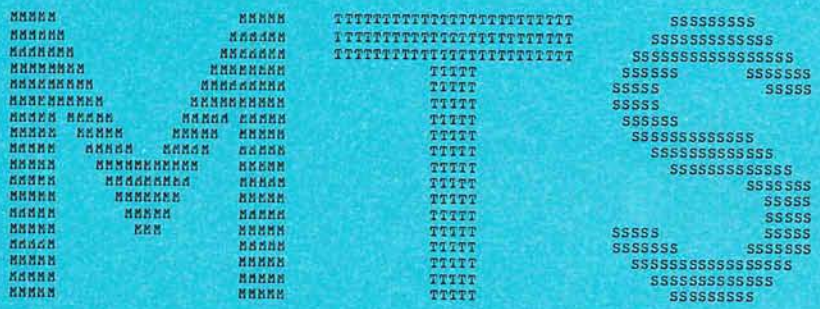
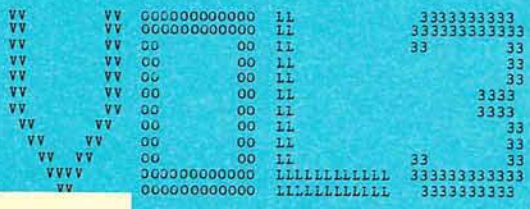
```
*********1*********2*********3*********4*********5*********6*********7*********8*********9*********0*********1*********2*********3*
--------333333----------333333----------333333----------333333----------333333----------333333----------333333-------

JOB NO. 333333        UNIVERSITY OF MICHIGAN TERMINAL SYSTEM  (MODEL EP136)                  19:37:45    THU OCT 07/76

   $COPY *RATES FOR NEW RATES
$SIG VOL3  PRIM1=1M
```





Michigan Terminal System for IBM 360/67 (and later virtual-storage processors)

```
333----------333333----------333333----------333333----------333333-------
```

VOLUME 3

SYSTEM SUBROUTINE DESCRIPTIONS

OCTOBER 1976

M T S

The Michigan Terminal System

VOLUME 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

Revised

The University of Michigan Computing Center
Ann Arbor, Michigan

****
*
* This obsoletes the May 1973 edition.
*
****

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 Computing Center Newsletter, Computing Center Memos, and future updates to this volume for the latest information about changes to MTS.

October 1976

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, when completed, will describe in detail the facilities provided by the Michigan Terminal System. Administrative policies of the Computing Center and the physical facilities provided are described in a separate publication entitled Introduction_to_the_Computing_Center.

The MTS volumes now in print are listed below. The date indicates the most recent edition of each volume; however, since volumes are updated by means of CCMemos, users should check the Memo list, copy the file *MTSVOLUMES, or watch for announcements in the Newsletter, to ensure that their MTS volumes are up to date.

Volume 1: The_Michigan_Terminal_System, April 1976
Volume 2: Public_File_Descriptions, January 1975
Volume 3: System_Subroutine_Descriptions, October 1976
Volume 4: Terminals_and_Tapes, August 1974
Volume 5: System_Services, June 1976
Volume 6: FORTRAN_in_MTS, March 1976
Volume 8: LISP_and_SLIP_in_MTS, June 1976
Volume 9: SNOBOL4_in_MTS, September 1975
Volume 10: BASIC_in_MTS, September 1974
Volume 11: Plot_Description_System, April 1971; reprinted June 1975 with Update 1
Volume 12: PIL/2_in_MTS, December 1974
Volume 14: 360/370_Assemblers_in_MTS, June 1976

Other volumes are in preparation. 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

October 1976

PREFACE TO REVISED VOLUME 3

The October 1976 edition reflects the changes that have been made to MTS since May 1973. Some of these changes were described in Update 1 which was issued in August 1974 and are incorporated in this revision. However, several new subroutines have been added since then and, hence, it was felt that a complete revision of this volume was in order. The revision bars have been deleted and the pages have been renumbered to facilitate the future issuing of updates.

The section "Mathematical Subroutines" has been deleted from this edition. Those subroutines, except for GRAND, RAND, and URAND, have been replaced by the Numerical Analysis and Applications Software (NAAS) package and are described in the corresponding documentation for that package. The descriptions of GRAND, RAND, and URAND remain in this volume.

The section "Macros" has been deleted from this edition and moved to MTS Volume 14.

The section "Carriage Control" has been deleted from this edition and moved to MTS Volume 1.

The following subroutine descriptions have been added to this volume since the August 1974 update to the May 1973 edition.

Array Management Subroutines
BINEBCD
BINEBCD2
CHGFSZ
CHGMBC
CHKACC
CMDNOE
CNFGINFO
CNTLNR
CVTOMR
DUMP, PDUMP
EDIT
EMPTYF
FNAMETRT
FTNCMD
GUINFUPD
IHEATTN
IOH
READBFR
RETLNR
SETKEY
SETLNR

SIOCP
SPELLCHK
STARTF
WRITEBUF

The following subroutine descriptions have been deleted from this volume. Some of these subroutines still exist in the system, but the Computing Center makes no guarantee as to how long they will exist.

DCVC
DCVD
DCVG
FCVTHB
GETIHC, PUTIHC
SETDSN
SETDSR
SETSTA

October 1976

Contents

Preface 3

Preface to Revised Volume 3 . . 5

Using Subroutine Libraries . . . 11

Subroutines Libraries Available in MTS 13

Subject Categories of Subroutines 17
- Character and Numeric Conversion 17
- Date and Time Conversion . . . 17
- File and Device Usage 18
- FORTRAN Usage 19
- Input/Output Routines 19
- Interrupt Processing 20
- PL/I Usage 20
- Status of User and System . . 20
- System Utilities 21
- Virtual Memory Management . . 21

Calling Conventions 23

Resident System and *LIBRARY Subroutines 33
- ADROF 35
- Array Management Subroutines . 37
  - ARINIT 39
  - ARRAY, ARRAY2 40
  - EXTEND, XTEND2 42
  - ERASE 43
  - ERASAL 43
- ASCEBC 45
- ATNTRP 49
- ATTNTRP 51
- BINEBCD 53
- BINEBCD2 55
- Bitwise Logical Functions . . 57
  - AND 57
  - COMPL 57
  - LAND 57
  - LCOMPL 57
  - LOR 57
  - LXOR 57
  - OR 57
  - SHFTL 57
  - SHFTR 57
  - XOR 57
- Blocked Input/Output Routines 59
  - QGETUCB 60
  - QOPEN 61
  - QGET 63
  - QPUT 65
  - QCLOSE 67
  - QFREEUCB 68
  - QCNTRL 69
- BLOKLETR 71
- CALC 73
- CANREPLY 77
- CASECONV 79
- CFDUB 81
- Character Manipulation Routines 83
  - BTD 85
  - COMC 86
  - DTB 87
  - EQUC 89
  - FINDC 90
  - FINDST 92
  - IGC 93
  - LCOMC 95
  - MOVEC 96
  - SETC 97
  - TRNC 98
  - TRNST 99
- CHGFSZ 101
- CHGMBC 103
- CHKACC 105
- CHKFDUB 107
- CHKFILE 109
- CLOSEFIL 111
- CMD 113
- CMDNOE 115
- CNFGINFO 117
- CNTLNR 121
- CONTROL 123
- COST 127
- CREATE 129
- CUINFO 131
- CVTOMR 133
- DESTROY 135
- DISMOUNT 137

DUMP, PDUMP 139
EBCASC 141
EDIT 145
EMPTY 155
EMPTYF 157
ERROR 159
E7090, D7090, E7090P, D7090P .161
FNAMETRT 163
FREAD 165
FREEFD 167
FREESPAC 169
FSIZE 171
FSRF, BSRF 173
FTNCMD 175
GDINF 177
GDINFO 179
GDINFO2 185
GDINFO3 187
GETFD 189
GETFST, GETLST 191
GETIME 193
GETSPACE 195
GFINFO 197
GPSECT, QPSECT, FPSECT 205
GRAND, GRAND1 207
GRGJULDT, GRGJULTM, GRJLSEC .209
GRJLDT, GRJLTM 211
GROSDT 213
GTDJMS 215
GTDJMSR 217
GUINFO, CUINFO 219
GUINFUPD 233
GUSER 235
GUSERID 237
IOH 239
JLGRDT, JLGRTM 241
JMSGTD, JTUGTD 243
JMSGTDR, JTUGTDR 245
JULGRGDT, JULGRGTM, JLGRSEC .247
KEYWRD 249
KWSCAN 253
LETGO 273
LINK, LINKF 275
LIOUNITS 281
LOAD, LOADF 283
LOADINFO 289
LOCK 293
LODMAP 297
Logical Operators 299
ICLC 299
IED 299
IEDMK 299
IMVC 299
INC 299
IOC 299
ITR 299
ITRT 299
IXC 299
MOUNT 303
MTS 305
MTSCMD 307
NOTE 309
OSGRDT 311
PERMIT 313
PGNTTRP 317
POINT 319
Printer Plot Routines 321
PLOT1 325
PLOT2 326
PLOT3 327
PLOT4 328
PLOT14 329
PRCHAR 330
PREND 331
PRPLOT 332
STPLT1 334
STPLT2 335
SETLOG 336
OMIT 337
QUIT 339
RCALL 341
READ 343
READBFR 347
RENAME 349
RENUMB 351
RETLNR 353
REWIND 357
REWIND# 359
RSTIME 361
SCANSTOR 363
SCARDS 365
SDUMP 367
SERCOM 371
SETIME 373
SETIOERR 377
SETKEY 379
SETLIO 383
SETLNR 385
SETPFX 389
SIOC 391
SIOCP 399
SIOERR 403
SKIP 405
SORT 409
SORT2, SORT3 411
SPELLCHK 413

SPIE 415
SPRINT 419
SPUNCH 421
STARTF 423
STDDMP 425
SYSTEM 427
TICALL 429
TIME 433
TIMNTRP 437
TRACER 439
TRUNC 443
TWAIT 445
UNLK 447
UNLOAD, UNLDF 449
URAND 451
WRITE 453
WRITEBUF 457
XCTL, XCTLF 459

PL/I Library Subroutines 465
ATTACH 467
BATCH 469
CNTL 471
CPUTIME 473
ELAPSED 475
FINFO, TFINFO, RFINFO 477
IHEATTN 479
IHENOTE, IHEPNT 481
IHEREAD, IHERITE 483
NEXTKEY, LASTKEY 485
PLCALL, PLCALLD, PLCALLE,
PLCALLF 487
PL1ADR 491
PL1RC 493
RAND 495
SIGNOFF 497
USERID 499

The Elementary Function Library 501

I/O Subroutine Return Codes . .515

I/O Modifiers 521

External Symbol Index 533

October 1976

October 1976

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, *GENLIB, is given in MTS Volume 2. Details of the form of library control records can be found in "The Dynamic Loader" section in MTS Volume 5.

October 1976

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 5.

*PL1LIB

This file contains subroutines needed to support PL/I programs. A few of these which were added or modified by the Computing Center are described in this volume. 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.

*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.

*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).

*IG

This file contains the subroutines for use with the Integrated Graphics (IG) system.

*ALGOLWLIB
*ALGOLWXLIB

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

*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.

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:SSP

This file contains the IBM Scientific Subroutine Package.

NAAS:OLDLIB

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

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 the 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 insure 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 are 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.

October 1976

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 from assembly language and/or FORTRAN.

Character and Numeric Conversion

ASCEBC	USASCII to EBCDIC translation	Assembly
BINEBCD	Binary input to EBCDIC translation	Assembly
BINEBCD2	Binary input to EBCDIC translation	Assembly
CASECONV	Lowercase to uppercase conversion	Assembly
CVTOMR	OMR card image to EBCDIC translation	Assembly, FORTRAN
EBCASC	EBCDIC to USASCII translation	Assembly
E7090,D7090,E7090P,D7090P	7090 to 360 floating-point conversion	Assembly, FORTRAN
IOH	Numeric input/output conversion	Assembly
SIOC	Numeric input/output conversion	Assembly, FORTRAN
SIOCP	Numeric input/output conversion	Assembly, FORTRAN

Date and Time Conversion

GRGJULDT	Gregorian to Julian date and time	Assembly
GRGJULTM	Gregorian to Julian time	Assembly
GRJLDT	Gregorian to Julian date and time	FORTRAN
GRJLSEC	Gregorian to Julian time	Assembly
GRJLTM	Gregorian to Julian time	FORTRAN
GROSDT	Gregorian to OS date	Assembly, FORTRAN
GTDJMS	Gregorian to Julian date and time	FORTRAN
GTDJMSR	Gregorian to Julian time	Assembly
JLGRDT	Julian to Gregorian date and time	FORTRAN
JLGRSEC	Julian to Gregorian time	Assembly
JLGRTM	Julian to Gregorian time	FORTRAN
JMSGTD	Julian to Gregorian date and time	FORTRAN
JMSGTDR	Julian to Gregorian date and time	Assembly
JULGRGDT	Julian to Gregorian date and time	Assembly
JULGRGTM	Julian to Gregorian time	Assembly
OSGRDT	OS to Gregorian date	Assembly, FORTRAN
TIME	Get time of day, CPU and elapsed time	Assembly, FORTRAN

File and Device Usage

CFDUB	Compare FDUB-pointers	Assembly, FORTRAN
CHGFSZ	Change file size	Assembly, FORTRAN
CHGMBC	Change number of file buffers	Assembly, FORTRAN
CHKACC	Check access to file	Assembly, FORTRAN
CHKFDUB	Get a FDUB-pointer for a file	Assembly, FORTRAN
CHKFILE	Determine existence of a file	Assembly, FORTRAN
CLOSEFIL	Close a file	Assembly, FORTRAN
CNTLNR	Count number of lines in a file	Assembly, FORTRAN
CREATE	Create a file	Assembly, FORTRAN
DESTROY	Destroy a file	Assembly, FORTRAN
EDIT	Edit a file	Assembly, FORTRAN
EMPTY	Empty a file	Assembly, FORTRAN
EMPTYF	Empty a file	Assembly, FORTRAN
FNAMETRT	Check for legal file name	Assembly
FREEFD	Free a file or device	Assembly, FORTRAN
FSIZE	Determine size required for a file	Assembly, FORTRAN
FSRF,BSRF	Forward and backspace records in a file	Assembly, FORTRAN
GDINF	Get file information	FORTRAN
GDINFO	Get file or device information	Assembly
GDINFO2	Get file or device information	Assembly
GDINFO3	Get file or device information	Assembly
GETFD	Get a file or device	Assembly, FORTRAN
GETFST,GETLST	Get first and last line numbers of a line file	Assembly, FORTRAN
GFINFO	Get file and catalog information	Assembly, FORTRAN
LETGO	Periodically unlock and lock a file	Assembly, FORTRAN
LOCK	Lock a file	Assembly, FORTRAN
NOTE	Remember sequential file pointers	Assembly, FORTRAN
PERMIT	Permit a file	Assembly, FORTRAN
POINT	Change sequential file pointers	Assembly, FORTRAN
RENAME	Rename a file	Assembly, FORTRAN
RENUMB	Renumber a file	Assembly, FORTRAN
RETLNR	Return line numbers of a file	Assembly, FORTRAN
REWIND	Rewind a logical I/O unit	FORTRAN
REWIND#	Rewind a file or magnetic tape	Assembly
SETKEY	Set program key for a file	Assembly, FORTRAN
SETLNR	Set line numbers of a file	Assembly, FORTRAN
TRUNC	Truncate a file	Assembly, FORTRAN
UNLK	Unlock a file	Assembly, FORTRAN
WRITEBUF	Write file buffers	Assembly, FORTRAN

October 1976

FORTRAN Usage

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

Input/Output Routines

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

Interrupt Processing

ATNTRP	Attention interrupt processing	FORTRAN
ATTNTRP	Attention interrupt processing	Assembly
GETIME	Timer interrupt processing	Assembly
PGNTTRP	Program interrupt processing	Assembly
RSTIME	Timer interrupt processing	Assembly
SETIME	Timer interrupt processing	Assembly
SPIE	Program interrupt processing	Assembly
TICALL	Timer interrupt processing	FORTRAN
TIMNTRP	Timer interrupt processing	Assembly
TRACER	Elementary function library error processing	Assembly, FORTRAN
TWAIT	Timer interrupt processing	Assembly, FORTRAN

PL/I Usage

ATTACH	Attach PL/I files	PL/I
BATCH	Terminal or batch status	PL/I
CNTL	Execute a device support operation	PL/I
CPUTIME	Get CPU time	PL/I
ELAPSED	Get elapsed time	PL/I
FINFO	Get file or device information	PL/I
IHEATTN	Attention interrupt processing	PL/I
IHENOTE	Remember sequential pointers	PL/I
IHEPNT	Change sequential pointers	PL/I
IHEREAD	Read from PL/I	PL/I
IHERITE	Write from PL/I	PL/I
NEXTKEY	Find key of next PL/I record	PL/I
PLCALL	S-type call from PL/I	PL/I
PL1ADR	Get address of a PL/I variable	PL/I
PL1RC	Determine return code from subroutine	PL/I
RAND	Uniform random numbers	PL/I
SIGNOFF	Signoff the user	PL/I
USERID	Get user ccid	PL/I

Status of User and System

CANREPLY	Terminal or batch status	Assembly, FORTRAN
CNFGINFO	Get system configuration information	Assembly
COST	Get cost of current signon	Assembly, FORTRAN
CUINFO	Change user status information	Assembly
GUINFO	Get user status information	Assembly
GUINFUPD	Update user status information	Assembly
GUSERID	Get user ccid	Assembly, FORTRAN
LOADINFO	Get symbol or address information	Assembly

System Utilities

BLOKLETR	Produce block letters	Assembly, FORTRAN
CALC	Call $CALC routines	Assembly, FORTRAN
CMD	Execute an MTS command	Assembly, FORTRAN
CMDNOE	Execute an MTS command without echoing	Assembly, FORTRAN
CONTROL	Execute a device support operation	Assembly, FORTRAN
DISMOUNT	Dismount a tape	Assembly, FORTRAN
ERROR	Terminate execution with error	Assembly, FORTRAN
GRAND	Normally distributed random number	Assembly, FORTRAN
KEYWRD	Keyword processing	Assembly
KWSCAN	Keyword processing	Assembly
MOUNT	Mount a tape	Assembly, FORTRAN
MTS	Return to MTS command mode	Assembly, FORTRAN
MTSCMD	Return to MTS and execute a command	Assembly, FORTRAN
Printer Plot Routines		
	Produce plots	Assembly, FORTRAN
QUIT	Signoff user at next MTS command	Assembly, FORTRAN
SETLIO	Assign logical I/O units	Assembly, FORTRAN
SETPFX	Set prefix character	Assembly, FORTRAN
SKIP	Space a magnetic tape	Assembly, FORTRAN
SORT	Sort and merge records	Assembly, FORTRAN
SORT2	Sort vectors	Assembly, FORTRAN
SORT3	Sort vectors	Assembly, FORTRAN
SPELLCHK	Spelling check	Assembly, FORTRAN
SYSTEM	Terminate execution	Assembly, FORTRAN
URAND	Uniformly distributed random number	Assembly, FORTRAN

Virtual Memory Management

DUMP,PDUMP		
	Dump storage	FORTRAN
FREESPAC	Release storage	Assembly, FORTRAN
GETSPACE	Acquire storage	Assembly, FORTRAN
GPSECT, FPSECT, QPSECT		
	Psect storage management	Assembly
LINK	Dynamic loading	Assembly
LINKF	Dynamic loading	FORTRAN
LOAD	Dynamic loading	Assembly
LOADF	Dynamic loading	FORTRAN
LOADINFO	Get loader table information	Assembly, FORTRAN
LODMAP	Produce loader map	Assembly, FORTRAN
SCANSTOR	Scan storage blocks	Assembly
SDUMP	Dump storage and registers	Assembly
STARTF	Dynamic loading	FORTRAN
STDDMP	Dump storage	Assembly
UNLDF	Dynamic unloading	FORTRAN
UNLOAD	Dynamic unloading	Assembly
XCTL	Dynamic loading	Assembly
XCTLF	Dynamic loading	FORTRAN

October 1976

October 1976

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 tc 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-byte 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 mode, respectively. Examples of subroutines which return results in this manner are the FORTRAN IV Library Subprograms, such as EXP, ALOG, or SIN. The fourth method can be used by a subprogram. An example would be a subprogram called by the statement

```
CALL MATADD(A,B,C,M,N)
```

which might add the MxN matrices A and B together and store the result in C.

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.

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.

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

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.

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     0,0(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.

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

```
SUBRA     ENTER 12,SA=SAVE
          LA    11,1
          •
          •
          •
SAVE      DS    18F
          •
          •
          •
          L     3,4(0,11)
          L     0,0(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.

October 1976

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.

October 1976

October 1976

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.

October 1976

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.

October 1976

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)
```

Parameters:

- noar — an integer in the range 1 to 52428, 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).

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 an 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.

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,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).

Examples:

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

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.

Returns Codes:

Normal	Array erased successfully.
&s1	Array name not recognized.

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
```

October 1976

October 1976

ASCEBC

TRANSLATE TABLE DESCRIPTION

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

Location: Resident System

Calling Sequences:

```
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: A USASCII/EBCDIC translation table is shown on the next two pages. This table is for 7-bit ASCII, i.e., the eighth (high order) parity bit is always shown as zero in this table. The translation is actually done using a 256-entry "folded" table in which the first and second halves are identical so that the effect is to ignore the USASCII parity bit.

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

Example:

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

The above example will translate the USASCII characters of the 100-byte region at location REG into EECDIC characters.

USASCII/EBCDIC Translation Table

USASCII (7-bit)				EBCDIC (8-bit)		USASCII (7-bit)				EBCDIC (8-bit)	
Oct	Hex	Name	TTY	Hex	Name	Oct	Hex	Name	TTY	Hex	Name
000	00	NUL	CT-SFT-P	00	NUL	040	20	Space	Space	40	Space
001	01	SOH	CTRL-A	01	SOH	041	21	!	!	5A	!
002	02	STX	CTRL-B	02	STX	042	22	"	"	7F	"
003	03	ETX	CTRL-C	03	ETX	043	23	#	#	7B	#
004	04	EOT	CTRL-D	37	EOT	044	24	$	$	5B	$
005	05	ENQ	CTRL-E	2D	ENQ	045	25	%	%	6C	%
006	06	ACK	CTRL-F	2E	ACK	046	26	&	&	50	&
007	07	BEL	CTRL-G	2F	BEL	047	27	'	'	7D	'
010	08	BS	CTRL-H	16	BS	050	28	(	(	4D	(
011	09	HT	CTRL-I	05	HT	051	29	)	)	5D	)
012	0A	LF	LINE FEED	25	LF	052	2A	*	*	5C	*
013	0B	VT	CTRL-K	0B	VT	053	2B	+	+	4E	+
014	0C	FF	CTRL-L	0C	FF	054	2C	,	,	6B	,
015	0D	CR	RETURN	0D	CR	055	2D	-	-	60	-
016	0E	SO	CTRL-N	0E	SO	056	2E	.	.	4B	.
017	0F	SI	CTRL-O	0F	SI	057	2F	/	/	61	/
020	10	DLE	CTRL-P	10	DLE	060	30	0	0	F0	0
021	11	DC1	CTRL-Q	11	DC1	061	31	1	1	F1	1
022	12	DC2	CTRL-R	12	DC2	062	32	2	2	F2	2
023	13	DC3	CTRL-S	13	DC3	063	33	3	3	F3	3
024	14	DC4	CTRL-T	3C	DC4	064	34	4	4	F4	4
025	15	NAK	CTRL-U	3D	NAK	065	35	5	5	F5	5
026	16	SYN	CTRL-V	32	SYN	066	36	6	6	F6	6
027	17	ETB	CTRL-W	26	ETB	067	37	7	7	F7	7
030	18	CAN	CTRL-X	18	CAN	070	38	8	8	F8	8
031	19	EM	CTRL-Y	19	EM	071	39	9	9	F9	9
032	1A	SUB	CTRL-Z	3F	SUB	072	3A	:	:	7A	:
033	1B	ESC	CT-SFT-K	27	ESC	073	3B	;	;	5E	;
034	1C	FS	CT-SFT-L	1C	IFS	074	3C	<	<	4C	<
035	1D	GS	CT-SFT-M	1D	IGS	075	3D	=	=	7E	=
036	1E	RS	CT-SFT-N	1E	IRS	076	3E	>	>	6E	>
037	1F	US	CT-SFT-O	1F	IUS	077	3F	?	?	6F	?

USASCII/EBCDIC Translation Table

USASCII(7-bit)				EBCDIC(8-bit)		USASCII(7-bit)				EBCDIC(8-bit)	
Oct	Hex	Name	TTY	Hex	Name	Oct	Hex	Name	TTY	Hex	Name
100	40	@	@	7C	@	140	60	Grave	NONE	9A	NONE
101	41	A	A	C1	A	141	61	a	NONE	81	a
102	42	B	B	C2	B	142	62	b	NONE	82	b
103	43	C	C	C3	C	143	63	c	NONE	83	c
104	44	D	D	C4	D	144	64	d	NONE	84	d
105	45	E	E	C5	E	145	65	e	NONE	85	e
106	46	F	F	C6	F	146	66	f	NONE	86	f
107	47	G	G	C7	G	147	67	g	NONE	87	g
110	48	H	H	C8	H	150	68	h	NONE	88	h
111	49	I	I	C9	I	151	69	i	NONE	89	i
112	4A	J	J	D1	J	152	6A	j	NONE	91	j
113	4B	K	K	D2	K	153	6B	k	NONE	92	k
114	4C	L	L	D3	L	154	6C	l	NONE	93	l
115	4D	M	M	D4	M	155	6D	m	NONE	94	m
116	4E	N	N	D5	N	156	6E	n	NONE	95	n
117	4F	O	O	D6	O	157	6F	o	NONE	96	o
120	50	P	P	D7	P	160	70	p	NONE	97	p
121	51	Q	Q	D8	Q	161	71	q	NONE	98	q
122	52	R	R	D9	R	162	72	r	NONE	99	r
123	53	S	S	E2	S	163	73	s	NONE	A2	s
124	54	T	T	E3	T	164	74	t	NONE	A3	t
125	55	U	U	E4	U	165	75	u	NONE	A4	u
126	56	V	V	E5	V	166	76	v	NONE	A5	v
127	57	W	W	E6	W	167	77	w	NONE	A6	w
130	58	X	X	E7	X	170	78	x	NONE	A7	x
131	59	Y	Y	E8	Y	171	79	y	NONE	A8	y
132	5A	Z	Z	E9	Z	172	7A	z	NONE	A9	z
133	5B	[	SHIFT-K	AD	[	173	7B	{	NONE	8B	{
134	5C	Bkslsh	SHIFT-L	BA	NONE	174	7C	|	NONE	4F	|
135	5D	]	SHIFT-M	BD	]	175	7D	}	ALT MODE	9B	}
136	5E	Carat	SHIFT-N	AA	NONE	176	7E	Tilde	NONE	5F	¬
137	5F	_	SHIFT-O	6D	_	177	7F	DEL	RUBOUT	07	DEL

October 1976

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.

October 1976

ATTNTRP

SUBROUTINE DESCRIPTION

Purpose: To allow control to be returned to the user on an attention interrupt from a terminal device (ATTN key on 2741, BREAK on Teletype, etc.).

Location: Resident System

Alt. Entry: ATTNT

Calling Sequences:

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

Parameters:

GR0 should contain zero or the location to transfer control to if an attention interrupt occurs.
GR1 should contain the location of a 72-byte save region for storing pertinent information.

Return Codes:

None.

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 of the save region contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region.

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.

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 interrupt 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 *,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'
```

October 1976

BINEBCD

SUBROUTINE DESCRIPTION

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

Location: Resident System

Calling Sequence:

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

Parameters:

GR1 contains the location of the 160-byte region containing the binary card image.

GR2 contains the location of the 80-byte region to contain the converted EBCDIC form.

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.

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.

October 1976

BINEBCD2

SUBROUTINE DESCRIPTION

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

Location: Resident System

Calling Sequence:

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

Parameters:

GR1 contains the location of the 160-byte region containing the binary card image.
GR2 contains the location of the 80-byte region to contain the converted EBCDIC form.
GR3 contains the location of an 80-byte work area for the subroutine.

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

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

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.

October 1976

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 in-line code by the FORTRAN-H compiler by specifying the XL option (see the section "*FTN Interface" in MTS Volume 6 for details). Caution should be exercised in their use. The functions AND, OR, and COMPL are always generated in-line 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 in-line 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.

October 1976

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:

- name 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.
- ptr is the location of a word in which the pointer to the UCB will be placed.
- rc4 is the statement label to transfer to if a nonzero return code is encountered.

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

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(num*len,len)

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 len. 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 is the 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-or-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 tape DSR and no deblocking is done by QGET since QOPEN has turned blocking on in the tape DSR.

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

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 tape DSR.

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 tape DSR.

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)

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.

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)

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 "Magnetic Tape User's Guide" in MTS Volume 4.

ptr is the location of a word which contains a UCB-pointer as returned by QGETUCB.

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 tape device support routines.

October 1976

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 "). 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
```

October 1976

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)

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
- Bit 26: 0 - call TRACER subrcutine on error if no character output is produced
  - 1 - do not call TRACER on error

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.

outparm (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.

rc4,rc8 are statement labels to transfer to if the corresponding return code occurs.

VL is a parameter to the CALL macro which signifies that the calling sequence has a variable number of parameters.

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.

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 inparm is 0 or omitted, then the input is read from SCARDS. If inparm 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 an error occurs, the only way the user can be notified is via a call to TRACER. This action can be inhibited by the setting of bit 26. 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.

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)
          .
          .
100       PRINT1
1         FORMAT(1X,'BAD EXPRESSION')
          .
          .
```

```
      200   PRINT 2
      2     FORMAT(1X,'INSUFFICIENT OUTPUT LENGTH')
```

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.

October 1976

CANREPLY

SUBROUTINE DESCRIPTION

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

Location: Resident System

Alt. Entry: CREPLY

Calling Sequences:

Assembly: CALL CANREPLY

FORTRAN: CALL CREPLY(&rc4)

Parameters:

rc4 is the statement label to transfer to if the return code 4 occurs.

Return Codes:

0	Yes	(conversational)
4	No	(batch)

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.

78 CANREPLY

October 1976

CASECONV

TRANSLATE TABLE DESCRIPTION

Contents: A translate table to convert lowercase letters (a-z) into their uppercase equivalents (A-Z), and to leave all other characters unchanged.

Location: Resident System

Calling Sequences:

```
Assembly: L     r,=V(CASECONV)
          TR    name,0(r)
```

Parameters:

r is a general register that will contain the address of the CASECONV translate table.

name is the location of the region to be translated.

Example:

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

The above example will convert the lowercase letters of the 100-byte region at location REG into uppercase letters.

October 1976

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

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.

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.

October 1976

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.

dnumb is set to zero if the field integer contains all blanks; dnumb is set to one if the field contains all zeros.

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.

October 1976

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.

October 1976

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)
```

October 1976

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.

October 1976

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 19), 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 are statement labels to transfer to if the corresponding return codes occur.

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 the (shared) file).
24 Bad parameters (i.e., bad FDUB-pointer, not a file, etc.).
28 Inconsistent size parameter (see Note 1 below).
32 No disk space available for expansion.
36 The space allocated to this account has been exceeded.

Notes:

(1) 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.
(2) A request for an absolute size of zero is defined to mean truncate the file.
(3) 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.

October 1976

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 19), 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 are statement labels to transfer to if the corresponding return codes occur.

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 the (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:

(1) The maximum number of buffers set is not 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 opened, and may then be changed dynamically by a call to CHGMBC.

(2) In general, large line files will be more efficient than large sequential files due to an increase in the maximum number of buffers allowed.

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.

October 1976

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

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 19), 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.

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 the way to obtain a FDUB-pointer for a file or device attached to a specific logical I/O unit. 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)
          .
          .
UNIT      DC    F'6'
```

FORTRAN:

```
          INTEGER*4 CHKFDB,X,UNIT
          DATA UNIT/6/
            .
            .
          X = CHKFDB(UNIT)
```

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

October 1976

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 are the statement labels to transfer to if the equivalent return codes occur.

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 Hardware error or software inconsistency encountered.
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.

October 1976

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)

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 19), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

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.

October 1976

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 len is a fullword-aligned address and the first two bytes specified 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.

Description: This subroutine does a return 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 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.

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.

October 1976

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 len is a fullword-aligned address and the first two bytes specified are zero, it is assumed len specifies a fullword integer. Otherwise, len is assumed to be a halfword.

Description: This subroutine does a return 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* or *MSINK*, regardless of the setting of the 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.

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.

October 1976

CNFGINFO

SUBROUTINE DESCRIPTION

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

Location: Resident System

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.

```
CNFGINFD DSECT
*        VECTOR OF INFORMATION CONCERNING CONFIGURATION OF MACHINE
*        ON WHICH WE ARE RUNNING
*
CISYSTEM DC    X'0370'           TYPE OF SYSTEM
CICPUID  DS    0XL8              RESULT OF STORE CPUID ON
*                                LOWEST ADDRESS CPU IN THE SYSTEM
CIVERSCD DC    X'00'             VERSION CODE
CIID#    DC    X'010002'         SERIAL NUMBER OF CPU
CIMODEL  DC    X'0470'           MODEL NUMBER OF SYSTEM
CIMCEL   DC    H'0'              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).
CIEXTIDL DC    H'0'              LENGTH OF EXTENDED CPU ID
CIEXTID  DC    A(0)              LOCATION OF EXTENDED CPU ID
*
*        THE FOLLOWING 64 BITS ARE EACH ASSOCIATED WITH A PARTICULAR
*        FEATURE OR RPQ AS INDICATED.
*
CIFEATUR DC    X'F780200000000000'
*        FIRST BYTE
CIDEC    EQU   X'80'             DECIMAL INSTRUCTIONS - AP,CP,DP,ED
*                                EDMK,MP,SP,ZAP, AND SRP IF 370
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'              SWAP REGISTER INSTRUCTION - SWPR
CISLT    EQU   X'01'              SEARCH LIST INSTRUCTION - SLT
*
*        THIRD BYTE
*
CIMXRDD  EQU   X'80'              360/67 MIXED PRECISION FLOATING
*                                 POINT WITH STORE ROUNDED - LX,AX,
*                                 SX,STRE,STRD
CIDIRCTL EQU   X'40'              DIRECT CONTROL (NOT 360/67
*                                 VERSION) - RDD,WRD
CIBAS    EQU   X'20'              370 BAS AND BASR RPQ
CIEXTADR EQU   X'10'              EXTENDED (I.E., 31 BIT) ADDRESSING

CIXTRA   DS    2D                 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    FE3'4.0'            GUARANTEED MINIMUM VERSION
CIVUMMPS DC    FE3'4.0'            SUPERVISOR VERSION
CIVMTS   DC    FE3'4.0'            MTS CMMD LANG/FILE SYSTEM VERSION
CIVSPOOL DC    FE3'4.0'            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')       ABSOLUTE (UNPAGED) SHARED MEMORY
CIVMSH   DC    A(X'100000',X'39FFFF') PAGED SHARED MEMORY
CIVMSYS  DC    A(X'500000',X'5FFFFF') PRIVATE SYSTEM STORAGE
CIVMUSER DC    A(X'600000',X'CFFFFF') PRIVATE USER STORAGE
*
*        THE FOLLOWING WORD GIVES THE FIRST ADDRESS IN THE SEGMENT
*        USED BY THE VIRTUAL MACHINE SUPPORT IN THE SUPERVISOR.
CIVMSEG  DC    A(X'800000')
```

October 1976

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 19), or
  (c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).
- first is the location of a fullword containing the internal line number of the first line 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 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 The file was counted successfully.
4 The file does not exist.
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 the (shared) file).
24 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,-99999999,99999999,CNT)
```

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

October 1976

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 19), 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 are statement labels to transfer to if the equivalent return codes occur.

Return Codes:

0 Successful return from DSR.
4 No control entry or illegal unit 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
9TP	- Any control command (9-track magnetic tape).
7TP	- Any control command (7-track magnetic tape).
PTPR	- Any control command (Paper tape reader).
7772	- Any Audio Response Unit device command.
PDP8	- Any of the Data Concentrator device support commands as normally entered after SOH SOH (i.e., CTRL-A CTRL-A from Teletypes or !A!A from IBM terminals). The SOH SOH sequence should not be given as part of the device control information.
2741 TTY MRXA	- Any of the Memorex device support commands as normally entered after a "%" sign. The percent sign should not be given as part of the device control information.
HPTR HPCH HBAT	- Any control command legal for *PRINT*, *PUNCH*, and *BATCH*, respectively.
MNET	- Any control command (MERIT Computer Netowrk).
3270	- Any 3270 device support command.
3066	- Any 3270 device support command.
FILE	- See MTS Volume 1.
SEQF	- See MTS Volume 1.
BNCH	- Any control command for the benchmark driver.

See the various terminal user's guides in MTS Volume 4 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.

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   H'3'
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.

October 1976

October 1976

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:
```
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, and unreleased paper-tape output.

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:     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.

October 1976

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.

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 are statement labels to transfer to if the equivalent return codes occur.

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.

October 1976

CUINFO

SUBROUTINE DESCRIPTION

Description: See the GUINFO, CUINFO subroutine description for the details of CUINFO and GUINFO.

October 1976

October 1976

CVTOMR

SUBROUTINE DESCRIPTION

Purpose: To convert an OMR card image contained in binary input format into EBCDIC format.

Location: Resident System

Calling Sequence:

Assembly: CALL CVTOMR,(inreg,inlen,outreg,outlen)

FORTRAN: CALL CVTOMR(inreg,inlen,outreg,outlen,&rc4, &rc8,&rc12)

Parameters:

- inreg is the location of the region containing the OMR card image.
- inlen is the location of the halfword (INTEGER*2) length of the region inreg. len must be at least 80.
- outreg is the location of the region containing the converted EBCDIC format of the OMR card.
- outlen is the location of the halfword (INTEGER*2) length of the converted OMR card. On the initial call to CVTOMR, outlen must be zero.

Return Codes:

0 Successful return.
4 Another OMR card image is needed since the previous card image indicated a continuation. outreg and outlen must be left unchanged for the next call to CVTOMR.
8 Not used.
12 Illegal OMR card type. The card image has been copied unconverted to the output region.

Description: The subroutine CVTOMR converts OMR cards originally read in column binary input format into EBCDIC format. The OMR card must be of the type currently used at the Computing Center or the type previously used from the University of Waterloo.

The subroutine must be called more than once in succession if the OMR card indicates a continuation to another card.

Example: Assembly:

```
          MVC   OUTLEN,=H'0'
          CALL  CVTOMR,(INREG,INLEN,OUTREG,OUTLEN)
          C     15,=F'4'
          BL    EXIT
          BE    CONT
          BH    ERROR
           .
           .
INREG     DS    CL80              OMR card image
OUTREG    DS    CL256             EBCDIC format
INLEN     DC    H'80'
OUTLEN    DS    H
```

FORTRAN:

```
      INTEGER INREG(20),OUTREG(64)
      INTEGER*2 INLEN/80/,OUTLEN
        .
        .
      OUTLEN = 0
      CALL CVTOMR(INREG,INLEN,OUTREG,OUTLEN,
     &            &100,&100,&200)
```

In the above examples, the OMR card image contained in the region INREG is converted to EBCDIC format and placed in the region OUTREG.

October 1976

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 are statement labels to transfer to if the equivalent return codes occur.

Return Codes:

0 Successful return.
4 The file is *SOURCE*, *SINK*, *MSOURCE*, *MSINK*, or *PUNCH* 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 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.

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 pseudo-device 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 pseudo-device 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.

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 pseudo-devices named *T1* and *T2*. The first assembly example uses the CALL macro and the second uses the DISMOUNT macro.

October 1976

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.

October 1976

EBCASC

TRANSLATE TABLE DESCRIPTION

Purpose: To translate EBCDIC characters into even-parity 8-bit USASCII characters. An inverse table (ASCEBC) is also available.

Location: Resident System

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: An EBCDIC/USASCII translation table is shown on the next two pages. Note that in this table, all EBCDIC characters which do not appear explicitly translate to USASCII NUL (octal 000 character). Both the EBCDIC character NL (new-line), hexadecimal 15, and the EBCDIC character LF (line-feed), hexadecimal 25, translate to USASCII LF, octal 012. All characters are translated to even-parity USASCII by EBCASC.

See the ASCEBC subroutine for a table to translate from USASCII into EBCDIC.

Example:

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

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

EBCDIC/USASCII Translation Table

EBCDIC (8-bit)		USASCII (7-bit)				EBCDIC (8-bit)		USASCII (7-bit)			
Hex	Name	Oct	Hex	Name	TTY	Hex	Name	Oct	Hex	Name	TTY
00	NUL	000	00	NUL	CT-SFT-P	3F	SUB	032	1A	SUB	CTRL-Z
01	SOH	001	01	SOH	CTRL-A	40	Space	040	20	Space	Space
02	STX	002	02	STX	CTRL-B	4B	.	056	2E	.	.
03	ETX	003	03	ETX	CTRL-C	4C	<	074	3C	<	<
05	HT	011	09	HT	CTRL-I	4D	(	050	28	(	(
07	DEL	177	7F	DEL	RUBOUT	4E	+	053	2B	+	+
0B	VT	013	0B	VT	CTRL-K	4F	|	174	7C	|	NONE
0C	FF	014	0C	FF	CTRL-L	50	&	046	26	&	&
0D	CR	015	0D	CR	RETURN	5A	!	041	21	!	!
0E	SO	016	0E	SO	CTRL-N	5B	$	044	24	$	$
0F	SI	017	0F	SI	CTRL-O	5C	*	052	2A	*	*
10	DLE	020	10	DLE	CTRL-P	5D	)	051	29	)	)
11	DC1	021	11	DC1	CTRL-Q	5E	;	073	3B	;	;
12	DC2	022	12	DC2	CTRL-R	5F	¬	176	7E	Tilde	NONE
13	DC3	023	13	DC3	CTRL-S	60	-	055	2D	-	-
15	LF	012	0A	LF	LINE FEED	61	/	057	2F	/	/
16	BS	010	08	BS	CTRL-H	6B	,	054	2C	,	,
18	CAN	030	18	CAN	CTRL-X	6C	%	045	25	%	%
19	EM	031	19	EM	CTRL-Y	6D	_	137	5F	_	SHIFT-O
1C	IFS	034	1C	FS	CT-SFT-L	6E	>	076	3E	>	>
1D	IGS	035	1D	GS	CT-SFT-M	6F	?	077	3F	?	?
1E	IRS	036	1E	RS	CT-SFT-N	7A	:	072	3A	:	:
1F	IUS	037	1F	US	CT-SFT-O	7B	#	043	23	#	#
25	LF	012	0A	LF	LINE FEED	7C	@	100	40	@	@
26	ETB	027	17	ETB	CTRL-W	7D	'	047	27	'	'
27	ESC	033	1B	ESC	CT-SFT-K	7E	=	075	3D	=	=
2D	ENQ	005	05	ENQ	CTRL-E	7F	"	042	22	"	"
2E	ACK	006	06	ACK	CTRL-F	81	a	141	61	a	NONE
2F	BEL	007	07	BEL	CTRL-G	82	b	142	62	b	NONE
32	SYN	026	16	SYN	CTRL-V	83	c	143	63	c	NONE
37	EOT	004	04	EOT	CTRL-D	84	d	144	64	d	NONE
3C	DC4	024	14	DC4	CTRL-T	85	e	145	65	e	NONE
3D	NAK	025	15	NAK	CTRL-U						

EBCDIC/USASCII Translation Table

EBCDIC (8-bit)		USASCII (7-bit)				EBCDIC (8-bit)		USASCII (7-bit)			
Hex	Name	Oct	Hex	Name	TTY	Hex	Name	Oct	Hex	Name	TTY
86	f	146	66	f	NONE	C5	E	105	45	E	E
87	g	147	67	g	NONE	C6	F	106	46	F	F
88	h	150	68	h	NONE	C7	G	107	47	G	G
89	i	151	69	i	NONE	C8	H	110	48	H	H
8B	{	173	7B	{	NONE	C9	I	111	49	I	I
91	j	152	6A	j	NONE	D1	J	112	4A	J	J
92	k	153	6B	k	NONE	D2	K	113	4B	K	K
93	l	154	6C	l	NONE	D3	L	114	4C	L	L
94	m	155	6D	m	NONE	D4	M	115	4D	M	M
95	n	156	6E	n	NONE	D5	N	116	4E	N	N
96	o	157	6F	o	NONE	D6	O	117	4F	O	O
97	p	160	70	p	NONE	D7	P	120	50	P	P
98	q	161	71	q	NONE	D8	Q	121	51	Q	Q
99	r	162	72	r	NONE	D9	R	122	52	R	R
9A	NONE	140	60	Grave	NONE	E2	S	123	53	S	S
9B	}	175	7D	}	ALT MODE	E3	T	124	54	T	T
A2	s	163	73	s	NONE	E4	U	125	55	U	U
A3	t	164	74	t	NONE	E5	V	126	56	V	V
A4	u	165	75	u	NONE	E6	W	127	57	W	W
A5	v	166	76	v	NONE	E7	X	130	58	X	X
A6	w	167	77	w	NONE	E8	Y	131	59	Y	Y
A7	x	170	78	x	NONE	E9	Z	132	5A	Z	Z
A8	y	171	79	y	NONE	F0	0	060	30	0	0
A9	z	172	7A	z	NONE	F1	1	061	31	1	1
AA	NONE	136	5E	Carat	SHIFT-N	F2	2	062	32	2	2
AD	[	133	5D	[	SHIFT-K	F3	3	063	33	3	3
BA	NONE	134	5E	Bkslh	SHIFT-L	F4	4	064	34	4	4
BD	]	135	5F	]	SHIFT-M	F5	5	065	35	5	5
C1	A	101	41	A	A	F6	6	066	36	6	6
C2	B	102	42	B	B	F7	7	067	37	7	7
C3	C	103	43	C	C	F8	8	070	38	8	8
C4	D	104	44	D	D	F9	9	071	39	9	9

October 1976

EDIT

SUBROUTINE DESCRIPTION

Purpose: To call the 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 CIS 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 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 not used (must be fullword integer '-1' or zero parameter pointer).

par_12 is not used (must be fullword integer '-1' or zero parameter pointer).

par_13 editor control switches 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 initial EDIT command using par_6 and par_7
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)

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 fullword current line number.
par_16 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

Return codes:

0 normal return, editor unloaded
4 normal return, editor not unloaded
8 error return, editor not unloaded
12 error return, editor bug

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
```

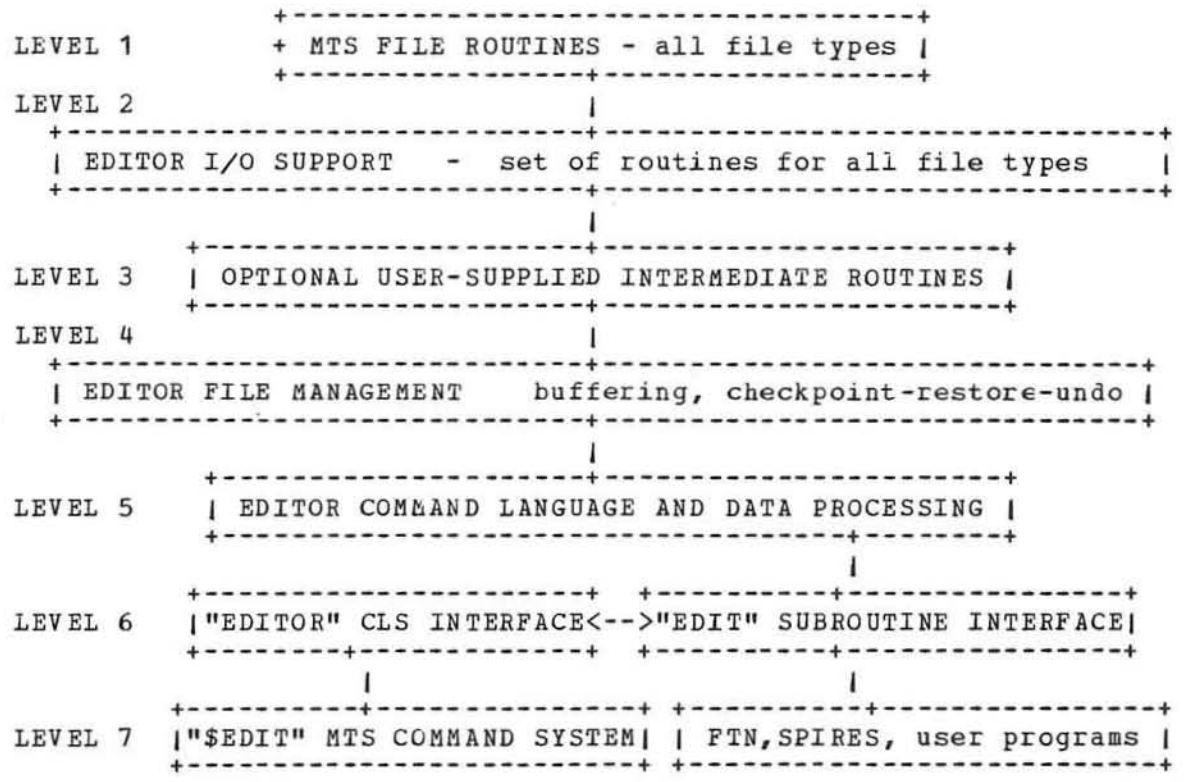
Special Features:

The remainder of this subroutine description provides information on special features of the EDIT subroutine that are of interest to system programmers; knowledge of these special features is not required to call EDIT in the manner described above.

Normal editing occurs when par_3 points to a fullword '-1'. To use the special features described here, par_3 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-



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 editor temporarily returns to caller and file could be modified invalidating the current line.

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 forwared 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.

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 '12' - 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

The above routines are available in the resident system through LCSYMBOL.

October 1976

EMPTY

SUBROUTINE DESCRIPTION

Purpose: To empty a file without destroying it.

Location: Resident System

Calling Sequence:

```
Assembly: L     0,fdub
          CALL  EMPTY
```

Parameters:

GR0 contains the location of a FDUB-pointer (as returned by GETFD).

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.

Note: FORTRAN programs should call the EMPTYF subroutine.

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.

October 1976

October 1976

EMPTYF

SUBROUTINE DESCRIPTION

Purpose: To empty a file without destroying it.

Location: Resident System

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 19), or
(c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

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

Return Codes:

0 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.

Examples:

Assembly:
```
        CALL EMPTYF,(UNIT)
          .
          .
UNIT    DC  CL8'SCARDS'
```

FORTRAN:
```
        CALL EMPTYF('SCARDS  ')
```

These examples will empty the file attached to SCARDS.

October 1976

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.

Description: A call to this subroutine returns control to MTS or debug command mode. If the return is made to MTS command mode, the comment "ERROR RETURN" is printed. In batch mode, a dump is automatically given if $ERRORDUMP or $SET ERRORDUMP=ON was specified in the appropriate mode.

Execution of the suspended program may be restarted from the point of suspension by the $RESTART command.

160 ERROR

October 1976

E7090, D7090, E7090P, D7090P

SUBROUTINE DESCRIPTION

Purpose: To allow users to convert IBM 7090 (or 7094) internal floating-point numbers to one of the two types of internal floating-point representations available on the System/370.

Location: *LIBRARY

Calling Sequences:

Assembly:
```
CALL E7090,(input,output)
CALL D7090,(input,output)
CALL E7090P,(input,output)
CALL D7090P,(input,output)
```

FORTRAN:
```
CALL E7090(input,output)
CALL D7090(input,output)
CALL E7090P(input,output)
CALL D7090P(input,output)
```

Parameters:

input is the region containing the 7090 floating-point number (either twelve or six bytes depending upon the entry used).

output is the region where the 370 floating-point number will be placed (either four or eight bytes depending upon the entry used).

Return Codes:

0 Conversion was successful.
4 Parameter list was not fullword-aligned.

Description: E7090 and D7090 expect the input to be twelve bytes long. The low-order three bits of each byte are taken as one octal digit. The sign of the number is assumed to be the first bit of the first octal digit. E7090P and D7090P assume a six-byte input region. The low-order six bits of each byte are taken as two octal digits. The first bit of the first octal digit is taken as the sign. D7090 and D7090P produce 370 "long" (8-byte) floating-point numbers. E7090 and E7090P produce 370 "short" (4-byte) floating-point numbers. Since the mantissa of a single-word floating-point number in the 370 contains only 24 bits, and the mantissa in a 7090 floating-point word contains 27 bits, rounding (if any) is done for the E-type conversions.

Examples: Assembly:

```
          CALL E7090,(REGION,OUTREG)
          .
          .
OUTREG    DS   E
REGION    DS   XL12
```

FORTRAN:

```
DIMENSION REGION(3)
REAL*4 OUTREG
  .
  .
CALL E7090(REGION,OUTREG)
```

The above two examples convert the 7090 floating-point number in the location REGION to a "short" (4-byte) 370 floating-point number.

Assembly:

```
          CALL D7090P,(REGION,OUTREG)
          .
          .
REGION    DS   XL6
OUTREG    DS   D
```

FORTRAN:

```
INTEGER*2 REGION(3)
REAL*8 OUTREG
  .
  .
CALL D7090P(REGION,OUTREG)
```

The above two examples convert the 7090 floating-point number in the location REGION to a "long" (8-byte) 370 floating-point number.

October 1976

FNAMETRT

SUBROUTINE DESCRIPTION

Purpose: A 256-byte translate table to check the legality of a file name.

Location: Resident System

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:

1 - legal file name with legal terminator.
2 - legal file name except for the CREATE subroutine.
3 - illegal file name.

The condition code is set to zero if the result is a legal file name without a legal terminator.

The following characters terminate a file name:

blank (+ , @ X'FF'

The following characters are illegal in a file name.

; :) = ' " ? &

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
```

The above example tests for the legality of the file name contained in FNAME.

October 1976

FREAD

SUBROUTINE DESCRIPTION

Purpose: To provide a free format input facility, especially for FORTRAN programs.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL FREAD(unit,type,list,type,list,...)

Assembly: CALL FREAD,(unit,type,list,type,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 19),
(c) the character string 'GUSER', 'SCARDS', '*', or
(d) a user buffer (generally an array).
This parameter indicates where input is to be read from.

type 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. A type string consists of a sequence of type codes separated by commas. The valid type codes are given below.

list is a list of variable or array names, separated by commas, into which the data values are to be placed. 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.

VL is a parameter to the CALL macro which signifies that the calling sequence has a variable number of parameters.

Values Returned:

GR0 and FR0 contain the number of fields successfully processed. This allows FREAD to be called as either a REAL or INTEGER function.

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 list parameter. The type of data item to be read into each variable location is determined by the type codes in the type parameter. There is a one-to-one correspondence between type codes and variable names in the list parameter.

FREAD will also recognize special calls which result in the setting or resetting of various switches which control subsequent FREAD actions. A special call is recognized by a unit number of -1 or -2. For further information on the FREAD subroutine, see the section "FREAD: Free Format Input Subroutine" in MTS Volume 6.

Examples: FORTRAN:

```
CALL FREAD('SCARDS','I:',J)
```

The above example reads an integer from SCARDS and places its value into the variable J.

```
CALL FREAD(5,'2I:',I,J)
```

The above example reads two integers from logical I/O unit 5 and places the values into the variables I and J.

```
CALL FREAD(9,'R VECTOR',VEC,13)
```

The above example reads 13 real numbers from logical I/O unit 9 into the array VEC.

October 1976

FREEFD

SUBROUTINE DESCRIPTION

Purpose: To free a file or device acquired by the GETFD subroutine.

Location: Resident System

Calling Sequences:

Assembly:
```
L     0,fdub
CALL FREEFD
```

Parameters:

GR0 should contain a FDUB-pointer (such as returned by GETFD or GDINFO).

Return Codes:

0 Successful return.
4 GR0 does not contain a legal FDUB-pointer.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine.

October 1976

FREESPAC

SUBROUTINE DESCRIPTION

Purpose: To release storage acquired by the GETSPACE subroutine.

Location: Resident System

Alt. Entry: FREESP

Calling Sequences:

Assembly:
```
L      0,len
L      1,loc
CALL   FREESPAC
```

or

```
FREESPAC loc[,LNG=len][,EXIT=err]
```

Parameters:

GR0 If zero, the entire region 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.

GR1 contains 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.

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+LEN-1) is not completely within a region originally allocated by GETSPACE.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine and giving FREESP as the entry point.

The complete description for using the FREESPAC macro is given in MTS Volume 14.

Examples: Assembly:

```
        SR    0,0
        L     1,LOC
        CALL FREESPAC

        FREESPAC LOC
```

The above two 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.

October 1976

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).

October 1976

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 19), 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 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 Records skipped successfully.
4 End-of-file encountered.
8 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.

Note: For both line and sequential files, a current (line or read) pointer is maintained with each FDUB (file or device usage block). Forward spacing or backspacing begins from the current pointer. See Appendix B of the section "Files and Devices" in MTS Volume 1 for details concerning how this current pointer is updated as a result of various I/O operations.

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,SKIPCT
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.

October 1976

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.

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.

Examples:

```
CALL FTNCMD('ASSIGN 7=*PUNCH*',16)
```

The above example assigns logical I/O unit 7 to *PUNCH*.

```
CALL FTNCMD('SET UVCHECK=OFF',15)
```

The above example suppresses the FORTRAN I/O library checking for undefined variables.

October 1976

October 1976

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), or
  (b) an 8-character logical I/O unit name left-justified with trailing blanks (e.g., SCARDS, SPRINT, 0 through 19, etc.).
- region — is a 44-byte array (11 fullwords) in which the information is returned.
- rc4 — is the statement label to transfer to if a return code of 4 occurs.

Return Codes:

0 Successful return.
4 Error. See the GDINFO subroutine description for the possible error conditions.

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.

Example: FORTRAN:

```
      INTEGER*4 SPRINT(2)/'SPRI','NT  '/,REG(11)
        .
        .
      CALL GDINF(SPRINT,REG,&99)
        .
        .
   99 WRITE(6,199)
  199 FORMAT(' SPRINT IS NOT ASSIGNED')
        .
        .
```

This example calls GDINF to obtain information about the file or device attached to SPRINT.

October 1976

GDINFO

SUBROUTINE DESCRIPTION

Purpose: To obtain information about a file or device.

Location: Resident System

Calling Sequence:

Assembly: (a)
```
L     0,fdub
SR    1,1
CALL GDINFO
```

(b)
```
LM    0,1,lname
CALL GDINFO
```

Parameters:

(a) GR0 contains a FDUB-pointer (such as returned by GETFD) or an integer logical I/O unit number (0 through 19), or

(b) GR0 and GR1 contain a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

Return Codes:

0 Successful return. GR1 contains the address of an information region (see below).
4 Error return. If (a) call, the FDUB-pointer was illegal or, if a logical I/O unit number was given, there was no file or device attached to that unit. If (b) call, either the name given was not a legitimate logical I/O unit name or else there was no file or device attached to that unit.
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

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 file, or access not allowed
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
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	- OPER
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 12 - FDUBTYPE field:
              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 13 - type index:
              0 = unit record
              1 = magnetic tape
              2 = terminal
              3 = file
              4 = dummy
              5 = paper tape
              6 = operator's console
              7 = test
              8 to 255 reserved for future expansion
        Byte 14 - 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 15 - unused

WORD 5: I/O modifiers given with FDname
        (see the "I/O Modifiers" section)

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 mes-
         sage), or zero
```

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.

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).

Description: The information returned by GDINFO is described by the dsect given on the following page (from the file *GDINFODSECT).

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.

```
*
*         DSECT FOR INFORMATION RETURNED BY GDINFO
*

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
GDMSINK   EQU   2
GDPUNCH   EQU   3
GDSOURCE  EQU   4
GDSINK    EQU   5
GDAFD     EQU   6
GDMOUNTD  EQU   7
GDDTYP    DS    X           DEVICE TYPE
GDUNIREC  EQU   0
GDMAGTAP  EQU   1
GDTERM    EQU   2
GDFILE    EQU   3
GDDUMMY   EQU   4
GDPAPTAP  EQU   5
GDOPER    EQU   6
GDTEST    EQU   7
GDSWS     DS    X           SWITCHES
GDOUTOK   EQU   X'80'
GDINOK    EQU   X'40'
GDINDXOK  EQU   X'20'
GDREWOK   EQU   X'10'
GDEXINCR  EQU   X'08'
GDDEFLT   EQU   X'04'       ON IF DEFAULTED
          DS    X           FOR EXPANSION
GDMODS    DS    XL4         MODIFIERS
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           LOCATION OF EXTERNAL NAME
GDERMSG   DS    A           LOCATION OF LAST ERROR MSG
```

October 1976

GDINFO2

SUBROUTINE DESCRIPTION

Purpose: To get information about a file or device.

Location: Resident System

Alt. Entry: GDINF2

Calling Sequence:

Assembly: (a)
```
L     0,fdub
SR    1,1
CALL GDINFO2
```

(b)
```
LM    0,1,lname
CALL GDINFO2
```

Description: This subroutine is exactly the same as the GDINFO subroutine with the following exceptions:

(1) The file is not opened. This means that file buffers are not allocated and file use and reference counts are not updated.
(2) The input and output lengths may be -1 to indicate that they are unknown (because of the above exception).

October 1976

GDINFO3

SUBROUTINE DESCRIPTION

Purpose: To get information about a file or device.

Location: Resident System

Alt. Entry: GDINF3

Calling Sequence:

Assembly: (a)
```
L     0,fdub
SR    1,1
CALL GDINFO3
```

(b)
```
LM    0,1,lname
CALL GDINFO3
```

Description: This subroutine is exactly the same as the GDINFO subroutine with the following exceptions:

(1) The file is opened, but not locked.
(2) The input and/or output record lengths may be -1 to indicate that they are unknown (because of the above exception).

October 1976

GETFD

SUBROUTINE DESCRIPTION

Purpose: To obtain a file or device.

Location: Resident System

Calling Sequence:

Assembly:
```
LA    1,FDname
CALL GETFD
```

Parameters:

GR1 contains 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.

Return Codes:

0 Successful return.
4 Illegal device name.
8 Device is busy.
12 Device is not operational.

Values Returned:

GR0 contains the FDUB-pointer if a successful return is made.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine.

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. 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.

Example: Assembly:

```
        LA    1,FNAME
        CALL  GETFD
         .
FNAME   DC    C'DATAFILE '
```

The above example calls GETFD to obtain a FDUB-pointer for the file DATAFILE.

October 1976

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 19), 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 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

0 Line number returned successfully.
4 The file is empty.
8 Hardware error or software inconsistency encountered.
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.

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.

October 1976

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,aexit)

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.

aexit — 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.

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 aexit 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.

Example:

```
Assembly:          CALL GETIME,(ONE,TIMLEFT,AEXIT)
                    .
                    .
           ONE      DC  F'1'
           TIMLEFT  DS  FL8
           AEXIT    DC  A(EXIT)
           EXIT     DS  19F
```

This example returns the time remaining for the interrupt with the identifier "1" and the exit region "EXIT". The value is returned in "TIMLEFT".

October 1976

GETSPACE

SUBROUTINE DESCRIPTION

Purpose: To acquire storage.

Location: Resident System

Alt. Entry: GETSPA

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]
```

Parameters:

GR0 contains 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.
0 Storage acquired is associated with the highest level program so that it is not released until execution terminates.
29 = 1 Attach storage acquired to system level.
28 = 1 Use storage index number given in GR2.
Other bits in GR0 must be zero.

GR1 contains 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 current maximum (currently 12)], or if this space request cannot be allocated in this segment, a return is made with a return code of 4.

GR2 contains the storage index number to use if bit 28 of GR0 is 1; otherwise, GR2 is ignored.

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.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine and giving GETSPA as the entry point.

The complete description for using the GETSPACE macro is given in MTS Volume 14.

Description: See the "Virtual Memory Management" section in MTS Volume 5 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'
```

```
        GETSPACE 256
```

The above two examples call GETSPACE to acquire 256 bytes of storage. The storage will be associated with the highest level program.

October 1976

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)

FORTRAN: CALL GFINFO(what,rtn,flag,cinfo,finfo,sinfo, ercode,errmsg,&rc4)

Parameters:

what is the location of either
(a) a file name (with a trailing blank), if flag=1,
(b) a fullword-integer 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 (e.g., SCARDS), if flag=2, or
(c) a 4-character signon ID of a catalog to be scanned, or *SYS (system file catalog), or *TMP (temporary file catalog), if flag=3.

rtn is the location of a 6-fullword integer region where the file name will be returned. If flag=1, this parameter on return will be the same as what. If flag=2, this parameter on return will be the file name associated with the FDUB-pointer or logical I/O unit. If flag=3, 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. 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=3) or if storage is being released (flag=0). 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 which specifies the type of what parameter given. If flag=0, any storage allocated by GFINFO will be released. This should be specified, for example, to release the variable-length sharing list if such was specified, or to release allocated 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.

cinfo is the location of a 16-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 16, 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 16-fullword region (array) where file information will be returned. The first two words of the region are as described for the cinfo parameter. Note that for sequential files, a maximum of 11 words of information is returned. 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 the remainder of the information. 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, permit access was not allowed. Fi-

nally, 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 POINTER IS BAD.
2	YOUR "FILE" IS NOT A FILE.
3	THE FILE DOES NOT EXIST.
4	NO FILES THIS CCID - CATALOG SCAN.
5	NO MORE FILES - CATALOG SCAN.
6	ACCESS NOT ALLOWED THIS FILE.
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.
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.

rc4 is the statement label to transfer to if the corresponding 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.

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 to 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.

Description: The information returned by GFINFO is described by the following dsects (from the file *GFINFODSECT).

```
*
*         CATALOG INFORMATION DSECT - ANY ACCESS IS
*         SUFFICIENT TO OBTAIN CATALOG INFORMATION
*
CIDSECT   DSECT
CIAL      DS  F   ARRAY LENGTH - WORDS
CIRL      DS  F   RETURN LENGTH - WORDS
CIONID    DS  F   OWNERID - EBCDIC
CIVOL     DS  2F  VOLUME NAME - 6 CHAR,TRAILING BLANKS
CIUC      DS  F   USECOUNT
CILRD     DS  F   LAST REFERENCE DATE - JULIAN DAYS
CICD      DS  F   CREATION DATE - JULIAN DAYS
CIFO      DS  F   FILE ORGANIZATION
*                 0=LINE, 1=SEQUENTIAL, 2=SEQWL
```

```
CIDT     DS  F  DEVICE TYPE
*               0=2311, 1=2314, 2=2321, 3=3330
CIFLG    DS  F
*
CIPRIV   EQU 1  PRIVILEGED PROGRAM
*
CILCD    DS  F  LAST CHANGE DATE-JULIAN DAYS
CIPKEY   DS  4F PROGRAM KEY - LEFT JUSTIFIED
*
*        FILE INFORMATION DSECT - ANY ACCESS IS
*        SUFFICIENT TO OBTAIN FILE INFORMATION
*
FIDSECT  DSECT
FIAL     DS  F  ARRAY LENGTH - WORDS
FIRL     DS  F  RETURN LENGTH - WORDS
FIFO     DS  F  FILE ORGANIZATION
*               0=LINE, 1=SEQUENTIAL, 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
FIFLN    DS  F  FIRST LINE NUMBER - INTERNAL
*               ZERO IF SEQUENTIAL OR EMPTY
FILLN    DS  F  LAST LINE NUMBER - INTERNAL
*               ZERO IF SEQUENTIAL OR EMPTY
FIMLL    DS  F  MAXIMUM LINE LENGTH
FIMXS    DS  F  MAXIMUM EXPANDABLE FILE SIZE - PAGES
*
*        IF LINE FILE, ALSO THE FOLLOWING
*
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 OF AVAILABLE SPACE
*
*        SHARING INFORMATION DSECT - SOME  ACCESS
*        REQUIRED FOR FIRST PART
*
SIDSECT  DSECT
SIAL     DS  F  ARRAY LENGTH - WORDS
SIRL     DS  F  RETURN LENGTH - WORDS
SIACC    DS  F  ACCESS OF THIS USERID/PRJNO/PKEY TO THIS
*              FILE
*
*               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
*               ADD FOR MULTIPLE ACCESSES
```

```
*
*          PERMIT ACCESS REQUIRED FOR REST
*
SIGA       DS   F  GLOBAL (OTHERS) ACCESS - SEE ABOVE
SIOA       DS   F  OWNER ACCESS - SEE ABOVE
                   128=OWNER HAS DEFAULT ACCESS
SIPTR      DS   F  POINTER TO VARIABLE-LENGTH SHARING LIST
*                  OR ZERO IF NO SHARING LIST
*
*          VARIABLE LENGTH SHARING LIST FORMATTED AS:
*          1 WORD TOTAL LENGTH (INCLUDING THIS) - WORDS
*          1 WORD USERID/PRJNO/PKEY ACCESS -SEE ABOVE
*          1 WORD USERID/PRJNO/PKEY CODE
*                 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
*          1 WORD USERID/PRJNO-EBCDIC,LEFT JUSTIFIED
*                 PADDED WITH BLANKS
*          OR
*          4 WORDS
                  PKEY-EBCDIC,LEFT JUSTIFIED
*                 PADDED WITH BLANKS
*          THUS YOU GET 4 WORDS (IF USERID/PRJNO)
*          OR 7 WORDS (IF PKEY)
*          FOR EACH SHARER (USERID/PRJNO/PKEY)
*          PERMITTED ACCESS TO THE FILE.
*          NOTE THAT FOR CODES 3 AND 4, YOU REALLY GET
*          4 WORDS (USERID/PRJNO) FOLLOWED BY
*          7 WORDS (PKEY).
*          THE ACCESS AND CODE WORDS WILL BE REPEATED
*          AND IDENTICAL FOR CODES 3 AND 4.
```

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)
          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 catalcg into wanted
FINFO   DC    F'0'       No file into wanted
SINFO   DC    F'0'       No sharing into 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 and prints them on logical I/O unit SPRINT.

```
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,FIMLI,FINE,
              COMMON /FI/ FINL,FINH,FILCNT,FIHCNT,FIMHL
              FIAL = 16
              CALL GFINFO,('DATAFILE ',RTN,1,0,FIAL,0,
                    ERCODE,ERRMSG,&RC4)
              IF(FIRL.EQ.0) GO TO 10
              WRITE(6,101) FICNS
              WRITE(6,102) FITS
              WRITE(6,103) FICPS
              CALL SYSTEM
          10  WRITE(6,104)
              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)
          104 FORMAT(' APPROPRIATE ACCESS NOT ALLOWED.')
              END
```

The above program will print the current, truncated, and copied file size in pages for the file DATAFILE.

October 1976

October 1976

GPSECT, QPSECT, FPSECT

SUBROUTINE DESCRIPTION

Purpose: To acquire, query, and release psect (dsect) storage allocations.

Location: Resident System

Calling Sequences:

Assembly:
```
L     0,id
L     1,length
CALL  GPSECT

L     0,id
CALL  QPSECT

L     0,id
CALL  FPSECT
```

Parameters:

GR0 contains an unique fullword identifier for the psect (i.e., a fixed address within the calling program could be used as such an identifier).

GR1 (GPSECT only) contains the length of the psect to be allocated.

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.

QPSECT:

0 Psect found.
4 Psect not found.
8 Not used.
12 Internal error in QPSECT.

FPSECT:

0 Psect released.
4 Psect not found.
8 Error return from FREESPAC subroutine.
12 Internal error in FPSECT.

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 length.

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.

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.

October 1976

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,(init)
CALL GRAND,(sd,amean)
```

FORTRAN:
```
CALL GRAND1(init)
x = GRAND(sd,amean)
```

Parameters:

- init is the location of the initial integer value for generating random numbers.
- 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 init 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).

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:

```
I=999
CALL GRAND1(I)
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.

October 1976

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

Calling Sequences:

Assembly:
```
LM   0,1,grgdat
CALL GRGJULDT

LM   0,3,grgtim
CALL GRGJULTM

LM   0,3,grgtim
CALL GRJLSEC
```

Parameters:

grgdat is the Gregorian date in the character form "MMxDDxYY", where "x" is any character.

grgtim is the Gregorian date and time in the character form "MMxDDxYYhhxmmxss", where "x" is any character.

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 GRGJUIDT, 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 GRJISEC.

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.

See GRJLDT, GRJLTM for S-type (e.g., FORTRAN and PL/I) interfaces.

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.

October 1976

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:
```
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:

GR0 contains the integer number of days through the given date starting with March 1, 1900, as "1" for calls on GRJLDT.

GR0 contains the integer number of minutes through the given time starting with March 1, 1900, at 00:01 as "1" for calls on GRJLTM.

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)
```

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)
```

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:

```
JULIAN=PLCALLF(GRJLDT,F1,DATE);
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');
```

The above example calls GRJLDT to convert the Gregorian date May 18, 1971 into its corresponding Julian date 26011.

```
JULIAN=PLCALLF(GRJLTM,F1,TIME);
DECLARE JULIAN FIXED BINARY(31),
        PLCALLF RETURNS(FIXED BINARY(31)),
        GRJLTM ENTRY,
        F1 FIXED BINARY(31) INITIAL(1),
        TIME CHARACTER(16);
```

The above example calls GRJLTM to convert the Gregorian date and time in the variable TIME into its corresponding Julian date and time.

October 1976

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)

REAL*8 GROSDT
date=GROSDT(grgdat,osdat)
```

PL/I:
```
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.

Values Returned:

FR0 contains the OS date in the character form "YYddd" with three leading 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)

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:

```
CALL PLCALL(GROSDT,F2,'05-18-71',OSDAT);
DECLARE GROSDT ENTRY,
        OSDAT CHARACTER(8);
        F2 FIXED BINARY(31) INITIAL(2),

UNSPEC(OSDAT)=UNSPEC(PLCALLD(GROSDT,F2,GRDAT,
        DUMMY));
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);
```

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.

October 1976

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: CALL PLCALL(GTDJMS,f2,grgtim,jms);

Parameters:

grgtim is the 16-byte (REAL*8(2) or CHARACTER(16)) Gregorian time and date in the character form "hhxmmxssMMxDDxYY", where "x" is any character.

f2 is a fullword (FIXED BINARY(31)) containing the integer 2.

jms 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:

```
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).

October 1976

October 1976

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).

October 1976

October 1976

GUINFO, CUINFO

SUBROUTINE DESCRIPTION

Purpose: To allow the user to obtain information about his status and his 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)

CALL CUINFO(item,loc,&rc4,&rc8,&rc12)

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.

Return Codes:

0 Successful return.
4 Error return. Item number too large.
8 Error return. Item name not in the list.
12 Error return. Illegal to change item (CUINFO only).
16 Error return. Illegal parameter address.

Description: The names given in the table below correspond to items of information from the system. All of these items can be obtained by GUINFO, but only a subset of these items can be changed by CUINFO. Each item may be referred to by its name or by its index number. The type of each item is the internal format in which it is kept in the system. The length of each item is the size of the region needed to

store the item (GUINFO) or fetch the new value of the item (CUINFO). The lengths corresponding to the different types are given below.

The items which may changed by CUINFO are indicated by an asterisk next to their item index number.

Types and Lengths of Items

Internal Type	Length Returned
Bit	4 Bytes (INTEGER*4) value of 0 or 1
Byte	4 Bytes (REAL*4) left-justified with 3 trailing blanks.
Halfword	4 Bytes (INTEGER*4)
Fullword	4 Bytes (INTEGER*4)
Dblword	8 Bytes (REAL*8)
8 Bytes	8 Bytes
4 Words	16 Bytes
6 Words	24 Bytes

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'PREFIXC'
CLOC      DC   CL4'%'
```

FORTRAN:

```
INTEGER*4 GLOC,CLOC
DATA CLOC/'%   '/
CALL GUINFO('BATCHMD ',GLOC)
IF(GLOC.EQ.1) GO TO 10
CALL CUINFO('PREFIXC ',CLOC)
```

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 % by calling CUINFO. This could also be accomplished by calling the CANREPLY and SETPFX subroutines.

Table of Items Arranged by Index

Index	Name	Type	Description
1*	LNS	Byte	Line number separator character (default is ",") ($SET LNS=c)
2	SIGNONID	Fullword	Current signon ID
3*	PREFIXC	Byte	Current prefix character as set by SETPFX
4	S8NBR	Dblword	S8-number of job in characters (batch only)
5*	FILECHAR	Byte	File name character (default is "#") ($SET FILECHAR=c)
6	STORUSED	Fullword	CPU storage integral to STORCPUT[1]
7*	SCRFCHAR	Byte	Scratch file character (default is "-") ($SET SCRFCHAR=c)
8	CURRSTOR	Fullword	Current number of half-pages of VM storage
9*	CONTCHAR	Byte	MTS command continuation character (default is "-") ($SET CONTCHAR=c)
10	BATCHMD	Bit	Batch (1) or conversational (0) mode
11*	ICFBIT	Bit	1 -> $SET IC=OFF (default is ON)
12	LOCSW	Bit	1 -> Local time estimate active
15*	ATNBIT	Bit	1 -> Attention interrupt occurred but not taken (may be set to cause an attention interrupt)
16	PROJNO	Fullword	Project (charge) number in characters
17*	UCBIT	Bit	1 -> $SET CASE=UC (default is LC)
18	MAXDISK	Halfword	Maximum number of disk pages allowed for ID
19*	NXTSEGSW	Bit	1 -> Skip to next set of command cards (batch only) (may be set to skip unread data cards)
20	MAXTERM	Fullword	Maximum terminal time allowed for ID (seconds)
21*	PRNTCDSW	Bit	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	Bit	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
24	CURRDISK	Halfword	Number of pages of disk space in current use
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	$SET ERRORDUMP={OFF|ON|FULL} (0|1|2) (default OFF)
28	CUMCPUTM	Fullword	Cum. CPU time for ID (milliseconds) (excluding active jobs)
29	CUMREAD	Fullword	Cum. number of cards read for ID (excluding active jobs)
30	CUMCORE	Fullword	Cum. storage integral over CPU time for ID (excluding active jobs)[2]
31	NRREAD	Fullword	Number of cards read for current job
32	CUMMONY	Fullword	Cum. charge used for ID (cents*100) (excluding active jobs)
33*	LDROPT	Byte	Loader options switch[10]
35*	SHFSEP	Byte	Shared-file separator character (default is ":") ($SET SHFSEP=c)
36	NRDISKF	Halfword	Number of disk files existing for ID
37*	RF	Fullword	$SET RF=xxxxxx (default is 0)

38	NRSIGS	Halfword	Cum. number of signons for ID (excluding active jobs)
39*	DEVCHAR	Byte	Device name character (default is ">") ($SET DEVCHAR=c)
40	NRBATCH	Halfword	Cum. number of batch jobs for ID (excluding active jobs)
41*	NUMBER	Bit	1 -> Automatic numbering active ($NUMBER)
42	CUMLINES	Fullword	Cum. number of lines printed for ID (excluding active jobs)
43*	LIBROFF	Bit	1 -> $SET LIBR=OFF (default is ON)
44	CUMPAGES	Fullword	Cum. number of pages printed for ID (excluding active jobs)
45*	AFDECHO	Bit	1 -> $SET AFDECHO=ON (default is OFF)
46	CUMPUNCH	Fullword	Cum. number of cards punched for ID (excluding active jobs)
47*	SYMTAB	Bit	1 -> $SET SYMTAB=ON (default is ON)
48	STORUSEE	Fullword	Elapsed storage integral to STORELT[1]
49*	ECHOOFF	Bit	1 -> $SET ECHO=OFF (default is ON)
51*	ATTNOFF	Bit	1 -> Stack attention interrupts (may be set to inhibit attention interrupts; pending interrupt may be taken on call to system subroutine)
54	EXPTIME	Fullword	ID expiration time and date[3]
55*	SIGSHORT	Fullword	$SIG {LONG\|SHORT\|$} (0|1|2) (default is LONG)
56	SOBCDTM	4 Words	Sign-on time and date in characters
57*	PFXOFF	Bit	1 -> $SET PFX=OFF (default is ON)
58	STORCPUT	Fullword	Current base for CPU storage integral[4]
59*	SEQCOFF	Bit	1 -> $SET SEQFCHK=OFF (default is ON)
60	NRCREATE	Halfword	Number of files created during current job
61*	PGNTTRP	Dblword	PGNTTRP exit subroutine address (1st word) and save area location (2nd word)
62	NRDESTRY	Halfword	Number of files destroyed during current job
63	NRLINES	Fullword	Number of lines printed for current job
64	SOCPUTP	Fullword	Problem state CPU time at signon[5]
65	NRPAGES	Fullword	Number of pages printed for current job
66	SOCPUTC	Fullword	Supervisor state CPU time at signon[5]
67	NRPUNCH	Fullword	Number of cards punched for current job
68	SOELT	Dblword	Time of day at signon[6]
69*	ATTNTRP	Dblword	ATTNTRP exit subroutine address (1st word) and save area location (2nd word)
70	STORELT	Fullword	Current base for elapsed storage integral[4]
71*	AFDNBR	Fullword	Next line number for *AFD* ($NUMBER)
72	SOPTOD	4 Words	Time and date for header page for batch output (characters)
73*	AFDINC	Fullword	Line number increment for *AFD* ($NUMBER)
74	ANSBACK	6 Words	Answerback code (characters)
75*	SETIOERR	Fullword	SETIOERR exit subroutine address
76	CUMDISK	Fullword	Cum. disk file storage integral to DISKTIME which has been charged for (page hours)
77*	ENDFILSW	Fullword	$SET ENDFILE={NEVER|OFF|ON} (0|1|2) (default OFF)
78	GLOBCPUT	Fullword	CPU time remaining in global time limit (from GLOBTTN)[5]
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	Bit	1 -> $SET TDR=ON (default is OFF)
86	LOCCPUT	Fullword	CPU time remaining in local time limit (from LOCTTN)[5]
87	MNETTIME	Fullword	Outbound MERIT time for this job (seconds)
88	LOCPGS	Fullword	Local page estimate
89*	CROUTE	Fullword	Default batch station for punched output (characters) ($SET CROUTE=rmid)
90	LOCPCH	Fullword	Local card estimate
91*	PROUTE	Fullword	Default batch station for printed output (characters) ($SET PROUTE=rmid)
92	LOCPTM	Fullword	Local plot time estimate (seconds)
93*	PRINT	Halfword	Print train specification (PN, TN, or binary 0 is first byte if ANY)
94	GLOBTTN	Fullword	Base for global time limit[5]
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
96	LOCTTN	Fullword	Base for local time limit[5]
98	TASKNBR	Halfword	Task (job) number
100	TASKTYPE	Halfword	Task type code[8]
102	BTCHMORE	Bit	1 -> MORE specified on non-HASP batch job
104	HASPJOB	Bit	1 -> HASP batch job
106	MAXCELL	Halfword	Maximum datacell pages allowed for ID
108	MAXPLOT	Fullword	Maximum plot time allowed for ID (seconds)
110	LSTRESET	Fullword	Last time cum. totals for this ID were reset[3]
112	DISKTIME	Fullword	Last time disk storage integral updated[3]
114	CELLTIME	Fullword	Last time datacell storage integral updated[3]
116	CURRCELL	Halfword	Number of pages of datacell files in current use
118	CUMCOREW	Fullword	Cum. storage integral over wait time for this ID (excluding active jobs)[2]
122	CUMPLOT	Fullword	Cum. plot time for ID (seconds) (excluding active jobs)
124	NRCELLF	Fullword	Number of datacell files existing for ID
126	CUMCELL	Fullword	Cum. datacell file storage integral to CELLTIME which has been charged for (page hours)
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
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
144	SCRFNAME	Dblword	Internal scratch file prefix
146	SCRFDISK	Halfword	Number of pages of disk scratch files for current job
148	SCRFCELL	Halfword	Number of pages of datacell scratch files for

			current job
150	SODRMRDS	Fullword	Number of drum reads at signon
152	LASTSOT	4 Words	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)
158	CUMPTLEN	Fullword	Cum. paper tape punched for ID (inches) (excluding active jobs)
160	BILLCLAS	Halfword	Billing class (0=University 1=Industrial)
162	SCRDSKTM	Fullword	Last time scratch disk file storage integral updated[3]
164	SCRCELTM	Fullword	Last time scratch datacell file storage integral updated[3]
166	SCRDSUSE	Fullword	Scratch disk file storage integral to SCRDSKTM[7]
167*	SIGFATTN	Bit	1 -> $SET SIGFILEATTN=OFF (default is ON)
168	SCRCLUSE	Fullword	Scratch datacell file storage integral tc SCRCELTM[7]
169*	TERSE	Bit	1 -> $SET TERSE=ON (default is OFF)
170	CUDRMRDS	Halfword	Current number of drum reads for current job
171*	$ON	Bit	1 -> $SET $=ON (default is OFF)
172	CLSID	Halfword	Code for CLS currently in control[9]
173*	CREAFD	Bit	1 -> $SET CREAFD=ON (default is ON)
174	PCLSID	Halfword	Code for CLS that called current CLS[9]
175*	EDITAFD	Bit	1 -> $SET EDITAFD=ON (default is ON)
176	DEBUGCMD	Bit	1 -> if $DEBUG command active
177*	USMSG	Bit	1 -> $SET USMSG=ON (default is ON)
178	DEBUG	Bit	1 -> $SET DEBUG=ON (default is OFF)
179*	AUTOHOLD	Bit	1 -> $SET AUTOHOLD=ON (default is OFF)
180	LSS	Bit	1 -> if limited service state active
181*	TRIMBIT	Bit	1 -> $SET TRIM=ON (default is ON)
182	MAXSIG	Halfword	Max. number of concurrent signons allowed for ID
183*	EFLUEM	Dblword	Elementary Function Library user error mcnitor address
184	CURSIG	Halfword	Number of times this ID currently signed on
185*	CMDSKP	Bit	1 -> $SET CMDSKIP=ON (default is OFF)
186	UNCHDISK	Fullword	Disk space to DISKTIME not yet charged for[7]
187*	PRMAPOFF	Bit	1 -> $SET PRMAP=OFF (default is ON)
188	UNCHCELL	Fullword	Datacell space to CELLTIME not yet charged for[7]
189*	PDMAPOFF	Bit	1 -> $SET PDMAP=OFF (default is ON)
190	MAXMNET	Fullword	Maximum outbound MERIT time (seconds)
191*	UXREF	Bit	1 -> $SET UXREF=ON (default is OFF)
192	CUMMNET	Fullword	Cum. outbound MERIT for this ID excluding active jobs (seconds)
193*	XREF	Bit	1 -> $SET XREF=ON (default is OFF)
194	MXMNETBT	Bit	1 -> Ignore maximum MNET time (item 190)
195*	NO*LIB	Bit	1 -> $SET *LIBRARY=OFF (default is ON)
196	MXPLOTBT	Bit	1 -> Ignore maximum plot time (item 108)
197*	MAPDOTS	Bit	1 -> $SET MAPDOTS=ON (default is ON)
198	RATENBR	Fullword	Number determining rate set in use
199*	NOERRMAP	Bit	1 -> $SET ERRMAP=OFF (default is ON)
226	INSIGFIL	Bit	1 -> currently processing sigfile

227	PLOTPAPR	Fullword	Plotter paper used for current job (.01 inches)
228	TOFFSET	8 Byte	Offset (microseconds times 4096) to be added to GMT to get local time
229	PLOTPENC	Fullword	Plotter pen changes for current job
230	TIMEFDGE	8 Byte	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)
233*	NOSDS	Bit	1 -> $SET SDSMSG=OFF (default is ON)
234	CUMPLPEN	Fullword	Cum. plot pen changes for ID (excluding active jobs)
236	PKEY	4 Words	Program key under which calling program is running
237*	RCPRINT	Byte	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS} (0|1|2|3)
238	RUNONLY	Bit	1 -> a "run only" program is loaded (from a file to which the user has no access)
239	LASTEXEC	Fullword	Return code of last program executed.
240	SYSOLOAD	Byte	System overload indicators[11]
242	PRIO	Byte	Priority of job[12]

Table of Items Arranged by Name

Index	Name	Type	Description
45*	AFDECHO	Bit	1 -> $SET AFDECHO=ON (default is OFF)
73*	AFDINC	Fullword	Line number increment for *AFD* ($NUMBER)
71*	AFDNBR	Fullword	Next line number for *AFD* ($NUMBER)
74	ANSBACK	6 Words	Answerback code (characters)
15*	ATNBIT	Bit	1 -> Attention interrupt occurred but not taken (may be set to cause an attention interrupt)
51*	ATTNOFF	Bit	1 -> Stack attention interrupts (may be set to inhibit attention interrupts; pending interrupt may be taken on call to system subroutine)
69*	ATTNTRP	Dblword	ATTNTRP exit subroutine address (1st word) and save area location (2nd word)
179*	AUTOHOLD	Bit	1 -> $SET AUTOHOLD=ON (default is OFF)
10	BATCHMD	Bit	Batch (1) or conversational (0) mode
160	BILLCLAS	Halfword	Billing class (0=University 1=Industrial)
102	BTCHMORE	Bit	1 -> MORE specified on non-HASP batch job
114	CELLTIME	Fullword	Last time datacell storage integral updated[3]
172	CLSID	Halfword	Code for CLS currently in control[9]
185*	CMDSKP	Bit	1 -> $SET CMDSKIP=ON (default is OFF)
9*	CONTCHAR	Byte	MTS command continuation character (default is "-") ($SET CONTCHAR=c)
128	COPIES	Fullword	Number of copies of printed output requested on $SIGNON command (batch)
173*	CREAFD	Bit	1 -> $SET CREAFD=ON (default is ON)
89*	CROUTE	Fullword	Default batch station for punched output (characters) ($SET CROUTE=rmid)
170	CUDRMRDS	Halfword	Current number of drum reads for current job
126	CUMCELL	Fullword	Cum. datacell file storage integral to CELLTIME which has been charged for (page hours)
30	CUMCORE	Fullword	Cum. storage integral over CPU time for ID (excluding active jobs)[2]
118	CUMCOREW	Fullword	Cum. storage integral over wait time for this ID (excluding active jobs)[2]
28	CUMCPUTM	Fullword	Cum. CPU time for ID (milliseconds) (excluding active jobs)
76	CUMDISK	Fullword	Cum. disk file storage integral to DISKTIME which has been charged for (page hours)
26	CUMELTM	Fullword	Cum. terminal time for ID (seconds) (excluding active jobs)
42	CUMLINES	Fullword	Cum. number of lines printed for ID (excluding active jobs)
192	CUMMNET	Fullword	Cum. outbound MERIT for this ID excluding active jobs (seconds)
32	CUMMONY	Fullword	Cum. charge used for ID (cents*100) (excluding active jobs)
154	CUMMOUNT	Fullword	Cum. number of tape mounts for ID (excluding active jobs)
44	CUMPAGES	Fullword	Cum. number of pages printed for ID (excluding active jobs)
122	CUMPLOT	Fullword	Cum. plot time for ID (seconds) (excluding

			active jobs)
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)
158	CUMPTLEN	Fullword	Cum. paper tape punched for ID (inches
46	CUMPUNCH	Fullword	Cum. number of cards punched for ID (excluding active jobs)
29	CUMREAD	Fullword	Cum. number of cards read for ID (excluding active jobs)
156	CUMTDRVT	Fullword	Cum. tape drive time for ID (seconds) (excluding active jobs)
116	CURRCELL	Halfword	Number of pages of datacell files in current use
24	CURRDISK	Halfword	Number of pages of disk space in current use
8	CURRSTOR	Fullword	Current number of half-pages of VM storage
184	CURSIG	Halfword	Number of times this ID currently signed on
178	DEBUG	Bit	1 -> $SET DEBUG=ON (default is OFF)
176	DEBUGCMD	Bit	1 -> if $DEBUG command active
39*	DEVCHAR	Byte	Device name character (default is ">") ($SET DEVCHAR=c)
112	DISKTIME	Fullword	Last time disk storage integral updated[3]
27*	DUMPTYPE	Fullword	$SET ERRORDUMP={OFF|ON|FULL} (0|1|2) (default OFF)
49*	ECHOOFF	Bit	1 -> $SET ECHO=OFF (default is ON)
175*	EDITAFD	Bit	1 -> $SET EDITAFD=ON (default is ON)
183*	EFLUEM	Dblword	Elementary Function Library user error monitor address
77*	ENDFILSW	Fullword	$SET ENDFILE={NEVER|OFF|ON} (0|1|2) (default OFF)
54	EXPTIME	Fullword	ID expiration time and date[3]
5*	FILECHAR	Byte	File name character (default is "#") ($SET FILECHAR=c)
78	GLOBCPUT	Fullword	CPU time remaining in global time limit (from GLOBTTN)[5]
82	GLOBPCH	Fullword	Global card estimate
80	GLOBPGS	Fullword	Global page estimate
84	GLOBPTM	Fullword	Global plot time estimate (seconds)
94	GLOBTTN	Fullword	Base for global time limit[5]
104	HASPJOB	Bit	1 -> HASP batch job
11*	ICFBIT	Bit	1 -> $SET IC=OFF (default is ON)
226	INSIGFIL	Bit	1 -> currently processing sigfile
239	LASTEXEC	Fullword	Return code of last program executed.
152	LASTSOT	4 Words	Last signon time in characters
33*	LDROPT	Byte	Loader options switch[10]
43*	LIBROFF	Bit	1 -> $SET LIBR=OFF (default is ON)
130	LINKLEVL	Fullword	Current link level (see MTS Vol. 5 Virtual Memory Management description)
1*	LNS	Byte	Line number separator character (default is ",") ($SET LNS=c)
86	LOCCPUT	Fullword	CPU time remaining in local time limit (from LOCTTN)[5]
90	LOCPCH	Fullword	Local card estimate

88	LOCPGS	Fullword	Local page estimate
92	LOCPTM	Fullword	Local plot time estimate (seconds)
12	LOCSW	Bit	1 -> Local time estimate active
96	LOCTTN	Fullword	Base for local time limit[5]
138	LODRSYMT	Fullword	Loader symbol table location
180	LSS	Bit	1 -> if limited service state active
110	LSTRESET	Fullword	Last time cum. totals for this ID were reset[3]
197*	MAPDOTS	Bit	1 -> $SET MAPDOTS=ON (default is ON)
106	MAXCELL	Halfword	Maximum datacell pages allowed for ID
18	MAXDISK	Halfword	Maximum number of disk pages allowed for ID
190	MAXMNET	Fullword	Maximum outbound MERIT time (seconds)
22	MAXMONY	Fullword	Maximum charge allowed for ID (cents*100)
108	MAXPLOT	Fullword	Maximum plot time allowed for ID (seconds)
182	MAXSIG	Halfword	Max. number of concurrent signons allowed for ID
20	MAXTERM	Fullword	Maximum terminal time allowed for ID (seconds)
87	MNETTIME	Fullword	Outbound MERIT time for this job (seconds)
194	MXMNETBT	Bit	1 -> Ignore maximum MNET time (item 190)
196	MXPLOTBT	Bit	1 -> Ignore maximum plot time (item 108)
136	MXSTRIND	Fullword	Maximum storage index number used (See MTS Vol. 5 Virtual Memory Management description)
199*	NOERRMAP	Bit	1 -> $SET ERRMAP=OFF (default is ON)
233*	NOSDS	Bit	1 -> $SET SDSMSG=OFF (default is ON)
195*	NO*LIB	Bit	1 -> $SET *LIBRARY=OFF (default is ON)
40	NRBATCH	Halfword	Cum. number of batch jobs for ID (excluding active jobs)
124	NRCELLF	Fullword	Number of datacell files existing for ID
60	NRCREATE	Halfword	Number of files created during current job
62	NRDESTRY	Halfword	Number of files destroyed during current job
36	NRDISKF	Halfword	Number of disk files existing for ID
63	NRLINES	Fullword	Number of lines printed for current job
79	NRMOUNT	Fullword	Number of tape and other mounts for current job
65	NRPAGES	Fullword	Number of pages printed for current job
67	NRPUNCH	Fullword	Number of cards punched for current job
31	NRREAD	Fullword	Number of cards read for current job
38	NRSIGS	Halfword	Cum. number of signons for ID (excluding active jobs)
41*	NUMBER	Bit	1 -> Automatic numbering active ($NUMBER)
19*	NXTSEGSW	Bit	1 -> Skip to next set of command cards (batch only) (may be set to skip unread data cards)
23*	OFFBIT	Bit	1 -> Sign off when next MTS command is read (same as QUIT subroutine)
174	PCLSID	Halfword	Code for CLS that called current CLS[9]
189*	PDMAPOFF	Bit	1 -> $SET PDMAP=OFF (default is ON)
57*	PFXOFF	Bit	1 -> $SET PFX=OFF (default is ON)
61*	PGNTTRP	Dblword	PGNTTRP exit subroutine address (1st word) and save area location (2nd word)
236	PKEY	4 Bytes	Program key under which the calling program is running
227	PLOTPAPR	Fullword	Plotter paper used for current job (.01 inches)
229	PLOTPENC	Fullword	Plotter pen changes for current job
25	PLOTTIME	Fullword	Total plot time for current job (seconds)
3*	PREFIXC	Byte	Current prefix character as set by SETPFX

93*	PRINT	Halfword	Print train specification (PN, TN, or binary 0 is first byte if ANY)
242	PRIO	Byte	Priority of job[12]
187*	PRMAPOFF	Bit	1 -> $SET PRMAP=OFF (default is ON)
21*	PRNTCDSW	Bit	1 -> Print next input line from source if not MTS command (batch only)
16	PROJNO	Fullword	Project (charge) number in characters
91*	PROUTE	Fullword	Default batch station for printed output (characters) ($SET PROUTE=rmid)
83	PTLEN	Fullword	Paper tape punched for current job (inches)
198	RATENBR	Fullword	Number determining rate set in use
237*	RCPRINT	Byte	$SET RCPRINT={NEVER|POS|NONNEG|ALWAYS} (0|1|2|3)
37*	RF	Fullword	$SET RF=xxxxxx (default is 0)
238	RUNONLY	Bit	1 -> A "run only" program is loaded (from a file to which the user has no access)
95	SCOPIES	Fullword	Number of copies of printed output requested on $SET COPIES=n command
164	SCRCELTM	Fullword	Last time scratch datacell file storage integral updated[3]
168	SCRCLUSE	Fullword	Scratch datacell file storage integral to SCRCELTM[7]
162	SCRDSKTM	Fullword	Last time scratch disk file storage integral updated[3]
166	SCRDSUSE	Fullword	Scratch disk file storage integral to SCRDSKTM[7]
148	SCRFCELL	Halfword	Number of pages of datacell scratch files for current job
7*	SCRFCHAR	Byte	Scratch file character (default is "-") ($SET SCRFCHAR=c)
146	SCRFDISK	Halfword	Number of pages of disk scratch files for current job
144	SCRFNAME	Dblword	Internal scratch file prefix
59*	SEQCOFF	Bit	1 -> $SET SEQFCHK=OFF (default is ON)
75*	SETIOERR	Fullword	SETIOERR exit subroutine address
35*	SHFSEP	Byte	Shared-file separator character (default is ":") ($SET SHFSEP=c)
167*	SIGFATTN	Bit	1 -> $SET SIGFILEATTN=OFF (default is ON)
2	SIGNONID	Fullword	Current signon ID
55*	SIGSHORT	Fullword	$SIG {LONG\|SHORT\|$} (0|1|2) (default is LONG)
56	SOBCDTM	4 Words	Sign-on time and date in characters
66	SOCPUTC	Fullword	Supervisor state CPU time at signon[5]
64	SOCPUTP	Fullword	Problem state CPU time at signon[5]
150	SODRMRDS	Fullword	Number of drum reads at signon
68	SOELT	Dblword	Time of day at signon[6]
72	SOPTOD	4 Words	Time and date for header page for batch output (characters)
231*	SPELLCOR	Fullword	$SET SPELLCOR={OFF|PROMPT|ON} (0|3|1) (default is PROMPT)
58	STORCPUT	Fullword	Current base for CPU storage integral[4]
70	STORELT	Fullword	Current base for elapsed storage integral[4]
134	STORINDX	Fullword	Current storage index number (See MTS Vol. 5 Virtual Memory Management description)
48	STORUSEE	Fullword	Elapsed storage integral to STORELT[1]

6	STORUSED	Fullword	CPU storage integral to STORCPUT[1]
47*	SYMTAB	Bit	1 -> $SET SYMTAB=ON (default is ON)
240	SYSOLOAD	Byte	System overload indicators[11]
4	S8NBR	Dblword	S8-number of job in characters (batch only)
98	TASKNBR	Halfword	Task (job) number
100	TASKTYPE	Halfword	Task type code[8]
85*	TDR	Bit	1 -> $SET TDR=ON (default is OFF)
81	TDRVT	Fullword	Tape drive time for current job (seconds)
169*	TERSE	Bit	1 -> $SET TERSE=ON (default is OFF)
230	TIMEFDGE	8 Byte	Value (microseconds times 4096) to be added to IBM time (as stored by a STCK instruction) to get time based on March 1, 1900
228	TOFFSET	8 Byte	Offset (microseconds times 4096) to be added to GMT to get local time
181*	TRIMBIT	Bit	1 -> $SET TRIM=ON (default is ON)
17*	UCBIT	Bit	1 -> $SET CASE=UC (default is LC)
188	UNCHCELL	Fullword	Datacell space to CELLTIME not yet charged for[7]
186	UNCHDISK	Fullword	Disk space to DISKTIME not yet charged for[7]
177*	USMSG	Bit	1 -> $SET USMSG=ON (default is ON)
191*	UXREF	Bit	1 -> $SET UXREF=ON (default is OFF)
193*	XREF	Bit	1 -> $SET XREF=ON (default is OFF)
171*	$ON	Bit	1 -> $SET $=ON (default is OFF)

October 1976

---

[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
    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=CLS (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)
[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

Notes:

(1) 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.

(2) 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.

(3) 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.

October 1976

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		
106	MAXCELL	160	BILLCLAS
108	MAXPLOT	182	MAXSIG
110	LSTRESET	184	CURSIG
116	CURRCELL	190	MAXMNET
118	CUMCOREW	192	CUMMNET
122	CUMPLOT	194	MXMNETBT
124	NRCELLF	196	MXPLOTBT
154	CUMMOUNT		
156	CUMTDRVT		
158	CUMPTLEN		
232	CUMPLPAP		
243	CUMPLPEN		

October 1976

October 1976

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.

The default FDname for GUSER is *MSOURCE*.

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.

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     .
```

October 1976

GUSERID

SUBROUTINE DESCRIPTION

Purpose: To obtain the current 4-character signon ID.

Location: Resident System

Alt. Entry: GETID

Calling Sequences:

Assembly: CALL GUSERID

A GR13 save area is not required for a call to this subroutine.

Values Returned:

GR1 contains the 4 character signon ID.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine and specifying GETID as the entry point, or by calling the subroutine GUINFO for the information item SIGNONID.

October 1976

October 1976

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 5.

October 1976

October 1976

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:
```
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:

- juldat is a fullword (INTEGER*4 or FIXED BINARY(31)) containing the integer number of days starting with March 1, 1900 as "1".
- 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.
- jultim 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".
- grgtim 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.
- f2 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.

FR0 and FR2 contain the Gregorian date and time in the character form "MM/DD/YYhh:mm:00" for calls on JLGRTM.

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:

```
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.

October 1976

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: CALL PLCALL(JMSGTD,f2,jms,grgtim);

CALL PLCALL(JTUGTD,f2,jtu,grgtim);

Parameters:

- jms is an 8-byte integer (INTEGER*4(2) or BIT(64)) containing the integer number of microseconds starting with March 1, 1900.
- jtu 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).
- grgtim 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.
- f2 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:
```
DECLARE JMSGTD ENTRY,
    F2 FIXED BINARY(31) INITIAL (2),
    TIME CHARACTER (16),
    JULIAN BIT(64) INITIAL
    ('0000000000001000001100001101000101110100011101110111100001001111'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.

October 1976

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:

julms is two fullwords containing the 8-byte integer number of microseconds through the given date starting with March 1, 1900.

jultu 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.

October 1976

October 1976

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.

October 1976

KEYWRD

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" (and its degenerate forms "keyword" and "value") and performing an associated program action when a specified keyword expression is found. Note: This subroutine has been superseded in capabilities by the KWSCAN subroutine that is described in this volume. The KWSCAN subroutine is recommended for use.

Location: Resident System

Calling Sequences:

Assembly: CALL KEYWRD,(len,lht,ext,text,rht)

Parameters:

The parameters are identical to the first five parameters of the KWSCAN subroutine.

Return Codes:

0 Keyword successfully processed.
4 Error occurred, or a prompting message was issued and the response "CANCEL" was given.

Description: See the description of the KWSCAN subroutine in this volume for the complete details of keyword processing. The KEYWRD subroutine performs actions similar to those of KWSCAN, but with the following differences:

(1) no spelling correction will be attempted;
(2) parenthesized or initial-substring, left-hand side expressions will not be recognized;
(3) only blanks, and commas not nested inside parentheses are recognized as delimiters. The character string indicated by text must end with a blank;
(4) at most one keyword expression is processed by the subroutine, and text is not updated to point to the end of the text scanned;
(5) all erroneous keyword expressions produce output on *MSINK* and a prompt for replacement from *MSOURCE*;
(6) no case conversion of the input is performed.

Example: The following example illustrates how keyword processing could be set up for scanning keyword expressions in the MTS $SIGNON and $DISPLAY commands.

For the $SIGNON command, the keyword options considered are the global time, page, and card estimates, the password, and the print train option. For the $DISPLAY command, the options considered are the displaying of registers and a single core location, and the hexadecimal conversion option.

The first byte in each entry of the left-hand table LHTABLE gives a displacement to an entry in the right-hand table RHTABLE. The second byte of the entry gives the execute index #1. The third byte gives the number of bytes for the left-hand side of the keyword. The last entry in LHTABLE is used for the case of a degenerate left-hand side (e.g.,$DISPLAY GR9).

The first byte in each entry of the right-hand table RHTABLE gives the entry type. The second byte is the execute number #2. The third byte is the number of bytes following which contain further "operations" for processing the entry. The last entry in RHTABLE is for the case of a degenerate right-hand side of the keyword (e.g.,$ DISPLAY GRS).

To further explain the function of RHTABLE, consider the entry TIMRHT for global time estimates. This is a type 4 entry which specifies 5 operations to be performed cn the right-hand side of the keyword. These operations are specified in units of 5 bytes each (hence a length of 25 is given), the first byte of each operation specifying the operation type. Here the first operation tests for a scale factor 'S' and, if found, multiplies the keyword by 1. The second operation tests for a scale factor 'M' and, if found, multiplies the keyword by 60 to convert minutes into seconds. The third operation tests to determine if the global time estimate is less than 36,000 seconds (10 hours). The last two operations perform a multiplication and division to convert the time estimate into CPU timer units.

The entry PGSRHT is also a type 4 entry which performs two operations on the page or card estimate. The first operation divides the estimate by 1000 to reduce it to an internal number of pages or cards and the second checks to determine if it does not exceed 99999 after the division.

The entry PRTRHT is a type 1 entry which checks the right-hand side for 'PN' or 'TN' specifying the type of print train requested.

The entry DISRHT is entry which contains both type 6 (initial substring literal) and type 5 (hex number) fields. This entry checks the right-hand side for the initial substrings 'FR' or 'GR' (e.g., $DISPLAY GR9). If neither is found, it then assumes a hex number (e.g., $DISPLAY 5002A6).

The execute table EXTABLE contains the actual instructions or subroutine calls to be made for each keyword expression successfully matched. For the $SIGNON command, the instructions in EXTABLE store information from GR1 and GR2 into specified locations (e.g., storing the global time estimate in CPU timer units into the location GLOBTIME). For the $DISPLAY command, the instructions are either for setting one switch or calling a subroutine to set two switches.

The short program given scans a $SIGNON or $DISPLAY command calling the KEYWRD subroutine for each keyword expression found in the command. The loop defines a keyword as beginning with the first nonblank character following a blank character. Note that for the case of "$SIGNON XXXX T=30S P=100 C=100 PRINT=TN", the call to KEYWRD for XXXX will cause an error return from the subroutine which will be ignored by the calling program.

```
          LA    6,81
          LA    7,TEXT-1
OUTLOOP   LA    7,1(7)
          CLI   0(7),' '
          BNE   OUTEND
INLOOP    LA    7,1(7)
          CLI   0(7),' '
          BE    INEND
          CALL  KEYWRD,(LHTLEN,LHTABLE,EXTABLE,(7),RHTABLE)
          B     OUTLOOP
INEND     BCT   6,INLOOP
          B     *+8
OUTEND    BCT   6,OUTLOOP
           .
           .
           .
TEXT      DS    80C
          DC    C' '

LHTABLE   DC    AL1(TIMRHT-RHTABLE,0,4),C'TIME'
          DC    AL1(TIMRHT-RHTABLE,0,1),C'T'
          DC    AL1(PGSRHT-RHTABLE,4,5),C'PAGES'
          DC    AL1(PGSRHT-RHTABLE,4,2),C'PP'
          DC    AL1(PGSRHT-RHTABLE,4,1),C'P'
          DC    AL1(PGSRHT-RHTABLE,8,5),C'CARDS'
          DC    AL1(PGSRHT-RHTABLE,8,1),C'C'
          DC    AL1(PWRHT-RHTABLE,12,2),C'PW'
```

```
            DC    AL1(PRTRHT-RHTABLE,16,5),C'PRINT'
            DC    AL1(NORHT-RHTABLE,22,3),C'FRS'
            DC    AL1(NORHT-RHTABLE,26,3),C'GRS'
            DC    AL1(NORHT-RHTABLE,30,3),C'HEX'
            DC    AL1(NORHT-RHTABLE,34,5),C'NOHEX'
            DC    AL1(DISRHT-RHTABLE,38,0)
LHTEND      EQU   *
LHTLEN      DC    Y(LHTEND-LHTABLE)

RHTABLE     EQU   *
TIMRHT      DC    AL1(4,0,25),C'S',AL4(1),C'M',AL4(60)
            DC    C'<',FL4'36000000',C'*',AL4(768)
            DC    C'/',AL(10)
            DC    X'FF'
PGSRHT      DC    AL1(4,0,10),C'/',AL4(1000),C'<',AL4(99999)
            DC    X'FF'
PWRHT       DC    AL1(3,0,1,6)
            DC    X'FF'
PRTRHT      DC    AL1(1,0,2),C'PN'
            DC    AL1(1,0,2),C'TN'
            DC    X'FF'
DISRHT      DC    AL1(6,0,2),C'FR'
            DC    AL1(6,4,2),C'GR'
            DC    AL1(5,8,0)
            DC    X'FF'
NORHT       DC    AL1(7,0,0)
            DC    X'FF'

EXTABLE     ST    2,GLOBTIME              +0
            ST    2,GLOBPAGE              +4
            ST    2,GLOBCARD              +8
            STM   1,2,PASSWORD            +12
            MVC   PRNTTYPE(2),0(2)        +16
            BAL   5,DISFRS                +22
            BAL   5,DISGRS                +26
            OI    DMPSW1,X'01'            +30
            OI    DMPSW1,X'FF'-X'01'      +34
            MVI   DMPSW2,8                +38+0
            MVI   DMPSW2,12               +38+4
            BAL   5,*+4                   +38+8

            ST    2,ADDR
            MVI   DMPSW2,16
            BR    5

DISFRS      OI    DMPSW1,X'40'
DISREG      MVI   DMPSW2,20
            BR    5
DISGRS      OI    DMPSW1,X'20'
            B     DISREG
```

October 1976

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)

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.

slist 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.

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 lht and rht. 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 lht. The forms of the right-hand sides are specified in the right-hand table, indicated by rht.

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 3+N or 5+N bytes in length, where N is the number of characters comprising the left-hand side keyword. The format is:

1 or 2 bytes - 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 - 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 - count of number of characters in the left-hand side.
N characters - the text of the left-hand side keyword.

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 comprising the left-hand side text may be zero, implying a null left-hand side (i.e., the degenerate form "RHSide").

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 the 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.

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 the 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.

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.

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 implicit 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 are forbidden to 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 the 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 the 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 the 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 maximum and minimum 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.

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.

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.

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	Effect
15	1	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 subrcutine 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 GR2 if not an address, or address of the first element of 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 bit 23, 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	Spelling correction of left-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	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	The return vector indicated by the rvec parameter is formatted in the following manner: 1 word - error code, listed below, 26 words - variable information, dependent upon error code:

	Code	Significance and Information Returned
	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 expressicn, 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.
	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	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	The slist parameter indicates a special set 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	The slist parameter need not be specified, but a relational set of separators are used as if the slist parameter had specified ">=", "<=", "¬=", ">", "<", "=" in the presented order. GR5 is also set up as described above.
	00	Only "=" is a valid separator character.
22	1	The dlist 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	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	The contents of the text parameter will be updated to indicate the delimiter at the end of the last keyword processed.
	0	text is not updated.
25	-	Reserved; should be zero.
26	1	Convert all keyword input to uppercase, including prompt input. Translation to uppercase and subsequent processing is performed upon a copy of the input text, not the input text itself.
	0	Leave all input as is.
27	1	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	Return to the calling program on the first invalid keyword expression encountered.
29	1	Prompt user for corrections if invalid expressions are found.
	0	Do not prompt user for correction.
30	1	Print error comments on *MSINK*.
	0	Do not print error comments, return them in the rvec return vector.
31	1	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 presented, in increasing order of complexity. The first example mimics the processing of some of the options of the MTS $SET command, namely:

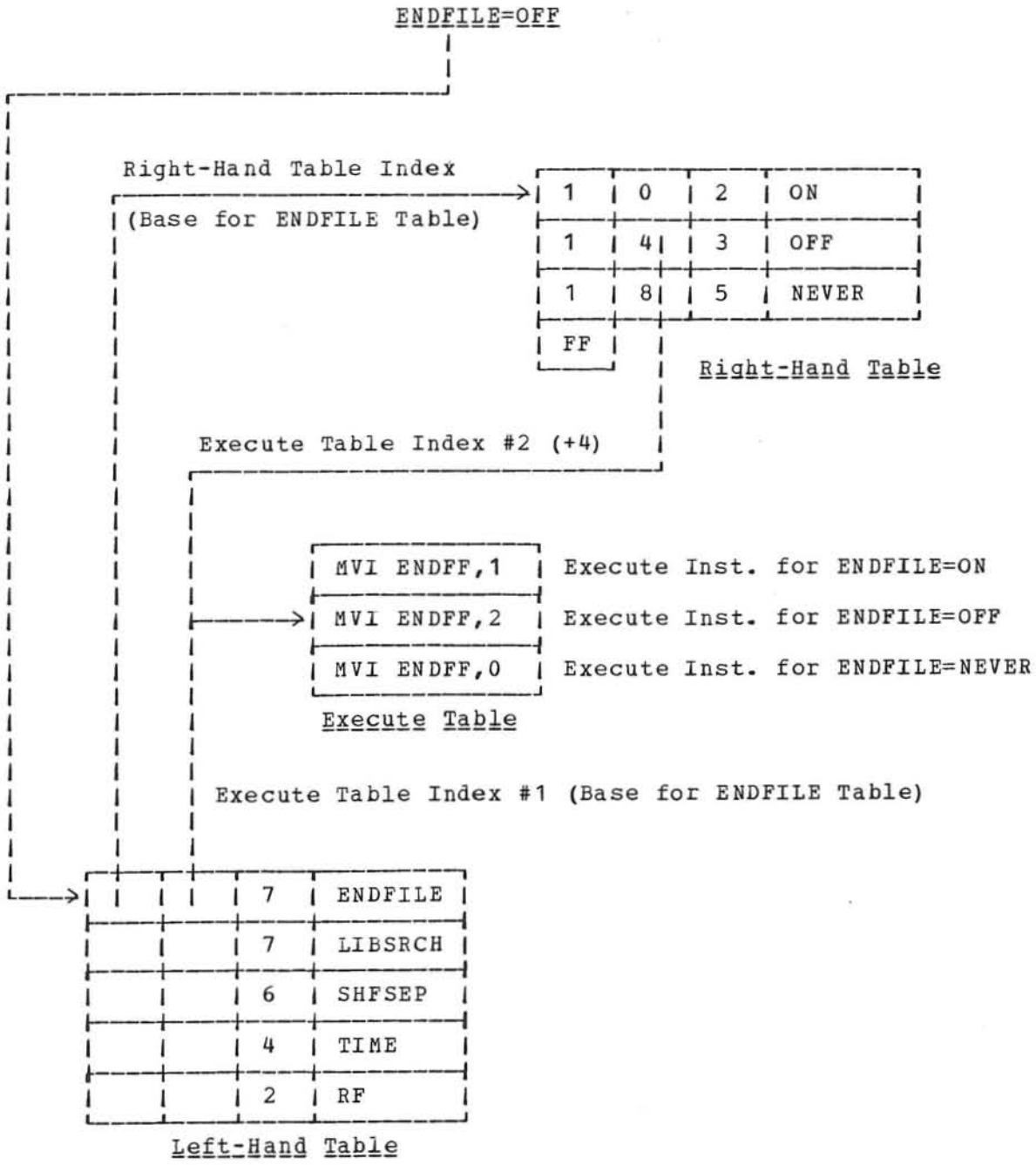
```
     ENDFILE=ON, ENDFILE=OFF, ENDFILE=NEVER
     LIBSRCH=OFF, LIBSRCH=FDname
     SHFSEP=c
     TIME=xxxx, TIME=xxxxS, TIME=xxxxM
     RF=<hex number>, RF=GRxx

          CALL  KWSCAN,(LHTL,LHT,EXT,STR,RHT,STRL,SWS,0)
*
*  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(4,0,15)
          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 second example draws from 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'0000E827'         Correct spelling, RVEC
*                                  format, relational
*                                  separators, uppercase,
*                                  print, prompt, multiple
*                                  keywords
RVEC     DS    27F
```

October 1976

LETGO

SUBROUTINE DESCRIPTION

Purpose: To periodically unlock and then relock a file.

Location: *LIBRARY

Calling Sequences:

Assembly: CALL LETGO,(fdub,howlck,delay)

FORTRAN: CALL LETGO(fdub,howlck,delay)

- fdub is the location of a fullword-integer (INTEGER*4) FDUB-pointer (as returned by the subroutine GETFD) for the file to be unlocked.
- 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.

Return Codes:

0 Successful return.
4 FDUB-pointer (first argument) is not valid for a file.
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. 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 write 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 during 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 not be periodically unlocked, for example, during the time a program is waiting for input from a terminal.

LETGO will stop unlocking and relocking the file as soon as the FDUB has been released (the subroutine FREEFD called).

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).

October 1976

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:

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

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

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

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

128 bit 0: 1, if errexit is specified.
64 bit 1: 1, if output is specified.
32 bit 2: 1, if lsw is specified.

info(1)

16 bit 3: 1, if gtsp is specified.
8 bit 4: 1, if frsp is specified.
4 bit 5: 1, if pnt is specified.
2 bits 6-7: 0

info(2)

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.
8 bit 12: 1, if the program to be loaded is to be merged with the program previously loaded.
4 bit 13: 1, to suppress prompting at a terminal.
2 bit 14: 1, to force allocation of a new loader symbol table.
1 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 (0-19).
(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 cf info is 0 (the default), the lsw parameter is ignored and the global MTS settings are used. The loader control bits are defined as follows:

bits 0-23: 0
bit 24: 1, to suppress the pseudo-register map.
bit 25: 1, to suppress the predefined symbol map.
bit 26: 1, to print undefined symbols.
bit 27: 1, to print references to undefined symbols.
bit 28: 1, to print references to all external symbols.
bit 29: 1, to print dotted lines arcund the

loader map.
bit 30: 1, to print a map.
bit 31: 1, to print nonfatal error messages.

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

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

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

Values Returned:

None.

Description: 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. 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,'PF','IL','E '/
          CALL LINKF('*CCQUEUE ',0,ADROF(PAR))
          END
```

The above FORTRAN program is equivalent to issuing the MTS command "$RUN *CCQUEUE PAR=PFILE".

```
Assembly:       CALL LINK,(INPUT,INFO,PAR,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'
          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.

October 1976

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.

Example:

```
      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
```

October 1976

October 1976

LOAD, LOADF

SUBROUTINE DESCRIPTION

Purpose: To effect the dynamic loading of a program.

Location: Resident System

Calling Sequences:

Assembly: CALL LOAD,(input,info,parlist,switches,
rtnlist,output,lsw,gtsp,frsp,pnt)

FORTRAN: indx = LOADF(input,info,parlist,switches,
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 (0-19).
(5) the address of an input subroutine to be called during loading via a READ subroutine calling sequence to read loader records (i.e., the input subroutine is called with a parameter list identical to the system subroutine READ). In this case, bit 9 in info must be 1.

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

(1) a halfword of LOADF 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.
bits 6-7: 0
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 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: 0
bit 30: 1, load into system storage (bits 0-7 contain the storage index number to be used).
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 (0-19).
(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 cf 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.

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.

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 8 arguments. For most applications, only the first 4 are required. Both info and switches contain LOAD control bits, some of which are duplicates. In these cases, LOAD produces 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. In particular, they describe the use of initial ESD lists, merging with previously loaded programs, and the relationship between LOAD, LOAD, and XCTL storage management.

Example:

```
FORTRAN:  LOGICAL*1 PAR(8)
          DATA PAR/'H','I',' ','T','H','E','R','E'/
          INTEGER SWITCH/Z00800041/
          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.

```
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.

October 1976

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:
  - 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.
- item — is either the location of an 8-character external symbol (left-justified with trailing blanks), the location of a fullword virtual memory address, or the location of a fullword integer index.
- bitsout — is the location of a fullword into which LOADINFO will put output code bits.
- regout — is the location of a region of 20 fullwords into which LOADINFO will put information about the symbol or virtual memory address. This region is cleared to zeros by LOADINFO before information is inserted.

Return Codes:

- 0 Successful return.
- 4 Symbol or control section 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 work properly. 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.

LOADINFO returns the information 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 Bit	Bitsout Value	Regout Displ	Regout Contents
31	1	0	External symbol name (left-justified with trailing blanks).
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: 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 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     F     ASSIGNED VM ADDRESS
SYMRF    DS     F     RELOCATION FACTOR
SYMLEN   DS     F     LENGTH IF CSECT OR COMMON SECTION
SYMSIN   DS     F     STORAGE INDEX NUMBER
SYMTYPE  DS     F     TYPE INFORMATION
PRADDR   DS     F     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     F     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.

October 1976

October 1976

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 19), 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.

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

Examples: Assembly:

```
          CALL LOCK,(UNIT,HOW,WAIT)
          .
          .
UNIT      DC    F'6'      Logical I/O unit 6
HOW       DC    F'1'      Lock for modification
WAIT      DC    F'-1'     Wait indefinitely
```

FORTRAN:

```
          INTEGER*4 UNIT
          DATA UNIT/6/

          CALL LOCK(UNIT,1,-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).

October 1976

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:

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 19), 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.

bits 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 unit 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.

October 1976

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:

- len — is the integer length in bytes. No restriction is placed on the size of len. An error message will be generated if len < 0; or, for the entries IED or IEDMK, if len > 256.
- base1 — is the base location of the first operand.
- displ1 — is the integer displacement in bytes for the first operand. No restriction is placed on the size of displ1.
- base2 — is the base location of the second operand.
- displ2 — is the integer displacement in bytes for the second operand. No restriction is placed on the size of displ2.
- dr — is an integer return parameter for ITRT and IEDMK only. For ITRT, dr will contain the displacement in bytes from the beginning of the argument list (base1+displ1), to the argument corresponding to the first nonzero function byte (if any). For IEDMK, dr will contain the displacement in bytes from the beginning of the source (base2+displ2), to the result character, whenever the latter is a zoned source digit and the significance indicator was off before the examination. In both cases, dr 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 for 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 func-

tion byte is nonzero, the operation 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.

October 1976

October 1976

MOUNT

SUBROUTINE DESCRIPTION

Purpose: To mount magnetic and paper tapes, Audio Response Unit lines, and connections on the MERIT Computer Network.

Location: Resident System

Calling Sequences:

Assembly: CALL MOUNT,(string,len)

CALL MOUNT,(par)

MOUNT 'string'

FORTRAN: CALL MOUNT(string,len)

CALL MOUNT(par)

Parameters:

- string is the location of a character string containing one or more mount requests separated by semicolons (see the $MOUNT command description in MTS Volume 1).
- 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 mount requests separated by semicolons (see the $MOUNT command description in MTS Volume 1).

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.

Note: The MOUNT subroutine prints messages on the logical I/O unit SERCOM or *MSINK* if SERCOM has not been assigned. The echoing of mount requests (on SERCOM or *MSINK*) can be suppressed by the MTS $SET ECHO=OFF command (or by calling the CUINFO

subroutine for the ECHOOFF item to perform the equivalent function).

The complete description for using the MOUNT macro is given in MTS Volume 14.

Examples: Assembly:

```
        CALL MOUNT,(STR,LEN)
         .
         .
LEN     DC    H'28'
STR     DC    C'POOL 9TP *T*;MNET *NET* D=MS'

        MOUNT 'POOL 9TP *T*;MNET *NET* D=MS'
```

FORTRAN:

```
      INTEGER*2 LEN
       .
       .
      LEN=28
      CALL MOUNT('POOL 9TP *T*;MNET *NET* D=MS',LEN)
```

The above three examples call MOUNT to mount a 9-track pool tape with pseudo-device name *T* and a MERIT connection to Michigan State University with pseudo-device name *NET*. The first assembly example uses the CALL macro and the second uses the MOUNT macro.

October 1976

MTS

SUBROUTINE DESCRIPTION

Purpose: To suspend execution of a program and return to MTS command mode. 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

PL/I: CALL MTS;

Return Codes:

None

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

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

October 1976

MTSCMD

SUBROUTINE DESCRIPTION

Purpose: To suspend execution of a program, return to MTS command mode, 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 length is a fullword-aligned address and the first two bytes so specified 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.

Description: This subroutine does a return 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.

October 1976

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 19), 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.

Note: The Read and Write Pointers have values which point to the next line to be read or written.

Description: See Appendix B of the section "Files and Devices" in MTS Volume 1 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.

October 1976

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)

```
REAL*8 OSGRDT
date=OSGRDT(osdat,grgdat)
```

PL/I: 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.

Values Returned:

FR0 contains the Gregorian date in the character form "MM/DD/YY".

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)

          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:

```
     CALL PLCALL(OSGRDT,F2,'   71120',GRDAT);
     DECLARE OSGRDT ENTRY,
             F2 FIXED BINARY(31) INITIAL(2),
             GRDAT CHARACTER(8);

     UNSPEC(GRDAT)=UNSPEC(PLCALLD(OSGRDT,F2,OSDAT,
             DUMMY));
     DECLARE GRDAT CHARACTER(8),
             PLCALLD RETURNS(FLOAT(16)),
             OSGRDT ENTRY,
             F2 FIXED BINARY(31) INITIAL(2),
             OSDAT CHARACTER(8) INITIAL('   71120'),
             DUMMY CHARACTER(8);
```

The above examples call OSGRDT to convert the OS date 71120 into the corresponding Gregorian date April 30, 1971.

October 1976

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)
```

FORTRAN:
```
CALL PERMIT(what,how,whotyp,wholen,who,info,
            wholen2,who2,ercode,errmsg,rc4)
```

Parameters:

what is the location of either
(a) a file name with trailing blank (if info=0),
(b) a fullword-integer FDUB-pointer (such as returned by GETFD) (if info=1),
(c) a fullword-integer logical I/O unit number (0 through 19) (if info=1), or
(d) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if info=1).

how 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
RW	7
UNLIM	63

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 program key	6
who is signon ID and program key	7
who is project number and program key	8

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) or the number of characters in the program key (1 to 13) specified by who (for whotype=6)

who is the location of the 1- to 4-character signon ID or project number (for whotype=0,1, 7, or 8) or the 1- to 13-character program key (for whotype=6)

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 required only when whotype= 7 or 8.

who2 is the location of the 1- to 13-character program key. This parameter is required only when whotype=7 or 8.

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:

number	message
1	ILLEGAL PARAMETER LIST POINTER
2	ILLEGAL "WHAT" PARAMETER ADDRESS
3	ILLEGAL "HOW" PARAMETER ADDRESS
4	"HOW" PARAMETER VALUE NOT 0 TO 63 OR 128
5	ILLEGAL "WHOTYPE" PARAMETER ADDRESS
6	"WHOTYPE" PARAMETER VALUE NOT 0 TO 8
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
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"

Error numbers 201 and above indicate a file system error of some sort.

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 is the statement label to transfer to if the corresponding 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)
         .
         .
         .
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.

October 1976

PGNTTRP

SUBROUTINE DESCRIPTION

Purpose: To allow control to be returned to the user on a program interrupt.

Location: Resident System

Alt. Entry: PGNTT

Calling Sequences:

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

Parameters:

GR0 should contain zero or the location to transfer control to if a program interrupt occurs.
GR1 should contain the location of a 72-byte save region for storing pertinent information.

Return Codes:

None.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine specifying PGNTT as the entry point.

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 of the save region contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region.

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 interrupt intercept again enabled.

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 interrupt 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 *,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'
```

October 1976

POINT

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:

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

info 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 code. 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.

code 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.

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 appropriately will result in a deadlock.
20 Hardware error or software inconsistency encountered.
24 Automatic wait for (shared) file was interrupted.

Note: 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.

Description: See Appendix B of the section "Files and Devices" in MTS Volume 1 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:

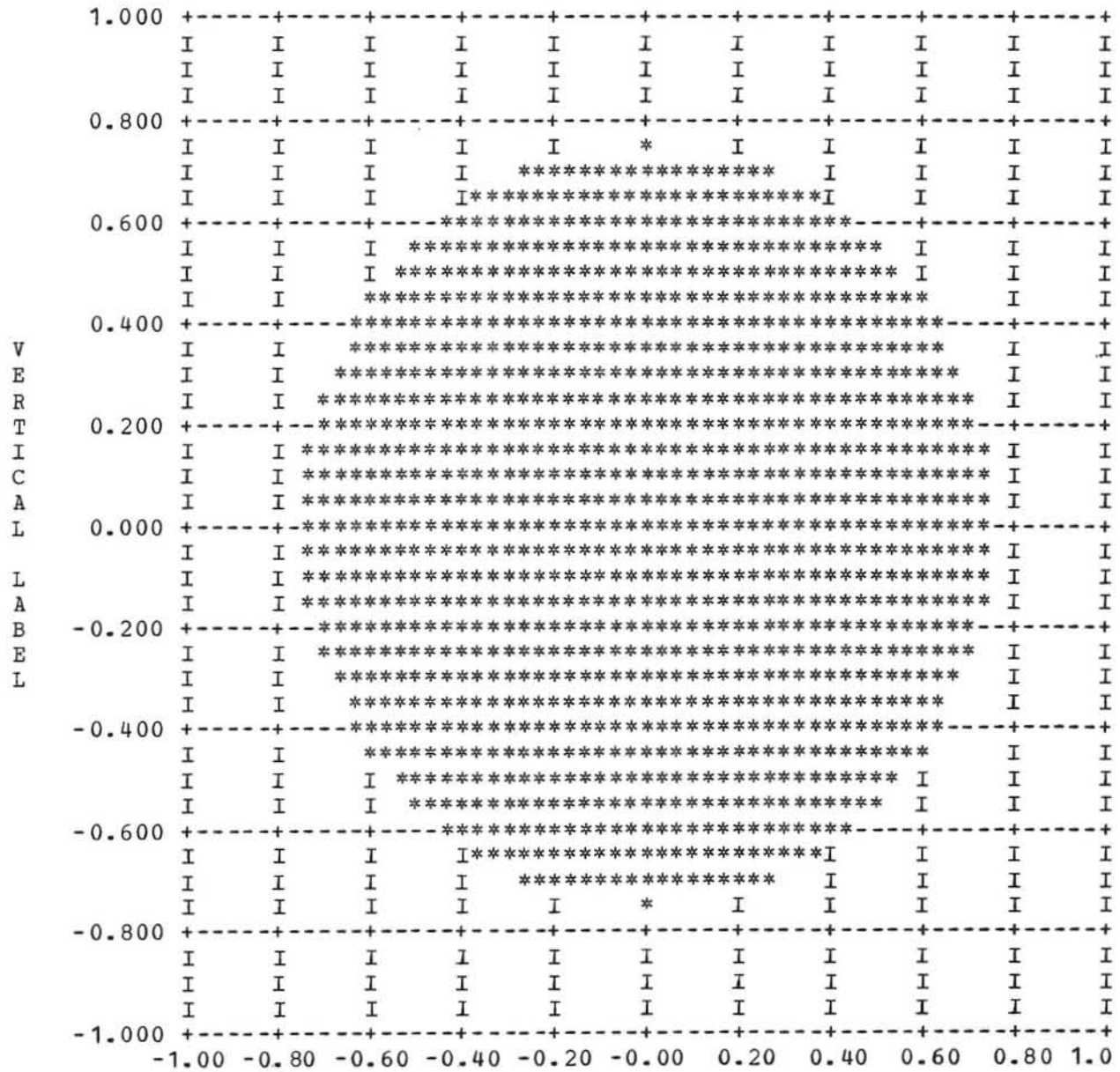
SPRINT - Output from the printer plot routines (the plot). Note: When the printer is used as the SPRINT device, a page skip is normally issued by the user before calling PLOT4 in order to force a skip to the top of the next page before starting the plot.

SERCOM - Error messages.

Example: FORTRAN:

```
      DIMENSION IMAGE(1500)
      DATA BCD/'*   '/,NSC(5)/1,0,3,0,2/
      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
```

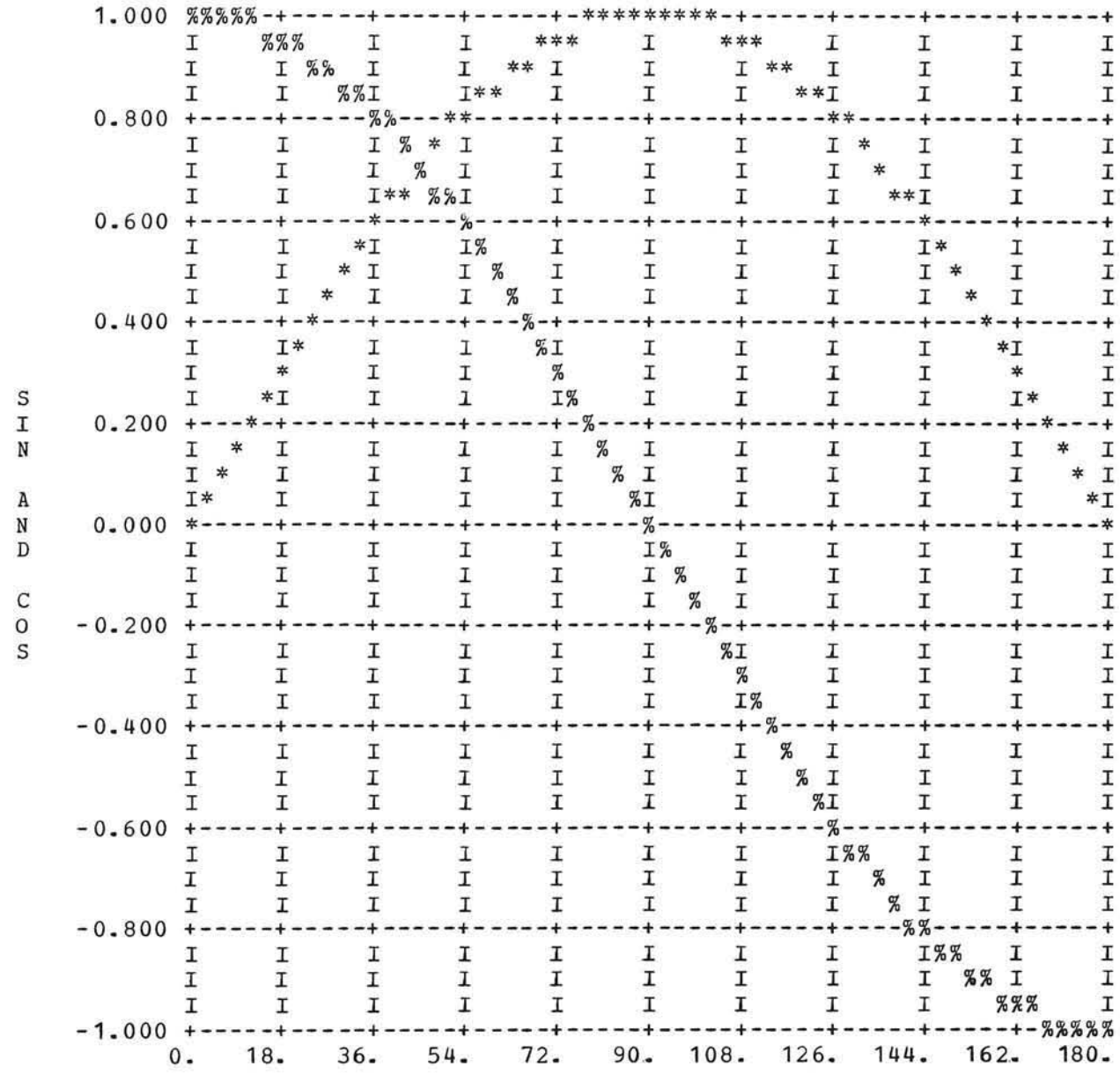
The above FORTRAN program will produce the plot given on the following page.


1.000
0.800
0.600
0.400
0.200
0.000
-0.200
-0.400
-0.600
-0.800
-1.000
VERTICAL LABEL
-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
0.800
0.600
0.400
0.200
0.000
-0.200
-0.400
-0.600
-0.800
-1.000
SIN AND COS
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:

nscale 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).
- nscale(1) If nscale(1)=0, the values 0,3,0,3 are used for nscale(2) through nscale(5).
- nscale(2) If nscale(2)=Y, the numbers printed along the Y-axis are 10**Y times their true value.
- nscale(3) The number of decimal places printed for Y values.
- nscale(4) If nscale(4)=X, the numbers printed along the X-axis are 10**X times their true values.
- nscale(5) The number of decimal places printed for X values.

nhl is the location of a fullword integer giving the number of horizontal lines in the plot. This number must be 2 or greater.

nsbh is the location of a fullword integer giving the number of spaces between horizontal lines. This number must be 1 or greater.

nvl is the location of a fullword integer giving the number of vertical lines in the plot. This number must be 2 or greater.

nsbv is the location of a fullword integer giving the number of spaces between the vertical lines. This number must be 1 or greater.

rc4 is the statement label to transfer to if a return code of 4 is encountered.

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 is the statement label to transfer to if a return code of 8 is encountered.

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 are the statement labels to transfer to if a return code of 12 or 16 is encountered.

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:

nchar is the location of the fullword integer number of characters in the vertical label. If this is zero, no label will be printed.

label is the location of a region containing the label to be printed.

rc20,rc24,rc28 are statement labels to transfer to if a return code of 20, 24, or 28 is encountered.

Return Codes:

0 Normal return.
20 PLOT2 has not been entered.
24 Using a log scale with a negative or zero xmin, xmax,ymin, or ymax value (see SETLOG and PLOT2).
28 Error in scaling; one or more values can not be printed in the form specified by nscale (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/Z00006A00/
.
.
CALL PRCHAR(CHARS)
```

The above example changes the vertical line character to "|" (vertical bar), and leaves the other three characters unchanged.

October 1976

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 are the statement labels to transfer to if a return code of 4 or 8 is encountered.

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	is the statement label to transfer to if a return code of 4 is encountered.

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.

October 1976

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)

Parameters:

arg is the location of a word 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.

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.

October 1976

October 1976

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
```

PL/I:
```
CALL QUIT;
```

Return Codes:

None.

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

MTS 3: SYSTEM SUBROUTINE DESCRIPTIONS

October 1976

340 QUIT

October 1976

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 11 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 11 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 is in GR15 as 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('FDname '),1,FDUB,&9)
```

This example calls GETFD with GR0 containing a zero and GR1 containing the address of the character string "FDname". 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
                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.

October 1976

READ

SUBROUTINE DESCRIPTION

Purpose: To read an input record from a specified logical I/O unit.

Location: Resident System

Alt. Entry: READ#

Calling Sequences:

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

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

PL/I: See the IHEREAD subroutine description.

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 19), or
(c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).

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

Return Codes:

0	Successful return.
4	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.

There are no default FDnames for READ.

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.

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    .
```

October 1976

October 1976

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(bfr,len,mod,lnum,unit)

Parameters:

- bfr is zero (not the location of a zero) on the first call and the location of the input buffer on subsequent calls (see "Description" below).
- len is the location of a halfword 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" section 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 FBUB-pointer (such as returned by GETFD),
  (b) a fullword-integer logical I/O unit number (0 through 19), or
  (c) a left-justified, 8-character logical I/O unit name (e.g., SCARDS).

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 bfr is zero, the subroutine READBFR will internally call the subroutine GDINFO to determine the length of the longest record that can be read from unit and will allocate a buffer that is large enough to

accommodate it; the address of this buffer will be stored into bfr 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 bfr 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 to unit 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 bfr.

Example:

```
Assembly: LABEL   CALL READBFR(BUFF,LEN,LNUM,UNIT)
                  LTR  15,15
                  BNZ  EOF
                  L    2,BUFF        Get address of buffer
                    .
                    .                Process record
                    .
                  B    LABEL
          EOF     L    1,BUFF        Release buffer
                  CALL FREESPAC
                    .
                    .
          BUFF    DC   F'0'
          LEN     DS   H
          LNUM    DS   F
          UNIT    DC   C'SCARDS  '
```

The above example reads records from SCARDS until a nonzero return code is encountered. After each call to READBFR, BUFF contains the location of the record read. When a nonzero return code is encountered, the buffer is not released.

October 1976

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:

oldname is the location of the old name (with a trailing blank) of the file to be renamed.
newname is the location of the new name (with a trailing blank).
rc4...rc36 are statement labels to transfer to if the corresponding return codes occur.

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).

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.

October 1976

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 19), or
  (c) a left-justified 8-character logical I/O unit name (e.g., SCARDS).
- first is the location of a fullword containing the internal line number of the first line 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 are statement labels to transfer to if the corresponding return codes occur.

Return Codes:

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

usage of the (shared) file).
24 Inconsistent parameters specified (renumbering will cause duplicate or nonincreasing line numbers, 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 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.

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 UNT
      DATA UNT/4/
        .
        .
      CALL RENUMB(UNT,-99999999,99999999,1000,1000)
```

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.

October 1976

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:

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

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

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

cnt is the location of a fullword in which the count of the number of lines in the specified range will be returned.

buffer 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).

Return Codes:

0 The line numbers were returned.
4 The file does not exist.

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 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.

Examples: Assembly:

```
        CALL GETFST,(UNIT,FSTLNR)
        CALL GETLST,(UNIT,LSTLNR)
        CALL RETLNR,(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
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 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).

356 RETLNR

October 1976

REWIND

SUBROUTINE DESCRIPTION

Purpose: To rewind a logical I/O unit in FORTRAN.

Location: *LIBRARY

Calling Sequences:

FORTRAN: CALL REWIND(arg)

Parameters:

arg is the location of a fullword integer corresponding to the logical I/O unit number to be rewound. These are 0 through 19.

Description: If the logical I/O unit number specified by arg is attached to a tape, it is rewound. If it is attached to a line file, it is reset so that the next 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, 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.

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.

October 1976

October 1976

REWIND#

SUBROUTINE DESCRIPTION

Purpose: To reset a magnetic tape or a file to be read from the beginning.

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 19.

Return Codes:

0 Successful return.
4 Unable to rewind the device specified by GR0 and GR1.

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

Description: If GR0 and GR1 specify a 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, 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 concatenation, this subroutine affects only the currently active member of the concatenation.

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 macrc.

October 1976

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,aexit)

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.

aexit 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 aexit and id are zero, all timer interrupts will be canceled.

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 aexit 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.

Example: Assembly:

```
          CALL RSTIME,(ONE,TIMLEFT,AEXIT)
            .
            .
ONE       DC  F'1'
TIMLEFT   DS  FL8
AEXIT     DC  A(EXIT)
EXIT      DS  19F
```

This example cancels the interrupt with the identifier "1" and the exit region "EXIT". The time remaining is returned in "TIMLEFT".

October 1976

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. Entry: SSTOR

Calling Sequences:

Assembly:
```
L     0,switch
L     1,sinbr
L     2,subr
CALL SCANSTOR
```

Parameters:

GR0 if 0, only storage with the specified storage index number (GR1).
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).

GR1 storage index number or zero. If zero, the storage index number of the current link level will be used.

GR2 location of the subroutine to call for each block. When this call is made, GR0 will have the length and GR1 will have the location of the block.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine and specifying SSTOR as the entry point.

Return Codes:

None

Description: For a further description of storage index numbers, see the "Virtual Memory Management" section in MTS Volume 5.

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)
```

The above example (coded in two different ways) 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.

October 1976

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:

- 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.
- 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.

The default FDname for SCARDS is *SOURCE*.

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.

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     .
```

October 1976

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.

wkarea is the location of a doubleword-aligned area of 400 bytes that may be used by SDUMP as a work area.

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

last 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 last which is less than the address in first will cause an error return.

Note: The default case for switch (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: Output Formats

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 any 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 on, the unit of storage presented in each print item is a halfword-aligned halfword.

(2) If, and only if, the mnemonic switch is off and the hexadecimal switch is on (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 "XXXX" 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 "XXXX" 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 Universal Instruction Set and those instructions exclusively used by the IBM 360/67. The five special mnemonics for the IBM 360/20 are also included.) 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'
```

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

October 1976

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.

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.

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)
```

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,aexit)

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 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.

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.

aexit 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 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 aexit 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.

Example: Assembly:

```
         CALL SETIME,(ZERO,ONE,TENSEC,AEXIT)
          .
         CALL SETIME,(ONE,TWO,FIVMIN,AEXIT)
          .
         CALL SETIME,(FIVE,THREE,TWO30,AEXIT)
          .
          .
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'
AEXIT    DC  A(EXIT)
EXIT     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.

October 1976

October 1976

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.

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.

October 1976

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)

FORTRAN: CALL SETKEY(what,pkey,info,ercode,errmsg,&rc4)

Parameters:

what is the location of either:
(a) a file name with trailing blank (if info=0),
(b) a fullword-integer FDUB-pointer (such as returned by GETFD) (if info=1),
(c) a fullword-integer logical I/O unit number (0 through 19) (if info=1), or
(d) a left-justified, 8-character logical I/O unit name (e.g., SCARDS) (if info=1).

pkey is the location of the program key to be associated with the file. One trailing blank is required.

info is the location of a fullword integer which specifies the kind of what parameter supplied.

ercode (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 errmsg 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 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" (Permit access is required to set the program key.)
104	DEADLOCK SITUATION, TRY LATER -FILE "XXXX"
105	INTERRUPTED OUT OR WAIT FOR LOCKED FILE "XXXX"

Error numbers 201 and above indicate a file system error.

errmsg (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 ercode parameter.

rc4 is the statement label to transfer tc 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 ercode and errmsg 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.