AT location	SIGNOFF
RERUN	

If any of these commands are used with CMDNOE, the subroutine will not return to the calling program. This would be the same as if the MTSCMD subroutine were used instead.

The START and RESTART commands will work properly unless an explicit restart address is given.

See also the description of the COMMAND subroutine in this volume.

Examples: FORTRAN:

```
CALL CMDNOE('$SINK FYLEB ',12)
```

The above example calls CMDNOE to reassign *SINK* to the file FYLEB.

Assembly:

```
                CALL CMDNOE,(CHAR,LEN)
                  .
                  .
          CHAR  DC   C'$CREATE ALPHA '
          LEN   DC   F'14'
```

The above example calls CMDNOE to create the file ALPHA.

CNFGINFO

Subroutine Description

Purpose: To obtain information about the type of system on which the program is running.

Location: Resident System

Alt. Entry: CFGINF

Calling Sequences:

```
Assembly: L     r,=V(CNFGINFO)
          USING CNFGINFD,r
```

Parameters:

_r_ is a general register containing the address of the CNFGINFO table.

Description: The information available in the table is described by the dsect given on the following pages (from the file *CNFGINFODSECT).

Example: Assembly:

```
L     3,=V(CNFGINFO)
USING CNFGINFD,3
TM    CIFEATUR,CI370      System 370?
BZ    SYS360
 .
 .
COPY  *CNFGINFODSECT
```

The above example illustrates how a program may determine whether it is running on a System/370- or System/360-compatible machine.

FORTRAN programs can obtain the system information by creating a common section describing the dsect. A RIP loader record (RIP CFGINF) must be inserted into the FORTRAN object file to force the loader to resolve the symbol CFGINF from the low-core symbol table.

```
*****************************************************************
*
*        Dsect of information concerning configuration of machine
*
*              (Last revised on January 12, 1984)
*
*****************************************************************
CNFGINFD DSECT
CISYSTEM DC    X'0370'              Type of system (360/370)
CICPUID  DS    0XL8                 Result of store CPU ID on lowest
*                                   address CPU in the system
CIVERSCD DC    X'02'                Version code
CIID#    DC    X'000001'            Serial number of CPU
CIMODEL  DC    X'0580'              Model number of system
CIMCEL   DC    H'0'                 Max length of MCEL
*
*        The following two fields will be zero unless the version
*        above is X'FF' indicating that we are running under
*        a hypervisor (aka virtual machine).  When the version
*        code is X'FF' the serial number and model number
*        stored in CIID# and CIMODEL are those for the real
*        machine on which the hypervisor is running and
*        additional information about the hypervisor
*        is stored as an extended CPU ID, the length and
*        location of which are given by CIEXTIDL and CIEXTID.
*        CIMCEL gives the max. MCEL length stored by the
*        hypervisor.
*
CIEXTIDL DC    H'0'                 Length of extended CPU ID
CIEXTID  DC    A(0)                 Location of extended CPU ID
*
*        An extended CPU ID is 16 bytes & has the following format:
*
*        DS    CL8                  Hypervisor name (EBCDIC)
*        DS    XL3                  Hypervisor version
*        DS    X                    Version code
*        DS    H                    Max. MCEL
*        DS    H                    CPU address
*
*        These 16 bytes will be repeated once for each
*        hypervisor that is in use. The version code, Max. MCEL
*        length, and CPU address are those of the machine (real
*        or virtual) on which the hypervisor is running.
*
*
*        The following 64 bits are each associated with a particular
*        feature or RPQ as indicated.  See Appendix D, Facilities,
*        in "IBM System/370 Principles of Operation" (GA22-7000-8)
*        for additional information.
*
CIFEATUR DC    X'F7806A1C00000000'
*
*        First byte
```

```
*
CIDEC    EQU   X'80'                 Decimal instructions - AP,CP,DP,ED,
*                                    EDMK,MP,SP,SRP,ZAP
CIFLPT   EQU   X'40'                 Floating point - ADR,AD,AER,AE,AWR,
*                                    AW,AUR,AU,CDR,CD,CER,CE,DDR,DD,DER,
*                                    DE,HDR,HER,LDR,LD,LER,LE,LTDR,LTER,
*                                    LCDR,LCER,LNDR,LNER,LPDR,LPER,MDR,MD,
*                                    MER,ME,STD,STE,SDR,SD,SER,SE,
*                                    SWR,SW,SUR,SU
CI370    EQU   X'20'                 Standard 370 features -
*                                    MVCL,CLCL,MC,STCTL,LCTL,CLM,STCM,ICM,
*                                    STIDP,STIDC,SCK,STCK,SIOF,CLRIO,
*                                    HDV,Fetch protect,
*                                    and SRP if CTDEC also on
CI370TRN EQU   X'10'                 370 translation feature -
*                                    LRA,PTLB,RRB,STNSM,STOSM
CI370MP  EQU   X'08'                 370 multiprocessor feature - SIGP,SPX
*                                    STAP,STPX
CICNDSWP EQU   X'04'                 370 conditional swapping feature -
*                                    CS and CDS
CIPSWKEY EQU   X'02'                 PSW-key handling feature  - IPK,SPKA
CICPUTIM EQU   X'01'                 CPU timer and clock comparator -
*                                    SCKC,SPT,STCKC,STPT
*
*        Second byte
*
CIEXTFLP EQU   X'80'                 Extended-precision floating point -
*                                    AXR,LRDR,LRER,MXR,MXDR,MXD,SXR
CIMOD67  EQU   X'40'                 360/67 standard features - BAS,BASR,
*                                    STMC,LRA,LMC, Fetch protect
CI32BT67 EQU   X'20'                 360/67 with 32-bit addressing
CI67DCTL EQU   X'10'                 360/67 extended direct control - WRD
CI67EXFP EQU   X'08'                 360/67 extended-precision floating
*                                    point - MDDR,ADDR,SDDR,MDD,ADD,SDD
CI67MXFP EQU   X'04'                 360/67 mixed-precision floating
*                                    point - LX,AX,SX,MX,DX
CISWPR   EQU   X'02'                 360/67 RPQ swap register instruction
*                                    SWPR
CISLT    EQU   X'01'                 360/67 RPQ search list instruction
*                                    SLT. The SLT instruction is simulated
*                                    in software by the supervisor when
*                                    SLT isn't available in the hardware.
*
*        Third byte
*
CIMXRDD  EQU   X'80'                 360/67 mixed-precision floating
*                                    point with store rounded - LX,AX,
*                                    SX,STRE,STRD
CIDIRCTL EQU   X'40'                 370 direct control facility -
*                                    RDD, WRD
CIBAS    EQU   X'20'                 370 branch and save facility -
*                                    BAS and BASR
CIEXTADR EQU   X'10'                 31-bit (extended) addressing facility
```

```
CICIDA   EQU   X'08'              Channel indirect data addressing
*                                 (CIDA) facility
CICSSW   EQU   X'04'              Channel-set switching facility -
*                                 CONCS, DISCS
CICLRIO  EQU   X'02'              Clear I/O feature
CIDAS    EQU   X'01'              Dual address space (DAS) facility -
*                                 EPAR, ESAR, IAC, IVSK, LASP, MVCP,
*                                 MVCS, MVCK, PC, PT, SAC, SSAR
*        Fourth byte
*
CIEXT    EQU   X'80'              Extended facility (Talk about
*                                 names with little information
*                                 content!) - IPTE, TPROT, Common
*                                 segment facility, Low-address
*                                 protection (does not include
*                                 the MVS dependent instructions).
CIEXTRA  EQU   X'40'              Extended real addressing facility -
*                                 26-bit page-frame real addresses
*                                 in the page-table entry for 4K-byte
*                                 pages.
CIEXTSIG EQU   X'20'              External signal facility
CIFREL   EQU   X'10'              Fast release facility
CIHDV    EQU   X'08'              Halt device facility
CIIOELOG EQU   X'04'              I/O extended logout facility
CILCLOG  EQU   X'02'              Limited channel logout facility
CIMVCIN  EQU   X'01'              Move inverse - MVCIN
*
*        FIFTH BYTE
*
CICLRCH  EQU   X'80'              Recovery extensions - CLRCH
CISEGPRT EQU   X'40'              Segment protection facility
CISERSIG EQU   X'20'              Service signal facility
CISIOFQ  EQU   X'10'              Start-I/O-fast queuing
CISKIEXT EQU   X'08'              Storage-key-instructions extensions -
*                                 ISKE, RRBE, SSKE
CISK4KB  EQU   X'04'              Storage-key 4K-byte block
CIRIO    EQU   X'02'              Suspend and resume - RIO
CITB     EQU   X'01'              Test block - TB
*
*        Sixth byte
*
CIBIDAWS EQU   X'80'              31-big (BIG) IDAWS
CIMVSEXT EQU   X'40'              MVS dependent instructions that
*                                 are part of the extended facility.
*
*        Seventh byte             Unused for now
*        Eighth byte              Unused for now
*
*
*        The following field contains the address (from a STAP
*        instruction) of the processor the system is running on.
*        If there is more than one processor in the configuration,
*        the lowest address of any online processor is used.
```

```
*
CICPUAD  DS    H                   Address of the CPU running on
*
*        The following field contains a machine hardware level
*        number or other similar identification needed by the
*        model-dependent machine-check handler to determine
*        which of several recovery actions to take for machine
*        checks.
*
CIMCHLVL DC    H'0'
*
CIXTRA   DS    XL12                Unused
*
*        System software version numbers
*        One number for the minimum version for the entire system,
*        one for the supervisor, one for the MTS command language/
*        file system, one for the spooling system, and one spare.
*        The format of each version number is the distribution
*        number times 1000.
*
CIVGM    DC    FE3'5.1'            Guaranteed minimum version
CIVUMMPS DC    FE3'5.1'            Supervisor version
CIVMTS   DC    FE3'5.1'            MTS cmnd lang/file system version
CIVSPOOL DC    FE3'5.1'            Spooling system version
CIVXTRA  DC    3FE3'0'             Spare
*
*        The following pairs of words give the assignment of virtual
*        memory used by the supervisor and MTS. Each entry consists
*        of two words giving the first and last location in a
*        particular type of VM.  The various types can be assumed to
*        be contiguous, non-overlapping areas, but not necessarily
*        contiguous with one another.
*
CIVMABS  DC    A(0,X'FFFFF')           Unpaged shared memory
CIVMSH   DC    A(X'100000',X'5FFFFF') Paged shared memory
CIVMSYS  DC    A(X'600000',X'7FFFFF') Private system storage
CIVMUSER DC    A(X'800000',X'EFFFFF') Private user storage
*                                      Segments 6 15 (F) is currently
*                                      unused at UM.
*
*        The following word gives the first address in the segment
*        used by the virtual machine support in the supervisor.
*
CIVMSEG  DC    A(X'A00000')
*
*        The following halfword contains a code indicating the
*        installation where we are running followed by the
*        character name of the installation.
*
CIICODE  DC    Y(CIIUM)  Numeric installation code
CIIOTHER EQU   0           Unknown/other
CIIUM    EQU   1           University of Michigan
CIIUBC   EQU   2           University of British Columbia
```

```
CIIUNE   EQU   3           University of Newcastle upon Tyne
CIIUQV   EQU   4           University of Alberta
CIIWSU   EQU   5           Wayne State University
CIIRPI   EQU   6           Rensselaer Polytechnic Institute
CIISFU   EQU   7           Simon Fraser University
* CIIEMB EQU   8           Unused (was EMBRAPA - Brasil)
CIIRIO   EQU   9           CNPQ/LCC - Brasil
CIIUD    EQU   10          University of Durham
CIIAMD   EQU   11          Amdahl
CIIUZ    EQU   12          University of Zagreb, Yugoslavia
*
CIINAME  DC    CL24'MTS Ann Arbor           ' Installation name
*
*        The following region contains the Ramrod system name
*        for the currently loaded resident system, followed
*        by the time and date when the currently loaded resident
*        system was written.
*
CIRSNAME DC    CL40' '     Resident system name
CIRSTIME DC    CL8' '      Resident system time (hh:mm:ss)
CIRSDATE DC    CL13' '     Resident system date (www mmm dd/yy)
*                            where 'www' is the day of the week,
*                                  'mmm' is the month, and
*                                  'dd/yy' is the date and year.
*
*        The following word contains the "SHARE" code of the
*        installation where this system is installed.  If the
*        installation doesn't belong to SHARE and thus doesn't
*        have a SHARE code, one is made up anyway.
*
         DS    0F
CISHARE  DC    CL3'UM '   Local installation's "SHARE" code
         DC    CL1' '     Unused, will be blank
*                'UM ' - University of Michigan
*                'UBC' - University of British Columbia
*                'NCL' - University of Newcastle upon Tyne
*                'UQV' - University of Alberta
*                'WSU' - Wayne State University
*                'RPI' - Rensselaer Polytechnic Institute
*                'SFU' - Simon Fraser University
*                'EMB' - EMBRAPA - Brasil (inactive)
*                'RIO' - CNPQ/LCC - Brasil
*                'DUR' - University of Durham
*                'AMD' - Amdahl
*                'UZ'  - University of Zagreb, Yugoslavia
*
CIHNAME  DC    CL8'UM '   Host name for those installations
*                         that run more than one production
*                         MTS system.
CIREALM  DC    A(X'01FE000')  Real memory size of machine.  This
*                         value can also be thought of as the first
*                         invalid real memory address.
*
```

```
CIPRIVAT DC    A(X'600000',X'FFFFFF')     Address range of storage
*                                         private to each task.
```

CNTLNR

Subroutine Description

Purpose: To count all or a subset of the lines in a line file.

Location: Resident System

Calling Sequences:

```
Assembly: CALL CNTLNR,(unit,first,last,cnt)

FORTRAN:  CALL CNTLNR(unit,first,last,cnt,&rc4,&rc8,
                      &rc12,&rc16,&rc20,&rc24,&rc28)
```

Parameters:

- unit — is the location of either
  - (a) a fullword-integer FDUB-pointer (such as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99), or
  - (c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).
- first — is the location of a fullword containing the internal line number of the first line to be counted.
- last — is the location of a fullword containing the internal line number of the last line to be counted.
- cnt — is the location of a fullword in which the count of the number of lines in the specified range will be returned.
- rc4,...,rc28 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

- 0 The file was counted successfully.
- 4 The file does not exist or unit is invalid.
- 8 Hardware error or software inconsistency encountered.
- 12 Read access not allowed.
- 16 Locking the file for read will result in a deadlock.
- 20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of a shared file).
- 24 Parameters not addressable or inconsistent parameters specified (first greater than last, etc.).
- 28 The file is not a line file.

Notes: If _first_ and _last_ do not correspond to actual line numbers in the file, the next and previous line numbers, respectively, will be used.

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

Examples: Assembly:

```
          CALL GETFST,(UNIT,FSTLNR)
          CALL GETLST,(UNIT,LSTLNR)
          CALL CNTLNR,(UNIT,FSTLNR,LSTLNR,CNT)
            .
            .
   UNIT   DC   F'4'
   FSTLNR DS   F     First line number
   LSTLNR DS   F     Last line number
   CNT    DS   F     Count
```

FORTRAN:

```
      INTEGER*4 UNIT,CNT
      DATA UNIT/4/
         ...
      CALL CNTLNR(UNIT,-2147483648,2147483647,CNT)
```

The above examples illustrate two ways to count all of the lines of the line file attached to logical I/O unit 4.

COMMAND

Subroutine Description

Purpose: To execute an MTS command from a program and return to the program after the command has been executed.

Location: Resident System

Alt. Entry: COMMND

Calling Sequences:

```
Assembly: CALL COMMAND,(char,length,sws,sumry,code,
                        origin),VL

FORTRAN:  CALL COMMND(char,length,sws,sumry,code,origin,
                      &rc4,&rc8,&rc12)
```

Parameters:

<u>char</u> is the location of a character string containing an MTS command.

<u>length</u> is the location of the length of the character string expressed as either a fullword (INTEGER*4) or a halfword (INTEGER*2). If the first two bytes of <u>length</u> are zero, it is assumed <u>length</u> specifies a fullword integer; otherwise, <u>length</u> is assumed to be halfword.

<u>sws</u> is the location of a fullword of switches defined as follows:

```
bits 30-31: command echo control.
            00  echo command if $SET ECHO=ON
            01  always echo the command
            10  do not echo the command
bits 28-29: command commentary control.
            00  print commentary if command
                was echoed
            01  always print commentary
            10  do not print commentary
bits  0-27: unused (must be zero).
```

<u>sumry</u> (optional) is the location of a fullword integer giving the error/status summary.

<u>code</u> (optional) is the location of a fullword integer giving more detailed information about the error/status summary.

<u>origin</u> (optional) is the location of a fullword integer giving the originator of the error/

status information.
rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return codes occur.

Return Codes:

0 Command successfully executed.
4 Command not successfully executed (sumry ≥ 2).
8 Reserved for future use.
12 Invalid parameters to COMMAND subroutine.

Description: This subroutine returns to MTS specifying a character string to be interpreted as an MTS command. After the command has been executed, a return is made to the program.

In addition, the COMMAND subroutine controls the echoing of the command text and the printing of any command commentary generated by the execution of the command, such as confirmation messages. This allows a program to emulate the command processing of MTS.

Normally, MTS commands are echoed when $SET ECHO=ON (the default) and the command line was not read from the user's terminal. The COMMAND subroutine will emulate this case when bits 30-31 of sws are zero; the other settings allow the program to have explicit control of echoing.

Normally, command commentary is printed if the command was read from the user's terminal or if the command was echoed. The COMMAND subroutine will emulate this case when bits 28-29 of sws are zero; the other settings allow the program to have explicit control of command commentary printing. The printing of command commentary is independent of the $SET TERSE option. When TERSE=ON, the commentary may be abbreviated (or suppressed in some cases).

A common use of this subroutine is the case in which the command line was read from the user's terminal or the command was already echoed by the program. In this case, the command commentary should be printed but the command not echoed; bits 28-31 of sws should be set to 0110. In the case in which the command was read from a file and has not been echoed by the user's program, bits 28-31 should be set to zero.

When sws is zero, the COMMAND subroutine will behave exactly as the CMD subroutine. When bits 28-31 of sws are 1010, the COMMAND subroutine will behave exactly the same as the CMDNOE subroutine.

The sumry, code, and origin parameters may be given to obtain error/status information from the system (see the description of the CSGET, CSSET subroutine for further details).

This subroutine cannot be used properly with character strings that specify the following commands:

DEBUG	LOAD
RUN	UNLOAD
START AT location	SIGNON
RESTART AT location	SIGNOFF
RERUN	

If any of the above commands are used with COMMAND, the subroutine will not return to the calling program. This would be the same as if the MTSCMD subroutine were used instead. The START and RESTART commands will work properly unless an explicit restart address is given.

Examples: Assembly:

```
               CALL COMMAND,(CHAR,LEN,SWS),VL
                .
                .
     CHAR  DC   C'$CREATE ALPHA '
     LEN   DC   A(L'CHAR)
     SWS   DC   X'00000006'
```

FORTRAN: `CALL COMMND('$CREATE ALPHA ',14,6)`

The above two examples call COMMAND to create the file ALPHA. The command commentary is printed but the command is not echoed, thus making it appear as if MTS had read the command instead of the program.

CONTROL

Subroutine Description

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

Location: Resident System

Alt. Entry: CNTRL

Calling Sequences:

Assembly: CALL CONTROL,(info,len,unit,ret)

FORTRAN: CALL CNTRL(info,len,unit,ret,&rc4,&rc8,&rc12)

Parameters:

info is the location of the device control information to be passed to the device support routines.

len is the location of the halfword (INTEGER*2) length of the control information.

unit is the location of either
- (a) a fullword integer FDUB-pointer (as returned by GETFD),
- (b) a fullword-integer logical I/O unit number (0 through 99), or
- (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

ret is the location of an area of 27 fullwords (108 bytes) to receive the return information from the device support routines. This area will contain:

Word 1: return code from the DSR
2: length of the DSR message, or zero
3-27: DSR error message (if given)

This parameter is optional and can be omitted (if called from FORTRAN) or zero (if called from assembly language).

_rc4,...,rc12_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return from DSR.
4 Illegal parameter specification.

8 Nonzero return code from DSR. This return code is given in ret(1).
12 DSR error. The DSR return code is in ret(1), the DSR message length is in ret(2), and the message is in ret(3)-ret(27).

Note: The return code given by the CONTROL subroutine is not the return code given by the DSR. The return code from the subroutine is given in GR15 and used to indicate the existence of a DSR return code which is given in ret.

Description: Only certain file and device types currently allow control operations. These are:

Type	Control Commands
MNET	- Any of the Merit/UMnet Computer Network device commands as normally entered after a percent sign "%". The percent sign should not be given as part of the control information.
MRXA TTY	- Any of the Memorex device commands as normally entered after a percent sign"%". The percent sign should not be given as part of the control information.
3270	- Any of the IBM 3278/Lee Data Terminal device commands as normally entered after a percent sign "%". The percent sign should not be given as part of the control information.
3036	- Any IBM 3278 device command.
3066	- Any IBM 3278 device command.
FDSK	- Any control command (floppy disk).
9TP	- Any control command (9-track magnetic tape).
HPTR HPCH HBAT	- Any control command legal for *PRINT*, PUNCH*, or *BATCH*, respectively.
FILE	- See MTS Volume 1, The Michigan Terminal System.
SEQF	- See MTS Volume 1, The Michigan Terminal System.
BNCH	- Any control command for the benchmark driver.

The return codes from the DSRs are summarized below:

Files

0 - Control operation successful
4 - File does not exist or is not available
8 - Hardware error or software inconsistency
12 - No access allowed to file
16 - Cannot wait to lock file due to deadlock
20 - Cannot lock file (not asked to wait to lock)
24 - Bad parameter in RENUMBER request
28 - Tried to renumber a file which is not a line file
32 - Inconsistent size requested
36 - No physical disk space available
40 - Account does not have enough file space allocated
44 - Error return from setting program key operation
48 - Error return from keyword scan operation
52 - Error return from setting privilege operation
56 - Error return from SAVE/NOSAVE operation
60 - Error return from TOUCH operation

UMnet/Merit

0 - Successful return
4 - Should not occur
8 - Control command not allowed--the remote host is attempting to send a record
12 - Successful command with returned text
16 - Connection is closed: no I/O may be done
20 - Invalid syntax or context for control command
24 - Attention interrupt received from network
64 - Internal network error

Magnetic Tapes

0 - Successful return
4 - End-of-file (BSR or FSR) or end-of-tape (FSF)
8 - Unit check
12 - End-of-tape
16 - Invalid CONTROL command or parameter, file not found (POSN), or permanent read/write error
20 - Attempt to write on unexpired file or without ring
24 - Fatal error
28 - Invalid volume, header, or trailer label
32 - Invalid I/O region or mode/blocking error
36 - Invalid blocking parameter
40 - Invalid mode
44 - Access not allowed

Floppy Disks
- 0 - Successful return
- 4 - Should not occur
- 8 - Should not occur
- 12 - Should not occur
- 16 - Should not occur
- 20 - Invalid CONTROL command or parameter

See the terminal and tape descriptions in MTS Volume 4, _Terminals and Networks in MTS_, and MTS Volume 19, _Tapes and Floppy Disks_, for further details on the different types of control commands that may be specified.

There is a macro CNTRL in the system macro library for generating the calling sequence to this subroutine. See the macro description for CNTRL in MTS Volume 14, _360/370 Assemblers in MTS_.

Example: FORTRAN:

```
            INTEGER*4 RET(27)
            INTEGER*2 LEN
            LEN = 3
            CALL CNTRL('REW',LEN,6,RET,&100,&200,&300)
              ...
      100   no control entry exit
              ...
      200   nonzero return code from DSR exit
              ...
      300   DSR error exit
```

Assembly:

```
            CALL CONTROL,(INFO,LEN,UNIT,RET)
            C    15,=F'12'
            BH   BADRC
            B    *+4(15)
            B    SUCCESS   normal exit
            B    ERROR1    no control entry exit
            B    ERROR2    nonzero DSR return code
            B    ERROR3    DSR error exit
             .
             .
      INFO  DC   C'REW'
      LEN   DC   Y(L'INFO)
      UNIT  DC   F'6'
      RET   DS   2F,CL100
```

The above examples set up a REW control command to the file or device attached to logical I/O unit 6.

COST

Subroutine Description

Purpose: To obtain the accumulated costs incurred by the current signon.

Location: Resident System

Calling Sequences:

```
Assembly: CALL COST

FORTRAN:  amount=COST(0)

PL/I(F):  amount=PLCALLF(COST,f0);
          amount=PLCALLE(COST,f0);
          amount=PLCALLD(COST,f0);
```

Parameter:

_f0_ is a fullword (FIXED BINARY(31)) location containing the integer zero.

Values Returned:

GR0 contains the cost of the current job in centicents (ten thousandths of a dollar).
FR0 contains the doubleword cost of the current job in dollars.

Return Codes:

 0 Successful return.
>0 Fatal error (should never occur).

Description: The result includes all billable amounts for the current signon to the time of the subroutine call with the exception of charges for _permanent_ file storage, tape-drive time for currently mounted tapes, unreleased paper-tape output, and open outbound Merit connections.

Examples: Assembly:

```
      CALL COST
      STD  0,CUR$
       .
       .
CUR$  DS   D
```

The above example returns the current cost in dollars in FR0 and stores the result in location CUR$.

FORTRAN:

```
INTEGER*4,CUM,REMAIN,COST
CALL GUINFO(22,REMAIN)
CALL GUINFO(32,CUM)
REMAIN=REMAIN-COST(0)-CUM
```

The above example calls the GUINFO subroutine to determine the maximum charge and cumulative charge used for the signon ID at the time of signon, calls COST to determine the cost of the current job, and then calculates a value for the charge remaining.

PL/I(F):

```
IF PLCALLF(COST,F0) > COSTLIM
   THEN GO TO END;
DECLARE PLCALLF RETURNS(FIXED BINARY(31)),
   COST ENTRY,
   F0 FIXED BINARY(31) INITIAL(0),
   COSTLIM FIXED BINARY(31);
```

The above example calls COST to determine whether the current job has exceeded a certain charge limit; if so, the program is terminated.

CREATE

Subroutine Description

Purpose: To create a file.

Location: Resident System

Alt. Entry: CREATE#

Calling Sequence:

Assembly: CALL CREATE,(name,size,vol,type)

FORTRAN: CALL CREATE(name,size,vol,type,&rc4,&rc8,&rc12,
&rc16,&rc20,&rc24,&rc28)

Parameters:

name is the location of the name (with a trailing blank) of the file to be created.

size is the location of a fullword integer containing two halfwords of information. The first halfword specifies the maximum expandable size of the file in pages (4096 bytes per page) or in tracks (7294 bytes per track); the type parameter indicates whether pages or tracks is being specified. If this halfword is zero, a default of 32,767 pages is used. The second halfword specifies the requested initial size of the file in pages or in tracks. The use of tracks is obsolete and is not recommended.

vol is the location of the name of the disk volume (as a six-character name) on which to create the file, or zero (the recommended value), in which case any available disk volume will be used.

type is the location of a fullword integer which indicates the type of file to create as well as whether the initial size and maximum expandable size requests are specified in pages or tracks.

- 0 - line file, sizes in tracks
- 1 - sequential file, sizes in tracks
- 2 - sequential-with-line-numbers file, sizes in tracks
- 256 - line file, sizes in pages
- 257 - sequential file, sizes in pages
- 258 - sequential-with-line-numbers file, sizes in pages

rc4,...,rc28 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return.
- 4 The file already exists.
- 8 Illegal _type_ parameter specified.
- 12 _Size_ parameter too large.
- 16 No space available for a file of that size.
- 20 Illegal parameter in calling sequence.
- 24 Hardware error or software inconsistency encountered.
- 28 The space allotted to this account has been exceeded.

Examples: Assembly:

```
         CALL  CREATE,(FNAME,FSIZE,FVOL,FTYPE)
           .
           .
   FNAME DC    C'DATAFILE '
   FSIZE DS    0F
   MSIZE DC    H'0'  Default maximum size
   ISIZE DC    H'1'  Initial size
   FVOL  DC    F'0'
   FTYPE DC    F'256'
```

FORTRAN:

```
         CALL CREATE('DATAFILE ',1,0,256,&100,&200)
```

These examples will create a line file by the name of DATAFILE with an initial size of 1 page and a default maximum expandable size of 32,767 pages.

CRYPT

Subroutine Description

Purpose: To encrypt or decrypt data according to a given user password.

Location: Resident System

Calling sequences:

Assembly: CALL CRYPT,(area,alen,flag,work,key,lkey)

FORTRAN: CALL CRYPT(area,alen,flag,work,key,lkey, &rc4,&rc8,&rc12,&rc16)

Parameters:

area is the location of the region that is to be processed by CRYPT. Upon return, the contents of the region will have been replaced by the converted data. This region must be at least 8 bytes long.

alen is the location of a fullword integer giving the length of _area_. It must be greater than 8.

flag is the location of a fullword integer indicating encryption or decryption (0=encryption; 1=decryption).

work is the location of a doubleword. Both words must be set to zero for the first call with a particular _key_ and not changed until a different _key_ is to be used.

key is the location of an encryption key. _key_ can be any length. _key_ must be positive.

lkey is the location of a fullword integer length of the encryption key.

_rc4,...,rc16_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return codes:

0 Successful return. _area_ contains converted data.
4 _alen_ was less than 8.
8 _flag_ was neither 0 nor 1.
12 _lkey_ was zero or negative.
16 Hardware error or software inconsistency.

Description: A call to this subroutine encrypts the line at location _area_ with length _alen_ using the *ENCRYPT algorithm. The encryption password used is _key_ with length _lkey_.

Upon initial entry to the subroutine, key is encrypted into an 8-byte doubleword and stored in the location work. This doubleword is used as an encryption code with a subroutine called DCRYPT, which takes three items as input. The first is a doubleword of data from area, the second is the computed value of work, and the last is the value of flag.

The DCRYPT subroutine is called repeatedly by CRYPT to encrypt successive doublewords from area. Each time the DCRYPT subroutine is called it performs a loop 32 times using two different bits of the key at each iteration. The first of these two bits indicates which of two translate tables is used to translate (using the machine translate instruction TR) the doubleword from. The two translate tables consist of distinct random permutations of all byte values from 0 to 255. The second bit is used to determine whether the doubleword is to be rotated by 3 or 5 bits. Finally, the iteration number is added to the low-order end of the 64-bit word.

The encryption algorithm is more efficient if area is fullword-aligned.

Further details on the algorithm can be found by looking at the source code (written in 360/370 assembler language) which is located in the file *ENCRYPT(2000).

CSGET, CSSET

Subroutine Description

Purpose: To enable the user to retrieve and set command status information.

Location: Resident System

Calling Sequences:

```
Assembly: CALL CSGET,(sumry,code,origin),VL

          CALL CSSET,(sumry,code,origin),VL

FORTRAN:  CALL CSGET(sumry,code,origin,&rc4)

          CALL CSSET(sumry,code,origin,&rc4)
```

Parameters:

sumry is the location of a fullword integer giving the error/status summary. The values may be:

0 - normal command status
1 - warning or informational message
2 - command error

Other values are illegal.

code is the location of a fullword integer giving more detailed information about the error/status summary. For MTS commands, the system will set the following values:

0 - normal command status
1 - untrapped attention interrupt
2 - untrapped program interrupt
3 - SVC error
4 - SVC EXIT
5 - untrapped timer interrupt
100 - command syntax error
101 - illegal with run-only program
102 - illegal in LSS (limited-state) mode
103 - only legal from CC Staff ccid
104 - only legal from privileged ccid
105 - error occurred while loading CLS
106 - error return from CLS
200 - unable to obtain sufficient storage
201 - user responded to prompt with CANCEL

For other commands, the system will set the value:

-1 - unassigned

In the future, each CLS and many public programs will have published lists of codes giving their error/status values. For the present, this value is almost always set to -1. User programs calling CSSET may select their own set of codes; the values must be ≥ -1.

<u>origin</u> (optional) is the location of a fullword giving the originator of the error/status information. If this parameter is omitted on a call to CSSET, the originator is set to -1 (indicating an undefined/undeclared state). Currently, only MTS sets the originator code to 1. In the future, each CLS and many public programs will have their own unique originator codes. For the present, user programs should either omit this parameter or set it to -1 when calling CSSET.

<u>&rc4</u> (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Status set or retrieved successfully.
4 Illegal call to CSGET or CSSET (illegal code, bad parameter list, no VL-bit specification, etc.).

Description: The CSGET subroutine may be used to retrieve command status information detailing the success or failure of a particular command. Currently, command status information is provided by the system for MTS commands. In the future, each CLS and many public files will provide more detailed command status information.

User programs may call CSSET to set private command status information. This information may be retrieved by a subsequent call to CSGET. Note: When using CSSET and CSGET with user programs, <u>sumry</u> may be set to zero if and only if <u>code</u> is set to zero.

This command status information is useful primarily in two situations:

(1) User programs that have called the COMMAND subroutine may call CSGET to determine whether the MTS command executed properly. The <u>sumry</u>, <u>code</u>, and <u>origin</u> values obtainable by calling CSGET are also available by specifying additional parameters

on the COMMAND subroutine.

(2) MTS command macros may be constructed to determine whether an MTS command executed properly. The <u>sumry</u>, <u>code</u>, and <u>origin</u> values are available as the predefined system macro variables CS_SUMMARY, CS_CODE, and CS_ORIGIN, respectively.

Examples: Assembly:

```
         CALL CSGET,(SUMRY,CODE,ORIGIN),VL
         .
         .
SUMRY    DS   F
CODE     DS   F
ORIGIN   DS   F
```

FORTRAN:

```
         INTEGER*4 SUMRY,CODE,ORIGIN
         CALL CSGET(SUMRY,CODE,ORIGIN)
```

DESTROY

Subroutine Description

Purpose: To destroy a file.

Location: Resident System

Alt. Entry: DESTRY

Calling Sequence:

Assembly: CALL DESTROY,(name)

FORTRAN: CALL DESTRY(name,&rc4,&rc8,&rc12,&rc16,&rc20,
&rc24,&rc28)

Parameters:

name is the location of the name (with a trailing blank) of the file to be destroyed.
rc4,...,rc28 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 name is not a file and therefore cannot be destroyed.
8 Reserved for future use.
12 File does not exist.
16 Locking the file for destroying will result in a deadlock.
20 Destroy access not allowed.
24 Error in calling parameter, hardware error, or software inconsistency encountered.
28 Automatic wait for (shared) file was interrupted.

If the return code is not zero, the file was not destroyed.

Note: If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from DESTROY with a return code of 28.

Examples: FORTRAN: CALL DESTRY('DATAFILE ',&2,&2,&9,&9,&99,&99,&99)

Assembly:

```
        CALL DESTROY,(FNAME)
         .
         .
FNAME DC   C'DATAFILE '
```

These examples will destroy the file DATAFILE.

DISMOUNT

Subroutine Description

Purpose: To release magnetic and paper tapes, Audio Response Unit lines, and connections on the Merit Computer Network.

Location: Resident System

Alt. Entry: DISMNT

Calling Sequences:

```
Assembly: CALL DISMOUNT,(string,len)

          CALL DISMOUNT,(par)

          DISMOUNT 'string'

FORTRAN:  CALL DISMNT(string,len)

          CALL DISMNT(par)
```

Parameters:

- _string_ is the location of a character string containing one or more pseudodevice names separated by blanks or commas.
- _len_ is the location of a halfword (INTEGER*2) length of _string_.
- _par_ is the location of a halfword (INTEGER*2) length of a character string immediately followed by that character string. The character string contains one or more pseudodevice names separated by blanks or commas.

Note: The DISMOUNT subroutine prints error messages on the logical I/O unit SERCOM or *MSINK* if SERCOM has not been assigned.

The complete description for using the DISMOUNT macro is given in MTS Volume 14, _360/370 Assemblers in MTS_.

Examples: Assembly:

```
             CALL DISMOUNT,(STR,LEN)
              .
              .
      LEN    DC   H'9'
      STR    DC   C'*T1* *T2*'
```

```
DISMOUNT '*T1* *T2*'
```

FORTRAN:

```
INTEGER*2 LEN
 ...
LEN=9
CALL DISMNT('*T1* *T2*',LEN)
```

The above three examples release the pseudodevices named *T1* and *T2*. The first assembly example uses the CALL macro and the second uses the DISMOUNT macro.

DUMP, PDUMP

Subroutine Description

Purpose: To print the values of specified memory regions in a FORTRAN program.

Location: *LIBRARY

Calling Sequences:

FORTRAN:
```
CALL DUMP(a1,b1,f1,...,an,bn,fn)

CALL PDUMP(a1,b1,f1,...,an,bn,fn)
```

Parameters:

_ai_ is a variable in the FORTRAN program specifying one end of the "i"th region to be printed.

_bi_ is a variable in the FORTRAN program specifying the other end of the "i"th region to be printed.

_fi_ indicates the format in which each data item between _ai_ and _bi_ is to be printed. _fi_ is a fullword integer and may be one of the following values:

0 - hexadecimal
1 - LOGICAL*1
2 - LOGICAL*4
3 - INTEGER*2
4 - INTEGER*4
5 - REAL*4
6 - REAL*8
7 - COMPLEX*8
8 - COMPLEX*16
9 - literal

Description: The DUMP and PDUMP subroutines print the values of the data items in the memory regions delimited by the _ai_ and _bi_ parameters. As many triples of parameters, _ai_, _bi_, and _fi_, may be given as desired. There is no order implied by the _ai_ and _bi_ parameters--either may mark the beginning or end of a region to be dumped. All output is printed on the logical I/O unit SERCOM.

The relative locations of the variables in a FORTRAN program may be obtained from the map produced by the MAP option to the FORTRAN compiler.

The only difference between DUMP and PDUMP is that DUMP terminates execution of the calling program by calling the system subroutine SYSTEM while PDUMP returns to the calling program.

Example: FORTRAN CALL DUMP(A(1),A(100),5,A(1),A(100),0)

The above example prints the values of the first 100 elements of the array A in both REAL*4 and hexadecimal format.

EBCASC

Translate Table Description

Purpose: To translate IBM EBCDIC characters into 8-bit ISO ASCII characters. An inverse table (ASCEBC) is also available.

Location: Resident System

Alt. Entries: IEBCASC, TREBCASC, TRIEA

Calling Sequences:

Assembly:
```
L    r,=V(EBCASC)
TR   d(l,b),0(r)
```

Parameters:

- r — is a general register that will contain the address of the EBCASC translate table.
- d(l,b) — is the location of the region to be translated. d is the displacement, l is the length of the region in bytes, and b is the base register for the region. This parameter may be given also in an assembly language symbolic format.

Description: The EBCDIC/ASCII translation table is shown on the next several pages. This table is for translating IBM Code Page 37 EBCDIC characters used in MTS into ISO 8859/1 8-bit ASCII characters. This table is also given in the file DOC:ALLCHARTABLE.

See the ASCEBC subroutine description for a table to translate from ASCII into EBCDIC.

Example: Assembly:
```
        L    6,=V(EBCASC)
        TR   REG(100),0(6)
         .
         .
REG     DS   CL100
```

FORTRAN:
```
LOGICAL*1 REG(100),TRTAB(256)
COMMON /EBCASC/TRTAB
 ...
CALL ITR(100,REG,0,TRTAB,0)
```

The above examples will translate the EBCDIC characters of the 100-byte region at location REG into ASCII characters.

The FORTRAN example uses the ITR entry point (see the description of the Logical Operators subroutines in this volume). In addition, a RIP loader record (RIP EBCASC) must be inserted into the object file to force the loader to resolve the symbol EBCASC from the low-core symbol table.

EDIT

Subroutine Description

Purpose: To call the MTS file editor from a user program.

Location: Resident System

Calling Sequence:

Assembly: CALL EDIT,(par1,par2,...,par16)

FORTRAN: CALL EDIT(par1,par2,...,par16,&rc4,&rc8,&rc12)

Parameters:

par 1 is the fullword editor dsect address; it is zero on the first call.
par 2 is a fullword integer '-1' or the CLS transfer vector.
par 3 is a fullword integer '-1' or the intermediate I/O routines transfer vector (see "Special Features" below).
par 4 is the initial file name to edit.
par 5 is the fullword length of initial file name.
par 6 is the initial EDIT command.
par 7 is the fullword length of the initial EDIT command.
par 8 is the fullword minimum line number allowed. Should be -2147483648 (-2**31) if not restricted. All line numbers are "internal," i.e., line 1.5 is represented as 1500.
par 9 is the fullword maximum line number allowed. Should be 2147483647 (2**31-1) if not restricted.
par 10 is the fullword line number relocation factor; the editor will subtract this number from the real line number in the file when interpreting line number parameters and printing verification.
par 11 is used to specify an external routine which examines all edit commands before the editor itself does. This routine may perform its own command scanning and provide additional services, return a modified command to the editor, instruct the editor to ignore the command, or signal an error condition. The editor may call this routine in either of two modes. The first mode is "scan only" which is used for syntax checking edit procedures,

etc. The second mode is "scan and execute" which intends for the editor to both parse and execute the command. The calling sequence for the external routine is as follows:

par 1 is the fullword address of the command to be examined.
par 2 is the fullword-integer length of the command.
par 3 is the fullword where the routine will place the address of the the command to be used by the editor.
par 4 is the fullword-integer length of the command to be used by the editor.
par 5 is a fullword integer indicating the mode:
0 = scan only
1 = scan and execute

The return codes from the routine are:

0 Editor should process command specified in par 3 and par 4.
4 Editor should ignore this command.
8 Error detected by routine; command suppressed.

par 11 in the call to EDIT should point to a V-type constant which either contains the address of the external routine to be used or an integer value of -1 (X'FFFFFFFF'); the -1 means no external routine is to be used.

par 12 is not used (must be fullword integer '-1' or zero parameter pointer).

par 13 are editor control switches that are specified as a fullword integer sum of the following. The actions of the following first 4 switches are performed in the order listed.

1	X'01'	set edit file using par 4 and par 5
2	X'02'	perform one-shot EDIT command, using par 6 and par 7, and return immediately. X'02' and X'04' are mutually exclusive; if both are specified, X'04' is ignored.
4	X'04'	read commands from SOURCE
8	X'08'	unload editor unconditionally on return
16	X'10'	prohibit EDIT command except for editing edit procedures

32 X'20'	prohibit MTS commands from the editor
64 X'40'	prohibit copy from or to external files
128 X'80'	return on any error
256 X'100'	return on null length editor command
512 X'200'	return on first ATTN
1024 X'400'	do not unload editor on STOP command or EOF in command stream
2048 X'800'	set initial current line number before any commands are processed on this call (par 15)
4096 X'1000'	ignore initialization file specified by $SET INITFILE(EDIT) command

The following parameters and par 1 are set on return:

par 14 is a 20-byte area to store current file name on return.
par 15 is the fullword current line number.
par 16 is a fullword to store the integer sum of the edit procedure switches on return:

1 - EOF switch enabled
2 - SUCCESS switch enabled
4 - return from STOP command or EOF in command stream

par 17 (optional) is the address of the caller's PSECT which is passed as an additional parameter to the user's command prescan routine.

rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return codes occur.

Return codes:

0 Normal return, editor unloaded.
4 Normal return, editor not unloaded.
8 Error return, editor not unloaded.
12 Error return, editor system error.

Example: This example is written in FORTRAN.

```
      INTEGER*4 EDWD/0/,FILENM(5),EDSW,LINE/3000/
C
C CALL THE EDITOR TO ALTER "C" TO "B" IN LINE 3.000 OF
C FILE -TESTF
C 2059 = 1+2+8+2048 WHICH ARE THE CONTROL SWITCHES FOR:
C            1  SET EDIT FILE
C            2  PERFORM INITIAL EDIT COMMAND
C            8  UNLOAD EDITOR WHEN RETURNING
C         2048  SET INITIAL CURRENT LINE POINTER
      CALL EDIT(EDWD,-1,-1,'-TESTF',6,'ALTER  * "C"B"',13,
     X-99999999,99999999,0,-1,-1,2059,FILENM,LINE,
     XEDSW,&2,&9,&9)
C
C EDSW WILL BE '2' IF ALTER WAS SUCCESSFUL, '0' IF NOT.
    2 PRINT 5,FILENM,EDSW
    5 FORMAT(1X,5A4,I10)
C
      STOP 0
    9 STOP 1
      END
```

Special Features:

The remainder of this subroutine description provides information on special features of the EDIT subroutine that are of interest to system programmers; <u>knowledge of these special features is not required to call EDIT in the manner described above</u>.

Normal editing occurs when <u>par 3</u> points to a fullword '-1'. To use the special features described here, <u>par 3</u> must point to an ordered vector of fullword subroutine addresses or zeros. Nonzero entries allows the user to provide alternate subroutines that replace those normally used by the editor. User-supplied routines allow the assembly language user to preprocess and postprocess file data. It is also possible to support user-implemented file organizations. This special facility is not intended for use from FORTRAN programs.

A small amount of knowledge about the structure of the editor is required to properly use the alternate subroutine interface. The accompanying diagram is a representation of the way the editor reads and writes files.

Level 7 represents the program calling the editor. MTS uses the editor command language subsystem (CLS) interface while other programs generally use the more complete "user interface". The editor in turn calls upon a set of routines which perform buffering and checkpoint operations. These then call a set of file-independent rou-

```
                    +------------------------------------+
     LEVEL 1        + MTS FILE ROUTINES - all file types |
                    +-----------------+------------------+
     LEVEL 2                          |
       +------------------------------+--------------------------------+
       | EDITOR I/O SUPPORT   -  set of routines for all file types    |
       +------------------------------+--------------------------------+
                                      |
               +----------------------+-----------------------+
     LEVEL 3   | OPTIONAL USER-SUPPLIED INTERMEDIATE ROUTINES |
               +----------------------+-----------------------+
     LEVEL 4                          |
       +------------------------------+-------------------------------+
       | EDITOR FILE MANAGEMENT    buffering, checkpoint-restore-undo |
       +------------------------------+-------------------------------+
                                      |
                +---------------------+-----------------------+
     LEVEL 5    | EDITOR COMMAND LANGUAGE AND DATA PROCESSING |
                +------------------------------------+--------+
                                                     |
               +----------------------+  +-----------+---------------+
     LEVEL 6   |"EDITOR" CLS INTERFACE<-->"EDIT" SUBROUTINE INTERFACE|
               +---------+------------+  +-----------+---------------+
                         |                           |
              +----------+---------------+ +---------+-----------------+
     LEVEL 7  |"$EDIT" MTS COMMAND SYSTEM| | FTN,SPIRES, user programs |
              +--------------------------+ +---------------------------+
```

tines. The file-independent routines of level 2 try to remove all irregularities in file access and also process all errors. For example, the READ INDEXED routine is given a line number and returns the line, length, and line number. A nonexistent line is represented by zero length. If an error occurs, a special error message routine is called by the file-independent routines. A message and severity level are included as parameters. The editor supplies the address of the routine to handle these errors. Attentions are handled in a similar manner.

The editor supplies the location of a switch which either inhibits or allows attentions to be processed at that point. If attentions are disabled and one occurs, the routines are responsible for calling the attention-handling routine when attentions are again permitted.

The user may supply his own version of the file-independent routines which in turn may or may not call the editor's. This is useful for modifying lines before the editor sees them. For example, a FORTRAN preprocessing system may use this to concatenate continued statements and provide statement indentation for loops and if-then structures on input, while splitting and unediting them on output.

File Independent Routine Descriptions:

The file-independent routines all use a storage area similar to an MTS FDUB called the "IODSECT". The EDGET routine (see the description below) is called by the editor to get a file, allocate storage for the IODSECT, and initialize it. The address of the IODSECT is stored in the fullword specified by the first parameter to EDGET. All of the remaining I/O routines must receive this as their first parameter in the calling sequence. The EDREL routine (see the description below) releases the IODSECT and all other storage acquired for such processing. All of the remaining I/O routines return a return code greater than zero only if the first parameter is not a valid IODSECT. The routines will buffer up to one line in VM and will not reread it if successive calls request that same line. A write is always executed to insure that the most recent version has been received by the MTS file routines. The routine's "current line" (not to be confused with * in the editor itself) is the last line accessed. The line number returned by the routines will always indicate the position in the file even if the line is not present (zero length). If the line number returned is 2147483647 (2**31-1), there is no current line or file position. Sequential files without line numbers, tape files, and other file types will have lines numbered starting with 1.000 and increments of 1.000. A call from the editor to any of these routines may be replaced with a user-supplied routine which behaves the same way from the viewpoint of the editor. The third parameter to the EDIT subroutine is a vector of entry points to these replacement routines. The user-supplied routine may in turn call any of the I/O routines described below if so desired, as long as they return the proper information to the editor.

EDGET - GET NEW FILE AND IODSECT

- par 2 file name (if shorter than len (par 3), delimit with blank).
- par 3 fullword length of name (maximum is 20 characters).
- par 4 fullword minimum accessible line number. Lines with numbers less than this will appear not to be in the file.
- par 5 fullword maximum accessible line number. Lines with numbers greater than this will appear not to be in the file.
- par 6 fullword relocation factor to the line number. The offset is subtracted from line numbers on input and added on output. Thus

an offset of 1000000 will make line 1000.000 look like line zero.

par 7 1-byte pad character if required by I/O routines.

par 8 error message routine. Calling sequence described elsewhere.

par 9 attention routine entry point (has no calling parameters). Described below in the section "Attention Processing."

par 10 1-byte attention bit described below.

par 11 1-byte attention hold count described below.

par 12 CLS transfer vector.

par 13 virtual memory file chain header (supplied by editor). The editor I/O routines use this to locate edit procedures.

Returns:

par 1 fullword address of IODSECT.

par 14 CL20 actual file name.

par 15 FDUB for file.

par 16 fullword file type code.

0	user-supported file type (no editor support)
4	file type is "NONE"
8	editor "edit procedure"
12	MTS line file
16	MTS sequential file
20	tape file
24	"other" file type

par 17 fullword maximum input-output length.

par 18 fullword current maximum input length. Minimum will always be 255.

EDSET - SET MIN MAX OFFSET LINE NUMBERS AND PAD CHARACTERS

par 1 IODSECT.

par 2 minimum accessible line number.

par 3 maximum accessible line number.

par 4 offset to line number (user sees this added to real number).

par 5 returns current maximum input-output length.

par 6 returns current maximum input length.

par 7 pad character if required by I/O routines.

EDREL - RELEASE FILE AND IODSECT

par 1 IODSECT.

EDCLO - CLOSE FILE AND INVALIDATE CURRENT BUFFER

Used when user requests the closing of the file.

par 1 IODSECT.

EDENT - ENTER ROUTINES AFTER EXIT FROM EDITOR

Used when editor restarts after possible external operations on the file being edited.

par 1 IODSECT.

EDRIX - READ INDEXED ROUTINE

par 1 IODSECT.
par 2 fullword line number to be used as index for read. -2147483648 and 2147483647 mean *F and *L, respectively.

Returns:

par 3 fullword length of record read. Zero means that record was not found but line number was made the current file position.
par 4 fullword line number.
par 5 fullword location of the record. The caller must not modify this region.

EDRSQ - READ SEQUENTIAL ROUTINE

par 1 IODSECT.
par 2 fullword number of records to read forward or backward from current. Zero means stay at current record. 1 means read next record and -1 means read previous record; 2 means read the second record after the current, and -2 means the second previous record before the current record, etc.

Returns:

par 3 fullword line length. Zero means no record (EOF or empty file).
par 4 fullword line number.
par 5 fullword address of record read.

EDWIX - WRITE INDEXED

par 1 IODSECT.
par 2 fullword new length.
par 3 fullword line number. *F or *L not allowed here.
par 4 new line data EDWIX makes it active line also.

EDSPA - FIND AVAILABLE LINE NUMBER SPACE AFTER CURRENT RECORD

par 1 IODSECT.

Returns:

par 2 fullword number of lines that can actually be inserted.
par 3 fullword line number of first line that may be inserted.
par 4 fullword minimum allowed increment.
par 5 fullword last unused line number in region.

EDRNM - RENUMBER OPERATION

par 1 IODSECT.
par 2 fullword first line number.
par 3 fullword last line number.
par 4 fullword begin line number.
par 5 fullword increment to line number.

EDCNT - COUNT NUMBER OF LINES BETWEEN TWO LINES

par 1 IODSECT.
par 2 fullword first line number.
par 3 fullword last line number.
par 4 returns fullword number of lines (inclusive).

EDGLN - GET VECTOR OF LINE NUMBERS

par 1 IODSECT.
par 2-5 same as par 2-5 of RETLNR subroutine.

EDPLN - PUT VECTOR OF LINE NUMBERS

par 1 IODSECT.
par 2-5 same as par 2-5 of SETLNR subroutine.

EDUNLK - UNLOCK FILE

Unlock the edit file.

par 1   IODSECT.

EDWRBF - WRITE CHANGED FILE BUFFERS

Used when editor temporarily returns to caller and the file could be modified thereby invalidating the current line.

par 1   IODSECT.

ERROR MESSAGE ROUTINE - supplied by editor

par 1   the message.
par 2   fullword message length.
par 3   fullword message severity:

0 Comment, return after printing
1 Warning, return after printing
2 Error, do not return
3 Severe error in editor, do not return

par 4   fullword message number.

Attention Processing:

Attention hold count is a one-byte count. If a routine enters a sensitive area of code, i.e., one that must not be interrupted, this count is incremented by one. A nonzero count tells the attention trap exit routine to set the attention bit byte to X'00' to indicate that an attention has occurred and to return to the point of attention. When the sensitive region of code is left, the attention hold count must be decremented by one. If the count goes to zero at that point, the attention bit must be examined for X'00' with the test and set instruction (which resets it to X'FF'). If it is zero the attention routine must be called to process the attention in the normal manner. This allows all levels of routines independent attention control in sensitive areas. The error routine resets attention hold count and attention bit on errors with severity greater than "warning". The user must be certain to reset attention hold count when leaving the sensitive area so as to enable interrupts.

I/O Routines Transfer Vector:

par 3 to the editor interface may point to a fullword '-1', which means there is no special transfer vector and the normal editor routines are used. Otherwise par 3 points to an ordered vector of fullword routine addresses or zeros. A zero in any position means that the normal editor I/O routine is to be used, otherwise the address is used instead of the normal routine. The vector order is defined to be:

0 '14' - fullword integer number of entries in vector
1 EDGET - get new file and IODSECT
2 EDREL - release file and IODSECT
3 EDCLO - close file and invalidate current buffer
4 EDRIX - read indexed routine
5 EDRSQ - read sequential routine
6 EDWIX - write indexed
7 EDSPA - find available line number space after current record
8 EDRNM - renumber operation
9 EDCNT - count number of lines between two lines
10 EDGLN - get vector of line numbers
11 EDPLN - put vector of line numbers
12 EDSET - set minimum and maximum offset line numbers and pad character
13 EDUNLK - unlock edit file
14 EDWRBF - write all changed buffers of edit file

The above routines are available in the resident system through LCSYMBOL.

EMPTY

Subroutine Description

Purpose: To empty a file without destroying it.

Location: Resident System

Calling Sequence:

Assembly: (a)

```
L    0,fdub
CALL EMPTY
```

(b)

```
LM   0,1,lname
CALL EMPTY
```

Parameters:

(a) GR0 contains an FDUB-pointer (such as returned by GETFD) or an integer logical I/O unit number (0 through 99), or

(b) GR0 and GR1 contain a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

Return Codes:

0 Successful return.
4 The file does not exist.
8 Hardware error or software inconsistency encountered.
12 Empty access not allowed.
16 Locking the file for modification will result in a deadlock.
20 Automatic wait for shared file was interrupted.

Notes: FORTRAN programs should call the EMPTYF subroutine.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from EMPTY with a return code of 20.

When a file is emptied, the entire contents of the file are discarded. The EMPTY subroutine cannot be used to empty only a portion of a file.

Example: Assembly:

```
        LA    1,FNAME
        CALL  GETFD
        ST    0,FDUB
        CALL  EMPTY
         .
         .
FNAME   DC    C'DATAFILE '
FDUB    DS    F
```

This example will empty the file DATAFILE.

EMPTYF

Subroutine Description

Purpose: To empty a file without destroying it.

Location: Resident System

Alt. Entry: EMPTYS

Calling Sequences:

Assembly: CALL EMPTYF,(unit)

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

Parameters:

unit is the location of either
- (a) a fullword-integer FDUB-pointer (such as returned by GETFD),
- (b) a fullword-integer logical I/O unit number (0 through 99), or
- (c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

rc4,...,rc20 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 File was emptied successfully.
4 The file does not exist.
8 Hardware error or software inconsistency encountered.
12 Empty access not allowed.
16 Locking the file for modification will result in a deadlock.
20 Automatic wait for shared file was interrupted.

Notes: EMPTYF (and EMPTY) handles MTS logical I/O units rather than FORTRAN I/O units. EMPTYF cannot handle I/O unit numbers greater than 19. If the EQUATE FTNCMD command has been used, MTS units 0 to 19 may not correspond to FORTRAN units 0 to 19.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from EMPTYF with a return code of 20.

When a file is emptied, the entire contents of the file are discarded. The EMPTYF subroutine cannot be used to empty only a portion of a file.

Examples: Assembly:

```
                CALL EMPTYF,(UNIT)
                  .
                  .
      UNIT  DC  CL8'SCARDS'
```

FORTRAN:

```
                CALL EMPTYF('SCARDS  ')
```

These examples will empty the file attached to SCARDS.

ERROR

Subroutine Description

Purpose: To suspend execution with an error indication.

Location: Resident System

Alt. Entry: ERROR#

Calling Sequence:

Assembly: CALL ERROR

or

ERROR

FORTRAN: CALL ERROR

Note: The complete description for using the ERROR macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

Description: A call to this subroutine returns control to MTS or to the previous command language subsystem. If the return is made to MTS command mode, the comment "ERROR RETURN" is printed. In batch mode, a dump is automatically given if $SET ERRORDUMP=ON was specified.

The program is not unloaded. The contents of registers and program storage may be inspected to determine the cause of the error. The execution return code is set to 8. This may be tested by the $IF command, e.g.,

```
$IF RUNRC=8, mts-command
```

The execution return code is displayed under the control of the $SET RCPRINT option (see MTS Volume 1, The Michigan Terminal System) and the GUINFO item LASTEXRC (239).

Execution of the suspended program may be restarted from the point of suspension by the $RESTART command or the CONTINUE debug command in debug mode.

This subroutine is intended to be used in situations in which the program can detect an internal error in its program logic or execution, e.g., illegal data, unexpected results, etc.

FILEINFO

Subroutine Description

Purpose: To return information about a file.

Location: Resident system.

Calling Sequences:

```
Assembly: CALL FILEINFO,(what,type,item1,loc1,...,
                         itemn,locn),VL

FORTRAN:  CALL FILEINFO(what,type,item1,loc1,...,
                        itemn,locn,&rc4,...,&rc28)
```

Parameters:

<u>what</u> is either:
(a) a file name (in either of two formats),
(b) a fullword FDUB pointer, an eight character I/O unit name, or a fullword logical I/O unit number, or
(c) a CATSCAN workarea pointer.

<u>type</u> if a fullword enumerated type describing <u>what</u>:
1 - a file name formatted as a halfword length followed by the file name (trailing blanks are not allowed).
2 - a file name formatted as the file name with one or more trailing blanks.
3 - a fullword FDUB pointer, an eight character logical I/O unit name, or a fullword logical I/O unit number.
4 - a workarea pointer returned by the CATSCAN subroutine.

<u>itemn</u> is an 8-character item name (padded with blanks). The item names may be in uppercase only.

<u>locn</u> is an area to return the information associated with <u>itemn</u>. The format of this area depends on the item requested. The legal items and the format of the returned information is given in the table below. The <u>item</u> and <u>loc</u> parameters are always specified in pairs.

&rc4,...,&rc28 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Caller parameter error.
8 Insufficient access for the requested information.
12 No access to the file.
16 File does not exist.
20 File-wait deadlock.
24 File-wait interrupt.
28 Hardware/software inconsistency.

Description: If FILEINFO is called with more than one parameter, the return code will be zero if and only if all of the parameters are successfully processed. If an access error (return code 4) or a parameter error (return code 8) occurs for one of the items in a multi-item call, then all of the return value locations will have unpredictable values. Also, a parameter or access error may mask other parameter or access errors.

The information return by FILEINFO is described in the table below.

FILEINFO is the preferred subroutine to use to return information about a file. FILEINFO can be used in conjunction with the CATSCAN subroutine to obtain information about a group of files.

Name	Format	Description
CINAME	Variable	File name, formatted as follows: The first fullword denotes the number of bytes (including this fullword) supplied by the user for this field. The second fullword is reserved for FILEINFO, and will denote the number of bytes (including this and the preceding fullwords) necessary for the file name (this value may be greater than the number of bytes supplied). This is followed by the actual file name.
CIONID	4 bytes	CCID of owner (EBCDIC characters)
CIVOL	8 bytes	Volume name (EBCDIC characters)
CIUC	Fullword	Use count for file
CIFO	Fullword	File organization: 0=line, 1=sequential, 2=SEQWL
CIDT	Fullword	Device type: 0=2311, 1=2314, 2=2321, 3=3330, 4=3350
CIFLG	Fullword	Flags: 1 - Priv 2 - Nosave
CIPKEY	16 bytes	PKEY for file
CILCCT	STCK	Last change time for contents of file
CILNCCT	STCK	Last change time for non-contents
CICT	STCK	Creation time
CILRT	STCK	Last reference time
FIFLAG	Fullword	1=backwards reads possible; 2=empty file
FIEMPTY	Fullword	EMPTY flag: 0=not empty, 1=empty
FICNS	Fullword	Current size in pages
FITS	Fullword	Truncated size in pages
FICPS	Fullword	Copies (or duplicated) size in pages
+FIFLN	Fullword	First line number (0 if empty)
+FILLN	Fullword	Last line number (0 if empty)
FIMLL	Fullword	Maximum line length
FIMXS	Fullword	Maximum expandable size (pages)
*+FINL	Fullword	Number of lines
*+FINH	Fullword	Number of chunks available
*+FILCNT	Fullword	Total bytes - lines
*+FIHCNT	Fullword	Total bytes - holes
*+FIMHL	Fullword	Maximum length available space
FIXF	Fullword	Expansion factor as follows: >0 - Absolute expansion in pages; 0 - Default used (currently 10%); <0 - Percent to expand
SIACC	Fullword	Access for this CCID/proj.-number/pkey expressed as a sum of the value for each access type allowed: 1 - read access allowed; 2 - write extend access allowed; 4 - write change/empty access allowed

```
                      8 - renumber/truncate access allowed
                     16 - destroy/rename access allowed
                     32 - permit access allowed
SIGA      Fullword  Global (OTHERS) access as above
SIOA      Fullword  Owner access as above
SIVAR     Variable  String of specific sharing information formated as
                    a varying field.

                    The first fullword denotes  the  number  of  bytes
                    (including this fullword) supplied by the user for
                    sharing "nodes".

                    The  second fullword is reserved for FILEINFO, and
                    will denote the number of  bytes  (including  this
                    and the preceding fullwords) necessary for sharing
                    "nodes".

                    A  value of zero means that no sharing information
                    is present.  This value may be  greater  than  the
                    actual number of bytes supplied.

                    Each sharing "node" is formatted as:
                      fullword - length of this "node"
                      fullword - access for this entity
                      fullword - type flag as follows:
                        0 - project number
                        1 - CCID
                        2 - pkey
                        3 - project number and pkey
                        4 - CCID and pkey

                      if project number present:
                        fullword - project number length
                        4 chars  - project number

                      if CCID present:
                        fullword - CCID present
                        4 chars  - CCID

                      if pkey present
                        fullword - pkey length
                        varying  - pkey
```

If a pkey is qualifying a CCID or project number, the pkey length and the pkey will be contained within the same sharing "node", where

+ - indicates information available only for LINE files
* - indicates expensive information

The format STCK indicates a doubleword time value in the same format as that returned by the STCK machine instruction.

FNAMETRT

Translate Table Description

Purpose: A 256-byte translate table to check the legality of a file name.

Location: Resident System

Alt. Entry: FNTRT

Calling Sequence:

Assembly:

```
SR   2,2
L    r,=V(FNAMETRT)
TRT  name,0(r)
```

Parameters:

- r — is a general register containing the address of the FNAMETRT translate table.
- name — is the location of the file name to be tested.

Values Returned:

GR2 will contain a value indicating the result of the test:

- 0 - legal file name without a legal terminator.
- 1 - legal file name with legal terminator.
- 2 - name contains a character that is illegal for the CREATE or RENAME subroutine (the remainder of the name may or may not be illegal).
- 3 - illegal file name.

The condition code is set to zero if the result is a legal file name without a legal terminator; otherwise, it is set to 1 or 2.

A file name may contain the letters A-Z (upper- or lowercase), the digits 0-9, and the following special characters:

< > $ * - % # / . _ !

The following characters terminate a file name:

blank (+ , @ X'FF'

If the file belongs to another signon ID, it must be specified without using the shared file separator character, e.g., 2AGADATAFILE specifies the file DATAFILE belonging to signon ID 2AGA.

Example: Assembly:

```
        SR   2,2
        L    3,=V(FNAMETRT)
        TRT  FNAME,0(3)
        BZ   EXIT           No legal terminator
        C    2,=F'1'
        BH   ERROR          Illegal file name
         .
         .
FNAME   DS   CL16           File name
```

FORTRAN:

```
      LOGICAL*1 FNAME(16),TRTAB(256)
      COMMON /FNTRT/TRTAB
        ...
      I = ITRT(16,FNAME,0,TRTAB,0,N,L)
      IF (I.EQ.0) GO TO 10
      IF (L.GT.1) GO TO 20
C     File is OK.
        ...
10    No legal terminator
        ...
20    Illegal character
```

The above examples test for the legality of the file name contained in FNAME.

The FORTRAN example uses the ITRT subroutine (see the description of the Logical Operators subroutines in this volume). In addition, a RIP loader record (RIP FNTRT) must be inserted into the FORTRAN object file to force the loader to resolve the symbol FNTRT from the low-core symbol table.

FREAD/FWRITE

Subroutine Description

Purpose: To provide a free format input/output facility, especially for FORTRAN programs.

Location: *LIBRARY

Calling Sequences:

```
FORTRAN:  CALL FREAD(unit,string,list,...,&rc4,&rc8,&rc12)

          CALL FWRITE(unit,string,list,...)

Assembly: CALL FREAD,(unit,string,list,...),VL

          CALL FWRITE,(unit,string,list,...),VL
```

Parameters:

unit is the location of one of the following:
(a) a FDUB-pointer,
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a character-string logical I/O unit name such as 'SCARDS' or 'SPRINT', or the character string 'PAR' or '*'.
This parameter indicates where input is to be read from or the output is to be written to.

string is the location of a string of characters (a literal or an array of characters) indicating how many and what types of variables are to be read or written. A type string consists of a sequence of type codes separated by commas. For FWRITE, this string is written without conversion except for the type codes which are enclosed in angle brackets (<,>).

list is a list of variable or array names, separated by commas, into which the data values are to be read or from which the data values are to be written. In the case of an array, the entry is a pair - the first member is the array name and the second member is the location of the number of elements to be read into the array.

rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return code occurs.

Description: The FREAD subroutine reads a specified amount of data in free format in response to each call. The data items to be read may appear in free format in the input records, i.e., in any position in the record, separated by blanks, commas, or other delimiters selected by the user. The amount of data to be read is indicated by the list of variables in the <u>list</u> parameter. The type of data item to be read into each variable location is determined by the type codes in the <u>string</u> parameter. There is a one-to-one correspondence between type codes and variable names in the <u>list</u> parameter.

The FWRITE subroutine writes onto a specified unit with the <u>string</u> parameter which must terminate with one of the following characters:

- ; implies that the output line is incomplete (the next call to FWRITE can add output to the same line).
- : implies that the output line is complete and should be written out.

Type codes are enclosed within angle brackets (<,>) and specify the type of conversion to be performed. There is a one-to-one correspondence between type codes and variable names in the <u>list</u> parameter.

FREAD and FWRITE have the special entry points FREADB, FREADC, FWRITB, and FWRITC. FREADB and FWRITB are used to read from or write to a user-specified buffer. FREADC and FWRITC are used to set or reset various switches that control subsequent FREAD and FWRITE actions.

For further information on the FREAD and FWRITE subroutines, see the section "FREAD/FWRITE: Free Format I/O Subroutines" in MTS Volume 6, <u>FORTRAN in MTS</u>.

Examples: FORTRAN: CALL FREAD('SCARDS','I:',J)

The above example reads an integer from SCARDS and places its value into the variable J.

CALL FWRITE(9,'<I><I>:',I,J)

The above example writes two integers onto logical I/O unit 9 from the variables I and J.

CALL FREAD(5,'R VECTOR:',VEC,13)

The above example reads 13 real numbers from logical I/O unit 5 into the array VEC.

FREEFD

Subroutine Description

Purpose: To release a file or device acquired by the GETFD subroutine.

Location: Resident System

Alt. Entries: FREEFDS, FREFDS

Calling Sequences:

```
Assembly: L    0,fdub
          CALL FREEFD

          CALL FREEFDS,(fdub),VL

FORTRAN:  CALL FREFDS(fdub,&rc4)
```

Parameters:

<u>fdub</u> (GR0) is a FDUB-pointer (such as returned by CHKFDUB, GDINFO, or GETFD).

<u>&rc4</u> (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Invalid FDUB-pointer or no VL bit specified.

Description: A call on the FREEFDS or FREFDS subroutines takes the S-type parameters and loads them into an R-type call on the FREEFD subroutine.

Examples:

```
Assembly:       L    0,FDUB
                CALL FREEFD

FORTRAN:        CALL FREFDS(FDUB,&4)
```

The above examples free the file or device associated with the FDUB-pointer in FDUB.

FREESPAC

Subroutine Description

Purpose: To release storage acquired by the GETSPACE subroutine.

Location: Resident System

Alt. Entries: FREESP, FREESPAS, FRESPS

Calling Sequences:

Assembly:
```
L     0,len
L     1,loc
CALL  FREESPAC

or

FREESPAC loc[,LNG=len][,EXIT=err]

CALL FREESPAS,(len,loc),VL
```

FORTRAN: `CALL FRESPS(len,loc,&rc4,&rc8)`

Parameters:

<u>len</u> (GR0) is either zero or the length of the block to return. If zero, the region (beginning from the address contained in GR1 and extending through to the end of the region originally acquired by GETSPACE) is to be released. If not zero, GR0 is the length of the region to be released. If it is not a multiple of 8, the next smallest multiple of 8 is used.

<u>loc</u> (GR1) is the location of the first byte of the region to be released. If it is a not a multiple of 8, the next larger multiple of 8 will be used.

<u>&rc4,&rc8</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

A GR13 save area is not required for a call to this subroutine.

Return Codes:

0 Successful return.

4 Error return. Either the region was not initially allocated by GETSPACE and cannot be released (the region either does not exist or is a part of the

resident system), or the region specified (loc to loc to loc+len-1) is not completely within a region originally allocated by GETSPACE.

8 VL bit not specified.

Notes: The Array Management Subroutines described in this volume also may be used to allocate and release storage.

The complete description for using the FREESPAC macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

Description: A call on the FREESPAS or FRESPS subroutines takes the S-type parameters and loads them into an R-type call on the FREESPAC subroutine.

Examples: Assembly:

```
SR   0,0
L    1,LOC
CALL FREESPAC

FREESPAC LOC
```

FORTRAN:

```
CALL FRESPS(0,LOC,&4)
```

The above three examples call FREESPAC to release the entire region whose starting address is contained in the location LOC. The first uses the CALL macro and the second uses the FREESPAC macro.

```
         L     0,LEN
         L     1,LOC
         CALL FREESPAC
          .
          .
LEN      DC   F'32'

         FREESPAC LOC,LNG=32
```

The above two examples call FREESPAC to release the first 32 bytes of the region whose starting address is contained in the location LOC.

FSIZE

Subroutine Description

Purpose: To determine the file size required to contain a certain amount of information without actually writing the file.

Location: Resident System

Calling Sequences:

Assembly: CALL FSIZE,(type,length,size)

FORTRAN: CALL FSIZE(type,length,size,&rc4)

Parameters:

- _type_ is the location of a fullword integer containing the file type:
  0 - line file
  1 - sequential file
  2 - sequential-with-line-numbers file
- _length_ is the location of a fullword integer containing the length of the current line which would be written into the file.
- _size_ is the location of a 16-word integer array (64 bytes). The first word is zero on the first call, and contains the current size in pages on subsequent calls (returned on each call). The second word is the "last pointer" as it would be returned by the NOTE subroutine for sequential or sequential-with-line-numbers files. The remainder of _size_ is used by FSIZE for internal storage between calls and should not be altered.
- _rc4_ is the statement label to transfer to if the equivalent return code occurs.

Return Codes:

0 Successful return (information returned normally).
4 Invalid parameter.

Description: The FSIZE subroutine is used to determine the minimum file size required to contain a specific set of data lines without actually writing them into a file. The subroutine must be called once for each line which would be written into the file. Before the first call, the first word of _size_ should be set to zero; on subsequent calls, only the _length_ parameter should be changed. The first word of *size* will contain the minimum file size required to contain the accumulated number of lines following each call.

Examples: Assembly:

```
           LA    2,100
      LOOP CALL  FSIZE,(TYPE,LEN,SIZE)
           BCT   2,LOOP
            .
            .
      TYPE DC    F'0'
      LEN  DC    F'50'
      SIZE DC    16F'0'
```

FORTRAN:

```
           INTEGER SIZE(16)
            ...
           SIZE(1) = 0
           DO 100 I=1,100
      100  CALL FSIZE(0,50,SIZE)
```

These examples compute the minimum size required for a line file containing 100 50-byte lines. This value will be contained in SIZE(1).

FSRF, BSRF

Subroutine Description

Purpose: To forward space or backspace records (lines) in a line file or sequential file.

Location: Resident System

Calling Sequence:

Assembly: CALL FSRF,(unit,skipct)

CALL BSRF,(unit,skipct)

FORTRAN: CALL FSRF(unit,skipct,&rc4,&rc8,&rc12,&rc16,
&rc20,&rc24)

CALL BSRF(unit,skipct,&rc4,&rc8,&rc12,&rc16,
&rc20,&rc24)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

skipct is the location of a fullword-integer count of the number of logical records (lines) to forward or backspace over.

rc4,...,rc24 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Records skipped successfully.
4 End-of-file encountered.
8 Illegal unit parameter, or hardware error or software inconsistency encountered.
12 Read or write access not allowed.
16 Locking the file for read will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent usage of the shared file).
24 The file does not exist.

Notes: For both line and sequential files, a current (line or read) pointer is maintained. Forward spacing or backspacing begins from the current pointer. See Appendix B of the section "Files and Devices" in MTS Volume 1, The Michigan Terminal System, for details concerning how this current pointer is updated as a result of various I/O operations.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from FSRF or BSRF with a return code of 20.

Examples: Assembly:

```
         CALL  FSRF,(UNIT,SKIPCT)
          .
          .
UNIT     DC    F'1'
SKIPCT   DC    F'2'
```

The above example will forward space two logical records (lines) on the file attached to logical I/O unit 1.

FORTRAN:

```
INTEGER*4 UNIT
DATA UNIT/1/
 ...
CALL BSRF(UNIT,2)
```

The above example will backspace two logical records (lines) on the file attached to logical I/O unit 1.

FTNCMD

Subroutine Description

Purpose: To allow a program to issue commands to the FORTRAN I/O library.

Location: Resident System

Calling Sequence:

FORTRAN: CALL FTNCMD(string,length)

Parameters:

- _string_ is the location of a character string that consists of the FORTRAN I/O library command.
- _length_ is the location of a fullword or halfword (INTEGER*4 or INTEGER*2) giving the length of _string_. This may be set to zero if a semicolon is used to terminate the character string.

Description: The FTNCMD subroutine allows a program to issue commands to the FORTRAN I/O library monitor in order to manipulate the I/O environment. Any command that is legal for the FORTRAN I/O library monitor may be given. In addition, an MTS command may be specified by prefixing the command with a dollar sign ($). The subroutine returns to the calling program unless an erroneous FORTRAN monitor command is specified, in which case the FORTRAN I/O monitor assumes control.

The FORTRAN I/O library and monitor are described in the section "FORTRAN I/O Library" in MTS Volume 6, _FORTRAN in MTS_.

Examples:

```
CALL FTNCMD('ASSIGN 7=*PUNCH*',16)
```

The above example assigns logical I/O unit 7 to *PUNCH*.

```
CALL FTNCMD('SET UVCHECK=OFF;',0)
```

The above example suppresses the FORTRAN I/O library checking for undefined variables.

GDINF

Subroutine Description

Purpose: To allow a FORTRAN program to obtain information returned from the subroutine GDINFO.

Location: *LIBRARY

Calling Sequence:

FORTRAN: CALL GDINF(unit,region,&rc4)

Parameters:

- unit — is the location of either
  - (a) a FDUB-pointer (as returned by GETFD),
  - (b) an 8-character logical I/O unit name left-justified with trailing blanks (e.g., SCARDS, SPRINT, 0 through 99, etc.), or
  - (c) an integer logical I/O unit number (0-99).
- region — is a 44-byte array (11 fullwords) in which the information is returned.
- rc4 — (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return.
- 4 Error. See the GDINFO subroutine description for the possible error conditions.
- 8 Hardware or software inconsistency.

Description: This subroutine calls the GDINFO subroutine and places the returned information in region which is provided by the FORTRAN calling program. See the description of the GDINFO subroutine in this volume for a description of this information. Note that only the first eleven words of GDINFO information is returned.

Example: FORTRAN:

```
          INTEGER*4 REG(11)
           ...
          CALL GDINF('SPUNCH  ',REG,&99)
           ...
       99 WRITE(6,199)
      199 FORMAT(' SPUNCH IS NOT ASSIGNED')
```

This example calls GDINF to obtain information about the file or device attached to SPUNCH.

GDINFO

Subroutine Description

Purpose: To obtain information about a file or device.

Location: Resident System

| Alt. Entries: GDINFOS, GDINFS

Calling Sequence:

```
Assembly:             L    0,fdub
                      CALL GDINFO

                      LM   0,1,name
                      CALL GDINFO

                      CALL GDINFOS,(unit,info),VL

           FORTRAN:   CALL GDINFS(unit,info,&rc4,&rc8)
```

Parameters:

<u>fdub</u> (GR0) is a FDUB-pointer (such as returned by GETFD) or an integer logical I/O unit number (0 through 99), or

<u>name</u> (GR0 and GR1) is a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

<u>unit</u> is the location of either
- (a) a fullword-integer FDUB-pointer (as returned by GETFD),
- (b) a left-justified, 8-character logical I/O unit name (e.g., CL8'SPRINT'),
- (c) a fullword-integer logical I/O unit number between 0 and 99, inclusive.

<u>info</u> is a pointer to a GETSPACE-allocated block of storage to contain the information about the specified unit.

<u>&rc4,&rc8</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return. GR1 or <u>info</u> holds the information requested (see below).
4 Error return. Illegal FDUB-pointer, illegal name, no file or device attached to specified I/O unit name or number, or no VL bit set.
8 Hardware error or software inconsistency.

Values Returned:

If the return code from GDINFO is zero, then GR1 contains the location of a fullword-aligned region of information. (If a concatenation was specified in the original logical I/O unit setup or GETFD call, the information returned in this region applies to the currently active member of the concatenation.) The region contains:

WORD 1: FDUB-pointer (in general, the FDUB-pointer returned here should not be used by programs; instead, the logical I/O unit name or number or the FDUB-pointer used to call GDINFO should be used).

WORD 2: 4-character BCD type (see below)

WORD 3: Maximum input length (halfword) and maximum output length (halfword)

"Var" means variable. The value returned depends on the current value of the blocking parameters (for tapes), the LEN device command (for terminals), the INLEN and OUTLEN device commands (for MNET), and the length of the maximum line (for files).

Input	Output	Type	Usage
Var	32767	FILE	- line file
Var	32767	SEQF	- sequential file
0	0	NONE	- nonexistent or invalid file or device, access not allowed, wait (on locked file) interrupted, or cannot wait due to deadlock
Var	Var	TTY	- Teletype
Var	Var	2741	- IBM 2741, 1050 Terminals
Var	Var	PDP8	- Data Concentrator
Var	Var	MRXA	- Memorex 1270 Controller
255	255	DISP	- IBM 2250 Display Station
160	80	2260	- IBM 2260 Display Station
255	Var	3270	- IBM 3270 Display Station
254	0	HRDR	- batch card input
0	133	HPTR	- *PRINT* output
0	80	HPCH	- *PUNCH* output
0	254	HBAT	- *BATCH* output
160	0	2501	- IBM 2501 Card Reader
160	0	RDR	- IBM 2540 Card Reader
0	80	PCH	- IBM 2540 Card Punch
0	133	PTR	- IBM 1403 Printer

0	133	1443	- IBM 1443 Printer
0	133	3211	- IBM 3211 Printer
Var	Var	9TP	- 9-track Magnetic Tape
Var	Var	7TP	- 7-track Magnetic Tape
0	255	PTPP	- Paper Tape Punch
Var	0	PTPR	- Paper Tape Reader
Var	Var	SDA	- Synchronous Data Adaptor
255	255	7772	- IBM 7772 ARU
0	32767	DUMY	- *DUMMY*
100	100	OPER	- Operator job
255	255	TEST	- variable
Var	Var	MNET	- Merit Computer Network
128	128	1052	- IBM 1052 Terminal
255	Var	3066	- IBM 3066 Console
255	132	BNCH	- benchmark driver

WORD 4: Byte 1 - FDUBTYPE field:
- 0 = other
- 1 = *MSOURCE*
- 2 = *MSINK*
- 3 = *PUNCH*
- 4 = *SOURCE*
- 5 = *SINK*
- 6 = *AFD*
- 7 = device mounted by $MOUNT command
- 8 to 255 reserved for future expansion

Byte 2 - type index:
- 0 = unit record
- 1 = magnetic tape
- 2 = terminal
- 3 = file
- 4 = dummy
- 5 = paper tape
- 6 = operator’s console
- 7 = test
- 8 = NONE or illegal type
- 9 to 255 reserved for future expansion

Byte 3 - switches:
- bit 0 - on if output is OK
- bit 1 - on if input is OK
- bit 2 - on if indexed operation makes sense
- bit 3 - on if can be rewound
- bit 4 - on if increment given in FDname
- bit 5 - on if defaulted on $RUN cmd.
- bit 6 - on if part of explicit concatenation and not last member
- bit 7 - on if at least one modifier was given on the FDname

Byte 4 - switches:
- bit 0 - explicit beginning line number

given
bit 1 - explicit ending line number given
bit 2 - FCB/device is open
bit 3 - info length (high-order half of word 14) is present
bit 4 - macro processing is enabled for this FDUB
bit 5 - last record returned was generated by macro processor

WORD 5: I/O modifiers (first word)

WORD 6: Starting line number

WORD 7: Last line number used in I/O operation

WORD 8: Ending line number

WORD 9: Line number increment

WORD 10: Pointer to FDname for current FDUB (halfword length followed by FDname), or zero

WORD 11: Pointer to last error message associated with FDUB (halfword length followed by message), or zero

WORD 12: Pointer to I/O error exit savearea (if SETIOERR has been called), or zero

WORD 13: Return code from last I/O subroutine call

WORD 14: GDINFO information region length in bytes (halfword) and device-carriage/screen width (or -1, if unknown) (halfword)

WORD 15: Macro processor invocation ID if macro processing is enabled for this FDUB

WORD 16: I/O modifiers (second word)

Notes: The line numbers given in words 6, 7, 8, and 9 are the line numbers associated with the FDname. These are given in internal format, which is the external format (specified on the FDname) times 1000.

GDINFO opens the file or device (and, if a file, locks the file for reading) in order to obtain the maximum input and output lengths. If opening and/or locking a file might cause unwanted waiting or possible deadlocks, and if the maximum lengths are not desired, the subroutines GDINFO2 or GDINFO3 should be called instead.

If GDINFO is used to return information about a concatenation of FDnames, the information returned refers to the current member of the concatenation.

The storage pointed to by GR1 was allocated by GETSPACE, and the user may call FREESPAC (with GR0 = 0) to release it when it is no longer needed. This storage region was allocated only if GDINFO gave a return code of zero.

The file use count and last reference date are not updated by a call to GDINFO (or GDINFO2 or GDINFO3).

The setting of bit 3 in byte 15 (GDLENSW in GDSWS2) can be used to determine if the GDINFO info region length (first halfword in word 14, GDLEN) is present. GDLEN can be used to determine if the items following GDLEN are present.

Description: A call on the GDINFOS subroutine takes the S-type parameters and loads them into an R-type call on the GDINFO subroutine.

The information returned by GDINFO is described by the dsect given on the following page (from the file *GDINFODSECT).

```
*****************************************************************
*
*        Dsect for information returned by GDINFO subroutine
*
*              (Last revised on July 20, 1985 )
*
*****************************************************************
GDDSECT  DSECT
GDFDUB   DS    A                 FDUB pointer
GDTYPE   DS    CL4               Type
GDINLEN  DS    H                 Input maximum length
GDOUTLEN DS    H                 Output maximum length
GDUTYP   DS    X                 Use type:
GDMSOURC EQU   1                   Master source
GDMSINK  EQU   2                   Master sink
GDPUNCH  EQU   3                   Batch punch output
GDSOURCE EQU   4                   Source
GDSINK   EQU   5                   Sink
GDAFD    EQU   6                   Active file
GDMOUNTD EQU   7                   Allocated by $MOUNT command
GDDTYP   DS    X                 Device type:
```

```
GDUNIREC EQU   0                   Unit record (incl. *...*)
GDMAGTAP EQU   1                   Magnetic tape
GDTERM   EQU   2                   Terminal
GDFILE   EQU   3                   Disk file (line or sequential)
GDDUMMY  EQU   4                   *DUMMY*
GDPAPTAP EQU   5                   Paper tape reader
GDOPER   EQU   6                   Operator's console
GDTEST   EQU   7                   DSR test device
GDNONE   EQU   8                   None: device does not exist
GDSWS    DS    X                 Switches:
GDOUTOK  EQU   X'80'               Output allowed
GDINOK   EQU   X'40'               Input allowed
GDINDXOK EQU   X'20'               @Indexed operations make sense
GDREWOK  EQU   X'10'               Can be rewound
GDEXINCR EQU   X'08'               Explicit increment was given
GDDEFLT  EQU   X'04'               Defaulted
GDCONCAT EQU   X'02'               Not the last member of expl concat
GDEXMOD  EQU   X'01'               Explicit modifiers given
GDSWS2   DS    X                 More switches:
GDEXBLN  EQU   X'80'               Explicit beg. line number given
GDEXELN  EQU   X'40'               Explicit ending line number given
GDOPEN   EQU   X'20'               FCB/Device is open
GDLENSW  EQU   X'10'               Information length is present
GDMACON  EQU   X'08'               Macro processing is enabled
*                                    for this FDUB
GDMACGEN EQU   X'04'               Last record returned was
*                                    generated by the macro processor
GD_RPC   EQU   X'02'               GDINFO info returned by RPC
GDMODS   DS    XL4               Modifiers on the FDname
GDBLNR   DS    F                 Beginning line number
GDPLNR   DS    F                 Previous line number
GDELNR   DS    F                 Ending line number
GDILNR   DS    F                 Increment for line number
GDNAME   DS    A                 Locn of external name
GDERMSG  DS    A                 Locn of last error message
GDERSA   DS    A                 Locn of I/O error exit save area
GDLASTRC DS    F                 Last I/O subroutine call return code
GDLEN    DS    H                 Length of returned information
GDWIDTH  DS    H                 Terminal carriage or screen width
GDMACID  DS    A                 Macro processor invocation ID if
*                                  macro processing is enabled
*                                  for this FDUB
GDMODS2  DS    XL4               Second word of modifiers on FDname
GDDSCTL  EQU   *-GDDSECT         Length of returned information
```

Example: Assembly:

```
      LM   0,1,SNAME
      CALL GDINFO
       .
       .
SNAME DC   CL8'SPRINT  '
```

The above example calls GDINFO to get information for the file or device attached to the logical I/O unit SPRINT.

GDINFO2

Subroutine Description

Purpose: To get information about a file or device.

Location: Resident System

| Alt. Entries: GDINF2, GDINFO2, GDIN2S

Calling Sequence:

```
| Assembly:                L    0,fdub
|                          CALL GDINFO2
|
|                          LM   0,1,name
|                          CALL GDINFO2
|
|                          CALL GDINFO2S,(unit,info),VL
|
|              FORTRAN:    CALL GDIN2S(unit,info,&rc4,&rc8)
```

Description: This subroutine is exactly the same as the GDINFO subroutine with the following exceptions:

(1) The file or device is not opened, and (if a file) is not locked.
(2) If the file or device is not already open, the input and output lengths are set to -1 to indicate that they are unknown.

GDINFO3

Subroutine Description

Purpose: To get information about a file or device.

Location: Resident System

Alt. Entries: GDINF3, GDINFO3S, GDIN3S

Calling Sequence:

Assembly:

```
          L    0,fdub
          CALL GDINFO3

          LM   0,1,name
          CALL GDINFO3

          CALL GDINFO3S,(unit,info),VL
```

FORTRAN:

```
          CALL GDIN3S(unit,info,&rc4,&rc8)
```

Description: This subroutine is exactly the same as the GDINFO subroutine with the following exceptions:

(1) The file or device is opened, but (if a file) is not locked.
(2) If a file, and it is not already locked, the input length is set to -1 to indicate that it is unknown.

GETFD

Subroutine Description

Purpose: To obtain a file or device.

Location: Resident System

Alt. Entry: GETFDS

Calling Sequence:

```
Assembly: LA   1,fdname
          CALL GETFD

          CALL GETFDS,(fdname,fdub),VL

FORTRAN:  CALL GETFDS(fdname,fdub,&rc4,&rc8,&rc12)
```

Parameters:

fdname (GR1) is the location of the first character of the FDname of the file or device wanted. The complete name must be terminated by a blank. The name does not have to be aligned.

fdub is the memory location in which to store the pointer of the obtained file or device.

_&rc4,&rc8,&rc12_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return. _fdub_ or GR0 holds the returned pointer, or the file or device is nonexistent, inaccessible, or invalid (see GDINFO).
4 Invalid address or illegal parameter.
8 Device is busy.
12 Device is not operational.

GETFD will give a zero return code for nonexistent, nonaccessible, or invalid file or device names. The type code given by word 2 of the information area from GDINFO, GDINFO2, or GDINFO3 can be used to check for the status of the file or device. This type code should always be tested for the validity of the result from GETFD since nonzero return codes are rarely returned by GETFD. A type code of "NONE" will indicate a nonvalid result from GETFD.

Values Returned:

GR0 contains the FDUB-pointer if a successful return is made.

Description: If the name is a device, the device is acquired. If the name is a file, the file is not opened until the first usage. Thus this subroutine cannot determine whether or not the file exists. The caller can determine whether the file exists by calling GDINFO. The name may be a concatenation of file or device names each followed by modifiers or a line number range as described in "Files and Devices" in MTS Volume 1, The Michigan Terminal System. If the FDUB-pointer returned is used in a call to READ or WRITE, the modifiers or line number ranges will be used, and if a concatenation was specified, the usual sequencing through the concatenation will take place.

A call on the GETFDS subroutine takes the S-type parameters and loads them into an R-type call on the GETFD subroutine.

Example:

Assembly:

```
          LA   1,FNAME
          CALL GETFD
            .
            .
    FNAME DC   C'DATAFILE '
```

FORTRAN:

```
LOGICAL*1 FNAME(9)/'DATAFILE '/
CALL GETFDS(FNAME,FDUB,&4)
```

The above examples call GETFD to obtain an FDUB-pointer for the file DATAFILE.

GETFST, GETLST

Subroutine Description

Purpose: To return the line number associated with the first or last line in a file, respectively.

Location: Resident System

Calling Sequence:

```
Assembly: CALL GETFST,(unit,linenb)

          CALL GETLST,(unit,linenb)

FORTRAN:  CALL GETFST(unit,linenb,&rc4,&rc8,&rc12,&rc16,
                      &rc20,&rc24)

          CALL GETLST(unit,linenb,&rc4,&rc8,&rc12,&rc16,
                      &rc20,&rc24)
```

Parameters:

unit is the location of either
- (a) a fullword-integer FDUB-pointer (as returned by GETFD),
- (b) a fullword-integer logical I/O unit number (0 through 99), or
- (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

linenb is the location of a fullword in which the *internal* line number (either first or last) will be returned.

rc4,...,rc24 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Line number returned successfully.
4 The file is empty.
8 Unaddressable parameter or hardware/software inconsistency.
12 Access not allowed (something other than NONE required).
16 Locking the file for read will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent usage of the shared file).
24 The file does not exist.

Notes: GETFST and GETLST may be used only with line files or sequential-with-line-numbers files.

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

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from GETFST or GETLST with a return code of 20.

Examples: Assembly:

```
         CALL  GETFST,(UNIT,FSTLN)
          .
          .
UNIT     DC    CL8'SPRINT'
FSTLN    DS    F         Put first line number here
```

The above example returns the first line number associated with the file attached to logical I/O unit SPRINT.

FORTRAN:

```
      INTEGER*4 UNIT,LSTLN
      DATA UNIT/3/
       ...
      CALL GETLST(UNIT,LSTLN)
```

The above example returns the last line number associated with the file attached to logical I/O unit 3.

GETIME

Subroutine Description

Purpose: To return the time remaining until a specified timer interrupt will occur without canceling the interrupt.

Location: Resident System

Calling Sequences:

Assembly: CALL GETIME,(id,value,aregion)

FORTRAN: CALL GETIME(id,value,aregion,&rc4)

Parameters:

- _id_ is the location of the fullword identifier which specifies the timer interrupt whose time remaining until interruption is to be returned. This is the same identifier which was given to SETIME when the interrupt was set up.
- _value_ is the location of a 4-, 8-, or 16-byte fullword-aligned region in which GETIME returns the time remaining until the interrupt will occur. The interpretation of this value depends upon the _code_ parameter given to SETIME when the interrupt was set up. For codes 0 and 2, the value is an 8-byte binary integer specifying microseconds of task CPU time; for codes 1, 3, and 5, the value is an 8-byte binary integer specifying microseconds of real time; for code 4, the value is a 4-byte binary integer specifying timer units of task CPU time.
- _aregion_ is the location of the address of the 76-byte exit region which was given to SETIME when the interrupt was set up. The combination of the identifier and the exit region address will always specify a unique timer interrupt.
- _rc4_ (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 No such timer interrupt was found. This means either:

(1) no such interrupt was ever set up, or
(2) the interrupt has occurred, and the exit was taken before the execution of the BALR instruction which branches to GETIME.

Description: A call on the GETIME subroutine returns the time remaining until a specified timer interrupt will occur without canceling the interrupt. The timer interrupt is specified by the combination of the id and aregion parameters and the time remaining is returned in the value parameter.

For further details, see also the RSTIME, SETIME, and TIMNTRP subroutine descriptions in this volume.

FORTRAN users should consult the TICALL subroutine description in this volume for details on using timer interrupts with FORTRAN.

Example: Assembly:

```
                 CALL GETIME,(ONE,TIMLEFT,AREG)
                  .
                  .
         ONE     DC  F'1'
         TIMLEFT DS  FL8
         AREG    DC  A(REG)
         REG     DS  19F
```

FORTRAN:

```
      EXTERNAL EXIT
      INTEGER TIME(2)/0,10000/,LEFT(2),TICALL
        ...
      IREG = TICALL(0,EXIT,TIME,&4,&8)
      CALL GETIME(EXIT,LEFT,IREG,&4)
```

The above example, coded in assembly language and FORTRAN, returns the time remaining for the interrupt with the identifier 1 and exit region REG. The value is returned in TIMLEFT.

GETSPACE

Subroutine Description

Purpose: To acquire storage.

Location: Resident System

| Alt. Entries: GETSPA, GETSPACS, GETSPS

Calling Sequences:

```
Assembly: L     0,switch
          L     1,length
          CALL  GETSPACE

          L     0,switch
          L     1,length
          L     2,index
          CALL  GETSPACE

          GETSPACE [length][,T=switch][,EXIT=err]

          CALL GETSPACS,(switch,length,index,addr),VL

FORTRAN:  CALL GETSPS(switch,length,index,addr,&rc4,&rc8)
```

Parameters:

| *switch* (GR0) is a fullword of binary switches:

Bit 31	= 1	Return not made unless space is available.
	0	Return always made with return code indicating whether space is available.
30	= 1	Storage acquired is associated with the current level of LINK so that it is released at the next return from a LINK, or the next XCTL. This bit is ignored if bit 28 is set.
	0	Storage acquired is associated with the highest level program so that it is not released until execution terminates.
28	= 1	Use storage index number in general register 2.
27	= 1	Allocate storage in the virtual machine segment (ignored if an

explicit segment number is given in general register 1).

Other bits in GR0 must be zero.

length (GR2) is the length (in bytes) of storage desired. If this is not a multiple of 8, the next largest multiple of 8 will be used. The upper limit for a storage request is 1,048, 576 bytes (1 segment).

Normally space will be allocated wherever available in virtual memory. However, if the first byte (byte 0) of GR1 is nonzero, it is assumed to be the number of the segment in which the storage is to be allocated. If this is an invalid number [is less than 6, or is greater than the maximum (currently 12)], or if this space request cannot be allocated in this segment, a return is made with a return code of 4.

index (optional) (GR2) is the storage index number to associate with the allocated block. If index is specified, the corresponding bit in switch (bit 28) must be 1.

addr is the returned address of the allocated block.

&rc4,&rc8 (optional) are statement labels to transfer to if a nonzero return code occurs.

A GR13 save area is not required for a call to this subroutine.

Values Returned:

GR1 contains the location of the first byte of the storage region acquired. The first word of this region is set to the length (in bytes) of the region.

Return Codes:

0 Successful return. Storage has been acquired.
4 Space is not available.
8 Illegal parameter or no VL bit specified.

Notes: The Array Management subroutines described in this volume also may be used to allocate and release storage.

The complete description for using the GETSPACE macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

Description: A call on the GETSPACS subroutine takes the S-type parameters and loads them into an R-type call on the GETSPACE subroutine.

See the "Virtual Memory Management" section in MTS Volume 5, System Services, for further details on storage allocation and storage index numbers.

Examples: Assembly:

```
        L    0,SWITCH
        L    1,LENGTH
        CALL GETSPACE
         .
         .
 SWITCH DC   F'0'
 LENGTH DC   F'256'
```

FORTRAN:

```
       INTEGER SPACE
       CALL GETSPS(0,256,0,SPACE,&400)
```

The above two examples call GETSPACE to acquire 256 bytes of storage. The storage will be associated with the highest level program.

GFINFO

Subroutine Description

Purpose: To obtain information about a particular file or (when called repeatedly) all of the files in a particular catalog.

Location: Resident System

Calling Sequences:

Assembly:
```
CALL GFINFO,(what,rtn,flag,cinfo,finfo,sinfo,
             ercode,errmsg),VL
```

FORTRAN:
```
CALL GFINFO(what,rtn,flag,cinfo,finfo,sinfo,
            ercode,errmsg,&rc4)
```

Parameters:

what — is the location of either
(a) an FDname (with a trailing blank), if flag bits 29-31 are 001,
(b) a fullword-integer FDUB-pointer (such as returned by GETFD), a fullword-integer logical I/O unit number (0 through 99), or a left-justified, 8-character logical I/O unit name (e.g., SCARDS), if flag bits 29-31 are 010,
(c) a 4-character signon ID of a catalog to be scanned, or *SYS (system file catalog), or *TMP (temporary file catalog), if flag bits 29-31 are 011, or
(d) a file-name pattern (with a trailing blank) containing question marks "?" as the match character, (e.g., 1CRB:A?, -?, TEST?DATA, *PASCAL?), if flag bits 29-31 are 100. The pattern algorithm is the same as that described for the $FILESTATUS command.

rtn — is the location of a 6-fullword integer region where the file name will be returned. If flag bits 29-31 are 001, this parameter on return will be the same as what. If flag bits 29-31 are 010, this parameter on return will be the file name associated with the FDUB-pointer or logical I/O unit. If flag bits 29-31 are 011, this parameter on return will be the file name of the next file in the catalog being scanned, for which the requested information has been returned. If _flag_ bits 29-31 are 100, this parameter on return will be the name of a file that matches the pattern and for which requested information has been returned. The last word of this region _must_ be zero when GFINFO is called initially. In addition, this region should not be altered on subsequent calls if a catalog is being scanned (_flag_ bits 29-31 are 011 or 100) or if storage is being released (_flag_ bits 29-31 are 000). The file name returned is a maximum of 5 fullwords (20 characters) left-justified and padded with trailing blanks. The last word is used internally by GFINFO.

flag is the location of a fullword integer of flags which affect the interpretation of the _what_ and _finfo_ parameters. The flags are as follows:

Bits 29-31:

- 000 Any storage allocated by GFINFO should be released. This should be specified, for example, to release the variable-length sharing list if such was specified, or to release storage if a catalog scan was terminated prematurely. If a catalog scan is terminated normally via the "NO MORE FILES" error return, all storage will be released automatically and the caller need not release it.
- 001 The _what_ parameter denotes the name of a file.
- 010 The _what_ parameter denotes a FDUB pointer for a file.
- 011 The _what_ parameter indicates a catalog name to scan.
- 100 The _what_ parameter contains a file-name pattern. A scan will be performed on the appropriate catalog to search for the matching file names.

Bit 28: If 1, this indicates the _finfo_ information returned should only contain items which are not "expensive" to retrieve (see "Notes" below). Note that if FIAL is less than 12, no expensive information is returned nor retrieved from the indicated

file.
Bits 0-27: Should be zero.

cinfo is the location of a 25-fullword region (array) where catalog information will be returned. The first word of the region indicates the size of the region (in words). If this is set to less than the maximum of 25, the caller is requesting that only the first "n" words of information are to be returned. If this word is set to zero, the caller is requesting that _no_ catalog information is to be returned. The second word of the region indicates how much information (in words) was actually returned by GFINFO. If the second word is zero on return, no information was returned because the appropriate access to the file was not allowed. Any access (other than none) is sufficient to obtain the catalog information.

finfo is the location of a 18-fullword region (array) where file information will be returned. The first two words of the region are as described for the _cinfo_ parameter. Any access (other than none) is sufficient to obtain the file information.

sinfo is the location of a 6-fullword region (array) where sharing information will be returned. The first and second words of the region are as described for the _cinfo_ and _finfo_ parameters. Any access (other than none) is sufficient to obtain the third word of information, i.e., the access the caller has to the file. Permit access is required to obtain complete access information; otherwise, only the access relevant to the current userID/project number is returned. Note that if the first word of the region is 5 or less, no variable-length sharing information will be returned. In addition, if the second word of the region is 3 or less on return, the current user has no access to the file. Finally, if the variable-length sharing information is requested and returned, the associated storage must be released either directly by calling FREESPAC or indirectly by calling GFINFO again with _flag_=0 and nothing else altered.

ercode (optional) is the location of a fullword integer in which GFINFO will place an error number if an error return (return code 4) is made. If _ercode_ is omitted, then the _errmsg_ parameter must also be omitted. Assembly language users wishing to omit these parameters should either follow the variable-length parameter list convention (high-order bit of the previous parameter adcon in the parameter list is 1) or else supply an adcon which is zero (rather than pointing to a zero).

errmsg (optional) is the location of a 20-fullword (80-character) region in which GFINFO will place the corresponding error message if an error return (return code 4) is made. Assembly language users should note the convention for omitting optional parameters described above.

Ercode	Errmsg
1	Parameter list is pointer bad
2	Your "file" is not a file
3	The file does not exist
4	No file this CCID - catalog scan
5	No more files - catalog scan
6	No access allowed - file xxxx
7	Waiting will deadlock - file xxxx
8	Wait interrupted - file xxxx
9	Hardware error or software inconsistency - file xxxx
10	Hardware error or software inconsistency - system catalog
11	Insufficient access for requested information - file xxxx
12	Invalid pattern was specified.
21	First parameter (what) is bad
22	Second parameter (rtn) is bad
23	Third parameter (flag) is bad
24	Fourth parameter (cinfo) is bad
25	Fifth parameter (finfo) is bad
26	Sixth parameter (sinfo) is bad

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from GFINFO with an error code of 8.

rc4 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Some information has been returned.
4 Error return. See the _ercode_ and _errmsg_ values returned for the specific error.
8 Error return. Invalid parameter addresses were given. No error code or error message is available.

Notes:

(1) On a catalog scan, if no information is requested, i.e., _cinfo_=_finfo_=_sinfo_=0, _rtn_ on return will contain the name of the next file for which some access (other than none) has been allowed.
(2) The catalog information is the least expensive to obtain, the sharing information is moderately expensive, and the file information is most expensive. Concerning the file information as it relates to line files only, the copied size as well as the last five words of information (i.e., number of lines, etc.) are quite expensive to determine. Consequently, if the first eleven words (or less) of file information are requested for a line file, only an approximation of the copied size will be returned. If any or all of the last five words are requested, a more accurate (but still approximate) copied size will be returned.
(3) The public file *GFINFODSECT contains 3 dsects for assembly language users which define the format of the catalog information, file information, and sharing information. Proper use of these dsects will enable user programs to adapt easily to any additional information GFINFO may return in the future.
(4) The file use count and the last reference date are _not_ updated by a call to the GFINFO subroutine.
(5) Specifying a file-name pattern will scan the catalog given or implied from the pattern, e.g.,

```
1CRB:A?   scans catalog for 1CRB
TEST?DATA scans catalog for current ID
-?        scans catalog for *TMP
*PASCAL?  scans catalog for *SYS
```

*TMP is the system catalog for temporary files; *SYS is the system catalog for public files.

Description: The information returned by GFINFO is described by the following dsects (from the file *GFINFODSECT).

```
****************************************************************
*
*  *GFINFODSECT  consists of CIDSECT, FIDSECT, SIDSECT
*
*              (Last revised on January 17, 1984)
*
*        Catalog Information DSECT - Any access is sufficient
*              to obtain the catalog information.
*
*              All dates in CIDSECT are returned as a Julian
*              date; that is, the number of days from March 1,
*              1900.
*              All times in CDSECT are returned in Store Clock
*              Units; that is, (the number of microseconds since
*              January 1, 1900) * (4096).
*
****************************************************************
CIDSECT  DSECT
CIAL     DS    F                Array Length - Num of words requested
CIRL     DS    F                Return Length - Num of words returned
CIONID   DS    CL4              OwnerID - In EBCDIC
CIVOL    DS    CL6              Volume Name - In EBCDIC
         DS    CL2              Blanks - unused
CIUC     DS    F                Use Count
CILRD    DS    F                Last Reference Date - a Julian date
*                               Obsolete. Use of CILRD_T is preferred
CICD     DS    F                Creation Date - a Julian date
*                               Obsolete. Use of CICD_T is preferred
CIFO     DS    F                File Organization:
*                                 0=LINE,1=SEQ,2=SEQWL
CIDT     DS    F                Device Type
*                                 0=2311,1=2314,2=2321,3=3330,4=3350
CIFLG    DS    F                Bit flags as follows:
CIPRIV   EQU   1                  Priviledged program
CINOSAVE EQU   2                  No file save requested
*
CILCD    DS    F                Last Changed Date - Julian date
*                                 This is the more recent of the
*                                 dates of the last contents and last
*                                 non-contents changes. This value
*                                 is obsolete. Use of CILCCT and
*                                 CILNCCD_T is preferred.
CIPKEY   DS    CL16             Program key: 1-13 characters
CILCCT   DS    2F               Last Contents Change Time - Store
*                                 Clock Units. This is accurate  to
*                                 one second but may become more
*                                 accurate in the future.
CILNCCD  DS    F                Last Non-Contents Change Date
*                                 - Julian date. This is the date
*                                 that information about the file
```

```
*                                (NOT the actual contents of the
*                                file) was last changed. Use of
*                                CILNCCD_T is preferred for this
*                                value.
*
*        Note: The following are the Non-Contents change,
*              Creation, and Reference dates returned in Store
*              Clock Units. Currently, these times represent
*              midnight of the corresponding date. In the future
*              these times may be kept with more accuracy.
*
CILNCCD_T DS   2F              Last Non-Contents Change Date Time
CICD_T   DS    2F              Creation Date Time
CILRD_T  DS    2F              Last Reference Date Time
CILEN    EQU   *-CIDSECT
*******************************************************************
*
*        File Information DSECT - Any access is sufficient to
*             obtain the File Information.
*
*        Note: * = expensive information - zeroed if only cheap
*                  information requested
*              + = information available for line files only
*
*******************************************************************
FIDSECT  DSECT
FIAL     DS    F               Array Length - Num of words requested
FIRL     DS    F               Return Length - Num of words returned
FIFO     DS    F               File Organization
*                                0=LINE,1=SEQ,2=SEQWL
FIFLG    DS    F               Flag
*                                1 = Backwards capability
*                                2 = Empty file
FICNS    DS    F               Current Size - pages
FITS     DS    F               Truncated Size - pages
FICPS    DS    F               Copied size - pages (see below)
FIFLN    DS    F               First Line Number - internal repr.
*                                Zero if file is SEQ or empty
FILLN    DS    F               Last Line Number - internal repr.
*                                Zero if file is SEQ or empty
FIMLL    DS    F               Maximum Length Line
FIMXS    DS    F               Maximum expandable file size - pages
FINE     EQU   FIMXS           Number of Extents - not returned
FINL     DS    F               Number of Lines*+
FINH     DS    F               Number of chunks of available space*+
FILCNT   DS    F               Total bytes - lines*+
FIHCNT   DS    F               Total bytes - available space*+
FIMHL    DS    F               Maximum length available space*+
FIXF     DS    F               File expansion factor (see below)
FIMBC    DS    F               Maximum Buffer Count
FILEN    EQU   *-FIDSECT
*
*        Note:  The format of information returned in FIXF is as
```

```
*               follows:  If the expansion factor is an absolute
*                         amount, the value is the absolute amount;
*                         If the expansion factor is a percentage,
*                         is the value is the percentage expressed
*                         as a negative number.
*
*                         If the expansion factor is zero then the
*                         default expansion factor is used
*                         (currently 10%).
*
*               The value returned in FICPS is the same as the
*               truncated size of the file (FITS) if only the
*               cheap information has been requested.
*
*********************************************************************
*
*        Sharing Information DSECT - Any access is sufficient
*             to obtain Sharing Information.
*
*********************************************************************
SIDSECT  DSECT
SIAL     DS    F           Array Length - Num of words requested
SIRL     DS    F           Return length - Num of words returned
SIACC    DS    F           Access allowed to this file for this
*                          USERID-PRJNO-PKEY:
*                             1 = read access
*                             2 = write extend access
*                             4 = write change/empty access
*                             8 = renumber/truncate access
*                             16= destroy/rename access
*                             32= permit access
*                             Add for multiple access
SIGA     DS    F           Global (others) access - see above
SIOA     DS    F           Owner access - see above
*                            Minus one (-1) unless the caller
*                            has permit access to the file
SIPTR    DS    F           Pointer to variable len sharing list
*                          or zero if no variable sharing list
SILEN    EQU   *-SIDSECT
*
*        Permit access is required to obtain complete access
*        information, otherwise just the access that could
*        apply to the current USERID/PRJNO is returned.
*
*        Variable-length sharing list is formatted as follows:
*
*        1 Word            Total length (including this) - words
*
*        1 Word            USERID/PRJNO/PKEY access - see above
*
*        1 Word            USERID/PRJNO/PKEY flag
*                             0=PRJNO,1=USERID,2=PKEY
*                             3=PRJNO&PKEY,4=USERID&PKEY
```

```
*
*        1 Word            USERID/PRJNO length: 1-4
*          or
*        1 Word            PKEY length: 1-13
*
*          followed by
*
*        4 Characters      USERID/PRJNO - EBCDIC, left-justified
*          or
*        16 Characters     PKEY - EBCDIC, left-justified
*
*        Thus, for each sharer (USERID/PRJNO/PKEY) permitted
*        access to the file, you get
*           a) 4 words (if USERID/PRJNO)
*        or b) 7 words (if PKEY).
*
*        Note that for codes 3 (PRJNO&PKEY) and 4 (USERID&PKEY),
*        you really get 4 words (USERID/PRJNO) followed by
*        7 words (PKEY).
*
*        The access and flag words will be repeated and
*        identical for codes 3 and 4.
*
*        Partially specified USERIDS, PROJNOs, and PKEYs
*        will be returned with a trailing question mark,
*        which in turn may be followed with trailing blanks.
*
*        Fully specified PROJNOs and PKEYs may be padded
*        with trailing blanks, which are included in the
*        length returned.  Fully specified public (*) PKEYs
*        are 12 characters long.  Fully specified private
*        PKEYs are
*           a) 13 characters long if the caller's CCID and the
*              CCID prefix of the PKEY are different
*        or b) 8 characters long if the CCIDs are the same.
*
*******************************************************************
```

Examples: Assembly:

```
CATALOG CSECT
        ENTER 12
        CALL  GUSERID     Get signon ID
        ST    1,WHAT      Store ID in par list
        XC    RTN(24),RTN Zero return region
AGAIN   CALL  GFINFO,(WHAT,RTN,FLAG,CINFO,FINFO,
              SINFO,ERCODE,ERRMSG),VL
        LTR   15,15       Test return code
        BNZ   ERROR       Error exit
        SPRINT RTN,20     Print file name
        B     AGAIN
ERROR   L     2,ERCODE   Check error number
        C     2,=F'5'    No more files?
        BNE   REALERR    Real error
        EXIT  0          Normal exit
REALERR SERCOM ERRMSG,80  Print error message
        CALL  ERROR
WHAT    DS    F          ID of catalog to scan
RTN     DS    6F         Return file name
FLAG    DC    F'3'       Scan catalog flag
CINFO   DC    F'0'       No catalog info wanted
FINFO   DC    F'0'       No file info wanted
SINFO   DC    F'0'       No sharing info wanted
ERCODE  DS    F          Return error number
ERRMSG  DS    CL80       Return error message
        END
```

The above program calls GFINFO to obtain all of the file names in the signon ID's catalog.

FORTRAN:

```
     IMPLICIT INTEGER*(A-Z)
     DIMENSION RTN(6),ERRMSG(20)
     DATA RTN/6*0/
     COMMON /FI/ FIAL,FIRL,FIFO,FIFLG,FICNS,FITS
     COMMON /FI/ FICPS,FIFLN,FILLN,FIMLL,FINE
     COMMON /FI/ FINL,FINH,FILCNT,FIHCNT,FIMHL
     COMMON /FI/ FIXF,FIMBC
     FIAL = 18
     CALL GFINFO('DATAFILE ',RTN,1,0,FIAL,0,
    *            ERCODE,ERRMSG,&10)
     IF(FIRL.EQ.0) GO TO 10
     WRITE(6,101) FICNS
     WRITE(6,102) FITS
     WRITE(6,103) FICPS
     CALL SYSTEM
10   CALL ERROR
101  FORMAT(' CURRENT SIZE IN PAGES=',I5)
102  FORMAT(' SIZE IN PAGES IF TRUNCATED=',I5)
103  FORMAT(' SIZE IN PAGES IF COPIED=',I5)
     END
```

The above program will print the current, truncated, and copied file size in pages for the file DATAFILE.

GPRJNO

Subroutine Description

Purpose: To obtain the current 4-character project ID.

Location: Resident System

| Alt. Entries: GPRJNOS, GPRJNS

Calling Sequences:

Assembly: CALL GPRJNO

CALL GPRJNOS,(region),VL

FORTRAN: CALL GPRJNS(region,&rc4)

Parameters:

region is the 4-byte region in which to store the projectID.

&rc4 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 VL bit not specified.

A GR13 save area is not required for a call to this subroutine.

Values Returned:

GR1 contains the 4-character project ID.

Description: A call on the GPRJNOS subroutine takes the S-type parameters and loads them into an R-type call on the GPRJNO subroutine.

FORTRAN: CALL GPRJNS(ID)

The above example returns the project ID in the region labelled ID.

GPSECT, QPSECT, FPSECT

Subroutine Description

Purpose: To acquire, query, and release psect (dsect) storage allocations.

Location: Resident System

| Alt. Entries: GPSECTS, GPSCTS, QPSECTS, QPSCTS, FPSECTS, FPSCTS

Calling Sequences:

Assembly:
```
L     0,id
L     1,len
CALL  GPSECT

L     0,id
CALL  QPSECT

L     0,id
CALL  FPSECT

CALL GPSECTS,(id,len,addr),VL

CALL QPSECTS,(id,addr),VL

CALL FPSECTS,(id),VL
```

FORTRAN:
```
CALL GPSCTS(id,len,addr,&rc4,&rc8,&rc12,&rc16)

CALL QPSCTS(id,addr,&rc4,&rc8,&rc12,&rc16)

CALL FPSCTS(id,&rc4,&rc8,&rc12,&rc16)
```

Parameters:

<u>id</u> (GR0) is an unique fullword identifier for the psect (i.e., a fixed address within the calling program could be used as such an identifier).

<u>len</u> (GR1) is the length to be allocated, in bytes.

<u>addr</u> is the address of the psect returned.

A GR13 save area is not required for a call to the GPSECT, QPSECT, or FPSECT subroutines.

Values Returned:

GR1 (GPSECT only) contains the address of the psect allocated.

GR1 (QPSECT only) contains the address of the psect if found, otherwise zero.

Return Codes:

GPSECT:
- 0 Psect found.
- 4 Psect not found but allocated.
- 8 Error return from GETSPACE subroutine.
- 12 Internal error in GPSECT.
- 16 Invalid parameter(s) specified.

QPSECT:
- 0 Psect found.
- 4 Psect not found.
- 8 Not used.
- 12 Internal error in QPSECT.
- 16 Invalid parameter(s) specified.

FPSECT:
- 0 Psect released.
- 4 Psect not found.
- 8 Error return from FREESPAC subroutine.
- 12 Internal error in FPSECT.
- 16 Invalid parameter(s) specified.

Description: The GPSECT, QPSECT, and FPSECT subroutines are used to acquire, query, and release storage to be used for psects (dsects) in the calling program. An identifier for the psect and the length of the psect are specified in _id_ and _len_.

The GPSECT subroutine is used to allocate storage for the psect. If a psect with the identifier _id_ already exists, its address is returned and a new psect is not allocated.

The QPSECT subroutine is used to query the existence of a psect with the identifier _id_. A new psect is not allocated.

The FPSECT subroutine is used to release the storage for the psect with identifier _id_.

A call on the GPSECTS, QPSECTS, and FPSECTS subroutines takes the S-type parameters and loads them into an R-type call on the corresponding GPSECT, QPSECT, and FPSECT subroutines.

Example: Assembly:

```
                 L     0,ID
                 L     1,LEN
                 CALL  GPSECT
                   .
                   .
                 L     0,ID
                 CALL  FPSECT
                   .
                   .
           ID    DC    A(ID)
           LEN   DC    F'4096'
```

The example allocates a psect of 4096 bytes with the identifier which is an address contained within the calling program (e.g., the address of ID). The psect is then released later in the program.

GRAND, GRAND1

Subroutine Description

Purpose: To compute normally distributed random numbers with a given mean and standard deviation.

Location: *LIBRARY

Calling Sequences:

Assembly:
```
CALL GRAND1,(value)
CALL GRAND,(sd,amean)
```

FORTRAN:
```
CALL GRAND1(value)
x = GRAND(sd,amean)
```

Parameters:

- value is the location of a fullword integer used for generating the random number.
- sd is the location of the fullword real (REAL*4) standard deviation.
- amean is the location of the fullword real (REAL*4) mean.

Values Returned:

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

Description: The function subroutine GRAND computes twelve uniformly distributed random numbers by the power-residue method and, based on the central limit theorem, uses these to compute a normally distributed random number x with mean amean and standard deviation sd. Note that the result is returned as a function value, not as a parameter.

If, before the first call to GRAND, the user wishes to specify the initial integer value from which the uniformly distributed random numbers are generated, he may do so by calling GRAND1 with value set equal to an odd integer between 1 and $2^{31}-1$ (2147483647). If GRAND1 is not called, GRAND will supply its own initial value (524287).

If the user wishes to obtain a sequence of random numbers, GRAND1 should be called initially followed by repeated calls to GRAND.

If the same sequence of random numbers is required on successive runs, the user must supply the same initial value of value to GRAND1.

Examples: Assembly:

```
               CALL GRAND1,(INTEG)
               CALL GRAND,(STDEV,MEAN)
               STE  0,RAND
                .
                .
         INTEG DC   F'999'
         STDEV DC   E'10.0'
         MEAN  DC   E'100.0'
         RAND  DS   E
```

FORTRAN:

```
INTEG=999
CALL GRAND1(INTEG)
X=GRAND(10.0,100.0)
```

In both examples above, GRAND is called with an initial value of 999, a standard deviation of 10.0, and a mean of 100.0.

GRGJULDT, GRGJULTM, GRJLSEC

Subroutine Description

Purpose: To convert the Gregorian date (MM/DD/YY) or time (MM/DD/YYhh:mm:ss) to the corresponding Julian date or time (based on March 1, 1900).

Location: Resident System

| Alt. Entries: GRJLSECS, GRJLSS

Calling Sequences:

```
Assembly: LM   0,1,grgdat
          CALL GRGJULDT

          LM   0,1,grgdat
          LM   2,3,grgtim
          CALL GRGJULTM

          LM   0,1,grgdat
          LM   2,3,grgtim
          CALL GRJLSEC

          CALL GRJLSECS,(grgdat,grgtim),VL

FORTRAN:  CALL GRJLSS(grgdat,grgtim,&rc4)
```

Parameters:

grgdat is the Gregorian date in the character form "MMxDDxYY", where "x" is any character.
grgtim is the Gregorian time in the character form "hhxmmxss", where "x" is any character.

Return Codes:

0 Successful return. Julian time is in grgtim.
4 Illegal parameter or no VL bit specified.

Values Returned:

GR0 contains the integer number of days through the given date starting with March 1, 1900 as "1".

GR1 contains the integer number of minutes through the given time starting with March 1, 1900, at 00:01 as "1" for GRGJULDT and GRGJULTM. For GRGJULDT, the time is assumed to be 00:00:00. GR1 contains the number of seconds through the given time starting with March 1, 1900, at 00:00:01 as "1" for GRJLSEC.

Description: The range of years is assumed to be 1900-1999. If the number of seconds passed to GRGJULTM is greater than or equal to 30, the result in GR1 is rounded up to the next minute. If the time is greater than 03/19/68 03:14:07 for GRJLSEC, the result requires 32 bits. The results for dates prior to 03/01/00 are undefined.

| A call on the GRJLSECS or GRJLSS subroutines takes the
| S-type parameters and loads them into an R-type call on
| the GRJLSEC subroutine.

Examples: Assembly:

```
              LM   0,1,=C'05/18/71'
              CALL GRGJULDT
              ST   0,DATE
                .
                .
        DATE  DS   F
```

The above example calls GRGJULDT to convert the Gregorian date May 18, 1971 into its corresponding Julian date 26011.

```
              LM   0,3,=C'05-06-7116:30:17'
              CALL GRGJULTM
              ST   0,DATE
              ST   1,TIME
                .
                .
        DATE  DS   F
        TIME  DS   F
```

The above example calls GRGJULTM to convert the Gregorian date and time May 6, 1971, 16:30:17 into its corresponding Julian date and time 25999 and 37438110, respectively.

GRJLDT, GRJLTM

Subroutine Description

Purpose: S-type (e.g., FORTRAN and PL/I) interfaces for GRGJULDT and GRGJULTM.

Location: *LIBRARY

Calling Sequences:

FORTRAN:
```
INTEGER*4 GRJLDT
juldat=GRJLDT(grgdat)

INTEGER*4 GRJLTM
jultim=GRJLTM(grgtim)
```

PL/I(F):
```
DCL PLCALLF RETURNS(FIXED BINARY(31));
juldat=PLCALLF(GRJLDT,f1,grgdat);

DCL PLCALLF RETURNS(FIXED BINARY(31));
jultim=PLCALLF(GRJLTM,f1,grgtim);
```

Parameters:

- grgdat is the 8-byte (REAL*8 or CHARACTER(8)) Gregorian date in the character form "MMxDDxYY", where "x" is any character.
- grgtim is the 16-byte (REAL*8(2) or CHARACTER(16)) Gregorian date and time in the character form "MMxDDxYYhhxmmxss", where "x" is any character.
- f1 is a fullword (FIXED BINARY(31)) containing the integer 1.

Values Returned:

juldat contains the integer number of days through the given date starting with March 1, 1900, as "1" for calls on GRJLDT.

jultim contains the integer number of minutes through the given time starting with March 1, 1900, at 00:01 as "1" for calls on GRJLTM.

Return Codes:

0 Successful return.
4 At least one digit position in the date or time does not contain a digit. Upon return, GR0 is set to zero.

Description: The Gregorian date or time in character form is passed to GRGJULDT or GRGJULTM, respectively, and is converted to the corresponding Julian date or time. The range of years is assumed to be 1900-1999. If the number of seconds passed to GRJLTM is greater than or equal to 30, the time is rounded up to the next minute. The results for dates prior to 03/01/00 are undefined.

Examples: FORTRAN:

```
      INTEGER*4 GRJLDT
      REAL*8 DATE
      JULIAN=GRJLDT(DATE)
      IF (JULIAN.EQ.0) GO TO 400
```

The above example calls GRJLDT to convert the Gregorian date in the variable DATE into its corresponding Julian date.

```
      INTEGER*4 GRJLTM
      REAL*8 TIME(2)
      JULIAN=GRJLTM(TIME)
      IF (JULIAN.EQ.0) GO TO 400
```

The above example calls GRJLTM to convert the Gregorian date and time in the array TIME into its corresponding Julian date and time.

PL/I(F):

```
JULIAN=PLCALLF(GRJLDT,F1,DATE);
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE JULIAN FIXED BINARY(31),
        PLCALLF RETURNS(FIXED BINARY(31)),
        GRJLDT ENTRY,
        F1 FIXED BINARY(31) INITIAL(1),
        DATE CHARACTER(8) INITIAL('05-18-71');
PL1RC RETURNS (FIXED BINARY(31));
```

The above example calls GRJLDT to convert the Gregorian date May 18, 1971 into its corresponding Julian date 26011.

```
JULIAN=PLCALLF(GRJLTM,F1,TIME);
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE JULIAN FIXED BINARY(31),
        PLCALLF RETURNS(FIXED BINARY(31)),
        GRJLTM ENTRY,
        F1 FIXED BINARY(31) INITIAL(1),
        TIME CHARACTER(16);
PL1RC RETURNS (FIXED BINARY(31));
```

The above example calls GRJLTM to convert the Gregorian date and time in the variable TIME into its corresponding Julian date and time.

GROSDT

Subroutine Description

Purpose: To convert the Gregorian date (MM/DD/YY) to the corresponding OS date (YYddd).

Location: *LIBRARY

Calling Sequences:

Assembly: CALL GROSDT,(grgdat,osdat)

FORTRAN: CALL GROSDT(grgdat,osdat,&rc4)

```
REAL*8 GROSDT
date=GROSDT(grgdat,osdat)
```

PL/I(F): CALL PLCALL(GROSDT,f2,grgdat,osdat);

```
DCL PLCALLD RETURNS(FLOAT(16));
date=PLCALLD(GROSDT,f2,grgdat,osdat);
```

Parameters:

grgdat is the 8-byte (REAL*8 or CHARACTER(8)) Gregorian date in the character form "MMxDDxYY", where "x" is any character.

osdat is 8 bytes (REAL*8 or CHARACTER(8)) into which the OS date, in the character form "YYddd" with three leading blanks, is placed on return.

_f2_ is a fullword (FIXED BINARY(31)) containing the integer 2.

_rc4_ is a statement label to transfer to if a return code of 4 occurs.

Values Returned:

FR0 contains the OS date in the character form "YYddd" with three leading blanks. This is assigned to _date_ for FORTRAN and PL/I programs using the function-call format.

Return Codes:

0 Successful return.
4 At least one digit position in the date does not contain a digit. Upon return, FR0 and _osdat_ contain blanks.

Description: The range of years is assumed to include 1900. The result for dates prior to 03/01/00 is undefined.

Examples: Assembly:

```
                CALL GROSDT,(GRDAT,OSDAT)
                 .
                 .
          GRDAT DC   C'05-18-71'
          OSDAT DS   0D,CL8

                CALL GROSDT,(GRDAT,DUMMY)
                STD  0,OSDAT
                 .
                 .
          GRDAT DC   C'05-18-71'
          DUMMY DS   CL8
          OSDAT DS   CL8
```

The above two examples call GROSDT to convert the Gregorian date May 18, 1971 into the corresponding OS date 71138. The result is stored in location OSDAT.

FORTRAN:

```
      REAL*8 GRDAT,OSDAT
      CALL GROSDT(GRDAT,OSDAT,&400)

      REAL*8 OSDAT,GROSDT,GRDAT,DUMMY
      OSDAT=GROSDT(GRDAT,DUMMY)
```

The above two examples call GROSDT to convert the Gregorian date in the variable GRDAT into the corresponding OS date 71138. The result is stored in the variable OSDAT.

PL/I(F):

```
CALL PLCALL(GROSDT,F2,'05-18-71',OSDAT);
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE GROSDT ENTRY,
        OSDAT CHARACTER(8);
        F2 FIXED BINARY(31) INITIAL(2),
PL1RC RETURNS (FIXED BINARY(31));

UNSPEC(OSDAT)=UNSPEC(PLCALLD(GROSDT,F2,GRDAT,
        DUMMY));
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE OSDAT CHARACTER(8),
        GROSDT ENTRY,
        PLCALLD RETURNS(FLOAT(16)),
        F2 FIXED BINARY(31) INITIAL(2),
        GRDAT CHARACTER(8) INITIAL('05-18-71'),
        DUMMY CHARACTER(8);
PL1RC RETURNS (FIXED BINARY(31));
```

The above two examples call GROSDT to convert the Gregorian date May 18, 1971 into the corresponding OS date 71138. The result is stored in the variable OSDAT.

GTDJMS

Subroutine Description

Purpose: S-type (e.g., FORTRAN and PL/I) interface for GTDJMSR.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL GTDJMS(grgtim,jms)

PL/I(F): CALL PLCALL(GTDJMS,f2,grgtim,jms);

Parameters:

<u>grgtim</u> is the 16-byte (REAL*8(2) or CHARACTER(16)) Gregorian time and date in the character form "hhxmmxssMMxDDxYY", where "x" is any character.

<u>f2</u> is a fullword (FIXED BINARY(31)) containing the integer 2.

<u>jms</u> is an 8-byte integer (INTEGER*4(2) or BIT(64)) containing the integer number of microseconds through the given time and date starting with March 1, 1900.

Description: The Gregorian time and date in character form is passed to GTDJMSR and is converted to the corresponding Julian time. The range of years is assumed to be 1900-1999. The results for dates prior to March 1, 1900 are undefined.

Examples: FORTRAN:

```
      INTEGER*4 JULIAN(2)
      REAL*8 TIME(2)
      DATA TIME/'17:59.33','03-21-73'/
         .
         .
      CALL GTDJMS(TIME,JULIAN)
```

PL/I(F):

```
DECLARE JULIAN BIT(64),
        GTDJMS ENTRY,
        F2 FIXED BINARY(31) INITIAL(2),
        TIME CHARACTER(16)
             INITIAL('17:59.3303-21-73');
CALL PLCALL(GTDJMS,F2,TIME,JULIAN);
```

The above two examples call GTDJMS to convert the Gregorian time and date 17:59.33 March 21, 1973 into the corresponding Julian time 000830D174704C60 (hex).

GTDJMSR

Subroutine Description

Purpose: To convert the Gregorian time and date (MM-DD-YY,hh:mm.ss) into Julian microseconds (number of microseconds since March 1, 1900).

Location: *LIBRARY

Calling Sequences:

Assembly:
```
LM   0,3,grgtim
CALL GTDJMSR
```

Parameter:

grgtim is the Gregorian time and date in the character form "hhxmmxssMMxDDxYY", where "x" is any character.

Value Returned:

GR0 and GR1 contain the (8-byte) integer number of microseconds through the given time starting with March 1, 1900.

Description: The range of years is assumed to be 1900-1999. The results for dates prior to March 1, 1900 are undefined.

See GTDJMS for S-type (e.g., FORTRAN and PL/I) interfaces.

Example: Assembly:
```
            LM   0,3,GRGDT
            CALL GTDJMSR
            STM  0,1,JMS
             .
             .
      GRGDT DC   C'17:59.3303-21-73'
      JMS   DS   2F
```

The above example calls GTDJMSR to convert the Gregorian time and date 17:59.33 March 21, 1973 into the corresponding Julian time 000830D174704C60 (hex).

GUINFO, CUINFO

Subroutine Description

Purpose: To allow the user to obtain information items about the status of the task (GUINFO) and to change some of the information items (CUINFO).

Location: Resident System

Calling Sequences:

Assembly:
```
CALL GUINFO,(item,loc)

CALL CUINFO,(item,loc)
```

FORTRAN:
```
CALL GUINFO(item,loc,&rc4,&rc8,&rc12,&rc16)

CALL CUINFO(item,loc,&rc4,&rc8,&rc12,&rc16)
```

Parameters:

item is the location of either
(a) a fullword integer index number, or
(b) an 8-character name of the item left-justified with trailing blanks.
This specifies what item is to be obtained or changed.

loc is the location of the region in which to place the information obtained (for GUINFO) or to obtain the replacement information from (for CUINFO). The size of the region depends upon the type of the item.

_rc4,...,rc16_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Error return. Item number too large or not in use.
8 Error return. Item name not in the list.
12 Error return. Illegal to change item (CUINFO only).
16 Error return. Illegal parameter address, or illegal length for variable length item.

Description: The names given in the tables below correspond to items of information from the system.

There are three tables given; they are organized by item number, item name, and item subject. The table of item subjects has the following categories:

Accounting - Batch Input and Output
Accounting - CPU, Memory, and Paging
Accounting - File System Storage
Accounting - Magnetic Tapes, Paper Tapes, and Floppy Disks
Accounting - Money
Accounting - Plotter Use
Accounting - Terminal and Merit Computer Network Use
Accounting - User ID and Project Information
Batch Mode Jobs
Command Language Options
Execution Processing
Interrupt Processing
I/O File and Device Names
System Information
Task Limits
Task Status
Terminal Information

All of the items can be obtained by GUINFO, but only a subset of these items can be changed by CUINFO (those marked with an "*" after the index number in the following tables). Each item may be referred to by its name or by its index number.

The size of the region required to contain the item and the interpretation of the returned value are both given in the following tables. The region size is independent of whether the item is being set (by CUINFO) or retrieved (by GUINFO) with the exception of the items of variable size.

For variable length items the <u>loc</u> parameter consists of two fullwords followed by a region in which the information to be returned will be placed (for GUINFO) or from which the new information will be obtained (for CUINFO). The first fullword must be set to the length (in bytes) of the <u>loc</u> region, including both fullwords. Upon return from calls to GUINFO, the second fullword will be set to the length (in bytes) of the information returned. If the information to be returned will not fit into the region provided (as indicated by the length supplied in the first fullword of the region), the information is truncated on the right, but the length returned in the second fullword gives the length before truncation. On calls to CUINFO the second fullword must be set to the length (in bytes) of the new information that follows the leading fullwords.

All cumulative fields are cumulative up to the time of the last call to GUINFUPD or later, but do not include the current job or any other active instances of this ID. CUMCELL and CUMDISK, however, are cumulative up to CELLTIME and DISKTIME, respectively.

Examples: Assembly:

```
                  CALL GUINFO,(GITEM,GLOC)
                  CLC  GLOC,=F'0'
                  BNE  BATCH
                  CALL CUINFO,(CITEM,CLOC)
                   .
                   .
            GITEM DC   CL8'BATCHMD '
            GLOC  DS   F
            CITEM DC   CL8'PFXSTR  '
            CLOC  DC   A(CLEN)        Region length
                  DC   F'2'           Prefix length
                  DC   C'-?'          New prefix
            CLEN  EQU  *-CLOC
                  END
```

FORTRAN:

```
      INTEGER*4 GLOC,CLOC(3)
      DATA CLOC/12,2,'-?  '/
      CALL GUINFO('BATCHMD ',GLOC)
      IF(GLOC.EQ.1) GO TO 10
      CALL CUINFO('PFXSTR  ',CLOC)
   10 CONTINUE
        ...
```

The above two examples call GUINFO to determine whether the job is running in batch or conversational mode. If the job is conversational, the prefix character is set to the two-character string "-?" by calling CUINFO.

Table of Items Arranged by Index

Index	Name	Size	Description
1*	LNS	4 Bytes	Line-number separator character, left-justified with trailing blanks (default is ","; $SET LNS=c)
2	SIGNONID	4 Bytes	Current signon ID
3*	PREFIXC	Fullword	Current prefix character, left-justified with trailing blanks, as set by the SETPFX subroutine or CUINFO item 257 (PFXSTR).
4	S8NBR	8 Bytes	Receipt number of job in characters, left-justified with trailing blanks (batch only)
5*	FILECHAR	4 Bytes	File-name character, left-justified with trailing blanks (default is "#"; $SET FILECHAR=c)
6	STORUSED	Fullword	CPU storage integral to STORCPUT[1]. See Note (1).
7*	SCRFCHAR	4 Bytes	Scratch-file character, left-justified with trailing blanks (default is "-"; $SET SCRFCHAR=c)
8	CURRSTOR	Fullword	Current number of half-pages of VM storage. See Note (1).
9*	CONTCHAR	4 Bytes	MTS command continuation character, left-justified with trailing blanks (default is "-"; $SET CONTCHAR=c)
10	BATCHMD	Fullword	Batch (1) or conversational (0) mode
11*	ICFBIT	Fullword	1 -> $SET IC=OFF (default is ON)
12	LOCSW	Fullword	1 -> Local time estimate active
13	SIGTMUT	18 Bytes	Signon time (Universal Time Units). See Note (4).
14	ACCTNO	Fullword	User account requisition number
15*	ATNBIT	Fullword	1 -> Attention interrupt occurred but not taken (may be set to cause an attention interrupt)
16	PROJNO	4 Bytes	Project (charge) ID in characters
17*	UCBIT	Fullword	1 -> $SET CASE=UC (default is MC)
18	MAXDISK	Fullword	Maximum number of disk pages allowed for ID
19*	NXTSEGSW	Fullword	1 -> Skip to next set of MTS command cards (batch only; may be set to skip unread data cards)
20	MAXTERM	Fullword	Maximum terminal time allowed for ID (seconds)
21*	PRNTCDSW	Fullword	1 -> Print next input line from source if not MTS command (batch only)
22	MAXMONY	Fullword	Maximum charge allowed for ID (cents*100)
23*	OFFBIT	Fullword	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
24	CURRDISK	Fullword	Number of pages of disk space in current use. See Note (2).
25	PLOTTIME	Fullword	Total plot time for current job (seconds)
26	CUMELTM	Fullword	Cum. terminal time for ID (seconds) (excluding active jobs)
27*	DUMPTYPE	Fullword	$SET ERRORDUMP={NOLIB|OFF|LIB} (0|1|2) (default NOLIB)
28	CUMCPUTM	Fullword	Cum. CPU time for ID (milliseconds) (excluding

			active jobs)
29	CUMREAD	Fullword	Cum. number of cards read for ID (excluding active jobs)
30	CUMCORE	Fullword	Cum. storage integral over CPU time for ID (excluding active jobs)[2]
31	NRREAD	Fullword	Number of cards read for current job
32	CUMMONY	Fullword	Cum. charge used for ID (cents*100) (excluding active jobs)
33*	LDROPT	4 Bytes	Loader options switches in leftmost byte[10]
35*	SHFSEP	4 Bytes	Shared-file separator character, left-justified with trailing blanks (default is ":"; $SET SHFSEP=c)
36	NRDISKF	Fullword	Number of disk files existing for ID
37*	RF	Fullword	Relocation factor for ALTER/DISPLAY/MODIFY commands (Default is 0; $SET RF=xxxxxx)
38	NRSIGS	Fullword	Cum. number of signons for ID (excluding active jobs)
39	DEVCHAR	4 Bytes	Device-name character, left-justified with trailing blanks (default is ">"; $SET DEVCHAR=c)
40	NRBATCH	Fullword	Cum. number of batch jobs for ID (excluding active jobs)
41*	NUMBER	Fullword	1 -> Automatic numbering active ($NUMBER)
42	CUMLINES	Fullword	Cum. number of lines printed for ID (excluding active jobs)
43*	LIBROFF	Fullword	1 -> $SET LIBR=OFF (default is ON)
44	CUMPAGES	Fullword	Cum. number of pages printed for ID (excluding active jobs)
45*	AFDECHO	Fullword	1 -> $SET AFDECHO=ON (default is OFF)
46	CUMPUNCH	Fullword	Cum. number of cards punched for ID (excluding active jobs)
47*	SYMTAB	Fullword	1 -> $SET SYMTAB=ON (default is ON)
48	STORUSEE	Fullword	Elapsed storage integral to STORELT[1]. See Note (1).
49*	ECHOOFF	Fullword	1 -> $SET ECHO=OFF (default is ON)
50	IDRNBR	Fullword	User inter-departmental requisition number
51*	ATTNOFF	Fullword	1 -> Stack attention interrupts (may be set to inhibit attention interrupts; pending interrupt may be taken on call to system subroutine)
52	UNITCODE	Fullword	User unit code
54	EXPTIME	Fullword	ID expiration time and date[3]
55*	SIGSHORT	Fullword	$SIGNOFF {LONG\|SHORT\|$} (0|1|2) (default is LONG)
56	SOBCDTM	16 Bytes	Signon time and date in characters
57*	PFXOFF	Fullword	1 -> $SET PFX=OFF (default is ON)
58	STORCPUT	Fullword	Current base for CPU storage integral[4]. See Note (1).
59*	SEQCOFF	Fullword	1 -> $SET SEQFCHK=OFF (default is ON)
60	NRCREATE	Fullword	Number of files created during current job
61*	PGNTTRP	2 Words	PGNTTRP exit subroutine address (1st word) and save area location (2nd word)
62	NRDESTRY	Fullword	Number of files destroyed during current job
63	NRLINES	Fullword	Number of lines printed for current job

64	SOCPUTP	Fullword	Problem state CPU time used by task before current signon[5]
65	NRPAGES	Fullword	Number of pages printed for current job
66	SOCPUTC	Fullword	Supervisor state CPU time used by task before current signon[5]
67	NRPUNCH	Fullword	Number of cards punched for current job
68	SOELT	Dblword	Time of day at signon[6]
69*	ATTNTRP	2 Words	ATTNTRP exit subroutine address (1st word) and save area location (2nd word)
70	STORELT	Fullword	Current base for elapsed storage integral[4]. See Note (1).
71*	AFDNBR	Fullword	Next line number for *AFD* ($NUMBER)
72	SOPTOD	16 Bytes	Time and date for header page for batch output (characters)
73*	AFDINC	Fullword	Line-number increment for *AFD* ($NUMBER)
74	ANSBACK	24 Bytes	Answerback code (characters) (see also item 276)
75*	SETIOERR	Fullword	SETIOERR exit subroutine address
76	CUMDISK	Fullword	Cum. disk file storage integral to DISKTIME which has been charged for (page hours). See Note (2).
77*	ENDFILSW	Fullword	$SET ENDFILE={NEVER|SOURCE|ALWAYS} (0|1|2) (default SOURCE)
78	GLOBCPUT	Fullword	CPU time remaining in global time limit[5]. See Note (3).
79	NRMOUNT	Fullword	Number of tape and other mounts for current job
80	GLOBPGS	Fullword	Global page estimate
81	TDRVT	Fullword	Tape drive time for current job (seconds)
82	GLOBPCH	Fullword	Global card estimate
83	PTLEN	Fullword	Paper tape punched for current job (inches)
84	GLOBPTM	Fullword	Global plot time estimate (seconds)
85*	TDR	Fullword	1 -> $SET TDR=ON (default is OFF)
86	LOCCPUT	Fullword	CPU time remaining in local time limit[5]. See Note (3).
87	MNETTIME	Fullword	Outbound Merit time for this job (seconds)
88	LOCPGS	Fullword	Local page estimate
89*	CROUTE	4 Bytes	Default batch station for punched output (characters) ($SET CROUTE=rmid)
90	LOCPCH	Fullword	Local card estimate
91*	PROUTE	4 Bytes	Default batch station for printed output (characters) ($SET PROUTE=rmid)
92	LOCPTM	Fullword	Local plot time estimate (seconds)
93*	PRINT	4 Bytes	Print train specification ("PN ", "TN ", "UC ", "MC ", or binary 0 in first byte if ANY)
94	GLOBTTN	Fullword	Base for global time limit[5]. See Note (3).
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
96	LOCTTN	Fullword	Base for local time limit[5]. See Note (3).
98	TASKNBR	Fullword	Task number
99*	SEE_DISP	Fullword	1 -> $SET DISPATCH=ON (default ON)
100	TASKTYPE	Fullword	Task type code[8]
104	HASPJOB	Fullword	1 -> Spooled batch job

106	MAXCELL	Fullword	Maximum datacell pages allowed for ID
107	SODISKIO	Fullword	Number of disk operations at signon for task
108	MAXPLOT	Fullword	Maximum plot time allowed for ID (seconds)
109	CUDISKIO	Fullword	Total number of disk operations for task
110	LSTRESET	Fullword	Last time cum. totals for this ID were reset[3]
111	ASYNCCTL	Fullword	Asynchronous event control switch[13]
112	DISKTIME	Fullword	Last time disk storage integral updated[3]
113*	SVCTRP	2 Words	SVCTRP exit subroutine address (1st word) and save area location (2nd word)
114	CELLTIME	Fullword	Last time datacell storage integral updated[3]. See Note (2).
115	RUNETIME	Dblword	Cumulative real time for program[5]
116	CURRCELL	Fullword	Number of pages of datacell files in current use. See Note (2).
118	CUMCOREW	Fullword	Cum. storage integral over wait time for this ID (excluding active jobs)[2]
119*	EBM	8 Bytes	The "execution begins" message--up to 7 characters, terminated with an *
120*	ETM	8 Bytes	The "execution terminated" message--up to 7 characters, terminated with an *
121*	EXECPFX	4 Bytes	Execution prefix character ($SET EXECPFX=c) (left-justified)
122	CUMPLOT	Fullword	Cum. plot time for ID (seconds) (excluding active jobs)
124	NRCELLF	Fullword	Number of datacell files existing for ID
125*	PAPER	12 Bytes	$SET PAPER={PLAIN|3HOLE|ANY} (characters) (default 0 (ANY))
126	CUMCELL	Fullword	Cum. datacell file storage integral to CELLTIME which has been charged for (page hours). See Note (2).
127*	PRINTER	4 Bytes	$SET PRINTER={LINE|PAGE|ANY} (characters) (default ANY)
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
129*	DELIVERY	8 Bytes	$SET DELIVERY={station|NONE} (characters) (default NONE)
130	LINKLEVL	Fullword	Current link level (see MTS Vol. 5 Virtual Memory Management description)
134	STORINDX	Fullword	Current storage index number (See MTS Vol. 5 Virtual Memory Management description)
136	MXSTRIND	Fullword	Maximum storage index number used (See MTS Vol. 5 Virtual Memory Management description)
138	LODRSYMT	Fullword	Loader symbol table location
146	SCRFDISK	Fullword	Number of pages of disk scratch files for current job. See Note (2).
148	SCRFCELL	Fullword	Number of pages of datacell scratch files for current job. See Note (2).
149	LSIGTMUT	18 Bytes	Last signon time (Universal Time Units). See Note (4).
150	SODRMRDS	Fullword	Number of page-ins by task before signon
151*	SIGOFRCT	Fullword	1 -> Display receipt summary at signoff
152	LASTSOT	16 Bytes	Last signon time in characters
154	CUMMOUNT	Fullword	Cum. number of tape mounts for ID (excluding

			active jobs)
156	CUMTDRVT	Fullword	Cum. tape drive time for ID (seconds) (excluding active jobs)
157	CUMPTSU	Fullword	Cum. phototypesetter units
159	CUMPTSM	Fullword	Cum. phototypesetter media (cm^2)
158	CUMPTLEN	Fullword	Cum. paper tape punched for ID (inches) (excluding active jobs)
162	SCRDSKTM	Fullword	Last time scratch disk file storage integral updated[3]. See Note (2).
164	SCRCELTM	Fullword	Last time scratch datacell file storage integral updated[3]. See Note (2).
166	SCRDSUSE	Fullword	Scratch disk file storage integral to SCRDSKTM[7]. See Note (2).
167*	SIGFATTN	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
168	SCRCLUSE	Fullword	Scratch datacell file storage integral to SCRCELTM[7]. See Note (2).
169*	TERSE	Fullword	1 -> $SET TERSE=ON (default is OFF)
170	CUDRMRDS	Fullword	Current number of page-ins for current job
171*	$ON	Fullword	1 -> $SET $=ON (default is OFF)
172	CLSID	Fullword	Code for CLS currently in control[9]
174	PCLSID	Fullword	Code for CLS that called current CLS[9]
175*	EDITAFD	Fullword	1 -> $SET EDITAFD=ON (default is OFF)
176	DEBUGCMD	Fullword	1 -> If $DEBUG command active
177*	USMSG	Fullword	1 -> $SET USMSG=ON (default is ON)
178*	DEBUG	Fullword	1 -> $SET DEBUG=ON (default is OFF)
179*	AUTOHOLD	Fullword	1 -> $SET AUTOHOLD=ON (default is OFF)
180	LSS	Fullword	1 -> If limited-service state active
181*	TRIMBIT	Fullword	1 -> $SET TRIM=ON (default is ON)
182	MAXSIG	Fullword	Max. number of concurrent signons allowed for ID (0=unlimited)
183*	EFLUEM	Fullword	Elementary Function Library, user error-monitor address
184	CURSIG	Fullword	Number of times this ID currently signed on
185*	CMDSKP	Fullword	1 -> $SET CMDSKP=OFF (default is OFF)
186	UNCHDISK	Fullword	Disk space to DISKTIME not yet charged for[7]
187*	PRMAPOFF	Fullword	1 -> $SET PRMAP=OFF (default is OFF)
188	UNCHCELL	Fullword	Datacell space to CELLTIME not yet charged for[7]
189*	PDMAPOFF	Fullword	1 -> $SET PDMAP=OFF (default is OFF)
190	MAXMNET	Fullword	Maximum outbound Merit time (seconds)
191*	UXREF	Fullword	1 -> $SET UXREF=ON (default is OFF)
192	CUMMNET	Fullword	Cum. outbound Merit for this ID excluding active jobs (seconds)
193*	XREF	Fullword	1 -> $SET XREF=ON (default is OFF)
194	MXMNETBT	Fullword	1 -> Ignore maximum MNET time (item 190)
195*	NO*LIB	Fullword	1 -> $SET *LIBRARY=OFF (default is ON)
196	MXPLOTBT	Fullword	1 -> Ignore maximum plot time (item 108)
197*	MAPDOTS	Fullword	1 -> $SET MAPDOTS=ON (default is ON)
199*	NOERRMAP	Fullword	1 -> $SET ERRMAP=OFF (default is ON)
226	INSIGFIL	Fullword	1 -> Currently processing sigfile
227	PLOTPAPR	Fullword	Plotter paper used for current job (.01 inches)
228	TOFFSET	Dblword	Offset (microseconds times 4096) to be added to GMT to get local time
229	PLOTPENC	Fullword	Plotter pen changes for current job

230	TIMEFDGE	Dblword	Value (microseconds times 4096) to be added to IBM time (as stored by a STCK instruction) to get time based on March 1, 1900
231*	SPELLCOR	Fullword	$SET SPELLCOR={OFF|PROMPT|ON} (0|3|1) (default is PROMPT)
232	CUMPLPAP	Fullword	Cum. plotter paper used for ID (.01 inches) (excluding active jobs)
234	CUMPLPEN	Fullword	Cum. plot pen changes for ID (excluding active jobs)
236	PKEY	16 Bytes	Program key under which calling program is running
237*	RCPRINT	Fullword	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS|NONZERO} (0|1|2|3|4)
238	RUNONLY	Fullword	1 -> A "run only" program is loaded (from a file to which the user has only RUN access)
239	LASTEXRC	Fullword	Return code of last program executed
240	SYSOLOAD	Fullword	System overload indicators, right-justified with leading zeros[11]
241	SIGCFLD	Variable	The comment field from the MTS $SIGNON command, without the enclosing primes (from 0 to 255 characters in length)
242	PRIO	Fullword	Priority of job[12]
247*	SFATTN	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
249	PSFATTN	Fullword	1 -> Project sigfile attention bit is off
251*	CMDSCNBT	Fullword	1 -> $SET CMDSCAN=UNAMBIGUOUS (default is UNAMBIGUOUS)
252	UNATMODE	Fullword	1 -> System running in "unattended mode" (see also item 277)
253	LOCLIMIT	Fullword	Local time limit in effect[5]
254	RUNTIME	Fullword	Amount of time used during execution of current program[5]. This total is updated only when execution mode is exited, e.g., if program calls MTS
255	PARSTRMC	Variable	The PAR string from the MTS $RUN command in mixed-case (from 0 to 255 characters)
257*	PFXSTR	Variable	Prefix string which normally appears at the beginning of terminal input and output lines (from 0 to 120 characters in length)
258	PARSTR	Variable	The PAR string from the MTS $RUN command converted to uppercase (from 0 to 255 characters)
259	EXPRESS	Fullword	1 -> User is at an express terminal
260	TERMLOC	4 Bytes	1 -> 4-character terminal location code or binary zero, if unknown
261	PRDRMNTS	Fullword	Current number of paper-tape reader mounts
262	PRDRDRVT	Fullword	Current paper-tape reader drive time (seconds)
263	PPCHMNTS	Fullword	Current number of paper-tape punch mounts
264	PPCHDRVT	Fullword	Current paper-tape punch drive time (seconds)
265	FLPYMNTS	Fullword	Current number of floppy-disk mounts
266	FLPYDRVT	Fullword	Current floppy-disk drive time (seconds)
267	CUMPTRMT	Fullword	Cum. number of paper-tape reader mounts
268	CUMPTRDT	Fullword	Cum. paper-tape reader drive time (seconds)
269	CUMPTPMT	Fullword	Cum. number of paper-tape punch mounts
270	CUMPTPDT	Fullword	Cum. paper-tape punch drive time (seconds)

271	CUMFLPMT	Fullword	Cum. number of floppy-disk mounts
272	CUMFLPDT	Fullword	Cum. floppy-disk drive time (seconds)
276	ANSBACKL	Variable	Answerback code (characters) (see also item 74)
277	NOMOUNTS	Fullword	1 -> No tape or floppy-disk mounts allowed (see also item 252)
293	ONSHORT	Fullword	1 -> $SIGNON SHORT
294	TAPEQ	Fullword	1 -> Tape mount queuing is enabled
295	TAPEQLEN	Fullword	Length of current tape mount queue
296	PWSETBYC	Fullword	1 -> Password set by Computing Center
298	USERNAME	Variable	$SET NAME=name (from 1 to 64 characters)
300	DFLTMBOX	16 Bytes	Default mailbox (characters)
301	NAMELIB	Variable	File name for $SET NAMELIB=filename
302*	INITEDIT	Variable	File name for $SET INITFILE(EDIT)=FDname
303*	INITSDS	Variable	File name for $SET INITFILE(SDS)=FDname
304*	INITCALC	Variable	File name for $SET INITFILE(CALC)=FDname
305*	INITTST	Variable	File name for $SET INITFILE(TST)=FDname
306*	INITNET	Variable	File name for $SET INITFILE(NET)=FDname
307*	INITSSTA	Variable	File name for $SET INITFILE(SSTA)=FDname
308*	INITACC	Variable	File name for $SET INITFILE(ACC)=FDname
309*	INITNEW	Variable	File name for $SET INITFILE(NEW)=FDname
310*	INITNEW2	Variable	File name for $SET INITFILE(NEW2)=FDname
311*	INITNEW3	Variable	File name for $SET INITFILE(NEW3)=FDname
312*	INITPMF	Variable	File name for $SET INITFILE(PMF)=FDname
313*	INITMESS	Variable	File name for $SET INITFILE(MSG)=FDname
314*	INITFMNU	Variable	File name for $SET INITFILE(FMNU)=FDname
315*	INITMAKE	Variable	File name for $SET INITFILE(MAKE)=FDname
316*	INITLIST	Variable	File name for $SET INITFILE(LIST)=FDname
320	PAGPRLIN	Fullword	Current number of page printer lines
321	PAGPRPAG	Fullword	Current number of page printer pages
322	PAGPRIMG	Fullword	Current number of page printer images
323	PAGPRSHT	Fullword	Current number of page printer sheets
324	CUMPPL	Fullword	Cum. number of page printer lines
325	CUMPPP	Fullword	Cum. number of page printer pages
326	CUMPPI	Fullword	Cum. number of page printer images
327	CUMPPS	Fullword	Cum. number of page printer sheets
328	MACECHO	Fullword	SET MACROECHO={OFF|ON|ALL|ERROR} (0|1|2|3) (default OFF)
329	MACTRACE	Fullword	SET MACROTRACE={OFF|ON} (0|1) (default OFF)
330	MACRO	Fullword	$SET MACROS={OFF|ON} (0|1|2) (default OFF)
334	TZONOFST	Fullword	Current time zone offset from GMT (minutes)
335	TZONNAME	8 Bytes	Current time zone name (characters, left-justified with trailing blanks)
375*	NEWFILAC	Variable	$SET NEWFILEACCESS={'string'|OFF} (default OFF) (from 0 to 255 characters)
377	MTSMODEL	5 Bytes	MTS model number (characters)
386	PROJSIGF	Variable	File name for project sigfile
387	USERSIGF	Variable	File name for user sigfile (current sign-on)
388*	NEWSIGF	Variable	File name for user sigfile (next sign-on)
391*	LIBSRCH	Variable	$SET LIBSRCH={FDname|OFF} (default OFF)
392*	TIMLIMIT	Fullword	$SET TIME={n|OFF}[5]
393	PRJPWCHG	Fullword	1 -> $SET PROJECTPWCHANGE=ON (default OFF)
400*	ERRPRMPT	Fullword	1 -> $SET ERRORPROMPT=ON (default ON)
416	CPUCOST	Fullword	Cum. CPU cost for task (cents*100)

417	VMICOST	Fullword	Cum. VMI cost for task (cents*100)
418	HOSTNAME	8 Bytes	Host name (characters)
429	TYPEPTUS	Fullword	Cum. phototypesetter units for task
430	TYPEPAPR	Fullword	Cum. phototypesetter media for task (cm^2)
432	CKIDNOPW	Fullword	1 -> CKID does not need to check password
433	SERVER	Fullword	1 -> The job is a server program
440	PKEYSTR	Variable	Current Pkey
451*	SRVREPLY	Fullword	1 -> $SET SRVREPLY=ON (default OFF)

Table of Items Arranged by Name

Name	Index	Size	Description
$ON	171*	Fullword	1 -> $SET $=ON (default is OFF)
ACCTNO	14	Fullword	User account requisition number
AFDECHO	45*	Fullword	1 -> $SET AFDECHO=ON (default is OFF)
AFDINC	73*	Fullword	Line-number increment for *AFD* ($NUMBER)
AFDNBR	71*	Fullword	Next line number for *AFD* ($NUMBER)
ANSBACK	74	24 Bytes	Answerback code (characters) (see also item 276)
ANSBACKL	276	Variable	Answerback code (characters) (see also item 74)
ASYNCCTL	111	Fullword	Asynchronous event control switch[13]
ATNBIT	15*	Fullword	1 -> Attention interrupt occurred but not taken (may be set to cause an attention interrupt)
ATTNOFF	51*	Fullword	1 -> Stack attention interrupts (may be set to inhibit attention interrupts; pending interrupt may be taken on call to system subroutine)
ATTNTRP	69*	2 Words	ATTNTRP exit subroutine address (1st word) and save area location (2nd word)
AUTOHOLD	179*	Fullword	1 -> $SET AUTOHOLD=ON (default is OFF)
BATCHMD	10	Fullword	Batch (1) or conversational (0) mode
CELLTIME	114	Fullword	Last time datacell storage integral updated[3]. See Note (2).
CKIDNOPW	432	Fullword	1 -> CKID does not need to check password
CLSID	172	Fullword	Code for CLS currently in control[9]
CMDSCNBT	251*	Fullword	1 -> $SET CMDSCAN=UNAMBIGUOUS (default is UNAMBIGUOUS)
CMDSKP	185*	Fullword	1 -> $SET CMDSKP=OFF (default is OFF)
CONTCHAR	9*	4 Bytes	MTS command continuation character, left-justified with trailing blanks (default is "-"; $SET CONTCHAR=c)
COPIES	128	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
CPUCOST	416	Fullword	Cum. CPU cost for task (cents*100)
CROUTE	89*	4 Bytes	Default batch station for punched output (characters) ($SET CROUTE=rmid)
CUDISKIO	109	Fullword	Total number of disk operations for task
CUDRMRDS	170	Fullword	Current number of page-ins for current job
CUMCELL	126	Fullword	Cum. datacell file storage integral to CELLTIME which has been charged for (page hours). See Note (2).
CUMCORE	30	Fullword	Cum. storage integral over CPU time for ID (excluding active jobs)[2]
CUMCOREW	118	Fullword	Cum. storage integral over wait time for this ID (excluding active jobs)[2]
CUMCPUTM	28	Fullword	Cum. CPU time for ID (milliseconds) (excluding active jobs)
CUMDISK	76	Fullword	Cum. disk file storage integral to DISKTIME which has been charged for (page hours). See Note (2).
CUMELTM	26	Fullword	Cum. terminal time for ID (seconds) (excluding active jobs)

CUMFLPDT	272	Fullword	Cum. floppy-disk drive time (seconds)
CUMFLPMT	271	Fullword	Cum. number of floppy-disk mounts
CUMLINES	42	Fullword	Cum. number of lines printed for ID (excluding active jobs)
CUMMNET	192	Fullword	Cum. outbound Merit for this ID excluding active jobs (seconds)
CUMMONY	32	Fullword	Cum. charge used for ID (cents*100) (excluding active jobs)
CUMMOUNT	154	Fullword	Cum. number of tape mounts for ID (excluding active jobs)
CUMPAGES	44	Fullword	Cum. number of pages printed for ID (excluding active jobs)
CUMPLOT	122	Fullword	Cum. plot time for ID (seconds) (excluding active jobs)
CUMPLPAP	232	Fullword	Cum. plotter paper used for ID (.01 inches) (excluding active jobs)
CUMPLPEN	234	Fullword	Cum. plot pen changes for ID (excluding active jobs)
CUMPPI	326	Fullword	Cum. number of page printer images
CUMPPL	324	Fullword	Cum. number of page printer lines
CUMPPP	325	Fullword	Cum. number of page printer pages
CUMPPS	327	Fullword	Cum. number of page printer sheets
CUMPTLEN	158	Fullword	Cum. paper tape punched for ID (inches) (excluding active jobs)
CUMPTPDT	270	Fullword	Cum. paper-tape punch drive time (seconds)
CUMPTPMT	269	Fullword	Cum. number of paper-tape punch mounts
CUMPTRDT	268	Fullword	Cum. paper-tape reader drive time (seconds)
CUMPTRMT	267	Fullword	Cum. number of paper-tape reader mounts
CUMPTSM	159	Fullword	Cum. phototypesetter media (cm^2)
CUMPTSU	157	Fullword	Cum. phototypesetter units
CUMPUNCH	46	Fullword	Cum. number of cards punched for ID (excluding active jobs)
CUMREAD	29	Fullword	Cum. number of cards read for ID (excluding active jobs)
CUMTDRVT	156	Fullword	Cum. tape drive time for ID (seconds) (excluding active jobs)
CURRCELL	116	Fullword	Number of pages of datacell files in current use. See Note (2).
CURRDISK	24	Fullword	Number of pages of disk space in current use. See Note (2).
CURRSTOR	8	Fullword	Current number of half-pages of VM storage. See Note (1).
CURSIG	184	Fullword	Number of times this ID currently signed on
DEBUG	178*	Fullword	1 -> $SET DEBUG=ON (default is OFF)
DEBUGCMD	176	Fullword	1 -> If $DEBUG command active
DELIVERY	129*	8 Bytes	$SET DELIVERY={station|NONE} (characters) (default NONE)
DEVCHAR	39	4 Bytes	Device-name character, left-justified with trailing blanks (default is ">"; $SET DEVCHAR=c)
DFLTMBOX	300	16 Bytes	Default mailbox (characters)
DISKTIME	112	Fullword	Last time disk storage integral updated[3]. See Note (2).

DUMPTYPE	27*	Fullword	$SET ERRORDUMP={NOLIB|ON|LIB} (0|1|2) (default NOLIB)
EBM	119*	8 Bytes	The "execution begins" message--up to 7 characters, terminated with an *
ECHOOFF	49*	Fullword	1 -> $SET ECHO=OFF (default is ON)
EDITAFD	175*	Fullword	1 -> $SET EDITAFD=ON (default is OFF)
EFLUEM	183*	Fullword	Elementary Function Library, user error-monitor address
ENDFILSW	77*	Fullword	$SET ENDFILE={NEVER|SOURCE|ALWAYS} (0|1|2) (default SOURCE)
ERRPRMPT	400*	Fullword	1 -> $SET ERRORPROMPT=ON (default ON)
ETM	120*	8 Bytes	The "execution terminated" message--up to 7 characters, terminated with an *
EXECPFX	121*	4 Bytes	Execution prefix character ($SET EXECPFX=c) (left-justified)
EXPRESS	259	Fullword	1 -> User is at an express terminal
EXPTIME	54	Fullword	ID expiration time and date[3]
FILECHAR	5*	4 Bytes	File-name character, left-justified with trailing blanks (default is "#"; $SET FILECHAR=c)
FLPYDRVT	266	Fullword	Current floppy-disk drive time (seconds)
FLPYMNTS	265	Fullword	Current number of floppy-disk mounts
GLOBCPUT	78	Fullword	CPU time remaining in global time limit[5]. See Note (3).
GLOBPCH	82	Fullword	Global card estimate
GLOBPGS	80	Fullword	Global page estimate
GLOBPTM	84	Fullword	Global plot time estimate (seconds)
GLOBTTN	94	Fullword	Base for global time limit[5]. See Note (3).
HASPJOB	104	Fullword	1 -> Spooled batch job
HOSTNAME	418	8 Bytes	Host name (characters)
ICFBIT	11*	Fullword	1 -> $SET IC=OFF (default is ON)
IDRNBR	50	Fullword	User inter-departmental requisition number
INITACC	308*	Variable	File name for $SET INITFILE(ACC)=FDname
INITCALC	304*	Variable	File name for $SET INITFILE(CALC)=FDname
INITEDIT	302*	Variable	File name for $SET INITFILE(EDIT)=FDname
INITFMNU	314*	Variable	File name for $SET INITFILE(FMNU)=FDname
INITLIST	316*	Variable	File name for $SET INITFILE(LIST)=FDname
INITMAKE	315*	Variable	File name for $SET INITFILE(MAKE)=FDname
INITMESS	313*	Variable	File name for $SET INITFILE(MSG)=FDname
INITNET	306*	Variable	File name for $SET INITFILE(NET)=FDname
INITNEW	309*	Variable	File name for $SET INITFILE(NEW)=FDname
INITNEW2	310*	Variable	File name for $SET INITFILE(NEW2)=FDname
INITNEW3	311*	Variable	File name for $SET INITFILE(NEW3)=FDname
INITPMF	312*	Variable	File name for $SET INITFILE(PMF)=FDname
INITSDS	303*	Variable	File name for $SET INITFILE(SDS)=FDname
INITSSTA	307*	Variable	File name for $SET INITFILE(SSTA)=FDname
INITTST	305*	Variable	File name for $SET INITFILE(TST)=FDname
INSIGFIL	226	Fullword	1 -> Currently processing sigfile
LASTEXRC	239	Fullword	Return code of last program executed
LASTSOT	152	16 Bytes	Last signon time in characters
LDROPT	33*	4 Bytes	Loader options switches in leftmost byte[10]
LIBROFF	43*	Fullword	1 -> $SET LIBR=OFF (default is ON)
LIBSRCH	391*	Variable	$SET LIBSRCH={FDname|OFF} (default OFF)
LINKLEVL	130	Fullword	Current link level (see MTS Vol. 5 Virtual

			Memory Management description)
LNS	1*	4 Bytes	Line-number separator character, left-justified with trailing blanks (default is ","; $SET LNS=c)
LOCCPUT	86	Fullword	CPU time remaining in local time limit[5]. See Note (3).
LOCLIMIT	253	Fullword	Local time limit in effect[5]
LOCPCH	90	Fullword	Local card estimate
LOCPGS	88	Fullword	Local page estimate
LOCPTM	92	Fullword	Local plot time estimate (seconds)
LOCSW	12	Fullword	1 -> Local time estimate active
LOCTTN	96	Fullword	Base for local time limit[5]. See Note (3).
LODRSYMT	138	Fullword	Loader symbol table location
LSIGTMUT	149	18 Bytes	Last signon time (Universal Time Units). See Note (4).
LSS	180	Fullword	1 -> If limited-service state active
LSTRESET	110	Fullword	Last time cum. totals for this ID were reset[3]
MACECHO	328	Fullword	SET MACROECHO={OFF|ON|ALL|ERROR} (0|1|2|3) (default OFF)
MACTRACE	329	Fullword	SET MACROTRACE={OFF|ON} (0|1) (default OFF)
MACRO	330	Fullword	$SET MACROS={OFF|ON} (0|1|2) (default OFF)
MAPDOTS	197*	Fullword	1 -> $SET MAPDOTS=ON (default is ON)
MAXCELL	106	Fullword	Maximum datacell pages allowed for ID
MAXDISK	18	Fullword	Maximum number of disk pages allowed for ID
MAXMNET	190	Fullword	Maximum outbound Merit time (seconds)
MAXMONY	22	Fullword	Maximum charge allowed for ID (cents*100)
MAXPLOT	108	Fullword	Maximum plot time allowed for ID (seconds)
MAXSIG	182	Fullword	Max. number of concurrent signons allowed for ID (0=unlimited)
MAXTERM	20	Fullword	Maximum terminal time allowed for ID (seconds)
MNETTIME	87	Fullword	Outbound Merit time for this job (seconds)
MTSMODEL	377	5 Bytes	MTS model number (characters)
MXMNETBT	194	Fullword	1 -> Ignore maximum MNET time (item 190)
MXPLOTBT	196	Fullword	1 -> Ignore maximum plot time (item 108)
MXSTRIND	136	Fullword	Maximum storage index number used (See MTS Vol. 5 Virtual Memory Management description)
NAMELIB	301	Variable	File name for $SET NAMELIB=filename
NEWFILAC	375*	Variable	$SET NEWFILEACCESS={'string'|OFF} (default OFF) (from 0 to 255 characters)
NEWSIGF	388*	Variable	File name for user sigfile (next sign-on)
NO*LIB	195*	Fullword	1 -> $SET *LIBRARY=OFF (default is ON)
NOERRMAP	199*	Fullword	1 -> $SET ERRMAP=OFF (default is ON)
NOMOUNTS	277	Fullword	1 -> No tape or floppy-disk mounts allowed (see also item 252)
NRBATCH	40	Fullword	Cum. number of batch jobs for ID (excluding active jobs)
NRCELLF	124	Fullword	Number of datacell files existing for ID
NRCREATE	60	Fullword	Number of files created during current job
NRDESTRY	62	Fullword	Number of files destroyed during current job
NRDISKF	36	Fullword	Number of disk files existing for ID
NRLINES	63	Fullword	Number of lines printed for current job
NRMOUNT	79	Fullword	Number of tape and other mounts for current job
NRPAGES	65	Fullword	Number of pages printed for current job

NRPUNCH	67	Fullword	Number of cards punched for current job
NRREAD	31	Fullword	Number of cards read for current job
NRSIGS	38	Fullword	Cum. number of signons for ID (excluding active jobs)
NUMBER	41*	Fullword	1 -> Automatic numbering active ($NUMBER)
NXTSEGSW	19*	Fullword	1 -> Skip to next set of MTS command cards (batch only; may be set to skip unread data
ONSHORT	293	Fullword	1 -> $SIGNON SHORT
OFFBIT	23*	Fullword	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
PAGPRIMG	322	Fullword	Current number of page printer images
PAGPRLIN	320	Fullword	Current number of page printer lines
PAGPRPAG	321	Fullword	Current number of page printer pages
PAGPRSHT	323	Fullword	Current number of page printer sheets
PAPER	125*	12 Bytes	$SET PAPER={PLAIN|3HOLE|ANY} (characters) (default 0 (ANY))
PARSTR	258	Variable	The PAR string from the MTS $RUN command converted to uppercase (from 0 to 255 characters)
PARSTRMC	255	Variable	The PAR string from the MTS $RUN command in mixed-case (from 0 to 255 characters)
PCLSID	174	Fullword	Code for CLS that called current CLS[9]
PDMAPOFF	189*	Fullword	1 -> $SET PDMAP=OFF (default is OFF)
PFXOFF	57*	Fullword	1 -> $SET PFX=OFF (default is ON)
PFXSTR	257*	Variable	Prefix string which normally appears at the beginning of terminal input and output lines (from 0 to 120 characters in length)
PGNTTRP	61*	2 Words	PGNTTRP exit subroutine address (1st word) and save area location (2nd word)
PKEY	236	16 Bytes	Program key under which calling program is running
PKEYSTR	440	Variable	Current Pkey
PLOTPAPR	227	Fullword	Plotter paper used for current job (.01 inches)
PLOTPENC	229	Fullword	Plotter pen changes for current job
PLOTTIME	25	Fullword	Total plot time for current job (seconds)
PPCHDRVT	264	Fullword	Current paper-tape punch drive time (seconds)
PPCHMNTS	263	Fullword	Current number of paper-tape punch mounts
PRDRDRVT	262	Fullword	Current paper-tape reader drive time (seconds)
PRDRMNTS	261	Fullword	Current number of paper-tape reader mounts
PREFIXC	3*	Fullword	Current prefix character, left-justified with trailing blanks, as set by the SETPFX subroutine or CUINFO item 257 (PFXSTR).
PRINT	93*	4 Bytes	Print train specification ("PN ", "TN ", "UC ", "MC ", or binary 0 in first byte if ANY)
PRINTER	127*	4 Bytes	$SET PRINTER={LINE|PAGE|ANY} (characters) (default ANY)
PRIO	242	Fullword	Priority of job[12]
PRJPWCHG	393	Fullword	1 -> $SET PROJECTPWCHANGE=ON (default OFF)
PRMAPOFF	187*	Fullword	1 -> $SET PRMAP=OFF (default is OFF)
PRNTCDSW	21*	Fullword	1 -> Print next input line from source if not MTS command (batch only)
PROJNO	16	4 Bytes	Project (charge) ID in characters
PROJSIGF	386	Variable	File name for project sigfile

PROUTE	91*	4 Bytes	Default batch station for printed output (characters) ($SET PROUTE=rmid)
PSFATTN	249	Fullword	1 -> Project sigfile attention bit is off
PTLEN	83	Fullword	Paper tape punched for current job (inches)
PWSETBYC	296	Fullword	1 -> Password set by Computing Center
RCPRINT	237*	Fullword	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS|NONZERO} (0|1|2|3|4)
RF	37*	Fullword	Relocation factor for ALTER/DISPLAY/MODIFY commands (Default is 0; $SET RF=xxxxxx)
RUNETIME	115	Dblword	Cumulative real time for program[5]
RUNONLY	238	Fullword	1 -> A "run only" program is loaded (from a file to which the user has only RUN access)
RUNTIME	254	Fullword	Amount of time used during execution of current program[5]. This total is updated only when execution mode is exited, e.g., if program calls MTS
SCOPIES	95	Fullword	Number of copies of printed output requested on $SET COPIES=n command
SCRCELTM	164	Fullword	Last time scratch datacell file storage integral updated[3]. See Note (2).
SCRCLUSE	168	Fullword	Scratch datacell file storage integral to SCRCELTM[7]. See Note (2).
SCRDSKTM	162	Fullword	Last time scratch disk file storage integral updated[3]. See Note (2).
SCRDSUSE	166	Fullword	Scratch disk file storage integral to SCRDSKTM[7]. See Note (2).
SCRFCELL	148	Fullword	Number of pages of datacell scratch files for current job. See Note (2).
SCRFCHAR	7*	4 Bytes	Scratch-file character, left-justified with trailing blanks (default is "-"; $SET SCRFCHAR=c)
SCRFDISK	146	Fullword	Number of pages of disk scratch files for current job. See Note (2).
SEE_DISP	99*	Fullword	1 -> $SET DISPATCH=ON (default ON)
SEQCOFF	59*	Fullword	1 -> $SET SEQFCHK=OFF (default is ON)
SERVER	433	Fullword	1 -> The job is a server program
SETIOERR	75*	Fullword	SETIOERR exit subroutine address
SFATTN	247*	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
SHFSEP	35*	4 Bytes	Shared-file separator character, left-justified with trailing blanks (default is ":"; $SET SHFSEP=c)
SIGCFLD	241	Variable	The comment field from the MTS $SIGNON command, without the enclosing primes (from 0 to 255 characters in length)
SIGFATTN	167*	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
SIGNONID	2	4 Bytes	Current signon ID
SIGOFRCT	151*	Fullword	1 -> Display receipt summary at signoff
SIGSHORT	55*	Fullword	$SIGNOFF {LONG\|SHORT\|$} (0|1|2) (default is LONG)
SIGTMUT	13	18 Bytes	Signon time (Universal Time Units). See Note (4).
SOBCDTM	56	16 Bytes	Signon time and date in characters
SOCPUTC	66	Fullword	Supervisor state CPU time used by task before

			current signon[5]
SOCPUTP	64	Fullword	Problem state CPU time used by task before current signon[5]
SODISKIO	107	Fullword	Number of disk operations at signon for task
SODRMRDS	150	Fullword	Number of page-ins by task before signon
SOELT	68	Dblword	Time of day at signon[6]
SOPTOD	72	16 Bytes	Time and date for header page for batch output (characters)
SPELLCOR	231*	Fullword	$SET SPELLCOR={OFF|PROMPT|ON} (0|3|1) (default is PROMPT)
SRVREPLY	451*	Fullword	1 -> $SET SRVREPLY=ON (default OFF)
STORCPUT	58	Fullword	Current base for CPU storage integral[4]. See Note (1).
STORELT	70	Fullword	Current base for elapsed storage integral[4]. See Note (1).
STORINDX	134	Fullword	Current storage index number (See MTS Vol. 5 Virtual Memory Management description)
STORUSED	6	Fullword	CPU storage integral to STORCPUT[1]. See Note (1).
STORUSEE	48	Fullword	Elapsed storage integral to STORELT[1]. See Note (1).
SVCTRP	113*	2 Words	SVCTRP exit subroutine address (1st word) and save area location (2nd word)
SYMTAB	47*	Fullword	1 -> $SET SYMTAB=ON (default is ON)
SYSOLOAD	240	Fullword	System overload indicators, right-justified with leading zeros[11]
S8NBR	4	8 Bytes	Receipt number of job in characters, left-justified with trailing blanks (batch only)
TAPEQ	294	Fullword	1 -> Tape mount queuing is enabled
TAPEQLEN	295	Fullword	Length of current tape mount queue
TASKNBR	98	Fullword	Task number
TASKTYPE	100	Fullword	Task type code[8]
TDR	85*	Fullword	1 -> $SET TDR=ON (default is OFF)
TDRVT	81	Fullword	Tape drive time for current job (seconds)
TERMLOC	260	4 Bytes	1 -> 4-character terminal location code or binary zero, if unknown
TERSE	169*	Fullword	1 -> $SET TERSE=ON (default is OFF)
TIMEFDGE	230	Dblword	Value (microseconds times 4096) to be added to IBM time (as stored by a STCK instruction) to get time based on March 1, 1900
TIMLIMIT	392*	Fullword	$SET TIME={n|OFF}[5]
TOFFSET	228	Dblword	Offset (microseconds times 4096) to be added to GMT to get local time
TRIMBIT	181*	Fullword	1 -> $SET TRIM=ON (default is ON)
TYPEPAPR	430	Fullword	Cum. phototypesetter media for task (cm^2)
TYPEPTUS	429	Fullword	Cum. phototypesetter units for task
TZONNAME	335	8 Bytes	Current time zone name (characters, left-justified with trailing blanks)
TZONOFST	334	Fullword	Current time zone offset from GMT (minutes)
UCBIT	17*	Fullword	1 -> $SET CASE=UC (default is MC)
UNATMODE	252	Fullword	1 -> System running in "unattended mode" (see also item 277)
UNCHCELL	188	Fullword	Datacell space to CELLTIME not yet charged for[7]

UNCHDISK	186	Fullword	Disk space to DISKTIME not yet charged for[7]
USERNAME	298	Variable	$SET NAME=name (from 1 to 64 characters)
USERSIGF	387	Variable	File name for user sigfile (current sign-on)
UNITCODE	52	Fullword	User unit code
USMSG	177*	Fullword	1 -> $SET USMSG=ON (default is ON)
UXREF	191*	Fullword	1 -> $SET UXREF=ON (default is OFF)
VMICOST	417	Fullword	Cum. VMI cost for task (cents*100)
XREF	193*	Fullword	1 -> $SET XREF=ON (default is OFF)

Table of System Items Arranged by Subject

Index	Name	Size	Description
Accounting - Batch Input and Output			
29	CUMREAD	Fullword	Cum. number of cards read for ID (excluding active jobs)
31	NRREAD	Fullword	Number of cards read for current job
40	NRBATCH	Fullword	Cum. number of batch jobs for ID (excluding active jobs)
42	CUMLINES	Fullword	Cum. number of lines printed for ID (excluding active jobs)
44	CUMPAGES	Fullword	Cum. number of pages printed for ID (excluding active jobs)
46	CUMPUNCH	Fullword	Cum. number of cards punched for ID (excluding active jobs)
63	NRLINES	Fullword	Number of lines printed for current job
65	NRPAGES	Fullword	Number of pages printed for current job
67	NRPUNCH	Fullword	Number of cards punched for current job
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
320	PAGPRLIN	Fullword	Current number of page printer lines
321	PAGPRPAG	Fullword	Current number of page printer pages
322	PAGPRIMG	Fullword	Current number of page printer images
323	PAGPRSHT	Fullword	Current number of page printer sheets
324	CUMPPL	Fullword	Cum. number of page printer lines
325	CUMPPP	Fullword	Cum. number of page printer pages
326	CUMPPI	Fullword	Cum. number of page printer images
327	CUMPPS	Fullword	Cum. number of page printer sheets
Accounting - CPU, Memory, and Paging			
6	STORUSED	Fullword	CPU storage integral to STORCPUT[1]. See Note (1).
8	CURRSTOR	Fullword	Current number of half-pages of VM storage. See Note (1).
28	CUMCPUTM	Fullword	Cum. CPU time for ID (milliseconds) (excluding active jobs)
30	CUMCORE	Fullword	Cum. storage integral over CPU time for ID (excluding active jobs)[2]
48	STORUSEE	Fullword	Elapsed storage integral to STORELT[1]. See Note (1).
58	STORCPUT	Fullword	Current base for CPU storage integral[4]. See Note (1).
64	SOCPUTP	Fullword	Problem state CPU time used by task before current signon[5]
66	SOCPUTC	Fullword	Supervisor state CPU time used by task before current signon[5]
70	STORELT	Fullword	Current base for elapsed storage integral[4]. See Note (1).

107	SODISKIO	Fullword	Number of disk operations at signon for task
109	CUDISKIO	Fullword	Total number of disk operations for task
118	CUMCOREW	Fullword	Cum. storage integral over wait time for this ID (excluding active jobs)[2]
150	SODRMRDS	Fullword	Number of page-ins by task before signon
170	CUDRMRDS	Fullword	Current number of page-ins for current job
254	RUNTIME	Fullword	Amount of time used during execution of current program[5]. This total is updated only when execution mode is exited, e.g., if program calls MTS

<u>Accounting - File System Storage</u>

18	MAXDISK	Fullword	Maximum number of disk pages allowed for ID
24	CURRDISK	Fullword	Number of pages of disk space in current use. See Note (2).
36	NRDISKF	Fullword	Number of disk files existing for ID
60	NRCREATE	Fullword	Number of files created during current job
62	NRDESTRY	Fullword	Number of files destroyed during current job
76	CUMDISK	Fullword	Cum. disk file storage integral to DISKTIME which has been charged for (page hours). See Note (2).
112	DISKTIME	Fullword	Last time disk storage integral updated[3]
146	SCRFDISK	Fullword	Number of pages of disk scratch files for current job. See Note (2).
162	SCRDSKTM	Fullword	Last time scratch disk file storage integral updated[3]. See Note (2).
166	SCRDSUSE	Fullword	Scratch disk file storage integral to SCRDSKTM[7]. See Note (2).
186	UNCHDISK	Fullword	Disk space to DISKTIME not yet charged for[7]

<u>Accounting - Magnetic Tapes, Paper Tapes, and Floppy Disks</u>

79	NRMOUNT	Fullword	Number of tape and other mounts for current job
81	TDRVT	Fullword	Tape drive time for current job (seconds)
83	PTLEN	Fullword	Paper tape punched for current job (inches)
154	CUMMOUNT	Fullword	Cum. number of tape mounts for ID (excluding active jobs)
156	CUMTDRVT	Fullword	Cum. tape drive time for ID (seconds) (excluding active jobs)
158	CUMPTLEN	Fullword	Cum. paper tape punched for ID (inches) (excluding active jobs)
252	UNATMODE	Fullword	1 -> System running in "unattended mode" (see also item 277)
261	PRDRMNTS	Fullword	Current number of paper-tape reader mounts
262	PRDRDRVT	Fullword	Current paper-tape reader drive time (seconds)
263	PPCHMNTS	Fullword	Current number of paper-tape punch mounts
264	PPCHDRVT	Fullword	Current paper-tape punch drive time (seconds)
265	FLPYMNTS	Fullword	Current number of floppy-disk mounts
266	FLPYDRVT	Fullword	Current floppy-disk drive time (seconds)
267	CUMPTRMT	Fullword	Cum. number of paper-tape reader mounts
268	CUMPTRDT	Fullword	Cum. paper-tape reader drive time (seconds)
269	CUMPTPMT	Fullword	Cum. number of paper-tape punch mounts

270	CUMPTPDT	Fullword	Cum. paper-tape punch drive time (seconds)
271	CUMFLPMT	Fullword	Cum. number of floppy-disk mounts
272	CUMFLPDT	Fullword	Cum. floppy-disk drive time (seconds)
277	NOMOUNTS	Fullword	1 -> No tape or floppy-disk mounts allowed (see also item 252)
294	TAPEQ	Fullword	1 -> Tape mount queuing is enabled
295	TAPEQLEN	Fullword	Length of current tape mount queue

Accounting - Money

2	SIGNONID	4 Bytes	Current signon ID
14	ACCTNO	Fullword	User account requisition number
16	PROJNO	4 Bytes	Project (charge) ID in characters
22	MAXMONY	Fullword	Maximum charge allowed for ID (cents*100)
32	CUMMONY	Fullword	Cum. charge used for ID (cents*100) (excluding active jobs)
50	IDRNBR	Fullword	User inter-departmental requisition number
52	UNITCODE	Fullword	User unit code
416	CPUCOST	Fullword	Cum. CPU cost for task (cents*100)
417	VMICOST	Fullword	Cum. VMI cost for task (cents*100)

Accounting - Phototypesetter Use

157	CUMPTSU	Fullword	Cum. phototypesetter units
159	CUMPTSM	Fullword	Cum. phototypesetter media (cm^2)
429	TYPEPTUS	Fullword	Cum. phototypesetter units for task
430	TYPEPAPR	Fullword	Cum. phototypesetter media for task (cm^2)

Accounting - Plotter Use

25	PLOTTIME	Fullword	Total plot time for current job (seconds)
108	MAXPLOT	Fullword	Maximum plot time allowed for ID (seconds)
122	CUMPLOT	Fullword	Cum. plot time for ID (seconds) (excluding active jobs)
196	MXPLOTBT	Fullword	1 -> Ignore maximum plot time (item 108)
227	PLOTPAPR	Fullword	Plotter paper used for current job (.01 inches)
229	PLOTPENC	Fullword	Plotter pen changes for current job
232	CUMPLPAP	Fullword	Cum. plotter paper used for ID (.01 inches) (excluding active jobs)
234	CUMPLPEN	Fullword	Cum. plot pen changes for ID (excluding active jobs)

Accounting - Terminal and Merit Computer Network Use

20	MAXTERM	Fullword	Maximum terminal time allowed for ID (seconds)
26	CUMELTM	Fullword	Cum. terminal time for ID (seconds) (excluding active jobs)
87	MNETTIME	Fullword	Outbound Merit time for this job (seconds)
190	MAXMNET	Fullword	Maximum outbound Merit time (seconds)
192	CUMMNET	Fullword	Cum. outbound Merit for this ID excluding active jobs (seconds)
194	MXMNETBT	Fullword	1 -> Ignore maximum MNET time (item 190)

Accounting - User ID and Project Information

2	SIGNONID	4 Bytes	Current signon ID
16	PROJNO	4 Bytes	Project (charge) ID in characters
38	NRSIGS	Fullword	Cum. number of signons for ID (excluding active jobs)
50	IDRNBR	Fullword	User inter-departmental requisition number
52	UNITCODE	Fullword	User unit code
54	EXPTIME	Fullword	ID expiration time and date[3]
110	LSTRESET	Fullword	Last time cum. totals for this ID were reset[3]
149	LSIGTMUT	18 Bytes	Last signon time (Universal Time Units). See Note (4).
152	LASTSOT	16 Bytes	Last signon time in characters
160	BILLCLAS	Fullword	Billing class (0=University 1=Industrial, 2=Exchange)
167*	SIGFATTN	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
182	MAXSIG	Fullword	Max. number of concurrent signons allowed for ID (0=unlimited)
184	CURSIG	Fullword	Number of times this ID currently signed on
247*	SFATTN	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
249	PSFATTN	Fullword	1 -> Project sigfile attention bit is off
296	PWSETBYC	Fullword	1 -> Password set by Computing Center
386	PROJSIGF	Variable	File name for project sigfile
387	USERSIGF	Variable	File name for user sigfile (current sign-on)
388*	NEWSIGF	Variable	File name for user sigfile (next sign-on)
393	PRJPWCHG	Fullword	1 -> $SET PROJECTPWCHANGE=ON (default OFF)
432	CKIDNOPW	Fullword	1 -> CKID does not need to check password

Batch Mode Jobs

4	S8NBR	8 Bytes	Receipt number of job in characters, left-justified with trailing blanks (batch only)
10	BATCHMD	Fullword	Batch (1) or conversational (0) mode
72	SOPTOD	16 Bytes	Time and date for header page for batch output (characters)
89*	CROUTE	4 Bytes	Default batch station for punched output (characters) ($SET CROUTE=rmid)
91*	PROUTE	4 Bytes	Default batch station for printed output (characters) ($SET PROUTE=rmid)
93*	PRINT	4 Bytes	Print train specification ("PN ", "TN ", "UC ", "MC ", or binary 0 in first byte if ANY)
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
104	HASPJOB	Fullword	1 -> Spooled batch job
127*	PRINTER	4 Bytes	$SET PRINTER={LINE|PAGE|ANY} (characters) (default ANY)
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
129*	DELIVERY	8 Bytes	$SET DELIVERY={station|NONE} (characters) (default NONE)
179*	AUTOHOLD	Fullword	1 -> $SET AUTOHOLD=ON (default is OFF)

Command Language Options

1*	LNS	4 Bytes	Line-number separator character, left-justified with trailing blanks (default is ","; $SET LNS=c)
5*	FILECHAR	4 Bytes	File-name character, left-justified with trailing blanks (default is "#"; $SET FILECHAR=c)
7*	SCRFCHAR	4 Bytes	Scratch-file character, left-justified with trailing blanks (default is "-"; $SET SCRFCHAR=c)
9*	CONTCHAR	4 Bytes	MTS command continuation character, left-justified with trailing blanks (default is "-"; $SET CONTCHAR=c)
11*	ICFBIT	Fullword	1 -> $SET IC=OFF (default is ON)
17*	UCBIT	Fullword	1 -> $SET CASE=UC (default is MC)
27*	DUMPTYPE	Fullword	$SET ERRORDUMP={NOLIB|OFF|LIB} (0|1|2) (default NOLIB)
35*	SHFSEP	4 Bytes	Shared-file separator character, left-justified with trailing blanks (default is ":"; $SET SHFSEP=c)
37*	RF	Fullword	Relocation factor for ALTER/DISPLAY/MODIFY commands (Default is 0; $SET RF=xxxxxx)
39	DEVCHAR	4 Bytes	Device-name character, left-justified with trailing blanks (default is ">"; $SET DEVCHAR=c)
41*	NUMBER	Fullword	1 -> Automatic numbering active ($NUMBER)
43*	LIBROFF	Fullword	1 -> $SET LIBR=OFF (default is ON)
45*	AFDECHO	Fullword	1 -> $SET AFDECHO=ON (default is OFF)
47*	SYMTAB	Fullword	1 -> $SET SYMTAB=ON (default is ON)
49*	ECHOOFF	Fullword	1 -> $SET ECHO=OFF (default is ON)
55*	SIGSHORT	Fullword	$SIGNOFF {LONG\|SHORT\|$} (0|1|2) (default is LONG)
57*	PFXOFF	Fullword	1 -> $SET PFX=OFF (default is ON)
59*	SEQCOFF	Fullword	1 -> $SET SEQFCHK=OFF (default is ON)
71*	AFDNBR	Fullword	Next line number for *AFD* ($NUMBER)
73*	AFDINC	Fullword	Line-number increment for *AFD* ($NUMBER)
77*	ENDFILSW	Fullword	$SET ENDFILE={NEVER|SOURCE|ALWAYS} (0|1|2) (default SOURCE)
85*	TDR	Fullword	1 -> $SET TDR=ON (default is OFF)
89*	CROUTE	4 Bytes	Default batch station for punched output (characters) ($SET CROUTE=rmid)
91*	PROUTE	4 Bytes	Default batch station for printed output (characters) ($SET PROUTE=rmid)
93*	PRINT	4 Bytes	Print train specification ("PN ", "TN ", "UC ", "MC ", or binary 0 in first byte if ANY)
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
99*	SEE_DISP	Fullword	1 -> $SET DISPATCH=ON (default ON)
119*	EBM	8 Bytes	The "execution begins" message--up to 7 characters, terminated with an *
120*	ETM	8 Bytes	The "execution terminated" message--up to 7 characters, terminated with an *

121*	EXECPFX	4 Bytes	Execution prefix character ($SET EXECPFX=c) (left-justified)
125*	PAPER	12 Bytes	$SET PAPER={PLAIN|3HOLE|ANY} (characters) (default 0 (ANY))
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
151*	SIGOFRCT	Fullword	1 -> Display receipt summary at signoff
169*	TERSE	Fullword	1 -> $SET TERSE=ON (default is OFF)
171*	$ON	Fullword	1 -> $SET $=ON (default is OFF)
175*	EDITAFD	Fullword	1 -> $SET EDITAFD=ON (default is OFF)
176	DEBUGCMD	Fullword	1 -> If $DEBUG command active
177*	USMSG	Fullword	1 -> $SET USMSG=ON (default is ON)
178*	DEBUG	Fullword	1 -> $SET DEBUG=ON (default is OFF)
179*	AUTOHOLD	Fullword	1 -> $SET AUTOHOLD=ON (default is OFF)
181*	TRIMBIT	Fullword	1 -> $SET TRIM=ON (default is ON)
185*	CMDSKP	Fullword	1 -> $SET CMDSKP=OFF (default is OFF)
187*	PRMAPOFF	Fullword	1 -> $SET PRMAP=OFF (default is OFF)
189*	PDMAPOFF	Fullword	1 -> $SET PDMAP=OFF (default is OFF)
191*	UXREF	Fullword	1 -> $SET UXREF=ON (default is OFF)
193*	XREF	Fullword	1 -> $SET XREF=ON (default is OFF)
195*	NO*LIB	Fullword	1 -> $SET *LIBRARY=OFF (default is ON)
197*	MAPDOTS	Fullword	1 -> $SET MAPDOTS=ON (default is ON)
199*	NOERRMAP	Fullword	1 -> $SET ERRMAP=OFF (default is ON)
231*	SPELLCOR	Fullword	$SET SPELLCOR={OFF|PROMPT|ON} (0|3|1) (default is PROMPT)
233*	NOSDS	Fullword	1 -> $SET SDSMSG=OFF (default is ON)
237*	RCPRINT	Fullword	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS} (0|1|2|3)
251*	CMDSCNBT	Fullword	1 -> $SET CMDSCAN=UNAMBIGUOUS (default is UNAMBIGUOUS)
293	ONSHORT	Fullword	1 -> $SIGNON SHORT
298	USERNAME	Variable	$SET NAME=name (from 1 to 64 characters)
300	DFLTMBOX	16 Bytes	Default mailbox (characters)
301	NAMELIB	Variable	File name for $SET NAMELIB=filename
302*	INITEDIT	Variable	File name for $SET INITFILE(EDIT)=FDname
303*	INITSDS	Variable	File name for $SET INITFILE(SDS)=FDname
304*	INITCALC	Variable	File name for $SET INITFILE(CALC)=FDname
305*	INITTST	Variable	File name for $SET INITFILE(TST)=FDname
306*	INITNET	Variable	File name for $SET INITFILE(NET)=FDname
307*	INITSSTA	Variable	File name for $SET INITFILE(SSTA)=FDname
308*	INITACC	Variable	File name for $SET INITFILE(ACC)=FDname
309*	INITNEW	Variable	File name for $SET INITFILE(NEW)=FDname
310*	INITNEW2	Variable	File name for $SET INITFILE(NEW2)=FDname
311*	INITNEW3	Variable	File name for $SET INITFILE(NEW3)=FDname
312*	INITPMF	Variable	File name for $SET INITFILE(PMF)=FDname
313*	INITMESS	Variable	File name for $SET INITFILE(MSG)=FDname
314*	INITFMNU	Variable	File name for $SET INITFILE(FMNU)=FDname
315*	INITMAKE	Variable	File name for $SET INITFILE(MAKE)=FDname
316*	INITLIST	Variable	File name for $SET INITFILE(LIST)=FDname
328	MACECHO	Fullword	SET MACROECHO={OFF|ON|ALL|ERROR} (0|1|2|3) (default OFF)
329	MACTRACE	Fullword	SET MACROTRACE={OFF|ON} (0|1) (default OFF)
330	MACRO	Fullword	$SET MACROS={OFF|ON} (0|1|2) (default OFF)

375*	NEWFILAC	Variable	$SET NEWFILEACCESS={'string'|OFF} (default OFF) (from 0 to 255 characters)
391*	LIBSRCH	Variable	$SET LIBSRCH={FDname|OFF} (default OFF)
392*	TIMLIMIT	Fullword	$SET TIME={n|OFF}[5]
400*	ERRPRMPT	Fullword	1 -> $SET ERRORPROMPT=ON (default ON)
451*	SRVREPLY	Fullword	1 -> $SET SRVREPLY=ON (default OFF)

Execution Processing

3*	PREFIXC	Fullword	Current prefix character, left-justified with trailing blanks, as set by the SETPFX subroutine or CUINFO item 257 (PFXSTR).
19*	NXTSEGSW	Fullword	1 -> Skip to next set of MTS command cards (batch only; may be set to skip unread data cards)
21*	PRNTCDSW	Fullword	1 -> Print next input line from source if not MTS command (batch only)
23*	OFFBIT	Fullword	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
27*	DUMPTYPE	Fullword	$SET ERRORDUMP={NOLIB|OFF|LIB} (0|1|2) (default NOLIB)
33*	LDROPT	4 Bytes	Loader options switches in leftmost byte[10]
43*	LIBROFF	Fullword	1 -> $SET LIBR=OFF (default is ON)
47*	SYMTAB	Fullword	1 -> $SET SYMTAB=ON (default is ON)
115	RUNETIME	Dblword	Cumulative real time for program[5]
119*	EBM	8 Bytes	The "execution begins" message--up to 7 characters, terminated with an *
120*	ETM	8 Bytes	The "execution terminated" message--up to 7 characters, terminated with an *
121*	EXECPFX	4 Bytes	Execution prefix character ($SET EXECPFX=c) (left-justified)
130	LINKLEVL	Fullword	Current link level (see MTS Vol. 5 Virtual Memory Management description)
134	STORINDX	Fullword	Current storage index number (See MTS Vol. 5 Virtual Memory Management description)
136	MXSTRIND	Fullword	Maximum storage index number used (See MTS Vol. 5 Virtual Memory Management description)
138	LODRSYMT	Fullword	Loader symbol table location
176	DEBUGCMD	Fullword	1 -> If $DEBUG command active
177*	USMSG	Fullword	1 -> $SET USMSG=ON (default is ON)
178*	DEBUG	Fullword	1 -> $SET DEBUG=ON (default is OFF)
187*	PRMAPOFF	Fullword	1 -> $SET PRMAP=OFF (default is OFF)
189*	PDMAPOFF	Fullword	1 -> $SET PDMAP=OFF (default is OFF)
191*	UXREF	Fullword	1 -> $SET UXREF=ON (default is OFF)
193*	XREF	Fullword	1 -> $SET XREF=ON (default is OFF)
195*	NO*LIB	Fullword	1 -> $SET *LIBRARY=OFF (default is ON)
197*	MAPDOTS	Fullword	1 -> $SET MAPDOTS=ON (default is ON)
199*	NOERRMAP	Fullword	1 -> $SET ERRMAP=OFF (default is ON)
236	PKEY	16 Bytes	Program key under which calling program is running
237*	RCPRINT	Fullword	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS|NONZERO} (0|1|2|3|4)
238	RUNONLY	Fullword	1 -> A "run only" program is loaded (from a

			file to which the user has only RUN access)
239	LASTEXRC	Fullword	Return code of last program executed
255	PARSTRMC	Variable	The PAR string from the MTS $RUN command in mixed-case (from 0 to 255 characters)
258	PARSTR	Variable	The PAR string from the MTS $RUN command converted to uppercase (from 0 to 255 characters)
440	PKEYSTR	Variable	Current Pkey

Interrupt Processing

15*	ATNBIT	Fullword	1 -> Attention interrupt occurred but not taken (may be set to cause an attention interrupt)
51*	ATTNOFF	Fullword	1 -> Stack attention interrupts (may be set to inhibit attention interrupts; pending interrupt may be taken on call to system subroutine)
61*	PGNTTRP	2 Words	PGNTTRP exit subroutine address (1st word) and save area location (2nd word)
69*	ATTNTRP	2 Words	ATTNTRP exit subroutine address (1st word) and save area location (2nd word)
75*	SETIOERR	Fullword	SETIOERR exit subroutine address
111	ASYNCCTL	Fullword	Asynchronous event control switch[13]
113*	SVCTRP	2 Words	SVCTRP exit subroutine address (1st word) and save area location (2nd word)
167*	SIGFATTN	Fullword	1 -> $SET SIGFILEATTN=OFF (default is ON)
183*	EFLUEM	Fullword	Elementary Function Library, user error-monitor address
249	PSFATTN	Fullword	1 -> Project sigfile attention bit is off

I/O File and Device Names

5*	FILECHAR	4 Bytes	File-name character, left-justified with trailing blanks (default is "#"; $SET FILECHAR=c)
7*	SCRFCHAR	4 Bytes	Scratch-file character, left-justified with trailing blanks (default is "-"; $SET SCRFCHAR=c)
11*	ICFBIT	Fullword	1 -> $SET IC=OFF (default is ON)
35*	SHFSEP	4 Bytes	Shared-file separator character, left-justified with trailing blanks (default is ":"; $SET SHFSEP=c)
39	DEVCHAR	4 Bytes	Device-name character, left-justified with trailing blanks (default is ">"; $SET DEVCHAR=c)
59*	SEQCOFF	Fullword	1 -> $SET SEQFCHK=OFF (default is ON)
75*	SETIOERR	Fullword	SETIOERR exit subroutine address
77*	ENDFILSW	Fullword	$SET ENDFILE={NEVER|SOURCE|ALWAYS} (0|1|2) (default SOURCE)
181*	TRIMBIT	Fullword	1 -> $SET TRIM=ON (default is ON)
375*	NEWFILAC	Variable	$SET NEWFILEACCESS={'string'|OFF} (default OFF) (from 0 to 255 characters)

System Information

228	TOFFSET	Dblword	Offset (microseconds times 4096) to be added to GMT to get local time
230	TIMEFDGE	Dblword	Value (microseconds times 4096) to be added to IBM time (as stored by a STCK instruction) to get time based on March 1, 1900
240	SYSOLOAD	Fullword	System overload indicators, right-justified with leading zeros[11]
252	UNATMODE	Fullword	1 -> System running in "unattended mode" (see also item 277)
277	NOMOUNTS	Fullword	1 -> No tape or floppy-disk mounts allowed (see also item 252)
294	TAPEQ	Fullword	1 -> Tape mount queuing is enabled
295	TAPEQLEN	Fullword	Length of current tape mount queue
334	TZONOFST	Fullword	Current time zone offset from GMT (minutes)
335	TZONNAME	8 Bytes	Current time zone name (characters)
377	MTSMODEL	5 Bytes	MTS model number (characters)
433	SERVER	Fullword	1 -> The job is a server program

Task Limits

12	LOCSW	Fullword	1 -> Local time estimate active
78	GLOBCPUT	Fullword	CPU time remaining in global time limit[5]. See Note(3).
80	GLOBPGS	Fullword	Global page estimate
82	GLOBPCH	Fullword	Global card estimate
84	GLOBPTM	Fullword	Global plot time estimate (seconds)
86	LOCCPUT	Fullword	CPU time remaining in local time limit[5]. See Note(3).
88	LOCPGS	Fullword	Local page estimate
90	LOCPCH	Fullword	Local card estimate
92	LOCPTM	Fullword	Local plot time estimate (seconds)
94	GLOBTTN	Fullword	Base for global time limit[5]. See Note (3).
96	LOCTTN	Fullword	Base for local time limit[5]. See Note (3).
253	LOCLIMIT	Fullword	Local time limit in effect[5]
392*	TIMLIMIT	Fullword	$SET TIME={n|OFF}[5]

Task Status

4	S8NBR	8 Bytes	Receipt number of job in characters, left-justified with trailing blanks (batch only)
10	BATCHMD	Fullword	Batch (1) or conversational (0) mode
13	SIGTMUT	18 Bytes	Signon time (Universal Time Units). See Note (4).
23*	OFFBIT	Fullword	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
55*	SIGSHORT	Fullword	$SIGNOFF {LONG\|SHORT\|$} (0|1|2) (default is LONG)
56	SOBCDTM	16 Bytes	Signon time and date in characters
68	SOELT	Dblword	Time of day at signon[6]
98	TASKNBR	Fullword	Task number
100	TASKTYPE	Fullword	Task type code[8]

104	HASPJOB	Fullword	1 -> Spooled batch job
172	CLSID	Fullword	Code for CLS currently in control[9]
174	PCLSID	Fullword	Code for CLS that called current CLS[9]
180	LSS	Fullword	1 -> If limited-service state active
226	INSIGFIL	Fullword	1 -> currently processing sigfile
228	TOFFSET	Dblword	Offset (microseconds times 4096) to be added to GMT to get local time
230	TIMEFDGE	Dblword	Value (microseconds times 4096) to be added to IBM time (as stored by a STCK instruction) to get time based on March 1, 1900
241	SIGCFLD	Variable	The comment field from the MTS $SIGNON command, without the enclosing primes (from 0 to 255 characters in length)
242	PRIO	Fullword	Priority of job[12]
334	TZONOFST	Fullword	Current time zone offset from GMT (minutes)
335	TZONNAME	8 Bytes	Current time zone name (characters, left-justified with trailing blanks)

Terminal Information

3*	PREFIXC	Fullword	Current prefix character, left-justified with trailing blanks, as set by the SETPFX subroutine or CUINFO item 257 (PFXSTR).
10	BATCHMD	Fullword	Batch (1) or conversational (0) mode
57*	PFXOFF	Fullword	1 -> $SET PFX=OFF (default is ON)
74	ANSBACK	24 Bytes	Answerback code (characters) (see also item 276)
257*	PFXSTR	Variable	Prefix string which normally appears at the beginning of terminal input and output lines (from 0 to 120 characters in length)
259	EXPRESS	Fullword	1 -> User is at an express terminal
260	TERMLOC	4 Bytes	1 -> 4-character terminal location code or binary zero, if unknown
276	ANSBACKL	Variable	Answerback code (characters) (see also item 74)
418	HOSTNAME	8 Bytes	Host name (characters)

---

[1]Half-pages*(1/300) seconds
[2]Page-seconds
[3]Minutes since Midnight, March 1, 1900
[4]Units of 1/300 second
[5]Timer units: 13 1/48 microseconds per unit
[6]Microseconds since Midnight, March 1, 1900
[7]Page-minutes
[8]Job type codes:

```
    0=Terminal
    1=Local batch (without batch monitor)
    2=Remote batch (without batch monitor)
    3=Normal batch (with batch monitor)
    4=*-File
    5=OPER
```

[9]CLS codes:

```
    0=MTS  (MTS command mode)
    1=USER (execution mode)
    2=EDIT (edit mode)
    3=SDS  (debug mode)
    4=CALC (calc mode)
    5=TST  (test CLS)
    6=NET  ($NET command)
    7=MNT  ($MOUNT command)
    8=PRMT ($PERMIT command)
    9=FSTA ($FILESTATUS command)
   10=SSTA (systemstatus mode)
   11=ACC  (accounting mode)
   12=NEW  (new CLS)
   13=NEW2 (new CLS)
   14=NEW3 (new CLS)
   15=LOG  ($LOG command)
   16=PMF  (program maintenance facility - under development)
   17=MESS ($MESSAGESYSTEM command)
   18=INFO ($INFO command - privileged)
   19=LIST ($LIST command)
   20=COPY ($COPY command)
   21=DEST ($DESTROY command)
   22=DUPL ($DUPLICATE command)
   23=EMPT ($EMPTY command)
   24=RENA ($RENAME command)
   25=TRUN ($TRUNCATE command)
   26=CREA ($CREATE command)
   27=DISP ($DISPLAY command)
   28=SET  ($SET command)
   29=FMNU ($FILEMENU command)
   30=MAKE ($MAKE command)
```

[10]Loader options (one byte)

```
    X'80'  1 -> Suppress pseudo-registers in map
    X'40'  1 -> Suppress predefined symbols in map
    X'20'  1 -> Print undefined symbols
    X'10'  1 -> Print undefined xrefs
    X'08'  1 -> Print all xrefs
```

X'04' 1 -> Print dotted lines
X'02' 1 -> Print map lines and entry point
X'01' 1 -> Print nonfatal errors

[11]System overload indicators (one byte)
X'80' 1 -> Processor
X'40' 1 -> Paging
X'20' 1 -> Disk I/O
X'10' 1 -> I/O activity
X'08' 1 -> Drum space

[12]Priority of job (one byte)
0=Low
1=Normal
2=High (currently not used)
3=Deferred
4=Minimum

[13]Asynchronous event control
Bit 31: 1 -> Stack attention interrupts
30: 1 -> Stack attention interrupts unless ATTNTRP exit is enabled
29: 1 -> Stack timer interrupts

Notes:

(1) The elapsed time virtual memory integral for this job is

STORUSEE+CURRSTOR*(time(2)*.3-STORELT)

and the CPU virtual memory integral for this job is

STORUSED+CURRSTOR*(time(1)*.3-STORCPUT)

where time(n) is the result of calling the TIME subroutine with key=n assuming no call has been made with key=0.

(2) The permanent disk and datacell space integrals for this ID are

60*CUMDISK+CURRDISK*(min-DISKTIME)

and

60*CUMCELL+CURRCELL*(min-CELLTIME)

and the scratch disk and datacell space integrals for this terminal session or batch job are

SCRDSUSE+SCRFDISK*(min-SCRDSKTM)

and

SCRCLUSE+SCRFCELL*(min-SCRCELTM)

where "min" is minutes since March 1, 1900 which is obtainable from the TIME and GRJLTM subroutines; the results are in page-minutes.

(3) GLOBTTN (or LOCTTN) is the base used for establishing the global (or local) time limit and is the total amount of CPU time used by the task up to that time. When the timer interrupt enforcing the global (or local) time limit is scheduled, GLOBCPUT (or LOCCPUT) is set to the CPU time available to the task before the interrupt will be triggered. GLOBCPUT and GLOBTTN (or LOCCPUT and LOCTTN) may be added to yield the CPU time point when the interrupt will occur. To obtain the time remaining in the global (or local) time limit, the current CPU time used by the task should be subtracted from the above sum. The current task CPU time may be obtained by using the TIME subroutine with key=9.

(4) The Universal (GMT) time is returned in the following format:

Bytes 0-7:	Universal time as Julian microseconds since March 1, 1900.
Bytes 8-9:	Time zone offset from Universal time (minutes).
Bytes 10-17:	Time zone name (8 characters, left-justified with trailing blanks, e.g., "EST ").

GUINFUPD

Subroutine Description

Purpose: To update certain items obtainable via the GUINFO subroutine.

Location: Resident System

Calling Sequence:

Assembly: CALL GUINFUPD

Return Codes:

- 0 Successful return.
- 4 Illegal signon ID.
- 8 Error return.

Description: The following items obtainable via the GUINFO subroutine are updated to the time of the call, excluding currently active jobs for this signon ID (including this job).

14	ACCTNO	36	NRDISKF
18	MAXDISK	38	NRSIGS
20	MAXTERM	40	NRBATCH
22	MAXMONY	42	CUMLINES
24	CURRDISK	44	CUMPAGES
26	CUMELTM	46	CUMPUNCH
28	CUMCPUTM	50	IDRNBR
29	CUMREAD	52	UNITCODE
30	CUMCORE	54	EXPTIME
32	CUMMONY	76	CUMDISK
106	MAXCELL	157	CUMPTSU
108	MAXPLOT	158	CUMPTLEN
110	LSTRESET	159	CUMPTSM
112	DISKTIME	160	BILLCLAS
114	CELLTIME	182	MAXSIG
116	CURRCELL	184	CURSIG
118	CUMCOREW	186	UNCHDISK
122	CUMPLOT	188	UNCHCELL
124	NRCELLF	190	MAXMNET
126	CUMCELL	192	CUMMNET
154	CUMMOUNT	194	MXMNETBT
156	CUMTDRVT	196	MXPLOTBT

232	CUMPLPAP	268	CUMPTRDT
234	CUMPLPEN	269	CUMPTPMT
246	ACCPRIV	270	CUMPTPDT
248	ACCCCPF	271	CUMLPMT
249	PSFATTN	272	CUMLPDT
250	ACCPUSE	296	PWSETBYC
267	CUMPTRMT		

324	CUMPPL	327	CUMPPS
325	CUMPPP	393	PRJPWCHG
326	CUMPPI		

GUSER

Subroutine Description

Purpose: To read an input record from the logical I/O unit GUSER.

Location: Resident System

Alt. Entry: GUSER#

Calling Sequences:

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

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

Parameters:

- reg is the location of the virtual memory region to which data is to be transmitted.
- len is the location of a halfword (INTEGER*2) integer in which will be placed the number of bytes read.
- mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.
- lnum is the location of a fullword integer giving the internal representation of the line number that is to be read or has been read by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.
- rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

- 0 Successful return.
- 4 End-of-file.
- >4 See the "I/O Subroutine Return Codes" description in this volume.

Description: All four of the above parameters in the calling sequence are required. The subroutine reads a record into the region specified by reg and puts the length of record (in bytes) into the location specified by len. If the mod parameter (or the FDname modifier) specifies the INDEXED bit, the lnum parameter must specify the line number to be read. Otherwise, the subroutine will put the line number of the record read into the location specified by lnum.

If the @MAXLEN FDname I/O modifier is specified, the len parameter is three halfwords which give the number of bytes actually read, the maximum number of bytes to be read, and the physical length of the record read. See the description of the @MAXLEN FDname I/O modifier in the section "I/O Modifiers" in this volume.

The default FDname for GUSER is *MSOURCE*.

Note that the contents of the input area reg may be changed even if the subroutine gives a nonzero return code.

There is a macro GUSER in the system macro library for generating the calling sequence to this subroutine. See the macro description for GUSER in MTS Volume 14, 360/370 Assemblers in MTS.

Examples: This example given in assembly language and FORTRAN calls GUSER specifying an input region of 20 fullwords. No modifier specification is made on the subroutine call.

Assembly:

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

               or

               GUSER REG,LEN   Subr. call using macro
```

FORTRAN:

```
             INTEGER*2 LEN
             INTEGER REG(20),LNUM
              ...
             CALL  GUSER(REG,LEN,0,LNUM,&30)
              ...
        30    ...
```

GUSERID

Subroutine Description

Purpose: To obtain the current 4-character signon ID.

Location: Resident System

Alt. Entry: GETID, GUSERIDS, GUSIDS

Calling Sequences:

Assembly: CALL GUSERID

CALL GUSERIDS,(ccid),VL

FORTRAN: CALL GUSIDS(ccid,&rc4)

A GR13 save area is not required for a call to this subroutine.

Parameters:

ccid is a region to store the 4-character signon ID.

Values Returned:

GR1 contains the 4-character signon ID.

Return Codes:

0 Successful return.
4 Invalid parameter or no VL bit specified.

Description: A call on the GUSERIDS or GUSIDS subroutines takes the S-type parameters and loads them into an R-type call on the GUSERID subroutine.

Example: FORTRAN: CALL GUSIDS(ID,&100)

The above example returns the signon ID.

IBSCH

Subroutine Description

Purpose: To perform a numeric or character binary search on an ordered FORTRAN array.

Location: *LIBRARY

Calling Sequence:

```
FORTRAN:  rslt = IBSCH(array,nelm,nrec,indx1,indx2,indx3,
                       type,order,key)
```

Parameters:

- array — is the array containing the data to be searched.
- nelm — is the number of numeric elements (all of the same type as key) composing each record (nelm is positive), or is the number of characters composing each record (nelm is negative and each record is |nelm| characters long).
- nrec — is the number of records in the entire array. If array is unidimensional, it must be dimensioned nelm*nrec; if it is two-dimensional, it must be dimensioned (nelm,nrec).
- indx1 — is the index in array of the first record to be searched.
- indx2 — is the index in array of the last record to be searched.
- indx3 — is the index of the numeric element within each record that is the search key (indx3 is positive), or is the index of character within each record that is the search key (indx3 is negative and specifies the |indx3|'th character).
- type — specifies the type of type, as follows:

type	type of key
-n	Character
0	INTEGER*2
1	INTEGER*4
2	Fullword character
3	REAL*4
4	REAL*8

For character searches, the search key is |n| characters long ($1 \le |n| \le 256$).

order specifies the order in which the data is sorted, as follows:

order	*order of data*
≥0	ascending
<0	descending

key is the key value for which the keys in *array* are to be searched.

Value Returned:

rslt is the the functional result of IBSCH to be interpreted as follows:

rslt	*meaning*
-1	invalid parameters
0	*key* was not found
1,2,..	record number in *array* in which *key* was found

Note: The parameters *nelm*, *nrec*, *indx1*, *indx2*, *indx3*, *type*, and *order* must be INTEGER*4.

Description: The IBSCH subroutine performs a numeric or character binary search on a FORTRAN array subject to the following constraints:

(1) All records must of equal length and each must be in one piece (not scattered through the array).
(2) The search will be performed on either all of or part of the array, in ascending or descending order, using a numeric or a character key. The records must have been previously sorted (or else a meaningless result will occur).
(3) A character-key search will use the standard EBCDIC collating sequence to locate the given key.
(4) The search key will be either all of or part of a record. If part of a record, the key must be in the same part of every record. Character keys of 1 to 256 characters and several kinds of numeric keys are recognized.
(5) Only one key field can be searched, for one key value, on each call to IBSCH.

IBSCH may be used with the output from the SORT2 subroutine, which means that unordered data may be readily searched by first sorting it on a given key using SORT2, then performing a binary search with that key value on the ordered data using IBSCH.

As stated above, the records to be searched must be in one piece. If the array in unidimensional, these records are simply stored sequentially from the first record to the last. If the array is two-dimensional, a FORTRAN program stores the array elements sequentially in column order. This means that the records to be searched must be arranged in the array as one record per column, with all the keys for a given key field starting in the same row.

For character searches, it must be noted that a character occupies one byte of storage, but FORTRAN arrays are dimensioned in terms of elements, not bytes. The following table gives the number of bytes per element for the FORTRAN data types likely to be used in searching.

FORTRAN <u>type</u>	Bytes per <u>element</u>
LOGICAL*1	1
INTEGER*2	2
LOGICAL*4	4
INTEGER*4	4
REAL*4	4
REAL*8	8

Where the key consists of four characters occupying one fullword of storage (e.g., one REAL*4 array element), a character search can be made up to one-fifth more efficient by using a numeric search with <u>type</u> having the value 2 to signify a fullword character key.

Example: FORTRAN:

```
      REAL*4 R(500)
      KEY=1562.33
      IRSLT=IBSCH(R,1,500,1,500,1,3,1,KEY)
```

The above example searches an entire array of 500 single-precision floating-point numbers, sorted in ascending order, for the value contained in the variable KEY (in this case, 1562.33).

FORTRAN:

```
      REAL*4 NAME(2,6),KEY(2)
      DATA NAME/'ANDE','RSON',
     +          'BROW','N   ',
     +          'HOLL','INGS',
     +          'JASP','ER  ',
     +          'ROWA','LING',
     +          'SCHM','IDT '/
      DATA KEY/'JASP','ER  '/
      IRSLT=IBSCH(NAME,2,6,1,6,1,2,1,KEY(1))
```

The above example searches a REAL*4 character array, sorted in ascending order, for the name "JASPER". IRSLT is 4 in this case. A numeric-style search is used.

FORTRAN:

```
      LOGICAL*1 NAME(8,6),KEY(8)
      DATA NAME/'A','N','D','E','R','S','O','N',
     +          'B','R','O','W','N',' ',' ',' ',
     +          'H','O','L','L','I','N','G','S',
     +          'J','A','S','P','E','R',' ',' ',
     +          'R','O','W','A','L','I','N','G',
     +          'S','C','H','M','I','D','T',' '/
      DATA KEY/'J','A','S','P','E','R',' ',' '/
      IRSLT=IBSCH(NAME,-8,6,1,6,-1,-8,1,KEY(1))
```

The above example searches a LOGICAL*1 character array, sorted in ascending order, for the name "JASPER". IRSLT is 4 in this case. A character-key search is used.

IOH

Subroutine Description

Purpose: IOH is an input/output conversion package that provides format-directed input and output for 360/370-assembler language programs and programs using the Plot Description System.

Location: *LIBRARY

Entry Points: IOH has the following entry points:

ROPEN, RCLOSE, POPEN, PCLOSE, PCOPEN, PCCLOSE, SEROPEN, SERCLOSE, GOPEN, GCLOSE, LOPEN, LCLOSE, SETFRVAR, SETIOHER, DROPIOER, GETIOHER, OWNCONVR, ACCEPT, and IOPMOD.

Description: For the complete description of IOH and its calling sequences, see the section "IOH" in MTS Volume 14, _360/370 Assemblers in MTS_.

JLGRDT, JLGRTM

Subroutine Description

Purpose: S-type (e.g., FORTRAN and PL/I) interfaces for JULGRGDT and JULGRGTM.

Location: *LIBRARY

Calling Sequences:

```
FORTRAN:  CALL JLGRDT(juldat,grgdat)

          REAL*8 JLGRDT
          date=JLGRDT(juldat,grgdat)

          CALL JLGRTM(jultim,grgtim)

          COMPLEX*16 JLGRTM
          time=JLGRTM(jultim,grgtim)

PL/I(F):  CALL PLCALL(JLGRDT,f2,PL1ADR(juldat),grgdat);

          DCL PLCALLD RETURNS(FLOAT(16));
          date=PLCALLD(JLGRDT,f2,PL1ADR(juldat),grgdat);

          CALL PLCALL(JLGRTM,f2,PL1ADR(jultim),grgtim);
```

Parameters:

<u>juldat</u> is a fullword (INTEGER*4 or FIXED BINARY(31)) containing the integer number of days starting with March 1, 1900 as "1".

<u>grgdat</u> is 8 bytes (REAL*8 or CHARACTER(8)) into which the Gregorian date in the character form "MM/DD/YY" is placed on return.

<u>jultim</u> is a fullword (INTEGER*4 or FIXED BINARY(31)) containing the integer number of minutes starting with March 1, 1900, at 00:01 as "1".

<u>grgtim</u> is 16 bytes (REAL*8(2) or CHARACTER(16)) into which the Gregorian date and time in the character form "MM/DD/YYhh:mm:00" is placed on return.

<u>f2</u> is a fullword (FIXED BINARY(31)) containing the integer 2.

Values Returned:

FR0 contains the Gregorian date in the character form "MM/DD/YY" for call on JLGRDT. This is assigned to date for FORTRAN and PL/I programs using the function-call format.

FR0 and FR2 contain the Gregorian date and time in the character form "MM/DD/YYhh:mm:00" for calls on JLGRTM. This is assigned to time for FORTRAN and PL/I programs using the function-call format.

Description: The Julian date or time is passed to JULGRGDT or JULGRGTM, respectively, and is converted to the corresponding Gregorian date or time in character form. The results are undefined for dates and times which are nonpositive or greater than 12/31/99.

Examples: FORTRAN:

```
REAL*8 DATE
CALL JLGRDT(25915,DATE)

REAL*8 DATE,JLGRDT,DUMMY
DATE=JLGRDT(25915,DUMMY)
```

The above two examples call JLGRDT to convert the Julian date 25915 into the corresponding Gregorian date February 11, 1971.

```
REAL JULIAN*4 TIME*8(2)
CALL JLGRTM(JULIAN,TIME)
```

The above example calls JLGRTM to convert the Julian date and time in the variable JULIAN into the corresponding Gregorian date and time.

PL/I(F):

```
CALL PLCALL(JLGRDT,F2,PL1ADR(JULIAN),DATE);
DECLARE JLGRDT ENTRY,
        F2 FIXED BINARY(31) INITIAL(2),
        JULIAN FIXED BINARY(31) INITIAL(25915),
        DATE CHARACTER(8);

UNSPEC(DATE)=UNSPEC(PLCALLD(JLGRDT,F2,
        PL1ADR(JULIAN),DUMMY));
DECLARE (DATE, DUMMY) CHARACTER(8),
        PLCALLD RETURNS(FLOAT(16)),
        JLGRDT ENTRY,
        F2 FIXED BINARY(31) INITIAL(2),
        JULIAN FIXED BINARY(31) INITIAL(25915);
```

The above two examples call JLGRDT to convert the Julian date 25915 into the corresponding Gregorian date February 11, 1971.

```
CALL PLCALL(JLGRTM,F2,PL1ADR(JULIAN),TIME);
DECLARE JLGRTM ENTRY, TIME CHARACTER(16),
        F2 FIXED BINARY(31) INITIAL(2),
        JULIAN FIXED BINARY(31);
```

The above example calls JLGRTM to convert the Julian date and time in the variable JULIAN into the corresponding Gregorian date and time.

JMSGTD, JTUGTD

Subroutine Description

Purpose: S-type (e.g., FORTRAN and PL/I) interface for JMSGTDR and JTUGTDR.

Location: *LIBRARY

Calling Sequences:

```
FORTRAN:  CALL JMSGTD(jms,grgtim)

          CALL JTUGTD(jtu,grgtim)

PL/I(F):  CALL PLCALL(JMSGTD,f2,jms,grgtim);

          CALL PLCALL(JTUGTD,f2,jtu,grgtim);
```

Parameters:

<u>jms</u> is an 8-byte integer (INTEGER*4(2) or BIT(64)) containing the integer number of microseconds starting with March 1, 1900.

<u>jtu</u> is an 8-byte integer (INTEGER*4(2) or BIT(64)) containing the integer number of timer units starting with March 1, 1900. A timer unit is 1/256 of 1/300 of a second (13 1/48 microseconds).

<u>grgtim</u> is 16 bytes (REAL*8(2) or CHARACTER(16)) into which the Gregorian time and date in the character form "hh:mm.ssMM-DD-YY" is placed on return.

<u>f2</u> is a fullword (FIXED BINARY(31)) containing the integer 2.

Description: The Julian time in microseconds or timer units is passed to JMSGTDR or JTUGTDR, respectively, and is converted to the corresponding Gregorian date and time in character form. The results are undefined for dates and times which are nonpositive or greater than 12/31/99.

Examples: FORTRAN:

```
      INTEGER*4 JULIAN(2)
      DATA JULIAN/Z000830D1,Z7477784F/
      REAL*8 TIME(2)
       ...
      CALL JMSGTD(JULIAN,TIME)
```

```
PL/I(F):  DECLARE JMSGTD ENTRY,
              F2 FIXED BINARY(31) INITIAL (2),
              TIME CHARACTER (16),
              JULIAN BIT(64) INITIAL
              ('00000000000010000011000011010001011101000
          11101110111100001001111'B);
          CALL PLCALL(JMSGTD,F2,JULIAN,TIME);
```

The above two examples call JMSGTD to convert the Julian time into the corresponding Gregorian time and date 17:59.33, March 21, 1973.

JMSGTDR, JTUGTDR

Subroutine Description

Purpose: To convert the Julian time in microseconds or timer units since March 1, 1900 to the corresponding Gregorian time and date hh:mm.ssMM/DD/YY.

Location: *LIBRARY

Calling Sequences:

```
Assembly: LM 0,1,julms
          CALL JMSGTDR

          LM 0,1,jultu
          CALL JTUGTDR
```

Parameters:

<u>julms</u> is two fullwords containing the 8-byte integer number of microseconds through the given date starting with March 1, 1900.

<u>jultu</u> is two fullwords containing the 8-byte integer number of timer units starting with March 1, 1900. A timer unit is 1/256 of 1/300 of a second (13 1/48 microseconds).

Value Returned:

GR0 through GR3 contain the Gregorian time and date in the character form "hh:mm.ssMM-DD-YY".

Description: The results are undefined for dates which are nonpositive or greater than 12/31/99.

See JMSGTD, JTUGTD for S-type (e.g., FORTRAN and PL/I) interfaces.

Example: Assembly:

```
               LM   0,1,JULMS
               CALL JMSGTDR
               STM  0,3,GREG
                .
                .
         JULMS DC   X'000830D17477784F'
         GREG  DS   CL16
```

The above example calls JMSGTDR to convert the Julian time in location JULMS to the corresponding Gregorian time and date 17:59.33, March 21, 1973.

JULGRGDT, JULGRGTM, JLGRSEC

Subroutine Description

Purpose: To convert the Julian date or time (based on March 1, 1900) to the corresponding Gregorian date (MM/DD/YY) or time (MM/DD/YYhh:mm:ss).

Location: Resident System

Calling Sequences:

Assembly:

```
L    1,juldat
CALL JULGRGDT

L    1,jultim
CALL JULGRGTM

L    1,julsec
CALL JLGRSEC
```

Parameters:

juldat is a fullword containing the integer number of days starting with March 1, 1900 as "1".
jultim is a fullword containing the integer number of minutes starting with March 1, 1900, at 00:01 as "1".
julsec is a fullword containing the integer number of seconds starting with March 1, 1900, at 00:00:01 as "1".

Values Returned:

GR0 and GR1 contain the Gregorian date in the character form "MM/DD/YY" for calls on JULGRGDT.

GR0 through GR3 contain the Gregorian date and time in the character form "MM/DD/YYhh:mm:00" for calls on JULGRGTM.

GR0 through GR3 contain the Gregorian date and time in the character form "MM/DD/YYhh:mm:ss" for calls on JLGRSEC.

Description: The results are undefined for dates which are nonpositive or greater than 12/31/99. For JLGRSEC, times greater than 03/19/68 03:14:07 require all 32 bits of the parameter in GR1.

See JLGRDT, JLGRTM for S-type (e.g., FORTRAN and PL/I) interfaces.

Examples: Assembly:

```
             L     1,JLDAT
             CALL  JULGRGDT
             STM   0,1,GRDAT
              .
              .
      JLDAT  DC    F'25915'
      GRDAT  DS    CL8
```

The above example calls JULGRGDT to convert the Julian date 25915 into the corresponding Gregorian date February 11, 1971.

```
             L     1,JLTIM
             CALL  JULGRGTM
             STM   0,3,GRTIM
              .
              .
      JLTIM  DC    F'37438110'
      GRTIM  DS    CL16
```

The above example calls JULGRGTM to convert the Julian date and time 37438110 into its corresponding Gregorian date and time May 6, 1971, 16:30:17.

KWSCAN

Subroutine Description

Purpose: To perform keyword processing on a character string. Keyword processing entails searching a character string for certain specified character strings of the form "keyword=value" (or the degenerate forms, "keyword" and "value") and performing an associated program action when a specified keyword string is found.

Location: Resident System

Calling Sequences:

Assembly: CALL KWSCAN,(len,lht,ext,text,rht,ltext,sws,
rvec,dlist,slist,sinfo)

Parameters:

- len — is the location of the halfword length of the table of valid keyword left-hand sides indicated by lht.
- lht — is the location of the table of valid keyword left-hand sides (see "Description" below for the form of its entries).
- ext — is the location of the execute table, a set of instructions selectively executed depending on the keyword that was found in the input string (see "Description" below for a discussion of its form and use).
- text — is the location of the character string to be processed for keywords.
- rht — is the location of the table of valid keyword right-hand sides (see "Description" below for the types and forms of its entries).
- ltext — is the location of the halfword length of the string referenced by text.
- sws — is the location of a fullword of bit flags that define the behavior of the keyword scanner. See "Subroutine Options" below for details.
- rvec — is the location of a 27-word return vector, or zero. It is optionally used to return error information from the subroutine. If rvec is zero, no error information is returned. See "Subroutine Options" below for the form of and control over the information returned.
- dlist — is the location of an optional set specifying

the characters to be considered as keyword expression delimiters. See "Subroutine Options" below for the specification of the set.

<u>slist</u> is the location of an optional set of character strings to be considered as separators of keyword expression left- and right-hand sides. See "Subroutine Options" below for the specification of the set.

<u>sinfo</u> is the location of an optional summary information buffer. See bit 13 of the <u>sws</u> parameter.

Return Codes:

0 Keywords successfully processed.
4 "CANCEL" response given in reply to prompt for replacement of incorrect input, or other error in keyword processing.

Description: The KWSCAN subroutine scans the given character string for valid keyword expressions as defined by the subroutine parameters. When a valid keyword expression is found, the calling program is given the "value", if any, of the expression, and the opportunity to perform processing pertinent to the keyword function.

Conceptually, every keyword expression has a left-hand side and a right-hand side, the left-hand side constituting the keyword portion of the expression, and the right-hand side defining the expression's "value". Physically, either, but not both, of these may be absent along with the associative character "=", yielding three possible keyword expression forms: "LHSide=RHSide", "LHSide", and "RHSide".

The left-hand side keyword and right-hand side values to be recognized in the input string are specified in the tables indicated by <u>lht</u> and <u>rht</u>. Whereas keyword right-hand sides can be any of a fixed number of different types, ranging from arbitrary strings to decimal numbers, left-hand sides, being keywords, can only be given character strings. The text of the left-hand sides, and their associations with right-hand sides, are specified in the left-hand table, pointed to by <u>lht</u>. The forms of the right-hand sides are specified in the right-hand table, indicated by <u>rht</u>.

Keyword expressions are scanned for as follows. The input string is searched from left to right for a substring bounded at the right and left extents by delimiter characters (the beginning and the end of a string are also considered delimiters). The substring text, up to the embedded "=" (or the entire substring if no "=" is present), is then compared to left-hand side text entries in the left-hand table. If no left-hand side match is found there, the substring is not considered a valid keyword expression and an error return is made. If an entry is found to match, the right-hand table is scanned beginning at a displacement specified in the left-hand table entry that matched the keyword expression's left-hand side. The text to the right of the "=" in the substring under consideration, the right-hand side, is then checked to see if it matches the right-hand side forms given by successive right-hand table entries. If it is of one of the given forms, the substring is considered a valid keyword expression, and a match takes place. Otherwise, the expression is not valid.

When a keyword expression is matched, the general registers are set up to contain information pertaining to the keyword expression (such as the keyword right-hand value). A single instruction in the table of instructions indicated by ext, specified by the sum of two displacements contained in the matching left- and right-hand table entries, is performed by an EX instruction. The calling program can thus perform an action associated with the given keyword, such as saving the value of the right-hand side. If more than one instruction is needed for the action, the subject of the EX instruction should be a BAL or BALR instruction to a pertinent internal subroutine. A return from this subroutine should be eventually made. If the return is made to the instruction specified by the contents of the link register, keyword processing will proceed normally (according to the options defined in the fullword indicated by sws). If a return is made to two bytes past the link register contents, the match to the keyword expression is rejected, and a scan for an alternate right-hand side match resumes after the right-hand table entry which matched previously. If the return is to 16 bytes past the contents of the link register, all keyword processing is aborted immediately and a return code of 4 is issued by the KWSCAN subroutine.

If text appears in the input string that does not match any of the defined keywords, various actions may be taken, depending on the subroutine options. One option is to generate an error message on *MSINK*, followed by a prompt, if the subroutine is not being used in batch mode, for corrective input from *MSOURCE*. If this option is selected, the prompted input does not replace or modify the contents of the original string in error, but is processed separately. Other options include spelling correction of the invalid text. See the section "Subroutine Options" below.

When the keyword input string contents are exhausted, or the keyword scan otherwise terminates, the subroutine returns with the return code set.

Format of Left-Hand Table Entries:

Left-hand table entries defining the keyword left-hand sides are variable-length entries. The format is:

1 or 2 bytes[1]	-	right-hand table index. This is the displacement into the right-hand table where the associated right-hand side entries for this left-hand side can be found.
1 or 2 bytes[1]	-	execute-table index. This is the partial displacement into the execute table where an instruction associated with a match to this left-hand side is located.
1 byte[2]	-	(optional) control code.
n bytes[2]	-	(optional) control code operands.
1 byte	-	count of number of characters in the left-hand side.
N characters	-	the text of the left-hand side keyword.

[1]The right-hand table index and execute-table index values are two bytes in length if bit 27 of the sws parameter is one. The number of characters which compose the left-hand side text may be zero, implying a null left-hand side (i.e., the degenerate form "RHSide").

[2]The left-hand table may contain optional left-hand table control codes followed by control-code operands (if applicable). Multiple control codes may be used in left-hand side entries. The control codes are distinguished from the following keyword text-length field by the initial bit being set to 1.

Control Code	Description
hex FE	Suppress spelling correction for the left-hand side entry.
hex FD	Explicit minimum initial substring length specified. The length is given in the byte following this control code. This control is effective even if bit 23 of the sws parameter is zero.

Right-Hand Side Type Codes:

The right-hand side types fall into two distinct classes: those which define the forms which a keyword right-hand side may take, and those affecting the scanning of the right- and left-hand tables for keyword matches (control codes). They are dealt with separately below.

Control Code	Description
hex FF	Terminate search of right-hand table. Forces scan for a keyword match to fail.
hex FE	Abort right-hand table search. Forces the keyword scanner to reject the match of the keyword's left-hand side, and to continue scanning for an alternate match to the left-hand side following the point in the left-hand table at which the previous left-hand side match was found.
hex FD	Process parenthesized right-hand sides. Causes the current keyword expression's right-hand side to be treated as a parenthesized list of right-hand sides if such a list appears (e.g., INFO=(SIZE,TYPE) would be processed as if INFO=SIZE,INFO=TYPE had been given).
hex FC	Separator filter. Used in conjunction with bits 20-21 of the _sws_ parameter (see "Subroutine Options" below) to provide a barrier to keyword expressions depending on the character string connecting the keyword expression's left- and right-hand sides. If the connecting string is not in the set defined by information following the type code, the expression is considered invalid at this point.
hex FB	Suppress spelling correction for the next right-hand side entry.
hex FA	Force uppercase conversion of right-hand side even if bits 25-26 of _sws_ are B'10'.

The remaining types follow.

Type Code	Description
1	Literal Characters. The right-hand side is matched against a specified character string.
2	FDname. The right-hand side is interpreted as an MTS FDname, or concatenation of FDnames, and an FDUB is acquired for it.
3	Characters. The right-hand side is taken as an arbitrary character string, possibly subject to minimum and maximum length restrictions.
4	MTS Line Number. The right-hand side is interpreted as an optionally signed decimal number of maximum 6 integral digits and 3

fractional digits followed by an optional scale factor, and then multiplied by 1000 to remove any fractional digits.

5 Hexadecimal Number. The right-hand side is interpreted as a hexadecimal number, maximum of 8 hex digits.

6 Initial Substring Literal. The right-hand text must begin with a specified string of characters.

7 No Right-Hand Side. No right-hand side may be given in the keyword expression (e.g., only the degenerate form "LHSide" is accepted).

8 Ignore Keyword. The entire keyword expression is ignored. No instructions in the execute table are performed.

9 Characters in Given Set. The characters constituting the keyword expression right-hand side must all be members a given set of characters.

10 Characters Except in Given Set. The characters constituting the keyword expression's right-hand side may not contain any of the characters in a given set.

11 Optionally Negated Characters. Same as the characters (3) type, but a preceding negating prefix (one of "-", "¬", "NO", or "N") is allowed. Different execute-table instructions may be performed, depending on whether the negating prefix was found.

12 Optionally Negated Literal. Same as the literal characters (1) type, with additional features of type 11.

13 Optionally Negated Initial Substring Literal. Same as initial substring literal (6) type, with additional features of type 11.

14 Delimited Character String. The right-hand side value is interpreted as a character string initiated and terminated by a string delimiter character in a set defined by information in the right-hand table entry. Doubled instances of the string delimiter are compressed into a single instance of

	the delimiter. A maximum and minimum length of the resultant string may be defined. The resultant string length must be less than 128 characters.
15	Integer Number. The right-hand side value may be an integer number consisting of an optional sign followed by at most 9 decimal digits, and possibly followed by a scale factor character.
16	Flagged Hexadecimal Number. The right-hand side value is interpreted as a hexadecimal number of 8 digits maximum, expressed in the form X'number'.
17	Floating-Point Number. The right-hand side value is interpreted as a FORTRAN-style long real number, optionally followed by a scale factor.
18	PAR Field. The right-hand side value is taken as the remainder of the input string.
20	Literal Substring. The right-hand side is compared against a specified string to determine whether the right-hand side represents an initial substring of it.
21	Optionally Negated Literal Substring. Same as the literal substring (20) with the additional features of type 11.

Formats of Right-Hand Table Entries:

Control Code	Format and Description
hex FF	1 byte X'FF'
hex FE	1 byte X'FE'
hex FD	1 byte X'FD'
hex FC	1 byte X'FC', 1 byte containing the number of bytes following (N), N bytes ordinal positions of the separators in the list passed as the _slist_ parameter, or implied by _sws_ bits 20 and 21 having the value 01 (see "Subroutine Options" below) with zero indicating no separator (a degenerate keyword expression). If the separator is not in the set described by the given N bytes, the keyword expression is considered

invalid.
hex FB 1 byte X'FB'

Noncontrol right-hand table entries are of the format:

1 byte - type code,
1 byte - execute table index,
1 byte - number of bytes following (N),
N bytes - variable information, dependent upon type code, described below.

Right-Hand Side Type Information:

Literal (1) The N characters of the literal string.

FDname (2) Either N=0, in which case any FDname is accepted, or N=1 and the letter N must follow, in which case no FDnames specifying explicit concatenation are matched.

Character (3) N is 0, 1, or 2. If N=0, any character string is accepted. If N=1, one byte of information is given containing the maximum permissible length of the character string. If N=2, two bytes of information should follow, respectively giving the minimum and maximum permissible lengths of the string.

MTS Line Number (4) N must be an integral multiple of 5. A series of N/5 operations are performed on the value of the number. The operations are specified by a 1-character operation code followed by a 4-byte unaligned integer operand associated with the operation code. The operations are applied in the order in which they appear.

The right-hand side value has already been multiplied by 1000 at the time of the first operation.

The operations are:

Opcode ">": the right-hand side value is compared to the operand value. If the right-hand side value is less, the right-hand side match fails.

Opcode "<": the right-hand side value is compared to the operand value. If the right-hand side value is greater, the right-hand side match fails.

Opcode "*": the right-hand side value multiplied by the operand value.

Opcode "/": the right-hand side value is divided by the operand value.

Any other opcode: the operation code character is interpreted as an optional scale factor, which, if present at the end of the right-hand side value, causes the value to be multiplied by the operand value.

Hex Number (5)	N should be zero.
Initial Substring Literal (6)	N characters constituting the text that must be an initial substring of the right-hand side text are given.
No Right-Hand Side (7)	N should be zero.
Ignore (8)	N should be zero.
Characters in Given Set (9)	2 bytes defining the minimum and maximum permissible lengths of the right-hand side text are given, followed by N-2 characters that constitute the set of which each character of the right-hand side must be a member.
Characters Except in Given Set (10)	2 bytes defining the minimum and maximum permissible lengths of the right-hand side text are given, followed by N-2 characters that may not be present in the right-hand side text.
Optionally Negated Characters (11)	N is either 1, 2, or 3. In all cases, a single byte giving the right-hand table execute-table index used in case a negating prefix is found, is given. If N=1, the character string may be of arbitrary length. If N=2, one further

byte containing the maximum permissible length of the character string must be present. If N=3, two further bytes containing, respectively, the minimum and maximum permissible lengths of the right-hand side string must be present. In all cases, the lengths do not include the negating prefix, if present.

Optionally Negated Literal (12)	N bytes of information follow, consisting of a 1-byte execute-table index used in case a negating prefix is found, followed by N-1 bytes of characters comprising the literal text of the right-hand side.
Optionally Negated Initial Substring Literal (13)	N bytes of information follow, consisting of a 1-byte execute-table index used in case a negating prefix is found, followed by N-1 bytes of characters constituting the text of the initial substring of the right-hand side text.
Delimited Character String (14)	The information contains 2 bytes defining the minimum and maximum permissible number of characters, excluding the string delimiter characters, in the string. Following this is a set of N-2 characters, any of which may delimit the character string. The following two characters, if present at the beginning of the delimiter list, have special meaning: O Optional delimiters. If no match is made for the following delimiters, return the right-hand side entry (up to the next zero-level delimiter) as-is. If used, O must appear first in the delimiter list. P The following delimiters are grouped in pairs, a left-side followed by a right-side delimiter.
Integer Number (15)	The information is identical to the information associated with the MTS Line Number (4) type, but the number is not multiplied by 1000 prior to application of the specified operations.

Flagged Hex Number (16)	N should be zero.
Floating-Point Number (17)	The information is similar to that for the MTS Line Number (4) type, differing in that the operand values are unaligned long floating-point numbers, and therefore the entries are 9 bytes in length. The right-hand side value is not multiplied by 1000.
PAR Field (18)	N should be zero.

Literal Table (19)	A 4-byte address of a table containing a list of literals (N must always be 4). The table is of the form:
	
	1-byte item width
	1-byte count of number of items
	Series of entries of specified width
	
	All items must be of the same length, left-justified with trailing blanks, e.g.,
	
	`DC   AL1(7,3)`
	`DC   CL7'NEW'`
	`DC   CL7'OLD'`
	`DC   CL7'CURRENT'`

Literal Substring (20)	A 1-byte number whose value defines the minimum length of the substring that must match the given text should be given. If this value is zero, no restriction on the substring length is enforced (note that this right-hand side type will never match a null substring). Following this byte, N-1 characters constituting the text of the string to be tested for substring containment are given.
Optionally Negated Literal Substring (21)	A 1-byte execute-table index used in the case when negating a prefix is encountered must be specified, followed by N-1 bytes formatted as the information following the literal substring type (20).

General Register Values When Execute Instruction is Performed:

Right-Hand Type	Register Contents
Literal (1)	GR1: Length-1 of the right-hand side string. GR2: Address of the first character of the string.
FDname (2)	GR2: FDUB pointer for the right-hand side FDname.
Characters (3)	As for type 1.
MTS Line Number (4)	GR2: Value of the number times 1000, and as altered by any operations in the matching right-hand table entry.
Hex number (5)	GR2: The hex number, right justified.
Initial Substring Literal (6)	As for type 1.
No Right-Hand Side (7)	No registers are set up.
Ignore (8)	No instruction is executed.
Characters in Given Set (9)	As for type 1.
Characters Except in Given Set (10)	As for type 1.
Optionally Negated Characters (11)	As for type 1, but any negating prefix is not indicated.
Optionally Negated Literal (12)	As for type 11.
Optionally Negated Initial Substring Literal (13)	As for type 11.
Delimited Character String (14)	As for type 1, except that the string delimiting characters are not indicated.
Integer Number (15)	GR2: Value of the number as altered by the right-hand table operations.

Flagged Hex Number (16)	GR2:	Value of the hex number, right-justified.
Floating-Point Number (17)	FR0:	Value of the right-hand side as altered by the right-hand table operations.
PAR Field (18)		As for type 1.
Literal Substring (20)		As for type 1.
Optionally Negated Literal Substring (21)		As for type 14.

In addition, GR3 always contains a logical index into the left-hand table to indicate which entry matched the keyword expression's left-hand side. The index is in the form of 4*(ordinal position - 1) of the entry in the

left-hand table. GR15 contains the address of the executed instruction in the execute table.

The remaining registers are set to their values at the time of the subroutine call (see "Subroutine Options," bits 20-22, for possible exceptions to this). Any registers in the GR1-GR2 range unused by a right-hand side type are not restored to their values at the time of the subroutine call.

Subroutine Options:

The bits of the fullword indicated by the sws parameter define the subroutine behavior options. The bits and their associated effects are given below.

Bit #	Value	Hex Value	Effect
11	1	X'00100000'	On return from the instruction or subroutine executed for a given matched keyword, KWSCAN checks GR0 for the following control bits. 31: Do not print error message (if any). 30: Do not print error prompting message (if any). GR0 is initialized to zero by KWSCAN before the execute instruction is executed.
12	1	X'00080000'	On return, KWSCAN will provide a scanned-keyword table in the buffer addressed by word 6 of the sinfo buffer. KWSCAN allocates this buffer. This bit is valid only if bit 13 is set. The buffer begins with a fullword giving the length (in bytes) of the keyword information followed by a fullword giving the number of keywords scanned. The format of the entries is as follows: HW 1: Entry length (in bytes), including this halfword. HW 2: Matched LHS index. HW 3: Displacement (in bytes) to LHS text. HW 4: LHS-text length (in bytes). HW 5: Displacement to separator text (e.g., "=") HW 6: Separator-text length HW 7: Matched RHS index. HW 8: Displacement to RHS text.

			HW 9: RHS-text length. HW 10-end: Text area.
13	1	X'00040000'	On return, KWSCAN will provide summary information in the buffer addressed by sinfo. The format of the information is as follows: Word 1: Length (in bytes) of this buffer. This is set by the calling program. Word 2: Length (in bytes) of the information returned. This is set by KWSCAN. Word 3: Total number of keywords processed. Word 4: Number of keywords successfully processed. Word 5: Number of characters processed. Word 6: Address of scanned keyword table if bit 12 is one; otherwise, zero. The table is allocated by KWSCAN but must be released by the calling program via FREESPAC.
	0		No summary information is returned.
14	1	X'00020000'	Upon entry, KWSCAN saves the previous attention-interrupt exit (if any) and sets its own local exit. Then, if an attention interrupt occurs during KWSCAN processing, KWSCAN immediately returns with the return code set to 4 and the rvec error code set to 4. The original attention-interrupt exit is restored upon return from KWSCAN.
	0		No attention-interrupting processing is performed by KWSCAN.
15	1	X'00010000'	Rather than leaving the pertinent right-hand side values in the general registers and executing a single instruction in the execute table, the ext parameter is interpreted as the address of a subroutine which is passed the register contents as parameters. The subroutine should obey OS type I (S) calling conventions. The parameters passed consist of: 1 word - sum of left- and right-hand table execute indices, 1 word - GR1 contents, 1 word - either contents of FR0 if its value is set as a result of a keyword match, or the contents of GR2 if it is not an address, or an

			array containing the information indicated by GR2 if it is, 1 word - GR3 value, 1 word - GR4 value (see bit 22, below), 1 word - GR5 value (see bits 20-21, below). A return code of 0 from this subroutine will cause the keyword match to be accepted; 4 will cause the match to be rejected; 8 will cause the scan for keywords to be aborted.
16-17	11	X'0000C000'	Spelling correction of left- and right-hand sides is performed (see the description of the SPELLCHK subroutine in this volume). Verification of the correction is requested if the subroutine is being invoked in conversational mode. If in batch mode, the correction is never performed.
	01	X'00004000'	Spelling correction is performed as above, but no verification is requested, only a warning message is issued.
	00		No spelling correction is attempted.
18	1	X'00002000'	The return vector indicated by the <u>rvec</u> parameter is formatted in the following manner: 1 word - error code, listed below, 26 words - variable information, dependent upon error code:

<u>Code</u>	<u>Significance and Information Returned</u>
1	"CANCEL" given in response to prompt for corrective input. No further information is returned.
2	Invalid keyword expression. Information returned: 1 word - address of first char. in invalid expression, 1 word - length of bad expression, 1 word - length of error comment pertaining to bad expression, 23 words - text of error comment.
3	Keyword processing aborted by execute code return. No further information returned.
4	Keyword processing aborted by an attention interrupt (only if

			bit 14 of sws is one). No further information returned. 10 Invalid right-hand side type in right-hand table. The address of the invalid entry is returned. 11 Invalid format of right-hand table entry. The address of the invalidly formatted entry is returned. 12 Invalid format of separator list. The address of the invalidly formatted entry is returned. 30 Internal error. 31 Internal error.
	0		The return vector indicated by the rvec parameter is formatted in the following manner: 1 word - address of invalid keyword expression, 1 word - length of error comment, 25 words - text of error comment. This format is only used if an erroneous keyword expression is encountered. In all other cases, no information is returned.
19	1	X'00001000'	Keyword expression left-hand sides may be parenthesized (e.g., keyword expressions of the form (EXP1,EXP2,...,EXPN)=value are processed as being equivalent to EXP1=value,EXP2=value,..., EXPN=value).
	0		Keyword expression left-hand sides are not processed specially if parenthesized.
20-21	11	X'00000C00'	The slist parameter indicates a special set of strings which separate keyword expression left- and right-hand sides, in lieu of the standard "=" (e.g., "<-" could be defined as a separator, making expressions "LHSide<-value" valid). The format of the slist set is: 1 byte - number of separators to be defined, (1 byte - length of separator, N bytes - text of separator) repeated for each separator. If this option is selected, at the time the executed instruction is performed, GR5 contains an indicator of which separator was

			found in the keyword expression, in the form of 4*(separator's ordinal position in the list) with 0 indicating that no separator was found (i.e., a degenerate keyword expression).
	01	X'00000400'	The <u>slist</u> parameter need not be specified, but a relational set of separators are used as if the <u>slist</u> parameter had specified ">=", "<=", "Ë=" or "¬=", ">", "<", "=" in the presented order. GR5 is also set up as described above.
	00		Only "=" is a valid separator character.
22	1	X'00000200'	The <u>dlist</u> parameter indicates a set of single characters to be considered as delimiting characters in keyword expressions. Additionally, a context is defined with each character, specifying a context in which the character is to be considered a delimiter. The format of the set is: 1 byte - number of delimiters to be defined (1 byte - delimiter character, 1 byte - context: 0 for balanced parenthesis context, 1 for all contexts), repeated for each delimiter defined in the set. If this option is selected, at the time the executed instruction is performed, GR4 contains the address of the right side delimiter character in the keyword expression.
	0		The only valid delimiters are the blank in all contexts, and the comma when not nested inside parentheses.
23	1	X'00000100'	Keyword left-hand sides may be given as initial substrings of the left-hand side texts defined in the left-hand table.
	0		Keyword left-hand sides must be presented exactly as in the left-hand table.
24	1	X'00000080'	The address given by the <u>text</u> parameter will be updated to indicate the delimiter at the end of the last keyword processed.
	0		<u>text</u> is not updated.
25-26	10	X'00000040'	Convert all keyword input to uppercase, including prompt input. Translation to uppercase and subsequent processing is per-

			formed upon a copy of the input text, not on the input text itself. However, whenever a character value is returned for a matched right-hand side entry, it is returned in its original, unconverted form.
	01	X'00000020'	Same as 10 except that alphabetic characters are returned converted to uppercase.
	00		Leave all input as is.
27	1	X'00000010'	In the left-hand table, the right-hand table and execute table indices occupy 2 bytes.
	0		The above entries occupy 1 byte.
28	1	X'00000008'	Return to the calling program on the first invalid keyword expression encountered.
29	1	X'00000004'	Prompt user for corrections if invalid expressions are found.
	0		Do not prompt user for correction.
30	1	X'00000002'	Print error comments on *MSINK*.
	0		Do not print error comments, return them in the rvec return vector.
31	1	X'00000001'	Process all keyword expressions until the input string is exhausted.
	0		Process a single keyword expression only.

The remaining bits should be zero.

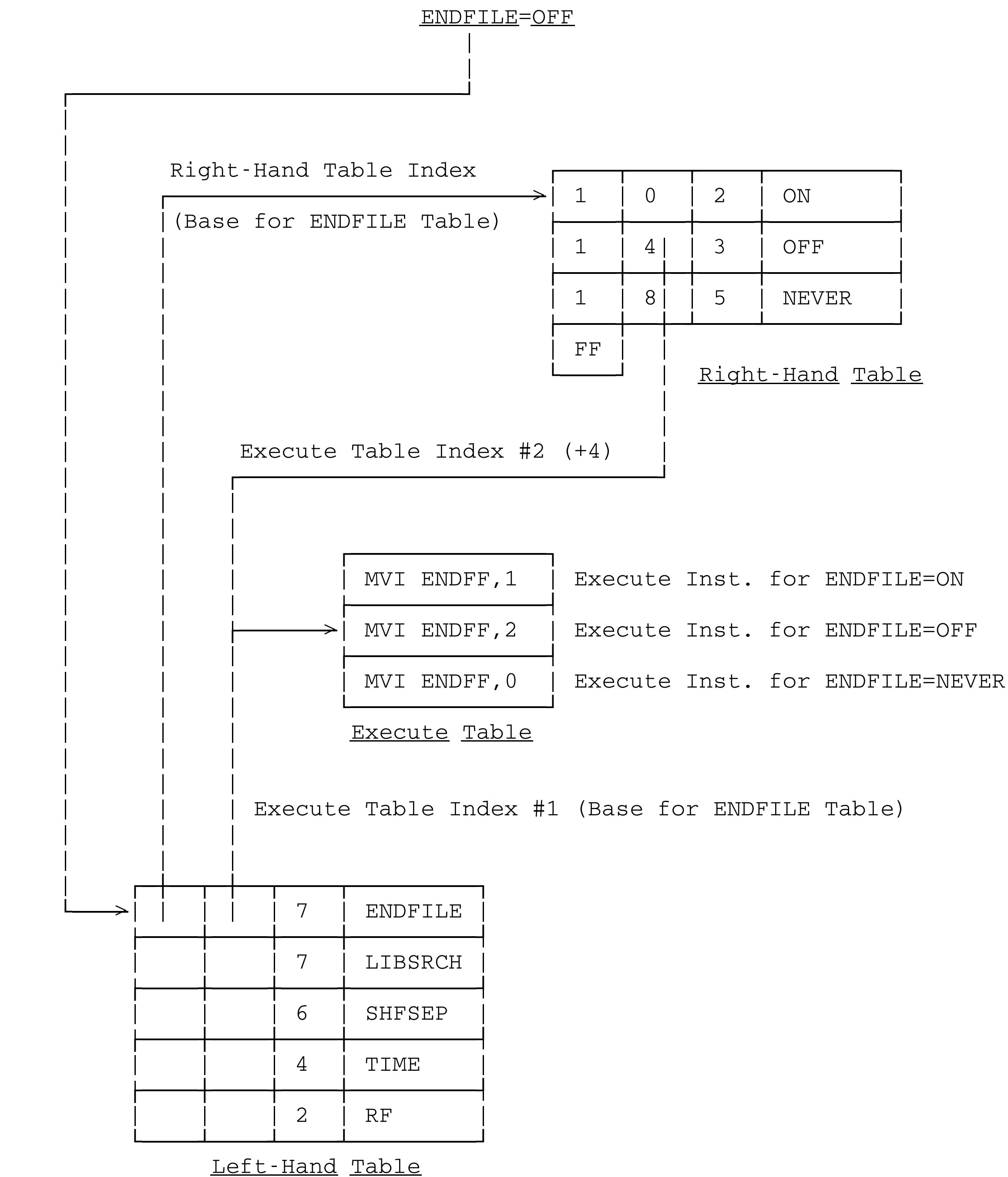
Examples: A series of examples are given in increasing order of complexity. The KWSCAN macros (KWLHT, KWRHT, and KWSET) described in MTS Volume 14, 360/370 Assemblers in MTS, should be used to set up the KWSCAN tables. Each example is presented both with and without the use of the KWSCAN macros.

It is possible to call KWSCAN directly from FORTRAN programs. If bit 15 in sws is set, KWSCAN will call a subroutine when it matches a keyword, instead of trying to execute some machine instructions directly. However, setting up the LHS and RHS tables in FORTRAN is very tedious and error prone. Several unsupported (UNSP) programs exist which can provide some help setting up these tables for FORTRAN. Another possible approach is to use the KWSCAN assembly macros, mentioned above, to set up the keyword tables separately. The Computing Center counselors should be contacted for further assistance in using KWSCAN from FORTRAN programs.

The first example mimics the processing of some of the options of the MTS $SET command, namely:

```
      ENDFILE=ON, ENDFILE=OFF, ENDFILE=NEVER
      LIBSRCH=OFF, LIBSRCH=FDname
      SHFSEP=c
      TIME=xxxx, TIME=xxxxS, TIME=xxxxM
      RF=<hex number>, RF=GRxx

          ... read into location STR
          CALL  KWSCAN,(LHTL,LHT,EXT,STR,RHT,STRL,SWS,0)
          ... process the keywords

*
*  Since SWS does not select the options requiring the
*  DLIST and SLIST parameters, they need not be given.
*

LHT      EQU   *
         DC    AL1(ENDF-RHT,ENDFE-EXT,7),C'ENDFILE'
         DC    AL1(LIBS-RHT,LIBSE-EXT,7),C'LIBSRCH'
         DC    AL1(SHFS-RHT,SHFSE-EXT,6),C'SHFSEP'
         DC    AL1(TIME-RHT,TIMEE-EXT,4),C'TIME'
         DC    AL1(RF-RHT,RFE-EXT,2),C'RF'
RHT      EQU   *
ENDF     DC    AL1(1,0,2),C'ON'   ENDFILE=ON
         DC    AL1(1,4,3),C'OFF'  ENDFILE=OFF
         DC    AL1(1,8,5),C'NEVER' ENDFILE=NEVER
         DC    X'FF'
LIBS     DC    AL1(1,0,3),C'OFF'  LIBSRCH=OFF
         DC    AL1(2,6,1),C'N'    LIBSRCH=<FDname>
         DC    X'FF'
SHFS     DC    AL1(3,0,2,1,1)     SHFSEP=c
         DC    X'FF'
TIME     DC    AL1(4,0,25)
         DC    C'>',FL4'0'        Make sure it's >0
         DC    C'M',FL4'60'       TIME=xxxM
         DC    C'S',FL4'1'        TIME=xxxS
         DC    C'*',FL4'768'      Convert to timer units
         DC    C'/',FL4'10'
         DC    X'FF'
RF       DC    AL1(5,0,0)         RF=xxxxxxxx
         DC    AL1(6,4,2),C'GR'   RF=GRxx
         DC    X'FF'
LHTL     DC    Y(RHT-LHT)

EXT      EQU   *
ENDFE    MVI   ENDFF,1            Set ENDFILE type code
         MVI   ENDFF,2
         MVI   ENDFF,0
LIBSE    XC    FDUB,FDUB          Zero FDUB signifies OFF
         ST    GR2,FDUB           Save fdub
SHFSE    MVC   SHFSEP(1),0(GR2)   Save new SHFSEP char
TIMEE    ST    GR2,TIMEVAL        Save TIME value
RFE      ST    GR2,RFVAL          Save hex value
         BAL   GR15,*+4           Make this a subroutine
```

```
*                                   for GRxx case
         CH    GR1,=H'1'
         BNH   2(,GR15)             -> no xx piece
         CH    GR1,=H'3'
         BH    2(,GR15)             -> more than just xx
*
*  Now can process value (much omitted here)
*
         BR    GR15

SWS      DC    XL4'0000C027'        Correct spelling, print,
*                                   prompt, multiple
*                                   keywords, uppercase
ENDFF    DS    X
SHFSEP   DS    C
STR      DS    CL80
STRL     DS    H
FDUB     DS    A
TIMEVAL  DS    F
RFVAL    DS    A
```



The diagram above illustrates the resultant processing for ENDFILE=OFF.

The above example is repeated below using the KWSCAN macros.

```
     ENDFILE=ON, ENDFILE=OFF, ENDFILE=NEVER
     LIBSRCH=OFF, LIBSRCH=FDname
     SHFSEP=c
     TIME=xxxx, TIME=xxxxS, TIME=xxxxM
     RF=<hex number>, RF=GRxx

         ... read into location STR
         CALL KWSCAN,(LHTL,LHT,EXT,STR,RHT,STRL,SWS,0)
         ... process the keywords

*
*    Since SWS does not select the options requiring the
*    DLIST and SLIST parameters, they need not be given.
*

*
*  The keyword scanner tables.
*
         KWSET RHTABLE=RHT,EXTABLE=EXT

LHTL     DC    Y(RHT-LHT)          (HW length of Left-hand table)

*
*  Left-hand table.
*
LHT      KWLHT ENDF,ENDFE,'ENDFILE'
         KWLHT LIBS,LIBSE,'LIBSRCH'
         KWLHT SHFS,SHFSE,'SHFSEP'
         KWLHT TIME,TIMEE,'TIME'
         KWLHT RF,RFE,'RF'

*
*  Right-hand Table.
*
RHT      KWSET EXTABLE=ENDFE
ENDF     KWRHT LIT,ENDFE,'ON'     Endfile = on
         KWRHT LIT,ENDFE2,'OFF'           = off
         KWRHT LIT,ENDFE3,'NEVER'         = never
         KWRHT END

         KWSET EXTABLE=LIBSE
LIBS     KWRHT LIT,LIBSE,'OFF'    Libsrch = off
         KWRHT FDNAME,LIBSE2,N            = FDname
         KWRHT END

         KWSET EXTABLE=SHFSE
SHFS     KWRHT CHARS,SHFSE,1,1    Shfsep = c
         KWRHT END

         KWSET EXTABLE=TIMEE      Time = xxxx | xxxxS | xxxxM
TIME     KWRHT INTEGER,TIMEE,(>,0),(M,60),(S,1)
```

```
         KWRHT END

         KWSET EXTABLE=RFE
RF       KWRHT HEX,RFE             RF = xxxxxxxx
         KWRHT SUBSTR,RFE2,'GR'       = GRxx
         KWRHT END

*
*  The executed code.
*
EXT      DS    0H

ENDFE    MVI   ENDFF,1             ENDFILE=ON
ENDFE2   MVI   ENDFF,2             ENDFILE=OFF
ENDFE3   MVI   ENDFF,0             ENDFILE=NEVER

LIBSE    XC    FDUB,FDUB           LIBSRCH=OFF (set FDUB to zero)
LIBSE2   ST    GR2,FDUB            LIBSRCH=FDname

SHFSE    MVC   SHFSEP(1),0(GR2)    SHFSEP=c

TIMEE    ST    GR2,TIMEVAL         TIME=xxxx | xxxxS | xxxxM

RFE      ST    GR2,RFVAL           RF=xxxxxxxx
RFE2     BAL   GR15,*+4            RF=GRxx (make a "subroutine")
         CH    GR1,=H'1'           (GR1 = RHS length - 1)
         BNH   2(,GR15)            Reject RHS.  No xx piece
         CH    GR1,=H'3'
         BH    2(,GR15)            Reject RHS.  Too long.

         ...process the value (much omitted here)...

         BR    GR15                Accept the RHS, return from RFE2.

         ...the rest is the same as before...
```

The second example shows the MTS $FILESTATUS command. It processes:

NAME=filename, filename
HEADING=ON, HEADING=OFF, HEAD, NOHEAD
OUTFORM=COL..., OUTFORM=KEY..., OUTFORM=LABEL...,
OUTFORM=PACK..., COL..., KEY..., LABEL..., PACK...
SIZE>=x, SIZE<=x, SIZE=x, SIZE<x, SIZE>x,
SIZE>=xP, SIZE<=xP, SIZE=xP, SIZE<xP, SIZE>xP

(This is a small subset of the parameters of the $FILESTATUS command).

```
         MVI   NAMEF,0            Initialize flag
TRYAGAIN CALL  KWSCAN,(LHTL,LHT,EXT,STR,RHT,STRL,SWS,RVEC)
         LTR   GR15,GR15
         BZ    OK                -> All ok
         CLC   =F'1',RVEC
         BE    ABORT             -> User said to CANCEL it
         CLC   =F'3',RVEC
         BNE   VERYBAD           -> Unexpected return code
         SERCOM 'TRY AGAIN.'
         B     TRYAGAIN          -> Sic

LHTL     DC    Y(RHT-LHT)        Length of left-hand table

LHT      EQU   *
         DC    AL1(JUNK-RHT,0,7),C'OUTFORM'
         DC    AL1(HEAD-RHT,HEADE-EXT,7),C'HEADING'
         DC    AL1(NAME-RHT,NAMEE-EXT,4),C'NAME'
         DC    AL1(SIZE-RHT,SIZEE-EXT,4),C'SIZE'
         DC    AL1(JUNK-RHT,0,0) Null left-hand side
RHT      EQU   *
HEAD     DC    X'FC',AL1(1,6)    Only let through "="
         DC    AL1(1,0,2),C'ON'  HEADING=ON
         DC    AL1(1,4,3),C'OFF  HEADING=OFF
         DC    X'FF'
SIZE     DC    X'FC',AL1(5,1,2,4,5,6) Don't let null left-
*                                hand sides or SIZE¬=xxx
*                                through here
         DC    AL1(4,0,5)        SIZE(>=,<=,>,<,=)xxxP
         DC    C'P',FL4'1'
         DC    X'FF'
NAME     DC    X'FC',AL1(1,6)    Only let through "="
         DC    AL1(3,0,2,1,17)   NAME=<1 to 17 characters>
         DC    X'FF'
JUNK     EQU   *
OUTF     DC    X'FC',AL1(2,0,6)  Only let through "=" and
*                                degenerates
         DC    AL1(6,OUTFE-EXT,3),C'COL' OUTFORM=COL
*                                        or COL
         DC    AL1(6,OUTFE-EXT+4,3),C'KEY' OUTFORM=KEY
*                                          or KEY
```

```
         DC    AL1(6,OUTFE-EXT+8,5),C'LABEL' OUTFORM=LABEL
*                                             or LABEL
         DC    AL1(6,OUTFE-EXT+12,4),C'PACK' OUTFORM=PACK
*                                             or PACK
         DC    X'FC',AL1(1,0)     Only let null left-hand
*                                 side through
         DC    AL1(12,HEADE-EXT,5,HEADE-EXT+4),C'HEAD'
*                                 HEAD or NOHEAD
         DC    AL1(3,NAMEE-EXT,2,1,17) <filename>
         DC    X'FF'

EXT      EQU   *
HEADE    MVI   HEADF,1            Header
         MVI   HEADF,0            No header
NAMEE    BAL   GR15,*+4           Make this a subroutine
         TM    NAMEF,1            Already have a name?
         BO    16(,GR15)          -> Yup, user blew it
         OI    NAMEF,1            Remember name was saved
         EX    GR1,FILEMVC        Save name
         BR    GR15               -> To KWSCAN
FILEMVC  MVC   FILENAME(0),0(GR2)
SIZEE    BAL   GR15,*+4           Make this a subroutine
         STC   GR5,RELATION       Save relational character
         ST    GR2,SIZEVAL        Save size value
         BR    GR15               -> To KWSCAN
OUTFE    MVI   FORMF,0            Select heading format
         MVI   FORMF,1
         MVI   FORMF,2
         MVI   FORMF,3

HEADF    DS    X
NAMEF    DS    X
FILENAME DS    CL17
RELATION DS    X
SIZEVAL  DS    F
FORMF    DS    X
STR      DC    CL80'OUTFORM=COL,JUNK,SIZE>5P,NOHEAD'
STRL     DC    H'80'
SWS      DC    X'0000E427'        Correct spelling, RVEC
*                                 format, relational
*                                 separators, uppercase,
*                                 print, prompt, multiple
*                                 keywords
RVEC     DS    27F
```

The above example is repeated below using the KWSCAN macros.

```
          MVI     NAMEF,0             Initialize flag
TRYAGAIN  CALL    KWSCAN,(LHTL,LHT,EXT,STR,RHT,STRL,SWS,RVEC)
          LTR     15,15
          BZ      OK                  -> All OK.
          CLC     =F'1',RVEC
          BE      ABORT               -> User said to CANCEL it.
          CLC     =F'3',RVEC
          BNE     VERYBAD             -> Unexpected return code
          SERCOM  ' Try again.'
          B       TRYAGAIN            -> Sic

          KWSET   RHTABLE=RHT
LHTL      DC      Y(RHT-LHT)          Length of left-hand table

LHT       KWLHT   JUNK,0,'OUTFORM'
          KWLHT   HEAD,HEADE-EXT,'HEADING'
          KWLHT   NAME,NAMEE-EXT,'NAME'
          KWLHT   SIZE,SIZEE-EXT,'SIZE'
          KWLHT   JUNK,0              Null left-hand side

RHT       EQU     *
HEAD      KWRHT   FILTER,(6)          Only let through "="
          KWRHT   LIT,0,'ON'          HEADING=ON
          KWRHT   LIT,4,'OFF'         HEADING=OFF
          KWRHT   END
SIZE      KWRHT   FILTER,(1,2,4,5,6) Don't let null left-hand
*                                     sides or SIZE¬=xx through
*                                     here
          KWRHT   LINENR,0,(P,1)      SIZE (>=,<=,>,<,=)xxxP
          KWRHT   END
NAME      KWRHT   FILTER,(6)          Only let through "="
          KWRHT   CHARS,0,1,17        NAME=<1 TO 17 characters>
          KWRHT   END
JUNK      KWRHT   FILTER,(0,6)        Only let through "=" and
*                                     degenerates
          KWRHT   SUBSTR,OUTFE-EXT,'COL' OUTFORM=COL or COL
          KWRHT   SUBSTR,OUTFE-EXT+4,'KEY' OUTFORM=KEY or KEY
          KWRHT   SUBSTR,OUTFE-EXT+8,'LABEL' OUTFORM=LABEL or
*                                     LABEL
          KWRHT   SUBSTR,OUTFE-EXT+12,'PACK' OUTFORM=PACK or PACK
          KWRHT   FILTER,(0)          Only let null left-hand
*                                     side through
          KWRHT   NEGLIT,(HEADE-EXT,HEADE-EXT+4),'HEAD'
*                                     HEAD or NOHEAD
          KWRHT   CHARS,NAMEE-EXT,1,17 <filename>
          KWRHT   END

EXT       DS      0H
HEADE     MVI     HEADF,1             Header
          MVI     HEADF,0             No header
NAMEE     BAL     15,*+4              Make this a subroutine
```

```
          TM      NAMEF,1             Already have a name?
          BO      16(,15)             -> Yup, user blew it
          OI      NAMEF,1             Remember name was saved
          EX      1,FILEMVC           Save name
          BR      15                  -> To KWSCAN
FILEMVC   MVC     FILENAME(0),0(2)
SIZEE     BAL     15,*+4              Make this a subroutine
          STC     5,RELATION          Save relational character
          ST      2,SIZEVAL           Save size value
          BR      15                  -> To KWSCAN
OUTFE     MVI     FORMF,0             Select heading format
          MVI     FORMF,1
          MVI     FORMF,2
          MVI     FORMF,3

HEADF     DS      X
NAMEF     DS      X
FILENAME  DS      CL17
RELATION  DS      X
SIZEVAL   DS      F
FORMF     DS      X
STR       DC      CL80'OUTFORM=COL,JUNK,SIZE>5P,NOHEAD'
STRL      DC      H'80'
SWS       DC      X'0000E427'         Correct spelling, RVEC
*                                     format, relational
*                                     separators, uppercase,
*                                     print, prompt, multiple
*                                     keywords
RVEC      DS      27F
```

LETGO

Subroutine Description

Purpose: To periodically unlock and then relock a file.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL LETGO,(unit,howlck,delay)

FORTRAN: index=LETGO(unit,howlck,delay)

Parameters:

- unit — is the location of either
  - (a) a fullword-integer FDUB-pointer (as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99),
  - (c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS), or
  - (d) a fullword index value (as returned by a previous call to LETGO).
- howlck — is the location of a fullword integer indicating how the file is to be relocked each time after it has been unlocked (see the description of the second argument for the subroutine LOCK).
- delay — is the location of a fullword-integer number of microseconds (elapsed time) after which the file will be momentarily unlocked and then relocked.

Value Returned:

- index — is a fullword value which can be used as the unit parameter on a subsequent call to LETGO to stop the unlocking and relocking of the file. For assembly language programs, this value is returned in GR0.

Return Codes:

- 0 Successful return.
- 4 unit (first argument) is not valid for a file, or howlck or delay are not addressable.
- 8 Timer interrupt could not be set up (nonzero return code from the subroutine SETIME).

Description: This subroutine will periodically unlock the specified file and then immediately attempt to relock it. If the file is not locked by another FDUB within the same job, the MTS shared-file system first will allow any other jobs, which are currently waiting, to access the file. This mechanism provides a convenient method whereby a job, which expects to be reading a shared-file for an extended period, can automatically have the file unlocked periodically, thereby permitting other jobs to <u>write</u> into the same file. Note that this procedure is not necessary if all of the jobs accessing the file are only reading it, since several jobs may simultaneously read the same file, i.e., several jobs may simultaneously have the file locked for reading.

Since this subroutine uses the system timer interrupt subroutines (SETIME and TIMNTRP) which will not interrupt a pending input/output operation, the file will not be periodically unlocked <u>during</u> an I/O operation. If a timer interrupt becomes pending during an I/O operation, the file will be unlocked and relocked upon completion of the operation. Thus, the file will <u>not</u> be periodically unlocked, for example, during the time a program is waiting for input from a terminal.

LETGO will stop unlocking and relocking a file if the index value returned on a call is used as the <u>unit</u> parameter on a subsequent call. LETGO will also stop unlocking and relocking the file when the associated unit is released, e.g., when the FDUB is released by calling the subroutine FREEFD.

Example: Assembly:

```
          LA   1,=C'DATABASE '
          CALL GETFD
          ST   0,FDUB
          CALL LETGO,(FDUB,READ,TIME)
            .
            .
     FDUB DS   A           FDUB-pointer
     READ DC   F'1'        Lock for read
     TIME DC   F'3000000'  3 seconds
```

This example will unlock the file DATABASE every 3 seconds and then relock it for reading. This would allow some other job, for example, to lock it for modification occasionally (every 3 seconds of elapsed time).

LINK, LINKF

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

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

Parameters:

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 8 in <u>info</u> must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an input subroutine to be called during loading via a READ subroutine calling sequence to read loader records (i.e., the input subroutine is called with a parameter list identical to the system subroutine READ). In this case, bit 9 in <u>info</u> must be 1.

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

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

bit 0: 1, if <u>errexit</u> is specified.
bit 1: 1, if <u>output</u> is specified.
bit 2: 1, if <u>lsw</u> is specified.

bit 3: 1, if gtsp is specified.
bit 4: 1, if frsp is specified.
bit 5: 1, if pnt is specified.
bit 6: 1, if to suppress search of LIBSRCH/*LIBRARY libraries.
bit 7: 0, unused (must be zero)
bit 8: 1, if input is the location of a logical I/O unit name.
bit 9: 1, if input is the location of an input subroutine address.
bit 10: 1, if output is the location of a logical I/O unit name.
bit 11: 1, if output is the location of an output subroutine address.
bit 12: 1, if the program to be loaded is to be merged with the program previously loaded.
bit 13: 1, to suppress prompting at a terminal.
bit 14: 1, to force allocation of a new loader symbol table.
bit 15: 0

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

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

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

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

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

12: no available storage index numbers.
16: maximum number of link levels exceeded.

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

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

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 10 of info must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an output subroutine to be called during loading via the SPRINT subroutine calling sequence to write loader output (i.e., the output subroutine is called with a parameter list identical to the system subroutine SPRINT). In this case, bit 11 of info must be 1.

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

bits 0-23: 0
bit 24: 1, to suppress the pseudo-register map.
bit 25: 1, to suppress the predefined symbol map.
bit 26: 1, to print undefined symbols.
bit 27: 1, to print references to undefined symbols.
bit 28: 1, to print references to all external symbols.

bit 29: 1, to print dotted lines around the loader map.
bit 30: 1, to print a map.
bit 31: 1, to print nonfatal error messages.

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

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

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

Values Returned:

None.

Description: LINK provides a method for dynamically loading and executing a program. LINK provides this facility as follows:

(1) The loader is called to dynamically load the specified program using input, info, output, lsw, gtsp, frsp, and pnt if specified.
(2) The dynamically loaded program is called with the address of parlist in GR1.
(3) If the dynamically loaded program returns to LINK, it is unloaded.
(4) LINK returns to the calling program preserving the return registers of the dynamically executed program.

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

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

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

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

Example:

FORTRAN:

```
INTEGER*2 PAR(4)
INTEGER*4 ADROF
DATA PAR/6,'*T','P1','* '/
CALL LINKF('*LABELSNIFF ',0,ADROF(PAR))
END
```

The above FORTRAN program is equivalent to issuing the MTS command "$RUN *LABELSNIFF PAR=*TP1*".

Assembly:

```
             CALL LINK,(INPUT,INFO,LPAR,ERRX,OUTPT,LSW)
              .
              .
      ERROR  STM  14,12,12(13)
              .
              .
      INPUT  DC   C'MYLIB '
      INFO   DS   0F
             DC   XL2'E00C'
             DC   H'1'
             DC   CL8'GETDATA',F'0'
      LPAR   DC   A(PAR)
      PAR    DC   A(0)
      ERRX   DC   A(ERROR)
      OUTPT  DC   C'-MAP '
      LSW    DC   A(X'02')
```

The above assembly language program will dynamically load and execute the routine GETDATA from the private library MYLIB. The initial ESD list is required to force the symbol GETDATA to be initially undefined so that it will be extracted from MYLIB. The INFO and LSW control bits specify:

(1) GETDATA is to be merged with currently loaded programs.
(2) No loader prompting will be done in an attempt to recover from a loading error.
(3) The statement labeled ERROR is to receive control if a loading error occurs.
(4) A complete loader map without dots is to be placed into the file -MAP.

LIOUNITS

Subroutine Description

Contents: A complete table of legal MTS logical I/O unit names.

Location: Resident System

Alt. Entry: LIOUNS

Description: This table can be used to test the validity of an I/O device unit name. The first fullword gives the number of entries in the table. Each entry following is an 8-character left-justified device unit name, e.g.,

```
"SCARDS  "
"SPRINT  "
"0       "
"99      "
```

Example: Assembly:

```
      L   15,=V(LIOUNITS)
      L   1,0(15)          Get number of entries
      LA  15,4(15)         Get address of first entry
LOOP  CLC 0(8,15),NAME     Compare name to table
      BE  FOUND            Branch if legal name
      LA  15,8(15)         Bump pointer to next entry
      BCT 1,LOOP           Reduce count
       .                   Here, if name is illegal
       .
       .
       .
NAME  DC  CL8'12'          Left-justified name for unit 12
```

FORTRAN:

```
      REAL*8 NAMES(1),NAME
      COMMON /LIOUNS/NUMBER,NAMES
       ...
      READ (5,100) NAME
100   FORMAT (A8)
      DO 10 I=1,NUMBER
      IF (NAME.EQ.NAMES(I)) GO TO 20
10    CONTINUE
       ...
20    Error exit
```

The above example, given in both assembly language and FORTRAN, checks for a valid I/O device unit name.

In addition for the FORTRAN example, a RIP loader record (RIP LIOUNS) must be inserted into the FORTRAN object file to force the loader to resolve the symbol LIOUNS from the low-core symbol table.

LOAD, LOADF

Subroutine Description

Purpose: To effect the dynamic loading of a program.

Location: Resident System

Calling Sequences:

```
Assembly: CALL LOAD,(input,info,switch,rtnlist,output,
                     lsw,gtsp,frsp,pnt)

FORTRAN:  indx = LOADF(input,info,switch,rtnlist,output,
                       lsw,gtsp,frsp,pnt)
```

Parameters:

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 8 in _info_ must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an input subroutine to be called during loading via a READ subroutine calling sequence to read loader records (i.e., the input subroutine is called with a parameter list identical to the system subroutine READ). In this case, bit 9 in _info_ must be 1.

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

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

bit 0: 1, if _rtnlist_ is to be ignored.
bit 1: 1, if _output_ is specified.
bit 2: 1, if _lsw_ is specified.

bit 3: 1, if gtsp is specified.
bit 4: 1, if frsp is specified.
bit 5: 1, if pnt is specified.
bit 6: 1, if to suppress search of LIBSRCH/*LIBRARY libraries.
bit 7: 0, unused (must be zero)
bit 8: 1, if input is the location of a logical I/O unit name.
bit 9: 1, if input is the location of an input subroutine address.
bit 10: 1, if output is the location of a logical I/O unit name.
bit 11: 1, if output is the location of an output subroutine address.
bit 12: 1, if the program to be loaded is to be merged with the program previously loaded.
bit 13: 1, to suppress prompting at a terminal.
bit 14: 1, to force allocation of a new loader symbol table.
bit 15: 0

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

switch is the location of a fullword of load control bits defined as follows:

bits 0-7: the storage index number to be used if bit 29 or 30 is 1; else, optionally, the number of the segment into which the program is to be loaded.
bit 8: 1, if rtnlist is to be ignored.
bit 9: 1, if output is specified.
bit 10: 1, if lsw is specified.
bit 11: 1, if gtsp is specified.
bit 12: 1, if frsp is specified.
bit 13: 1, if pnt is specified.
bits 14-19: 0
bit 20: 1, if input is the location of a logical I/O unit name.
bit 21: 1, if input is the location of an input subroutine address.
bit 22: 1, if output is the location of a logical I/O unit name.

bit 23: 1, if output is the location of an output subroutine address.
bit 24: 0
bit 25: 1, if the program to be loaded is to be merged with those previously loaded.
bit 26: 1, to return if a loading error occurs.
0, to call MTS if a loading error occurs.
bit 27: 1, to suppress prompting at a terminal.
bit 28: 1, to force allocation of a new loader symbol table.
bit 29: 1, to load using the storage index number specified in bits 0-7.
bit 30: 1, load into system storage (bits 0-7 contain the storage index number to be used). This bit is only valid for systems programs.
bit 31: 0, load at the highest link level;
1, load at the current link level.

rtnlist is either 0 or the address of an area into which the loader will place an ESD list of all the symbols in the loader symbol table.

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 10 of info must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an output subroutine to be called during loading via the SPRINT subroutine calling sequence to write loader output (i.e., the output subroutine is called with a parameter list identical to the system subroutine SPRINT). In this case, bit 11 of info must be 1.

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

bits 0-23: 0
bit 24: 1, to suppress the pseudo-register map.
bit 25: 1, to suppress the predefined symbol map.
bit 26: 1, to print undefined symbols.
bit 27: 1, to print references to undefined symbols.
bit 28: 1, to print references to all external symbols.
bit 29: 1, to print dotted lines around the loader map.
bit 30: 1, to print a map.
bit 31: 1, to print nonfatal error messages.

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

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

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

Values Returned:

LOAD: If loading was successful,

GR15 contains the loader-defined entry point,
GR0 contains the storage index number used.

If a loading error occurred,

GR15 contains zero,
GR0 contains the loader status word, and
GR1 contains the error code:

0: Attempt to load a null program.
4: Fatal loading error (bad object program).
8: Undefined symbols referenced by the loaded program.
12: No available storage index numbers.
16: Loading aborted by attention interrupt. This error code will be returned only if bits 26 and 27 of _switch_ are set on a call to LOAD.

LOADF: If loading was successful, a positive INTEGER*4 storage index number is returned as the value of LOADF. This number is used to uniquely identify the dynamically loaded program on subsequent calls to STARTF and UNLDF.

If a loading error occurred, a negative INTEGER*4 error code is returned as the value of LOADF, and is defined as follows:

-1: Attempt to load a null program.
-2: Fatal loading error (bad object program).
-3: Undefined symbols referenced by the loaded program.
-4: No available storage index numbers.
-5: Loading aborted by attention interrupt. This error code will be returned only if bits 26 and 27 of _switch_ are set on a call to LOADF.

Description: LOAD provides a method for dynamically loading a program. LOAD provides this facility as follows:

(1) The loader is called to dynamically load the specified program using _input_, _info_, _output_, _lsw_, _gtsp_, _frsp_, and _pnt_ if specified.
(2) LOAD returns to the calling program with the return values described above.

Note that LOAD accepts a variable-length parameter list of 4 to 9 arguments. For most applications, only the first 4 are required. Both _info_ and _switch_ contain load control bits, some of which are duplicates. In these cases, LOAD and LOADF produce a single control bit by ORing the two together.

FORTRAN programs (or programs that use the FORTRAN I/O library) that dynamically load other FORTRAN programs (or programs using the FORTRAN I/O library) should use the alternate entry point LOADF. LOADF is required to provide the dynamically loaded program with a FORTRAN I/O environment consistent with the "merge" bit specified in _info_. If the "merge" bit is one, the dynamically loaded program will have the same I/O environment as the calling program. If the "merge" bit is zero, the dynamically loaded program will have a separate, reinitialized I/O environment. Both FORTRAN main programs and subroutines can be dynamically loaded using LOADF. However, the effect of executing a STOP statement from a dynamically loaded subroutine will depend on the setting of the "merge" bit. If the "merge" bit is 1, a return is made to the calling program; if the "merge" bit is 0, a return is made to MTS. LOADF returns an INTEGER*4 storage index number used to uniquely identify the dynamically loaded program on subsequent calls to STARTF and UNLDF.

Because the rate structure for usage of MTS includes a charge for allocated virtual memory integrated over CPU time, the cost of running a large software package in MTS can often be reduced by dynamically loading and executing seldom-used subroutines via a call to LOAD. Such savings in the storage integral must be weighed against the additional CPU time required to open a second file, reinvoke the loader, and rescan the required libraries.

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

Examples: Assembly:

```
                CALL LOAD,(NAME,INFO,SWIT,0)
                  .
                  .
          INPUT STM  14,12,12(13)
                  .
                  .
          NAME  DC   C'*LIBRARY '
          INFO  DS   0F
                DC   XL2'0',H'2'
                DC   CL8'SPRINT ',A(INPUT)
                DC   CL8'PLOT1',F'0'
          SWIT  DC   F'0'
```

The above example will load the modules defining PLOT1 from *LIBRARY and will intercept any calls they make to SPRINT. An initial ESD list entry with a value of zero is interpreted as a request to include that symbol in the loader tables as referenced, but not defined. Note that the value returned by register 15 is the entry point of the modules loaded which may or may not be PLOT1. To get the address of PLOT1, the LOADINFO subroutine may be called, or the "return ESD list" parameter may be specified on the call to LOAD.

```
FORTRAN:  LOGICAL*1 PAR(8)
          DATA PAR/'H','I',' ','T','H','E','R','E'/
          INTEGER SWITCH/Z00800040/
          INTEGER*2 LPAR(5)/8/
          EQUIVALENCE (LPAR(2),PAR)
          ID = LOADF('FORTOBJ ',0,SWITCH,0)
          CALL STARTF(ID,LPAR)
          CALL UNLDF(0,ID,0)
```

The above FORTRAN program dynamically loads the program in the file FORTOBJ at the highest link level with the "merge" bit set to 1. Subsequently, the loaded program is executed via a call to STARTF and unloaded via a call to UNLDF.

LOADINFO

Subroutine Description

Purpose: To return information about an external symbol or a virtual memory address.

Location: Resident System

Alt. Entry: LDINFO

Calling Sequences:

Assembly: CALL LOADINFO,(type,item,bitsout,regout)

Parameters:

type is the location of a fullword-integer type code:
- 0 = _item_ parameter specifies a fullword-integer ESDID (external symbol dictionary ID).
- 1 = _item_ parameter specifies the name of an external symbol.
- 2 = _item_ parameter specifies a virtual memory address.
- 3 = _item_ parameter specifies a fullword-integer index.
- 4 = _item_ parameter specifies a two fullword-integer RLD (relocation dictionary) index vector, N and M.
- 11 = _item_ parameter specifies a long-symbol-name area.
- 13 = _item_ parameter specifies a fullword integer index.
- 15 = _item_ parameter specifies the name of an external symbol.

If 256 is added to the _type_ code, information is returned from the system loader tables instead of from the loader table used to load the current user program, e.g., _type_=257 may be used to obtain loader information for the system symbol name specified by _item_.

item is either the location of a fullword-integer ESDID of a symbol, the location of an 8-character external symbol (left-justified with trailing blanks), the location of a fullword virtual memory address, the location of a fullword integer index, or the location of a two fullword-integer index vector, N and M.

item can also be a long-symbol-name area. This area consists of three halfwords followed by eight or more characters. The first halfword is the length of the character area, the second halfword is the returned length of a symbol, and the third length is the actual length of the symbol (which may be longer if the symbol does not fit in the area). A sample area might look like the following:

```
MAXLEN  DC  H'100'  Length of area
RETLEN  DS  H       Returned length
SYMLEN  DS  H       Actual symbol length
SYMBOL  DS  CL100   Symbol
```

bitsout is the location of a fullword into which LOADINFO will put output code bits or if the _type_ parameter is 4, the address of a fullword into which LOADINFO will place the flag byte of the RLD item specified by the _item_ parameters N and M.

regout is either the location of a region of 20 fullwords into which LOADINFO will put information about the symbol or virtual memory address or if the _type_ parameter is 4, a fullword into which LOADINFO will place the relocated address of the RLD item specified by the _item_ parameters N and M. This region is cleared to zeros by LOADINFO before information is inserted. If the _type_ parameter is 15, this area must be a long-symbol-name area, with the maximum length filled in properly.

Return Codes:

0 Successful return.
4 Symbol or csect not found in loader tables.
8 Loader tables are not available.
12 Illegal parameter.

Description: The global switch SYMTAB must be ON for this subroutine to return information about the current user program.

For a type 0 call, information for the symbol of the specified ESDID is returned only if the ESDID is currently in the loader ESDID table. This table is available for a particular module only while the loader is reading the module; the table is no longer available after the END record is read.

For a type 1 call, the loader tables are searched for the symbol specified.

For a type 2 call, the loader tables are searched for information about the control section containing the specified virtual memory address.

The type 3 call can be used to return all the information in the loader tables as follows: If the index specified is negative, LOADINFO replaces it with the number of entries in the loader tables. If the index is nonnegative, LOADINFO will return the (n+1)th entry in the loader tables and increment the index by 1. Thus, by setting the index initially to zero, and then calling LOADINFO repeatedly until a nonzero return code is detected, all the information in the loader tables can be accessed.

The type 4 call can be used to return all the relocation dictionary information in the loader tables as follows: The _item_ parameter is a two fullword-integer vector of indices, N and M, where the (M+1)th RLD item for the Nth symbol table entry will be returned in _bitsout_ and _regout_. The _bitsout_ parameter will contain the RLD flag byte (TTTTLLST) in bits 24-31 of the fullword and the _regout_ parameter will contain the relocated address in bits 8-31 of the fullword. The index M, which must be zero on the first call, will be incremented by one on each call. Thus, by setting M initially to zero and then by calling LOADINFO repeatedly until a nonzero return code is detected, all the relocation information for the Nth symbol table entry can be accessed. A type 4 call to LOADINFO can only be used in conjunction with a type 3 call, i.e., a type 3 call must first by made to access the Nth symbol table entry before the type 4 calls are made to serially access the RLD information. Normally, RLD information is retained for intermodule references (i.e., for RLD items whose position pointer is not the same as the relocation pointer) and only if the program was loaded under control of the symbolic debugging system (SDS).

A type 11 call is similar to a type 1 call, except that a long-symbol-name area is expected as the _item_ rather than an 8-character external symbol name. The full symbol is expected, so the actual length (SYMLEN) is used to determine the length of the symbol.

A type 13 call is similar to a type 3 call, except that only long-symbol-name entries are returned. A type 3 call returns all entries, including long-symbol-name entries.

A type 15 call is used to find the actual name of a long-symbol name. As long-symbol names will not fit into the 8 characters reserved for the external symbol name in the _regout_ area, a unique 8-character identifier is put there instead. The first fullword of this identifier is a fullword X'FFFFFFFF'. The type 15 call returns the full long-symbol name given this unique identifier.

LOADINFO returns the information for type 0-3, 11, and 13 calls as follows: The bitsout word indicates which pieces of information have been filled in the region regout. Each bit corresponds to a piece of information. If the bit is set, the corresponding information is given. The bit number and the equivalent integer value of the bit are given as the first two columns in the table below. The third column indicates the displacement (in bytes) from the beginning of regout for the particular piece of information.

Bitsout		Regout	
Bit	Value	Displ	Contents
31	1	0	External symbol name (left-justified with trailing blanks) or unique long-symbol identifier (see type 15 call).
30	2	8	Address assigned to the symbol.
29	4	12	Relocation factor if csect or common section.
28	8	16	Length if a csect or common section.
27	16	20	Storage index number.
26	32	24	Symbol type: 0=Undefined symbol 1=Entry point 2=Control section 3=Common section 4=Predefined 5=Library entry point 6=Library control section 7=Library common section
25	64	28	Pseudo-register displacement
24	128	32	Pseudo-register length
23	256	36	Pseudo-register storage index number
22	512	40	Name of the closest entry with a virtual memory address equal to or less than the given address
21	1024	48	Address assigned to the entry named above.
20	2048	52	Loader-assigned internal name for private control section.
		56-79	Reserved for future expansion.

The regout area for type 0-3, 11, and 13 calls can be represented in assembler language with the following dsect (which is available in the public file *LOADINFODSECT).

```
INFOAREA DSECT
SYMNAME  DS     CL8   SYMBOL/CSECT NAME
SYMADDR  DS     A     ASSIGNED VM ADDRESS
SYMRF    DS     A     RELOCATION FACTOR
SYMLEN   DS     F     LENGTH IF CSECT OR COMMON SECTION
SYMSIN   DS     F     STORAGE INDEX NUMBER
SYMTYPE  DS     F     TYPE INFORMATION
PRADDR   DS     A     ASSIGNED PSEUDO-REG DISPLACEMENT
PRLEN    DS     F     LENGTH OF PSEUDO-REGISTER
PRSIN    DS     F     PSEUDO-REG STORAGE INDEX NUMBER
EPNAME   DS     CL8   CLOSEST ENTRY POINT NAME
EPADDR   DS     A     VM ADDRESS OF ABOVE ENTRY POINT
PCID     DS     F     PRIVATE CONTROL SECTION ID
         DS     6F    RESERVED FOR FUTURE EXPANSION
```

If LOADINFO is called with a blank external symbol, it will look only for blank-named common sections and will fail if there are none (even though there may be blank-named control sections). If LOADINFO is called with an external symbol which has been defined at several link levels, it will return the most recent definition.

Examples: FORTRAN:

```
INTEGER*4 TYPE,BITS,REG(20)
DATA TYPE/1/
CALL LDINFO(TYPE,'PLOT1   ',BITS,REG,&98,&99)
```

The above example calls LOADINFO to get information about the symbol PLOT1.

Assembly:

```
LOOP  CALL  LOADINFO,(TYPE,ITEM,BITS,REG)
      LTR   15,15
      BNZ   DONE
        .
        .
      B     LOOP
        .
        .
TYPE  DC    F'3'
ITEM  DC    F'0'
BITS  DS    XL4
REG   DS    20A
```

This example calls LOADINFO repeatedly to get information about each symbol in the loader tables. The loop is done when LOADINFO gives a nonzero return code.

LOCK

Subroutine Description

Purpose: To request that a file be locked in the indicated manner, i.e., to dynamically restrict access to a file which has been permitted to be shared by others.

Location: Resident System

Alt. Entry: SETLCK

Calling Sequence:

```
Assembly: CALL LOCK,(unit,howflg,wtflg)

FORTRAN:  CALL LOCK(unit,howflg,wtflg,&rc4,&rc8,&rc12,
                    &rc16,&rc20)
```

Parameters:

- _unit_ is the location of either
  - (a) a fullword-integer FDUB-pointer (as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99), or
  - (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- _howflg_ is the location of a fullword indicating how to lock the file:
  - >0 lock for read
  - =0 lock for modification (write, empty, truncate, etc.)
  - <0 lock for destroy (rename, permit)
- _wtflg_ is the location of a fullword indicating whether or not to wait if the requested locking is not possible at this time:
  - <0 wait indefinitely
  - =0 do not wait
  - >0 the maximum number of milliseconds to wait. If this expires and the file has not been locked, a return code of 20 will be given.
- _rc4...rc20_ are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 The file has been locked in the requested manner.
4 The file does not exist.

8 Hardware error or software inconsistency encountered.
12 Access appropriate to the locking request not allowed.
16 Locking the file as requested will result in a deadlock.
20 Locking the file as requested can not be accomplished at this time, no wait was requested, or the wait was interrupted.

Notes:

Any number of jobs can have a file locked for reading at any given time, but only one job can have a file locked for modification at any given time and then only if no job has the file locked for reading, or locked for destroying. Only one job can have a file locked for destroying at any given time, and then if no job has the file open or locked for reading, or locked for modification.

The three locking levels are inclusive in the sense that locking a file for modification also locks the file for reading and locking a file for destroying also locks the file for modification and reading.

The file is always locked as requested in the case where there is only one FDUB with a locking request on the file within a job. Thus, if a file is already locked for modification via a particular FDUB and it is requested, via the same FDUB, that the file be locked for reading, the file will be essentially unlocked for modification and left locked for reading.

If more than one FDUB within a job has a locking request on the file, the file will be locked at the level of the highest request.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from LOCK with a return code of 20.

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

Examples: Assembly:

```
                    CALL LOCK,(UNIT,HOW,WAIT)
                     .
                     .
              UNIT  DC    F'6'      Logical I/O unit 6
              HOW   DC    F'0'      Lock for modification
              WAIT  DC    F'-1'     Wait indefinitely
```

FORTRAN:

```
      INTEGER*4 UNIT
      DATA UNIT/6/
       ...
      CALL LOCK(UNIT,0,-1)
```

The above examples will lock the file attached to logical I/O unit 6 for modification and wait indefinitely if someone else has the file locked (in any manner).

LODMAP

Subroutine Description

Purpose: To produce a loader map from the current contents of the loader tables.

Location: Resident System

Calling Sequences:

Assembly: CALL LODMAP,(unit,bits)

FORTRAN: CALL LODMAP(unit,bits)

Parameters:

<u>unit</u> is the location of either
(a) a FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SPRINT).
This specifies where the loader map is to be written.

<u>bits</u> is the location of a fullword of switches defined as follows:

bits 0-23: zero
bit 24: one to suppress pseudo-registers
25: one to suppress predefined symbols
26: one to print undefined symbols
27: one to print undefined xrefs
28: zero
29: one to print dotted lines
30: one to print entry point names
31: zero

Return Codes:

0 Successful return.
4 Illegal <u>unit</u> parameter specified.
8 Loader tables not available.

Description: The current contents of the loader tables will be used to produce a loader map under the control of the switches specified. If the global SYMTAB switch is OFF, the loader tables will not be available, generating a return code of 8.

Examples: Assembly:

```
                CALL  LODMAP,(UNIT,BITS)
                LTR   15,15
                BNZ   NOMAP
                 .
                 .
                DS    0F
          BITS  DC    XL3'0',X'C6'
          UNIT  DC    CL8'SERCOM'
```

This example will produce a partial loader map on the logical I/O unit SERCOM.

FORTRAN:

```
      INTEGER UNIT/2/,BITS/6/
       ...
      CALL LODMAP(UNIT,BITS,&98,&99)
```

This example will produce a loader map with dotted lines on logical I/O unit 2.

Logical Operators

Subroutine Description

Purpose: To make the following System/360/370 machine instructions directly available to the FORTRAN user: MVC, CLC, NC, OC, XC, TR, TRT, ED, and EDMK.

Location: *LIBRARY

Entry Points: IMVC, ICLC, INC, IOC, IXC, ITR, ITRT, IED, and IEDMK.

Calling Sequences:

```
FORTRAN:  I = IMVC(len,base1,displ1,base2,displ2)
          I = ICLC(len,base1,displ1,base2,displ2)
          I = INC(len,base1,displ1,base2,displ2)
          I = IOC(len,base1,displ1,base2,displ2)
          I = IXC(len,base1,displ1,base2,displ2)
          I = ITR(len,base1,displ1,base2,displ2)
          I = ITRT(len,base1,displ1,base2,displ2,dr,fb)
          I = IED(len,base1,displ1,base2,displ2)
          I = IEDMK(len,base1,displ1,base2,displ2,dr)
```

Parameters:

<u>len</u> is the integer length in bytes. No restriction is placed on the size of <u>len</u>. An error message will be generated if <u>len</u> < 0; or, for the entries IED or IEDMK, if <u>len</u> > 256.

<u>base1</u> is the base location of the first operand.

<u>displ1</u> is the integer displacement in bytes for the first operand. No restriction is placed on the size of <u>displ1</u>.

<u>base2</u> is the base location of the second operand.

<u>displ2</u> is the integer displacement in bytes for the second operand. No restriction is placed on the size of <u>displ2</u>.

<u>dr</u> is an integer return parameter for ITRT and IEDMK only. For ITRT, <u>dr</u> will contain the displacement in bytes from the beginning of the argument list, (<u>base1</u>+<u>displ1</u>), to the argument corresponding to the first nonzero function byte (if any). For IEDMK, <u>dr</u> will contain the displacement in bytes from the beginning of the source, (<u>base2</u>+<u>displ2</u>), to the result character, whenever the latter is a zoned source digit and the significance indicator was off before the examination. In both cases, <u>dr</u> will be set to zero if the

resulting condition code is zero.

_fb_ is an optional integer return parameter for ITRT. When a nonzero function byte is found, it will be returned in _fb_ as an integer in the range (0,255); otherwise, _fb_ will be zero.

Description: For the description of the machine instructions, see the IBM publication, _IBM System/370 Principles of Operation_, form GA22-7000. These subroutines are coded as integer-valued functions with the resulting condition code (0, 1, or 2) as the value.

In the abbreviated descriptions below, the first operand consists of _len_ bytes beginning at location _base1_+_displ1_, and the second operand consists of _len_ bytes beginning at location _base2_+_displ2_. These two operands may overlap in any manner. For all five of these entry points, processing is carried out left to right one byte at a time. Note that the result of performing an operation on the first bytes of the two operands is stored before the second bytes are fetched so that overlap can have a significant effect on the result.

IMVC - Move the second operand into the first operand location.
INC - Replace the first operand by the logical product (AND) of the operands.
IOC - Replace the first operand by the logical sum (OR) of the operands.
IXC - Replace the first operand by the modulo-two sum (exclusive OR) of the two operands.
ICLC - Compare the two operands. The operation is terminated as soon as two unequal bytes are found.

The result of an IMVC is always zero. The result of an INC, IOC, or IXC is zero if the result operand is zero, and one, otherwise. The result of an ICLC is 0, 1, or 2, depending on whether the first operand is equal to, less than, or greater than the second operand.

For the ITR and ITRT entries, the first operand consists of _len_ bytes beginning at location _base1_+_displ1_, and the second operand consists of a 256-byte function table beginning at location _base2_+_displ2_. These operands may overlap, but probably not too fruitfully. The ITR entry translates each byte of the first operand by replacing it with the corresponding byte from the function table. The result of an ITR operation is always zero. The ITRT entry does not change either operand. Processing the first operand bytes left to right, the corresponding function byte is interrogated. If the function byte is zero, the processing of the first operand continues. If the function byte is nonzero, the operation is terminated. When terminated, processing is terminated with the byte at location base1+displ1+dr, and the corresponding nonzero function byte is available in fb. The result of the ITRT will be 1 if this byte is not the last byte of the first operand, and 2 if it is the last byte. If no nonzero function byte is encountered, the result of an ITRT will be zero, and dr and fb will be indeterminate.

The complexity of the IED and IEDMK instructions precludes any short descriptions here.

Examples:

```
      INTEGER A, B
      B = 31
      LEN = 4
      IR = INC(LEN,A,0,B,0)
```

The logical AND product of A and B will replace A. In this case, B = 31, so A will be replaced by (A mod 32). IR will be set to 0 or 1 depending on whether the result in A is zero or nonzero.

```
      INTEGER A(4),B(4),D1,D2
      READ 2, (A(I),I=1,4), (B(I),I=1,4)
    2 FORMAT(4A4)
      D1 = 8
      D2 = 0
      IR = ICLC(8,A,D1,B,D2)
```

This program logically compares the string in A(3), A(4), to the string in B(1), B(2). IR will be set to 0, 1, or 2 depending on whether the first string is equal to, less than, or greater than the second string.

LSFILE

Subroutine Description

Purpose: To allow the user to obtain information about the locking status of a file.

Location: Resident System

Calling Sequences:

```
Assembly: CALL LSFILE,(file,filter,length,icount,needed,
                       lsinfo)

FORTRAN:  CALL LSFILE(file,filter,length,icount,needed,
                      lsinfo,&rc4,&rc8,&rc12,&rc16)
```

Parameters:

<u>file</u> is the location of a region containing a left-justified filename with a trailing blank for which locking information is being requested.

<u>filter</u> is the location of a fullword of bit switches which are used for filtering the information to be returned. Lock information will only be returned for those tasks whose lock status includes at least one item specified by a '1' bit in <u>filter</u>. Bits 0-21 in <u>filter</u> are unused and must be 0.

Bit	Hex Value	Lock Status
22	00000200	File is not open/not locked.
23	00000100	File buffers are invalid.
24	00000080	Waiting to destroy the file.
25	00000040	Waiting to modify the file.
26	00000020	Waiting to read the file.
27	00000010	Waiting to open the file.
28	00000008	File locked for destroy.
29	00000004	File locked for modify.
30	00000002	File locked for read.
31	00000001	File is open.

Note: $LOCKSTATUS uses a <u>filter</u> value of '000003FF' (1023) when calling LSFILE.

<u>length</u> is a fullword location specifying the size of <u>lsinfo</u> in bytes.

<u>icount</u> is an integer variable which will be set to

the number of locking status records returned by LSFILE.

<u>needed</u> is an integer variable which will be set to the number of bytes actually needed by LSFILE to return all requested locking status information.

<u>lsinfo</u> is the user-provided area in which locking status information is returned in the form of two-fullword (eight-byte) records. The first fullword contains the lock state of the file in the same format used by <u>filter</u>. The second fullword is the task number for the job with the file locked. The records are stored contiguously beginning at the first byte of <u>lsinfo</u>, the number of records present being indicated by <u>icount</u>.

<u>rc4,...,rc16</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 All available lock information was returned.
4 No lock information was found for this file.
8 More space was needed than provided to return all lock information. Only as many records as would fit into <u>length</u> bytes were returned.
12 Illegal or invalid parameters.
16 System Error.

Description: If the return code from LSFILE is 12 or 16, no value for <u>needed</u> is returned, and <u>lsinfo</u> remains unchanged. A zero value is returned for <u>icount</u>.

A <u>filter</u> value of hex '000003FF' (decimal 1023) causes all available locking status information to be returned.

Example: Assembly:

```
          CALL LSFILE,(FILENAME,FILTER,REGLEN,
                       COUNT,NEEDED,REGION)
            .
            .
FILENAME DC  C'MYFILE '
FILTER   DC  X'000003FF'
REGLEN   DC  F'400'
COUNT    DS  F
NEEDED   DS  F
REGION   DS  400CL1
```

FORTRAN:

```
      INTEGER*4 FILENME(2)/'MYFI','LE  '/,
     1  REGLEN/400/,FILTER/Z000003FF/,COUNT,
     2  NEEDED,REGION(100)
      CALL LSFILE(FILENAME,FILTER,REGLEN,COUNT,
     1             NEEDED,REGION,&10,&20,&30,&40)
```

The above examples obtain lock status information for the file MYFILE and place the information into the 400-byte area REGION.

LSTASK

Subroutine Description

Purpose: To allow the user to obtain information about the locking status of files by a given task.

Location: Resident System

Calling Sequences:

Assembly: CALL LSTASK,(task,filter,length,icount,needed, lsinfo)

FORTRAN: CALL LSTASK(task,filter,length,icount,needed, lsinfo,&rc4,&rc8,&rc12,&rc16)

Parameters:

task is the location of a fullword region containing the task number for which locking information is being requested.

filter is the location of a fullword of bit switches which are used for filtering the information to be returned. Lock information will only be returned for those tasks whose lock status includes at least one item specified by a '1' bit in filter. Bits 0-21 in filter are unused and must be 0.

Bit	Hex Value	Lock Status
22	00000200	File is not open/not locked.
23	00000100	File buffers are invalid.
24	00000080	Waiting to destroy the file.
25	00000040	Waiting to modify the file.
26	00000020	Waiting to read the file.
27	00000010	Waiting to open the file.
28	00000008	File locked for destroy.
29	00000004	File locked for modify.
30	00000002	File locked for read.
31	00000001	File is open.

Note: $LOCKSTATUS uses a filter value of '000003FF' (1023) when calling LSTASK.

length is a fullword location specifying the size of lsinfo in bytes.

icount is an integer variable which will be set to the number of locking status records returned by LSTASK.

needed — is an integer variable which will be set to the number of bytes actually needed by LSTASK to return all requested locking status information.

lsinfo — is the user-provided area in which locking status information is returned in the form of variable-length records, with each record formatted as follows. The first fullword contains the length of the record. The second fullword contains the lock state of the file in the same format used by filter. The third fullword contains the length of the returned file name. The remainder of the record is the name of a file which the task has locked. The file name is padded on the right to make the length divisible by 4, ensuring that records are fullword-aligned. The records are stored contiguously beginning at the first byte of lsinfo, the number of records present being indicated by icount.

rc4,...,rc16 — (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

- 0 All available lock information was returned.
- 4 No lock information was found for this task.
- 8 More space was needed than provided to return all lock information. Only as many records as would fit into length bytes were returned.
- 12 Illegal or invalid parameters.
- 16 System Error.

Description: If the return code from LSTASK is 12 or 16, no value for needed is returned, and lsinfo remains unchanged. A zero value is returned for icount.

A filter value of hex '000003FF' (decimal 1023) causes all available locking status information to be returned.

Examples: Assembly:

```
                CALL GUINFO,(ITEM,TASKNUM)
                CALL LSTASK,(TASKNUM,FILTER,REGLEN,
                             COUNT,NEEDED,REGION)
                 .
                 .
      ITEM      DC   CL8'TASKNBR '
      TASKNUM   DS   F
      FILTER    DC   X'000003FF'
      REGLEN    DC   F'400'
      COUNT     DS   F
      NEEDED    DS   F
```

```
            REGION   DS  400CL1
            END
```

FORTRAN:

```
      INTEGER*4 TASKNUM,NEEDED,REGLEN/400/,
     1 FILTER/Z000003FF/,COUNT,NEEDED,
     2 REGION(100)
      CALL GUINFO('TASKNBR ',TASKNUM)
      CALL LSTASK(TASKNUM,FILTER,REGLEN,COUNT,
     1            NEEDED,REGION,&10,&20,&30,&40)
```

The above examples obtain lock status information for the user's current task (as determined by a call to GUINFO) and place the information into a 400-byte area REGION.

MOUNT

Subroutine Description

Purpose: To mount magnetic and paper tapes, floppy disks, and connections on the Merit Computer Network.

Location: Resident System

Calling Sequences:

Assembly:

```
CALL MOUNT,(mntreq,reqlen)

CALL MOUNT,(par)

CALL MOUNT,(numreq,string,len,option,ercode,
            errmsg),VL
```

FORTRAN:

```
CALL MOUNT(mntreq,reglen)

CALL MOUNT(par)

CALL MOUNT(numreq,string,len,option,ercode,
           errmsg)
```

Parameters:

- mntreq is the location of a character string containing one or more mount requests, each separated by a semicolon.
- reqlen is the location of a halfword (INTEGER*2) length of mntreq.
- par is the location of a halfword (INTEGER*2) length of a character string immediately followed by that character string. The character string contains one or more mount requests, each separated by a semicolon.
- numreq is the location of a fullword number of mount requests specified in string.
- string is the location of a character string containing numreq mount requests, each separated by a semicolon.
- len (optional) is the location of the total length of the mount request string, expressed as either a fullword (INTEGER*4) or a halfword (INTEGER*2). If the first two bytes specified are zero, it is assumed that len specifies a fullword integer. Otherwise, len is assumed to be a halfword. If len specifies a fullword zero or is omitted, the last

mount request in _string_ must be terminated by a semicolon.

option (optional) is the location of a fullword containing mount control switches defined as follows:

bits 0-15: must be zero.
bit 16: 1, to suppress the echoing of mount requests.
bit 17: 1, to suppress the printing of any error messages.
bit 18: 1, to suppress the prompting of a terminal user for replacement of an erroneous mount request.
bit 19: 1, to mount any request that can be fulfilled, even if other requests could not be. By default, the MOUNT subroutine will abort all requests if one or more are erroneous and cannot be fulfilled.
bit 20: 1, to suppress verification of a successful mount.
bit 21: 1, to suppress attention interrupts while processing the mount requests. If this bit is set, the user will not be able to interrupt the operator wait.
bit 22: 1, to suppress the pseudodevice name/rack number prefix from error messages printed or returned by the MOUNT subroutine.
bit 23: 1, to wait in the tape mount queue, if necessary, without prompting the terminal user. Bit 23 will be ignored if the mount request is issued from a batch job, or if bit 24 is set.
bit 24: 1, to prohibit the request from being queued if there are not enough drives, without prompting the terminal user. A "busy" return will be made by the MOUNT subroutine in this case. Bit 24 will be ignored if the mount request is issued from a batch job.
bits 25-31: must be zero.

ercode (optional) is the location of a vector of _numreq_ fullword integers in which the MOUNT subroutine will place an error number for each mount request if an error return (return code > 0) is made. This parameter should be dimensioned as INTEGER ERCODE(n), where "n" is greater than or equal to the number of mount requests numreq.

Error numbers less than 100 indicate an error in the mechanics of the subroutine call or in the values of the parameters. Note that it may be impossible to return some of these error numbers if the appropriate parameters are not addressable.

Number	Message
1	Illegal "numreq" parameter.
2	Illegal "string" parameter.
3	Illegal "len" parameter.
4	Illegal "option" parameter.
5	Illegal "ercode" parameter.
6	Illegal "errmsg" parameter.
7	Missing "string" parameter.
8	Missing "len" parameter.
9	Invalid "option" bits specified.
99	Request not processed. This error number is returned if a mount request was not processed because a previous request was aborted. This may occur if a terminal user entered "CANCEL" when prompted for replacement of an erroneous request.

Error numbers between 100 and 199 indicate syntax errors in the mount request:

100	Rack number was not given.
101	Device type was not specified.
102	Pseudodevice name was not given.
103	Invalid pseudodevice name.
104	Pseudodevice name too long.
105	Invalid device type.
106	Invalid rack number.
107	Invalid block size.
108	Invalid logical record length.
109	Invalid keyword "xxx".
110	Invalid expiration date.
111	Invalid data set name.
112	"xxx" has invalid syntax.
113	Missing required prime field.
114	EOR hex character count not between 1 and 8.

115 EOR field contains illegal hex character.
116 Length of EOR hex field does not match count.
120 POSN specifies an invalid track number.
121 POSN specifies an invalid sector number.
122 Invalid SECMAP sector number.
123 SECMAP does not specify 26 sector numbers.

Error numbers between 200 and 299 indicate semantic errors in the mount request:

200 Read access not allowed to tape.
201 Write access not allowed to tape (cannot mount tape with RING=IN).
202 INIT=YES valid only for labeled tape with RING=IN.
203 MODE=xxx is inconsistent with device type.
204 MODE=xxx is not available on device "yyy".
205 Not enough devices available to satisfy this request.
206 Pseudodevice name already requested for "xxx".
207 Pseudodevice name is in use by MTS.
208 Pseudodevice name is already in use for device type "xxx".
209 POOL is invalid for paper tape reader/punch.

Error numbers between 300 and 399 indicate errors determined by the operator:

300 System in unattended mode; no mounts allowed at this time.
301 Mounts are temporarily disabled; try again later.
302 Incorrect rack number.
303 Incorrect tape id.
304 All units busy at this time.
305 Volume label is incorrect.
306 Tape is not of specified mode.
307 Permanent I/O error on first tape block.
308 Volume name not given for labeled tape.
309 Aborted by operator (reason given).
310 Not available (reason given).
311 Aborted (by user attention

interrupt).
312 Aborted (due to error in another request).

Error numbers between 400 and 499 indicate errors from a control operation on the device.

400 Initialization failed.
401 Positioning failed.
402 Return code 4 from CONTROL.
403 Error message from CONTROL.

450 Invalid host name
451 Network path to host is shutdown
452 No host ports of desired type exist
453 Host is down
454 No socket available in local PCP
455 Invalid connection type
456 Remote PCP/SCP is isolated from network
457 No socket available in remote PCP/SCP
458 No connections currently allowed to remote PCP/SCP
459 Host does not accept surcharges
460 Should not occur
461 No more wraparound connections allowed
462 Host ports are busy
463 Should not occur
464 Should not occur
465 Should not occur
466 Should not occur
467 Should not occur
468 Internal error in network DSR
469 Network connections not allowed now
470 Connection establishment interrupted
471 Internal network error
472 Internal network error
473 Connection establishment interrupted
474 Connection establishment interrupted
475 Internal network error
476 Network not responding

Error numbers 500 and above indicate a system error and should not occur.

The error number for a particular mount request will be zero if the tape or device was mounted successfully even if some other mount request had an error or was not fulfilled.

errmsg (optional) is the location of a vector of numreq elements in which the MOUNT subroutine will place the corresponding error message if an error code > 0 is returned for a particular mount request. Each element of the errmsg vector is 20 fullwords (80 characters) long. This parameter should be dimensioned as INTEGER*4 ERRMSG(20,n), LOGICAL*1 ERRMSG (80,n), etc., where "n" is greater than or equal to the number of mount requests numreq in string. The MOUNT subroutine will initially clear this vector to blanks.

Return codes:

- 0 All requests were successfully processed.
- 4 One or more of the requests could not be fulfilled.
- 8 The operator or user caused one or more of the requests to be aborted.
- 12 System error.
- 16 Illegal parameter(s) in call to MOUNT.

Notes:

The MOUNT subroutine prints messages on the logical I/O unit SERCOM. MOUNT subroutine error messages can be suppressed by setting bit 17 of option to 1. The echoing of mount requests on SERCOM can be suppressed by setting bit 16 of option to 1, or by the MTS $SET ECHO=OFF command (or by calling the CUINFO subroutine for the ECHOOFF item to perform the equivalent function).

Assembly language users wishing to omit the optional parameters len, option, ercode, or errmsg should either follow the variable-length parameter list convention (high-order bit of the previous parameter adcon in the parameter list is 1) or else supply an adcon which is zero (rather than pointing to a zero). FORTRAN users should note that if an optional parameter is omitted, all parameters in the calling sequence following the omitted parameter must also be omitted. For example, if ercode is omitted, errmsg must also be omitted.

Description: See the $MOUNT command description in MTS Volume 1, The Michigan Terminal System, for details on the form of a mount request. For a complete description of the available mount parameters, see the appropriate sections in MTS Volume 19, Tapes and Floppy Disks.

Examples: Assembly:

```
                    CALL MOUNT,(STR,LEN),VL
                     .
                     .
          LEN       DC    H'28'
          STR       DC    C'POOL 9TP *T*;MNET *MSU* D=MS'
```

FORTRAN:

```
          INTEGER SWS/Z00006000/,ERR(2)
          LOGICAL*1 MSG(80,2)
           ...
          CALL MOUNT(2,'POOL 9TP *T*;MNET *MSU* D=MS;',
                     0,SWS,ERR,MSG,&4,&8,&12,&16)
```

The above examples call MOUNT to mount a 9-track pool tape with pseudodevice name *T* and a Merit connection to Michigan State University with pseudodevice name *MSU*. The FORTRAN example specifies the error code and message vectors in order to obtain more specific error return information. MOUNT option bits are specified to suppress printing of error messages and prompting of terminal users. Note also that the FORTRAN example specifies a length of zero for the len parameter, so the mount request string is terminated by a semicolon.

MTS

Subroutine Description

Purpose: To suspend execution of a program and return to MTS command mode or to the previous command language subsystem. Issuing a $RESTART command will cause execution of the program to resume by causing a return from the MTS subroutine call.

Location: Resident System

Alt. Entry: MTS#

Calling Sequences:

Assembly: CALL MTS

or

MTS

FORTRAN: CALL MTS

Return Codes:

None

Note: The complete description for using the MTS macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

MTSCMD

Subroutine Description

Purpose: To suspend execution of a program, return to MTS command mode or to the previous command language subsystem, and feed a character string to the MTS command interpreter.

Location: Resident System

Alt. Entry: MTSCMD#

Calling Sequence:

Assembly: CALL MTSCMD,(locn,length)

or

MTSCMD locn[,length]

FORTRAN: CALL MTSCMD(locn,length)

Parameters:

- _locn_ is the location of a character string containing a command.
- _length_ is the location of the length of the character string expressed as either a fullword (INTEGER*4) or a halfword (INTEGER*2). If the first two bytes of _length_ are zero, it is assumed _length_ specifies a fullword integer. Otherwise, _length_ is taken as halfword.

Return codes:

The subroutine does not return except as described below.

Note: The complete description for using the MTSCMD macro is given in MTS Volume 14, _360/370 Assemblers in MTS_.

Description: This subroutine returns to MTS, as does the subroutine MTS, but in addition gives it a character string to interpret as a command. If a $RESTART command is issued before the next $RUN, $RERUN, $LOAD, or $DEBUG command, the subroutine will "return," i.e., the program calling MTSCMD will restart following the subroutine call.

Examples: FORTRAN: CALL MTSCMD('$RESTART SPRINT=*DUMMY* ',24)

Assembly:

```
              CALL MTSCMD,(INREG,INLEN)
               .
               .
        INREG DC   C'$RESTART SPRINT=*DUMMY* '
        INLEN DC   F'24'

              MTSCMD '$RESTART SPRINT=*DUMMY* '
```

The above three examples call MTSCMD to reassign the logical I/O unit SPRINT to *DUMMY*. The first assembly example uses the CALL macro and the second uses the MTSCMD macro.

NOTE

Subroutine Description

Purpose: To "remember" the values of the logical pointers for a sequential file. This information is used by the POINT subroutine to change the values of the logical pointers.

Location: Resident System

Alt. Entry: NOTE#

Calling Sequences:

Assembly: CALL NOTE,(unit,info)

FORTRAN: CALL NOTE(unit,info,&rc4,&rc8,&rc12,&rc16,&rc20,
&rc24,&rc28)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

info is the location of a region of four fullwords into which the NOTE subroutine will return the values of the Read, Write, and Last Pointers, as well as the the last line number respectively for the sequential file pointed to by unit.

rc4,...,rc24 are the statement labels to transfer to if a nonzero return code is encountered.

Return Codes:

0 Successful return.
4 Illegal FDUB-pointer specified.
8 Illegal parameter specified.
12 Read or write access not allowed.
16 Locking the file for reading will result in a deadlock.
20 Hardware error or software inconsistency encountered.
24 Automatic wait for shared file was interrupted.

Notes: The Read and Write Pointers have values which point to the next line to be read or written.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from NOTE with a return code of 24.

Description: See Appendix B of the section "Files and Devices" in MTS Volume 1, The Michigan Terminal System, for details concerning using sequential files with the NOTE and POINT subroutines.

Examples:

Assembly:

```
            CALL NOTE,(UNIT,INFO)
              .
              .
      UNIT  DC   F'6'
      INFO  DS   4F
```

FORTRAN:

```
      INTEGER*4 UNIT,INFO(4)
      DATA UNIT/6/
       ...
      CALL NOTE(UNIT,INFO)
```

The above examples will call NOTE for the sequential file attached to logical I/O unit 6.

NPAR

Subroutine Description

Purpose: To count the number of parameters passed to a subroutine.

Location: *LIBRARY

Calling Sequences:

FORTRAN: i = NPAR(n)

Parameters:

n is the number of subroutine or function calls to be counted. That is, a value of 1 will return the number of parameters passed to the subroutine in which NPAR is called. A value of 2 would return the number of parameters passed to the subroutine that called the subroutine that called NPAR. For most uses, *n* will be 1. An error message is generated if *n* exceeds the nesting level of the subroutine calling NPAR.

Multiple return statement numbers are not counted as parameters by NPAR.

i is number of parameters passed.

Notes: Standard OS Type-I(S) calling conventions must be used in all subroutine calls. See the section "Calling Conventions" in this volume.

If the subroutine calling NPAR has more parameters in its parameter list than are provided by its caller, then the excess parameters must be enclosed in slashes. Otherwise, a program interrupt may occur during the entry prolog code to the subroutine.

Example: FORTRAN:

```
      CALL SUBR(X)
      STOP
      END

      SUBROUTINE SUBR(/X/,/Y/,/Z/)
      I = NPAR(1)
      IF (I .GE. 4) GO TO 10
      IF (I .EQ. 3) GO TO 300
      IF (I .EQ. 2) GO TO 200
      IF (I .EQ. 1) GO TO 100
```

```
10    WRITE(6,11)
11    FORMAT('ERROR')
        ...
100     ...
200     ...
300     ...
        ...
      RETURN
      END
```

In the above example, NPAR counts the number of parameters passed to SUBR and sets up a branch accordingly. In this case, one parameter was passed.

OSGRDT

Subroutine Description

Purpose: To convert the OS date (YYddd) to the corresponding Gregorian date (MM/DD/YY).

Location: *LIBRARY

Calling Sequences:

Assembly: CALL OSGRDT,(osdat,grgdat)

FORTRAN: CALL OSGRDT(osdat,grgdat,&rc4)

```
REAL*8 OSGRDT
date=OSGRDT(osdat,grgdat)
```

PL/I(F): CALL PLCALL(OSGRDT,f2,osdat,grgdat);

```
DCL PLCALLD RETURNS(FLOAT(16));
date=PLCALLD(OSGRDT,f2,osdat,grgdat);
```

Parameters:

- _osdat_ is the 8-byte (REAL*8 or CHARACTER(8)) OS date in the character form "xxxYYddd", where "x" is any character.
- _grgdat_ is 8 bytes (REAL*8 or CHARACTER(8)) into which the Gregorian date in the character form "MM/DD/YY" is placed on return.
- _f2_ is a fullword (FIXED BINARY(31)) containing the integer 2.
- _rc4_ is a statement label to transfer to if a return code of 4 occurs.

Values Returned:

FR0 contains the Gregorian date in the character form "MM/DD/YY". This is assigned to _date_ for FORTRAN and PL/I programs using the function-call format.

Return Codes:

0 Successful return.
4 At least one digit position in the date does not contain a digit. Upon return, FR0 and _grgdat_ contain blanks.

Description: The range of years is assumed to include 1900. The result for dates prior to 00060 is undefined.

Examples: Assembly:

```
                CALL OSGRDT,(OSDAT,GRDAT)
                 .
                 .
          OSDAT DC   C'   71120'
          GRDAT DS   CL8

                CALL OSGRDT,(OSDAT,DUMMY)
                STD  0,GRDAT
                 .
                 .
          OSDAT DC   C'   71120'
          DUMMY DS   CL8
          GRDAT DS   0D,CL8
```

The above examples call OSGRDT to convert the OS date 71120 into the corresponding Gregorian date April 30, 1971.

FORTRAN:

```
REAL*8 OSDAT,GRDAT
CALL OSGRDT(OSDAT,GRDAT,&400)

REAL*8 GRDAT,OSGRDT,OSDAT,DUMMY
GRDAT=OSGRDT(OSDAT,DUMMY)
```

The above examples call OSGRDT to convert the OS date in the variable OSDAT into the corresponding Gregorian date.

PL/I(F):

```
CALL PLCALL(OSGRDT,F2,'   71120',GRDAT);
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE OSGRDT ENTRY,
        F2 FIXED BINARY(31) INITIAL(2),
        GRDAT CHARACTER(8);
PL1RC RETURNS (FIXED BINARY(31));

UNSPEC(GRDAT)=UNSPEC(PLCALLD(OSGRDT,F2,OSDAT,
        DUMMY));
IF PL1RC¬=0 THEN GO TO ERROR;
DECLARE GRDAT CHARACTER(8),
        PLCALLD RETURNS(FLOAT(16)),
        OSGRDT ENTRY,
        F2 FIXED BINARY(31) INITIAL(2),
        OSDAT CHARACTER(8) INITIAL('   71120'),
        DUMMY CHARACTER(8);
PL1RC RETURNS (FIXED BINARY(31));
```

The above examples call OSGRDT to convert the OS date 71120 into the corresponding Gregorian date April 30, 1971.

PAR

Subroutine Description

Purpose: To give a program access to the system parameter string given on the $RUN command.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL PAR,(reg,len,max)

FORTRAN: CALL PAR(reg,len,max,&rc4,&rc8)

Parameters:

- reg — is the location of a region into which the parameter string text will be placed. For FORTRAN programs, this should be declared as a LOGICAL*1 array.
- len — is the location of a fullword integer (INTEGER*4) which will be set to the actual number of characters placed in the region.
- max — is the location of a fullword integer (INTEGER*4) giving the maximum number of characters to be placed in the region. The PAR string may be from 0 to 255 characters in length.
- rc4,rc8 — (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

- 0 Successful return. Parameter string passed back.
- 4 No PAR string was given on $RUN command or the PAR string is currently of zero length. reg and len are left unchanged.
- 8 The actual length of the PAR string is greater than max. max characters are placed into reg and len is set equal to max.

Notes: This same information is also available from the PARSTR item of the GUINFO/CUINFO subroutine.

The PAR string subroutine converts the parameter string to uppercase. The PARSTR subroutine should be used to return the parameter string if uppercase conversion is not desired.

Examples: Assembly:

```
                CALL PAR,(PARREG,LPAR,MAX)
                C    15,=F'4'
                BL   OK             Successful return
                BE   NULLPAR        No PAR string
                BH   LONGPAR        Long PAR string
                 .
                 .
        PARREG  DS   CL100
        LPAR    DS   F
        MAX     DC   F'100'
```

FORTRAN:

```
        LOGICAL*1 PARREG(100)
        CALL PAR(PARREG,LPAR,100,&10,&20)
```

The above two examples retrieve the PAR string and place it into the array PARREG.

PARSTR

Subroutine Description

Purpose: To give a program access to the system parameter string given in the PAR field of the $RUN command.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL PARSTR,(reg,len,max,sws),VL

FORTRAN: CALL PARSTR(reg,len,max,sws,&rc4,&rc8,&rc12)

Parameters:

- reg is the location of a region into which the parameter string text will be placed. For FORTRAN programs, this should be declared as a LOGICAL*1 array.
- len is the location of a fullword integer (INTEGER*4) which will be set to the actual number of characters placed in the region.
- max is the location of a fullword integer (INTEGER*4) giving the maximum number of characters to be placed in the region. The PAR string may be from 0 to 255 characters in length.
- sws - is the location of a fullword of switches:

  bit 31: 0 - convert string to uppercase.
  1 - do not convert string.
  bits 0-30: unused, must be zero.

- rc4,...,rc12 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return. Parameter string passed back.
4 No PAR string was given on $RUN command or the PAR string is currently of zero length. reg and len are left unchanged.
8 The actual length of the PAR string is greater than max. max characters are placed into reg and len is set equal to max.
12 Illegal parameter or VL not specified.

Notes: This same information is also available from the PARSTRMC item of the GUINFO/CUINFO subroutine.

This subroutine is similar to the PAR subroutine except that it provides the option of not converting the parameter string to uppercase.

Examples: Assembly:

```
                 CALL PARSTR,(PARREG,LPAR,MAX,SWS),VL
                 C    15,=F'4'
                 BL   OK         Successful return
                 BE   NULLPAR    No PAR string
                 BH   LONGPAR    Long PAR string
                  .
                  .
         PARREG  DS   CL100
         LPAR    DS   F
         MAX     DC   F'100'
         SWS     DC   F'1'       Specify mixed-case string
```

FORTRAN:

```
      LOGICAL*1 PARREG(100)
      CALL PARSTR(PARREG,LPAR,100,1,&10,&20)
```

The above two examples retrieve the PAR string and place it into the array PARREG.

Pattern-Matching Routines

Three system subroutines, PATBUILD, PATMATCH, and PATFREE, are available for implementing $FILESTATUS-like pattern-matching capabilities from user programs.

PATBUILD will build a pattern from an input string. The input string may be of any length and may specify a file name or a generic string. For example:

```
File names:       "2CYB:data?", "?.doc", "2ABC:?"

Generic strings:  "Bill R?", "in the state of ?"
```

PATMATCH will compare an input string against the pattern built by PATBUILD. The input string may be of any length.

PATFREE will free the storage used to build the pattern.

These subroutines must be used together. The form of a typical program may be as follows;

```
PATBUILD(...)            Build the pattern

DO  WHILE <condition>    Loop
  READ (a string)        Get a string
  PATMATCH(...)          See if that string matches
  IF  Return-code = 0    A match
     ...
  ELSE                   No match
     ...
  ENDIF
ENDDO

PATFREE(...)             Done with the pattern
```

A complete example is given at the end of this description.

PATBUILD

Subroutine Description

Purpose: To scan a patterned input string (a string with zero or more wildcard characters) and construct a pattern that PATMATCH can use to match against other strings. The input string may be of any length and may specify a file name or a generic character string.

Location: Resident System

Alt. Entry: PATBLD

Calling Sequences:

```
Assembly: CALL PATBUILD,(patstring,strlen,work,switches,
                      ccid,chars),VL

FORTRAN:  CALL PATBLD(patstring,strlen,work,switches,
                    ccid,chars)
```

Parameters:

<u>patstring</u> is the location of an input string (that possibly contains wildcard characters).

<u>strlen</u> is the location of the fullword length of <u>patstring</u>. If <u>strlen</u> is given as -1, the length will be determined by this subroutine. In determining the length, it is assumed that <u>patstring</u> is followed by a delimiter.

<u>work</u> is the location of a fullword for use by PATBUILD. This area must be passed unchanged to the PATMATCH and PATFREE subroutines.

<u>switches</u> (optional) is the location of a fullword that contains switches as follows:

bit 31: 0 - <u>patstring</u> is a file name.
        1 - <u>patstring</u> is not a file name.
bit 30: 0 - This bit must be zero.
bit 29: 0 - Upper and lowercase for a string is not significant; i.e., match occurrences in both cases.
        1 - Use the string as it is given; case is significant.
bits 0-28: These bits must be zero.

If this parameter is omitted, _switches_ is assumed to be zero.

ccid (optional) is the location of a 4-character field that will be set to the ccid of the file name given in _patstring_. If omitted or if bit 31 of _switches_ is 1, no ccid is returned.

chars (optional) is the location of a 2-character string that contains the wildcard character and the pattern delimiter character (in that order) to be used in building the pattern. If omitted, the default values ("?" and " ") are used.

Return Codes:

0 A pattern was built since a string with wildcard characters in it was given in _patstring_.
4 _patstring_ specified a file name pattern indicating that all file names in a particular catalog will match (e.g., "2CRN:?", "-?").
8 _patstring_ had no wildcard characters.
20 Error return - a partially specified ccid (e.g., "W1??") was encountered in _patstring_.
24 Error return - _patstring_ specified a file name which is too long. That is, a file name longer than the allowed maximum file name length would be required to match it.
28 Error return - _patstring_ was not specified correctly or is missing.
32 Error return - invalid parameter address or bad parameter value (_strlen_=0, illegal _switches_ value, VL-bit not set, etc.).

Description: An input string will be built into a pattern. The string may contain wildcard characters (a character that will match arbitrary characters) and may be followed by a delimiter. The default wildcard character is a question mark ("?"). The default delimiter is a blank. The input string may specify a file name, in which case the delimiters are blank, comma, "(", "+", and "@" and may not be set to anything else.

Pattern matching rules:

(1) A single wildcard character will match zero or more arbitrary characters in a string. Thus,

A?Q?B will match all strings that begin with "A", end with "B", and contain the letter "Q".

(2) "n" consecutive wildcard characters will match "n-1" arbitrary characters in the string. Thus,

???.s matches all strings that are four characters long and end with ".s". The string "ab.s" will match while the strings "abc.s", "a.s", and ".s" will not.

(3) A wildcard character cannot be used in the signon ID portion of a shared file name.

(4) When strlen is given as -1, patstring is scanned up to the first delimiter in order to determine its length. When a specific length is given in strlen, any delimiter character encountered is ignored. For example, to build the pattern for "?day is tomorrow" (assuming blank as delimiter), strlen must be 16. If strlen is given as -1, then the pattern for "?day" only will be built since the first blank terminates the patterned string.

PATBUILD constructs a pattern only if its return code is 0, 4, or 8; otherwise a subsequent call to PATMATCH will generate a return code of 8 (no pattern to test). Note that when the return code from PATBUILD is 4 or 8, it may not be necessary to use PATMATCH since the pattern match in those cases is trivial; however, PATMATCH will work correctly if it is called.

Notes:

(1) When the chars parameter is included, both characters are assigned, and therefore both must be given. For example, if a user desires a delimiter character of "%", the character string should be "?%"; that is, the default wildcard character "?" must be included as the first element of the chars character string.
(2) When an optional parameter is desired, any optional parameters listed before the desired one must also be included.
(3) The "case bit" (bit 29) of switches is ignored when file names are being matched. Upper and lowercase is not considered significant for file names.
(4) The VL-bit is required on the parameter list.

PATMATCH

Subroutine Description

Purpose: To compare an input string against a pattern constructed earlier by PATBUILD. The input string may be of any length.

Location: Resident System

Alt. Entry: PATMCH

Calling Sequences:

Assembly: CALL PATMATCH,(compstr,strlen,work),VL

FORTRAN: CALL PATMCH(comstr,strlen,work)

Parameters:

- _compstr_ is the location of an input string to compare against the pattern.
- _strlen_ is the location of the fullword length of _compstr_. If _strlen_ is given as -1, the length of _compstr_ will be determined by scanning for the first delimiter. In that case, it is assumed that _compstr_ is followed by a delimiter.
- _work_ is the location of the fullword pattern work area. This _must_ be the value returned by PATBUILD.

Return Codes:

- 0 The string matched the pattern.
- 4 The string did not match the pattern.
- 8 Error return - there was no previous pattern to match; no match was made.
- 12 Error return - bad parameter; either a bad address was given, _compstr_ was empty, or VL-bit was not set.

Description: PATMATCH is used (in conjunction with PATBUILD and PATFREE) to determine if its input string fits a pattern built previously by PATBUILD. PATMATCH's behavior depends on the return code from PATBUILD as follows:

RC was 0: _compstr_ must fit the pattern built by PATBUILD. (Ccids must match exactly for file names.)

RC was 4: Ccid fields must match exactly; the file name is ignored.
RC was 8: compstr must match PATBUILD's patstring character for character. The ccids must match as well if file names are being matched.
RC was 20, 24, 28, 32: Error return - no pattern to match against.

Notes:

(1) See PATBUILD description for rules on pattern matching.
(2) compstr is assumed to be either a file name or a generic string depending on whether the pattern built by PATBUILD was for a file name or a generic string.
(3) The VL-bit is required on the parameter list.

PATFREE

Subroutine Description

Purpose: To free the storage used for building a pattern (see PATBUILD and PATMATCH descriptions).

Location: Resident System

Alt. Entry: PATFRE

Calling Sequences:

Assembly: CALL PATFREE,(work),VL

FORTRAN: CALL PATFRE(work)

Parameters:

work is the location of the fullword pattern work area. This _must_ be the value returned from PATBUILD.

Return Codes:

0 Successful return.
4 Illegal value in _work_ parameter or VL-bit not set. Storage was not released.

Note:

(1) The VL-bit is required on the parameter list.

The following example programs read input and decide if the input matches the pattern "?day". Note that in setting up the pattern, the blank after "?day" is necessary since we are calling PATBUILD with strlen as a -1. strlen could also be given the value 4 here. switches is set to 1 indicating that the pattern is not a file name and that upper/lower case is not significant for the purpose of pattern matching.

Assembly:

```
     MATCHIT CSECT
             REQU  TYPE=DEC
             ENTER R10,SA=SAVE


             MVC   PATTERN(5),=CL5'?day ' Set up the pattern
             MVC   STRLEN(4),=F'-1'    Let length be determined
             MVC   SWITCHES(4),=F'1'   Set switches

            Build the pattern

             CALL  PATBUILD,(PATTERN,STRLEN,WORK,SWITCHES),VL

            Read in strings for comparison and see if they match

             DO
               SCARDS COMPSTR,(R3)     Read in comparison string
               ST    R3,STRLEN         Store returned length
               IF    COMPSTR(4),EQ,'stop'
                 EXITDO ,              Quit when user types "stop"
               ENDIF
               CALL PATMATCH,(COMPSTR,STRLEN,WORK),VL A match?
               IF   R15,NZ             If no match
                 SPRINT 'No, it does not match.'
               ELSE  ,                 If a match
                 SPRINT 'Yes, it matches.'
               ENDIF
             ENDDO

            Free up the pattern work area

             CALL PATFREE,(WORK),VL    Return work area

             EXIT (15)                 Done.

     SAVE    DS  18F                   Register save area
     PATTERN DS  CL5                   Pattern string
     WORK    DS  A                     Work space
     COMPSTR DS  CL100                 Comparison string
     STRLEN  DS  F                     Length of comparison string
     SWITCHES DS F                     Type of pattern switch
             END
```

FORTRAN:

```
        INTEGER*4 COMLEN, WORK
        INTEGER*2 LEN
        CHARACTER*100 COMSTR
        CHARACTER*4 COMBEG
        CHARACTER*5 PATTRN
        EQUIVALENCE (COMSTR,COMBEG)
        DATA PATTRN/'?day '/
        CALL PATBLD(PATTRN,-1,WORK,1)
  1     CALL SCARDS(COMSTR,LEN,0,LNUM,*200)
        IF (COMBEG.EQ.'stop') GOTO 200
        COMLEN = LEN
        CALL PATMCH(COMSTR,COMLEN,WORK,*100,*100,*100)
        WRITE (6,10)
  10    FORMAT('Yes, it matches')
        GOTO 1
  100   WRITE (6,101)
  101   FORMAT('No, it does not match')
        GOTO 1
  200   CALL PATFRE(WORK)
        STOP
        END
```

Pascal/JB:

```
Program MATCHIT(Input,Output);

Type
  PATTERN_TYPE = Packed Array[1..10]  of Char;
  COMPSTR_TYPE = Packed Array[1..200] of Char;

Var
  PATTERN                 : PATTERN_TYPE;     { Pattern }
  COMPSTR                 : COMPSTR_TYPE;     { Comparison string }
  INPUT_TEXT              : String(200);      { User input }
  STRLEN, WORK, SWITCHES  : Integer;          { Parameters }

{  Pascal definitions for PATBUILD, PATMATCH, PATFREE.
   All parameters must be of type VAR for a FORTRAN type routine }

Procedure PATBUILD(VAR PATTERN:PATTERN_TYPE;
                   VAR STRLEN, WORK, SWITCHES :Integer); Fortran;
Procedure PATMATCH(VAR COMPSTR:COMPSTR_TYPE;
                   VAR STRLEN,WORK:Integer); Fortran;
Procedure PATFREE (VAR WORK:Integer);        Fortran;


Begin   { Main program }

  PATTERN := '?day ';             { Set up the pattern }
  STRLEN  := -1;                  { Length to be figured out }
```

```
  SWITCHES := 1;                  { Not a filename; anycase }

  PATBUILD (PATTERN,STRLEN,WORK,SWITCHES); { Build pattern }

  Reset(Input,'File=*source*,interactive'); { Read terminal }
  Readln  (INPUT_TEXT);           { Read 1st comparison string }

  While  INPUT_TEXT <> 'stop' Do  { Continue until "stop" }
  Begin
    STRLEN := Length(INPUT_TEXT); { Get length of input }
    COMPSTR := INPUT_TEXT;        { Move text to array }

    PATMATCH (COMPSTR,STRLEN,WORK); { See if a match }
    If    FortranRC = 0 Then        { 0 => a match }
      Writeln ('Yes, it matches.')
    Else                            { >0 => no match }
      Writeln ('No, it does not match.');

    Readln (INPUT_TEXT)           { Next comparison string }
  End;

  PATFREE (WORK)                  { Free up the pattern }

End.
```

The following is an example run of the above programs. Program output is underlined.

```
$RUN program
 Doris Day
 Yes, it matches.
 Tuesday
 Yes, it matches.
 This is a nice day
 Yes, it matches.
 DAY
 Yes, it matches.
 Dayton
 No, it does not match.
 stop
```

PERMIT

Subroutine Description

Purpose: To permit a file so that it can be shared by other users.

Location: Resident System

Calling Sequences:

Assembly:
```
CALL PERMIT,(what,how,whotyp,wholen,who,info,
             wholen2,who2,ercode,errmsg),VL
```

FORTRAN:
```
CALL PERMIT(what,how,whotyp,wholen,who,info,
            wholen2,who2,ercode,errmsg,&rc4,&rc8)
```

Parameters:

<u>what</u> is the location of either
(a) a file name with trailing blank (if <u>info</u>=0),
(b) a fullword-integer FDUB-pointer (such as returned by GETFD) (if <u>info</u>=1),
(c) a fullword-integer logical I/O unit number (0 through 99) (if <u>info</u>=1), or
(d) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if <u>info</u>=1).

<u>how</u> is the location of a fullword integer specifying the access. There are six independent accesses; add the values below for the combinations wanted.

Access	Value
Read	1
Write-expand	2
Write-change,empty	4
Truncate, renumber	8
Destroy, rename	16
Permit	32
Default	128

Some popular combinations are:

None	0
Write	6
Read-write	7
Unlimited	63

This parameter is ignored for _whotype_=9.

whotyp is the location of a fullword integer whose value indicates what sort of _who_ is being specified, as follows:

	Value
who is a signon ID	0
who is a project number	1
who is OTHERS	2
who is ALL	3
who is ME	4
who is OWNER	5
who is a program key	6
who is a signon ID and program key	7
who is a project number and program key	8
who is a how/who string	9

wholen is the location of a fullword integer which specifies the number of characters in the signon ID or project number (1 to 4) specified by _who_ (for _whotype_=0,1,7, or 8), the number of characters in the program key (1 to 13) specified by _who_ (for _whotype_=6), or the number of characters in the how/who string (for _whotype_=9).

who is the location of the 1- to 4-character signon ID or project number (for _whotype_=0,1, 7, or 8), the 1- to 13-character program key (for _whotype_=6), or the how/who string (for _whotype_=9). Short signon IDs, project numbers, program keys, and how/who strings may end with a trailing question mark.

info is the location of a fullword integer that specifies the kind of _what_ parameter supplied.

wholen2 is the location of a fullword integer which specifies the number of characters in the program key (1 to 13) specified by _who2_. This parameter is present only when _whotype_=7 or 8.

who2 is the location of the 1- to 13-character program key. This parameter is present only when _whotype_=7 or 8. Short program keys may end with a trailing question mark.

ercode (optional) is the location of a fullword in which the PERMIT subroutine will place an error number if an error return (return code 4) is made. If this parameter is omitted, then the _errmsg_ parameter must also be omitted. Assembly code users who wish to omit these parameters should either follow the variable parameter list convention (high-order bit of the previous parameter's adcon in the parameter list should be 1) or else supply an adcon which is zero (rather than pointing to a zero).

Error numbers less than 100 indicate something was wrong with either the mechanics of the subroutine call or the values of the parameters:

<u>Number</u>	<u>Message</u>
1	Illegal parameter list pointer
2	Illegal "what" parameter address
3	Illegal "how" parameter address
	Illegal "onoff" address
	Illegal "pkey" address
4	"How" parameter value not 0 to 63 or 128
	"onoff" parameter value not 0 or 1
	Illegal program key "xxxx"
5	Illegal "whotype" parameter address
6	"Whotype" parameter value not 0 to 9
7	Illegal "wholen" parameter address
8	Bad "wholen" parameter value
9	Illegal "who" parameter address
10	Illegal "info" parameter address
11	"Info" parameter value not 0 to 1
12	Illegal "wholen2" parameter address
13	Bad "wholen2" parameter value
14	Illegal "who2" parameter address
15	Illegal program key

Error numbers between 100 AND 200 describe errors common to the $PERMIT command:

101	Illegal file name "xxxx" or "what" parameter
102	File not found - file "xxxx"
103	Access not allowed to file "xxxx" (Permit access required to permit a file.)
104	Deadlock situation, try later - file "xxxx"
105	Interrupted out of wait for locked file "xxxx"
106	"Default Others" does not do anything
107	Illegal character in CCID or Project - "xxxx"
108	Invalid combination of NONE with other access

109 Invalid combination of DEFAULT with other access
110 Invalid combination of ALL with other accessors
111 Invalid CCID "xxxx"
112 Invalid project "xxxx"
113 Pkey cannot be used in combination with RUN or EDIT access
114 Invalid access/accessor specification
115 Invalid operand "xxxx"
116 Expected access specification missing
Invalid use of PKEY with ALL or OTHERS
117 Missing file name
118 Missing closing parenthesis

Error numbers 201 and above indicate a file system error of some sort.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from PERMIT with an error code of 105.

errmsg (optional) is the location of a 20-fullword (80-character) region in which the PERMIT subroutine will place the corresponding error message if an error return (return code 4) is made. Assembly language users should see the previous instructions on omitting optional parameters for the ercode parameter.

rc4,rc8 (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 The file has been permitted in the requested manner.
4 Error. The file has not been permitted. See the ercode and errmsg values returned for the specific error.
8 Illegal errmsg or ercode parameter.

Examples: Assembly:

```
                 CALL PERMIT,(WHAT,HOW,WHOTYP,WHOLEN,WHO,
                           INFO,ERCODE,ERRMSG),VL
                  .
                  .
                  .
         WHAT    DC  C'PROB1DATA '
         HOW     DC  F'1'
```

```
WHOTYPE DC  F'1'
WHOLEN  DC  F'3'
WHO     DC  C'2AA'
INFO    DC  F'0'
ERCODE  DS  F
ERRMSG  DS  CL80
```

FORTRAN:
```
CALL PERMIT('PROB1DATA ',1,1,3,'2AA',0)
```

The above examples permit the file PROB1DATA for read access by all users whose project number begins with the three characters 2AA.

Assembly:
```
          CALL PERMIT,(WHAT,HOW,WHOTYP,WHOLEN,WHO,
                       INFO,ERCODE,ERRMSG),VL
           .
           .
           .
WHAT    DC  C'PROB1DATA '
HOW     DS  F
WHOTYPE DC  F'9'
WHOLEN  DC  F'11'
WHO     DC  C'READ P=2AA?'
INFO    DC  F'0'
ERCODE  DS  F
ERRMSG  DS  CL80
```

FORTRAN:
```
CALL PERMIT('PROB1DATA ',0,9,11,
            'READ P=2AA?',0)
```

The above examples are similar to the first set except that the how/who access is specified by a character string (whotype=9).

PGNTTRP

Subroutine Description

Purpose: To allow control to be returned to the user on a program interrupt.

Location: Resident System

Alt. Entries: PGNTT, PGNTTRPS, PGNTPS

Calling Sequences:

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

          CALL PGNTTRPS,(exit,region),VL

FORTRAN:  CALL PGNTPS(exit,region,&rc4)
```

Parameters:

exit (GR0) should be zero or the location to transfer to if a program interrupt occurs.

region (GR1) should should contain the location of a 72-byte save region for storing pertinent information.

_&rc4_ (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Illegal parameter or no VL bit specified.

Description: A call on the subroutine PGNTTRP sets up a program interrupt intercept for one interrupt only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of a program interrupt. When an interrupt occurs and the exit is taken, the intercept is cleared so that another call to PGNTTRP is necessary to intercept the next program interrupt. When a program interrupt occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location previously specified. If the exit subroutine returns to MTS (BR 14), MTS will handle the interrupt as if PGNTTRP had not been called originally. This feature allows the user to take brief control of the interrupt before MTS takes complete control of the interrupt. When MTS takes control of the interrupt, execution of the program will be terminated and a message will be printed providing the location of the interrupt.

If GR0 is zero on a call to PGNTTRP, the program interrupt intercept is disabled. GR1 should be zero or point to a valid save region.

When the program interrupt exit is taken, the first eight bytes of the save region contain the program interrupt PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The PSW stored in the savearea is always in BC mode (bit 12 is zero). The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region. The contents of GR0 and GR2 to GR12 are unpredictable.

If, on a call to PGNTTRP, the first byte of the save region is X'FF', PGNTTRP does not return to the calling program; rather the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on PGNTTRP, after the first program interrupt exit is taken, allows the user to set a switch (for example) and to return to the point at which he was interrupted with the program interrupt intercept again enabled.

The PGNTTRP item of the GUINFO/CUINFO subroutine may be used to save a previous exit address and associated region so that it may be later restored.

| A call on the PGNTTRPS or PGNTPS subroutines takes the S-type parameters and loads them into an R-type call on the PGNTTRP subroutine.

Example: In this example, the program interrupt intercept is enabled for a specified portion of the program. When the interrupt occurs, a branch will be made to the label EXIT where a switch will be set marking the interrupt occurrence. The intercept will be reenabled by a second call to PGNTTRP with the FF flag set, and a branch will be made back to the point where the interrupt occurred.

```
         LM   0,1,=A(EXIT,REGION)
         CALL PGNTTRP   The intercept is enabled.
          ...
         SR   0,0
         SR   1,1
         CALL PGNTTRP   The intercept is disabled.
          ...
         USING EXIT,15
EXIT     OI   SW,X'01'
         MVI  0(1),X'FF'
         LA   0,EXIT
         CALL PGNTTRP   The intercept is reenabled.
REGION   DS   18F
SW       DC   X'00'
```

PKEY

Subroutine Description

Purpose: To push and pop program keys.

Location: Resident System

Calling Sequences:

Assembly: CALL PKEY,(string,pkey),VL

FORTRAN: CALL PKEY(string,pkey,&rc4,&rc8,&rc12,&rc16)

Parameters:

string is the location of a command (see below) terminated with a trailing blank.
pkey (optional) is the location of a new program key terminated with a trailing blank.
rc4,...,rc16 (optional) are statement labels to transfer to if a nonzero return code occurs.

VL must be specified even if both parameters are given in order to facilitate the addition of new parameters.

Return Codes:

0 Successful return.
4 Invalid command string.
8 Invalid program key.
12 Attempt to push or pop too many times.
16 Invalid parameters.

Description: The legal command strings are:

PUSH The current program key is pushed onto the stack of program keys and the new program key is made the current key. If no new program key is specified, the current program key is pushed onto the stack, but remains the current key.

POP The program key on the top of the stack is made the current program key and is removed from the stack. The pkey parameter is not required.

SET The new program key is made the current program key. The old program key is not pushed onto the stack of program keys.

RESET The stack of program keys is cleared, and the current program key is reset to its original value.

Currently, user programs may only specify the program key *EXEC in addition to the program key assigned to the file being executed. This will be expanded to include other program keys in the future.

Example: FORTRAN:

```
      CALL FTNCMD('ASSIGN 99=WXYZ:LOGFILE; ')
      CALL PKEY('PUSH ','*EXEC ')
       ...
      CALL PKEY('POP ')
      WRITE(99,100) A,B,C
100   FORMAT(3F10.2)
      CALL PKEY('PUSH ','*EXEC ')
       ...
```

The above example assigns FORTRAN I/O unit 99 to the file WXYZ:LOGFILE, which is permitted to a program key. When the program writes into this file, the PKEY subroutine is called to switch the program key from *EXEC to the program key of the file, and subsequently is called to restore the program key back to *EXEC.

<u>POINT</u>

Subroutine Description

Purpose: To alter the values of any or all of the logical pointers for a sequential file.

Location: Resident System

Alt. Entry: POINT#

Calling Sequences:

Assembly: CALL POINT,(unit,info,code)

FORTRAN: CALL POINT(unit,info,code,&rc4,&rc8,&rc12,&rc16,
&rc20,&rc24)

Parameters:

<u>unit</u> is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

<u>info</u> is the location of a region of four fullwords from which the POINT subroutine will set any or all of the logical pointers according to the value of <u>code</u>. The region contains the pointers in the same order as returned by the NOTE subroutine, that is, the Read, Write, and Last Pointers as well as the last line number, respectively.

<u>code</u> is the location of a fullword containing a value from 1 to 15 indicating which of the 4 logical pointers should be set. The conventions are as follows:

1 Set Read Pointer
2 Set Write Pointer
4 Set Last Pointer
8 Set last line number

These values should be added for multiple action, i.e., 7 means to set the Read, Write and Last Pointers only.

<u>rc4,...,rc24</u> are the statement labels to transfer to if a nonzero return code is encountered.

Return Codes:

0 Successful return.
4 Illegal FDUB-pointer specified.
8 Illegal parameter specified.
12 Read or write access not allowed.
16 Locking the file appropriately will result in a deadlock.
20 Hardware error or software inconsistency encountered.
24 Automatic wait for shared file was interrupted.

Notes: If any of the first three values of the region info are set to zero and the POINT subroutine is called, the effect will be to reset the indicated pointers (Read, Write and/or Last depending on the value of code) to the beginning of the file.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from POINT with a return code of 24.

Description: See Appendix B of the section "Files and Devices" in MTS Volume 1, The Michigan Terminal System, for details concerning using sequential files with the NOTE and POINT subroutines.

Examples: Assembly:

```
          CALL POINT,(UNIT,INFO,CODE)
            .
            .
UNIT  DC   F'6'
INFO  DS   4F
CODE  DC   F'7'
```

FORTRAN:

```
INTEGER*4 UNIT,INFO(4)
DATA UNIT/6/
 ...
CALL POINT(UNIT,INFO,7)
```

These examples call POINT (assuming that the NOTE subroutine was called previously) for the sequential file attached to logical I/O unit 6. The CODE parameter (7) specifies that the pointers are to be set for Read, Write, and Last.

Printer Plot Routines

Subroutine Description

Purpose: To produce plots in the normal output stream.

Location: *LIBRARY

Entry Points: The printer plot routines have the following entry points: PLOT1, PLOT2, PLOT3, PLOT4, PLOT14, PRCHAR, PREND, PRPLOT, STPLT1, STPLT2, OMIT, and SETLOG. The standard approach to produce a plot is to call PLOT1, PLOT2, PLOT3, and PLOT4 in that order. PLOT2 must be called for each plot to be produced.

Logical I/O Units Referenced:

SPRINT - Output from the printer plot routines (the plot). Note: When the printer is used as the SPRINT device, a page skip is normally issued _by the user_ before calling PLOT4 in order to force a skip to the top of the next page before starting the plot.

SERCOM - Error messages.

Example: FORTRAN:

```
      DIMENSION IMAGE(1500)
      INTEGER NSC(5)/1,0,3,0,2/
      DATA BCD/'*   '/
      CALL PLOT1(NSC,11,3,11,5)
      CALL PLOT2(IMAGE,1.0,-1.0,1.0,-1.0)
      DO 20 I=1,60
      DO 20 J=1,40
      X = (I-30.)/30.
      Y = (J-20.)/20.
      IF (X**2+Y**2.GT.0.75**2) GO TO 20
      CALL PLOT3(BCD,X,Y,1,4)
   20 CONTINUE
      CALL PLOT4(14,'VERTICAL LABEL')
      STOP
      END
```

The above FORTRAN program will produce the plot given on the following page.

```
            1.000 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
                  I     I     I     I     I     I     I     I     I     I     I
                  I     I     I     I     I     I     I     I     I     I     I
                  I     I     I     I     I     I     I     I     I     I     I
            0.800 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
                  I     I     I     I     I     *     I     I     I     I     I
                  I     I     I     I   *****************   I     I     I     I
                  I     I     I     I***********************I     I     I     I
            0.600 +-----+-----+----***************************----+-----+-----+
                  I     I     I  *******************************  I     I     I
                  I     I     I ********************************* I     I     I
                  I     I     *************************************     I     I
            0.400 +-----+----***************************************----+-----+
       V          I     I    ***************************************    I     I
       E          I     I   *****************************************   I     I
       R          I     I  *******************************************  I     I
       T    0.200 +-----+--*******************************************--+-----+
       I          I     I ********************************************* I     I
       C          I     I ********************************************* I     I
       A          I     I ********************************************* I     I
       L    0.000 +-----+-*********************************************-+-----+
                  I     I ********************************************* I     I
       L          I     I ********************************************* I     I
       A          I     I ********************************************* I     I
       B   -0.200 +-----+--*******************************************--+-----+
       E          I     I  *******************************************  I     I
       L          I     I   *****************************************   I     I
                  I     I    ***************************************    I     I
           -0.400 +-----+----***************************************----+-----+
                  I     I     *************************************     I     I
                  I     I     I ********************************* I     I     I
                  I     I     I  *******************************  I     I     I
           -0.600 +-----+-----+----***************************----+-----+-----+
                  I     I     I     I***********************I     I     I     I
                  I     I     I     I   *****************   I     I     I     I
                  I     I     I     I     I     *     I     I     I     I     I
           -0.800 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
                  I     I     I     I     I     I     I     I     I     I     I
                  I     I     I     I     I     I     I     I     I     I     I
                  I     I     I     I     I     I     I     I     I     I     I
           -1.000 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
                -1.00 -0.80 -0.60 -0.40 -0.20 -0.00  0.20  0.40  0.60  0.80 1.0
```

FORTRAN:

```
      REAL ARG/0./,X(61),YSIN(61),YCOS(61)
      REAL PI60/.0523599/
      INTEGER CSIN/'*   '/,CCOS/'%   '/
      INTEGER NSCALE(5)/1,0,3,0,0/
      CALL PLOT1(NSCALE(1),11,3,11,5)
      CALL PLOT2(0,180.,0,1.,-1.)
      X(1) = 0.
      YSIN(1) = 0.
      YCOS(1) = 1.
      DO 1 I = 2,61
        X(I) = X(I-1) + 3.
        ARG = ARG + PI60
        YSIN(I) = SIN(ARG)
1       YCOS(I) = COS(ARG)
      CALL PLOT3(CSIN,X(1),YSIN(1),61,4)
      CALL PLOT3(CCOS,X(1),YCOS(1),61,4)
      CALL PLOT4(11,'SIN AND COS')
      CALL SYSTEM
      END
```

The above FORTRAN program will produce the plot given on the following page.

```
            1.000 %%%%%-+-----+-----+-----+-*********-+-----+-----+-----+-----+
                  I    %%%    I     I    ***    I     ***    I     I     I     I
                  I     I %%  I     I  ** I     I     I **  I     I     I     I
                  I     I   %%I     I**   I     I     I   **I     I     I     I
            0.800 +-----+-----%%---**-----+-----+-----+-----**----+-----+-----+
                  I     I     I % * I     I     I     I     I *   I     I     I
                  I     I     I  %  I     I     I     I     I  *  I     I     I
                  I     I     I** %%I     I     I     I     I   **I     I     I
            0.600 +-----+-----*-----%-----+-----+-----+-----+-----*-----+-----+
                  I     I    *I     I%    I     I     I     I     I*    I     I
                  I     I   * I     I %   I     I     I     I     I *   I     I
                  I     I  *  I     I  %  I     I     I     I     I  *  I     I
            0.400 +-----+-*---+-----+---%-+-----+-----+-----+-----+---*-+-----+
                  I     I*    I     I    %I     I     I     I     I    *I     I
                  I     *     I     I     %     I     I     I     I     *     I
S                 I    *I     I     I     I%    I     I     I     I     I*    I
I           0.200 +---*-+-----+-----+-----+-%---+-----+-----+-----+-----+-*---+
N                 I  *  I     I     I     I  %  I     I     I     I     I  *  I
                  I *   I     I     I     I   % I     I     I     I     I   * I
A                 I*    I     I     I     I    %I     I     I     I     I    *I
N           0.000 *-----+-----+-----+-----+-----%-----+-----+-----+-----+-----*
D                 I     I     I     I     I     I%    I     I     I     I     I
                  I     I     I     I     I     I %   I     I     I     I     I
C                 I     I     I     I     I     I  %  I     I     I     I     I
O          -0.200 +-----+-----+-----+-----+-----+---%-+-----+-----+-----+-----+
S                 I     I     I     I     I     I    %I     I     I     I     I
                  I     I     I     I     I     I     %     I     I     I     I
                  I     I     I     I     I     I     I%    I     I     I     I
           -0.400 +-----+-----+-----+-----+-----+-----+-%---+-----+-----+-----+
                  I     I     I     I     I     I     I  %  I     I     I     I
                  I     I     I     I     I     I     I   % I     I     I     I
                  I     I     I     I     I     I     I    %I     I     I     I
           -0.600 +-----+-----+-----+-----+-----+-----+-----%-----+-----+-----+
                  I     I     I     I     I     I     I     I%%   I     I     I
                  I     I     I     I     I     I     I     I  %  I     I     I
                  I     I     I     I     I     I     I     I   % I     I     I
           -0.800 +-----+-----+-----+-----+-----+-----+-----+----%%-----+-----+
                  I     I     I     I     I     I     I     I     I%%   I     I
                  I     I     I     I     I     I     I     I     I  %% I     I
                  I     I     I     I     I     I     I     I     I    %%%    I
           -1.000 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-%%%%%
                 0.   18.   36.   54.   72.   90.  108.  126.  144.  162.  180.
```

PLOT1

Purpose: PLOT1 sets up the information required to construct the plot.

Calling Sequences:

Assembly: CALL PLOT1,(nscale,nhl,nsbh,nvl,nsbv)

FORTRAN: CALL PLOT1(nscale(1),nhl,nsbh,nvl,nsbv,&rc4)

Parameters:

<u>nscale</u> is the location of a region of five fullword integers supplying information about scaling and the number of places to be printed to the right of the decimal point. The field width for printing Y values is 8, and for X values is min(nsbv,8).

- <u>nscale(1)</u> If <u>nscale(1)</u>=0, the values 0,3,0,3 are used for <u>nscale(2)</u> through <u>nscale(5)</u>.
- <u>nscale(2)</u> If <u>nscale(2)</u>=Y, the numbers printed along the Y-axis are 10**Y times their true value.
- <u>nscale(3)</u> The number of decimal places printed for Y values.
- <u>nscale(4)</u> If <u>nscale(4)</u>=X, the numbers printed along the X-axis are 10**X times their true values.
- <u>nscale(5)</u> The number of decimal places printed for X values.

<u>nhl</u> is the location of a fullword integer giving the number of horizontal lines in the plot. This number must be 2 or greater.

<u>nsbh</u> is the location of a fullword integer giving the number of spaces between horizontal lines. This number must be 1 or greater.

<u>nvl</u> is the location of a fullword integer giving the number of vertical lines in the plot. This number must be 2 or greater.

<u>nsbv</u> is the location of a fullword integer giving the number of spaces between the vertical lines. This number must be 1 or greater.

<u>rc4</u> (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Normal return.
4 Improper Argument. PLOT1 has not been entered.

PLOT2

Purpose: PLOT2 prepares the grid and sets up the information required by PLOT3 to place a point correctly in the graph.

Calling Sequences:

Assembly: CALL PLOT2,(image,xmax,xmin,ymax,ymin)

FORTRAN: CALL PLOT2(image,xmax,xmin,ymax,ymin,&rc4,&rc8)

Parameters:

- image — is either the location of a zero or the location of a region equal to or greater in length than

  (nsbh*nhl-nsbh+nhl)*(nsbv*nvl-nsbv+nvl+8)+8

  bytes. This region is used to form the image of the graph.
- xmax — is the location of the largest X value of the points to be plotted.
- xmin — is the location of the smallest X value of the points to be plotted.
- ymax — is the location of the largest Y value of the points to be plotted.
- ymin — is the location of the smallest Y value of the points to be plotted.
  Note: The preceding four arguments are either short or long floating-point numbers.
- rc8 — (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Normal return.
8 xmax ≤ xmin or ymax ≤ ymin. PLOT2 has not been entered.

Description: If PLOT1 has not been entered by the time PLOT2 is called, defaults are assumed for nscale, nhl, nsbh, nvl, and nsbv. In particular, nscale=0, nhl=6, nsbh=9, and nsbv=9. The value of nvl depends on the SPRINT device; for a printer, nvl=11, and for a Teletype, nvl=6.

If a zero is specified for image, then PLOT2 will automatically allocate sufficient space for the image region. On successive calls to PLOT2, space will released and reallocated as needed.

PLOT3

Purpose: PLOT3 places the plotting character in the graph for each point (X,Y).

Calling Sequences:

Assembly: CALL PLOT3,(bcd,x,y,ndata,int)

FORTRAN: CALL PLOT3(bcd,x,y,ndata,int,&rc4,&rc8,&rc12,
&rc16)

Parameters:

bcd is the location of the plotting character to be used.
_x_ is the location of a floating-point region of X values.
_y_ is the location of a floating-point region of Y values.
ndata is the location of the fullword integer number of points to be plotted.
int is the location of the fullword integer number of bytes between the addresses of successive numbers to be used as coordinates. For a short form vector, this is 4. _int_ should be a multiple of 4.
_rc12,rc16_ (optional) are the statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Normal return.
12 Using a log scale with a negative or zero _xmin_, _xmax_,_ymin_, _ymin_, or _ymax_ value, or, _int_ not a multiple of 4.
16 PLOT2 has never been entered, or has not been entered since the last call to PLOT4.

PLOT4

Purpose: PLOT4 prints the completed graph with values along the X- and Y-axes and a centered vertical label down the left side.

Calling Sequences:

Assembly: CALL PLOT4,(nchar,label)

FORTRAN: CALL PLOT4(nchar,label,&rc4,&rc8,&rc12,&rc16,
&rc20,&rc24,&rc28)

Parameters:

<u>nchar</u> is the location of the fullword integer number of characters in the vertical label. If this is zero, no label will be printed.

<u>label</u> is the location of a region containing the label to be printed.

<u>rc20,rc24,rc28</u> (optional) are statement labels to transfer to if a nonzero return code occurs. encountered.

Return Codes:

0 Normal return.
20 PLOT2 has not been entered.
24 Using a log scale with a negative or zero <u>xmin</u>, <u>xmax</u>,<u>ymin</u>, or <u>ymax</u> value (see SETLOG and PLOT2).
28 Error in scaling; one or more values can not be printed in the form specified by <u>nscale</u> (see PLOT1).

Description: See OMIT for the possibility of deleting grid values and the last line of the graph.

If return code 28 is given, the plot will be printed with all grid values which can be printed.

PLOT14

Purpose: PLOT14 allows the user to combine successive calls on PLOT1, PLOT2, PLOT3, and PLOT4 into one call on PLOT14.

Calling Sequences:

```
Assembly: CALL PLOT14,(nscale,nhl,nsbh,nvl,nsbv,image,
                       xmax,xmin,ymax,ymin,bcd,x,y,ndata,
                       int,nchar,label)

FORTRAN:  CALL PLOT14(nscale(1),nhl,nsbh,nvl,nsbv,image,
                      xmax,xmin,ymax,ymin,bcd,x,y,ndata,
                      int,nchar,label,&rc4,&rc8,&rc12,
                      &rc16,&rc20,&rc24,&rc28)
```

Parameters:

See the descriptions of PLOT1, PLOT2, PLOT3, and PLOT4 for the parameters and return codes used.

Description: This routine executes the appropriate calls on PLOT1, PLOT2, PLOT3, and PLOT4.

PRCHAR

Purpose: PRCHAR allows the user to change the characters used in printing the grid.

Calling Sequences:

Assembly: CALL PRCHAR,(arg)

FORTRAN: CALL PRCHAR(arg)

Parameter:

arg is the location of a fullword integer whose bytes are used to define the grid character. The bytes are used as follows:

byte 0: intersection character (initially +)
byte 1: horizontal line character (initially -)
byte 2: vertical line character (initially I)
byte 3: fill character (initially blank)

A X'00' in any byte indicates that no change is to be made to that character.

Return Code:

None.

Description: Changes made by a call to this subroutine affect all plots starting with the next call to PLOT2, STPLT1, STPLT2, or PREND.

Example: FORTRAN:

```
INTEGER CHARS/Z00004F00/
 ...
CALL PRCHAR(CHARS)
```

The above example changes the vertical line character to "|" (vertical bar), and leaves the other three characters unchanged.

PREND

Purpose: PREND constructs and prints a plot using the points saved by PRPLOT. Values are printed along the X- and Y-axes, and a centered label is printed on the left-hand side. See the description of PRPLOT.

Calling Sequences:

Assembly: CALL PREND,(nchar,label)

FORTRAN: CALL PREND(nchar,label,&rc4,&rc8)

Parameters:

- nchar — is the location of a fullword integer giving the number of characters in the vertical label. If this is less than or equal to zero, no label will be printed.
- label — is the location of a region containing the label.
- rc4,rc8 — (optional) are the statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Normal return.
4 PRPLOT has not been successfully called.
8 Log argument ≤ 0 (occurs only when a log scale is used).

PRPLOT

Purpose: PRPLOT collects points to be plotted by a subsequent call to PREND.

Calling Sequences:

Assembly: CALL PRPLOT,(bcd,x,y,ndata,int)

FORTRAN: CALL PRPLOT(bcd,x,y,ndata,int,&rc4)

Parameters:

- _bcd_ is the location of the plotting character to be used.
- _x_ is the location of a floating-point region of X values.
- _y_ is the location of a floating-point region of Y values.
- _ndata_ is the location of the fullword integer number of points.
- _int_ is the location of the fullword integer number of bytes between the addresses of successive coordinate values. For a short form vector (REAL*4), this is 4. _int_ should be a multiple of 4.
- _rc4_ (optional) is the statement label to transfer to if a a nonzero return code occurs.

Return Codes:

0 Normal return.
4 _int_ is not a multiple of 4.

Description: PRPLOT saves points to be plotted; PREND determines the minima and maxima and constructs the actual plot. PRPLOT may be called many times before calling PREND. PRPLOT allows the user to obtain a printer plot without knowing in advance how many points will be accumulated or what the minimum and maximum X and Y values will be. It is _least_ efficient (in terms of CPU time) to call PRPLOT for one point at a time. When plotting in log mode, points for which the logarithm is undefined will be ignored.

Example: FORTRAN:

```
      REAL X(10),Y(10)
      INTEGER LABEL(2),/'A LA','BEL'/
      X(1) = 1.
      Y(1) = 2.
      DO 1 I=2,10
         X(I) = X(I-1)+1.
1        Y(I) = 2.*X(I)
```

```
          CALL PRPLOT('*',X(1),Y(1),3,4,&4)
          CALL PRPLOT('<',X(4),Y(4),7,4,&4)
          CALL PREND(7,LABEL(1))
          CALL SYSTEM
    4     CALL ERROR
```

STPLT1

Purpose: STPLT1 is called by the user who wishes the plot routine to inspect his data and then make appropriate calls on PLOT1 and PLOT2. The default grid size (see PLOT2) is always used, but the scaling and decimal places to be printed are determined by STPLT1. The user must call on PLOT3 and PLOT4 to have the graph printed.

Calling Sequences:

Assembly: CALL STPLT1,(image,x,y,ndata,int)

FORTRAN: CALL STPLT1(image,x,y,ndata,int,&rc4,&rc8,
&rc12,&rc16,&rc20,&rc24,&rc28)

Parameters:

See the descriptions of PLOT1, PLOT2, PLOT3, and PLOT4 for the parameters and return codes used.

Description: STPLT1 will cause grid values to be printed in FORTRAN E-type format when necessary.

STPLT2

Purpose: STPLT2 does the work of STPLT1 and in addition calls on PLOT3 and PLOT4 to print the graph.

Calling Sequences:

```
Assembly: CALL STPLT2,(image,x,y,ndata,int,bcd,nchar,
                        label)

FORTRAN:  CALL STPLT2(image,x,y,ndata,int,bcd,nchar,label,
                      &rc4,&rc8,&rc12,&rc16,&rc20,&rc24,
                      &rc28)
```

Parameters:

See the descriptions of PLOT1, PLOT2, PLOT3, PLOT4, and STPLT1 for the parameters and return codes used.

SETLOG

Purpose: SETLOG is called by the user to specify whether he wants a normal, semi-log, or log-log plot.

Calling Sequences:

Assembly: CALL SETLOG,(arg)

FORTRAN: CALL SETLOG(arg,&rc4)

Parameters:

<u>arg</u> is the location of a byte with bits 6 and 7 interpreted as follows:

bit 7	0	Y scale is normal.
	1	Y scale is logarithmic.
bit 6	0	X scale is normal.
	1	X scale is logarithmic.

The plotting mode is initially set to normal.

<u>rc4</u> (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Normal return.
4 Mode not changed.

Description: If PLOT2 or STPLT1 has been called, but the graph has not yet been printed by PLOT4, or if PRPLOT has been called, and has not yet been followed by a call to PREND, the plotting mode will not be changed. This is because the grid has already been set up. Base 10 logarithms are used for the grid.

Example: FORTRAN:

```
LOGICAL*1 XLOG/Z02/,YLOG/Z01/,XYLOG/Z03/
 ...
CALL SETLOG(XLOG)   Plot with log X, normal Y
 ...
CALL SETLOG(YLOG)   Plot with log Y, normal X
 ...
CALL SETLOG(XYLOG)  Log-log plot
 ...
CALL SETLOG(0)      Normal plot
```

OMIT

Purpose: OMIT is called by the user to specify whether the last graph line, the vertical grid values, and the horizontal grid values will be printed.

Calling Sequences:

Assembly: CALL OMIT,(arg)

FORTRAN: CALL OMIT(arg)

Parameters:

arg is the location of a fullword integer interpreted as follows: if _arg_ is positive, the function designated by the appropriate bit is turned off. To turn it back on, _arg_ is made negative and OMIT is called again.

bit 28	scaling factor messages (PRPLOT, STPLT1 only).
bit 29	the last graph line.
bit 30	vertical grid values.
bit 31	horizontal grid values.

Return Code:

None.

Description: A graph can be produced by producing the graph in pieces, deleting the horizontal grid values and the last graph line (_arg_=5) for each piece except the last, and starting the next graph segment where the last graph line would have been printed. When the last segment is to be printed, OMIT can be called (_arg_=-5) to restore the functions. Initially, all four functions are turned on.

If STPLT1 or PRPLOT scales the X or Y values, a message is normally printed stating what was done. Bit 28 of _arg_ controls the printing of this message.

QUIT

Subroutine Description

Purpose: To cause the user to be signed off when the next MTS command is encountered.

Location: Resident System

Calling Sequences:

Assembly: CALL QUIT

or

QUIT [WHO={BATCH|ALL},][WHEN={NOW|LATER}]

FORTRAN: CALL QUIT

Return Codes:

None.

Note: The complete description for using the QUIT macro is given in MTS Volume 14, _360/370 Assemblers in MTS_. Additional parameters may be given to the QUIT macro to control whether the subroutine is called in batch mode only and whether the effect is immediate.

Description: This subroutine does _not_ cause the user to be signed off immediately. It does set a flag so that the next time the user returns to MTS command mode (due to termination of execution, attention interrupt, etc.) the effect will be the same as if the user entered a $SIGNOFF command.

It is also possible to use

```
CALL CMD('$SIGNOFF ',9)
```

which does cause the user to be signed off immediately.

Calling the QUIT subroutine has the same effect as using the OFFBIT item of the GUINFO/CUINFO subroutine. The effect of calling the QUIT subroutine may be disabled by calling the CUINFO subroutine to reset the OFFBIT item to zero.

RCALL

Subroutine Description

Purpose: To call R-type subroutines (such as GETFD) from FORTRAN.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL RCALL(a,m,ir(1),...,ir(m),n,rr(1),..., rr(n),&rc4,...)

Parameters:

- a is the address of the R-type subroutine which is to be called. This should be declared EXTERNAL.
- m is the fullword integer number of general registers starting with GR0 to be set up prior to calling the R-type subroutine. m may range between 0 and 13, inclusive.
- ir(1),...,ir(m) are the values to be placed in GR0 through GR(m-1) respectively. These parameters must be fullword-aligned and four bytes in length.
- n is the fullword integer number of general registers starting with GR0 to be stored after calling the R-type subroutine. n may range between 0 and 13, inclusive.
- rr(1),...,rr(n) are the n variables into which the contents of GR0 through GR(n-1) will be stored after calling the R-type subroutine. These parameters must be fullword-aligned and four bytes in length.
- rc4,... is the statement label to transfer to upon receiving a nonzero return code from the subroutine called via RCALL.

Return Codes:

The return code from RCALL is identical to the return code returned by the R-type subroutine. The contents of the general registers have been returned after the R-type subroutine call as specified by the parameters.

Description: The general registers starting with 0 are set up as specified by the parameter list. The second parameter specifies the number of registers to be set up, and the parameters following specify the values to be placed into the registers. The R-type subroutine is called, and when it returns, the general registers starting with 0 are stored as specified by the parameter list. The return code is as returned by the R-type subroutine.

Many R-type subroutines require that addresses be placed in registers before calling them. These addresses can be computed by using the subroutine ADROF. See the ADROF subroutine description in this volume.

If the subroutine also requires an S-type parameter list, the address of the parameter list must be placed in GR1. This may be done by using the ADROF subroutine where the argument to ADROF is a scalar variable for a single-element parameter list or an array for a multiple-element parameter list.

Example: FORTRAN:

```
EXTERNAL GETFD
INTEGER*4 ADROF,FDUB
CALL RCALL(GETFD,2,0,ADROF('name '),1,FDUB,&9)
```

This example calls GETFD with GR0 containing a zero and GR1 containing the address of the character string "name". GETFD returns the FDUB-pointer in GR0, and this is stored in the variable FDUB. A return code of four from GETFD will cause control to be transferred to statement 9 of the FORTRAN program.

FORTRAN:

```
EXTERNAL CHKFIL
INTEGER*4 ADROF,X,PAR
DATA MASK/Z00000001/
PAR = ADROF('2AGA:DATAFILE ')
CALL RCALL(CHKFIL,2,0,ADROF(PAR),1,X,&100)
X = LAND(X,MASK)
IF(X.EQ.1) GO TO 10
```

This example illustrates a call to the subroutine CHKFIL which uses both an S-type calling sequence parameter list and a R-type return of a value. In this case, the first parameter to CHKFIL is the location of the name of a file.

READ

Subroutine Description

Purpose: To read an input record from a specified logical I/O unit.

Location: Resident System

Alt. Entry: MTSREAD, READ#

Calling Sequences:

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

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

Parameters:

- reg is the location of the virtual memory region to which data is to be transmitted.
- len is the location of a halfword (INTEGER*2) integer in which is placed the number of bytes read.
- mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.
- lnum is the location of a fullword integer giving the internal representation of the line number that is to be read or has been read by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.
- unit is the location of either
  - (a) a fullword-integer FDUB-pointer (such as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99), or
  - (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

- 0 Successful return.
- 4 End-of-file.
- >4 See the "I/O Subroutine Return Codes" description in this volume.

Description: All five of the parameters in the calling sequence are required. The subroutine reads a record from the I/O unit specified by _unit_ into the region specified by _reg_ and puts the length of the record (in bytes) into the location specified by _len_. If the _mod_ parameter (or the FDname modifier) specifies the INDEXED bit, the _lnum_ parameter must specify the line number to be read. Otherwise, the subroutine will put the line number of the record read into the location specified by _lnum_.

If the @MAXLEN FDname I/O modifier is specified, the _len_ parameter is three halfwords which give the number of bytes actually read, the maximum number of bytes to be read, and the physical length of the record read. See the description of the @MAXLEN FDname I/O modifier in the section "I/O Modifiers" in this volume.

There are no default FDnames for READ.

Note that the contents of the input area _reg_ may be changed even if the subroutine gives a nonzero return code.

There is a macro READ in the system macro library for generating the calling sequence to this subroutine. See the macro description for READ in MTS Volume 14, _360/370 Assemblers in MTS_.

Examples: The example below, given in assembly language and FORTRAN, calls READ specifying an input region of 20 fullwords. The logical I/O unit specified is 5 and there is no modifier specification made in the subroutine call.

Assembly:

```
                 CALL READ,(REG,LEN,MOD,LNUM,UNIT)
                  .
                  .
          REG    DS   CL80
          LEN    DS   H
          MOD    DC   F'0'
          LNUM   DS   F
          UNIT   DC   F'5'

                 or

                 READ 5,REG,LEN   Subr. call using macro
```

FORTRAN:

```
        INTEGER*2 LEN
        INTEGER REG(20),LNUM
         ...
        CALL  READ(REG,LEN,0,LNUM,5,&30)
         ...
30       ...
```

The example below, given in assembly language and FORTRAN, sets up a call to READ specifying that the input will be read from the file FYLE.

Assembly:

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

FORTRAN:

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

READBFR

Subroutine Description

Purpose: To allow programs to read from an arbitrary file or device without knowing the maximum record length in advance.

Location: *LIBRARY

Calling Sequence:

Assembly: CALL READBFR,MF=(E,pars)

The MF form of the CALL macro is normally used to call this subroutine. The MF form generates the code to call the subroutine without generating the actual parameter list (see the description of the CALL macro in MTS Volume 14 for complete details).

Parameters:

pars is the location of a remote parameter list suitable for calling the READ subroutine. The first parameter in the list, the input area address, must be set to zero on the first call to READBFR.

Return Codes:

0 Successful return.
4 End-of-file return.
>4 See the "I/O Subroutine Return Codes" section in this volume.

Description: If the first parameter of the remote parameter list is zero, the subroutine READBFR will internally call the subroutine GDINFO to determine the length of the longest record that can be read from the file, device, or logical I/O unit and will allocate a buffer that is large enough to accommodate it; the address of this buffer will be stored into the first parameter location in place of the zero. The READ subroutine will then be called internally to read a record using the READBFR parameter list as the parameter list for READ; the NOTIFY modifier will also be set for the read operation.

If the first parameter location is not zero (usually on the second and subsequent calls to READBFR), READBFR will call READ directly using the READBFR parameter list and setting the NOTIFY modifier.

If the file or device attached changes, READBFR will release the current buffer and allocate a new buffer of the appropriate size and will store the address of the new buffer into the first parameter location.

Note: If the maximum input length of the file or device changes without concatenation, this subroutine may not have a buffer large enough for all cases, e.g., if another FDUB is used to write a line into the file that is longer than the current maximum line length.

Example: Assembly:

```
LABEL   CALL READBFR,MF=(E,PARS),EXIT=EOF
        L    1,PARS        Get address of buffer
         .
         .                 Process record
         .
        B    LABEL
EOF     L    1,PARS        Release buffer
        CALL FREESPAC
         .
         .
PARS    READ 'SCARDS',,LEN,MF=L
LEN     DS   H
```

The above example reads records from SCARDS until a nonzero return code is encountered. After each call to READBFR, PARS contains the location of the record read. When a nonzero return code is encountered, the buffer is released by calling FREESPAC. The MF=L form of the READ macro generates the parameter list to the READ subroutine without generating the code to call the READ subroutine. This parameter list is then used as the remote parameter list for the READBFR subroutine.

RENAME

Subroutine Description

Purpose: To change the name of a file.

Location: Resident System

Calling Sequence:

Assembly: CALL RENAME,(oldname,newname)

```
FORTRAN:  CALL RENAME(oldname,newname,&rc4,&rc8,&rc12,
                      &rc16,&rc20,&rc24,&rc28,&rc32,&rc36)
```

Parameters:

<u>oldname</u> is the location of the old name (with a trailing blank) of the file to be renamed.
<u>newname</u> is the location of the new name (with a trailing blank).
<u>rc4,...,rc36</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 The file was renamed successfully.
4 Illegal old name specified.
8 Old name does not exist.
12 Rename access not permitted (old file name).
16 Locking the file for renaming will result in a deadlock.
20 Illegal new name specified.
24 New name already exists.
28 Disk space allotment exceeded.
32 Hardware error or software inconsistency encountered.
36 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent usage of the shared file).

Notes: Temporary as well as permanent old file names may be renamed.

The old file name may belong to another user.

The new file name may not specify a file belonging to another signon ID unless the old file name also belonged to that same signon ID (and rename access was permitted).

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from RENAME with a return code of 36.

If the old file belongs to another signon ID and the new file name specifies a file belonging to the ID currently signed on, the owner of the file is changed to the current ID if there is sufficient disk space allotted.

If the old file is a temporary file and the new file name specifies a permanent file, the file becomes a permanent file if there is sufficient disk space allotted.

If the old file is a permanent file and the new file name specifies a temporary file, the file becomes a temporary file and is destroyed at signoff.

Examples: Assembly:

```
         CALL RENAME,(OLDNAME,NEWNAME)
          .
          .
OLDNAME DC   C'-TEST '
NEWNAME DC   C'TEST.0 '
```

The above example renames the temporary file -TEST to the permanent file TEST.0.

FORTRAN:

```
CALL RENAME('STAT:TEST ','MYTEST ')
```

The above example renames the file TEST under the signon ID STAT to the file MYTEST under the calling signon ID. After the renaming has occurred, the file STAT:TEST will no longer exist under the signon ID STAT and the disk storage in use by that signon ID will have been updated accordingly.

RENUMB

Subroutine Description

Purpose: To renumber all or a subset of the lines in a line file.

Location: Resident System

Calling Sequence:

```
Assembly  CALL RENUMB,(unit,first,last,beg,inc)

FORTRAN:  CALL RENUMB(unit,first,last,beg,inc,&rc4,&rc8,
                      &rc12,&rc16,&rc20,&rc24,&rc28)
```

Parameters:

unit — is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

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

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

beg — is the location of a fullword containing the new internal line number to be associated with the first line to be renumbered.

inc — is the location of a fullword containing the internal increment to be used while renumbering the requested lines in the file.

rc4,...,rc28 — (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 The file was renumbered successfully.
4 The file does not exist or unit is invalid.
8 Hardware error or software inconsistency encountered.
12 Renumber (or read-write) access not allowed.
16 Locking the file for modification will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent

usage of the shared file).
24 Parameters not addressable or inconsistent parameters specified (renumbering will cause duplicate or nonincreasing line numbers, etc.).
28 The file is not a line file.
32 Invalid increment specified.

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

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

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from RENUMB with a return code of 20.

Examples: Assembly:

```
         CALL  GETFST,(UNIT,FSTLN)
         CALL  GETLST,(UNIT,LSTLN)
         CALL  RENUMB,(UNIT,FSTLN,LSTLN,BEGLN,INC)
          .
          .
UNIT  DC    F'4'
FSTLN DS    F           First line number
LSTLN DS    F           Last line number
BEGLN DC    F'1000'     1 in internal form
INC   DC    F'1000'     1 in internal form
```

FORTRAN:

```
INTEGER*4 UT
DATA UT/4/
  ...
CALL RENUMB(UT,-2147483648,2147483647,1000,1000)
```

The above examples illustrate two ways to renumber all of the lines of the line file attached to logical I/O unit 4. The lines are renumbered starting at line 1 by increments of 1.

RETLNR

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

FORTRAN: CALL RETLNR(unit,first,last,cnt,buffer,&rc4,
&rc8,&rc12,&rc16,&rc20,&rc24,&rc28,
&rc32)

Parameters:

<u>unit</u> is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

<u>first</u> is the location of a fullword containing the internal line number of the first line number to be returned.

<u>last</u> is the location of a fullword containing the internal line number of the last line number to be returned.

<u>cnt</u> is the location of a fullword in which the count of the number of lines in the specified range will be returned.

<u>buffer</u> is the location of a buffer. The buffer is supplied by the caller; bytes 8 and on are filled in by the subroutine. This buffer should be of the form:

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

<u>rc4,...,rc32</u> (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 The line numbers were returned.
4 The file does not exist or _unit_ is invalid.
8 Hardware error or software inconsistency encountered.
12 Read or renumber access not allowed.
16 Locking the file for reading will result in a deadlock.
20 An attention interrupt has canceled the automatic wait on the file (waiting caused by concurrent use of the shared file).
24 Parameters not addressable or inconsistent parameters specified (_first_ greater than _last_, etc.).
28 The file is not a line file.
32 Buffers exhausted before line-number range was exhausted.

Notes: If _first_ and _last_ do not correspond to actual line numbers in the file, the next and previous line numbers, respectively, will be used.

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

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from RENUMB with a return code of 20.

Examples: Assembly:

```
                  CALL GETFST,(UNIT,FSTLNR)
                  CALL GETLST,(UNIT,LSTLNR)
                  CALL RETLNR,(UNIT,FSTLNR,LSTLNR,CNT,BUFF)
                   .
                   .
        UNIT   DC  F'4'
        FSTLNR DS  F            First line number
        LSTLNR DS  F            Last line number
        CNT    DS  F            Count of lines in file
        BUFF   DC  F'0'         The only buffer
               DC  F'808'       This many bytes
               DS  200F         Line numbers go here
```

The above example illustrates how to return all of the line numbers of the line file attached to logical I/O unit 4 (assuming there are less than 200 lines in the file).

FORTRAN:

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

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

REWIND

Subroutine Description

Purpose: To rewind a logical I/O unit in FORTRAN.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL REWIND(unit)

Parameters:

<u>unit</u> is the location of a fullword integer corresponding to the logical I/O unit number to be rewound. These are 0 through 99.

Description: If the logical I/O unit number specified by <u>unit</u> is attached to a magnetic tape, it is rewound. If it is attached to a line file, it is reset so that the next sequential reference to it will read or write the line specified by the beginning line number given when the file was attached. If it is attached to a sequential file, or a floppy disk, it is reset so that the next reference to it will read or write from the beginning of the file. In all other cases, an error comment is produced on the logical I/O unit SERCOM, and the subroutine ERROR is called.

If the logical I/O unit specified by <u>unit</u> is part of an explicit or implicit concatenation, only the currently active member is rewound.

The REWIND subroutine generates a call to the REWIND# subroutine.

Example: FORTRAN: CALL REWIND(1)

The file or device attached to logical I/O unit 1 is rewound.

REWIND#

Subroutine Description

Purpose: The rewind a line file, a sequential file, a magnetic tape, or a floppy disk.

Location: Resident System

Calling Sequences:

Assembly: (a)

```
L    0,unit
SR   1,1
CALL REWIND#

or

REWIND unit
```

(b)

```
LM   0,1,unit
CALL REWIND#

or

REWIND 'unit'
```

Parameters:

(a) GR0 contains an FDUB-pointer (such as GETFD returns) or a fullword logical I/O unit number (0-19), and GR1 contains zero.

(b) GR0 and GR1 contain an 8-character logical I/O unit name left-justified with trailing blanks. The logical I/O unit names are: SCARDS, SPRINT, SPUNCH, SERCOM, GUSER, and 0 through 99.

Return Codes:

0 Successful return.

4 Unable to rewind the device specified by GR0 and GR1.

Notes: The complete description for using the REWIND macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

FORTRAN programs should use the REWIND subroutine which is described in this volume.

Description: If GR0 and GR1 specify a magnetic tape, it is rewound. If they specify a line file, it is reset so that if the next reference to this FDUB or logical I/O unit is sequential, it will read or write the line specified by the beginning line number given when the file was attached. If they specify a sequential file or a floppy disk, the FDUB is reset so that the next read or write will be at the beginning of the file. For all other cases, a return code of 4 is given.

If the logical I/O unit or FDUB-pointer specified by GR0 and GR1 is part of an explicit or implicit concatenation, only the currently active member is rewound.

Example: Assembly:

```
              LM   0,1,LNAME
              CALL REWIND#
               .
               .
        LNAME DC   CL8'SPRINT  '

             REWIND 'SPRINT'
```

The above two examples reset the magnetic tape or file attached to the logical I/O unit SPRINT. The first uses the CALL macro and the second uses the REWIND macro.

RSSAS

Subroutine Description

Purpose: To reset *SOURCE* to *MSOURCE* and *SINK* to *MSINK*.

Location: Resident System

Calling Sequences:

Assembly: CALL RSSAS,(sws),VL

FORTRAN: CALL RSSAS(sws)

Parameters:

sws is the location of a fullword integer specifying what is to be reset. The legal values are:

0 both *SOURCE* and *SINK* are reset.
1 only *SOURCE* is reset.
2 only *SINK* is reset.

Return Codes:

0 Successful return.
4 Nothing is reset (SIGFILEATTN or the project sigfile attention bit is OFF and a sigfile is being processed).
8 Invalid parameter.

Description: The RSSAS subroutine may be used by interactive programs to reset *SOURCE* and/or *SINK* when an attention interrupt is received and *SOURCE* is not the same as *MSOURCE* or *SINK* is not the same as *MSINK*. This action is similar to the action taken by MTS when an attention interrupt is received while reading commands from a file as the result of the $SOURCE command.

Example: Assembly:

```
             LA   1,FNAME
             CALL GETFD          Get FDUB for *MSOURCE*
             ST   0,FDUB
              .
              .
             LM   0,1,=A(EXIT,REGN)
             CALL ATTNTRP        Enable attn intercept
              .
              .
             USING EXIT,10
      EXIT   LR   10,15
```

```
               LA   0,EXIT
               CALL CFDUB,(SCRDS,FDUB)  Compare FDUBs
               LTR  15,15
               BE   EXITA          Same
               CALL RSSAS,(SWS),VL Reset *SOURCE*
         EXITA MVI  0(1),X'FF'
               CALL ATTNTRP        Reenable intercept
                .
                .
         FNAME DC   C'*MSOURCE* '
         SCRDS DC   C'SCARDS  '
         FDUB  DS   F
         SWS   DC   F'1'
         REGN  DS   18F
```

FORTRAN:

```
               EXTERNAL GETFD
               INTEGER ADROF,FDUB
               LOGICAL ATTN
                ...
               CALL RCALL(GETFD,2,0,ADROF('*MSOURCE* '),
              C           2,DUMMY,FDUB)
                ...
               CALL ATNTRP(ATTN)
                ...
          10   IF (ATTN) GO TO 20
                ...
                ... Program loop
                ...
               GO TO 10
          20   CALL ATNTRP(ATTN)
               CALL CFDUB('SCARDS  ',FDUB,&30)
               GO TO 10
          30   CALL RSSAS(1)
               GO TO 10
```

The above examples, coded both in assembler and FORTRAN, reset SCARDS to *MSOURCE* if an attention interrupt is taken during the program loop. GETFD is called to get an FDUB-pointer for *MSOURCE* which is subsequently tested by CFDUB against the current assignment of SCARDS; if they are different, RSSAS is called to reset *SOURCE* (the SCARDS assignment) to *MSOURCE*.

RSTIME

Subroutine Description

Purpose: To cancel timer interrupts set up by the SETIME subroutine and return the time remaining until the interrupt would have occurred.

Location: Resident System

Calling Sequences:

Assembly: CALL RSTIME,(id,value,aregion)

FORTRAN: CALL RSTIME(id,value,aregion,&rc4)

Parameters:

_id_ is the location of the fullword identifier which specifies the timer interrupt to be canceled. This is the same identifier which was given to SETIME when the interrupt was set up. If this identifier is zero, all timer interrupts with the specified exit region will be canceled.

value is the location of a 4-, 8-, or 16-byte fullword-aligned region in which RSTIME returns the time remaining until the interrupt would have occurred. The interpretation of this value depends upon the _code_ parameter given to SETIME when the interrupt was set up. For codes 0 and 2, the value is an 8-byte binary integer specifying microseconds of task CPU time; for codes 1, 3, and 5, the value is an 8-byte binary integer specifying microseconds of real time; for code 4, the value is a 4-byte binary integer specifying timer units of task CPU time.

aregion is the location of the address of the 76-byte exit region which was given to SETIME when the interrupt was set up. The combination of the identifier and the exit region address will always specify a unique timer interrupt. If both _aregion_ and _id_ are zero, all timer interrupts will be canceled.

_rc4_ (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 No such timer interrupt was found. This means either
(1) no such interrupt was ever set up, or
(2) the interrupt has occurred, and the exit was taken before the execution of the BALR instruction which branches to RSTIME.

Description: A call on the RSTIME subroutine cancels a timer interrupt set up by the SETIME subroutine, and returns the time remaining until the interrupt would have occurred in the value parameter. The timer interrupt to be canceled is specified by the combination of the id and aregion parameters. The interrupt will be canceled even if it has already occurred and is pending.

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

FORTRAN users should consult the TICALL subroutine description in this volume for details on using timer interrupts with FORTRAN.

Example: Assembly:

```
                  CALL RSTIME,(ONE,TIMLEFT,AREG)
                   .
                   .
          ONE     DC  F'1'
          TIMLEFT DS  FL8
          AREG    DC  A(EXIT)
          REG     DS  19F
```

FORTRAN:

```
      INTEGER TICALL
      EXTERNAL EXIT
      INTEGER TIME(2),/0,10000/,LEFT(2)
      IREG = TICALL(0,EXIT,TIME,&4,&8)
      CALL RSTIME(EXIT,LEFT,IREG,&4)
```

The above example, coded in assembly language and FORTRAN, cancels the interrupt with the identifier 1 and the exit region REG. The time remaining is returned in TIMLEFT.

SCANSTOR

Subroutine Description

Purpose: To "scan" storage blocks. For each block of allocated storage in the range specified, SCANSTOR will call a subroutine specified, giving it the location and length of that block.

Location: Resident System

| Alt. Entries: SSTOR, SCANSTOS, SCNSTS

Calling Sequences:

```
Assembly: L    0,switch
          L    1,sinbr
          L    2,subr
          CALL SCANSTOR

          CALL SCANSTOS,(switch,sinbr,subr),VL

FORTRAN:  CALL SCNSTS(switch,sinbr,subr,&rc4)
```

Parameters:

<u>switch</u> (GR0) controls the scanning.
- if 0, only storage with the specified storage index number (<u>sinbr</u>).
- if +1, storage with index numbers less than or equal to the one given (this and lower link levels).
- if -1, storage with index numbers greater than or equal to the one given (this and higher link levels).

<u>sinbr</u> (GR1) storage index number or zero. If zero, the storage index number of the current link level will be used.

<u>subr</u> (GR2) location of the subroutine to call for each block. When this call is made, GR0 will have the length, GR1 will have the location of the block, and GR2 will have the storage index number of the block. This call conforms to the OS R-type calling convention.

Return Codes:

0 Successful return.
4 Invalid parameter (no VL bit specified).

Description: A call on the SCANSTOS or SCNSTS subroutines takes the S-type parameters and loads them into an R-type call on the SCANSTOR subroutine.

For a further description of storage index numbers, see the "Virtual Memory Management" section in MTS Volume 5, System Services.

Examples: Assembly:

```
              LA   0,1
              SR   1,1
              LA   2,MYDUMP
              L    15,=V(SCANSTOR)
              BALR 14,15

     or

              LM   0,2,SPAR
              CALL SCANSTOR
               .
               .
     SPAR     DC   A(1,0,MYDUMP)
```

FORTRAN:

```
      COMMON /DUMP/ MYDUMP
      CALL SCNSTS(1,0,MYDUMP,&4)
```

The above example, coded in assembly language and FORTRAN, calls SCANSTOR specifying that storage is to be scanned which has storage index numbers equal to or less than the current link level storage index number.

SCARDS

Subroutine Description

Purpose: To read an input record from the logical I/O unit SCARDS.

Location: Resident System

Alt. Entry: SCARDS#

Calling Sequences:

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

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

Parameters:

<u>reg</u> is the location of the virtual memory region to which data is to be transmitted.

<u>len</u> is the location of a halfword (INTEGER*2) integer in which is placed the number of <u>bytes</u> read.

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

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

<u>rc4,...</u> is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

0 Successful return.
4 End-of-file.
>4 See the "I/O Subroutine Return Codes" description in this volume.

Description: All four of the above parameters in the calling sequence are required. The subroutine reads a record into the region specified by <u>reg</u> and puts the length of record (in bytes) into the location specified by <u>len</u>. If the <u>mod</u> parameter (or the FDname modifier) specifies the INDEXED bit, the _lnum_ parameter must specify the line number to be read. Otherwise, the subroutine will put the line number of the record read into the location specified by _lnum_.

If the @MAXLEN FDname I/O modifier is specified, the _len_ parameter is three halfwords which give the number of bytes actually read, the maximum number of bytes to be read, and the physical length of the record read. See the description of the @MAXLEN FDname I/O modifier in the section "I/O Modifiers" in this volume.

The default FDname for SCARDS is *SOURCE*.

Note that the contents of the input area _reg_ may be changed even if the subroutine gives a nonzero return code.

There is a macro SCARDS in the system macro library for generating the calling sequence to this subroutine. See the macro description for SCARDS in MTS Volume 14, _360/370 Assemblers in MTS_.

Examples: The example below, given in assembly language and FORTRAN, calls SCARDS specifying an input region of 20 fullwords. There is no modifier specification made on the subroutine call.

Assembly:

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

                 or

                 SCARDS REG,LEN   Subr. call using macro
```

FORTRAN:

```
                INTEGER*2 LEN
                INTEGER REG(20),LNUM
                 ...
                CALL SCARDS(REG,LEN,0,LNUM,&30)
                 ...
          30     ...
```

Screen-Support Routines

Subroutine Description

Purpose: To provide user-program control for a video-terminal screen.

Location: Resident System

Description: The screen-support routines have the following entry points:

SSATTR
SSBGNS
SSCREF
SSCTNS
SSCTRL
SSCURS
SSDEFF
SSDELF
SSDELS
SSENDS
SSINFO
SSINIT
SSLOCN
SSREAD
SSTERM
SSTEXT
SSWRIT

The complete description of the Screen-Support Routines is given in MTS Volume 4, Terminals and Networks in MTS.

SDUMP

Subroutine Description

Purpose: To produce a dump of any or all of the following:

(1) general registers,
(2) floating-point registers,
(3) a specified region of virtual storage.

Location: Resident System

Calling Sequences:

Assembly:
```
EXTRN outsub
CALL SDUMP,(switch,outsub,wkarea,first,last)
```

Parameters:

switch is the location of a fullword containing switches that govern the content and format of the dump produced. The switches are assigned as follows:

bit 31: on if hexadecimal conversion of the storage region is desired.
30: on if mnemonic conversion of the storage region is desired.
29: on if EBCDIC conversion of the storage region is desired.
28: on if double spacing is desired; off if single spacing is desired.
27: on if long output records (130 characters) are to be formed; off if short output records (70 characters) are to be formed.
26: on if general registers are to be dumped.
25: on if floating-point registers are to be dumped.
24: on if a storage region is to be dumped.
23: on if no column headers are to be produced for the dump of the storage region.

outsub is the location of a subroutine (e.g., SPRINT) that causes the printing, punching, etc., of the output line images formed by SDUMP. This subroutine should be declared as

EXTRN.

<u>wkarea</u> is the location of a doubleword-aligned area of 400 bytes that may be used by SDUMP as a work area.

<u>first</u> is the location of the first byte of a storage region to be dumped. There are no boundary requirements for this address.

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

Note: The default case for <u>switch</u> (all switches off) produces a dump as though bits 24, 25, 26, and 31 were on. Furthermore, if bit 30 (mnemonics) is on, bit 31 (hexadecimal) is implied. Note that bits 24, 25, and 26 specify what is to be dumped, bits 27 and 28 specify the page format, and bits 29, 30, and 31 specify the interpretation(s) to be placed on the region of storage specified. Bits 29 through 31 have significance only if bit 24 is on.

Return Codes:

0 Successful return.
4 Illegal parameters specified.

Description: <u>Output Formats</u>

Registers:

General and floating-point registers, if requested, are always given in labeled hexadecimal format. The length of the output record is governed by the setting of bit 27 of the switch.

Virtual Storage:

Although <u>any</u> combination of switches is acceptable, the appearance of the dump output for a region of virtual storage is determined as follows:

(1) If, and only if, the mnemonic switch is <u>on</u>, the unit of storage presented in each print item is a halfword-aligned halfword.

(2) If, and only if, the mnemonic switch is <u>off</u> and the hexadecimal switch is <u>on</u> (through intent or default), the unit of storage presented in each print item is a fullword-aligned fullword.

(3) If, and only if, the mnemonic and hexadecimal switches are _off_ but the EBCDIC switch is _on_, the unit of storage presented in each print item is a doubleword-aligned doubleword.

In all cases, the output includes:

(1) the entire storage unit (halfword, fullword, or doubleword) in which the first specified location (parameter _first_) is found,

(2) the entire storage unit in which the last location (parameter _last_) is found, and

(3) all intervening storage.

Thus, the first and last printed items of a storage dump may include up to a maximum of seven bytes more than actually requested in the parameter list.

If mnemonics are requested and SDUMP discovers a byte that cannot be interpreted as an operation code, then instead of a legal mnemonic, the characters "****" appear directly below the hexadecimal presentation of the halfword in storage that should have contained an operation code. When this occurs, the mnemonic scanner jumps ahead as though the illegal operation code specified an RR-type instruction (two bytes) and tries to interpret the byte at the new location as an operation code, etc. Any mnemonic print line that contains the "****" for at least one of its entries is also marked with a single "X" directly below the line address that prefixes the hexadecimal presentation of that same region of storage. (The mnemonic conversion routine includes the full IBM 370 Model 168 instruction set.) To facilitate the location of particular items in the output, line addresses _always_ have a _zero_ in the least significant hexadecimal position. Column headers are provided which give the value of the least significant hexadecimal digit of the address of the first byte in each print item.

A line of dots is printed to indicate that a region of storage contains identical items. The storage unit used for comparisons is halfword, fullword, or doubleword depending upon the type(s) of conversion specified. In all cases, the storage unit corresponding to the last item printed before the line of dots, the storage unit for the first item after the line, and all intervening storage units have identical contents. The last line is always printed (even if all of its entries exactly match the previously printed line).

Example: Assembly:

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

FORTRAN:

```
      REAL*8 WK(50)
      LOGICAL*1 FIRST(4)
      EXTERNAL SPRINT
       ...
      CALL SDUMP(0,SPRINT,WK,FIRST(1),FIRST(3),&4)
```

The above example, coded in assembly language and FORTRAN, will cause SDUMP to print the contents of the location FIRST.

SERCOM

Subroutine Description

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

Location: Resident System

Alt. Entry: SERCOM#

Calling Sequences:

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

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

Parameters:

reg is the location of the virtual memory region from which data is to be transmitted.
len is the location of a halfword (INTEGER*2) integer giving the number of bytes to be transmitted.
mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.
lnum (optional) is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.
rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

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

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

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

If len is zero when writing to a line file , the line is deleted from the file.

The default FDname for SERCOM is *MSINK*.

There is a macro SERCOM in the system macro library for generating the calling sequence to this subroutine. See the macro description for SERCOM in MTS Volume 14, 360/370 Assemblers in MTS.

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

Assembly:

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

                 or

                SERCOM REG       Subr. call using macro
```

FORTRAN:

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

SETFSAVE

Subroutine Description

Purpose: To enable or disable the saving of files by the system file-save utility. (Unless otherwise directed, all user files are saved so that there will be backup copies of files in case of inadvertent destruction or damage due to hardware failure.)

Location: Resident System

Alt. Entry: SETFS

Calling Sequence:

Assembly: CALL SETFSAVE,(what,onoff,info,errcode,
errmsg),VL

FORTRAN: CALL SETFS(what,onoff,info,errcode,errmsg,&rc4)

Parameters:

<u>what</u> is the location of either
- (1) a file name with a trailing blank (if <u>info</u>=0),
- (2) a fullword-integer FDUB pointer (such as returned by GETFD) (if <u>info</u>=1),
- (3) a fullword-integer logical I/O unit number (0 through 99) (if <u>info</u>=1), or
- (4) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if <u>info</u>=1).

<u>onoff</u> is the location of a fullword-integer 0 or 1. If 1, the file is <u>not</u> to be saved by the system file-save program.

<u>info</u> is the location of a fullword-integer 0 or 1 which identifies the type of the <u>what</u> parameter.

<u>errcode</u> (optional) is the location of a fullword in which the SETFSAVE subroutine will place the error number if an error return (return code 4) is made. If <u>errcode</u> is omitted, the <u>errmsg</u> parameter must also be omitted. Assembly language users who wish to omit this parameter should either follow the variable parameter list convention (high-order bit of the previous parameter adcon is set to 1) or supply an adcon which is zero (rather than pointing to a zero).

Error numbers less than 100 indicate an error in the mechanics of the subroutine call or in the values of the parameters:

Number	Message
1	ILLEGAL PARAMETER LIST POINTER
2	ILLEGAL "WHAT" PARAMETER ADDRESS
3	ILLEGAL "ONOFF" PARAMETER ADDRESS
4	"ONOFF" PARAMETER VALUE NOT 0 OR 1
5	ILLEGAL "INFO" PARAMETER ADDRESS
11	"INFO" PARAMETER VALUE NOT 0 OR 1

Error numbers between 100 and 105 indicate errors that occur in accessing the file.

101	ILLEGAL FILE NAME
102	FILE NOT FOUND - FILE "XXX"
103	ACCESS NOT ALLOWED TO FILE "XXXX" (Permit access is required to set the save status.)
104	DEADLOCK SITUATION, TRY LATER - FILE "XXXX"
105	INTERRUPTED OUT OF WAIT FOR LOCKED FILE "XXXX"

Error numbers 201 and above indicate a system error.

<u>errmsg</u> (optional) is the location of a 20-fullword (80-character) region in which the SETFSAVE subroutine will place the corresponding error message if an error occurs. Assembly language users should see instructions above on omitting optional parameters for the <u>errcode</u> parameter.

<u>rc4</u> is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

0 The save status has been set as requested.

4 Error return. The save status has not been changed, but the <u>errcode</u> and <u>errmsg</u> values have been set, if specified.

Examples: Assembly:

```
                CALL SETFSAVE,(WHAT,ONOFF,INFO,ERRCOD,
                              ERRMSG),VL
                 .
                 .
         WHAT   DC   C'TOPSECRET '
         ONOFF  DC   F'1'
         INFO   DC   F'0'
         ERRCOD DS   F
         ERRMSG DS   CL80
```

FORTRAN:

```
CALL SETFS('TOPSECRET ',1,0,ERRCOD,ERRMSG,&100)
```

In both examples above, the file TOPSECRET is not to be saved by the system file-save program.

SETIME

Subroutine Description

Purpose: To set up a timer interrupt to occur after a specified time interval (either real time or CPU time for the current task).

Location: Resident System

Calling Sequences:

Assembly: CALL SETIME,(code,id,value,aregion)

Parameters:

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

0 _value_ is an 8-byte binary integer which specifies a time interval in microseconds of task CPU time, relative to the time of the call.
1 _value_ is an 8-byte binary integer which specifies a time interval in microseconds of real time, relative to the time of the call.
2 _value_ is an 8-byte binary integer which specifies a time interval in microseconds of task CPU time, relative to the time at signon.
3 _value_ is an 8-byte binary integer which specifies a time interval in microseconds of real time, relative to the time at signon.
4 _value_ is a 4-byte binary integer which specifies a time interval in timer units (13 1/48 microseconds per unit) of task CPU time, relative to time of the call.
5 _value_ is a 16-byte EBCDIC string giving the time and date at which the interrupt is to occur, in the form HH:MM.SSMM-DD-YY.
6 _value_ is a 8-byte binary integer which specifies a time interval in microseconds of real time since March 1, 1900 (local time).
7 _value_ is a 8-byte binary integer which specifies a time interval in microseconds of real time since January 1, 1900 (GMT).

id is the location of a fullword identifier which will be passed to the exit routine when the interrupt occurs and the exit is taken. *id* should be nonzero.

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

aregion is the location of the address of the 76-byte exit region to be used when the interrupt occurs and the exit is taken. This is the same exit region address used in the call on TIMNTRP which enables the exit for this interrupt.

Return Codes:

0 Successful return.
4 Invalid *code* or *aregion* parameter.
8 Too many interrupts set up.

Description: Each call on the SETIME subroutine sets up a new timer interrupt to occur at the time specified by the *code* and *value* parameters. When the interrupt occurs, an exit will be taken using the exit region specified by the *aregion* parameter, if that exit is enabled. Exits are enabled or disabled by the TIMNTRP subroutine, and all exits are disabled until enabled by TIMNTRP subroutine. The combination of the identifier specified by *id* and the exit region is forced to be unique, since the SETIME subroutine will cancel any previously set up interrupt with the same identifier and exit region address.

A maximum of 100 interrupts is allowed. This restriction is for error-checking purposes only.

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

FORTRAN users should consult the TICALL subroutine description in this volume for details on using timer interrupts with FORTRAN.

Example: Assembly:

```
CALL SETIME,(ZERO,ONE,TENSEC,AREG)
LM   0,1,=A(EXIT,REG)
CALL TIMNTRP
  .
CALL SETIME,(ONE,TWO,FIVMIN,AREG)
LM   0,1,=A(EXIT,REG)
CALL TIMNTRP
  .
CALL SETIME,(FIVE,THREE,TWO30,AREG)
LM   0,1,=A(EXIT,REG)
```

```
                CALL TIMNTRP
                 .
                 .
        ZERO    DC  F'0'
        ONE     DC  F'1'
        TWO     DC  F'2'
        THREE   DC  F'3'
        FIVE    DC  F'5'
        TENSEC  DC  FL8'10000000'
        FIVMIN  DC  FL8'300000000'
        TWO30   DC  C'02:30.00',C'04-12-72'
        AREG    DC  A(REG)
        REG     DS  19F
```

This example sets up three timer interrupts. The first interrupt is a task CPU time interrupt 10 seconds after the call; the second is a real-time interrupt 5 minutes after the call; the third is a real-time interrupt at 2:30 a.m. on April 12, 1972. All the interrupts are enabled by calls to TIMNTRP and will cause the subroutine EXIT to be invoked after the designated intervals have passed.

SETIOERR

Subroutine Description

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

This subroutine is obsolete. The @ERRRTN I/O modifier should be used instead.

Location: Resident System

Calling Sequence:

Assembly: CALL SETIOERR,(loc)

Parameters:

loc is either:
(a) the location of a subroutine to transfer to when an I/O error occurs, or
(b) zero, in which case the error exit is reset.

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

When the error routine is called, registers 0 and 1 both contain what was in GR13 upon entry to the I/O routine, i.e., the location of the save area in which the I/O routine saved registers at the time of the call. This can be used to obtain the parameter list for the call on the I/O subroutine.

If the error routine returns (BR 14), a return is made to the user's program from the I/O routine with the return code indicating the type of error that occurred. The return code depends upon the type of device in use when the error occurred. See the section "I/O Subroutine Return Codes" in this volume. This is the same behavior as if the @ERRRTN I/O modifier had been set for the I/O call. If the @ERRRTN modifier is used on an I/O call, the SETIOERR exit is never taken.

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

Example: Assembly:

```
      CALL SETIOERR,(SUBR)
      SCARDS DATAREG,LEN,EXIT=(EOF,IOERR)
       .
       .
SUBR  ENTER 12
      SPRINT 'TAPE READ ERROR'
      EXIT 0
```

The call to SETIOERR enables the error exit. If on a succeeding I/O operation, a transmission occurs, SETIOERR will call SUBR, thus allowing the user to take his own error exit.

SETKEY

Subroutine Description

Purpose: To set the program key associated with a file.

Location: Resident System

Calling Sequences:

Assembly: CALL SETKEY,(what,pkey,info,ercode,errmsg),VL

FORTRAN: CALL SETKEY(what,pkey,info,ercode,errmsg,&rc4)

Parameters:

<u>what</u> is the location of either:
(a) a file name with trailing blank (if <u>info</u>=0),
(b) a fullword-integer FDUB-pointer (such as returned by GETFD) (if <u>info</u>=1),
(c) a fullword-integer logical I/O unit number (0 through 99) (if <u>info</u>=1), or
(d) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if info=1).

<u>pkey</u> is the location of the program key to be associated with the file. One trailing blank is required.

<u>info</u> is the location of a fullword integer which specifies the kind of <u>what</u> parameter supplied.

<u>ercode</u> (optional) is the location of a fullword in which the SETKEY subroutine will place an error number if an error return (return code 4) is made. If this parameter is omitted, then the <u>errmsg</u> parameter must also be omitted.

Assembly language users who wish to omit this parameter should either follow the variable parameter list convention (high-order bit of the previous parameter's adcon in the parameter list should be 1) or else supply an adcon which is zero (rather than pointing to a zero).

Error numbers less than 100 indicate something was wrong with either the mechanics of the subroutine call or the values of the parameters:

Number	Message
1	Illegal parameter list pointer
2	Illegal "what" parameter address
3	Illegal "pkey" parameter address
4	Illegal program key
5	Illegal "info" parameter address
6	"Info" parameter value not 0 to 1

Error numbers between 100 and 105 describe errors that occur in accessing the file.

101	Illegal file name
102	File not found - file "xxxx"
103	Access not allowed to file "xxxx" (Permit access required to set the program key).
104	Deadlock situation, try later - file "xxxx"
105	Interrupted out of wait for locked file "xxxx"

Error numbers 201 and above indicate a file system error.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from SETKEY with an error code of 105.

<u>errmsg</u> (optional) is the location of a 20-fullword (80-character) region in which the SETKEY subroutine will place the corresponding error message if an error return (return code 4) is made. Assembly language users should see instructions above on omitting optional parameters for the <u>ercode</u> parameter.

<u>rc4</u> is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

0 The program key has been set as requested.

4 Error. The program key has not been set. See the <u>ercode</u> and <u>errmsg</u> values returned for the specific error.

Examples: Assembly:

```
              CALL SETKEY,(WHAT,PKEY,INFO,ERCODE,ERRMSG)
                .
                .
       WHAT   DC   C'PROGRAM '
       PKEY   DC   C'DBMS '
       INFO   DC   F'0'
       ERCODE DS   F
       ERRMSG DS   CL80
```

FORTRAN:

```
              CALL SETKEY('PROGRAM ','DBMS ',0)
```

The above examples set the program key for file PROGRAM to DBMS.

SETLCL

Subroutine Description

Purpose: To set a local time limit for the executing program.

Location: Resident System

Calling Sequences:

Assembly: CALL SETLCL,(value)

FORTRAN: CALL SETLCL(value)

x=SETLCL(0)

Parameter:

value is the location of a fullword-integer (INTEGER*4) value (in timer units) giving the local time limit to be established. If the value is zero, the current local time limit is canceled. One timer unit is 13 1/48 microseconds or 1/(256*300) seconds.

Value Returned:

GR0 contains the value of the local time limit (in timer units). If the time limit was canceled (value=0), GR0 contains the amount of time remaining before the time limit would have expired. For FORTRAN programs, this value is returned as a function value in x.

Return Codes:

0 Successful return.
4 LSS is in effect and call to SETLCL attempted to set too large a local time limit.

Description: The SETLCL subroutine allows a program to establish, cancel, or change the local time limit. The local time limit set takes effect immediately and applies to the remaining execution time of the program.

Example: Assembly:

```
                CALL SETLCL,(LIMIT)
                 .
                 .
                CALL SETLCL,(ZERO)
                 .
                 .
          LIMIT DC   AL4(10*256*300)
          ZERO  DC   F'0'
```

The above example initially sets up a local time limit of 10 seconds and then subsequently cancels the time limit on the second call to SETLCL.

FORTRAN:

```
                CALL SETLCL(10*256*300,&4)
```

The above example sets a local time limit of 10 seconds.

SETLIO

Subroutine Description

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

Location: Resident System

Calling Sequences:

Assembly: CALL SETLIO,(unit,FDname)

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

Parameters:

unit is the location of the left-justified, 8-character logical I/O unit name (e.g., SCARDS), or a fullword logical I/O unit number (0-99).

FDname is the location of the file or device name to be assigned. This name must be terminated with a trailing blank.

rc4 is the statement label to transfer to if the return code of 4 occurs.

Return Codes:

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

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

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

Examples: Assembly:

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

FORTRAN:

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

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

Assembly:

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

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

SETLNR

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

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

Parameters:

unit is the location of either:
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

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

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

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

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

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

Return Codes:

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

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

Notes: If <u>first</u> and <u>last</u> do not correspond to actual line numbers in the file, the next and previous line numbers, respectively, will be used.

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

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from SETLNR with a return code of 20.

Examples: Assembly:

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

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

FORTRAN:

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

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

SETPFX

Subroutine Description

Purpose: To set the input/output prefix character for the program currently executing. This character is issued during program execution as the first character of every input or output line on a terminal. This subroutine may only be used to set and return single-character prefixes. Longer prefixes may be set and returned using the PFXSTR item of the GUINFO and CUINFO subroutines.

Location: Resident System

Calling Sequences:

Assembly: CALL SETPFX,(char)

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

Parameter:

char is the location of the prefix character.

Return Codes:

0 Successful return.
4 Successful return, but only the first character of a multiple-character prefix is returned in GR0.

Values Returned:

GR0 contains the previous prefix character, right-justified with leading hexadecimal zeros. For FORTRAN users, the value returned by the integer function call to SETPFX will be the previous prefix character, right-justified. If the previous prefix contains more than one character, only the first character is returned. Because of this restriction, the use of the GUINFO and CUINFO subroutines to save and restore prefixes is recommended.

Examples: Assembly:

```
               CALL SETPFX,(PCHAR)
               STC  0,OCHAR
                .
                .
         PCHAR DC   C'?'
         OCHAR DS   C
```

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

FORTRAN:

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

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

SIOC

Subroutine Description

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

Location: Resident System

Alt. Entry: SIOC#

Calling Sequences:

Assembly: CALL SIOC,(buffer,cvarea)

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

Parameters:

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

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

<u>rc4,rc8</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.

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

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

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

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

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

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

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

RF: Fullword-integer replication factor.

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

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

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

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

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

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

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

Relative Field Position - RFP

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

Replication Factor Processing

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

Blanks in Numeric Input Fields

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

Floating-Point Mapping

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

Direct Conversion

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

Free-Format

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

Conversion Code Byte

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

Datum-Length Byte

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

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

Input Picture Byte

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

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

Error Processing

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

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

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

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

Illegal parameter values, which cause a return code of 8 with _no_ changes in any SIOC parameters, arise when one or more of the explicit restrictions given in the parameter descriptions above are violated. If a return code of 8 is given for exceeding the range appropriate for the internal representation, the RFP will be correctly updated and RF and V will be indeterminate.

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

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

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

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

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

SIOCP

Subroutine Description

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

Location: Resident System

Calling Sequence:

Assembly: CALL SIOCP,(format,cvarea)

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

Parameters:

- format — is the location of the first character of the extended format descriptor to be translated. This character string must be terminated by a blank.
- cvarea — is the location of a doubleword-aligned block of storage that will be subsequently used in calling SIOC.
- rc4 — (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

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

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

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

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

The SIOC modifier names and corresponding functions are:

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

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

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

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

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

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

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

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

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

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

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

```
      GO TO 10
      END
```

SIOERR

Subroutine Description

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

This subroutine is obsolete. The @ERRRTN I/O modifier, the FORTRAN ERR exit feature, or the error recovery features of the FTNCMD subroutine should be used instead.

Location: *LIBRARY

Calling Sequence:

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

Parameters:

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

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

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

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

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

Example: FORTRAN:

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

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

SKIP

Subroutine Description

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

Location: *LIBRARY

Calling Sequences:

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

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

Parameters:

- nfiles is the location of the number of files to skip (must be zero for files).
- nrcds is the location of the number of records to skip.
- unit is the location of either
  - (a) a fullword-integer FDUB-pointer (as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99), or
  - (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- rc4,...,rc12 are statement numbers to transfer to if a nonzero return code is encountered.

Return Codes:

- 0 Successful return.
- 4 An end-of-file (filemark) was reached during a forward space or backspace record operation. The unit is left positioned immediately after (on forward space) or before (on backspace) the filemark.
- 8 The load point (beginning of tape) was detected on a backspace operation (tape is left at load point) or the logical end of a labeled tape was detected on a forward space operation (tape is left at the end). This return code cannot occur for files.
- 12 The unit parameter is illegally specified, the unit is not a magnetic tape or file, an I/O error condition was detected, or nfiles is not zero and the unit is a file.

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

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

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

Examples: Assembly:

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

FORTRAN:

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

The above two examples will cause the tape assigned to logical I/O unit 3 to be positioned to the beginning of the current file by backspacing past one filemark, then forward spacing over the filemark (by forward spacing one record). If the current file was the first file on the tape, the tape would backspace to loadpoint and a return code of 8 would be issued by the tape routines, causing SKIP to return with the tape positioned at the beginning of the tape. In FORTRAN, this would cause statement 150 in the calling program to be executed. If the current file was not the first file on the tape, SKIP would perform a forward space record after the backspace file. Note that this forward space record will result in a return code of 4 from SKIP because the forward space record will space over a filemark. This would cause statement 100 in the FORTRAN program to be executed.

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

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

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

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

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

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

SORT

Subroutine Description

Purpose: To sort or merge records.

Location: *LIBRARY

Alt. Entry: SORT1

Calling Sequences:

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

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

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

Parameters:

- cstmt — is the location of the control statement.
- unit — (optional) is the location of a FDUB-pointer (as returned by GETFD), or the location of a fullword-integer logical I/O unit number (0-99).
- vds — (optional) is the location of the virtual data set to be processed.
- num — (optional) is the location of a positive, nonzero, fullword integer that specifies a numeric value in the control statement.
- err — (optional) is the statement label to transfer to if an error (nonzero return code) is detected by the subroutine.
- n — is the number of arguments (FIXED BINARY(31)) to be passed to the subroutine.

Return Codes:

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

Description: See the section "The SORT Utility Program" in MTS Volume 5, *System Services*.

Summary of the Control Statement

Prototype:

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

Collating fields:

TYPE	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
alignment	AL	no	1 - 4095
binary	BI	no	1 - 256
bit	BT	no	1 - 255 (mask)
call	CA	-	1 - 4095
character	CH	no	1 - 256
defined sequence	DS(i)	no	1 - 256
fixed-point	FI	yes	1 - 260
floating-point	FL	yes	2 - 16
length	LE	-	-
packed decimal	PD	yes	1 - 16
sequence	SE	-	-
signed decimal	SD	yes	1 - 17
zoned decimal	ZD	yes	1 - 16

Record structures:

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

Additional parameters:

```
CHK    (exit check facility)
DEC    (delete comments)
DEL=x[,x]...  (delete output records)
LIO    (list data set characteristics)
{REC|MNR}=x   (number of records)
RES=x  (restart)
SIG    (sign off on error)
TPS[={x|name,name[,name]...}]  (tape-merge sort facility)
```

SORT2, SORT3, SORT4

Subroutine Description

Purpose: To sort arrays.

Location: *LIBRARY

Calling Sequences:

```
Assembly: CALL SORT2,(cstmt,loc1,loc2,len[,num]...)
          CALL SORT3,(cstmt,loc1,loc2,len,loc3,len3
                      [,num]...)
          CALL SORT4,(cstmt,loc4,loc2[,num]...)

FORTRAN:  CALL SORT2(cstmt,loc1,loc2,len[,num]...[,&err])
          CALL SORT3(cstmt,loc1,loc2,len,loc3,len3
                     [,num]...[,&err])
          CALL SORT4(cstmt,loc4,loc2[,num]...[,&err])

PL/I(F):  CALL PLCALL(SORT2,n,cstmt,ADDR(loc1),ADDR(loc2),
                      ADDR(len)[,ADDR(num)]...);
          CALL PLCALL(SORT3,n,cstmt,ADDR(loc1),
                      ADDR(loc2),ADDR(len),ADDR(loc3),
                      ADDR(len3)[,ADDR(num)]...);
          CALL PLCALL(SORT4,n,cstmt,ADDR(loc4),ADDR(loc2)
                      [,ADDR(num)]...);
```

Parameters:

- _cstmt_ is the location of the control statement.
- _loc1_ is the location of the first element of the data set or array to be sorted.
- _loc2_ is the location of the last element of the data set or array to be sorted.
- _len_ is the location of the fullword integer length of each element in the data set to be sorted. The value of _len_ may range between 1 and 256 bytes.
- _num_ (optional) is the location of a positive, nonzero, fullword integer that specifies a numeric value in the control statement.
- _loc3_ is the location of the first element in the tag data set or array.
- _len3_ is the location of the fullword integer length of each element of the tag data set. The value of _len3_ may range between 1 and 256 bytes.
- _loc4_ is the location of the first element of the data set or array of 4-byte addresses to be

sorted according to attributes of the data referenced by the addresses.

err (optional) is the statement label to transfer to if an error (nonzero return code) is detected by the subroutine.

n is the number of arguments (FIXED BINARY(31)) to be passed to the subroutine.

Return Codes:

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

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

Summary of the Control Statement

Prototype:

```
[[SORT][=[[type],[aspect],[location],[length],]...
               [type][,[aspect][,[location][,[length]]]]]]
[DS=delimiter[string]delimiter▪]...
[additional parameter]...
END▪
```

Collating fields:

TYPE	CODE	SIGN PRESENT	FIELD LENGTH (BYTES)
alignment	AL	no	1 - 4095
binary	BI	no	1 - 256
bit	BT	no	1 - 255 (mask)
call	CA	-	1 - 4095
character	CH	no	1 - 256
defined sequence	DS(i)	no	1 - 256
fixed-point	FI	yes	1 - 260
floating-point	FL	yes	2 - 16
packed decimal	PD	yes	1 - 16
signed decimal	SD	yes	1 - 17
zoned decimal	ZD	yes	1 - 16

Additional parameter:

DEC (delete comments)

SORT4F

Subroutine Description

Purpose: To sort an array of FORTRAN indexes such that if the data referenced by the indexes were substituted for the indexes, the data would be in the order described by the control statement.

Location: *LIBRARY

Calling Sequence:

FORTRAN: CALL SORT4F(cstmt,loc1,loc2,dim,array,dimary, len[,num]...[,&err1[,&err2]])

Parameters:

- _cstmt_ is the location of the SORT control statement, which has the same requirements and restrictions as for SORT4.
- _loc1_ is the location of the first element of the _dim_-by-N, INTEGER*4 array containing the subscripts to be sorted. Each of the N columns of this array contains a set of subscripts for an element in _array_.
- _loc2_ is the location of the last element of the array containing the subscripts to be sorted. If the subscripts for the first element of this array are (1,1), the subscripts for the last element will be (_dim_,N).
- _dim_ is the location of the INTEGER*4 number of dimensions for _array_.
- _array_ is the location of the array containing the data referenced by the subscripts to be sorted.
- _dimary_ is the location of the first element of a _dim_-element, INTEGER*4 array containing the size of each dimension of _array_.
- _len_ is the location of the INTEGER*4 length of each element of _array_.
- _num_ (optional) is the location of a positive, nonzero, INTEGER*4 specification of a numeric value in the control statement.
- _err1_ (optional) is the statement label to transfer to if an error is detected by SORT4.
- _err2_ (optional) is the statement label to transfer to if a parameter error is detected by SORT4F. These errors include _loc1_ or _loc2_ not being the location of an appropriate

array element, an index being greater than the size of the corresponding dimension specified in dimary, and excessive dimary or len values.

Description: The indexes in the array delimited by loc1 and loc2 are converted to addresses which are passed to SORT4. On return from SORT4, the addresses are converted back to indexes. If an error is detected, the values in the index array will be invalid.

Examples: FORTRAN:

```
      INTEGER DIM(2)/5,256/,INDEX(2,256),
     1  NAMES(5,256)
      ...
      DO 1010 I=1,N
        INDEX(1,I)=1
1010    INDEX(2,I)=I
      CALL SORT4F('SORT=CH,A,1,20 END ',
     1  INDEX(1,1),INDEX(2,N),2,NAMES,DIM,4,
     2  &9910,&9900)
      WRITE (6,2000) ((NAMES(I,INDEX(2,J)),
     1  I=1,5),J=1,N)
2000  FORMAT (1X,5A4)
      ...
9900  WRITE (6,9990)
9910  STOP
9990  FORMAT (' SORT4F ERROR')
      ...
```

The above example generates indexes for the N, 20-character names in the array NAMES, sorts the indexes, and prints the names in alphabetical order.

```
      INTEGER INDEX(256),NAMES(5,256)
      ...
      DO 1010 I=1,N
1010    INDEX(I)=I
      CALL SORT4F('SORT=CH,A,1,20 END ',INDEX,
     1  INDEX(N),1,NAMES,256,20,&9910,&9900)
      WRITE (6,2000) ((NAMES(I,INDEX(J)),
        I=1,5),J=1,N)
2000  FORMAT (1X,5A4)
      ...
9900  WRITE (6,9990)
9910  STOP
9990  FORMAT (' SORT4F ERROR')
      ...
```

The above example is the same as the preceding one except that the call on SORT4F assumes that NAMES is a 1-dimensional array with elements of length 20.

SPELLCHK

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

Location: Resident System

Alt. Entry: SPELCK

Calling Sequences:

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

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

Parameters:

- _goodwd_ is the location of the word that is known to be correctly spelled.
- _testwd_ is the location of the word that is to be compared against _goodwd_.
- _goodl_ is the location of a fullword integer (INTEGER*4) giving the length of _goodwd_. The length must be between 1 and 32 (inclusive).
- _testl_ is the location of a fullword integer (INTEGER*4) giving the length of _testwd_. The length must be between 1 and 32 (inclusive) and must not differ from _goodl_ by more than 1.

Values Returned:

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

Return Codes:

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

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

The algorithm will detect spelling errors consisting of:

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

Examples: Assembly:

```
          CALL  SPELLCHK,(=C'GOOD',TEXT,4,N)
          ST    0,I
           .
           .
    TEXT  DS    CL4
    N     DS    F
    I     DS    F
```

FORTRAN:

```
      INTEGER SPELCK
      LOGICAL*1 TEXT(4)
       ...
      I = SPELCK('GOOD',TEXT,4,N)
```

The above example, coded in assembly language and FORTRAN, check the character string contained in TEXT against the string "GOOD".

SPIE

Subroutine Description

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

Location: *LIBRARY

| Alt. Entry: SPIES

Calling Sequences:

Assembly:
```
LA   1,pica
CALL SPIE

CALL SPIES,(pica,oldpica),VL
```

Note: This subroutine is normally called by using the SPIE macro. See the SPIE macro description in MTS Volume 14, _360/370 Assemblers in MTS_.

FORTRAN: CALL SPIES(pica,oldpica,&rc4)

Parameters:

pica (GR1) is the location of a 6-byte region containing the program interrupt control area. The first byte contains the bits that are to be set into the program mask in the PSW. When a bit is set, the corresponding interrupt type is enabled and can occur. The bits are:

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

The next three bytes contain the address of the exit routine to be given control after a program interrupt of the type specified in

----

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

the interruption mask. The last two bytes contain the interruption mask for the program interrupt types to cause control to be given to the exit routine. Each bit corresponds to a program interrupt type. These are:

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

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

| A call on SPIE with pica containing zero cancels the effect of the previous call.

| oldpica is a region to store the address of the
| previous PICA.

Value Returned:

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

| Return Codes:
|
| 0 Successful return.
| 4 Invalid parameter or no VL bit specified.

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

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

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

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

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

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

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

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

A call on the SPIES subroutine takes the S-type parameters and loads them into an R-type call on the SPIE subroutine.

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

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

The same example using the SPIE macro.

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

SPRINT

Subroutine Description

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

Location: Resident System

Alt. Entry: SPRINT#

Calling Sequences:

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

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

Parameters:

- <u>reg</u> is the location of the virtual memory region from which data is to be transmitted.
- <u>len</u> is the location of a halfword (INTEGER*2) integer giving the number of <u>bytes</u> to be transmitted.
- <u>mod</u> is the location of a fullword of modifier bits used to control the action of the subroutine. If <u>mod</u> is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.
- <u>lnum</u> (optional) is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.
- <u>rc4,...</u> is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

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

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

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

If len is zero when writing to a line file, the line is deleted from the file.

The default FDname for SPRINT is *SINK*.

There is a macro SPRINT in the system macro library for generating the calling sequence to this subroutine. See the macro description for SPRINT in MTS Volume 14, 360/370 Assemblers in MTS.

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

Assembly:

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

                or

                SPRINT REG       Subr. call using macro
```

FORTRAN:

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

SPUNCH

Subroutine Description

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

Location: Resident System

Alt. Entry: SPUNCH#

Calling Sequences:

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

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

Parameters:

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

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

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

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

_rc4,..._ is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

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

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

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

If len is zero when writing to a line file , the line is deleted from the file.

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

There is a macro SPUNCH in the system macro library for generating the calling sequence to this subroutine. See the macro description for SPUNCH in MTS Volume 14, 360/370 Assemblers in MTS.

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

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

                 or

                 SPUNCH REG       Subr. call using macro

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

SRCHI

Subroutine Description

Purpose: To perform a binary-search based on the results of user-supplied comparisions of the search argument and successive subroutine-selected elements of an ordered list.

Location: *LIBRARY

Calling Sequences:

Assembly:
```
CALL SRCHI0,(num)
CALL SRCHI,(switch)
```

FORTRAN:
```
INTEGER*4 SRCHI
CALL SRCHI0(num)
index=SRCHI(switch)
```

PL/I(F):
```
DECLARE PLCALLF RETURNS(FIXED BINARY(31));
CALL PLCALL(SRCHI0,f1,ADDR(num));
index=PLCALLF(SRCHI,f1,ADDR(switch));
```

Parameters:

num — is the location of the fullword integer specifying the number of elements in the ordered list to be searched.

switch — is the location of a fullword switch indicating whether the search value precedes or follows the comparand specified by the index returned by the previous call on SRCHI or whether a new search is to begin. The choices are:

- 0 Initialize a search of an ordered list of num elements and return the index of the first comparand of the search (the "middle" element).
- >0 The search argument value is greater than the comparand specified by the index returned by the previous call on SRCHI.
- <0 The search argument value is less than the comparand specified by the index returned by the previous call on SRCHI.

f1 — is a fullword (FIXED BINARY(31)) containing the integer 1.

Values Returned:

index is the location of a fullword integer containing the index of the next ordered list value to be compared with the search value. (The first element of the ordered list has index 1; the last element of the list has index _num_.) The return value possibilities are as follows:

<0 The ordered list is exhausted. The absolute value of this number is the index of the list element where the search value could be inserted to maintain the list order. If this value is "-i", then the search value lies between the list values with indices "i-1" and "i".

0 Either (1) SRCHI0 was not called or was called with a negative argument _num_, or (2) SRCHI was not called with a zero switch argument either after SRCHI0 was called or after SRCHI returned a negative index indicating list exhaustion.

>0 The value indicates which element of the ordered list is to be examined next.

For assembly language programs, this value will be returned in general register 0.

Description: The index values returned by the SRCHI subroutine indicate which elements of an ordered list should be examined while performing a binary search. Note that if the list has "n" elements, then the maximum number of comparisons for a binary search is log(base 2)n=log n/log 2. In contrast, the average number of comparisons for a sequential search is "n/2" for uniformly distributed search values. Hence, for large lists, the binary search method is far more efficient than simple linear sequential searches. For example, a binary search of a 256-element list will have at most 8 comparisons while a linear search of that list will have, on the average, 128 comparisons with uniformly-distributed search values. Tests using a FORTRAN array indicate that the use of SRCHI may produce more efficient results than a linear search when the number of elements in the array is approximately 32 or greater.

Because only the calling program accesses the list elements, the list may have any data structure of any size with data types of the user's choice. For example, the list need not be an array, but its elements should be accessible via some user-formulated index function of the SRCHI-returned index.

The list elements must be ordered according to the rules used to determine the value of <u>switch</u>. The element having the value which precedes all other values in the list must be the first element of the list, etc. In the case of arrays, it may be possible to produce the required ordering by calling SORT2, SORT3, SORT4, or SORT4F prior to beginning the search portion of the program.

Examples: FORTRAN:

```
      INTEGER*4 DIFF, SRCHI
        ...
C     Define the list size.
      CALL SRCHI0(NUM)
        ...
C     Initialize the search.
      SWITCH=0
C     Produce an element index.
10    INDX=SRCHI(SWITCH)
C     Check for exhausted list, invalid
C     argument, or valid new index.
      IF (INDX) 30,40,20
C     Compare indexed value with search value.
20    SWITCH=KEY-LIST(INDX)
C     If KEY value not found, continue search.
      IF (SWITCH.NE.0) GO TO 10

        ...

C     This section executed if KEY=LIST(INDX).

30      ...

C     This section executed if KEY is NOT in
C     LIST.  If KEY were to be inserted in LIST,
C     it would be the (-INDX)th element of LIST.

40      ...

C     This section is executed if SRCHI
C     is not properly initialized.
```

The above example searches the integer array LIST of N elements for a value equal to KEY.

PL/I(F):

```
     DECLARE (DIFF,INDX,KEY,N) FIXED BINARY(31);
     DECLARE F1 FIXED BINARY(31) INIT(1);
     DECLARE PLCALLF RETURNS
             (FIXED BINARY(31));
     DECLARE (SRCHI0,SRCHI) ENTRY;
     DECLARE SWITCH BIT(1);
       ...
/*   Define the list size.  */
     CALL PLCALL(SRCHI0,F1,ADDR(N));
```

```
          ...
     /*   Initialize the search.  */
          DIFF=0;
          SWITCH='1'B;
          DO WHILE(SWITCH);
     /*   Produce an index.  */
            INDX=PLCALLF(SRCHI,F1,ADDR(DIFF));
            IF INDX>0 THEN DO;
              DIFF=KEY-LIST(INDX);
              SWITCH=DIFF¬=0;
            END;
            ELSE SWITCH='0'B;
          END;
          IF INDX>0 THEN DO;
           ...
     /*   This section executed if KEY=LIST(INDX) */
           ...
          END;
          ELSE IF INDX<0 THEN DO;
           ...
     /*   This section executed if KEY is NOT in LIST
          If KEY were inserted in LIST, it would be
          the (-INDX)th element in LIST. */
           ...
          END;
          ELSE DO;
           ...
     /*   This section executed if SRCHI is not
          properly initialized. */
           ...
          END;
```

The above PL/I(F) example performs the same search as the preceding FORTRAN example.

STARTF

Subroutine Description

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

Location: Resident System

Calling Sequence:

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

Parameters:

- id — is the location of the fullword integer storage index number of the program that was dynamically loaded by LOADF (the value returned by LOADF), or is the location of an 8-character entry point name, left-justified with trailing blanks.
- par1,par2,... — (optional) are the parameters to be passed to the program being executed. There may be any number of parameters passed, including none.

Values Returned:

None.

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

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

Example:

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

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

STDDMP

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

Parameters:

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

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

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

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

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

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

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

Return Codes:

0 Successful return.
4 Illegal parameters.

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

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

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

Example: Assembly:

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

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

SVCTRP

Subroutine Description

Purpose: To suspend program execution whenever an SVC instruction is executed by a user program.

Location: Resident System

| Alt. Entries: SVCTRPS, SVCTPS

Calling Sequences:

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

          CALL SVCTRPS,(exit,region),VL

FORTRAN:  CALL SVCTPS(exit,region,&rc4)
```

Parameters:

exit	(GR0) should be zero or the location to transfer to if an SVC instruction is executed.
region	(GR1) should should contain the location of a 72-byte save region for storing pertinent information.
&rc4	(optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Illegal parameter or no VL bit specified.

Description: A call on the subroutine SVCTRP sets up an SVC intercept for one SVC instruction only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of an SVC instruction in the user program. When an SVC instruction is encountered and the exit is taken, the intercept is cleared so that another call to SVCTRP is necessary to intercept the next SVC instruction. When a SVC instruction occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location specified by the GR0 value in the call to SVCTRP. If the exit subroutine returns to MTS (BR 14), MTS will declare the SVC instruction invalid, suspend program execution, and print a message providing the location of the intercept.

If GR0 is zero on a call to SVCTRP, the SVC intercept is disabled. GR1 should point to a valid save region in this case also.

When the SVC intercept exit is taken, the first eight bytes of the save region contain the PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the intercept. The PSW stored in the savearea is always in BC mode (bit 12 is zero). The floating-point registers remain as they were at the time of the intercept. GR1 will contain the location of the save region. The contents of GR0 and GR2 to GR12 are unpredictable.

If, on a call to SVCTRP, the first byte of the save region is X'FF', SVCTRP does not return to the calling program; rather the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on SVCTRP, after the first SVC instruction has been intercepted, allows the user to set a switch (for example) and to return to the point following the SVC instruction with the intercept again enabled.

The SVCTRP item of the GUINFO/CUINFO subroutine may be used to save a previously set exit to allow nesting of SVC intercepts.

Note: This subroutine will intercept only SVC instructions that are executed by the user's program; it will not intercept those that are executed by the operating system.

| A call on the SVCTRPS or SVCTPS subroutines takes the
| S-type parameters and loads them into an R-type call on
| the SVCTRP subroutine.

Example: In this example, the location of the first SVC instruction in a user program is recorded and execution is resumed with at the SVC instruction.

```
         LM   0,1,=A(EXIT,REGION)
         CALL SVCTRP     The intercept is enabled.
          ...
         USING EXIT,15
EXIT     L    0,4(,1)    Get address of SVC
         SL   0,=F'2'    Back up to SVC instruction
         ST   0,FIRSVC   Remember location
         ST   0,4(,1)    Restart at SVC
         MVI  0(1),X'FF'
         SR   0,0        Disable further intercepts
         CALL SVCTRP     Note GR1 points to REGION
REGION   DS   18F
FIRSVC   DS   A
```

<u>SYSTEM</u>

Subroutine Description

Purpose: To terminate execution successfully.

Location: Resident System

Alt. Entry: SYSTEM#

Calling Sequence:

Assembly: CALL SYSTEM

or

SYSTEM

FORTRAN: CALL SYSTEM

Note: The complete description for using the SYSTEM macro is given in MTS Volume 14, <u>360/370 Assemblers in MTS</u>.

Description: The SYSTEM subroutine terminates execution and returns control to MTS or to the previous command language subsystem.

The execution return code is set to 0. This may be tested by the $IF command, e.g.,

```
$IF RUNRC=0, mts-command
```

The execution return code and the message "EXECUTION TERMINATED" is displayed under the control of the $SET RCPRINT and ETM options (see MTS Volume 1, <u>The Michigan Terminal System</u>) and the GUINFO item LASTEXRC (239).

Execution that is terminated by this subroutine cannot be restarted by the $RESTART command. Calling this subroutine is equivalent to the program doing a normal return (BR 14) from the call that started execution.

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

TAPEINIT

Subroutine Description

Purpose: To initialize a labeled or unlabeled magnetic tape.

Location: *LIBRARY

Alt. Entry: TPINIT

Calling Sequences:

```
Assembly: CALL TAPEINIT,(tape,mode,volume,owner,
                         lbltype),VL

FORTRAN:  CALL TPINIT(tape,mode,volume,owner,lbltype,
                      &rc4,&rc8,&rc12,&rc16,&rc20)
```

Parameters:

<u>tape</u> is the location of either
(a) an FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS)
for the tape which is to be initialized.

<u>mode</u> is the location of the 4-character density at which the tape is to be mounted (e.g., ’1600’, ’6250’, ’556 ’).

<u>volume</u> is the location of a 6-character volume name. This parameter may be omitted.

<u>owner</u> is the location of a 10-character ownerid. This parameter may be omitted.

<u>lbltype</u> (optional) is the location of a 6-character label type (e.g., "OS/VS ", "VLO   ", or "ANSI ").

<u>rc4,...,rc20</u> (optional) are statement labels to transfer to if a nonzero return codes occur.

Return Codes:

0 Successful return--tape was initialized.
4 <u>tape</u> does not specify a labelable tape--it was not a magnetic tape, it was mounted without the file protect ring in, or it was a pool tape.
8 <u>mode</u> was not valid for the tape drive on which the tape was mounted.
12 Write error occurred while attempting to initial-

ize the tape.
16 volume is invalid (contains an embedded comma or blank).
20 owner was not valid (a program interrupt occurred while attempting to access it).

Description: The tape must have been mounted with WRITE=YES or RING=IN specified on the mount request.

If volume and owner are omitted, the tape is initialized as an unlabeled tape, i.e., label processing is disabled, 6 filemarks are written at the specified density at the beginning of the tape, and the tape is rewound. If volume is given, (1 to 6 characters without embedded commas or blanks, padded to 6 character with trailing blanks as necessary), the tape is initialized as a labeled tape, is rewound, and label processing is enabled. owner will be included in the label as the ownerid if it is given; otherwise, the ownerid will be blanks.

The label type parameter specifies that the tape is to be labeled according to the IBM standard if the lbltype is OS/VS or VLO, or specifies that the tape is be labeled according to the American National Standard Institute (ANSI) exchange format if lbltype is ANSI. If lbltype is omitted, OS/VS is assumed.

Assembly language users wishing to omit optional parameters should either follow the variable-length parameter list convention (the high-order bit of the last parameter adcon present in the parameter list is set to 1), or else supply zero adcons.

Examples:

Assembly:

```
          CALL TAPEINIT,(TWO,MODE),VL
           .
           .
     TWO  DC   F'2'
     MODE DC   CL4'800'
```

FORTRAN:

```
          CALL TPINIT,(2,'800 ',&99)
           ...
     99   CALL ERROR
```

Each of the above examples initializes the tape attached to logical I/O unit 2 as an unlabeled tape at 800 bpi.

```
Assembly:        CALL TAPEINIT,(TFDUB,MODE,VOL,OWNER),VL
                  .
                  .
          TFDUB  DS   A
          MODE   DC   CL4'6250'
          VOL    DC   CL6'TAPE1'
          OWNER  DC   CL10'UOFMICH'

FORTRAN:       CALL TPINIT(TFDUB,'6250','TAPE1 ',
                         'UOFMICH    ',&999)
                 ...
          999    ...
```

Each of the above examples initializes the tape whose FDUB-pointer is in TFDUB as an OS/VS labeled tape at 6250 bpi with volume name TAPE1 and ownerid UOFMICH.

TICALL

Subroutine Description

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

Location: *LIBRARY

Calling Sequence:

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

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

Parameters:

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

0 value is an 8-byte integer which specifies a time interval in microseconds of task CPU time, relative to the time of the call.
1 value is an 8-byte binary integer which specifies a time interval in microseconds of real time, relative to the time of the call.
2 value is an 8-byte binary integer which specifies a time interval in microseconds of task CPU time, relative to the time at signon.
3 value is an 8-byte binary integer which specifies a time interval in microseconds of real time, relative to the time at signon.
4 value is a 4-byte binary integer which specifies a time interval in timer units (13 1/48 microseconds per unit) of task CPU time, relative to the time of the call.
5 value is a 16-byte EBCDIC string giving the time and date at which the interrupt is to occur, in the form HH:MM.SSMM-DD-YY.

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

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

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

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

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

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

Return Codes:

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

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

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

Example:

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

SUBROUTINE TIMOUT
 ...
(Process interrupt and reenable interrupts)
 ...
RETURN
```

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

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

TIME

Subroutine Description

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

Location: Resident System

Alt. Entry: MTSTIME

Calling Sequences:

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

FORTRAN: CALL TIME(key,pr,res,&rc4,&rc8)

Parameters:

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

0 the CPU, elapsed, supervisor, and problem state times are initialized (see below).
1 the CPU time in milliseconds is returned as a fullword integer in <u>res</u>.
2 the elapsed time in milliseconds is returned as a fullword integer in <u>res</u>.
3 the CPU time in milliseconds is placed in the first word of the fullword-integer array <u>res</u> and the elapsed time in milliseconds is placed in the second word of <u>res</u>.
4 the time of day is returned in <u>res</u> as an 8-character value in the form "hh:mm:ss".
5 the date is returned in <u>res</u> as a 12-character value in the form "mmm dd, 19yy". If "dd" is less than 10, the leading zero is replaced by a blank.
6 the time of day is placed in the first 8 characters of <u>res</u> (see <u>key</u>=4) and the date is placed in the 9th through 20th characters of <u>res</u> (see <u>key</u>=5).
7 the supervisor state CPU time in seconds multiplied by 300x256 is placed in <u>res</u> as a fullword integer.
8 the problem state CPU time in seconds multiplied by 300x256 is placed in <u>res</u> as a fullword integer.
9 the supervisor state CPU time (see <u>key</u>=7) is

placed in the first word of the fullword-integer array res and the problem state CPU time (see key=8) is placed in the second word of res.

10 the date is returned in res as an 8-character value in the form "mm-dd-yy".

11 the time of day is placed in the first 8 characters of res (see key=4 above) and the date is placed in the 9th through 16th characters of res (see key=10 above).

12 the date is placed in the first 8 characters of res (see key=10 above) and the time of day is placed in the 9th through 16th characters of res (see key=4 above).

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

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

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

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

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

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

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

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

21 the date is returned in res as a 16-character value in the form "www mmm dd/yy ", where "www" are the first three characters of the day of the week.

22 the date (see key=21) is placed in the first 16 characters of res and the time of day (see key=4) is placed in the 17th through 24th characters of res.

23 the current number of microseconds starting

with March 1, 1900, 00:00:00.000001 as "1" is placed in the first and second words of the fullword-integer array res as a 64-bit integer, the date in the form "mm-dd-yy" (see key=10) is placed in the third and fourth words of res, the date in the form "www mmm dd/yy " (see key=21) is placed in the fifth through eighth words of res, and the time of day in the form "hh:mm:ss" (see key=4) is placed in the ninth and tenth words of res.

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

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

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

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

If pr is 0, the values are returned.

res is the location of a fullword integer or vector or a character string, as appropriate, in which the results are placed.

rc4,rc8 (optional) are statement labels to transfer to if a nonzero return code occurs.

Values Returned:

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

FR2 contains the doubleword, real, second value in seconds if a second returned value is numeric (key=3, 9, 17, 20).

Return Codes:

0 Successful return.
4 Error, due to an improper value for key.
8 System error (should not occur).

Index to key Values:

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

Examples: Assembly:

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

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

FORTRAN: CALL TIME(5,1)

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

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

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

Time Routines

Subroutine Description

The time routines are used to perform time and date conversions between MTS internal formats, general character-string formats, and "exploded" formats.

Three subroutines are provided in this package:

TIMEIN	To convert an MTS internal or character-string time and date into an exploded format.
TIMEOUT	To convert an exploded time and date into an MTS internal or character-string format.
TIMEGIN	To convert a general character-string time and date into an exploded format.

MTS Internal Time and Date Formats

Time and dates can be represented internally in MTS in several "standard" formats. These are either 4-byte or 8-byte quantities giving the time and date in various units such as microseconds, minutes, or days since March 1, 1900.

The *format* parameter for the TIMEIN and TIMEOUT subroutines points to a character string that specifies the particular internal format being used. This character string may be:

MICROSECONDS (or *MMS*)

The time and date is expressed as an 8-byte field containing the number of microseconds that have elapsed since March 1, 1900 00:00:00.000000.

MILLISECONDS (or *MS*)

The time and date is expressed as an 8-byte field containing the number of milliseconds that have elapsed since March 1, 1900 00:00:00.000.

SECONDS (or *S*)

The time and date is expressed as a 4-byte field containing the number of seconds that have elapsed since March 1, 1900 00:00:00.

MINUTES (or *M*)

The time and date is expressed as a 4-byte field containing the number of minutes that have elapsed since March 1, 1900 00:00.

HOURS (or *H*)

The time and date is expressed as a 4-byte field containing the number of hours that have elapsed since March 1, 1900 00:.

DAYS (or *D*)

The time and date is expressed as a 4-byte field containing the number of days that have elapsed since March 1, 1900.

IBM MICROSECONDS (or *IBMMMS*)

The time and date is expressed as an 8-byte field containing the time and date as returned by the STCK instruction in 370-assembler language. The STCK instruction returns the number of microseconds*4096 that have elapsed since January 1, 1900 00:00:00.000000 GMT (see the IBM publication, IBM System/370 Principles of Operation, form GA22-7000, for details.

In all the above forms, except for *IBM MICROSECONDS*, time-zone information is not given and for many applications is not needed. However, time-zone information can be included by the use of one of the following modifiers.

(1) @GMT specifies that the Julian time/date is based from March 1, 1900 00:00 GMT.
(2) @UT is same as above but is followed by a 2-byte field that contains the offset (in minutes) of the time zone of the time/date from which its Julian representation was computed and an 8-byte field, left-justified and padded with blanks, that contains the character abbreviation of the original time zone.
(3) @TAGGED specifies that following the Julian time/date there is an 8-byte field, left-justified and padded with blanks, that contains the character representation of the time zone in which the Julian quantity was computed.
(4) @TZ=zzz specifies that the Julian time/date is to interpreted as being computed for the time zone specified by "zzz" (i.e., "zzz" can be EST, CST, EDT, etc.). This modifier is valid only for the TIMEIN subroutine.
(5) @TZ=LOCAL specifies that the Julian time/date is to be interpreted as being computed in the current local time zone. At U of M, this would either be EST in winter and EDT in summer. This modifier is valid only for the TIMEIN subroutine.

For example, *MINUTES@GMT* indicates a time/date in GMT. If no modifier is specified, the time/date contains no time-zone information and therefore cannot be used in a time-zone transformation.

Note that in all the above forms, the @L=val modifier may be also used to change the default lengths of the Julian time/dates. "val" may be either 2, 4, or 8 and specifies the length in bytes of the time/date. For example, *HOURS@L=8* causes the time/date to be 8 bytes long rather than the usual four bytes, and *SECONDS@L=8@GMT* causes the time/date to be 8 bytes long and specifies the time is in GMT.

Character-String Time and Date Formats

To specify a time and date in a more easily readable character format, the conventions covered below are used to describe the different formats available. A character-string time and date consists of three parts, the time, the date, and the weekday, all of which may or may not be present. The fashion in which these components are specified is described below.

The time component of a character-string time and date can be built up of the following picture elements.

(1a) HH is a two-digit hour number.
H+ is a one- or two-digit hour number.

(1b) HH.[H...][+...], where HH is a two-digit hour number, [H...] represents the number of fractional hour positions that must be present, and [+...] represents the additional fractional positions that may be present if significant. The nonsignificant positions are assumed to be nulls. The number of H's and +'s specified must be greater than or equal to one and less than or equal to six.
H+.[H...][+...] is the same as above but the hour number can be one or two digits long.

(2a) MM is a two-digit minute number.
M+ is a one- or two-digit minute number.

(2b) MM.[M...][+...], where MM is a two-digit minute number, [M...] represents the number of fractional minute positions that must be present, and [+...] represents the additional fractional positions that may be present if significant. The nonsignificant positions are assumed to be nulls. The number of M's and +'s must be greater than or equal to one and less than or equal to six.
M+.[M...][+...] is the same as above but the minute number can be one or two digits long.

(3a) SS is a two-digit second number.
S+ is a one- or two-digit second number.

(3b) SS.[S...][+...], where SS is a two-digit second number, [S...] represents the number of fractional second positions that must be present, and [+...] represents the additional fractional positions that may be present if significant. The positions that are not significant are assumed to be nulls. The number of S's and +'s must be greater than or equal to one and less than or equal to six.
S+.[S...][+...] is the same as above but the second number can be one or two digits long.

(4) A is an am/pm meridian marker in the form of "a" or "p".
AM is an am/pm meridian marker in the form of "am" or "pm".
A.M. is an am/pm meridian marker in the form of "a.m." or "p.m.".

(5) ZZZ[Z...][+...] is used to specify the presence of a time-zone marker. The Z's represent the number of characters that must be present in the time-zone marker and the +'s represents the number of characters that may or may not be present in the marker. The maximum number of Z's and +'s that can be used is 8.

Thus, a time may be specified by putting together appropriate picture elements. A delimiter may occur between picture elements. For time elements, the valid delimiters are ":", ".", and blank. The ":" and "." delimiters are valid between numeric time picture elements and the blank is valid to delimit the time from a meridian marker and/or time-zone marker. The delimiter is required after a time picture element only if the picture element is of variable length. The following is a list of valid combinations of picture elements:

(a) (1a) optionally with (4) and/or (5). That is, an hour possibly followed by a meridian marker and/or time-zone marker, e.g.,

```
H+ AM ZZZ  which would describe
6 am EDT
12 pm EST
4 PM CST

HH ZZZ  which would describe
06 EDT
12 EST
16 CST
```

(b) (1b) optionally with (4) and/or (5). That is, an hour followed by fractional hours and possibly followed by a meridian marker and/or time-zone marker, e.g.,

```
HH.H++ AM ZZZ
06.5 am EDT
12.333 PM EST
```

(c) (1a) with (2a) and optionally with (4) and/or (5). That is, an hour followed by minutes and optionally a meridian marker and/or time-zone marker, e.g.,

```
H+:MM A  which would describe
10:15 a
3:30 p
```

(d) (1a) with (2b) and optionally with (4) and/or (5). That is, an hour followed by minutes and fractional minutes and possibly terminating with a meridian marker and/or time-zone marker, e.g.,

```
HH:MM.MM A.M.
10:15.25 a.m.
03:30.50 p.m.
```

(e) (1a) with (2a) with (3a) and optionally with (4) and/or (5). That is, an hour followed by minutes and seconds that can optionally be followed by a meridian marker and/or time-zone marker, e.g.,

```
H+:MM.SS A.M. ZZZ  which would describe
9:20.00 a.m. EST
11:30.45 P.M. EST

HHMMSS  which would describe
092000
233045
```

(f) (1a) with (2a) with (3b) and optionally with (4) and/or (5). That is, an hour followed by minutes, seconds, and fractional seconds possibly followed by a meridian marker and/or time-zone marker, e.g.,

```
H+:MM.SS.S++ A.M. ZZZ  which would describe
9:20.00.0 a.m. EST
11:30.45.25 P.M. EST
```

Note that if a meridian marker is not used in specifying a date, a 24-hour clock is assumed, whereas if one is present, a 12-hour clock is used. Also note that the order of the elements in the above list cannot be varied.

In a similar fashion, a date can be made up of picture elements and appropriate delimiters, if desired, in two types of date formats.

The first form is numeric. In this form, the date, except for delimiters, is made up entirely of numeric characters. Valid delimiters between elements in a numeric date are the "-", "/", and ".". The "-" and "/" delimiters are valid in calendar forms (month, day, and year) and the "." delimiter is valid in the OS-date forms (year and day of year). A delimiter is not required after a numeric date picture unless the element is of variable length.

The second form of the date is the character date. The character date has the month element in character format and the day and year elements in numeric format. Valid delimiters between elements in a character date are " ", ",", and "/". A delimiter is not required after a date picture elements unless it is numeric and of variable length. With character dates, unlike times and numeric dates, more than one delimiter may be used to separate the picture elements.

The following is a list of valid date picture elements.

(1a) YYYY is a four-digit year number.
(1b) YY is the last two digits of the year number.
(2a) MM is a two-digit month number (valid only in numeric forms of the date).
M+ is a one- or two-digit month number (valid only in numeric

forms of the date).

(2b) MMM is a three-letter month abbreviation (valid only in character forms of the date).
MMMB is a three-letter month abbreviation followed by either a blank or "." depending whether if the name of the month abbreviated (valid only in character forms of the date).
MMMN is a three-letter month abbreviation followed by an optional "." depending on whether the name of the month was abbreviated (valid only in character forms of the date).
MMM++++++ is a variable-length name of the month which must be fully spelled out (valid only in character forms of the date).
MMMMMMMMM is a nine-character, left-justified, fully spelled out month name with trailing blanks in unused positions (valid only in character forms of the date).

(3a) DD is two-digit day of month (valid in all but the OS-date form.)
D+ is a one- or two-digit day of the month (valid in all but the OS-date form).

(3b) DDD is a three-digit day of year number (valid only in OS-date form).
D++ is a one- to three-digit day of year number (valid only in OS-date form).

The following is a list of how the above picture elements can be combined to create the various date forms recognized by this subroutine. Note that unlike the time forms, the picture elements in a date can be specified in any order.

(a) (1a) with (2a) and (3a); numeric form (month, day, and year), e.g.,

```
MM/DD/YYYY  which would describe
01/05/1982
06/30/1983
12/31/1984

D+-M+-YYYY  which would describe
5-1-1982
30-6-1983
31-12-1984
```

(b) (1b) with (2a) and (3a); numeric form (month, day, and year).

```
M+-DD-YY
1-05-82
6-30-83
12-31-84
```

(c) (1a) with (3b); numeric form (OS date), e.g.,

```
YYYY.DDD  which would describe
1982.005
```

```
        1983.181
        1984.365
```

(d) (1b) with (3b); numeric form (OS date), e.g.,

```
        YY.D++  which would describe
        82.05
        83.181
        84.365
```

(e) (1a) with (2b) and (3a); character form (month, day, and year), e.g.,

```
        MMMB DD, YYYY  which would describe
        Jan. 05, 1982
        June 30, 1983
        Dec. 31, 1984
```

(f) (1b) with (2b) and (3a); character form (month, day, and year), e.g.,

```
        MMMN DD/YY  which would describe
        Jan. 05/82
        June 30/83
        Dec. 31/84
```

The final component that can appear in a date is the weekday. The following picture elements can be used to specify a weekday.

(a) WW is a two-letter weekday abbreviation.
(b) WWW is a three-letter weekday abbreviation.
(c) WWW++++++ is a variable-length name of weekday fully spelled.
(d) WWWWWWWWW is a nine-character, left-justified name of weekday fully spelled with unused portion padded with blanks.

A complete time and date consists of the above components combined with separators between the components, optionally before the first component, and optionally after the last component. Note that although separators are not normally required after the last component in a time/date, if the last component ends with a variable picture element, at least one separator must follow. The separators that may be chosen by the user with the stipulation that a component that ends in a variable-length element cannot be followed by a separator string whose first character is alphanumeric. The separator strings are defined by a starting prime (’) or quote ("), followed by an arbitrary string, and ending with a prime or quote. Separator strings whose characters are not alphanumeric need not be delimited by primes or quotes. The order of the components in an external time/date is optional. The following is a list of valid components that make up a time/date.

(1) Time, date, and weekday.
(2) Time and date.

(3) Date.
(4) Time.

Some examples are as follows:

```
(a)  WWW., MM-DD-YY HH.HH+' ' which would describe
     Tue., 01-05-82 13.25
     THU., 06-30-83 01.625
     Mon., 12-31-84 23.50

(b)  MMMN D+ YYYYHH:MM:SS AM which would describe
     Jan. 5 198201:15:00 PM
     Jun. 30 198301:37:30 AM
     DEC. 31 198423:30:00 PM

(c)  MMMN D+ YYYY" at "HH:MM:SS A.M.  which would describe
     Jan. 5 1982 at 01:15:00 P.M.
     Jun. 30 1983 at 01:37:30 A.M.
     DEC 31 1984 at 11:30:00 P.M.
```

Any of the above patterns used to specify an external time and date of an external form is referred to as a time pattern. Thus, to specify the form of an external time and date, format would point to:

```
time pattern*
```

The interpretation of an external time and date can be modified by the presence of certain modifiers in the above as displayed below.

```
time pattern@mod1@mod2...*
```

The possible modifiers for the TIMEIN subroutine are:

(a) @ARB - The @ARB modifier indicates that delimiters specified in the time pattern should not be considered exact. Instead of only the delimiters indicated in the time pattern being valid, all other legal delimiters in the time/date are also valid.
(b) @MDATE={CURR|PAST|FUTURE|ZERO} - @MDATE=CURR indicates that the date component is partial and fills in the missing parts with the current date. @MDATE=PAST fills in missing parts such that the resulting date is the nearest date to the current date that is before the current date. @MDATE=FUTURE fills in the missing parts such that the resulting date is the nearest date to the current date that is later than the current date. @MDATE=ZERO indicates that the date component is partial and fills in the missing components with zeros. If none of these modifiers are specified, the @MDATE=FUTURE modifier is assumed.

The possible modifiers for the TIMEOUT subroutine are:

(a) @M=UC, @M=LC, @M=UCLC - @M=LC causes character months to be returned in lowercase. @M=UC causes character months to be returned in uppercase. @M=UCLC causes character months to be returned in lowercase except for the first character which is in uppercase. If none of these modifiers is present, @M=UCLC is defaulted.

(b) @W=UC, @W=LC, @W=UCLC - @W=LC causes the weekday name to be returned in lowercase. @W=UC causes the weekday name to be returned in uppercase. @W=UCLC causes the weekday name to be returned in lowercase except for the first character which is in uppercase. If none of these modifiers are specified, @W=UCLC is defaulted.

(c) @AM=UC or @AM=LC - @AM=LC causes returned meridian markers to be lowercase. @AM=UC causes the meridian markers to be returned in uppercase. If neither of these modifiers is present, @AM=LC is defaulted.

<u>General Time and Date Formats</u>

The TIMEGIN subroutine can recognize two forms of general time and date strings, the absolute time/date string and the relative time/date string.

The absolute time/date string is composed of three substrings - the time string, the date string, and the weekday string. An "arbitrary" general time/date string consists of one or more of the above three components, in any order, along with certain delimiter strings.

One or more of the following delimiters can occur between the substrings of a general time/date string: null, blank, ".", or ",". Note however that the null delimiter cannot be used if it would cause the juxtaposition of two numeric fields or two character fields when the first field is not fully specified. Also note that the "." can be used only after alphabetic fields.

The time string must be one of the following forms.

(a) HH:MM:SS.SSSSSS am|pm|a.m.|p.m. ZZZ
(for our American friends)

(b) HH:MM.SS.SSSSSS am|pm|a.m.|p.m. ZZZ
(for our Canadian friends)

(c) HH.MM.SS.SSSSSS am|pm|a.m.|p.m. ZZZ
(for our British friends)

In the above time strings, all character components are recognized in either upper-, lower-, or mixed case. The hour, minute, and second fields can have their leading zeros omitted. The order in which components of a time appear must be in the above order, however all fields need not be specified. If no meridian marker is used with the time, the time string will be interpreted as a 24-hour clock; otherwise, it will be interpreted as a 12-hour clock. If only an hour field is entered without a meridian and/or time-zone marker, the ":" must appear after hour. Normally, all fields not specified in the input will be set to zero in the exploded form.

TIMEGIN can process two types of date strings - numeric-date strings and character-date strings. Numeric-date strings must be in one the forms and normally in the order specified below:

    MM-DD-YYYY
    MM/DD/YYYY

Character-date strings must be in the following form (but not necessarily in the order specified):

    monthXDDXYYYY

where "X" stands for one or more of the following delimiters: null, blank, ".", or ",". Note that the null delimiter cannot be used after a numeric field that is partially specified when the following field is also numeric. Note also that the "." can only be used immediately after a month field. In either type of date string, MM and DD are numeric characters with optional leading zeros, the month is a character string consisting of at least three initial characters of a month name in upper-, lower-, or mixed case, and YYYY are numeric characters with optional leading zeros. Normally, if the YYYY portion of a date string is only two characters long, it is interpreted as specifying a year in the "current century". Under normal circumstances not all the components of a date string need be specified. If components of a date are missing, they are normally replaced in the exploded format by corresponding components of the "current date".

The weekday string must consist of at least the first two initial characters of a weekday name. The characters of a weekday name can be in upper-, lower-, or mixed case.

A relative time string can be specified by any of the following strings:

    NOW
    nn.nn YEAR[S]
    nn.nn MONTH[S]
    nn.nn WEEK[S]
    nn.nn DAY[S]
    nn.nn HOUR[S]
    nn.nn MINUTE[S]
    nn.nn SECOND[S]

The "nn.nn" in the above can be preceded by an optional "+" or "-" and the trailing blank can be omitted. This form of input creates a date in exploded format by adding or subtracting the appropriate quantity from the "current date".

Exploded Time and Date Formats

Exploded time and date formats are presented in a 12-fullword vector expressed as follows:

FW1 - contains a the characters 'GREG'.
FW2 - contains the Gregorian year as fullword integer.
FW3 - contains the month of the year as a fullword integer.
FW4 - contains the day of the month as a fullword integer.
FW5 - contains the hour of the day (24-hour clock) as a fullword integer.
FW6 - contains the minute as a fullword integer.
FW7 - contains the second as a fullword integer.
FW8 - contains the microsecond as a fullword integer.
FW9 - contains the weekday as a fullword integer.
  0 - No weekday is associated with this date.
  1...7 - Sunday, ..., Saturday.
FW10 - contains the time offset (in minutes) from GMT, or zero if no time zone was used.
FW11 and FW12 - contain the characters of the time-zone marker, left-justified and padded with blanks, or just blanks if no time-zone marker was specified.

If the time pattern specifies that only a date is being inputted, the time fields in the exploded format will be zeroed. If the time pattern specifies that only a time is being inputted, the date fields and the weekday field will be zeroed.

Examples

The following program, given both in 370-assember and FORTRAN, illustrates the used of the time routines. The 370-assembler version is as follows:

```
TIMETST CSECT
        REQU TYPE=DEC
        PRINT NOGEN
        ENTER R12,SA=SAVE
        DO
          CALL  READ,(TIME,TIMLEN,MOD,LNR,UNIT5)
          EXITDO R15,NZ
          CALL  TIMEGIN,(TIME,TIMLEN#,ZERO,TIMEOT,ZERO,LENGTH#),VL
          IF    R15,EQ,=F'8'
            CALL WRITE,(E_MSG,E_MSGL,MOD,LNR,UNIT6)
            REDO
          ELSEIF R15,NZ
            CALL WRITE,(W_MSG,W_MSGL,MOD,LNR,UNIT6)
          ENDIF
          L     R1,LENGTH#
          CVD   R1,NUM
          OI    NUM+7,X'0F'
          UNPK  L_MSG+L'L_MSG(3),NUM+6(2)
          CALL  WRITE,(L_MSG,L_MSGL,MOD,LNR,UNIT6)
          CALL  TIMEOUT,(TIMEOT,OUTFORM,TIME2,LENGTH#),VL
          CALL  WRITE,(TIME2,LENGTH,MOD,LNR,UNIT6)
        ENDDO
        EXIT 0
```

```
         LTORG
SAVE     DS   18F
UNIT6    DC   F'6'
UNIT5    DC   F'5'
LENGTH#  DC   F'0'
         ORG  LENGTH#+2
LENGTH   DS   H
MOD      DC   F'0'
LNR      DC   F'0'
NUM      DS   D
ZERO     DC   F'0'
TIMLEN#  DC   F'0'
         ORG  TIMLEN#+2
TIMLEN   DS   H
TIME     DS   CL80
TIME2    DS   CL80
TIMEOT   DS   12F
OUTFORM  DC   C'WWW DD/YY MMMB HH:MM:SS AM ZZZ*'
E_MSG    DC   C'0Input time is invalid.'
W_MSG    DC   C'0Time interpretation may be suspect.'
L_MSG    DC   C'0Number of characters in time used is '
         DC   CL4'   .'
E_MSGL   DC   AL2(L'E_MSG)
W_MSGL   DC   AL2(L'W_MSG)
L_MSGL   DC   AL2(L'L_MSG+4)
         END
```

The FORTRAN version is as follows:

```
       LOGICAL*1 TIME(80),TIME2(80),OUTFOR(80)
       DIMENSION TIMEOT(12)
       DATA OUTFOR/'WWW DD/YY MMMB HH:MM:SS AM ZZZ*'/,IZERO/0/
10     READ (5,1001,END=60) (TIME(I),I=1,80)
       CALL TIMGIN(TIME,I,IZERO,TIMEOT,IZERO,LEN,&40,&50)
       WRITE(6,1000) LEN
1000   FORMAT ('0Number of characters in time used is ',I2)
30     CALL TIMOUT(TIMEOT,OUTFOR,TIME2,LEN)
       WRITE (6,1003) (TIME2(N),N=1,LEN)
1001   FORMAT (80A1)
1003   FORMAT (' ',80A1)
       GO TO  10
40     WRITE(6,1004)
1004   FORMAT ('0Time interpretion may be suspect')
       GO TO  30
50     WRITE(6,1005)
1005   FORMAT ('0Input time is invalid.')
       GO TO  10
60     STOP
       END
```

If the following program input is processed by the program

```
dec 3
12/3/82
jan 8 16: pm
jan 9 3:pm pst
feb 21
jan 3 16: jiberish
wed feb 25
fri feb 25 3 pm pdt
```

the following program output will be generated.

```
Number of characters in time used is 80
Mon 03/84 Dec. 12:00:00 am

Number of characters in time used is 80
Fri 03/82 Dec. 12:00:00 am

Input time is invalid.

Number of characters in time used is 80
Mon 09/84 Jan. 03:00:00 pm PST

Number of characters in time used is 80
Tue 21/84 Feb. 12:00:00 am

Number of characters in time used is 10
Tue 03/84 Jan. 04:00:00 pm

Time interpretion may be suspect
Sat 25/84 Feb. 12:00:00 am

Time interpretion may be suspect
Sat 25/84 Feb. 03:00:00 pm PDT
```

TIMEIN

Purpose: To convert an MTS internal or a character-string time and date into an exploded format.

Location: Resident System

Calling Sequences:

Assembly: CALL TIMEIN,(format,tdinp,tdout,optns,errmsg),VL

FORTRAN: CALL TIMEIN(format,tdinp,tdout,optns,errmsg,
&rc4,&rc8)

Parameters:

format points to a character string describing the format of the time and date being used as input. This specifies whether tdinp contains a 4-byte or 8-byte internal time and date, or a character-string time and date that corresponds to time and date picture specification.

If tdinp is an internal time and date, format may be

(a) *MICROSECONDS* (or *MMS*)
(b) *MILLISECONDS* (or *MS*)
(c) *SECONDS* (or *S*)
(d) *MINUTES* (or *M*)
(e) *HOURS* (or *H*)
(f) *DAYS* (or *D*)
(g) *IBM MICROSECONDS* (or *IBMMMS*)

All the above forms and the modifiers that may be appended are described in the preface to this subroutine description. In addition, the following two formats may also be specified.

(h) *EXPLODED* (or *X*)

tdinp points to a 12-fullword exploded time vector.

(i) *NOW*

tdinp is not used; instead, the current time and date is used.

If tdinp is a character-string time and date, format must be a picture specification as described above in the preface.

tdinp points to the time and date to be converted into the exploded format.

tdout points to a 12-fullword vector that is to contain the exploded time. The format of this vector is described in the introduction above.

optns If the optns pointer is zero, points to a fullword zero, or is not present, the transformation as described above is carried out. Otherwise, optns points to a character string that is used to modify the standard transformation. The form of this string is displayed below.

    @mod1@mod2...*

where the possible modifiers are from the following list.

(a) @TZ=zzz - This modifier causes all time/dates that were entered without time-zone information to use the specified time zone "zzz" to fill in the last three fullwords in the resultant exploded format. If time-zone information was included in the input, the resultant exploded format will be transformed to the time zone specified by "zzz".

(b) @TZ=LOCAL - This works the same as above except that the current time zone will be used instead of a specified time zone. At U of M this will be either EST or EDT.

(c) @ROUND=val, @CEIL=val, or @TRUNC=val - @ROUND causes the resultant exploded format to be rounded to the specified unit, @TRUNC causes the resultant exploded format to be truncated to the specified unit, and @CEIL causes the resultant exploded format to be truncated to the specified unit and then incremented by one if the truncated unit is not zero. For any of the above modifiers, the unwanted fields are zeroed after the operation. The "val" portion of the modifier specifies the unit at which the action is to take place. Valid values for "val" are YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, and MILLISECOND. If none of these modifiers are present, the resulting exploded format will be expressed to the nearest microsecond.

errmsg — If errmsg is zero or omitted, no extended error information is returned. Otherwise, errmsg points to a 76-fullword vector. The first word of the vector contains the error code, the second word contains the length of the associated error message, and the remainder contains the error message padded with blanks.

&rc4,&rc8 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return codes:

0 Successful conversion.
4 Conversion completed but might not be what the caller had in mind (see errmsg above).
8 Conversion not performed (see errmsg above).

TIMEOUT

Purpose: To convert an exploded time and date into an MTS internal or a character-string format.

Location: Resident System

Calling Sequences:

Assembly: CALL TIMEOUT,(tdinp,format,tdout,len,errmsg),VL

FORTRAN: CALL TIMOUT(tdinp,format,tdout,len,errmsg,&rc4,
&rc8)

Parameters:

tdinp points to the 12-fullword vector that contains the exploded time and date to be converted.

format points to a character string describing the format of the time and date being produced as output. This specifies whether _tdout_ contains a 4-byte or 8-byte internal time and date, or a character-string time and date that corresponds to time and date picture specification.

If _tdout_ is an internal time and date, _format_ may be

(a) *MICROSECONDS* (or *MMS*)
(b) *MILLISECONDS* (or *MS*)
(c) *SECONDS* (or *S*)
(d) *MINUTES* (or *M*)
(e) *HOURS* (or *H*)
(f) *DAYS* (or *D*)

All the above forms and the modifiers that may be appended are described in the preface to this subroutine description. If _tdout_ is a character-string time and date, _format_ must be a picture specification as described above in the preface.

tdout points to the output region that will contain the converted time and date.

len points to a fullword that contains the length of the returned time and date. If _len_ is zero or omitted, no time and date will be returned.

errmsg If the _errmsg_ pointer is zero or omitted, no extended error information is returned. Otherwise, errmsg points to a 76-fullword vector. The first word of the vector contains the error code, the second word contains the length of the associated error message, and the remainder contains the error message padded with blanks.

&rc4,&rc8 (optional) are statement labels to transfer to if a nonzero return code occurs.

Return codes:

0 Successful conversion.
4 Conversion completed but might not be what the caller had in mind (see errmsg above).
8 Conversion not performed (see errmsg above).

TIMEGIN

Purpose: To convert a "general" time and date into an exploded format.

Location: Resident System

Alt. Entry: TIMGIN

```
Assembly: CALL TIMEGIN,(tdinp,tdlen,tdopt,tdout,optns,len,
                        tdcurr,status,errmsg),VL

FORTRAN:  CALL TIMGIN(tdinp,tdlen,tdopt,tdout,optns,len,
                      tdcurr,status,errmsg,&rc4,&rc8)
```

Parameters:

<u>tdinp</u> points to a general time and date string (see the introduction above for a description of this form of time and date).

<u>tdlen</u> points to a fullword containing the length of the string containing the general time and date. The subroutine will use as much of the string as necessary to create a time/date or until a element is reached that cannot be deciphered as a time/date element (see also the @ZCB and @DS modifiers below). The actual length used may be returned by the <u>len</u> parameter.

<u>tdopt</u> If this pointer is zero or points to a fullword zero, the conversion as outlined above is performed. Otherwise, <u>tdopt</u> points to a string that modifies the interpretation of the input string. The form of this string is

@mod1@mod2...*

where the "@mod1@mod2..." are from the following list.

(a) @TR1, @TR2, @TR3 - These modifiers specify the time range over which the subroutine is valid. The default modifier @TR1 specifies a time range from January 1 of the 32nd year of the "current century" to the end of the "current century". The @TR2 modifier specifies a time range from the beginning of the "current century" to the end of the "current century". This time range can lead to ambiguity when a date such as Jan 28 or 28 Jan is specified since it can be interpreted as January 28 of the "current year" or as January of the 28th year of the "current century". In this case, the date will be interpreted in the first sense and a return code of 4 will be issued. The @TR3 modifier specifies a time range from January 1 of the year 0 to December 31, 9999. If this time range option is specified, then all years must be expressed exactly. Note that TR is an abbreviation for TIME RANGE which may be used instead of TR (i.e., @TR1 is equivalent to @TIME RANGE1.)

(b) @ND=MDY, @ND=DMY, or @ND=YMD - The @ND=MDY modifier (the default) specifies that numeric-date strings are to be interpreted in the order month, day, year. The @ND=DMY modifier specifies that numeric-date strings are to be interpreted in the order day, month, year. The @ND=YMD modifier specifies that numeric-date strings are to be interpreted in the order year, month, day. Note that ND is an abbreviation for NUMERIC DATE FORM which may be be used instead of ND.

(c) @MT=ZERO, @MT=CURR, MT=HIGH - The @MT=ZERO modifier (the default) specifies that missing time components in the input time/date are to be set to zero in the output exploded format. The @MT=CURR modifier specifies that missing time components in the input time/date are to be set to the corresponding components of the "current time" in the output exploded format. The @MT=HIGH modifier specifies that missing time components in the input time/date are to be set to the highest value they can obtain in the output exploded format. Note that MT is an abbreviation for MISSING TIME DEFAULT which which may be used instead of MT.

(d) @MDATE=CURR, @MDATE=ZERO,
@MDATE=FIRST(CURR|PAST|FUTURE),
@MDATE=LAST(CURR|PAST|FUTURE),
@MDATE=PAST, or @MDATE=FUTURE

@MDATE=CURR (the default) specifies that missing date components are to be filled in with corresponding components of the "current date" or, if a weekday component appears without a date component in the input time/date, the date components of the resulting exploded format are to be set to the components of the date closest to the "current date" that falls on the specified weekday.

The @MDATE=ZERO modifier specifies that missing components of the date string are to be replaced by zeros in the resulting exploded format.

The @MDATE=FIRST(CURR) modifier specifies that if, on input, a month component appears but no day component is specified, the day field of the resulting exploded format will be set to the first of the month. If the year component is also missing, it will be set to the "current year". If no month or day components are specified on input, the resulting exploded format will contain the first day of the "current month". (If the year component is also missing, it will be set to the "current year".) In either of the above cases, if a weekday component is specified on input, the day field will be set to the first occurrence of the specified weekday in the appropriate month. In all other cases, the @MDATE=FIRST(CURR) modifier will act the same as the @MDATE=CURR modifier.

The @MDATE=FIRST(PAST) specifies that if, on input, a month component appears without a day component, the resulting exploded format will contain the first day of the specified month. If the year component is also missing on input, the year in the resulting exploded format will contain the "current year" if the resulting date would not be in the future; otherwise the year in the resulting exploded format will contain the year before the "current year". If the input time/date is missing both the month and day components, the resulting exploded format will contain the first day of the "current month". If the year component is also missing on input, the resulting exploded format will contain the "current year". In either of the above cases, if a weekday component is also specified on input, the day field in the resulting exploded format will be set to the first occurrence of the specified weekday in the appropriate month. In all other cases, the @MDATE=FIRST(PAST) will act like the @MDATE=PAST modifier.

The @MDATE=FIRST(FUTURE) specifies that if, on input, a month component appears without a day component, the resulting exploded format will contain the first day of the specified month. If the year component is also missing on input, the year in the resulting exploded format will contain the "current year" if the resulting date would not be in the past; otherwise the year in the resulting exploded format will contain the year after the "current year". If the input time/date is missing both the month and day components, the resulting exploded format will contain the first day of the "current month". If the year component is also missing on input, the resulting exploded format will contain the "current year". In either of the above cases, if a weekday component is also specified on input, the day field in the resulting exploded format will be set to the first occurrence of the specified weekday in the appropriate month. In all other cases, the @MDATE=FIRST(FUTURE) modifier acts like the @MDATE=FUTURE modifier.

The MDATE=LAST(CURR) modifier specifies that if, on input, a month component appears but no day component is specified, the day field of the resulting exploded format will be set to the last of the month. If the year component is also missing, it will be set to the "current year". If no month or day components are specified on input, the resulting exploded format will contain the last day of the "current month". If the year component is also missing, it will be set to the "current year". In either of the above cases, if a weekday component is specified on input, the day field will be set to the last occurrence of the specified weekday of the appropriate month. In all other cases, the @MDATE=LAST(CURR) modifier will act the same as the @MDATE=CURR modifier.

The @MDATE=LAST(PAST) specifies that if, on input, a month component appears without a day component, the resulting exploded format will contain the last day of the specified month. If the year component is also missing on input, the year in the resulting exploded format will contain the "current year" if the resulting date would not be in the future; otherwise the year in the resulting exploded format will contain the year before the "current year". If the input time/date is missing both the month and day components, the resulting exploded format will contain the last day of the "current month". If the year component is also missing on input, the resulting exploded format will contain the "current year". In either of the above cases if a weekday component is also specified on input, the day field in the resulting exploded format will be set to the last occurrence of the specified weekday in the appropriate month. In all other cases, the @MDATE=LAST(PAST) will act like the @MDATE=PAST modifier.

The @MDATE=LAST(FUTURE) specifies that if, on input, a month component appears without a day component, the resulting exploded format will contain the last day of the specified month. If the year component is also missing on input, the year in the resulting exploded format will contain the "current year" if the resulting date would not be in the past; otherwise the year in the resulting exploded format will contain the year after the "current year". If the input time/date is missing both the month and day components, the resulting exploded format will contain the last day of the "current month". If the year component is also missing on input, the resulting exploded format will contain the "current year". In either of the above cases, if a weekday component is also specified on input, the day field in the resulting exploded format will be set to the last occurrence of the specified weekday in the appropriate month. In all other cases, the @MDATE=LAST(FUTURE) modifier acts the same as the @MDATE=FUTURE modifier.

The @MDATE=PAST modifier specifies that if, on input, components of the date are missing, the missing components will be replaced in the resulting exploded format by components of a generated date that is the closest possible date to the "current date" that can be constructed from the missing components that is less that the "current date". In the case that a weekday sting is specified in the input time/date string and the date string is missing, the date fields in the resulting date will be set to the date of specified weekday before the "current date".

The @MDATE=FUTURE modifier specifies that if, on input, components of the date are missing, the missing components will be replaced in the resulting exploded format by components of a generated date that is the closest possible date to the "current date" that can be constructed from the missing components of a time/date that is greater that the "current date". In the case that a weekday sting is specified in the input time/date string and the date string is missing, the date fields in the resulting date will be set to the date of specified weekday after the "current date".

Note that MDATE is an abbreviation for MISSING DATE DEFAULT which may be used instead of MDATE.

(e) @LI=AR, @LI=A, or @LI=R - The @LI=AR modifier (the default) specifies that absolute time/dates and relative time/dates are legal input to this subroutine. The @LI=A modifier specifies that only absolute time/dates are legal input. The @LI=R modifier specifies that only relative time/dates are legal input. Note that LI is an abbreviation for LEGAL INPUT which may be used instead of LI.

(f) @ZCB=YES or @ZCB=NO - These modifiers specify whether the subroutine is to handle zero-level commas and blanks. @ZCB=YES (the default) allows zero-level commas and blanks while @ZCB=NO does not. If ZCB=NO is specified, the first zero-level comma or blank will terminate the input string. Note that ZCB is an abbreviation for ZERO LEVEL COMMAS AND BLANKS which may be used instead of ZCB.

(g) @DS=NO or @DS=YES - These modifiers specify whether the subroutine should accept input time/date strings delimited by "...", '...', or (...). The default is DS=NO. If DS=YES is specified, the input string may or may not be so delimited; if it is not delimited, then terminating of input on commas and blanks depends on the setting of the ZCB modifier. If DS=YES is specified and the string is delimited, input is terminated by the trailing delimiter (since any internal commas and blanks are not zero-level). Note that DS is an abbreviation for DELIMITED STRING which may be used instead of DS.

<u>tdout</u> points to a 12-fullword area in which the resulting exploded form of the time/date is to be placed.

<u>optns</u> If this parameter is zero, <u>optns</u> points to a fullword zero. If this parameter is omitted, no modifications are made to the exploded time/date. Otherwise this parameter points to a string specifying how the resulting exploded format is to be modified. The form of this string is as follows:

@mod1@mod2...*

The valid modifiers are as follows.

(a) @TZ=ZZZ - The presence of this modifier causes all time/dates that were entered without time-zone information to use the specified time zone (ZZZ) to fill in the last three fullwords in the resultant exploded format. If time-zone information was included in the input, the resultant exploded format will be transformed to the time zone specified by ZZZ.

(b) @TZ=LOCAL - This modifier the same as above except that the current time zone will be used instead of a specified time zone. At U of M, this will be either EST or EDT.

(c) @ROUND=val, @CEIL=val, or @TRUNC=val - The @ROUND modifier causes the resultant exploded format to be rounded to the specified unit. The @TRUNC modifier causes the resultant exploded format to be truncated to the specified unit. The @CEIL modifier causes the resultant exploded format to be truncated to the specified unit and then incremented by one if the truncated unit is not zero. For any of the above modifiers, the unwanted fields are zeroed after the operation. The "val" portion of the modifier specifies at which unit the action is to take place. Valid values for "val" are YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, or MILLISECOND.

<u>len</u> If this parameter is zero or is omitted, no length is returned. Otherwise this parameter points to a fullword that will return the actual length of general time/date extracted from the input string, that is, the actual length of the string used to create the resulting exploded form.

<u>tdcurr</u> If this parameter is zero, points to a fullword zero, or is omitted, the "current date", "current time", "current year", etc. will be determined by actual time of call. Otherwise this parameter points to a 9-fullword vector containing a time and date in exploded format to be used as the "current time", "current date", "current year", etc.

<u>status</u> If this parameter is zero or is omitted, no status information is returned. Otherwise this parameter points to a fullword containing a series of switches indicating the status of the conversion, if successful. The possible switches that can be set are as follows:

F'1' - Input string was a relative time
F'2' - Input string was a absolute time
F'4' - Year was defaulted
F'8' - Month was defaulted
F'16' - Day was defaulted
F'32' - Hour was defaulted
F'64' - Minute was defaulted
F'128' - Second was defaulted
F'256' - Microsecond was defaulted
F'512' - Weekday was defaulted

errmsg If this parameter is zero or is omitted, no extended error information is returned. Otherwise this points to a 76-fullword vector. The first word of the vector contains the error code, the second word contains the length of the associated error message, and the rest of vector contains the error message padded with blanks.

_&rc4,&rc8_ (optional) are statement labels to transfer to if a nonzero return code occurs.

Return codes:

0 Successful conversion in all probability.
4 Conversion completed but might not be what the caller had in mind (see _errmsg_).
8 Conversion not performed (see _errmsg_).

TIMNTRP

Subroutine Description

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

Alt. Entries: TIMTRP, TIMNTRPS, TIMTPS

Calling Sequences:

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

          CALL TIMNTRPS,(exit,region),VL

FORTRAN:  CALL TIMTPS(exit,region,&rc4)
```

Parameters:

- _exit_ (GR0) should be zero or the location of the exit routine transfer control to when a timer interrupt occurs.
- _region_ (GR1) should should contain the location of a 72-byte save region for storing pertinent information.
- _&rc4_ (optional) is the statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return.
4 Illegal parameter or no VL bit specified.

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

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

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

When a timer interrupt exit is taken, the exit is disabled, so that further timer interrupts which specify this exit region will remain pending while the current one is being processed. The exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save area provided). At the time of this call, GR1 will point to the exit region, whose contents will be

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

The contents of GR0 and GR2 to GR12 are unpredictable.

If the exit routine returns to MTS (BR 14), the user's program will be restarted at the point of the interrupt. The PSW stored in the savearea is always in BC mode (bit 12 is zero). The exit will be reenabled if the return code in GR15 is zero; otherwise, the exit will remain disabled until another call on TIMNTRP. The registers must be restored in the standard fashion when the exit routine returns.

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

A call on the TIMNTRPS or TIMTPS subroutines takes the S-type parameters and loads them into an R-type call on the TIMNTRP subroutine.

Example: Assembly:

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

```
              .
             LM   14,12,12(13)
             SR   15,15
             BR   14
       REG   DS   19F
```

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

TOUCH

Subroutine Description

Purpose: To update the last data-change time for a file.

Location: Resident System

Calling Sequences:

Assembly: CALL TOUCH,(what,info,ercode,errmsg),VL

FORTRAN: CALL TOUCH(what,info,ercode,errmsg,&rc4)

Parameters:

<u>what</u> is the location of either:
(a) a file name with trailing blank (if <u>info</u>=0),
(b) a fullword-integer FDUB-pointer (such as returned by GETFD) (if <u>info</u>=1),
(c) a fullword-integer logical I/O unit number (0 through 99) (if <u>info</u>=1), or
(d) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if info=1).

<u>info</u> is the location of a fullword integer which specifies the kind of <u>what</u> parameter supplied.

<u>ercode</u> (optional) is the location of a fullword in which the TOUCH subroutine will place an error number if an error return (return code 4) is made. If this parameter is omitted, then the <u>errmsg</u> parameter must also be omitted.

Assembly language users who wish to omit this parameter should either follow the variable parameter list convention (high-order bit of the previous parameter's adcon in the parameter list should be 1) or else supply an adcon which is zero (rather than pointing to a zero).

Error numbers less than 100 indicate something was wrong with either the mechanics of the subroutine call or the values of the parameters:

Number	Message
1	Illegal parameter list pointer
2	Illegal "what" parameter address
5	Illegal "info" parameter address
6	"Info" parameter value not 0 or 1

Error numbers between 100 and 105 describe errors that occur in accessing the file.

101	Illegal file name
102	File not found - file "xxxx"
103	Access not allowed to file "xxxx" (Write access required to update the last data-chage time).
104	Deadlock situation, try later - file "xxxx"
105	Interrupted out of wait for locked file "xxxx"

Error numbers 201 and above indicate a file system error.

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from TOUCH with an error code of 105.

errmsg (optional) is the location of a 20-fullword (80-character) region in which the TOUCH subroutine will place the corresponding error message if an error return (return code 4) is made. Assembly language users should see instructions above on omitting optional parameters for the ercode parameter.

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

Return Codes:

0 The last data-change time has been set to the current time.

4 Error. The last data-change time has not been set. See the ercode and errmsg values returned for the specific error.

Examples: Assembly:

```
              CALL TOUCH,(WHAT,INFO,ERCODE,ERRMSG)
                .
                .
       WHAT   DC    C'PROGRAM '
       INFO   DC    F'0'
       ERCODE DS    F
       ERRMSG DS    CL80
```

FORTRAN:

```
              CALL TOUCH('PROGRAM ',0)
```

The above examples set the last data-change time for the file PROGRAM to the current time.

Translation Routines

Subroutine Description

Purpose: To allow convenient access to the standard MTS translation tables from a FORTRAN program.

Location: Resident System

Calling Sequences:

```
FORTRAN:  CALL TASEB(buffer,length)
          CALL TEBAS(buffer,length)
          CALL TLCUC(buffer,length)
          CALL TUCLC(buffer,length)
          CALL TIASEB(buffer,length)
          CALL TIEBAS(buffer,length)
```

Parameters:

<u>buffer</u> is the location of the characters to be translated.

<u>length</u> is the location of the number of characters to be translated. This should be declared INTEGER*4.

Description: The translation subroutines translate a buffer of characters of a given length. The translation is performed in place.

The correspondence of entry points to the MTS translation tables is as follows:

Entry Pt.	MTS Table
TASEB	ASCEBC
TEBAS	EBCASC
TLCUC	TRLCUC
TUCLC	TRUCLC
TIASEB	IASCEBC (TRIAE)
TIEBAS	IEBCASC (TRIEA)

See the descriptions of the MTS translation tables in this volume for the complete details of each table.

Example:

```
FORTRAN:  LOGICAL*1 INBUFF(256),OUTBUF(256)
             ...
          CALL TASEB(INBUFF,LENGTH)
          CALL TLCUC(INBUFF,LENGTH)
             ...
          CALL TEBAS(OUTBUF,256)
```

In the above example, a 256-character buffer of ASCII characters in translated on input to EBCDIC and then to uppercase. On output, the buffer is translated back to ASCII characters.

TRLCUC, TRUCLC

Translate Table Description

Contents: Translate tables to convert lowercase letters into uppercase letters, or uppercase letters into lowercase letters.

Location: Resident System

Alt. Entry: CASECONV is an alternate entry for TRLCUC.

Calling Sequences:

Assembly:
```
L    r,=V(TRLCUC)
TR   name,0(r)

L    r,=V(TRUCLC)
TR   name,0(r)
```

Parameters:

_r_ is a general register that will contain the address of the translate table.

name is the location of the region to be translated.

Description: The TRLCUC table translates lowercase letters (a-z) to uppercase letters (A-Z). The TRUCLC table translates uppercase letters to lowercase letters. Both tables leave nonalphabetic characters unchanged.

Example: Assembly:
```
      L    6,=V(TRLCUC)
      TR   REG(100),0(6)
       .
       .
REG   DS   CL100
```

FORTRAN:
```
LOGICAL*1 REG(100),TRTAB(256)
COMMON /TRLCUC/TRTAB
 ...
CALL ITR(100,REG,0,TRTAB,0)
```

The above examples will convert the lowercase letters of the 100-byte region at location REG into uppercase letters.

The FORTRAN example uses the ITR subroutine (see the description of the Logical Operators subroutines in this volume). In addition, a RIP loader record (RIP TRLCUC) must be inserted into the FORTRAN object file to force the loader to resolve the symbol TRLCUC from the low-core symbol table.

TRTLC, TRTUC, TRTNONAN

Translate Table Description

Purpose: 256-byte translate tables that may be used to detect the presence of lowercase letters, uppercase letters, or nonalphanumeric characters.

Location: Resident System

Calling Sequence:

Assembly:
```
SR   2,2
L    r,=V(TRTLC)
TRT  char,0(r)

SR   2,2
L    r,=V(TRTUC)
TRT  char,0(r)

SR   2,2
L    r,=V(TRTNONAN)
TRT  char,0(r)
```

Parameters:

<u>r</u> is a general register containing the address of the desired translate table.

<u>char</u> is the location of the character string to be tested.

Values Returned:

GR1 will contain the address of the detected lowercase letter (for TRTLC), the detected uppercase letter (for TRTUC), or the detected nonalphanumeric character (for TRTNONAN). If no corresponding letter or character is detected, GR1 will be unchanged.

GR2 will contain the detected lowercase or uppercase letter, or will be unchanged if none is detected.

The condition code is set to zero if the character string contains no lowercase letters (for TRTLC), uppercase letters (for TRTUC), or nonalphanumeric characters (for TRTNONAN).

Description: The TRTLC table may be used to detect the presence of lowercase letters (a-z) in a character string. The TRTUC table may be used to detect the presence of uppercase letters (A-Z) in a character string. The TRTNONAN table may be used to detect the presence of nonalphanumeric characters (not a-z, A-Z, 0-9, or _) in a character string.

Example: Assembly:

```
                SR   2,2
                L    3,=V(TRTLC)
                TRT  NAME,0(3)
                BZ   EXIT           No lowercase letters
                STC  GR2,LTR        Save detected letter
                 .
                 .
        NAME    DS   CL16           Character string
        LTR     DS   C              Detected letter
```

FORTRAN:

```
              LOGICAL*1 NAME(16),TRTAB(256)
              COMMON /TRTLC/TRTAB
               ...
              I = ITRT(16,NAME,0,TRTAB,0,N,LTR)
              IF (I.EQ.0) GO TO 10
        C     LTR contains the detected letter
        C     N contains the displacement of detected
        C     letter
               ...
        10    No lowercase letters
```

The above examples test for the presence of a lowercase letter in the 16-byte character string contained in NAME.

The FORTRAN example uses the ITRT subroutine (see the description of the Logical Operators subroutines in this volume). In addition, a RIP loader record (RIP TRTLC) must be inserted into the FORTRAN object file to force the loader to resolve the symbol TRTLC from the low-core symbol table.

TRUNC

Subroutine Description

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

Location: Resident System

Calling Sequence:

Assembly: CALL TRUNC,(unit)

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

Parameters:

<u>unit</u> is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

<u>rc4,...,rc20</u> (optional) are statement labels to transfer to if a nonzero return code occurs.

Return Codes:

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

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

If a wait to lock is interrupted by an attention interrupt, control passes to MTS unless the user program has established an attention interrupt exit (by calling the ATTNTRP subroutine). Following a $RESTART command or a return to the point of interruption from the attention exit, a return is made from TRUNC with a return code of 20.

Examples: Assembly:

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

FORTRAN:

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

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

TWAIT

Subroutine Description

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

Location: Resident System

Calling Sequences:

Assembly: CALL TWAIT,(code,value)

FORTRAN: CALL TWAIT(code,value,&rc4)

Parameters:

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

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

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

_rc4_ (optional) is a statement label to transfer to if a nonzero return code occurs.

Return Codes:

0 Successful return
4 Invalid _code_ parameter

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

Example: FORTRAN:

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

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

UNLK

Subroutine Description

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

Location: Resident System

Alt. Entry: UNLCK

Calling Sequence:

Assembly: CALL UNLK,(unit)

FORTRAN: CALL UNLK(unit,&rc4)

Parameters:

<u>unit</u> is the location of either
(a) a fullword-integer FDUB-pointer (as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS) used to lock the file (either explicitly in a call to LOCK or implicitly in a call to WRITE, for example).

<u>rc4</u> is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

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

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

Description: See Appendix D of the section "Files and Devices" in MTS Volume 1, <u>The Michigan Terminal System</u>, for details concerning concurrent use of shared files.

Examples: Assembly:

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

FORTRAN:

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

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

UNLOAD, UNLDF

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

FORTRAN: CALL UNLDF(name,sinbr,sws,&rc4,&rc8)

index = UNLDF(name,sinbr,sws,&rc4,&rc8)

Parameters:

<u>name</u> is either the location of the "name" (specified by <u>sws</u>) or zero.

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

<u>sws</u> is the location of a fullword switch:

- 0 <u>name</u> is the FDname from which the material was LOADed.
- 1 <u>name</u> is an 8-character, left-justified, external symbol.
- 2 <u>name</u> is a fullword virtual memory location (the SYMTAB option must be ON).
- 128 same as <u>sws</u>=0, except that on return <u>index</u> contains the storage index number of the storage that is released.
- 129 same as <u>sws</u>=1, except that on return <u>index</u> contains the storage index number of the storage that is released.
- 130 same as <u>sws</u>=2, except that on return <u>index</u> contains the storage index number of the storage that is released.

<u>index</u> (GR0) contains the storage index number of the storage that is released if <u>sws</u> is 128, 129, or 130.

<u>rc4,rc8</u> are statement labels to transfer to if a nonzero return code is encountered.

Return Codes:

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

Description: Each time the LOAD subroutine is called, a new storage index number is assigned for use with storage acquired in order to load the material in the file specified for that LOAD call. In order to unload the material, either the storage index number or the name of the file LOADed from may be given. In addition, if the global switch SYMTAB is ON, the name of an external symbol or a virtual memory location in the material loaded may be specified. In any case, _all_ of the material loaded on that call on LOAD is unloaded. See the "Virtual Memory Management" section in MTS Volume 5, _System Services_, for a further description of using storage index numbers with the LOAD and UNLOAD subroutines.

Examples:

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

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

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

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

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

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

URAND

Subroutine Description

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

Location: *LIBRARY

Calling Sequences:

Assembly: CALL URAND,(value)

FORTRAN: x = URAND(value)

Parameters:

value is the location of a fullword integer used for generating the random number.

Values Returned:

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

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

$$\underline{value}=(65539*\underline{value})(\text{mod } 2^{31}-1).$$

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

On each successive call to URAND, *value* is updated according to the expression given above. The program calling URAND should provide an odd integer value for *value* when URAND is called for the first time; subsequent calls to URAND will automatically use the latest updated value.

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

As a special case, the initial value of *value* may be zero. In this case, the next integer random number will be supplied by URAND and will depend upon the time of day.

The new integer random number that is generated will be stored in value. Thus, X = URAND(0) is not permissible in FORTRAN; a variable containing zero must be used instead.

Examples: Assembly:

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

FORTRAN:

```
       INTEG=999
       X=URAND(INTEG)
```

In both examples above, URAND is called with the initial value of 999. INTEG should not be modified between calls to URAND unless a new random-number sequence is to be initiated.

WRITE

Subroutine Description

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

Location: Resident System

Alt. Entry: MTSWRITE, WRITE#

Calling Sequences:

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

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

Parameters:

- reg is the location of the virtual memory region from which data is to be transmitted.
- len is the location of a halfword (INTEGER*2) integer giving the number of bytes to be transmitted.
- mod is the location of a fullword of modifier bits used to control the action of the subroutine. If mod is zero, no modifier bits are specified. See the "I/O Modifiers" description in this volume.
- lnum is the location of a fullword integer giving the internal representation of the line number that is to be written or has been written by the subroutine. The internal form of the line number is the external form times 1000, e.g., the internal form of line 1 is 1000, and the internal form of line .001 is 1.
- unit is the location of either
  - (a) a fullword-integer FDUB-pointer (such as returned by GETFD),
  - (b) a fullword-integer logical I/O unit number (0 through 99), or
  - (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- rc4,... is the statement label to transfer to if the corresponding nonzero return code is encountered.

Return Codes:

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

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

If _len_ is zero when writing to a line file , the line is deleted from the file.

There are no default FDnames for WRITE.

There is a macro WRITE in the system macro library for generating the calling sequence to this subroutine. See the macro description for WRITE in MTS Volume 14, _360/370 Assemblers in MTS_.

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

Assembly:

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

                or

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

FORTRAN:

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

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

Assembly:

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

FORTRAN:

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

WRITEBUF

Subroutine Description

Purpose: To write out all changed disk file buffers.

Location: Resident System

Alt. Entry: WRITBF

Calling Sequences:

Assembly: CALL WRITEBUF,(unit)

FORTRAN: CALL WRITBF(unit,&rc4)

Parameters:

unit is the location of either
(a) a fullword-integer FDUB-pointer (such as returned by GETFD),
(b) a fullword-integer logical I/O unit number (0 through 99), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

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

Return Codes:

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

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

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

Examples: Assembly:

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

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

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

<u>XCTL, XCTLF</u>

Subroutine Description

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

Location: Resident System

Calling Sequences:

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

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

Parameters:

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 8 in <u>info</u> must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an input subroutine to be called during loading via a READ subroutine calling sequence to read loader records (i.e., the input subroutine is called with a parameter list identical to the system subroutine READ). In this case, bit 9 in <u>info</u> must be 1.

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

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

bit 0: 1, if <u>errexit</u> parameter is specified.
bit 1: 1, if <u>output</u> is specified.

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

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

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

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

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

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

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

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

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

(1) an FDname terminated by a blank.
(2) a FDUB-pointer (as returned by GETFD).
(3) an 8-character logical I/O unit name, left-justified with trailing blanks. In this case, bit 10 of info must be 1.
(4) a fullword-integer logical I/O unit number (0-99).
(5) the address of an output subroutine to be called during loading via the SPRINT subroutine calling sequence to write loader output (i.e., the output subroutine is called with a parameter list identical to the system subroutine SPRINT). In this case, bit 11 of info must be 1.

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

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

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

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

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

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

Values Returned:

None.

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

(1) XCTL makes a copy of all its parameter values and releases all storage associated with the current link level.
(2) The loader is called to dynamically load the specified program using _input_, _info_, _output_, _lsw_, _gtsp_, _frsp_, and _pnt_ if specified.
(3) The dynamically loaded program is called with the address of _parlist_ in GR1.
(4) If the dynamically loaded program returns to XCTL, it is unloaded.
(5) XCTL returns to the program which initiated the current link level, preserving the return registers of the dynamically executed program.

Note that XCTL accepts a variable-length parameter list of three to eight arguments. For most applications, only the first three are required. These parameters passed to XCTL may be part of the current link level to be released, since XCTL makes copies of them. However, the parameter list and parameters passed to the program XCTLed to, as well as the optional subroutines specified by <u>input</u>, <u>output</u>, <u>errexit</u>, <u>gtsp</u>, <u>frsp</u>, and <u>pnt</u> may <u>not</u> be part of the current link level since it is released before the program transferred to, is loaded and executed.

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

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

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

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

Example: Assembly:

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

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

Xerox 9700 Font Routines

Subroutine Description

Purpose: To access the Xerox 9700 font information tables.

Location: Resident System

Entry Points: The Xerox 9700 font routines have the following entry points: FNTINF, FNTSCN, FNTWID, FNTBLK.

Description: These subroutines allow user programs to obtain information about Xerox 9700 page printer fonts. This information is used mainly by text processors, but also may be of use to other programs. The most common uses of these subroutines are by text-processors for obtaining the widths of characters in the font, and by user programs for determining whether a given 6-character name is a valid Xerox 9700 font name.

The FNTINF subroutine returns information about a particular font. The information includes the name of the typeface, the style of the font (roman, bold, italic, etc.), which character positions actually contain characters, the orientation of the font (portrait or landscape), the name of the corresponding font(s) in the other orientation, and several other items.

The FNTSCN subroutine returns the names of fonts satisfying certain criteria such as all 10-point fonts in portrait orientation.

The FNTWID subroutine returns the table of character widths for a proportionally spaced font. Since each character in such a font may have a different width, the table must be used by the text processor to determine how much text will fit on a line.

The FNTBLK subroutine returns a list of "blanks" in a font. A proportionally spaced font contains blanks of several different widths which are used for positioning text within a line.

FNTINF

Purpose: To find information about a specific font.

Calling Sequence:

Assembly: CALL FNTINF,(name,n,array)

FORTRAN: CALL FNTINF(name,n,array,&rc4)

or

```
INTEGER*4 FNTINF,rc
rc = FNTINF(name,n,array)
```

Parameters:

- name — is a six-character font name (left-justified with trailing blanks, if shorter than six characters).
- n — is the number of words in array.
- array — is an integer-valued array whose elements will be set to the information returned. Only the first n of these will be set. The information returned is described at the end of this description.
- rc — is the fullword-integer value returned indicating the result of the subroutine call (see "Return Codes" below). This value is returned both in GR0 and GR15 (i.e., both as a function value and as a return code).

name and n should be set by the user before the call; the first n words of array to values as described below (at end after all calling sequences).

Return Codes:

0 Information is successfully returned.
4 Font does not exist.

FNTSCN

Purpose: To scan the font table for the names of fonts that satisfy specified criteria.

Calling Sequence:

Assembly: CALL FNTSCN,(ctl,name,n,array)

FORTRAN: CALL FNTSCN(ctl,name,n,array,&rc4,&rc8)

or

```
INTEGER*4 FNTSCN,rc
rc = FNTSCN(ctl,name,n,array)
```

Parameters:

- name — is a six-character font name (left-justified with trailing blanks, if shorter than six characters).
- n — is the number of words in array.
- array — is an integer-valued array whose elements will be set to the information returned. Only the first n of these will be set. The information returned is described at the end of this description.
- ctl — should be set to zero for the first call of a given scan and untouched on other calls.
- rc — is the fullword-integer value returned indicating the result of the subroutine call (see "Return Codes" below). This value is returned both in GR0 and GR15 (i.e., both as a function value and as a return code).

The fields of array (nb: currently only the first 13 fields are looked at; this will be changed later to handle all fields) should be set to a value to be matched or to -1 for "don't care" before a call is made. When the subroutine returns, all values will be changed to the values for the next font found (as if FNTINF had been called), and name will be set to the name of the font. Before the call to get the next font in the current scan, the caller must set all the fields to -1 or value being looked for again.

Return Codes:

0 A font was found.
4 No (more) fonts satisfying requirements.
8 ctl was changed by user to an illegal value.

Example: Thus, calling FNTSCN with _array_(1) set to 1, _array_(3) set to 10, _array_(13) set to 0, and all the other fields set to -1 will cause it to return in succession all the portrait fonts that are 10 point, fixed-pitch.

FNTWID

Purpose: To get width tables for a specific font.

Calling Sequence:

Assembly: CALL FNTWID,(name,type,region)

FORTRAN: CALL FNTWID(name,type,region,&rc4,&rc8,&rc12)

or

```
INTEGER*4 FNTWID,rc
rc = FNTWID(name,type,region)
```

Parameters:

name is a six-character font name (left-justified with trailing blanks, if shorter than six characters).

type should be set as follows:
- 0 - table returned in _region_ is 256 bytes
- 1 - table returned in _region_ is 256 halfwords
- 2 - table returned in _region_ is 256 fullwords

region is the location of a region where the width table is returned.

_rc_ is the fullword-integer value returned indicating the result of the subroutine call (see "Return Codes" below). This value is returned both in GR0 and GR15 (i.e., both as a function value and as a return code).

Return Codes:

0 Width table returned successfully.
4 Font name not found.
12 Unable to return table because _type_=0 and at least one character of font has width > 255.

Description: Not all fonts that exist (_rc_=0 from FNTINF) will have width tables (_rc_=0 from FNTWID). The ones that do not have width tables are fixed-pitch fonts and the width of all characters in those fonts is returned in _array_(5) by FNTINF. There are, however, some fixed-pitch fonts that do have width tables. These are fonts for which all the printing characters have the same width, but which also have several blanks of varying widths.

FNTBLK

Purpose: To get list of blank characters for a specific font.

Calling Sequence:

Assembly: CALL FNTBLK,(name,nbr,region)

FORTRAN: CALL FNTBLK(name,nbr,region,&rc4)

or

```
INTEGER*4 FNTBLK,rc
rc = FNTBLK(name,nbr,region)
```

Parameters:

name is a six-character font name (left-justified with trailing blanks, if shorter than six characters).

nbr is the location of a fullword integer which the caller sets before the call to indicate the number of bytes available in _region_. This routine will set _nbr_ to the number of blank characters actually returned in _region_.

region is the location of the region where the blank characters are returned. These are put in _region_ one character per byte. To find out how wide each of these blanks is, you will have to use these characters as subscripts into the width table returned by FNTWID.

_rc_ is the fullword-integer value returned indicating the result of the subroutine call (see "Return Codes" below). This value is returned both in GR0 and GR15 (i.e., both as a function value and as a return code).

Return Codes:

0 Blank information returned in _region_.
4 Font name not found.

Description: Fonts that exist (_rc_=0 from FNTINF) but have no width table (_rc_=4 from FNTWID) will also return _rc_=4 from FNTBLK. These fonts are usually fixed-pitch fonts that have one blank the same width as all the other characters (returned as _array_(5) from FNTINF) at position x'40' for a normal font and x'20' for an ASCII font.

Information in the <u>array</u> array:

Item	Subscript	Description
Font Orientation	1	See below
Point Size	2	
Font Size	3	Number of bits of font memory needed
Linespacing	4	In dots (300 dots per inch)
Charspacing	5	In dots
Cell Height	6	In dots
Baseline	7	Distance from cell bottom in dots
Leading	8	In dots
Typestyle Code	9	See below
Typemod Code	10	See below
Typecharset Code	11	See below
Font Access	12	See below
Font Kind	13	See below
Raster Bitmap	14-21	See below
Location	22	See below
High Code	23	Highest ASCII value
Landscape Name	24-25	Left-justified with trailing blanks
Portrait Name	26-27	Left-justified with trailing blanks
Inverse Landscape Name	28-29	Left-justified with trailing blanks
Inverse Portrait Name	30-31	Left-justified with trailing blanks
Font code set	32	0=EBCDIC, 1=ASCII

For the four name fields in 24-31, if a given field is all blank, then either the font in that rotation is not on any machine or else the CCID making the call has no access to it.

Font Orientation:
Landscape is 0
Portrait is 1
Inverse landscape is 2
Inverse portrait is 3
Landscape or inverse landscape is 100
Portrait or inverse portrait is 101

The last two may be used on calls to FNTSCN; they will never be returned by FNTINF or FNTSCN.

Font Access:
Anyone is 0
Staff is 1
Pageid is 2 (restricted to ccid PAGE)
File is 3 (who can access depends on access to a file)
Not On 9700 is 4
Deprecated is 5
Anticipated is 6

Deprecated is the same as Anyone but the font is not documented. Anticipated is the same as Staff, but it is documented as if it was Anyone.

Font Kind:
Fixed is 0
Proportional is 1

The raster bitmap is a sequence of 256 bits. Each bit is 1 if the corresponding code position in the font has a printing character (the leftmost bit of the word at subscript 14 corresponds to X'00'; the rightmost bit of the word at subscript 21 corresponds to X'FF'). One should not assume that the characters of Xerox 9700 fonts are located in any standard position, e.g., they do not necessarily correspond to the locations used for the EBCDIC collating sequence.

Location:
CNTR is 1 ┐
NUBS is 2 │ additive
UNYN is 4 ┘

The Typestyle, Typemod, and Typecharset fields are designed to do a simple classification that is sufficient for structuring the documentation of fonts for casual users. Although existing values will probably not be changed, others will certainly be added.

Typestyle is a grouping in which some of the entries are actually typefaces and some are just a collection of things. Typemod includes various modifiers, none or more than one of which may be applied. Typecharset is some additional words on the characters in those fonts (see CCMemo 803 for some more explanation).

The current meanings for values in those fields are:

Typestyle Code:

Unclassified	0	
Xerox 1200	1	
APL	2	
Serif	3	
Scientific 10	4	
Letter Gothic	5	
Prestige Elite	6	(7-8 skipped)
Univers	9	
Press Roman	10	(11-12 skipped)
Helvetica	13	
Century Schoolbook	14	
Script 10	15	
Form Font	16	
Bar Codes	17	
Plot Fonts	18	
Shading Font	19	
Spacing Font	20	
Computer Modern Roman	21	
Computer Modern Typewriter	22	
Computer Modern Sanserif	23	
Computer Modern Dunhill	24	
Titan 10	25	

Titan 12	26
Trend PS	27
Artisan 12	28
OCR-A	29
OCR-B	30
Courier 12	31
Metagraphics	32
Times Roman	33
Script 12	34
Times Greek	35
Devanagari	36
Scientific Greek 10	37
Scientific Greek 12	38
USC Greek	39
Computer Modern Funny Font	40
Computer Modern Fibonacci	41
Computer Modern Symbol	42
Comp. Mod. Sanserif Quotation	43
Comp. Mod. Variable Typewriter	44
Miscellaneous	255

Typemod Code:

Normal	0	
Italic	1	
Bold	2	these
Slanted	4	are
Unslanted	8	additive
Extended	16	
Condensed	32	
Demibold	64	
Caps and small caps	129	

Typecharset Code:

Normal	0
Extended	1
ALA	2
Pi	3
Cyrillic	4
Greek	5
Hindi	6
Text	7
Math Extension	8
Math Symbol	9
Math	10
Dingbats	11
UBC Extended	12
Combined	13
IBM PC Extra	14
IBM PC	15
Vertical Spacing	16
Halfspace	17
UM Default	18
Rule	19

THE ELEMENTARY FUNCTION LIBRARY

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

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

List of Entry Points by General Function

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

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

Mathematical Functions

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

FORTRAN Implicitly Called Functions

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

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

Exponentiation: name(base,exponent)

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

ANSI FORTRAN Minimum/Maximum Value

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

---

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

Calling Conventions

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

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

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

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

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

The elementary functions return their values as follows:

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

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

Error Processing

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

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

(1) by using one or more of the optional arguments described below,
(2) by calling the user error monitor, or
(3) by printing an error message on SERCOM and then calling the subroutine ERROR#.

Whenever an elementary function detects an error situation, it generates a default function value and passes control to the EFL error monitor. Although this error monitor is in fact a separate program, it is logically a part of each elementary function and is transparent with respect to the normal linkage conventions.

The EFL error monitor initially attempts to process the optional arguments. If no such arguments were given, or if their processing does not result in the resumption of execution, then the error monitor will formulate an appropriate message. This message is passed, as the sole argument, to the user error monitor or is printed on SERCOM.

With all optional arguments attached, the calling sequence becomes

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

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

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

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

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

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

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

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

where "x.x" and "y.y" are decimal representations of the argument and function value, respectively. The "dod-message" is dependent on the elementary function involved, but generally expresses the set of argument values for which the function is defined in the form

```
(x: a < x < k )
```

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

Messages generated for exponentiation errors take the form:

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

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

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

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

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

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

This rather simple example prints the message on logical I/O unit SERCOM, and then resumes execution with the default function value. Since the messages are generally longer than a terminal output line, some of the message will be lost. Unless the user error monitor returns to the EFL error monitor, the virtual memory space allocated by this latter program will not be released.

Finally, if the optional argument processing did not result in the resumption of execution and no user error monitor is established, then the EFL error monitor will provide, on SERCOM, an error message and a trace of the programs in the current linkage chain, i.e., the sequence of programs which have been called, but which have not yet returned to their calling programs. For example, if a main program named MAIN calls a subroutine named SUB, which attempts to compute DLOG(-5.D0), then the linkage chain is SUB, MAIN, and MTS. After providing this information, the error monitor will call the resident system subroutine ERROR#. If a subsequent $RESTART command is issued, execution will resume with the default function value.

<u>Example 1</u>:

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

<u>Example 2</u>:

If the fourth statement in example 1 is replaced by

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

then execution will be suspended when the 11th negative argument is passed to SQRT.

<u>Example 3</u>:

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

<u>Example 4</u>:

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

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

In addition, the word ERRCNT would be automatically updated to maintain a count of the total number of errors.

Mathematical Functions

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

Square Root

Because SQRT and DSQRT are specifically defined as real-valued functions, they are not defined for negative real arguments. The default function value when the argument is negative is the square root of the absolute value of the argument.

Common and Natural Logarithm

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

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

Exponential

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

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

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

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

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

EXP(x)•[COS(y)+i•SIN(y)]

and that of CDEXP(x+i•y) will be

DEXP(x)•[DCOS(y)+i•DSIN(y)]

Trigonometric Functions

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

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

while the limit for the double-precision functions is

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

The default function value is uniformly zero.

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

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

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

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

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

Inverse Trigonometric Functions

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

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

Hyperbolic Functions

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

$$|x| \le \text{AF.5DC0} = 175.366211,$$

and that of DCOSH and DSINH by

$$|x| \le \text{AF.5DC0FFFFFFFF} = 175.366226196289059.$$

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

Gamma and Log-gamma Functions

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

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

while in decimal these intervals become

GAMMA	$[.138178829 \cdot 10^{-75}, 57.5744171]$
DGAMMA	$[.13817882865895404 \cdot 10^{-75}, 57.5744171142578089]$

ALGAMA $[0,.429370581 \cdot 10^{74}]$
DLGAMA $[0,.429370581008241143 \cdot 10^{74}]$.

Implicitly Called Functions

Complex Arithmetic Operations

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

Algorithm:

The multiplication algorithm takes the form

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

The division algorithm is likewise direct and takes the form

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

with appropriate scaling of the divisor u+iv to avoid floating-point overflow or underflow of the denominator.

Error Conditions:

Both underflow and overflow exceptions may occur during the formation of the final result. Zero-divide exceptions may also occur, but only if u=v=0.

Exponentiation

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

Algorithm:

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

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

replace S by S•S, increment k by 1, and return to step (2); otherwise, R=B**|I|.

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

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

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

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

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

Error Conditions:

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

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

Algorithm:

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

e **(exponent•ln(base)).

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

Error Conditions:

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

DREAL and DIMAG Functions

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

Algorithm:

Although these routines are described in the IBM FORTRAN language manual, the currently available FORTRAN compilers do not recognize these names as anything special. Consequently, it is necessary to explicitly declare them as REAL*8 functions. Otherwise, they will be assigned the default mode of REAL*4.

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

Error Conditions:

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

ANSI Minimum/Maximum Value Functions

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

Algorithm:

These routines are identical in structure, accepting a variable number of arbitrary arguments of the appropriate mode and recognizing no error situations. The resultant modes of these entry points are determined by the first character of the function names as follows: M=INTEGER*4, A=REAL*4 and D=REAL*

8. The number of arguments processed is determined by the last argument flag; and, consequently, addressing or protection exceptions may occur if this flag is not properly set.

I/O SUBROUTINE RETURN CODES

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

```
SCARDS=A+B
```

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

Return codes greater than 4 are normally not passed back to the caller but instead, an error comment is printed and control is returned to MTS command or debug mode. There are two ways to suppress this action and gain control in this situation. First, specifying the ERRRTN modifier on an I/O subroutine call will cause all return codes to be passed back. Second, specifying the NOPROMPT modifier on an I/O subroutine call will suppress prompting messages for a replacement FDname and will cause the return code to be passed back.

A description of the return codes that may occur with a particular file or device is given with the appropriate sections of MTS Volume 4, Terminals and Networks in MTS, and MTS Volume 19, Tapes and Floppy Disks. In addition, a summary is given below. Nonzero return codes marked with an asterisk are normally not passed to the calling program; the others are always passed to the calling program.

Files:

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

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

Magnetic Tape:

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

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

Paper Tape:

Input	0	Successful return
	4	End-of-file
	8*	End-of-tape
	12*	Invalid control command
	16*	Hardware malfunction
	20*	Parity error
Output	8*	Attempt to write on paper-tape reader

Batch Monitor Input:

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

Printed Output:

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

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

Punched Output:

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

Merit/UMnet Network:

Input	0	Successful return
	4	End-of-file read from network. This does not necessarily mean that there is no more data to be read from the network, only that the remote terminal or host has sent an end-of-file.
	8*	Read not allowed; must do a write. This means that the remote host is requesting input from the network connection and, to avoid a deadlock, the local program must not read from the network. The prompting characters, if any, sent by the remote host when it did the read are returned to the user.
	12*	Should not occur
	16*	Connection is closed: no I/O may be done
	20*	Should not occur
	24*	Attention interrupt received from MOUNTed network connection
	28*	Same as return code 8 except that the remote host has requested that the input area be blanked for "n" characters, where "n" is returned as a 2-digit decimal number followed by the prompting characters. A value of "00" means that no specific number of characters has been specified.
	44*	Read on a MOUNTed network connection was terminated by an attention interrupt from the user's terminal. No data is returned.
	48*	Read on a MOUNTed network connection was terminated because no data was received from the network by MTS within the time specified by the "timeout" network device command. No data was returned. (Note: This will not occur unless a "timeout" device command was issued since, by default, input operations are not timed.)
	64*	Should not occur
Output	0	Successful return
	4*	Should not occur
	8*	Write not allowed; must do a read. This means

		that the remote host has issued a write on the network connection and, to avoid a deadlock, the local program must not write on the network.
	12*	Should not occur
	16*	Connection is closed: no I/O may be done
	20*	Should not occur
	24*	Attention interrupt received from MOUNTed network connection
	64*	Should not occur

Floppy Disks:

Input	0	Successful return
	4	End-of-file on diskette
	8	DDAM detected with DDAM=OFF
	12*	CRC error on read operation
	16*	Nonrecoverable error
Output	0	Successful return
	4	Attempt to write nonexistent track or sector
	8	Should not occur
	12*	CRC error on write operation
	16*	Nonrecoverable error
	20*	Attempt to write on write-protected diskette

Most Other Devices:

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

I/O MODIFIERS

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

The device types discussed below in the exceptions to the default modifier bit specifications are the device types as returned by the GDINFO subroutine (see GDINFO subroutine description in this volume). Some of the device types discussed are given below; the remainder are given in the section "System Device List" in this volume.

FILE	Line files
SEQF	Sequential files
HRDR	Batch monitor card input
HPTR	Batch monitor printed output
HPCH	Batch monitor punched output
9TP	9-track magnetic tape
MNET	Merit/UMnet Computer Network
3270	IBM 3278 Display Station terminals

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

First Fullword of Modifier Bits

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

Default: SEQUENTIAL
Exceptions: None

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

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

I/O operations involving <u>line files</u> may be performed with either SEQUENTIAL or INDEXED specified. I/O operations involving <u>sequential files</u> must be done SEQUENTIALly. If the user specifies INDEXED on an I/O operation to a sequential file, an error message is generated unless the global switch SEQFCHK is OFF, in which case the operation is performed as if SEQUENTIAL was specified. Attempting a sequential operation with a starting line number other than 1, e.g., COPY FYLE(2), also gives an error comment if SEQFCHK is ON.

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

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

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

Default: EBCD
Exceptions: None

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

Other device support routines that recognize this modifier pair are:

(1) The UMnet Computer Network routines
(2) The Merit Computer Network routines
(3) The IBM 3278 Display Station routines
(4) The paper-tape routines

For information on the use of this modifier pair in specifications involving the devices listed above, see the respective sections of MTS Volume 4, Terminals and Networks in MTS, and MTS Volume 19, Tapes and Floppy Disks. The list of device support routines recognizing this modifier is subject to change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

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

Default: LOWERCASE
Exceptions: None

The LOWERCASE/CASECONV modifier pair is not device-dependent. If the LOWERCASE modifier is specified, the characters are transmitted unchanged. If the CASECONV modifier is specified, lowercase letters are changed to uppercase letters. This translation is performed in the user's virtual memory region. On input operations, the characters are read into the user's buffer area and then translated. On output operations, the characters are translated in the user's buffer area and then written out. Only the alphabetic characters (a-z) are affected by this modifier. Unlike IBM programming systems, MTS considers the characters ¢, ", and ! as special characters rather than "alphabetic extenders," and thus, the UC modifier does not convert ¢, ", and ! into @, #, and $, respectively. Note that the conversion to uppercase may also be performed by the terminal support routines (see MTS Volume 4, Terminals and Networks in MTS).

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

Default: CC

Exceptions: Line files (FILE), sequential files (SEQF), 9TP, and HPCH
Controlled by device commands for MNET

The NOCC/CC modifier pair is device-dependent. This modifier pair controls whether logical carriage control on output records is enabled. For printers and terminals, the first character of each record is taken as logical carriage control if it is a valid carriage-control character and if the CC modifier is specified. If the first character is not valid as a carriage-control character, the record is written as if NOCC were specified. For further information on logical carriage control, see Appendix H to the section "Files and Devices" in MTS Volume 1, The Michigan Terminal System.

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

Default: ¬PREFIX
Exceptions: None

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

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

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

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

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

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

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

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

    RUN -LOAD 2=A 3=*SINK*

is equivalent to

    COPY A *SINK*@I@PFX

which is also similar to

    LIST A

with a slightly different formatting of the line numbers.

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

Default: ¬PEEL
Exceptions: None

The PEEL modifier pair has two functions, depending upon whether it is specified on input or on output. On input, if the PEEL (GETLINE#) modifier is specified, a line number is removed from the front of the current input line. The line number is converted to internal form (external value times 1000) and returned in the line number parameter during the read operation (see the subroutine descriptions of SCARDS, GUSER, and READ). The complete input line including the line number is read into the user input region, PEEL processing is performed, the line number (if any) is removed, the remainder of the line is shifted left by the number of characters in the line number, and the length to be returned is decremented by the number of characters removed. Thus, the user input region must be large enough to accommodate both the line number and the line itself. The line number must begin in column 1 (leading zeros are permitted). The line-number separator character (defaults to ",") may be used to separate the line number from the line. As an example, consider the following FORTRAN program segment:

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

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

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

is equivalent to

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

Listing the file ABC produces

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

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

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

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

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

is equivalent to

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

Listing the file ABC produces

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

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

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

Default: ¬MCC
Exceptions: None

The machine carriage-control modifier pair is device-dependent and in general its use is discouraged. The MCC modifier is used for printing output (via printers or terminals) from programs producing output in which the first byte of each line is to be used as a machine carriage-control command for output to an IBM 1403 (or 1443) printer. If the MCC modifier is specified and the first byte of the output line is a valid 1403 machine carriage-control command code, the line is spaced accordingly and printing starts with the next byte as column 1. If the first byte is not a valid 1403 machine carriage-control command code, the entire line is printed using single-spacing. Spacing operations performed by machine carriage control occur _after_ the line is printed (as opposed to logical carriage control in which the spacing is performed _before_ each line is printed). Most programs do not produce output using machine carriage control. The MCC modifier pair is ignored for files and devices other than printers, terminals connected through the UMnet or Merit Computer Networks, or IBM 3278 Display Station terminals. For further information on machine carriage control, see Appendix H to the section "Files and Devices" in MTS Volume 1, _The Michigan Terminal System_.

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

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

The TRIM modifier pair is used to control the trimming of trailing blanks from input or output lines. If the TRIM modifier is specified, all trailing blanks _except_ _one_ are trimmed from the line. If ¬TRIM is specified, the line is not changed. For an input operation, trimming does _not_ physically delete the trailing blanks from the line, but only changes the line length count. Note that the UMnet or Merit Computer Network termimal routines unconditionally trim blanks from output lines.

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

Default: ¬SP
Exceptions: None

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

(1) The file routines
(2) The UMnet Computer Network routines
(3) The Merit Computer Network routines
(4) The IBM 3278 Display Station routines
(5) The paper-tape routines

The file routines use the SPECIAL modifier to mean skip on a read operation to a sequential file, and to mean replace on a write operation to a sequential file. For further details, see the section "Files and Devices" in MTS Volume 1, _The Michigan Terminal System_.

For information on the use of this modifier pair in specifications involving the devices listed above, see the corresponding sections of MTS Volume 4, _Terminals and Networks in MTS_, and MTS Volume 19, _Tapes and Floppy Disks_. The list of device support routines recognizing this modifier is subject to change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

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

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

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

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

Default: FWD
Exceptions: None

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

Bit 9	¬ENDFILE	Value: 4194304 (dec)	00400000 (hex)
8	ENDFILE	Value: 8388608	00800000

Default: The setting of the ENDFILE global switch (initially OFF)
Exceptions: None

The ENDFILE modifier pair controls the recognition of the $ENDFILE command delimiter in the input stream. If ENDFILE is specified, the $ENDFILE line is always recognized as a command delimiter. If ¬ENDFILE is specified, the $ENDFILE line is never recognized as a command delimiter (the line is taken as a data line). If neither is specified, the recognition of the $ENDFILE line is governed by the setting of the ENDFILE global switch (initially OFF). See the SET command for further details.

Bit 7 FDUBCONT Value: 16777216 (dec) 01000000 (hex)

Default: ¬FDUBCONT
Exceptions: None

The FDUBCONT modifier may be used to specify that another fullword of modifier bits follows the current fullword. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

Bit 5 NOPROMPT Value: 67108864 (dec) 04000000 (hex)

Default: ¬NOPROMPT
Exceptions: None

The NOPROMPT modifier may be used to allow control to be returned to a program after certain errors occur that would otherwise result in a request for a replacement FDname in conversational mode or program termination in batch mode. If the NOPROMPT modifier is specified (bit 5 in the modifier word is 1) when an I/O subroutine call is made, GR0 will be set to a value (see the section "Special Returns" below) indicating that either the I/O operation terminated because of an error while attempting to open a new logical I/O unit or FDUB, or that the I/O operation was completed with its success or failure indicated by the return code in GR15. This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

Bit 4 MAXLEN Value: 134217728 (dec) 08000000 (hex)

Default: ¬MAXLEN
Exceptions: None

If the MAXLEN modifier is specified (bit 4 in the modifier word is 1) when an I/O <u>input</u> subroutine call is made, only a maximum specified number of bytes of an input record will be returned by the read operation. The second parameter of the input subroutine points to three halfwords instead of the normal single halfword. The first halfword is set to the length of the record returned by the read operation; the second halfword is preset by the caller to specify the maximum record length that is desired; and the third halfword is set to the actual (physical) length of the record. If the incoming record is longer than the maximum length as specified by the second halfword, the record returned will be truncated to the maximum specified length. If the DSR cannot determine the actual length of the record, the third halfword will be set to -1. If the incoming record is less than or equal to the maximum specified length, the first and third halfwords are not guaranteed to be identical values if the TRIM modifier is in effect. This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

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

Default: ¬NOEC
Exceptions: None

If the NOEC modifier is specified (bit 3 in the modifier word is 1) when an I/O subroutine call is made, explicit concatenation will be inhibited, i.e., if an end-of-file (return code 4) occurs, a return will be made to the calling program instead of proceeding with the next member of the concatenation (if any). This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

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

Default: ¬NOATTN
Exceptions: None

If the NOATTN modifier is specified (bit 2 in the modifier word is 1) when an I/O subroutine call is made, all pending attention and timer interrupts, and all attention and timer interrupts occurring during the call, are left pending. This modifier is useful only when used by systems programs (by systems programmers). It may be used only with an I/O subroutine call; it may not be used with an FDname.

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

Default: ¬ERRRTN
Exceptions: None

If the ERRRTN modifier is specified (bit 1 in the modifier word is 1) when an I/O call is made, and if an I/O error occurs, the error return code is passed back to the calling program instead of printing an error comment. The error return code is returned in general register 15. If the NOPROMPT modifier is also specified, an error indication may be returned in register 0 for some error conditions. The error comment may be retrieved by calling the subroutine GDINFO. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

This modifier will cause any calls to the subroutines SETIOERR or SIOERR to be ignored.

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

Default: ¬NOTIFY
Exceptions: None

If the NOTIFY modifier is specified (bit 0 in the modifier word is 1) when an I/O subroutine call is made, GR0 will be set to a value (see the section "Special Returns" below) indicating that the I/O operation did or did not cause a new FDUB to be opened. A new FDUB is opened when

(1) implicit concatenation occurs,
(2) explicit concatenation occurs,
(3) a FDUB or logical I/O unit is used for the first time,
(4) a return is made from implicit concatenation, or
(5) the maximum line length increases.

This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

Second Fullword of Modifier Bits

Bit 31	¬LOG	Value:	1 (dec)	00000001 (hex)
30	LOG		2	00000002

Default: LOG
Exceptions: None

If the LOG modifier is specified, the read or write operation will be logged in the log file, if logging is enabled by the LOG command. By specifying ¬LOG, the user may suppress information from being written into the log file.

Bit 29	¬MACRO	Value:	4 (dec)	00000004 (hex)
28	MACRO		8	00000008

Default: MACRO
Exceptions: None

If the MACRO modifier is specified and the input is being read from *SOURCE* (or equivalent), the MTS macro processor is called to interpret lines for macro commands or macro invocations. If the ¬MACRO is specified, the macro processor is not called. SET MACROS=ON must be specified for this modifier to be effective. The MACRO modifier pair has no effect on the generation of lines by a macro once it is invoked; these lines are always generated whether or not the MACRO or ¬MACRO modifier is subsequently specified.

Bit 27	¬MFR	Value:	16 (dec)	00000010 (hex)
26	MFR		32	00000020

Default: MFR

If the MFR (macro flag required) modifier is specified and the input is being read from *SOURCE* (or equivalent), the ">" macro flag character must be given for lines that are macro <u>invocations</u>. If the ¬MFR modifier is specified, the ">" is not required. The MACRO modifier and SET MACROS=ON must be also be specified for this modifier to be effective. The MFR modifier pair does not affect lines that are macro <u>commands</u>; these always require the flag character.

Certain programs including the MTS command processor and several command-language subsystems (CLSs) read using the ¬MFR modifier.

<u>Special Returns</u>

If the NOPROMPT (bit 5) or NOTIFY (bit 0) modifiers are specified when an I/O subroutine call is made, the bits in GR0 will indicate the result of the subroutine call. If no bits are set (GR0 is zero), the I/O operation was completed and its success or failure is indicated by the return code in GR15. If GR0 is nonzero, the I/O operation terminated without completion. The bit assignments are:

- Bit 31 - The NOTIFY modifier was enabled and a new FDUB was opened as the result of this call, or an old FDUB was used for the first time with the @NOTIFY modifier.
- Bit 30 - The NOPROMPT modifier was enabled and an error occurred while opening a new logical I/O unit or FDUB.
- Bit 29 - The DSR says that no password is required (system mode only).

The values of bits 0-28 are unpredictable and are reserved for future expansion.

SYSTEM DEVICE LIST

The following is a list of all the devices in the University of Michigan hardware configuration as of the date of publication. Each class of device in the system is identified by a three- or four-character device type; each specific device in the system is identified by a three- or four-character device name. In the list below only the form of the device name is given since the actual device names are subject to change without notice.

The device type is the type field returned by the subroutine GDINFO when it is called for information about a particular device (see the GDINFO subroutine description in this volume).

Device Type	Device Name	Explanation
MRXA	LAnn	Memorex 1270 Terminal Controller line
PDP8	CCnn	PDP-11 Data Concentrator line
PDP8	CCOP	PDP-11 Data Concentrator Oper. Console
PDP8	PLTn	Plotter
FDSK	FLPn	Floppy Disk
3270	DSnn	IBM 3270-type Display Station
3284	PTRn	IBM 3284, 3286, or 3287 Printer
MNET	AAnn	Merit/UMnet Network Commun. line
MNET	ABnn	Merit/UMnet Network Commun. line
MNET	ADnn	Merit/UMnet Network Commun. line
MNET	AEnn	Merit/UMnet Network Commun. line
MNET	AFnn	Merit/UMnet Network Commun. line
MNET	ANnn	Merit/UMnet Network Commun. line
MNET	AAOP	Merit/UMnet Network Oper. Console
MNET	ABOP	Merit/UMnet Network Oper. Console
MNET	ADOP	Merit/UMnet Network Oper. Console
MNET	AEOP	Merit/UMnet Network Oper. Console
MNET	AFOP	Merit/UMnet Network Oper. Console
MNET	ANOP	Merit/UMnet Network Oper. Console
3203	PTRn	Memorex 3203 Line Printer
9700	PTRn	Xerox 9700 Page Printer
RDR	RDRn	IBM 2501 Card Reader
PCH	PCHn	IBM 1442 Card Punch
SDA	SDAn	Synchronous Data Adaptor II (BSC) line (remote batch service)
9Tp	T9nn	IBM 3420-compatible Magnetic Tape Unit
3380	Dnnn	IBM 3380-compatible Disk Storage Unit
6280	Dnnn	Amdahl 6280 Disk Storage Units
3805	FBnn	Intel 3805 or 3825 Paging Device

SUBROUTINES USING FILES AND DEVICES

This section provides a summary of the system subroutines that use file name, logical I/O units, and FDUB-pointers. The access column gives the type of file access necessary to call the subroutine (where appropriate); if marked as "---", file access is not checked or is irrelevant to the function of the subroutine. The following access abbreviations are used in the table:

Access	Meaning
R	Read
WC	Write-change
WE	Write-expand
W	Write-change and write-expand
D/R	Destroy/Rename
T/R	Truncate/Renumber
P	Permit

Subroutine	Purpose	Access
BSRF	To backspace records in a file.	R, WC, or WE
CFDUB	To determine if two FDUB-pointers refer to the same file or device.	---
CHGFSZ	To change the SIZE or MAXSIZE of a file.	See below[1]
CHGMBC	To change the number of buffers used to read or write a file.	Any access
CHGXF	To change the expansion factor of a file.	T/R
CHKACC	To determine the access to a file.	Any access
CHKFDUB	To get a FDUB-pointer for a given logical I/O unit; to check if a FDUB-pointer is valid.	---
CHKFILE	To determine the existence of a file.	Any access
CLOSEFIL	To close a file.	See below[2]
CNTLNR	To count a set of lines in a line file.	R
CONTROL	To perform a control operation on a file or device.	See below[3]
CREATE	To create a file.	---
DESTROY	To destroy a file.	D/R
EMPTY	To empty a file.	WC
EMPTYF	To empty a file (FORTRAN-callable).	WC
FREEFD	To release a FDUB-pointer.	See below[2]
FSIZE	To determine the file size needed for a data set.	---
FSRF	To forward space records in a file.	R, WC, or WE
GDINFO	To get information about a file or device.	Any access
GDINFO2	To get information about a file or device (without opening it).	Any access
GDINFO3	To get information about a file or device (without locking it).	Any access
GETFD	To get a FDUB-pointer for a file or device.	---
GETFST	To get the line number of the first line in a file.	Any access
GETLST	To get the line number of the last line in a file.	Any access

Subroutine	Purpose	Access
GFINFO	To get file and catalog information about a file.	See below[4]
GUSER	To read from logical I/O unit GUSER.	R
LETGO	To unlock and relock a file.	See below[5]
LOCK	To explicitly lock a file.	See below[5]
NOTE	To return position information for a sequential file.	Any access
PERMIT	To permit a file.	P
POINT	To position a sequential file.	See below[6]
READ	To read from a file or device.	R
RENAME	To rename a file.	D/R
RENUMB	To renumber a line file.	T/R, or R and W
RETLNR	To return a set of line numbers in a line file.	R or T/R
REWIND	To rewind a logical I/O unit.	See below[7]
REWIND#	To rewind a file or device.	See below[7]
SCARDS	To read from logical I/O unit SCARDS.	R
SETFSAVE	To control system file saving.	P
SETKEY	To set the program key for a file.	P
SETLIO	To attach a file or device to a logical I/O unit.	---
SETLNR	To set a set of line numbers in a line file.	T/R, or R and W
SERCOM	To write on logical I/O unit SERCOM.	See below[8]
SKIP	To position a magnetic tape.	---
SPRINT	To write on logical I/O unit SPRINT.	See below[8]
SPUNCH	To write on logical I/O unit SPUNCH.	See below[8]
TRUNC	To truncate a file.	T/R or WE
UNLK	To explicitly unlock a file.	See below[2]
WRITE	To write on a file or device.	See below[8]
WRITEBUF	To write all changed file buffers.	See below[2]

----

[1]WE to increase SIZE, MAXSIZE, or expansion factor; T/R to decrease SIZE, MAXSIZE, or expansion factor.
[2]Checked by previous operations.
[3]Same as corresponding subroutine for type of operation performed.
[4]P for full sharing information; any access for all other information.
[5]Any access for read lock; P, D/R, T/R, WC, or WE for modify lock; P or D/R for destroy lock.
[6]R to change read pointer; WC or WE to change write pointer; WC to change last pointer or last line number.
[7]Any access except for sequential file, the write pointer will not be reset without WC or WE access.
[8]WE if new line; WC if replacing existing line.

Reader’s Comment Form

System Subroutine Descriptions
Volume 3
April 1981

Errors noted in publication:

Suggestions for improvement:

Your comments will be much appreciated. The completed form may be sent to the Computing Center by Campus Mail or U.S. Mail, or dropped in the Suggestion Box at the Computing Center, NUBS, or BSAD.

Date ____________________

Name ____

Address ____

____

____

Publications
Computing Center
University of Michigan
Ann Arbor, Michigan 48109

Update Request Form

System Subroutine Descriptions
Volume 3
April 1981

Updates to this manual will be issued periodically as errors are noted or as changes are made to MTS. If you desire to have these updates mailed to you, please submit this form.

Updates are also available in the memo files at both the Computing Center and NUBS; there you may obtain any updates to this volume that may have been issued before the Computing Center receives your form. Please indicate below if you desire to have the Computing Center mail to you any previously issued updates.

Name ____

Address ____

____

____

Previous updates needed (if applicable): __________

The completed form may be sent to the Computing Center by Campus Mail or U.S. Mail, or dropped in the Suggestion Box at the Computing Center, NUBS, or BSAD. Campus Mail addresses should be given for local users.

Publications
Computing Center
The University of Michigan
Ann Arbor, Michigan 48109

Users associated with <u>other MTS installations</u> (except the University of British Columbia) should return this form to their respective installations. Addresses are given on the reverse side.

Addresses of other MTS installations:

The University of Alberta
Information Coordinator
352 General Services Bldg.
Edmonton, Alberta
Canada T6G 2H1

Information Officer, NUMAC
Computing Laboratory
The University of Newcastle upon Tyne
Newcastle upon Tyne
England NE1 7RU

Rensselaer Polytechnic Institute
Documentation Librarian
130 Amos Eaton Hall
Troy, New York 12181

Simon Fraser University
Computing Centre
User Services Information Group
Burnaby, British Columbia
Canada V5A 1S6

Wayne State University
Computing Services Center
Academic Services Documentation Librarian
5925 Woodward Ave.
Detroit, Michigan 48202