GH20-0859-0

Type III Class A Program

Control Program-67/Cambridge Monitor System (CP-67/CMS) Version 3 Program Number 360D-05.2.005 User's Guide

CP-67/CMS is a general purpose time-sharing system developed for the IBM System/360. This guide describes the facilities of CP-67/CMS and provides detailed information about the user commands available and their usage.

First Edition (October 1970)

This Type III Program performs functions that may be fundamental to the operation and maintenance of a system.

It has not been subjected to formal test by IBM.

Until the program is reclassified, IBM will provide for it: (a) Central Programming Service, including design error correction and automatic distribution of corrections; and (b) FE Programming Service, including design error verification, APAR documentation and submission, and application of Program Temporary Fixes or development of an emergency bypass when required. IBM does not guarantee service results or represent or warrant that all errors will be corrected.

You are expected to make the final evaluation as to the usefulness of this program in your own environment.

THE FOREGOING IS IN LIEU OF ALL WARRANTIES EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

This edition applies to Version 3, Modification Level 0, of Control Program-67/Cambridge Monitor System (360D-05.2.005) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604.

PREFACE

The following documents are referenced in the CP-67/CMS User's Guide.

Fortran -
- OS/360 Fortran IV Language GC28-6515
- System/360 Basic Fortran IV Language GC28-6629
- System/360 Fortran IV Library Subprograms GC28-6596
- OS/360 Fortran IV (G and H) Programmer's Guide GC28-6817

Scientific Subroutine Package

Scientific Subroutine Package - Version II Programmer's Manual GH20-0205

PL/I -
- OS/360 PL/I (F) Programmer's Guide GC28-6594
- OS/360 PL/I Subroutine Library Computational Subroutines - GC28-6590
- A PL/I Primer - GC28-6808

Assembler-
- OS/360 Assembler Language GC28-6514
- OS/360 Assembler (F) Programmer's Guide GC26-3756

CP-67/CMS Documents - CP-67 Operator's Guide GH20-0856
- CP-67 Program Logic Manual GY20-0590
- CMS Program Logic Manual GY20-0591
- CP-67/CMS Installation Guide GH20-0857
- CMS SCRIPT User's Manual GH20-0860

SNOBOL - CMS SNOBOL User's Manual, Type III Documentation, Program 360D-03.2.016, IBM Corporation, DP Program Information Department, 40 Saw Mill River Road, Hawthorne, New York, October, 1970.

BRUIN - CMS BRUIN User's Manual, Type III Documentation, Program 360D-03.3.013, IBM Corporation, DP Program Information Department, 40 Saw Mill River Road, Hawthorne, New York, October, 1970.

Miscellaneous- System/360 Principles of Operation
GA22-6821

- 2740/2741 Communications Terminal
Operator's Guide
GA27-3001

- OS/360-Supervisor & Data Management
Macro-Instructions - GC28-6647

CONTENTS

FIGURES

INTRODUCTION

COMPONENTS OF THE SYSTEM

The CP-67/CMS time-sharing system consists of two independent components: the Control Program (CP-67, or CP for short) and the Cambridge Monitor System (CMS). The Control Program creates the time-sharing part of the system to allow many users to access the computer simultaneously. The Cambridge Monitor System provides the conversational part of the system to allow a user to monitor his work from a remote terminal.

Both components are independent of each other. CP-67 can be used on an appropriate configuration without CMS, and CMS can be run on a properly configured System/360 as a single-user system without CP-67. If CP-67 is used without CMS, an operating system (or systems) must be chosen to provide the conversational or production aspect of the system, as CP provides only the time-sharing capability.

CP-67 is capable of running many System/360 operating systems concurrently (including CP-67, OS/360, DOS, RAX, and APL/360) as long as they do not include any timing dependencies or dynamically-modified channel programs. Dynamically-modified channel programs are those which are changed between the time the Start Input/Output (SIO) instruction is issued and the end of the I/O operation (that is, changed by the channel program or the CPU). However, certain types of self-modifying channel programs can be translated, including those generated by the OS/360 Indexed Sequential Access Method (ISAM).

CP-67 is also capable of running System/360 operating systems along with CMS in a multiprogramming mode concurrent with its usual time-shared, multiaccess operation. If the System/360 operating system contains telecommunication facilities or remote job entry/remote job output support, CP allows that system to control the lines of the 2701, 2702, or 2703 transmission control unit and allows the user to dial into that system from a remote terminal.

SYSTEM ENVIRONMENT

The environment of CP-67 is one of "virtual machines". A virtual machine is a functional simulation of a real computer and its associated I/O devices.

CP-67 builds and maintains for each user a virtual System/360 machine from a predescribed configuration. The virtual 360 is indistinguishable to the user and his programs from a real System/360, but it is really one of many that CP-67 is managing. CP allocates the resources of

the real machine to each virtual machine in turn for a short "slice" of time, then moves on to the next virtual machine--thus time-sharing.

Since the virtual machines are simulated, their configurations may differ from each other and from the real machine. For instance, the real machine may have 512K and eight disk drives, and the virtual machine can have 768K and two disk drives. One virtual machine may have a virtual 2702 and run an OS teleprocessing system, and another virtual machine that does not have a virtual 2702 may run CMS. One virtual machine may have a remote printer and a remote card punch, while another virtual 360 may have a dedicated printer and 2250. Regardless of the configuration, each user controls his virtual machine from his remote terminal, which is, in effect, his operator's console.

Like real machines, virtual machines operate most efficiently under an operating system. The Cambridge Monitor System (CMS) is designed to allow full use of a System/360 through a simple command language entered at the console (in the case of a CP-67 virtual machine, at the remote terminal). CMS gives the user a full range of capabilities--creating and managing files, compiling and executing problem programs, and debugging--using only his remote terminal. Since each user has his own virtual machine with his own copy of CMS residing "in it", nothing he does should affect any other user; if he destroys the CMS nucleus or abends the CMS system, he can reIPL his virtual machine and continue without disturbing other users. In addition, since users cannot get "outside" their virtual machines, CP-67 is protected from any user error.

CMS also provides a batch version intended primarily for "compile, load, and go" type jobs coming from tape or cards. The batch monitor can be run from a virtual machine as a background system with conversational CMS users on other virtual machines.

THE CONTROL PROGRAM, CP-67

Before a user is authorized to use CP, he must be assigned a USERID, which identifies him to the system, and a password, which is checked when he "logs in". Associated with each USERID is information concerning accounting, privilege class, options desired, and a table describing the virtual machine assigned to that user. Whenever he logs in, CP sets up this virtual 360 machine for him. Although all the virtual machines may be different, most are set up with the configuration expected by CMS, the most commonly used operating system. They include at least 256K of virtual core storage, a minimum of two disk drives, a console (the

terminal), a card read punch unit, and a printer. The real system usually has a larger number of disk drives, a drum, tape drives, and perhaps more core storage.

Because there is not room in real core for all users' virtual core, a technique called "paging" is used by the system. Virtual core is divided into 4096-byte blocks of storage called "pages". All but currently active pages are kept by the system on direct access secondary storage; this direct access space is allocated only on a demand basis; as active and inactive pages change status they are "paged" in and out of real core on demand. While the paging operation is being performed for one virtual machine, another can be operating. The paging operation, and resultant allocation of real core to a given user's pages, is transparent to the user. Special hardware is provided on the System/360 Model 67 that translates, at execution time, the user's (or user program's) addresses into the current real addresses of the relocated pages. This is called "dynamic address translation" and is transparent to the user.

All virtual machine I/O operations are handled by CP, which must translate them into real machine I/O operations. This requires two translations, accomplished as follows: CP intercepts all user I/O when Start I/O (SIO) is issued. It translates virtual device addresses into real device addresses, translates virtual core storage addresses into real core storage addresses, ensures that all necessary pages are in real core storage, builds a channel command word (CCW) string for the user, and issues SIO when the channel is free. The virtual machine is not given control from the time it issues an SIO until CP issues the real SIO and delivers the resulting condition code to the virtual machine. In the meantime, other virtual machines may be executing. When CP receives an interrupt indicating I/O completion, it sets an interrupt pending flag in the user's virtual machine status table; when control is returned to the virtual machine, the proper I/O interrupt is simulated.

Virtual machine unit record I/O is normally spooled onto disk by CP. Thus, any card deck to be "read" by a virtual machine would, in the normal case, have been read by CP before the user's call for it on his virtual machine, or transferred to that user from another user's files via the XFER console function in CP; the physical deck must have been preceded by a card containing the USERID, so that CP can know who will read the card-image file. Later, when the virtual machine has read the card deck, a card reader end of file is simulated. Card and printer output, similarly spooled, is not queued for physical output until CP is notified of end of file in one of three ways: the user logs off the system (end of file is assumed); the CLOSF console function specifies the (virtual) address of the device to be

closed; or CP detects an invalid CCW addressed to the device (end of file is assumed). Further output for a closed device is assumed to start a new file. So that the system operator can separate physical output, CP precedes all printed and punched output files with a record containing the USERID.

The CP console functions allow the user to control his virtual machine from the terminal much as an operator controls a real machine. To perform an IPL, for instance, the user types IPL and a device address or the name of a "named" operating system, such as CMS. The user can stop his virtual machine at any time by hitting the ATTN key and can request display of any portion of his storage and registers. He can modify the contents, if desired, and restart his machine. CP also recognizes a few special purpose commands, such as the XFER function mentioned above, the QUERY function to obtain the number of users on the system and their USERID's, as well as the number of current spooled files, the MSG function to communicate with other users, the DIAL function to connect the terminal to a multiaccess system, and the ATTACH and DETACH functions to add or remove I/O devices from a virtual machine configuration (ATTACH can only be issued by the Operator).

CAMBRIDGE MONITOR SYSTEM

The Cambridge Monitor System (CMS) is a single-user, conversational operating system capable of running on a real machine as well as on a virtual machine. It interprets a simple command language typed in at the operator's console (under CP-67, at the user's remote terminal).

When running on a real machine, CMS operates in the supervisor state. When running under CP-67, CMS operates in the pseudo-supervisor state; that is, CMS "thinks" it is running in the supervisor state, but CP is actually intercepting and translating supervisor interrupts.

Whether running on a real machine (see note below) or a virtual machine, CMS expects the following machine configuration:

Device Number	Virtual Address	Symbolic Name	Device Type
1052	009	CON1	console
2311,2314	190	DSK1	system disk (read-only)
2311,2314	191**	DSK2	permanent disk (user files)
*2311,2314	192**	DSK3	temporary disk (workspace)
*2311,2314	000**	DSK4	A disk (user files)
*2311,2314	000**	DSK5	B disk (user files)
*2311,2314	19C**	DSK6	C disk (user files)
1403	00E	PRN1	line printer
2540	00C	RDR1	card reader
2540	00D	PCH1	card punch
*2400	180	TAP1	tape drive
*2400	181	TAP2	tape drive

at least 256K bytes of core storage, 360/40 and up

*Optional devices not included in the minimum configuration

**This virtual address may be changed at any time by the CMS LOGIN command.

Note: For use on a real machine not having this I/O configuration, the device addresses can be redefined at load time.

Under CP, of course, these devices are simulated and mapped to different addresses and/or different devices. For instance, CMS expects a 1052 Printer-Keyboard operator's console, but most remote terminals are 2741s; CP handles all channel program modifications necessary for this simulation.

CMS allows the user to add his own programs for I/O devices not supported by the standard system. CMS also provides for dynamic specification of SVC routines.

CMS Batch Monitor

As well as being a conversational monitor, CMS provides a batch facility for running CMS jobs. The CMS batch monitor accepts a job stream from a tape unit or the card reader and writes the output on tapes, on the printer, or on the card punch. The job stream can consist of a System/360 Operating System SYSIN job stream with FORTRAN (G) and ASSEMBLER (F) compile, load, and go jobs calling certain cataloged procedures; or it can consist of CMS commands, along with control cards and card decks for compile, load, and go jobs for all the CMS-supported compilers.

Like the conversational CMS, the batch monitor can run either from a virtual machine or from a real machine. Under CP, it can be used as a background monitor along with other conversational CMS users.

To eliminate the possibility of one job modifying the CMS batch monitor's nucleus in such a way as to affect the next job, CMS is reIPL'ed before each job begins. Files can also be written onto the batch monitor's permanent disk and then punched or printed (such as files written by FORTRAN programs); these files should be of limited size and considered temporary, as they are erased at the completion of each job.

CP-67/CMS SAMPLE TERMINAL SESSION

A sample terminal session is described and illustrated below. User input is in lowercase; typeout from the system appears in uppercase.

After logging in to CP-67 and initializing CMS, the user, CSC1, issues a LISTF command to obtain a list of all files stored on his read-write disks. To allow multiple commands to be entered on one line, there is a line-end character. The LINEND command is issued to define the line-end character as the exclamation point (!) and to allow the pound sign (#) to be used as the logical tab character in EDIT. The file MAIN FORTRAN is then created and filed on the user's permanent disk. Compilation of the file is terminated due to program errors (indicated by a $ symbol below the error encountered). The file is then modified and edited to correct the line in error, and the new source file stored on disk. Again an error is encountered and the file reedited.

After a successful compilation, the $ command is called to load the file into core and execute it. LOAD and START perform the same function as $ (as shown). Specifying the XEQ option with the LOAD command also causes execution to begin after the file is loaded.

LISTF and ERASE commands are used to selectively list and erase files, and the PRINTF command is used to print all, and then part, of the contents of a file. KT causes typeout to be discontinued if entered after the ATTN key is hit twice.

The OFFLINE command punches or prints the specified file on offline devices. The ALTER command changes the identifier of a file. KX, entered after hitting ATTN twice, stops execution of the current program, reloads a new copy of CMS, and returns control to CMS.

An EXEC file (consisting of CMS commands) is created and filed. The file is then executed by issuing the EXEC command, which causes each of the commands contained in the file to be executed individually. Operand substitution is illustrated by modifying and reexecuting the file using

ampersand (&) arguments.

Hitting ATTN once transfers control to the Control Program, where the QUERY console function is issued to determine the number of users on the system, their names, and the message of the day from the operator. The BEGIN console function then returns control to CMS and the user logs out from both CMS and CP-67.

The sample terminal session follows.

```
#
cp-67 online        xd.65 qsyosu

login csc1        <-------The user's id is specified upon logging in.
ENTER PASSWORD::
      <----------------------------------The protected password does
CP WILL BE UP 24 HOURS A DAY                not print when entered.
READY AT 09.14.49 on 11/27/69
CP

ipl cms
CMS...VERSION nn   LEVEL  mm

listf
FILENAME FILETYPE MODE NO.REC. DATE
INDIAN   LISTING   P1   003    8/18
DUMPREST SYSIN     P1   009    8/20
SUPERSCR SYSIN     P1   070    8/22
MY       FORTRAN   P1   001    8/26
INDIAN   TEXT      P1   002    8/29
FORTCLG  EXEC      P1   001    8/29
LOAD     MAP       P5   003    8/30
FIN      SCRIPT    P1   001    8/30
TUES     SCRIPT    P1   001    8/31
FRST     SCRIPT    P1   001    8/31
DUMPREST LISTING   P1   007    9/01
AGENDA   SCRIPT    P1   001    9/01
INDIAN   FORTRAN   P1   001    9/01
R; T=0.06/0.21 09.16.12

linend !
R; T=0.01/0.02 09.16.20

edit main fortran
NEW FILE.
INPUT:
c main program  Nov. 27, 1969
      write (6,10)
10#format (' a = ') <--------The # is a logical tab character that
#read (5,20) a                inserts blanks and places the following
20#format (8.3)               characters typed on the line into column
#write (6,25) a,x             7 of the card image in a FORTRAN file.
#call exit
#end
                    <--------Carriage return with no data entered
EDIT:                         on line (that is, a null line) takes
file                          user out of Input mode into Edit mode
R; T=0.08/0.57 09.18.31       of the EDIT command.
```

```
fortran main
       0004        20        FORMAT (8.3)
                                        $
               01)  IEY013I  SYNTAX
                                              IEY022I      UNDEFINED LABEL
          25
E(00008); T=0.29/0.92 09.18.42


edit main fortran
EDIT:
print 20

C MAIN PROGRAM   NOV. 27, 1969
      WRITE (6,10)
10    FORMAT (' A = ')
      READ (5,20) A
20    FORMAT (8.3)
      WRITE (6,25) A,X
      CALL EXIT
      END
EOF:
locate /format/
10    FORMAT (' A = ')
locate /format/
20    FORMAT (8.3)
change /8/f8/
20    FORMAT (F8.3)
up 2
10    FORMAT (' A = ')
change / '/ ?'/
10    FORMAT (' A = ?')
find 20
20    FORMAT (F8.3)
insert #x = a**2
                  <--------Carriage return hit to confirm
EDIT:                       environment the user is in.
print
      X = A**2
top
print 20

C MAIN PROGRAM   NOV. 27, 1969
      WRITE (6,10)
10    FORMAT (' A = ?')
      READ
20    FORMAT (F8.3)
      X = A**2
      WRITE (6,25) A,X
      CALL EXIT
      END
EOF:
file
```

```
R; T=0.16/1.08 09.21.34

fortran main
                              IEY022I     UNDEFINED LABEL
      25
E(00008); T=0.30/0.94 09.21.43

edit main fortran
EDIT:
locate /25/
      WRITE (6,,25) A,X
insert 25#format (' a = 'f8.3,' x = 20.3)
print
25    FORMAT (' A = 'F8.3,' X = 20.3)
change /20/' f20/
25    FORMAT (' A = 'F8.3,' X = ' F20.3)
file
R; T=0.12/0.64 09.23.16

fortran main (list source)
R; T=0.31/0.97 09.23.35

$ main
EXECUTION BEGINS...
 A = ?
2.5
 A =    2.500 X =            6.250
R; T=0.28/0.94 09.24.01

load main
R; T=0.23/0.78 09.24.14

start
EXECUTION BEGINS...
 A = ?
3.1
 A =    3.100 X =            9.610
R; T=0.04/0.18 09.24.29

load main (xeq)
EXECUTION BEGINS...
 A = ?
3.2
 A =    3.200 X =           10.240
R; T=0.26/0.93 09.24.50

load main!start    <------The ! allows multiple
R; T=0.23/0.74 09.25.04   commands per line.

EXECUTION BEGINS...
 A = ?
2.5
 A =    2.500 X =            6.250
```

```
R; T=0.04/0.13 09.25.15

listf main *
FILENAME FILETYPE MODE NO.REC. DATE
MAIN     LISTING   P1   003     11/27
MAIN     FORTRAN   P1   001     11/27
MAIN     TEXT      P1   002     11/27
R; T=0.02/0.07 09.25.23

listf * listing
FILENAME FILETYPE MODE NO.REC. DATE
INDIAN   LISTING   P1   003     8/18
MAIN     LISTING   P1   003    11/27
DUMPREST LISTING   P1   007     8/20
R; T=0.02/0.07 09.26.08

erase * listing
R; T=0.03/0.10 09.26.30

listf * listing
FILE NOT FOUND
E(00002); T=0.01/0.04 09.27.05

printf main fortran

C MAIN PROGRAM   NOV. 27, 1969
      WRITE (6,,10)
10    FORMAT (' A = ? ')
      READ (5,20) A
20    FORMAT (F8.3)
      X = A**2
      WRITE (6,,25) A,X
25    FORMAT (' A = 'F8.3,' X = ' F20.3)
      CALL EXIT
      END

R; T=0.03/0.09 09.27.23

printf main fortran * 3 25

C MAIN PROGRAM   NOV. 27
      WRITE (6,,10)
10    FORMAT (' A = ? ')

R; T=0.02/0.08 09.30.47

printf main fortran

C MAIN PROGRAM   NOV. 27, 1969
      WRITE (6,10)
10 "
CP   <------------------------ATTN was hit once to enter CP
    <-------------------------ATTN was hit a second time
```

```
kt  <------------------------to kill typeout
R; T=0.03/0.09 09.31.02

offline punch main text@@@@fortran <--The four @ characters delete
R; T=0.03/0.14 09.31.32                the previous four characters.

offline print main fortran
R; T=0.03/0.10 09.31.40

offline print main listing
FILE NOT FOUND
E(00002); T=0.01/0.05 09.31.53

listf
FILENAME FILETYPE MODE NO.REC. DATE
DUMPREST SYSIN      P1   009     8/20
SUPERSCR SYSIN      P1   070     8/22
MY       FORTRAN    P1   001     8/26
FORTCLG  EXEC       P1   001     8/27
LOAD     MAP        P5   003     8/30
MAIN     FORTRAN    P1   001    11/27
FIN      SCRIPT     P1   001     8/30
TUES     SCRIPT     P1   001     8/31
FRST     SCRIPT     P1   001     8/31
AGENDA   SCRIPT     P1   001     9/1
MAIN     TEXT       P1   002    11/27
INDIAN   FORTRAN    P1   001     9/1
R; T=0.05/0.14 09.33.05

alter main fortran * mainone * *
R; T=0.02/0.12 09.33.28

listf main fortran
FILE NOT FOUND
E(00002); T=0.02/0.05 09.33.35

listf * fortran
FILENAME FILETYPE MODE NO.REC. DATE
MY       FORTRAN    P1   001     8/27
MAINONE  FORTRAN    P1   001    11/27
INDIAN   FORTRAN    P1   001     9/1
R; T=0.01/0.01 09.33.45

$ main
CP   <--------------------ATTN was hit once to enter CP
    <--------------------ATTN was hit a second time
kx  <--------------------to kill execution

CMS..VERSION n   LEVEL  m

listf mainonn@e *  <----- The @ deletes one character.
FILENAME FILETYPE MODE NO.REC. DATE
MAINONE  FORTRAN   P1   002   11/27
```

```
R; T=0.03/0.17 09.34.25

edit fortclgo exec
NEW FILE.
INPUT:
fortran mainone
$@load mainone (xeq)

EDIT:
file
R; T=0.07/0.43 09.35.02

printf fortclgo exec

FORTRAN MAINONE
LOAD MAINONE (XEQ)

R; T=0.02/0.06 09.35.11

exec fortclgo
09.35.23 FORTRAN MAINONE
09.35.27 LOAD    MAINONE (XEQ)
EXECUTION BEGINS...
 A = ?
3.4
 A =    3.400 X =          11.560
R; T=0.60/1.94 09.35.40

edit fortclgo exec
EDIT:
change /mainone/ &1/ * G
FORTRAN &1
LOAD &1 (XEQ)
EOF:
file
R; T=0.10/0.60 09.36.17

exec fortclgo mainone
09.36.53 FORTRAN MAINONE
09.36.59 LOAD    MAINONE (XEQ)
EXECUTION BEGINS...
 A = ?
5.1
 A =    5.100  X =          26.010
R; T=0.62/2.00 09.37.10

edit fortclgo exec
EDIT:
insert &set err exit
print 9
&SET ERR EXIT
FORTRAN  &1
LOAD &1  (XEQ)
```

```
EOF:
file
R; T=0.10/0.57 09.37.39

edit mainone fortran
EDIT:
print 4

C MAIN PROGRAM   NOV. 27, 1969
      WRITE (6,10)
10    FORMAT (' A = ?')
blank aa
      FORMAT (' A = ?')
next
      READ (5,20) A
change /read/red/
      RED (5,20) A
file badone
R; T=0.13/0.71 09.40.15

listf * fortran
FILENAME FILETYPE MODE NO.REC. DATE
MY       FORTRAN   P1   001     8/27
MAINONE  FORTRAN   P1   001    11/27
INDIAN   FORTRAN   P1   001     9/1
BADONE   FORTRAN   P1   001    11/27
R; T=0.03/0.17 09.41.23

exec fortclgo badone
09.41.36 FORTRAN BADONE
    0002           FORMAT (' A = ?')
                          $
           01) IEY002I   LABEL
    0003             RED (5,20) A
                       $ $
           01) IEY001I ILLEGAL TYPE        02)  IEY013I SYNTAX
                              IEY022I    UNDEFINED LABEL
      10
  !!! E(00008) !!!
R; T=0.36/1.02 09.42.03

edit fortclgo exec
EDIT:
change /&1/&1 &2 &3 &4 &5/ * g
FORTRAN  &1 &2 &3 &4 &5
LOAD  &1 &2 &3 &4 &5 (XEQ)
EOF:
file
R; T=0.11/0.65 09.43.00
```

```
exec fortclgo mainone
09.43.19 FORTRAN MAINONE
09.43.28 LOAD    MAINONE   (XEQ)
EXECUTION BEGINS...
 A = ?
1.9
 A =    1.900 X =          3.610
R; T=0.64/2.27 09.44.10


CP   <-----------ATTN was hit once to enter CP.
query user
14 USERS, 00 DIALED

query user names
LOVE    - 044,SEYMOUR-02A,OPERATOR-009,MEYER  -045,
ROSATO - 024,NEWSON -040,LEVEY   -027,BOYD   -028,
DJL     - 056,BURR   -062,SHIELDS -057,SCHUPP -055,
EDNA    - 043,CSC1   -026

query logmsg
CP WILL BE UP 24 HOURS A DAY

begin   <-----------------  BEGIN returns control to CMS.
CMS

logout
T=5.49/20.53 10.24.42
CP ENTERED, REQUEST, PLEASE.
CP

logout
CONNECT= 02.50.77  VIRTCPU= 000.05.49  TOTCPU=000.20.54
LOGOFF AT 10.25.06 ON 11/27/69
```

CP-67 TERMINAL USAGE

The conversational input/output device used to access the CP-67/CMS system is referred to as a "terminal" and is operated by a "user" who types information that is transmitted either by telephone line or by permanently-connected wiring to a computer, where the information is received and processed by the system. In addition to receiving and processing information, the system may cause information to be typed out at the terminal. Information typed from the terminal keyboard by the user is called "input"; that typed out at the terminal by the system or by a user program is called "output".

Any one of four terminals may be used to access the CP-67/CMS system. These are the IBM 2741 Communication Terminal, the IBM 1050 Data Communications System Terminal, the Type 33 Teletype Terminal, and the Type 35 Teletype Terminal. Any of these terminals may be connected to the computer by direct wiring or by telephone line. If the terminal is not directly wired to the computer, a data-phone is placed near the terminal keyboard, and must be used to dial an installation-specified number in order to establish a connection with the computer. The procedure for using a data-phone is described under "CP Login" in the "Terminal Usage-Logging Procedures" section.

Terminals which are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on such terminals.

2741 CHARACTERISTICS

The IBM 2741 Communication Terminal consists of an IBM Selectric typewriter mounted on a typewriter stand. The stand includes the electronic controls needed for communications, a cabinet for mounting a data-phone, a rack for mounting a roll of paper, and a working surface. For use with the CP/CMS system, the 2741 should be equipped with the Transmit Interrupt and the Receive Interrupt features.

The 2741 has two modes of operation: communicate mode and local mode. The mode of the terminal is controlled by the terminal mode switch, which is located on the left side of the typewriter stand. When in local mode, the terminal is disconnected from the computer. It then functions as a typewriter only, and no information is transmitted or received. When in communicate mode, the terminal may be connected to the communications line to the computer. The power switch on the right side of the keyboard must be set to ON before the terminal can operate in either communicate or local mode. The procedure for establishing connections with the computer and the terminal switch settings which should be used are discussed below under "2741 Initiation Procedures".

Either of two 2741 keyboard configurations may be used in accessing the CP/CMS system. These are the PTTC/EBCD configurations (shown in Figure 1) and the standard Selectric configuration (shown in Figure 2). On either keyboard, the alphameric and special character keys, the space bar, power switch, the SHIFT, LOCK, TAB, tab CLR SET, and MAR REL keys all operate in the same way as standard Selectric typewriter keys.

On most 2741 terminals, the space bar, backspace, and hyphen/underline keys have the typamatic feature. If one of these keys is operated normally, the corresponding function occurs only once. If the key is pressed and held, the function is repeated until the key is released. The RETURN and ATTN keys have special significance on the 2741 keyboard.

> The RETURN key is hit to signal the termination of each input line. When RETURN is hit, control is transferred to the system, and the keyboard is locked until the system is ready to accept another input line.
>
> The ATTN key is used to generate an attention interrupt. It may be hit at any time (since it is never locked out) and causes the keyboard to be unlocked to accept an input line. Refer to "Attention Interrupt" for a discussion of the transfer between environments that occurs when an attention interrupt is generated.

The 2741 paper controls (such as the paper release lever, line-space lever, impression control lever, etc.) are identical to the corresponding controls on an IBM Selectric typewriter and operate accordingly.

Any invalid output character (one which cannot be typed by the terminal and for which no keyboard function, such as tab or carriage return, exists) appears in terminal output as a space. For a further discussion of 2741 characteristics, refer to the 2741 component description manual (GA24-3415).

2741 INITIATION PROCEDURES

The steps for preparing the 2741 for use are described below. After these steps have been performed, log in.

1. Set the terminal mode switch located on the left side of the typewriter stand to LCL. This ensures that the terminal is disconnected from the computer.

2. After making sure that the terminal is plugged in, turn the power on by pressing the ON portion of the terminal power switch at the right side of the keyboard.

3. Check to see that the margin stops, which are located on the typing guide just above the keyboard, are set at the desired positions (normally 0 and 130). If so, proceed to step 4. To reset a margin stop, push it in, move it to the desired position, and release it.

4. Check that the tabs are set at the desired intervals by tabbing an entire line using the TAB key. If the settings are satisfactory, proceed to step 5. Note that these tab settings do not govern the internal positioning of input characters. For a discussion of internal tab settings, refer to EDIT. If the tabs are to be reset, position the typing element to the right margin, press and hold the CLR portion of the tab control key, and hit the RETURN key. This clears all previous tab settings. New settings may be made by spacing the typing element to the desired locations and pressing the SET portion of the tab control key. After tab stops have been set for the entire line, hit RETURN to position the typing element at the left margin.

5. Set the terminal mode switch on the left side of the typewriter stand to COM. The terminal is now ready for use.

1050 CHARACTERISTICS

The IBM 1050 terminal is composed of the 1051 Control Unit and a 1052 Printer-Keyboard. The 1051 Control Unit includes the power supplies, printer code translator, data channel,

and control circuitry needed for 1050 operation. To be used with the CP/CMS system, the 1051 should be equipped with the Time-Out Suppression and the Transmit Interrupt and Receive Interrupt special features. The 1052 keyboard is similar in appearance to the standard IBM typewriter keyboard. Figures 3 and 4 illustrate the 1050 switch panel and keyboard. The alphameric and special character keys, the space bar, LOCK, SHIFT, and TAB keys, and the paper controls operate in the same way as those on a standard IBM typewriter. The following keys are of special significance on the 1052 keyboard:

RETURN. If the Automatic EOB special feature is included on the terminal being used, and if the EOB switch on the switch panel is set to AUTO, the RETURN key may be used to terminate an input line. Otherwise, (if the Automatic EOB special feature is not available on the terminal being used, or if EOB on the switch panel is set to MANUAL) the character transmitted when RETURN is hit is considered part of the input line.

ALTN CODING. This key, when pressed and held while one of the other keys is hit, originates a single character code such as restore, bypass, reader stop, end of block (EOB), end of address (EOA), prefix, end of transaction (EOT), or cancel. Note that input lines from 1050 terminals not equipped with the automatic EOB special feature must be terminated by pressing the ALTN CODING key and holding it down while hitting the 5 key. This procedure causes a carriage return at the terminal.

RESET LINE. Hitting this key (at the left side of the keyboard) causes an attention interrupt (provided the terminal is equipped with the Transmit Interrupt special feature). The RESET LINE key may be hit at any time, since it is never locked out, and causes the keyboard to be unlocked to accept an input line. Refer to "Attention Interrupt" for a discussion of the transfer between environments which occurs when an attention interrupt is generated.

RESEND. This key and its associated light (both located on the right of the keyboard) are used during block checking. The light comes on when an end-of-block character is sent by the terminal; it is turned off when receipt is acknowledged by the system. If the light remains on, indicating an error, RESEND may be hit to turn off the light, and the previous input line may then be reentered. While the light is on, no input is accepted from the keyboard.

LINE FEED. This key causes the paper to move up one or two lines, according to the setting of the line space lever, without moving the typing element.

DATA CHECK. This key should be hit to turn off the associated light (to its left), which comes on whenever a longitudinal or vertical redundancy checking error occurs, or when power is turned on at the terminal.

Any invalid output character (one which cannot be typed by the terminal and for which no keyboard function, such as tab or carriage return, exists) appears in terminal output as a space. For further information on the characteristics and handling of the 1050 terminal, refer to the 1050 reference digest (GA24-3020).

1050 INITIATION PROCEDURES

The procedure for preparing the 1050 for use are described below. When these steps have been performed, log in.

1. After making sure that the terminal is plugged in, set the panel switches (shown in Figure 3) as follows:

Switch	Setting
SYSTEM	ATTEND
MASTER	OFF
PRINTER1	SEND REC
PRINTER2	REC
KEYBOARD	SEND
READER1	OFF
READER2	OFF
PUNCH1	OFF
PUNCH2	OFF
STOP CODE	OFF
AUTO FILL	OFF
PUNCH	NORMAL
SYSTEM	PROGRAM
EOB	see below
SYSTEM	(up)
TEST	OFF
SINGLE CY	OFF
RDR STOP	OFF

If an EOB switch appears on the terminal, it may be set to either AUTO or MANUAL. If it is set to AUTO, the RETURN key may be used to terminate an input line. If the EOB switch is set to MANUAL, or if it does not appear on the terminal, all input lines must be terminated by hitting the 5 key while the ALTN CODING key is pressed and held down.

2. Check to see that the margin stops--the two blue indicators invisible in the transparent strip just below the switch panel--are set as desired (normally at 0 and 130). If so, proceed to step 3. To change margin settings, set

the PRINTER1 and KEYBOARD switches to HOME. Turn power on at the terminal by setting the mainline switch to POWER ON. Move the typing element to the center of the line by spacing or tabbing. Turn power off at the terminal. Lift the top cover of the 1052 and tilt down the hinged portion of the front panel. Press the blue margin indicators toward the back of the 1052 and slide them to the new locations. Return the hinged panel to its original position and close the top cover.

3. Check the tab settings by setting PRINTER1 and KEYBOARD switches to HOME, turning power on at the terminal, positioning the typing element at the left margin, and hitting the TAB key repeatedly. If the tab settings are satisfactory, proceed to step 4. Note that terminal tab settings do not govern internal positioning of input characters. For a discussion of internal tab settings, refer to EDIT. If the tabs are to be reset, position the typing element to the right margin. Lift the tab setting switch, labeled CLR/SET, and hold it while hitting the RETURN key. This clears all previous tab settings. New settings may be made by spacing the typing element to the desired locations and then pressing down on the tab setting switch. After tab stops have been set for the entire line, hit the RETURN key to position the typing element at the left margin. Turn off power at the terminal.

4. Reset the PRINTER1 switch to SEND REC and the KEYBOARD switch to SEND.

5. Turn the mainline switch to POWER ON and continue with the Login procedure.

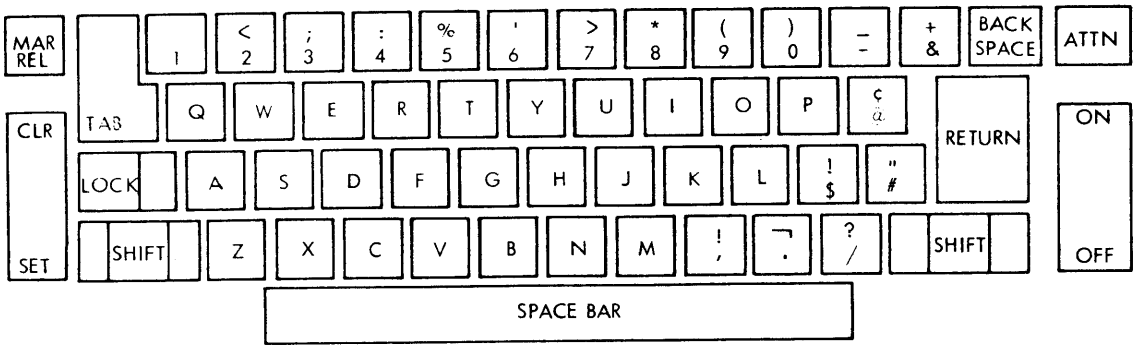
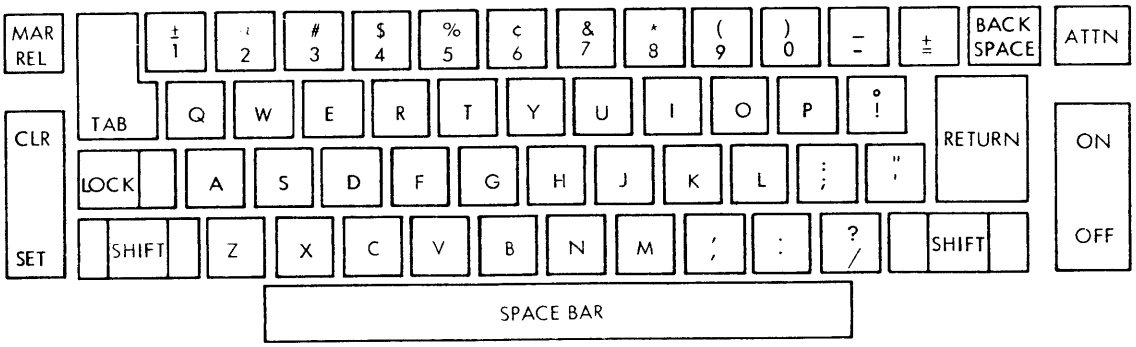


Figure 1. IBM 2741 Keyboard (PTTC/EBCD Configuration)



Note: When this keyboard and associated print elements are specified the mechanical changes in the keyboard mechanism determine the line code assignments of the graphic characters. These arrangements are not compatible with the assignments provided by the use of the PTTC/BCD and PTTC/EBCD keyboards and associated print elements (see Code Chart, Figure 6).

Figure 2. IBM 2741 Keyboard (Standard Selectric Configuration)

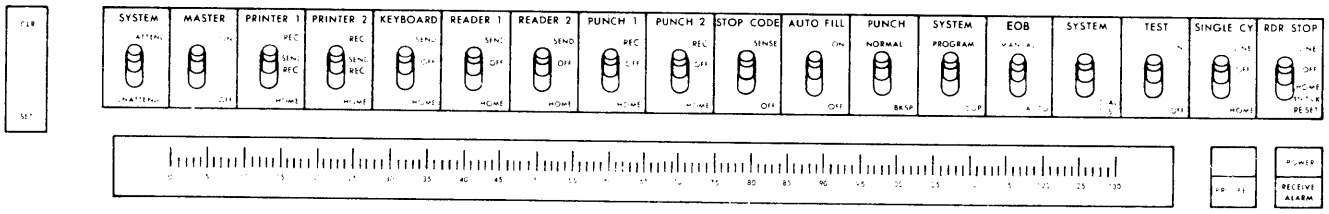


Figure 3. IBM 1052 Switch Panel

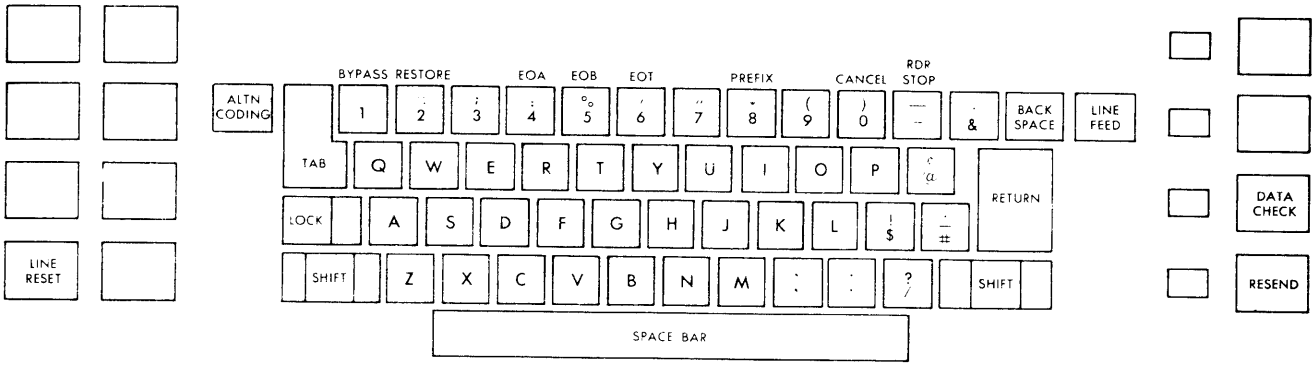


Figure 4. IBM 1052 Keyboard

TYPE 33 TELETYPE CHARACTERISTICS

The KSR (Keyboard Send/Receive) model of the Teletype Type 33 terminal is supported by CP-67. The Type 33 KSR includes a typewriter keyboard, a control panel, a data-phone, control circuitry for the teletype, and roll paper. The Type 33 KSR keyboard contains all standard characters in the conventional arrangement, as well as a number of special symbols. All alphabetic characters are capitals. The SHIFT key is used only for typing the "uppershift" special characters. The CTRL key (Control key) is used in conjunction with other keys to perform special functions. Neither the SHIFT nor CTRL key is self-locking; each must be depressed when used.

In addition to the standard keys, the keyboard contains several non-printing keys with special functions. These function keys are as follows:

LINE FEED generates a line-feed character and moves the paper up one line without moving the printing mechanism. When the terminal is used offline, the LINE FEED key should be depressed after each line of typing to avoid overprinting of the next line.

RETURN is the carriage return key and signifies the physical end of the input line.

REPT repeats the action of any key depressed.

BREAK generates an attention interrupt and interrupts program execution. After breaking program execution, the BRK-RLS button must be depressed to unlock the keyboard.

CNTRL is used in conjunction with other keys to perform special functions. The tab character (Control-I) acts like the tab key on the 2741. Control-H acts like the backspace key on the 2741. Control-Q and Control-E produce an attention interrupt like BREAK if the teletype is in input mode. Control-S (X-OFF) and Control-M act as RETURN. Control-D (EOT) should not be used as it may disconnect the terminal. Control-G (bell), Control-R (tape), Control-T (tape), and all other Control characters are legitimate characters even though they have no equivalent on the 2741.

HERE IS and RUBOUT are ignored by CP-67.

ESC (ALT MODE on some units) is not used by CP-67, but generates a legal character.

The control panel to the right of the keyboard contains six buttons below the telephone dial, and two lights, a button, and the NORMAL-RESTORE knob above the dial. The buttons and lights are as follows:

ORIG (ORIGINATE). This button obtains a dial tone before dialing. The volume control on the loudspeaker (under the keyboard shelf to the right) should be turned up such that the dial tone is audible. After connection with the computer has been made, the volume can be lowered.

CLR (CLEAR). This button, when depressed, turns off the typewriter.

ANS (Answer). This button is not used by CP-67.

TST (TEST). This button is used for testing purposes only.

LCL (Local). This button turns on the typewriter for local or offline use.

BUZ-RLS (Buzzer-Release). This button turns off the buzzer that warns of a low paper supply. The light in the BUZ-RLS button remains on until the paper has been replenished.

BRK-RLS (Break-Release). This button unlocks the keyboard after program execution has been interrupted by the BREAK key.

REST. This light is not used by CP-67.

NORMAL-RESTORE. This knob is set to NORMAL, except to change the ribbon, in which case the knob is twisted to the OUT-OF-SERV light. The knob is then set to RESTORE and returned to NORMAL when the operation has been completed.

OUT-OF-SERV (Out of Service). This light goes on when the NORMAL-RESTORE knob is pointed to it for ribbon changing.

Most teletype units have a loudspeaker and a volume control knob (VOL) located under the keyboard shelf. The knob is turned clockwise to increase the volume.

TYPE 35 TELETYPE CHARACTERISTICS

The KSR (Keyboard Send/Receive) model of the Teletype Type 35 terminal is supported by CP-67. The Type 35 KSR, like the Type 33 KSR, includes a typewriter keyboard, a control

panel, a data-phone, control circuitry, as well as roll paper. The Type 35 has basically the same features as the Type 33. The additional features of a Type 35 are the following:

LOC-LF (Local/Line Feed). This button operates as the LIND FEED button without generating a line-feed character. It is used along with the LOC-CR.

LOC-CR (Local/Carriage Return). This button returns the carrier as RETURN does without generating an end-of-line character. LOC-CR is normally used only to continue a line of input to the next line.

LOC-BSP (Logical/Backspace). This button generates a character but it has no meaning with the KSR model.

BREAK. This button generates an attention interrupt and interrupts program execution. After execution has been interrupted, BRK-RLS, and then the K buttons must be depressed to unlock the keyboard.

K (Keyboard). This button unlocks the keyboard and sets the terminal for page copy only.

Most Type 35 terminals have a volume control knob (SPKR VOL) for the loudspeaker located to the right of the keyboard. Turning the knob clockwise increases the volume.

A column indicator at the upper right of the keyboard indicates the column that has just been printed. When the LOC-CR key is used, no end of line is recorded and the column indicator does not reset.

A red light to the right of the column indicator warns the user that the carrier is approaching the right margin.

CP-67/CMS CONVENTIONS

LOGGING PROCEDURES

This section describes the procedures which must be performed at the terminal to begin and to terminate use of the CP-67/CMS system. For the procedures of connecting a user to a multiaccess system such as RAX or APL, refer to "Dialing a Multiaccess System". Before the facilities of the CP-67/CMS system are made available to a user, he must identify himself to the Control Program by giving his userid and his password (two identifiers which are assigned to him at the time he is authorized to use the system). This identification procedure is referred to as CP Login. When CP Login is completed, a console function may be issued to initialize CMS, as described below.

Note. During the LOGIN procedure CP-67 uses the line time-out feature when reading the userid and password. If the user fails to type any character for 28 seconds, the line will time-out and be disabled.

When the user has completed his use of the system, he signals this fact by issuing a "logout" to the Control Program. The period between CP Login and CP Logout is referred to as a terminal session.

CP Login

After the terminal has been prepared for use (as described under "Terminal Characteristics") the procedure described below must be performed in order to gain access to the CP-67/CMS system. (Note that input may be entered in either uppercase or lowercase. Uppercase is used below to indicate words which must be typed as they are shown; lowercase indicates fields whose contents may vary.)

1. A communications line to the computer must be established. If the terminal is directly wired to the computer this is automatic, and you may proceed to step 2. If the terminal is a Teletype 33 or 35, depress the ORIG button, make sure the dial tone is audible, and then dial the installation-specified number and proceed to step 2; the ORIG button is lighted at this point--if the light goes out during the terminal session, this CP Login procedure must be repeated. Otherwise, a data-phone is placed near the terminal and should be used to establish a communication line with the computer as follows: After making sure that the plug from the data-phone is connected to the walljack, press the button labeled TALK, lift the receiver, and dial the installation-specified number. When a continuous tone is heard, press the button labeled DATA and replace the receiver. The DATA button should now be lighted, and

remains lighted as long as the terminal remains connected to the computer. If this light goes out at any point during the terminal session, the CP Login procedure must be repeated.

2. The system acknowledges that a communication line has been established by typing one of the following messages:

```
CP-67 ONLINE    xxxxxxxxxxxx

xxxxxxxxxxxx    CP-67 ONLINE

CP-67 ONLINE
```

The first message is typed if the terminal is a 1052 or 2741 equipped with an EBCD character set. If the second message is typed, the 2741 has a standard Selectric or correspondence character set. In either case, the xxxxxxxxxxxx portion of the message consists of meaningless characters and should be ignored. If the terminal is a Teletype Type 33 or 35, the third message is typed.

3. At this point the system must be notified that someone wishes to use the terminal. To do this, hit ATTN once. On the Teletype 33 or 35, hit BREAK and then BRK-RLS.

4. The system responds by unlocking the keyboard on a 2741 or 1052 or waiting for input on the Teletype 33 or 35.

5. Identify yourself to the system by typing LOGIN userid, followed by a carriage return, where userid is your user identification.

Note. The LOGIN and userid cannot be edited using character-delete or line-delete symbols.

6. The system responds with one of the following messages:

```
ENTER PASSWORD:
```

This message indicates that your user identification has been accepted. Proceed to step 7.

```
RESTART
```

If this message is typed, the word LOGIN has been entered incorrectly. Return to step 5 and retype the input line.

```
LOGGED IN ON DEV xxx
RESTART
```

This message indicates that another user with the same userid is logged on at the terminal whose address is xxx. You will not be able to log in with the same userid until the other user has logged off.

USER NOT IN DIRECTORY.
RESTART

If this message is typed, an invalid userid has been specified. Return to step 5 and log in again.

MAX NO. OF USERS EXCEEDED
LOGGED OUT AT xx.xx.xx ON xx/xx/xx *** BY OPERATOR***

If the keyboard unlocks or CP-67 waits for input, return to step 5. If the message is typed, the system is already servicing the maximum number of users and the login procedure is terminated. In this case wait for a few minutes, and then try again by returning to step 1.

UPDATING DIRECTORY
RESTART

The CP-67 system directory is being updated. In this case, wait a few minutes, and then try again by returning to step 5.

7. Type your password, followed by a carriage return; the password may be edited. If the 2741 terminal is equipped with the Print Inhibit feature, the password is not typed at the terminal as the keys are hit. The Print Inhibit feature applies only to the typing of a password. If the terminal is a Teletype 33 or 35, three lines of characters are overprinted before you are allowed to enter your password.

8. At this time, if there are any cards in the virtual card reader or output for the printer or punch, the message

FILES:- xx RDR, xx PRT, xx PUN

is typed. If the CP operator has set any log messages for the day, they are typed also.

9. The system responds with one of the following messages:

READY AT xx.xx.xx ON xx/xx/xx

where xx.xx.xx is the time of day and xx/xx/xx is the date. This message indicates that the password has been accepted and the CP log in procedure is completed. The Control Program environment has been entered, and any console function may be issued. To initialize CMS, proceed to step 11. To initialize any other operating system proceed to step 10.

PASSWORD INCORRECT.
RESTART

If this message is typed, an invalid password has been specified and the log in procedure is repeated. Return to step 5.

10. Any operating system can now be loaded into the virtual machine. To load in CMS, go to step 11. To load in another operating system, issue the IPL console function to CP-67, specifying the device from which the system is to loaded. For example, IPL 293 or IPL 00C. If the device that is IPL'ed contains an operating system (such as OS/360), your terminal becomes the operator's console. For information on running OS under CP-67, see "OS/360 in a CP-67 Virtual Machine", by C. I. Johnson, IBM Cambridge Scientific Center Report 320-2035, Cambridge, Massachusetts, March 1969.

CMS Initialization

11. To initialize CMS, issue the console function

```
IPL 190
```

or

```
IPL CMS
```

followed by a carriage return. This causes a copy of the CMS nucleus to be brought into core from disk.

12. The message

```
CMS...VERSION nn LEVEL mm
```

where nn is the version level and mm is the modification level, is typed, and the keyboard is unlocked. The CMS Command environment has control at this point, and any CMS command may be issued.

CP Logout

When the user has finished using the system and wishes to end his terminal session, he should do so by logging out from the Control Program. If the user is not already in the Control Program environment at the time he wishes to log out, he may enter this environment by hitting ATTN once. The keyboard is unlocked and the user types LOGOUT, followed by a carriage return. The system responds with

```
CONNECT=hh.mm.ss VIRTCPU=mmm.ss.hs TOTCPU=mmm.ss.hs
LOGOUT AT xx.xx.xx ON xx/xx/xx
```

and the connection to the computer is lost. The connect time is in hours, minutes, and seconds; the virtual CPU, and total CPU times are in minutes, seconds, and hundredths of a second. The logout procedure is then completed, and the user may turn power off at the terminal.

If the user desires to end his terminal session, but not lose the connection with the computer so that another user may log in from the terminal, the user types LOGOUT

anything, followed by a carriage return. The "anything" must be at least one character or any combination of characters. The connection with the computer is not lost and the CP-67 ONLINE message is typed out for the next user to log in, as in step 2.

DIALING A MULTIACCESS SYSTEM

This section describes the procedures which must be performed to connect a user to a system that provides multiterminal facilities, such as APL or RAX. The process of placing a user into a multiaccess system is referred to as "dialing". The system to be dialed into must be logged onto CP-67 (as in the logging procedures writeup) with some 2702 or 2703 lines available and enabled before the connection can be made. When the connection is made, dialing has been completed, and the terminal is under the control of the system dialed into; consequently, the user is not known to CP-67 as a regular logged in user but as a dialed user.

When the user has completed his use of the multiaccess system, he should log out of that system in the appropriate manner; when that multiaccess system issues a "disable" command for that terminal, the terminal will be free for another user to login to CP-67/CMS or to dial a multiaccess system.

Dialing

After the terminal has been readied for use (as described in "Terminal Characteristics"), the procedure described below must be performed to gain access to a multiaccess system. (Note that input may be entered in either uppercase or lowercase. Uppercase is used below to indicate words which must be typed as they are shown; lowercase indicates fields whose contents may vary.)

1-4. These steps are, as for "CP Login", described several pages earlier.

5. Specify the multiaccess system to which you wish to gain access by typing

```
DIAL system
```

followed by a carriage return, where "system" is the userid of the multiaccess system.

Note. DIAL and system cannot be edited using character-delete or line-delete symbols.

6. The system then responds with one of the following messages

```
...connected...
```

This message indicates a connection has been made between the terminal and the multiaccess system, and the terminal is now under control of that system. Further responses will be those of the multiaccess system, as the user cannot get to

CP-67 to issue console functions.

```
system ALL LINES BUSY
RESTART
```

There are no 2702 or 2703 lines available on system. The lines may not be available for any one of the three following reasons: 2702 or 2703 lines are not defined in the virtual machine, the virtual lines are not enabled by system, or all of the lines are in use.

```
system NOT AVAILABLE
RESTART
```

The system being dialed is not logged in to CP-67.

```
system LINES NOT READY
RESTART
```

The system has not issued an enable for the 2702 or 2703 lines.

```
system NO DIAL LINES
RESTART
```

The system has no 2702 or 2703 lines in its virtual machine description.

Disconnecting

The dialed terminal remains connected to system until one of the three following events occur:

(1) system issues a disable for that terminal. This is usually brought about by logging out of system in the correct manner.

(2) system issues the CP console function DETACH, specifying the terminal address.

(3) system logs out of CP-67.

When the terminal is disconnected from system, the following message is typed out:

```
CP-67 LOGOUT
```

The terminal can now be used to log in to CP-67, or to dial into a multiaccess system again.

GENERAL TYPING CONVENTIONS

The typing conventions described below should be observed when entering input to the CP-67/CMS system from a 2741, 1050, or a Teletype 33 or 35 terminal.

Input may be entered in either uppercase or lowercase.

When the keyboard is unlocked on the 2741 or 1050, the terminal is ready to accept input. The keyboard remains unlocked on the teletype, therefore a > (greater than sign) is typed at the left margin when the teletype is ready to accept input. If the user types too soon on the teletype, an interrupt may occur which will probably cause the user to go back to CP; if this happens, type BEGIN to return to where you were previously.

The character-delete symbol (@) may be used to delete the preceding character in the input line. n character-delete symbols delete the preceding n characters in the input line and themselves. Exception: This feature does not apply with the K-level commands or the RT command. The character-delete symbol can be redefined by the CHARDEF command for use in CMS; it can never be redefined for CP.

The line-delete symbol, which is the ¢ on the 2741 or 1050, and the shift K (left bracket) on the teletype, may be used to delete all characters in the current input line and itself. A line-delete symbol (¢) cannot be deleted by a character-delete symbol (@). Exception: This feature does not apply with the K-level commands or the RT command. The line-delete symbol can be redefined by the CHARDEF command for use in CMS; it can never be redefined for CP.

An input line may be a maximum of 130 characters in length. Any line longer than 130 characters--including delete symbols, blanks, and the tab character--is truncated to 130 characters. On the teletype 33, an input line can only contain a maximum of 72 characters.

An input line from the 2741 or teletype is terminated by hitting RETURN. To terminate an input line from the 1050, hit the 5 key while holding down the ALTN CODING key, unless the 1050 is equipped with the Automatic EOB special feature. If this special feature is available and the EOB panel switch is set to AUTO, input lines may be terminated by hitting RETURN.

Input lines, can contain a number of logical input lines which are separated by the line-end character (#). Each call to read a line from the terminal returns a logical input line. Subsequent calls to read a line from the terminal return the logical input line which was typed

following the previous logical input line. For example, the single input line

```
FORTRAN ABLE # $ ABLE
```

causes the file ABLE FORTRAN to be compiled, loaded, and executed. The single input line to the EDIT environment

```
T # L /BUFT/ # C /FT/FFER/
```

causes the characters BUFT to be located and changed to BUFFER. The line-end character can be changed with the LINEND command for use in CMS; it cannot be redefined for CP.

An output line on the 2741 and 1050 can be a maximum of 130 characters. Any line longer than 130 characters causes overprinting at the right margin. On the teletypes, lines longer than 72 characters in length are printed as two lines; the first line cuts off after the 71st character, and an up arrow is printed at the end of the line to indicate continuation.

Illegal output characters appear in terminal typeout as spaces. An illegal output character is one which cannot be typed and for which no keyboard function, such as a carriage return or a tab, can be generated.

On the Teletype 33 or 35 terminal, the arrow translates to underscore (_), the backslash translates to a not sign (¬), and the uparrow to a vertical bar (|).

One or more blanks are used to delimit the fields of an input line to the CMS command environment.

ATTENTION INTERRUPT

After a phone connection with the computer has been made and the message "CP-67 Online" has been printed, pressing the attention key causes the CP login procedure to begin, or the dialing of a multiaccess system to begin.

The user's machine may be interrupted (stopped) and the terminal readied for input at any time by pressing the attention key (marked ATTN) on a 2741, hitting the RESET button on a 1050, or the BREAK key on the Teletype 33 or 35. This causes control to pass to CP provided that the Control Program was not already in control. CP console functions can then be issued.

After a previous attention (pressing ATTN) causes control to pass from CMS to CP, a subsequent attention passes control back to CMS. If CMS previously had control and was reading a line from the terminal (that is, the keyboard was unlocked), the CMS program is restarted and the keyboard is unlocked again. If CMS previously had control and was not reading a line from the terminal (that is, the keyboard was locked), the interruption permits a single line of input to be entered and stacked. Stacked lines of input are used on successive calls to read a line from the terminal until there are no more stacked lines. Input is then taken directly from the terminal. After a line to be stacked is entered, CMS continues from the environment which was last interrupted. Any number of lines may be input and stacked in this way.

When entering two attentions to the system to stack input, ATTN should not be pressed the second time until the keyboard has been unlocked in response to the first attention. If the second attention is entered before the keyboard is unlocked, it is ignored.

CMS FILE CONVENTIONS

One of the purposes of CMS is to provide the user with various file-handling facilities. Files to be used under CMS may be stored on disk, cards, or magnetic tape. However, most CMS commands assume that files are stored on disk. This means that files stored on media other than disk must be transferred to disk before many of the CMS commands can be issued for them. The commands which deal with transferring files between disk and other media are discussed under "File Creation, Maintenance, and Manipulation".

Conventions given in this section apply to disk files only.

Disk Facilities

Disk areas are available to each CMS user for storing information. A user may have up to five read/write disks and one read/only (that is, the system) disk attached to his CMS virtual machine at any one time. These areas are referred to as the user's disks although the size of each area seldom constitutes an entire physical disk. The sizes of a user's "mini" disks are assigned by the system administrator at the time he establishes that person as an authorized user of the CP-67/CMS system. These assigned sizes are based on the amount of disk space available and the amount which the user is likely to require. Assigned disk sizes may vary among users.

The logical names of the disks a user may have are P, T, A, B, S, and C. This order is normally the standard order of search for CMS files. All users normally have a P (191) and an S (190) disk. The P disk contains user files retained between terminal sessions until the user erases them either collectively or singly. The S disk (that is, system disk) contains the CMS nucleus of which each user receives a copy, and the disk resident portion of CMS which is normally shared by all users. The system disk is read-only--any attempt to write on it is denied and causes an error message to be typed to the user.

The T (192) disk contains information that is retained only from the time it is created until the user terminates his terminal session. The A, B, and C disks contain information that is retained between terminal sessions as does the P disk.

Before a user can access the P, T, A, B, or C disk for the first time, he must format each one by issuing the FORMAT command in CMS. If the disks are not properly formatted, I/O errors occur.

Information stored on disk is organized into files. Files on the system disk are referred to as system files.

File Identifiers

Each file must have a unique identifier, which is composed of a filename, a filetype, and a filemode. This identifier (or a portion of it) is used by the various CMS commands to access user and system files. If a new file is created with an identifier identical to that of an existing user file, the original file is erased.

The filename may be any combination of from one to eight nonblank EBCDIC characters, provided the first character is not a zero or an asterisk. With system files, the filename is the name which is issued by the user in calling a specific command, and it is also the name of the program whose code constitutes that command. Permanent and temporary files may be assigned any filename the user wishes, since filenames in themselves do not have any special implications in CMS.

Filetype may be any combination of from one to eight nonblank EBCDIC characters, provided the first character is not a zero or an asterisk. Certain filetypes imply specific file characteristics to CMS. Filetypes which have specific implications for user files are given in Figure 5. The user may assign any of the filetypes in this figure to any file he wishes, but he should note that the commands which use these filetypes are not successfully executed if the contents of the file are in any form other than that which the assigned filetype implies.

Filetypes	Record Format and Length	Created by These Commands	For Use by These Commands*	Usage
AED	F 80	EDIT, OFFLINE	-	AED source code
ALPHABET	F 80	MAPPRT	-	Alpha listing of nucleus load map
ALPHANUM	F 80	MAPPRT	-	Alpha and numeric listing of nucleus load map
ASP360	F 80	OFFLINE, EDIT	MACLIB	Assembler language macros
ASl130	F 80	OFFLINE, EDIT	-	1130 Assembler source code
BRUIN	F 80	BRUIN	-	Bruin source code
COBOL	F 80	OFFLINE, EDIT	-	Cobol source code
COPY	F 80	EDIT, OFFLINE	MACLIB	Assembler source code
CVTUT1		CVTFV	-	Utility file created by CVTFV
DATA	F 80	EDIT, OFFLINE	-	User data files
DAxx	F	FORTRAN	-	Direct access Fortran files
DIAG	F 132	ASSEMBLE		Assembler diagnostics
EXEC	F 80	EDIT, OFFLINE, LISTF	EXEC, $	CMS and/or EXEC commands
FILE	V or F	EDIT, CEDIT, OFFLINE	-	User data files or utility file created by EDIT and CEDIT
(FILE)	V or F	COMBINE, DISK, TAPE	-	Utility file created by either COMBINE, DISK or TAPE
FLOW	F 80	EDIT, OFFLINE	-	Flowchart input data
FT01F001	F 80	FORTRAN	-	Fortran input/output file
FT02F001	F 80	FORTRAN	-	Fortran input/output file
FT03F001	F 80	FORTRAN	-	Fortran input/output file
FT04F001	F 80	FORTRAN	-	Fortran input/output file
FT05F001	F 80	FORTRAN	-	Fortran output file
FT07F001	F 80	FORTRAN	-	Fortran input/output file
FT08F001	F 133	FORTRAN	-	Fortran input/output file
FT09F001	F 140	FORTRAN	-	Fortran input/output file
FT10F001	F 80	FORTRAN	-	Fortran input/output file
FORTRAN	F 80	EDIT, OFFLINE	FORTRAN	Fortran source code
INTER	F 80	UPDATE	-	Utility file created by UPDATE
LISTING	F 121	ASSEMBLE, FORTRAN, PLI	PRINTF, EDIT, OFFLINE	Assembler or compiler output containing source statements and machine code
MACLIB	F 80	MACLIB	ASSEMBLE, GLOBAL	Assembler language macro definition library
MAP	F 72	LOAD, USE, REUSE, $	-	Map of locations of loaded programs (Filename is LOAD)
MAP	F 132	MACLIB, TXTLIB	-	Map of library file
MEMO	F 80	EDIT, OFFLINE	-	Memorandum file - contains upper and lower case data
MODULE	V max:65K	GENMOD	LOADMOD, $	Non-relocatable object code
NUMERIC	F 80	MAPPRT	-	Numeric listing of nucleus load map
PLI	F 80	EDIT, OFFLINE	PLI	PL/I source code
PRINTER	F 132	EDIT, OFFLINE	PRINTF	Printer output and listing files read onto disk
REPS	F 80	EDIT, OFFLINE	-	Replace cards for modifying TEXT files
SCRIPT	V max:132	EDIT, SCRIPT	SCRIPT	Script input and/or output
SNOBOL	F 80	EDIT, OFFLINE	SNOBOL	SNOBOL source code
SPLI	F 80	SNOBOL	SNOBOL	SNOBOL object code
SYSIN	F 80	EDIT,OFFLINE	ASSEMBLE, UPDATE	Assembler language source code
SYN	F 80	EDIT	SYN	Synonyms or abbreviations
SYSUT1	V	ASSEMBLE, PLI	-	Utility file created by Assembler or PL/I
SYSUT2	V	ASSEMBLE, PLI	-	Utility file created by Assembler or PL/I
SYSUT3	V	ASSEMBLE, PLI	-	Utility file created by Assembler or PL/I
TEMPFILE		SNOBOL	-	Utility file created by SNOBOL
TEXT	F 80	ASSEMBLE, PLI, FORTRAN	LOAD, USE, REUSE, $	Relocatable object code
TXTLIB	F 80	TXTLIB	GLOBAL,LOAD,USE, REUSE, $	Library of relocatable object code
..TYPE..	F 80	OFFLINE	-	Utility file created by OFFLINE
UPDATE	F 80	EDIT, OFFLINE	UPDATE	Update control and replacement cards
UPDLOG	F 80	UPDATE	-	Log of applied updates
UTILITY		CNVTFV	-	Utility file created by CNVTFV

* Other commands may use these filetypes; these are only the pertinent commands.

Figure 5. Filetype implication and characteristics

Several of the CMS commands create files for their own use and assign specific filename-filetype combinations to these files. These filename-filetype combinations are listed below, and should not be assigned by the user

filename	filetype	creating command
.TEMP	BRUIN	BRUIN
CMS	EXEC	LISTF
(INPUT1)	FILE	EDIT, CEDIT
(INPUT2)	FILE	CEDIT
(TEMP)	(FILE)	COMBINE
LOAD	MAP	LOAD,$,USE,REUSE
SNOBOL	TEMPFILE	SNOBOL, SPL1
.DUMMY	TXTLIB	TXTLIB
(DISK)	(TFILE)	DISK, TAPE
..NAME..	..TYPE..	OFFLINE
BCDEBC	UTILITY	CNVT26

In addition to the above, note that if a user file is created whose filename and filetype are identical to those of a file on the disk-resident portions of CMS, the standard order of search may access one user's file in place of the system file. Reasons for this are described in "CMS Commands".

The third portion of the identifier, filemode, consists of two characters. The first character is a letter indicating the disk area on which the file resides: S for system disk, P for permanent disk, and T for temporary disk, etc. For system files, the second character is always Y. For user files, the second character is a number from one to six. These numbers have the following meanings, although the restrictions they imply may not currently be implemented in all cases:

1 or 5--file may be written or read.

2 or 6--file is read-only.

3--file may be written or read, but is erased when the file is closed.

4--file is read-only, and is erased when the file is closed.

Files are matched on the three identifiers. If, however, both filename and filetype are explicitly given, the mode letter only is checked (that is, the mode number is ignored on the match). However, if either the filename or filemode is implied with a *, the mode number is matched together with the mode letter.

File Sizes

Files stored on disk are formatted into records 829 bytes long. This formatting is handled internally by CMS, and is not controlled by the user. The maximum CMS file size (assuming that the user's assigned disk area can accommodate it), is 24.358 million bytes, or 65,533 records. If a file consists of a source language program, a size limitation may be imposed by the language in which that program is written, and this size may be smaller than the 24.358 million bytes allowed by CMS. The maximum disk file for user disks is 203 cylinders each. Although there is no inherent limitation to the number of files a user may create, he is limited practically by the sizes of his disk areas. When a user has filled either of these areas, a message to this effect is typed at his terminal. Refer to "Recovery Procedures" for steps to be taken in this case.

A file is "accessed" when any portion of it is read or written. Whenever a file is accessed for the first time by a CMS command or function, the file is automatically opened. Opening in this case consists of making an entry into the user's active file table. All open files are closed by CMS after the successful execution of any CMS command. The user does not need to be concerned with opening and closing files.

Disk Considerations:

Read-only disks. There are several ways a user can force a normally read-write disk (that is, P,T,A,B,C) to be read-only.

If the user LINKs to a disk that is read-only in the CP directory, then he has read-only access to that disk.

A disk may be logged into CMS as a read-only extension of the P disk (for example, LOGIN 196 B,P). This forces the disk to a read-only status, and concatenates the B disk to the P disk in the standard order of search.

A disk may be logged into CMS as a forced read-only disk by making it an extension of itself (for example, LOGIN 195 A,A). A user may thereby protect certain files by forcing them to be read-only.

On read-only disks it is possible to login a portion of the files on that disk. See the LOGIN command for details on how this done.

T-Disk. The T-disk is automatically logged in when the user IPL's CMS if 192 is attached and ready, and the label on that disk is readable and is CMS=T-DISK. If the label is

readable but not CMS=T-DISK, the disk is not logged in. If the label is not readable, an automatic FORMAT T (NOTYPE) is issued.

C-DISK. The C disk's (19C) special usage lies in its ability to be a read-only extension of the system disk by means of a modification to the system at installation time. Thus, if the S disk is full and additional routines are needed as part of the system, they may be placed on the C disk. For example, the C disk could contain an old version of the CMS system disk. Thus it's routines would be used only if they were not present on the more recent system disk.

Formatting disks. The very first time a user logs in to CMS, he should issue a FORMAT command to format his disk areas into the CMS format. For details on FORMAT, see the CMS FORMAT command.

ENVIRONMENT CONVENTIONS

Each input line which is typed at the terminal by a user is transmitted to the CP/CMS system, where it is processed (examined, and accepted or rejected) by a given routine. The particular routine by which input is processed is determined by which portion of the system has control at the time the input line is entered. Each portion of the system to which input may be entered constitutes a unique environment, and only a subset of all possible input is acceptable to any given environment. The following are the environments of the CP/CMS system:

Control Program environment
CMS Command environment
Debug environment
Edit environment
Input environment
Echo environment

In addition to these six specific environments, input may be entered to any other executing program which expects terminal input. These other input-processing programs are grouped into a seventh, Program environment, in which the acceptability of an input line is determined by the executing program.

With the exception of the Program environment, the input-processing routines fall into three main categories: input is received by either the Control Program (CP environment), a central CMS service routine (CMS Command environment), or a particular CMS command (Debug, Edit, Input, and Echo environments). Input lines which are acceptable to the CP environment are referred to as "console functions", since for the most part they simulate functions that can be performed at a System/360 console. Input to the CMS Command environment may be any CMS command. Note that CP console functions can also be issued from the CMS command environment (see CPFUNCTN command).

A certain number of the CMS commands cause environments of their own to be entered. These are the DEBUG, EDIT, and ECHO commands. Lines acceptable to the environments of these commands are referred to as "requests", or merely "input", depending on the particular environment which is entered when the command is issued. The EDIT command causes either one of two environments to be entered. If it is issued for a file which already exists, the editing environment is entered, allowing the contents of the existing file to be examined and modified. If an EDIT command is issued for a file which does not currently exist, the input environment is entered, allowing the file to be created. The Input environment accepts any input typed at

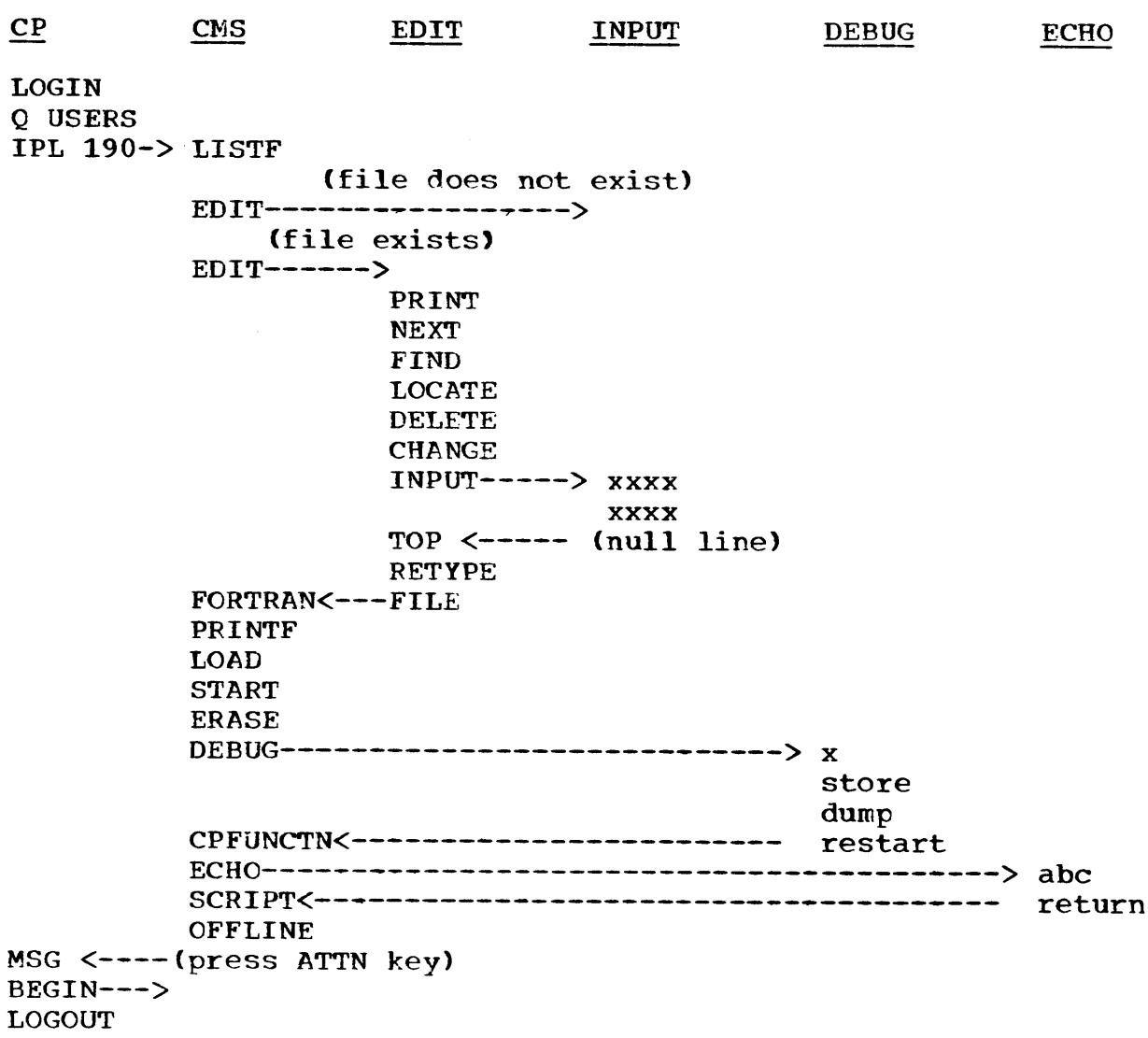
the terminal, and this input becomes a part of the file being created. The Echo environment also accepts any input line, but repeats that line as output in order to test terminal transmisssion. Because no check is made to determine the acceptability of input to these two environments, lines which are acceptable are termed merely input. The Debug and Edit environments, on the other hand, accept only specific input lines, which are referred to as requests.

To verify at any time which environment the user is in, RETURN can be hit.

Various actions by the user cause control to pass from one environment to another. These actions are specified in detail throughout this guide. Note that ATTN can always be used to transfer control to the CP environment from any of the other environments. Hitting ATTN while in the CP environment causes the keyboard to be unlocked, permitting one line of input to be entered. If the line entered is a K-type CMS command (KT, KX, KO) or the RT command, it is executed immediately and control returns to the environment from which CP was entered. If the line entered is not a K-type CMS command or the RT command, the line is stacked to be used as terminal input and control returns to the environment from which CP was entered. As many lines as desired may be stacked in this way to be used in place of successive lines of terminal input.

CP-67/CMS Environments, Commands, and Requests

These are illustrated below.



CMS COMMANDS

The CMS commands provide user facilties for file maintenance and manipulation, execution control, debugging, language processing, and various utility and control operations, and are described below in alphabetical order under these general headings.

These commands may be issued from the terminal or called from user programs. Each command consists of a command name and its operands, if any. Abbreviations have been established which allow the user to specify only as many characters of each command name as uniquely identify that command. In the case of commands with the same leading characters, the more commonly used command has been assigned the shorter abbreviation. For example, A is the abbreviation for the ASSEMBLE command, and AL for ALTER. Any number of additional characters beyond the minimum may be specified in the command name. For this reason, AL, ALT, ALTE, and ALTER all identify the ALTER command.

The following is a list of the minimum number of characters required to invoke CMS commands:

System Commands	Shortest Form
ASSEMBLE	A
ALTER	AL
CLOSIO	CL
CPFUNCTN	CP
DEBUG	DE
EDIT	E
FORTRAN	F
GENMOD	G
LISTF	L
OFFLINE	O
PRINTF	P
SCRIPT	SC
STAT	S
TAPE	T
UPDATE	U

If the user wants to change the abbreviations or to use synonyms, the SYN command can be used to create the command names, (see SYN).

The command name and each of its operands must be separated by one or more blanks. The operands which are valid for each command are discussed under "Format" in each command description, and also in Appendix B. Each command must be specified in a single line of input. The carriage return signals the end of a typical input line. To stack multiple

CMS commands on one line of input, use the line-end character, which is defined as # . The line-end character can be redefined via the LINEND command. CMS commands cannot be continued onto additional lines.

Each CMS command has a corresponding command program which resides in the nucleus, in a transient area, or in the disk-resident portion of CMS. This program is identified by the command word or its abbreviation, which is issued as the leftmost input on the command line.

When a command is issued from the terminal, the user's directories and the system directory are searched in the standard order for a file with the specified filename and a filetype of EXEC. The first file found which meets these requirements has control transferred to it as if "EXEC filename" had been issued. If the EXEC file is not found, a check is made for abbreviations by checking for user-defined synonyms (see SYN) and then standard abbreviations; if a match is found for a synonym or abbreviation, the typed command is expanded to the original CMS command name and the above searching sequence is repeated.

If the EXEC file search is not satisfied from above, the tables of the transient area commands and then the nucleus-resident commands are searched for the corresponding command program. If the program is located in one of the tables, it is assumed to be in core, and control is transferred to it directly by a BALR instruction. If not, a LOADMOD is issued to bring the command program into core. In attempting to locate this program on disk, the user's directories and the system directory are searched for a file with the specified filename (command name) and a filetype of MODULE, indicating that the file is in core-image form. The first file found which meets these requirements is loaded into core and control is transferred to it. If the MODULE file is not found, a check is made first for abbreviations by checking for user-defined synonyms (see SYN), and then for standard system abbreviations; if a match is found for a synonym or abbreviation, the typed command is internally expanded to the original CMS command name, and the above searching sequence is repeated.

This means that the user is able to substitute his own programs for disk-resident commands by creating a core-image file of the program and assigning it a filename identical to that of the command it is to replace. This also means that any user file in core-image form may be called directly as a command, by issuing the filename (and any operands or parameters expected by the program) as an input line to the CMS Command environment.

A brief description of each CMS command is given in Appendix A. Any invalid command, that is, one whose command program does not reside in the CMS nucleus or transient-area and for which an EXEC file or a core-image module cannot be located, is ignored, and the message INVALID CMS COMMAND is typed at the terminal. Operand processing is handled by the individual command programs, and these programs provide all messages dealing with command format.

The format and usage of each of the CMS commands are described in detail in the following sections. The general format for CMS commands follows.

operation	< operands >
command name	one or more operands delimited by spaces field may be blank

The symbols used to represent command formats in this document are described below.

UPPERCASE	information given in capitals must be typed exactly as shown, although it may be entered in either uppercase or lowercase.
lowercase	lowercase information designates the contents of a field, and does not in itself constitute meaningful input.
( )	parentheses must be typed as shown when any of the information appearing within them is specified.
.	a period designates the beginning of a Script control word, and must be typed as shown.
-	a hyphen must be typed where shown, and must not be offset by blanks.
/	a slash denotes any string delimiter, other than blank, which does not appear in the string.
*	an asterisk, specified where shown, indicates the universality of an item or items.

The following are logical symbols only, and should not be typed:

< >	brackets indicate information which may be omitted.
< >< >	successive brackets enclose items which, if specified, may appear in any order.
<< >>	nested brackets indicate items which, if specified, must appear in the order shown.
...	an ellipsis indicates that the preceding item(s) may be repeated more than once in succession.
1 2 3 N	these suffixes indicate first, second, third, and Nth items respectively.
_____	underlining indicates the value which is assumed if none is specified. When no underlined item appears in bracketed < > information, the default value is none.
	Stacked items not enclosed in anything indicate that only one item may be specified.

All parameters for CMS commands are positional unless otherwise stated in the individual command description.

Examples of command usage and each response and error message which may be issued are also given. A response is any message typed at the terminal to indicate the cause of an error return code in register 15, which terminates command execution.

FILE CREATION, MAINTENANCE, and MANIPULATION

File Creation. Facilities are available in CMS for the handling of disk, card, and tape files. Most of the CMS commands, however, require that the files they access be stored on disk. This means that card and tape files must be transferred to disk before many of the commands can be issued for them.

Disk files can be created from terminal, card, or magnetic tape input, or from other disk files. Issuing the EDIT command for a disk file which does not currently exist allows the specified file to be created from terminal input. To create a disk file from card input, the OFFLINE READ command can be used. OFFLINE READ accepts card input in any format.

Card files are created in CMS by requesting that the contents of disk files be punched. The OFFLINE PUNCH command punches out any disk file whose records are 80 characters or less in length.

File Maintenance. Several commands provide facilities for maintaining disk files. UPDATE and EDIT allow any portion of an existing file to be changed, deleted, or added to. UPDATE processes the existing file against an update file, also stored on disk. EDIT allows the contents of an existing disk file to be changed from the terminal. To change the identifier of a disk file without changing its contents, the ALTER command may be used. A file or group of files can be deleted from disk by issuing the ERASE command. CLOSIO is used to signal the completion of output to the card punch or printer from a user program.

File Manipulation. CMS file manipulation consists of copying, combining, moving, splitting, and listing disk files. To copy a disk file, the EDIT or COMBINE commands can be used. COMBINE also creates a new disk file from the contents of two or more existing disk files and may be used to transfer a file between the disk areas. The SPLIT command creates a new disk file composed of the specified portions of an existing disk file or files.

LISTF and PRINTF cause file information to be typed at the console. The LISTF command types the identifiers and sizes of any or all files stored on the user and system disk areas. PRINTF types out all or the specified part of a disk file.

To print the contents of a disk file on the offline printer, the OFFLINE PRINT, OFFLINE PRINTCC, or OFFLINE PRINTUPC commands can be issued. OFFLINE PRINT prints the file with single spacing and CMS headings, the PRINTCC version uses

the first character of each record as a printer carriage control character, and the PRINTUPC version prints the file in uppercase (for MEMO or SCRIPT files when the PN train is on the printer).

ALTER

Purpose:

The ALTER command changes the name, type, or mode of the specified file(s).

Format:

ALTER	ofn	oft	ofm	nfn	nft	nfm
	*	*	*	*	*	*
				=	=	=

ofn oft ofm are the original filename, filetype, and filemode, respectively.

nfn nft nfm are the new filename, filetype, and filemode, respectively.

* An asterisk for ofn means all files with that name, for oft means all files with that type, and for ofm means any read-write disk. An asterisk for nfn, nft, or nfm means no change.

= means the same as before, no change.

Usage:

The name, type, or mode number of files on a user's read-write disk may be changed with the ALTER command. All fields of the command must be specified.

Notes:

a. ALTER does not move files between disks. The mode letter, may not be changed (see the COMBINE command).

b. ALTER may not be used for files on a read-only disk such as the sytem (SY) disk.

Responses:

ALTER gives no response except the Ready message, or an error message and code.

Examples:

a. ALTER BEN SYSIN P5 PROG3 SYSIN P2
The file formerly called BEN is now referenced by the filename PROG3 and is read-only instead of read-write.

b. ALTER JBS LISTING * JLEVEL3 = =
The LISTING file from an assembly of JBS is now referenced by the filename JLEVEL3.

c. ALTER * LISTING P5 = = P2
All files with LISTING type and mode of P5 are changed to mode P2.

Error Messages:

E(00001) OLD SPECIFIED FILE CANNOT BE FOUND
The file to be redesignated is not in the file directory. Check whether it was specified correctly.

E(00002) NEW SPECIFIED FILE ALREADY EXISTS
A file with the new name, type, and mode already exists. Change one of the fields of the new designation. If concatenation with an existing file is wanted, use the COMBINE command.

E(00003) OLD MODE IS ILLEGAL FOR A CHANGE
The original mode specified is invalid--it may be for a read-only disk, or it may not end with a number from one to six.

E(00004) NO CHANGES WERE MADE AT ALL
The new designation specified is the same as the original designation.

E(00005) CHANGE-OF-MODE IS ILLEGAL
An attempt was made to change the mode letter. ALTER does not move files between disks (see the COMBINE command).

E(00006) NEW MODE IS ILLEGAL
The new mode specified is not a valid one or the mode number is not between one and six.

E(00007) INCORRECT "ALTER" PARAMETER-LIST
Name, type, and mode were not specified for both files.

E(00008) SPECIFIED FILE IS IN "ACTIVE" FILE TABLE
A file may not be changed while it is active.

CEDIT

Purpose:

CEDIT creates and makes changes to card image files only.

Format:

```
 ----------------------------
| CEDIT |<filename>  filetype |
 ----------------------------
```

filename is the name of the file to be created or modified

filetype is the type of file being created or modified

Usage:

CEDIT works only with card-image files (fixed-length 80-character records). If CEDIT is issued for larger records (for example, LISTING files), each record is truncated to 80 characters. If CEDIT is issued for variable-length records (for example, SCRIPT files), they are made fixed-length.

CEDIT should only be issued for files too large for EDIT, and then only for card-image files, as EDIT is faster. CEDIT uses work files during editing, whereas EDIT is an in-core editor.

The same requests used for EDIT are used for CEDIT, with the exception of X, Y, and ZONE. The responses are also basically the same. (See the EDIT command for all requests, responses, and error messages).

CEDIT searches all disks for the specified file. If the file is found its mode is saved and CEDIT writes the modified file back to the same disk. If the file is not found, the P disk is assumed.

CLOSIO

Purpose:

CLOSIO notifies the system that I/O operations to an offline unit record device are complete, or that output to an offline device is to be collected before it is actually output to the physical device.

Format:

CLOSIO	READER	OFF
	PRINTER	ON OFF
	PUNCH	ON

OFF collects files output to the specified device, but does not output the files to the physical device.

ON notifies the system that I/O operations to the specified device are complete and output should begin on the physical device. ON is the default.

Usage:

CLOSIO is normally used as a supervisor-supplied function within programs written in assembly language. However, it may be used as a command in cases where user-written programs that include unit record I/O routines terminate abnormally, or do not include a call to CLOSIO, or where concatenation of spooled output is wanted.

CLOSIO either notifies the system of an end-of-file condition for the devices specified or collects output to offline devices. Any combination of the three devices may be specified with the command. Undefined devices are ignored. When CP-67 detects the end-of-file condition, the disk spooling area assigned to the user for the specified device is closed. Printer and card-punch files are queued for actual output. Card-reader input files are erased.

If it is desired to collect output of punch or printer files into one spooled output file per device preceded by only one header record, the CLOSIO command can be issued to the PRINTER or PUNCH with OFF specified. All succeeding output to the OFF device is collected until ON is specified with CLOSIO for that device, at which time the collected files are queued for the physical device.

Notes:

a. CLOSIO is not used after CMS I/O commands.

b. All unit record devices are closed by CP-67 when the user logs out.

c. CP-67 interprets any invalid CCW as a CLOSIO for the device to which it is addressed.

d. If READER, PRINTER, or PUNCH is not specified, all three devices are closed as if ON were specified.

e. If the 'OFF' option has been specified, the device must be explicitely turned 'ON' to be reactivated.

Responses:

None.

Examples:

a. CLOSIO PRINTER READER
Spooling areas assigned to the user for the printer and card reader are closed. The reader area is erased. The printer file is queued for actual output.

b. CLOSIO PRINTER OFF
All further output to the printer is collected and no end-of-file condition is generated.

c. CLOSIO PRINTER ON
An end-of-file condition for the printer is generated; thus the spooling areas assigned to the user for the printer are closed. The printer file is queued for actual output.

Error Messages:

None.

COMBINE

Purpose:

The COMBINE command joins two or more disk files into a single file, moves files between disks, and changes file designations.

Format:

```
| COMBINE | nfn nft nfm ofn1 oft1 ofm1 ... ofnN oftN ofmN |
```

nfn fnt nfm are the filename, filetype, and filemode of the file to be created.

ofn oft ofm are the filename, filetype, and filemode of existing file(s) to be included in the new file.

Usage:

The file to be created and each of the included files must be specified by filename, filetype, and filemode. Input files must have the same record format (fixed or variable-length). Input files of fixed-length records must have the same record length. Any number of input files can be included in the new file, in the order named, but the command must not exceed a single input line.

The output file is created on the specified disk, according to the mode letter of the new file. Input files may be on any disk.

If the new filename, filetype, and filemode are those of an existing file, the old file will be erased when the new file is created. The old file may be among the input files.

Notes:

a. Files may not be copied to the system (SY) disk.

b. As the input files are processed, a temporary work file is created with the identifiers (TEMP) (FILE) mm, where mm is the specified mode of the output file. When processing is completed, this file is given the designation specified for the output file. If an error occurs such that input files are destroyed, records can be retrieved from this work file.

Responses:

None.

Examples:

a. COMBINE FILE DA01 P3 TST1 FILE P5 TST2 FILE P5
The file FILE DA01 P3 is created on the P-disk. It contains all the records of TST1, followed by the records of TST2. TST1 and TST2 are not changed.

b. COMBINE JOBS EXEC T5 JOBS EXEC P5
A copy of the P-disk file JOBS is made on the T-disk.

c. COMBINE JOBA FORTRAN P5 JOBA FORTRAN P5 SUBR1 FORTRAN P5
The file SUBR1 is appended to a copy of JOBA, and the new file replaces the original JOBA.

Error Messages:

E(00001) FILE "filename filetype filemode" NOT FOUND.
The file specified in the message was not found. Files remain as they were before the command.

E(00002) DISK ERROR WHILE READING.
An I/O error occurred, or there is insufficient core space for buffers.

E(00003) DISK ERROR WHILE WRITING.
An I/O error occurred, or the user's allotted disk space is filled.

E(00005) ERROR IN NAME, TYPE, OR MODE OF OUTPUT FILE.
Correct the designation of the output file.

E(00010) INCORRECT PARAMETER LIST
The command format was incorrect. Files were not changed.

E(00011) MODE SPECIFIED FOR OUTPUT FILE IS ILLEGAL.
Correct the mode specification of the output file. If the first input file had mode number 3 or 4, it has been erased.

E(00012) ATTEMPT TO WRITE OUTPUT FILE ON READ-ONLY DISK.
Files may not be written on READ-ONLY disks. Change the mode of the output file.

E(00013) ATTEMPT TO COMBINE FIXED AND VARIABLE LENGTH FILES.
The input files must all have the same record format.

EDIT

Purpose:

EDIT has three purposes: (1) to create card image and SCRIPT files, (2) to make changes to existing files, and (3) to allow context-directed, online perusal of files.

Format:

EDIT	<filename> filetype

filename specifies the filename of the file to be edited or created

filetype specifies the filetype of the file to be edited or created

Usage:

If a file with the specified filename and filetype does not exist, EDIT assumes that the file is being created, the Input environment is entered, and information typed by the user thereafter becomes input to that file. If such a file does exist, the Edit environment is entered, enabling the user to issue EDIT requests and to modify the specified file.

EDIT searches all disks for the file. If the file is found, its mode is saved and EDIT writes the new file back to that disk. If the file is not found, the newly created file is put on the P disk.

OPERATION OF THE CONTEXT EDITOR

The Editor is a program designed to provide facilities for the creation and modification of card-image and SCRIPT files from an online terminal. Editing is performed upon a main-storage-resident copy of the file. This approach provides for rapid movement both forward and backward through the file. It does, however, limit files which may be edited to those which may be wholly contained within the available main storage. It is possible to perform edit operations upon larger files by using the CEDIT command, (but the movement within the file may be slower, as CEDIT works with disk files and not a core copy of the file) or by issuing SPLIT for the file, to break it up into smaller files that can be handled by EDIT.

LINE POINTER

Associated with each file is a pointer which refers to a line in the file considered to be the current line. The current line is defined as the line that is being created or edited in the file.

Various Edit requests are provided for moving the current line pointer. This pointer may be moved (1) to a specific record (indicated by its record number), (2) to a record specified by its relative displacement (in number of records) forward or backward from the current position of the pointer, or (3) to a record containing a specified string of characters or having a specified label. The ability to search for strings allows the user to concern himself only with the textual context of the desired movement, relieving him of the concern of keeping track of record numbers and counts of inserted and deleted records.

Many of the record modification requests also reposition the current line pointer (for example, LOCATE, FIND, DELETE, or CHANGE). When a FIND, LOCATE, or CHANGE request is issued, if the pointer is positioned at the end of file, the pointer is automatically moved to the TOP of the file before executing the command.

When the environment is changed from Input to Edit, the pointer is positioned to the last line entered from the terminal. When it changes from Edit to Input,the pointer is positioned such that the lines being entered follow the last line edited.

When the EDIT command begins in the Edit environment, the pointer is positioned before the first line in the file. During the Edit a null line is automatically placed in front of the first line of the file to permit the insertion of lines at the beginning of the file. When EDIT begins in the Edit environment or when a TOP request, or possibly the UP request, is issued, the pointer is positioned at this null line. The null line never gets written onto disk, nor is it

ever printed by the terminal. When the pointer is positioned at the null line, a PRINT request types a blank line.

If a null line is entered in the Input environment, the Edit environment is entered. If a null line is entered in the Edit environment, the confirming message "EDIT:" is typed out. To enter the Input environment from the Edit environment, issue the INPUT (I) request.

SAVING INTERMEDIATE RESULTS

If extensive input and/or changes are being made to a file, it is a good time-sharing practice to make a few additions and/or changes at a time, issue the SAVE request, and then continue making additions or changes to the file. The process should be repeated until all additions and/or changes are made. The final copy of a file being edited, or the first copy of a newly-created file, will not be permanently written onto disk until the FILE or SAVE request is issued. This procedure will ensure that a minimum of work would be redone in the case of a system failure or a gross user editing error.

A file must consist of at least one line to be written permanently on disk. A file consisting of only a null line may not be saved.

INPUT ENVIRONMENT

The Input environment is indicated by the message "INPUT:", a carriage return, and the unlocking of the keyboard. The user may then type successive lines of input to the file as fast as he wishes. One card image is created from each input line. To insert a blank line in a file, type at least one space and hit carriage return. A null line (that is, a carriage return with no prior blanks or characters) entered from Input mode does not add a blank line to the file.

Entering the Edit Environment The Edit environment is entered from the Input environment by typing a null line (that is, a carriage return with no prior input on the line).

EDIT ENVIRONMENT

The Edit environment is indicated by the message "EDIT:", a carriage return, and the unlocking of the keyboard. The user may then type requests to the EDIT command. All changes to the file become effective immediately in core, thus allowing recursive modifications to be made to a file. Changes are written out on disk with the FILE and SAVE commands; QUIT keeps the original file as it existed before any changes were made.

Response Modes. There are two response modes in which the Edit environment may operate: "verify" mode and "brief"

mode. Verify is the normal mode and causes an automatic typeout of each line that has been changed or found as the result of a request. The brief mode does not respond by retyping the specified lines, and thus the user must issue a PRINT request to get the typeout if it is wanted.

The messages "EDIT", "INPUT", "EOF:", and "TRUNCATED", are always printed even if the user has selected the brief message mode.

End of File. If the end of file is reached by an Edit request, an "EOF:" message is typed, and the pointer is positioned after the last line of the file.

Entering the Input Environment. The Input environment may be entered by typing the INPUT request, and a carriage return.

EDIT REQUESTS

Requests are issued to the EDIT command only when the user is in the Edit environment. These requests allow the user to manipulate and edit files. If requests are issued during the Input environment, they become lines of input to the file.

Request Formats. Each request is separated from its operand by one or more spaces unless otherwise specified. These spaces can be inserted by using the space bar, the tab key, or the logical tab character, which is discussed in a later section. If the tab key or the logical tab character is used and the request has "line" as the operand, the "line" is placed in the card image as if the tab key or logical tab character were the first character in "line".

The tab settings discussed are internal or logical tab settings, not the external or physical tab settings on the terminal. Detailed information concerning record formats, serialization, and special character conventions follows.

String Arguments. Several of the Edit requests require arguments called string arguments. These arguments either are matched against strings in the text, or replace a string in the text. A string argument begins with a delimiter and continues as a sequence of any legal characters until the initial character (that is, the delimiter) is again encountered. Neither delimiter character is involved in the actual matching or replacement operation. Although, by convention, the slash (/) is used in the following request descriptions to denote the string delimiter, any legal character may be used as the delimiter. The delimiter character is redefined in each new request by its appearance at the head of a string. If two strings exist in one request, the same delimiter character must be used in each string. Only one delimiter may be used to separate two adjacent string arguments in a request (for example, as in the CHANGE request).

FILE (RECORD) FORMATS

In general, the editor is used for the preparation of fixed-length (logical) records of 80-character card images with uppercase characters. The exceptions are defined below.

All input to the file being edited is converted from lowercase to uppercase unless a filetype of MEMO or SCRIPT has been specified. If a filetype of SCRIPT has been specified, the file consists of variable-length (logical) records. If a filetype of MEMO is specified, the file consists of 80-character card images.

MEMO Files

A filetype of MEMO is used to input 80-character card images containing both uppercase and lowercase letters.

SCRIPT Files

The EDIT command allows processing of SCRIPT files in a form compatible with the SCRIPT command. If the filetype is SCRIPT, all input lines introduced in the Input environment and strings introduced in the Edit environment through use of CHANGE, OVERLAY, RETYPE, and INSERT are interpreted without converting lowercase characters to uppercase. The character-delete and line-delete symbols have the usual effect; all other characters are stored without modification, including the tab key.

For SCRIPT files, the file format is set to V (variable-length records) so that the SCRIPT command can be used to edit or print files created by the EDIT command.

Input lines containing a backspace character are converted into canonical form, such that only one backspace follows any character and only one backspace precedes the character that overprints the character preceding the backspace.

Record Lengths

The truncation columns used for EDIT requests are as follows:

INPUT and REPLACE: column 71 for SYSIN, ASP360, UPDATE, COPY
column 72 for FORTRAN, COBOL, and PLI files
column 132 for SCRIPT, LISTING, PRINTER files
column 80 for other files

FIND, OVERLAY, and BLANK: column 80

LOCATE and CHANGE: end ZONE column

PRINT: column 72, if serialization is defaulted on, or if VERIFY column is set after serialization is turned on; otherwise all 80 columns are printed.

VERIFY: column 72, unless set otherwise.

(See the ZONE command.)

Tab Settings

Internal or logical tab settings indicate a column position that defines the beginning of a field within a record. The logical tab settings are automatically assumed according to the filetype specified. These internal tab settings have no relation to the external tab settings on the terminal. The assumed internal tab settings are given below, where the first of these numbers indicates the column of the record in which input is to begin.

Filetype Specified in EDIT Command	Assumed Tab Settings
AED	1,10,15,20,25,30,35,40,45
ASP360	1,10,16,31,36,41,46,51,56,72
AS1130	1,21,27,32,35,45,50,55,60
COBOL	1,7,10,15,20,25,30
COPY	same as ASP360
DATA	default
EXEC	1,5,8,17,27,31
FLOW	same as ASP360
FORTRAN	1,7,10,15,20,25,30
LISTING	default
MEMO	default
PLI	2,10,15,20,25,30,35,40,45,50,55,60
PRINTER	default
REPS	7,17,31,36
SNOBOL	1,10,20,25,30,35,40,45,50,55,60
SPL1	1,7,17,30,40,50,60
SYSIN	same as ASP360
UPDATE	same as ASP360
default	1,5,10,15,20,25,30,35,40,45,50

If the specified filetype is not one of those listed, the default setting of every five spaces is assumed. The assumed tab settings can be redefined by the TABSET request from the Edit environment.

UPDATE files have the same tab settings as SYSIN files but are not sequenced.

If a filetype of REPS is EDITed, tab settings are assumed and a 12-2-9 character is inserted into column 1 if it appears blank.

Note. COBOL as a filetype is included to allow COBOL source decks to be entered under CMS. Note that the COBOL compiler is not included under CMS.

Serialization of Records

If serialization is assumed for the filetype being edited, or if it is specified by the SERIAL request, an identifier is placed in columns 73-80 of each record. The identifier consists of a three-character identification followed by a five-digit sequence number. The identification is taken from the first three characters of the filename, or from the first parameter of the SERIAL request. The sequence number begins at 00010 and is incremented by 10. The line identifier in columns 73-80 can be changed by issuing the SERIAL request when in the Edit environment.

Serialization is suppressed unless requested by the SERIAL request, for all filetypes except FORTRAN, COBOL, PLI, SYSIN, UPDATE, SPL1, COPY, SNOBOL, AED, FLOW, AS1130, and REPS. If serialization is selected for other filetypes, and neither ZONE 1 71 nor SERIAL id is issued during subsequent Edits of the file, serialization characters may be inadvertently shifted into column 72, or columns 73-80 may be overwritten.

The placing of the identifier in columns 73-80 can be eliminated by specifying no serialization in the SERIAL request; this causes the truncation of input lines at column 80 and no identifier to be assigned to each record. A file whose filetype is MEMO, SCRIPT, PRINTER, LISTING, or EXEC should not be serialized.

SPECIAL CHARACTERS

Logical Tab Character

There is a character, called the logical tab character, used in conjunction with the logical tab settings. This character causes blanks to be inserted into the record from the column position at which this character is input, until the first column position of the next field defined by the logical tab settings. The next character from the input line after the logical tab character is inserted as the first character of the next field. This blank insertion is done for all filetypes except SCRIPT; for blank insertions in SCRIPT files, refer to the SCRIPT command.

The standard logical tab character is the pound sign (#). The logical tab character can be redefined by the TABDEF request in the Edit environment, or by the CHARDEF command in the CMS environment to allow the pound sign to be used as a normal input character. The physical tab key on the terminal can also be used for blank insertions, as it is interpreted in the same manner as the logical tab character. The only difference between the tab key and the logical tab

character is that the tab key moves the typing element to the next physical tab stop and does not print, while the logical tab character prints when the character is depressed and the typing element does not move to the next physical tab stop.

Note that the logical line-end character is also defined as the # and it takes precedence over the logical tab character.

Logical Backspace

There is also a character, called the logical backspace which is used to backspace one column position in the record. For n logical backspace characters, n column positions are backspaced in the record, and the n backspaced positions are overlaid with the n characters which follow the n logical backspace characters. If no character is given after a logical backspace character, the previously entered character is not overlaid. The backspacing is performed for the column positions of a record, and not for the characters of an input line.

The standard logical backspace character is the % character. It may be redefined by the BACKSPACE request in the Edit environment or the CHARDEF command in the CMS environment to allow the % sign to be used as a normal input character. The logical backspace character has the same effect as the physical backspace key on the terminal for all filetypes except SCRIPT. With SCRIPT files, the physical backspace key moves the typing element back one position, and generates a valid input character that takes up one column in the record for each time the key is depressed; the backspace key does not print when entered on the terminal, nor does it print on the offline printer, but it backspaces the typing element one position per character when the record is printed out at the terminal. Thus, the backspace key allows underscoring and overprinting at the terminal for SCRIPT files. The logical backspace character prints only when entered and does not take up a column in the record; it logically backspaces one column in the record for all filetypes.

The logical tab character and the logical backspace character can be used in the following requests in the Edit environment to insert blanks, and to backspace in the record respectively: BLANK, FIND, INSERT, OVERLAY, and RETYPE. The logical tab character and the logical backspace character can be used as normal input characters in the string operands of the CHANGE, LOCATE, and DELETE requests.

Responses:

NEW FILE.
The specified file does not exist, thus the Input environment is entered. All subsequent input lines will be accepted as input to the file.

INPUT:
The Input environment is entered. The logical tab settings may be either those defined by the user or those assumed from the filetype. All subsequent input lines will be accepted as input to the file.

INVALID REQUEST: xxx...xxx
The invalid request xxx...xxx was issued to EDIT.

DEFAULT TABS SET
The filetype specified is not recognized by the EDIT command; thus, the default settings are taken for the logical tab settings.

EDIT:
The Edit environment is entered. The logical tab settings may be either those defined by the user or those assumed from the filetype. An edit request may now be issued.

NO PRIMARY NAME SPECIFIED
The EDIT command was issued specifying only the filetype. When the file request is issued, a name must be specified.

TRUNCATED
The input line was too long. If the filetype was FORTRAN, PLI, SPL1, AED, FLOW, EXEC, ASM1130, or REPS, then the input line was truncated after 72 columns. If the filetype was ASP360, SYSIN, COPY, or UPDATE, then it was truncated after column 71. If the filetype was SCRIPT or LISTING, then it was truncated after 132 columns. Otherwise, it was truncated after column 80. Continue typing more input.

EOF:
The end of the file has been reached during an EDIT request. The request has been terminated and the pointer is positioned after the last line of the file. Another EDIT request may be issued.

EDIT WORK FILE "(INPUT1) FILE P1" EXISTS;
IF GOOD, ALTER IT TO APPROPRIATE FILENAME & FILETYPE;
OTHERWISE, ERASE IT.
The EDIT command was issued but the EDIT environment was not entered since the EDIT workfile, (INPUT1) FILE, exists. The workfile contains whatever file the user had been editing during a previous EDIT command that had not terminated (that is, if CP has crashed while the user was editing a file and the READY message had not appeared yet, this file may contain the updated file). If the contents are good, alter its identifiers; otherwise, erase it and then reissue EDIT.

Error Message

E(00002) FILE EMPTY - EXIT TAKEN
The user has attempted to save an empty file. The file is not written on the disk and the EDIT command is terminated.

SUMMARY OF EDIT REQUESTS

The requests are listed below by functional group

Environment Selection

INPUT	(See also SAVE)
(null line)	returns to EDIT mode
QUIT	(See also FILE)

Message Mode Selection

VERIFY
BRIEF

Pointer Movement

BOTTOM
FIND
LOCATE
NEXT
TOP
UP

* Requests marked with * in the rest of this list may move the pointer under certain conditions.

Modification of Records

BLANK
CHANGE (*)
DELETE (*)
INSERT (*)
OVERLAY
REPEAT (*)
RETYPE (*)

File Handling

FILE
SAVE

Information Requests

PRINT (*)

Special Characters and Format Conventions

BACKSPACE
SERIAL
TABDEF
TABSET
ZONE

Miscellaneous

REPEAT (*)
X (*)
Y (*)

BACKSPACE Request

Format:

BACKSPACE BACK	<character>

character is any valid character. The default character is % or the character specified by the CMS CHARDEF command.

Usage:

The BACKSPACE request defines the character to be used as the logical backspace character. If the request is not issued, the default character is assumed. The logical backspace character causes one column to be backspaced in the card image for each logical character in the input line.

The backspace character must be redefined to allow the % character to be used as a normal input character.

If BACKSPACE is issued without specifying a character, the logical backspace character is reset to the character specified by the CMS command CHARDEF, which defaults to the % character.

The backspace character is very useful for defining continuation cards in FORTRAN files. If the first logical tab setting is set to column 7, the tab key or logical tab character followed by a logical backspace character may be used to enter a character in column 6 instead of counting forward the appropriate number of spaces. An example of the use of the logical backspace character follows:

```
                                column
                                1    6
input line:                     i #%5'page')
card image in file:                  5'PAGE')
```

This places "5'PAGE')" beginning in column 6 of the card image, assuming the first logical tab setting is set to column 7.

Note:

If the logical backspace character is redefined in the Edit environment, the BACKSPACE request must be issued each time a file is edited if it is necessary to use the same redefined logical backspace character. If the logical backspace character is redefined by the CMS CHARDEF command, that character is remembered from one Edit command to the next.

Responses:

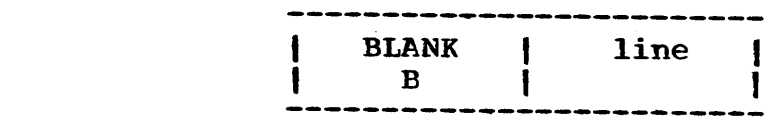
The keyboard is unlocked.

Example:

```
BACK $
```
The character $ is defined as the logical backspace character. The % character can now be used as a normal input character.

BLANK Request

Format:



"line" is any valid input line.

Usage:

The BLANK request places blanks in the current line wherever nonblank characters occur in "line".

Either the tab key or the logical tab character can be used to generate blanks in the card image.

The "line" is separated from the request by only one blank. All other blanks are considered part of "line".

Responses:

Verify Mode: The changed line is printed out.
The keyboard is unlocked.

Any mode: EOF:
The end of file was reached by the request. For BLANK to reach the end of file, the repeat request had to be issued before BLANK.

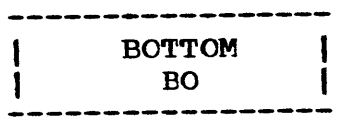
Example:

```
BLANK AAAAAA A
                                  column
                                  1    6
     line before request:         ABCDFJMNOP
     request:                     blank aaaaaa a
     line after request:                M OP
```

Blanks are placed in columns 1-6, and 8, of the current line in the file.

BOTTOM Request

Format:



Usage:

BOTTOM positions the pointer at the last line of the file.

Example:

```
BO
The pointer is positioned at the bottom of the file.
```

BRIEF Request

Format:

Usage:

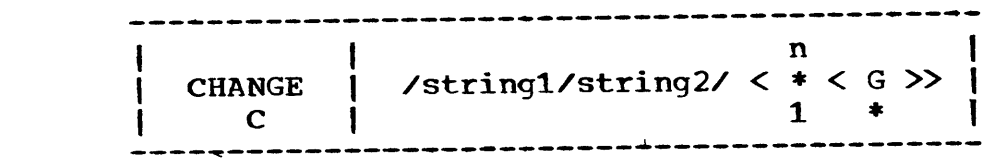
The BRIEF request deselects the verify message mode (see the VERIFY request), and selects the brief message mode in the Edit environment. In the brief mode, lines that are changed in the file are not typed out automatically. The requests that are affected by BRIEF are BLANK, CHANGE, FIND, LOCATE, NEXT, OVERLAY, and UP. If INSERT or REPLACE was issued and the line was truncated, the line is not verified in the brief mode.

Example:

BR
This request selects the brief mode.

CHANGE Request

Format:



/ is any unique delimiting character that does not appear in string1 or string2.

string1 is the group of characters to be replaced

string2 is the group of characters to replace string1

n specifies the number of lines to be searched for string1. The default is one line.

G signifies that the change is to be applied to every occurrence of string1 in the lines.

* is used to mean "to EOF" or "GLOBAL" (G), or both, in a change request as follows:
C /a/b/ * *

Usage:

The CHANGE request replaces the occurrence of string1 in n lines by string2. If G or * is specified, every occurrence of string1 in n or * lines is changed; if G is not specified, only the first occurrence of string1 is changed. If neither n nor * is specified, only the current line is searched for string1.

If the occurrence of string1 is not found, the line(s) is (are) not altered. The pointer remains positioned at the last line searched for the occurrence of string1.

String1 and string2 can be of different lengths. Each of the n or * lines is expanded or compressed accordingly.

If an end-of-file condition immediately preceded the CHANGE, an automatic TOP request is performed before CHANGE begins.

Notes:

a. The n or * is required if G is to be specified.

b. If n is greater than the number of lines to the end of the file, every occurrence of string1 from the current line to the end is changed.

c. That part of each record which is scanned for the occurrence of string1 is the part defined by the ZONE

request, or it defaults to those columns defined earlier under "File Record Formats".

Responses:

Verify Mode: The changed line(s) is (are) printed out and the keyboard is unlocked.

Brief Mode: The keyboard is unlocked.

Any mode: EOF:
The end of file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued. When a CHANGE request is issued after the occurrence of an end-of-file condition, a TOP request is automatically issued before the request begins.

Examples:

a. C /ALPHA/DELTA/

```
                              Column
                             1     7
                             |     |
line before request:         ALPHA=ALPHA - BETA
request:                     c /alpha/delta/
line after request:          DELTA=ALPHA - BETA
```

The first occurrence of ALPHA in the one line is changed to DELTA.

The above, lowercase, example works for files where automatic capitalization is standard. For other files, such as SCRIPT or MEMO, where capitalization is not automatic, characters must be typed as they were input.

b. C *ALPHA*DELTA* 1 G

```
                              Column
                             1     7
                             |     |
line before request:         ALPHA=ALPHA - BETA
request:                     c *alpha*delta* 1 g
line after request:          DELTA=DELTA - BETA
```

Every occurrence of ALPHA in the one line is changed to DELTA.

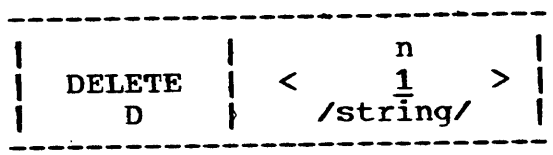
c. C .card-image.card-image. * *

Every occurrence of card-image from the current line to the end of file is changed to itself.

In verify mode, every line containing the string "card-image" is printed, although this example does not effectively change the contents of those lines.

DELETE Request

Format:



n specifies the number of lines to be deleted. The default is 1.

/string/ specifies the string which, when matched, terminates the delete operation.

Usage:

If /string/ is specified, all lines, starting with the current line and up to (but not including) the first line in which /string/ is matched, are deleted.

If n is specified, the DELETE request removes n lines from the file, starting with the line at which the pointer is currently positioned. Upon completion of this request, the pointer is positioned after the last deleted line. If n is 0, the current line is deleted.

If n or /string/ is not specified, only the current line is deleted.

Responses:

EOF:
The end of file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

a. D 5
The current line, plus the next four lines, are deleted. The current line pointer is then positioned at the fourth line from the original current line.

b. D -/*-
All lines, starting with the current line and up to (but not including) the first record containing a /* sequence, are deleted.

FILE Request

Format:

```
-------------------------
|  FILE  |  <filename>  |
-------------------------
```

filename specifies the name to be used as the filename.

Usage:

The FILE request terminates the editing of a file. A permanent copy of the file is written onto the disk as it existed after the last pass through the file. If the file is being permanently stored for the first time, it is written onto the disk. If the file already exists on the disk, it is written on the same disk in the same mode as it previously existed. This latest copy replaces any existing copy of the file on the disk, and the file directory is updated.

If "filename" is specified, it is used as the filename of the file. If "filename" is not specified, the filename used at the time of the invocation of the EDIT command is used.

After the file has been written onto disk, control is returned to CMS. A file must consist of at least one line to be permanently written on disk. A file consisting of a null line only may not be filed.

Note:

If it is desired to move a partially-edited file to disk as a precaution against system or user failure, use the SAVE request. It does not terminate the EDIT session.

Responses:

The EDIT command is terminated and an entry for the file is made in the appropriate file directory.

FILE EMPTY - EXIT TAKEN
The FILE request was issued for an empty file. No file is saved on the disk and its entry is removed from the file directory. The error E(00002) is generated, and the EDIT command terminated.

NO PRIMARY NAME SPECIFIED - RETRY
When EDIT was issued, only the filetype was specified. Therefore, a filename must be given with the file request.

Examples:

a. FILE

```
request:        file
response:       R; xx.xx/xx.xx  hh.mm.ss
```

This request writes the latest copy of the file onto disk.

b. FILE RECALC

```
request:        file recalc
response:       R; xx.xx/xx.xx  hh.mm.ss
```

This request writes the latest copy of the file onto disk, and RECALC is its filename.

FIND Request

Format:

line is any valid input line. It may contain blanks and the logical tab character and/or tab key.

Usage:

The FIND request compares the nonblank characters in "line" with each line in the file. The compare begins on the next line from where the pointer is currently positioned and continues down the file until a match occurs or until the end-of-file is reached. If an end-of-file condition immediately preceded the FIND request, an automatic TOP request is performed before FIND begins. If "line" is found, the pointer is positioned to the record in which "line" is contained. If "line" is not found, the pointer is positioned after the last line of the file. The compare is column-dependent, as the only columns compared in each record are the ones specified by nonblank characters in "line". For example, if "line" contains A C, the search will be for an A in column 1 and a C in column 3.

"Line" is separated from the request by only one blank. All other blanks are considered part of "line".

FIND can be used to search for a specific line identifier in columns 73-80. One technique is to issue FIND, followed by the appropriate number of tabs to position the specified identifier to column 73.

Responses:

Verify Mode: The record is typed and the keyboard is unlocked.
Brief Mode: The keyboard is unlocked.

Any MODE: EOF
The end-of-file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued. When a FIND, LOCATE, or CHANGE request is issued after the occurrence of an end-of-file condition, a TOP request is automatically issued before the request begins.

Examples:

a. FIND 90

```
                          Column
                        1     7
request:                f 90
line found:             90    FORMAT (516)
```

The FIND request searched for 90 in columns 1 and 2. The first line found is typed out in the verify mode.

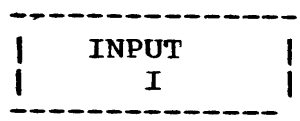
b. FIND $$SUMX
If $ selected as
logical tab char.

```
                          Column
                        1        10    16
request:                f $$sumx
line found:             LOOP     A     SUMX,X
```

Assuming that the logical tab settings are set in 1, 10, 16, the request searches for SUMX in columns 16-19. The first line found is typed out in the verify mode.

INPUT Request

Format:



Usage:

This request causes the Input environment to be entered from the Edit environment. All subsequent input lines--including Edit request--are treated as input to the file, and are placed after the line at which the pointer is currently positioned. If the INPUT request is given at the top of the file, the lines appear before the first line of the file.

If no lines were entered while in the Input environment and return is made to the Edit environment, the pointer is positioned to the line pointed to before the Input environment was entered. The line after the pointer is the same line before and after the Input environment was entered.

To insert a blank line, type at least one space and hit carriage return. If a null line is entered, a return to the Edit environment occurs.

Responses:

INPUT:
The Input environment is entered.

INSERT Request

Format:

INSERT	line
I	

line is the exact input line to be inserted into the file. It can contain blanks and tabs (logical tab character and/or tab key).

Usage:

This request inserts the "line" into the file without entering the Input environment. The line is inserted following the line at which the pointer is currently positioned, and the pointer is advanced to point to this inserted line. The line is separated from the request by only one blank; all other blanks are considered part of "line".

The conventions of the Input environment hold true during the INSERT request.

A blank line can be inserted in the file by using one or more spaces for "line". If "line" is omitted from the INSERT request, it is interpreted as the INPUT request and the Input environment is entered.

Responses:

The keyboard is unlocked. Examples:

a. IbABLEbbbbbSbbbbbSUM,X

```
                        Column
                   1        10  12    16  18
request:           i able       s         sum,x
line after request: ABLE     S         SUM,X
```

The input line ABLEbbbbbSbbbbbSUM,X is inserted in the file. The letter b is used here to indicate a single space.

b. I $DO 10 I=1,25

```
                        Column
                   1     7
request:           i $do 10  i=1,25
    (assuming $ is the logical tab)
line after request:       DO  10 I=1,25
```

Assuming that the logical tab settings are in 1, 7, 10, and 15, this request inserts the FORTRAN statement DO 10 I=1,25 in columns 7-18.

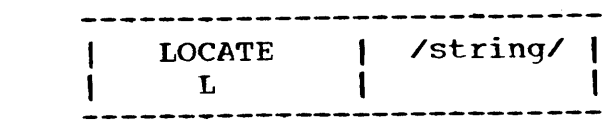
c. INSERT

```
request:        insert
response:       INPUT:
```

The Input environment is entered, as "line" was not specified with the INSERT request.

LOCATE Request

Format:



/ is any unique delimiting character that is not contained in the string

string is any group of characters to be searched for in the file

Usage:

LOCATE scans the characters of each record (as defined by the ZONE request) for the string specified between the two delimiters. The scan begins on the next line from which the pointer is currently positioned and continues until the string is found or until the end-of-file is reached. If an end of file condition immediately preceded the LOCATE, an automatic TOP request is performed before LOCATE begins. If string is located, the pointer is positioned at the line that contains it. If string is not located, the pointer is positioned after the last line of the file.

The request is not column-dependent, because all characters are scanned, as specified by the ZONE request. The logical tab character and the logical backspace character can be used as normal input characters in string.

LOCATE can be used to scan for a line identifier in columns 73-80.

Responses:

Verify Mode: The located line is typed and the keyboard is unlocked.
Brief Mode: The keyboard is unlocked.

Any Mode: EOF
The end of file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued. When a FIND or LOCATE request is issued after the occurrence of an end-of-file condition, a TOP request is automatically issued before the request begins.

Examples:

a. L /FORMAT/

```
                        Column
                       1     7
request:               1 /format/
line located:          55    FORMAT ('DAILY AUDIT')
```

LOCATE searches all characters of each line for FORMAT . In the verify mode the first line found is typed out.

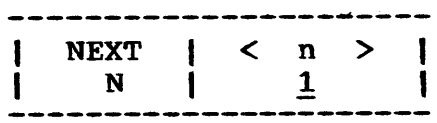
b. L 61

```
                              Column
                             1   5   9
request:                     1    $61$
line located:                12316XXX61987654321
```

LOCATE searches for 61 in all columns of each line. In the verify mode the first line found is typed out.

NEXT Request

Format:



n is an integer indicating the number of lines by which the pointer should be advanced. The default is 1.

Usage:

This request advances the pointer in the file by n lines. If n is 0 or unspecified, a value of 1 is assumed and the pointer is advanced to the next line in the file. If the end of file is reached before the pointer is advanced n lines, the pointer is positioned after the last line. Specifying a value of n that is greater than the number of lines to the end of file is one method of reaching the bottom of the file.

Responses:

Verify Mode: Line typed and keyboard unlocked.

Brief Mode: Keyboard unlocked.

Any Mode: EOF
The end of file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

a. N 5
This request advances the pointer five lines.

b. N
This request advances the pointer one line.

OVERLAY Request

Format:

OVERLAY O	line

line is an input line that replaces parts of the current line.

Usage:

This request takes the nonblank characters from "line" and places them in the corresponding position of the current line. Blank characters in "line" do not replace corresponding positions in the current line. If there is more than one space after the request, these spaces are considered as part of "line". The logical tab character, the tab key, and the logical backspace character can be used in specifying "line".

Note. The line typed as a result of this command does not print directly below the corresponding characters resulting from commands NEXT, PRINT, etc., as the characters O or OVERLAY precede the line entered. Thus, the characters appear in the column which is n+1 characters to the right of the character being overlaid, where n is the number of characters typed in the request's name. (See examples below.)

Responses:

Verify Mode: The changed line is typed out and the keyboard is unlocked.

Brief Mode: The keyboard is unlocked.

Any Mode: EOF
The end of file was reached by the request. For OVERLAY to reach the end of file, the REPEAT request had to be issued before the OVERLAY.

Examples:

a. OBbbbbbbbbING

```
                          Column
                          1       9
line before request:      PROGRAMMER
request:                  O         ing
line after request:       PROGRAMMING
```

Columns 9-11 in the current line are replaced by the nonblank characters in "line". The letter b is used here to indicate a single space.

b. Ob5b33b9

```
                          Column
                          1 3  6
line before request:      ABCDMNOP
request:                  o 5 33 9
line after request:       5B33M9OP
```

Columns 1,3,4, and 6 in the current line are replaced by the nonblank characters in the "line". The letter b is used here to indicate a single space.

c. Ob#%c

```
                          Column
                          1     7     13
line before request:            F10.5,I10)
request:                  o #%C
line after request:             CF10.5,I10)
```

Assuming that a logical tab setting is set to column 7, the logical tab character (#) followed by the logical backspace character (%) places the next character from the input line into column 6. The C overlays the blank in column 6 of the current line. The letter b is used here to indicate a single space.

PRINT Request

Format:

PRINT	< n < LINENO >>
P	1 L

n is an integer specifying the number of lines to be typed out. The default is 1.

L signifies that the line identifiers should be typed out.

Usage:

PRINT types out n lines from the file, starting with the current line. Upon completion of this request, the pointer is positioned at the last line printed unless an end-of-file condition occurred, in which case the pointer is positioned after the last line printed. If n is 0 or unspecified, it is assumed to be 1 and the current line is typed.

If L or LINENO is specified, the line identifier in columns 73-80 is typed out with each line. If L or LINENO is not specified, only the nonblank characters in column 1-72 of each line are typed.

The n is required if L or LINENO is to be specified.

Responses:

The line(s) is (are) printed out and the keyboard is unlocked.

EOF:
The end of file was reached by the request. To position the pointer at the top of the file, a TOP request must be issued.

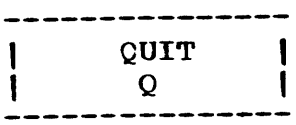
Example:

```
P 5
     request:          p 5
     lines printed:         WRITE (6,30)
                       30   FORMAT (' HERE I AM')
                            CALL SUB1
                            WRITE (6,10)
                       10   FORMAT (' BACK AGAIN')
```

This request types five lines. The line identifier is not included. The pointer is positioned at the last line typed.

QUIT Request

Format:



Usage:

QUIT terminates the EDIT command and returns to the CMS environment without causing a file to be written on the disk, or making permanent updates to an existing file.

Response:

The EDIT command is terminated and the file is not written out or permanently updated.

Example:

```
Q
     request:          q
     response:         R; xx.xx/xx.xx  hh.mm.ss
```

EDIT is terminated and the file is not written on the disk.

REPEAT Request

Format:

REPEAT	< n >
	1

n is an integer specifying the number of times to repeat the following BLANK or OVERLAY request. The default is 1.

Usage:

This request executes the following BLANK or OVERLAY request n times. If n is 0 or unspecified, it is assumed to be 1. If n is greater than the number of lines between the current line and the end of file, REPEAT is in effect until the end of the file. Thus, the REPEAT request can provide global BLANK and OVERLAY requests.

Response:

The keyboard is unlocked.

Example:

REPEAT 25
The following BLANK or OVERLAY request is executed 25 times.

RETYPE Request

Format:

RETYPE	<line>
R	

line is an input line that replaces the current line.

Usage:

This request replaces the current line with "line". The logical tab character, the tab key, and the logical backspace character can be used in "line". "line" is separated from the request by only one blank; any other blanks are considered part of "line". If no line is specified, the current line is deleted and the INPUT environment is entered.

The pointer is not advanced by this request unless the INPUT environment is entered.

Responses:

The keyboard is unlocked.

INPUT:
No line was specified, The current line is deleted, and the INPUT environment is entered with the keyboard unlocked.

Example:

```
Rb#IREG = J + K**2
                                Columns
                              1       7       14
     line before request:             IRE555 = 1 - K
     request:                 r #ireg = j + k**2
     line after request:              IREG = J + K**2
```

The "line" specified with the request replaces the current line. Assuming that the logical tabs are set for a FORTRAN filetype, the statement IREG = J + K**2 begins in column 7.

SAVE Request

Format:

```
-------------------------
|  SAVE  |  <filename>  |
-------------------------
```

filename is the name to be given to the file, as the latest copy is permanently written on disk.

Usage:

The SAVE request writes the latest copy of the file onto the appropriate disk and returns to the Input environment with the pointer positioned at the same current line as before the SAVE was issued. If the file already exists on the disk, it is written onto the same disk in the same mode as it previously existed. This latest copy replaces any existing copy of the file on the disk. The file directory is updated.

If "filename" is specified, it is used as the filename of the file. If "filename" is not specified, the filename used at the time of the invocation of the EDIT command is used. If "filename" is not specified at any time, a message is typed out.

Responses:

INPUT:
The latest copy of the file has been saved on disk and the Input environment has been entered. The pointer is positioned at the same current line as before the SAVE was issued.

FILE EMPTY - EXIT TAKEN
The SAVE request was issued for an empty file. The file is not written on disk and the Input environment is entered.

NO PRIMARY FILE NAME SPECIFIED - RETRY
When EDIT was issued, only a filetype was given. To save the file, a filename must be specified with SAVE.

Example:

```
SAVE MY
     request:     save  my
     response:    INPUT:
```

SAVE MY was issued to write the latest copy of the file on disk and to give it the filename of MY. The Input environment of EDIT is then entered. The current line pointer is repositioned as in the INPUT request. If a null line is entered, EDIT environment is entered and the pointer is positioned as it was when the SAVE request was issued.

SERIAL Request

Format:

```
-----------------------------
| SERIAL  |   id   < n >  |
|   SER   |  (NO)    10   |
-----------------------------
```

id specifies the three-character identification to be used in columns 73-75.

(NO) specifies no serialization or identifier is to be placed in columns 73-80.

n specifies the increment for the line number in columns 76-80. This number also becomes the first line number. The default value is 10.

Usage:

This request allows the user to specify the three identification characters and the increment of line numbers to be used as the identifier in columns 73-80 of each card image. If the SERIAL request is not issued, the standard identifier is used. The standard identifier is formed from the first three characters of the filename; the increment and beginning sequence number is 00010.

If columns 73-80 are to be used for data or if no identifier is desired, the SERIAL (NO) request should be issued. If a file is being created, the SERIAL (NO) request should be issued before any input lines are typed. When the EDIT command for a new file is issued, the Input environment is entered. Before any lines are typed in, the user should immediately enter the Edit environment by typing a null line, issue the SERIAL (NO) request, and then return to the Input environment by issuing the INPUT request to enter lines of input. Eighty character input lines can then be entered.

If a file already exists with no identifiers, the SERIAL (NO) request must be given each time the EDIT command is issued to maintain the data that currently exists in columns 73-80 of the file. If a file already exists with no identifiers and the SERIAL (NO) request is not issued, the standard identifier is placed in columns 73-80.

If a file already exists and the SERIAL request is issued with an id and/or increment, the new identifier replaces the contents of columns 73-80; the replacement does not occur until a FILE or SAVE request is issued. The entire file is then resequenced with the new identifier.

If a file already exists with identifiers, or if input lines have been entered which were serialized, the SERIAL (NO)

request has no effect and the identifiers are not changed. Once serialization has begun, it cannot be nullified.

Note:

For a filetype of MEMO, SCRIPT, LISTING, or EXEC, the default option for serialization is SERIAL (NO). That is, if serialization is wanted, it must be explicitly stated. (See "Serialization of Records" under "File (Record) Formats".)

Response:

The keyboard is unlocked.

Examples:

a. SERIAL REP 20
The request causes REP to be placed in columns 73-75 of each card image, the first input line to be numbered 00020, and the line numbers to be incremented by 20.

b. SERIAL (NO)
If the file is being created, this request allows the user to create 80-character card images from each input line, as no identifier is placed in columns 73-80. If the file already exists without identifiers, the data in columns 73-80 is maintained. If a file already exists with identifiers, or if input lines have been entered which were serialized, the SERIAL (NO) request has no effect until the current pointer is positioned at the top of the file, after which no new serialization takes place.

TABDEF Request

Format:

```
-------------------------------
|  TABDEF   |  < character > |
|   TABD    |                |
-------------------------------
```

character is any valid character to be used as the logical tab character. The default character is the # or the character specified in the CMS CHARDEF command.

Usage:

TABDEF respecifies the character to be recognized as the logical tab character. If TABDEF is not issued, the default character is assumed. Note that the # is also defined as the logical line-end character. To use the # as the tab character, the CMS LINEND command must be issued.

If the # character is to be used as a valid input character, the logical tab character and the line-end character must be redefined.

IF TABDEF is issued without specifying any character, the logical tab character is reset to the # character or the character specified by the CHARDEF command.

If the logical tab character is redefined in EDIT, the TABDEF request must be issued each time the EDIT command is issued, if the user desires to use the same redefined logical tab character. If the logical tab character is redefined by the CMS CHARDEF command, that character is remembered from EDIT to EDIT.

Response:

The keyboard is unlocked.

Examples:

a. TABD $
This frees the # as a valid input character and defines the $ as the logical tab character.

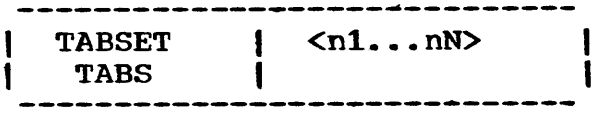
b. An example of the use of the logical tab character in a FORTRAN filetype is shown below

```
                          Column
                         1     7
input line:              #x = a + b
record in file:                X = A + B
```

If the logical tab setting is set in column 7, the expression X=A+B begins in column 7 of the record.

TABSET Request

Format:



n1 is the column in the record at which the line is to begin.

n2...nN are column positions for logical tab settings. If omitted, the default tab settings are used.

Usage:

This request enables the user to establish his own internal or logical tab settings for a record. The tab settings determine the number of spaces to be inserted in the line when either the logical tab character or the tab key is used.

The TABSET request is followed by from one to eleven numbers that do not exceed the value of 80. The first number indicates the column in which the record begins, and the following ten numbers specify the logical tab settings.

If more than eleven numbers are specified, only the first eleven are used. Input lines are truncated to 71 or 72 characters, when tabbing indicates a column position above 71, and when serialization is in effect. Otherwise input lines are truncated to 80 characters, even if a number greater than 80 is specified.

If the first number, specifying the column in which the record begins, is not equal to 1, all input to the file will start at the specified column in the record and all previous columns are set to blanks. The EDIT requests BLANK, FIND, and OVERLAY consider each record to begin in column 1, regardless of the first specified tab value. The EDIT requests INSERT and RETYPE interpret the beginning column position and process the specified line in the same manner as input to the Input environment.

The TABSET request overrides the assumed logical tab settings, such as for FORTRAN, PLI, and SYSIN filetypes.

The user-defined tab settings apply only to the file during the current EDIT command. If EDIT is issued again for the same file, the assumed logical tab settings are in effect until TABSET is reissued.

If TABSET is issued without specifying any values, the logical tab settings are reset to the default settings for the filetype of the file being edited.

Logical tab settings, which are redefined by the user for a file, must be redefined each time the EDIT command is issued, if the same logical tab settings are desired.

Response:

The keyboard is unlocked.

Examples:

a. TABS 1 7 13 19 25 60
This request sets the logical tab settings in columns 7, 13, 19, 25, and 60, and input starts in column 1.

b. TABS 2 5 10
This request would be used to prevent entry of data into column 1, and to set the tabs at columns 5 and 10.

TOP Request

Format:

Usage:

This request repositions the pointer to the top of the file (that is, to the null line in front of the user's first line in the file). An automatic TOP is performed by the FIND, LOCATE, and CHANGE requests if an end-of-file file condition immediately preceded the FIND, LOCATE, or CHANGE request.

Response:

The keyboard is unlocked.

Example:

```
T
The pointer is positioned at the top of the file.
```

UP Request

Format:

UP	< n >
U	1

n is an integer indicating the number of lines by which the pointer should be moved back. The default is 1.

Usage:

The UP request repositions the pointer n lines before the current line. If n is 0 or unspecified, a value of 1 is assumed, and the pointer is moved up to the previous line in the file. If n is greater than the number of lines between the top of the file and the current line, the request functions as a TOP request.

Responses:

Verify mode: The line at which the pointer is repositioned is printed, and the keyboard is unlocked.

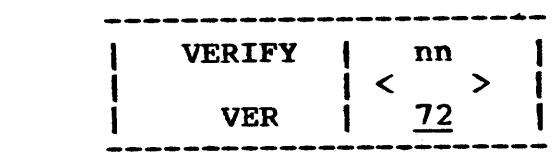
Brief mode: The keyboard is unlocked.

Example:

```
U 9
```
This request repositions the pointer nine lines before the current line.

VERIFY Request

Format:



nn specifies the number of columns to verify in each card image. The default is all columns for MEMO and SCRIPT, 72 columns for all others.

Usage:

This request terminates the brief mode (see the BRIEF request), and selects the verify mode, causing the automatic typeout of lines changed or searched for by other EDIT requests. If nn is specified, nn columns are verified. If nn is not specified, 72 columns are verified; this is the normal mode of operation in the Edit environment. If a MEMO or SCRIPT file is being edited, all columns are automatically verified.

The requests which are affected by VERIFY are BLANK, CHANGE, FIND, LOCATE, NEXT, OVERLAY, and UP.

Response:

The keyboard is unlocked.

Example:

VERIFY 50
The first 50 columns are verified in each card image.

X and Y Request

Format:

X	<request>
Y	n
	1

request is any edit request

n is the number of times the saved request is to be executed. Default is 1.

Usage:

X and Y allow the user to save a request for later execution and temporarily name it X or Y. (Thus two requests may be in a saved status at one time.) This is done with the X request form.

The X n form allows the user to direct that the request currently named X be executed n times, or until end of file. If the form X with no parameter is used, the command currently named X is executed once.

Examples:

a. Y CHANGE /ABCDE FG//
The change request is saved as Y, allowing the user to delete the string ABCDE FG each time he types the request, Y.

b. Y 3
The saved request (as in the above, saved, CHANGE request) executes 3 times.

ZONE Request

Format:

ZONE	n1	n2
Z	1	truncol

n1 specifies the initial column of the zone of each record which is to be scanned. The default value is column 1.

n2 specifies the terminal column of the zone of each record which is to be scanned. If serialization is in effect, the default value is 71 for SYSIN, ASP360, COPY, and UPDATE files and 72 for FORTRAN, PLI, SNOBOL, AED, FLOW, MAD, AS1130, and REPS files. If serialization is suppressed, the default value is 132 for all SCRIPT and LISTING files, and 80 for all others, including MEMO, DATA, and EXEC.

Usage:

This request causes subsequent LOCATE and CHANGE requests to apply only to the zone of the records specified by the integer values for starting and ending columns.

The initial zone column, n1, must be strictly less than the final zone column, n2, and n2 must be less than or equal to the truncation column.

If serialization is requested by use of SERIAL and the current zone overlaps columns 73-80, the ending zone column is reset to 72.

Notes.

a. If serialization is in effect for SYSIN, ASP360, COPY, or UPDATE files, the default value of truncol (n2) is 71, but the ZONE ending column can be set to column 72 to insert a continuation character in a record.

b. In addition to its obvious uses in searching and modifying fixed-format card files, the ZONE request has utility in source program editing for adding comment fields, adding continuation characters, and searching on the serialization field, columns 73-80.

c. To change the limits for fixed-column requests, such as FIND, BLANK, and OVERLAY, the TABSET request should be used. For example, to overlay starting in column 52 of each card, the request TABSET 1 52 should be used.

Response:

END ZONE RESET TO 72 FOR SERIALIZATION
If serialization is in effect, and the ZONE request specifies a terminal column (n2) of 73 or greater, the terminal column is set to 72.

Example:

ZONE 1 72
The initial ZONE column is set to 1 and the terminal ZONE column is set to 72, independent of whether serialization is in effect.

ERASE

Purpose:

The ERASE command deletes a file or a related group of files from a user's read-write disks.

Format:

ERASE	filename filetype < filemode >
	* * *
	P

filename filetype filemode specify the file that is to be erased.

* specifies all filenames, all filetypes and/or all filemodes.

Usage:

Filename and filetype must be specified in the ERASE command, either by name or with an asterisk. If the filemode is omitted, the P-disk is assumed. If the filemode is specified with an asterisk, all read-write disks are searched.

Those parts of the file identifier not specified by asterisks are used to search the file directories. Entries for all files matching the specified identifiers are deleted from the appropriate directory(s), and disk space occupied by these files is made available for new files.

Note.

ERASE deletes read-only files.

Response:

ERASE gives no response except the Ready message or an error code.

Examples:

a. ERASE DLFAC MODULE P5
The file specified is deleted from the file directory, and its space on the P-disk is freed.

b. ERASE * LISTING
All files with the type LISTING are deleted from the P-disk.

c. ERASE * * *
All user files on read-write disks are deleted, and the directory is cleared.

Error Messages:

E(00001) INVALID PARAMETER-LIST
The filename or type was omitted, or the filemode was incorrect. Correct the command.

E(00002) FILE SPECIFIED DOES NOT EXIST
The file specified was not found in the user's file directory.

E(00003)
An I/O error occurred. Processing may not have been completed. It may be necessary to initialize the disk again. See FORMAT.

FILEDEF

Purpose:

The FILEDEF command allows the user to specify the Input/Output devices as well as certain file characteristics which will be used by a program at execution time; FILEDEF is also used to modify, delete, and list current file definitions. FILEDEF is not currently used by any of the languages in CMS.

Format:

```
-------------------------------------------------
|         | DDNAME   device                       |
|FILEDEF  |< DSRN    CLEAR   <def1...defN>>       |
|         |   *      DUMMY                        |
-------------------------------------------------
```

DDNAME is the DDNAME of a file in the user's program. The first letter cannot be numeric or an *.

DSRN is the data set reference number as referred to in the user's program. The DSRN can be a one or two digit number. A DDNAME is created of the form FTxxF001, where xx is the DSRN (right justified).

* this parameter along with CLEAR deletes all assignments previously made by the FILEDEF command.

DEVICE is one of the following which specifies the device for input and/or output of the file being defined.

RDR	Card reader
PCH	Card punch
PTR	Printer
CON	User's terminal (see note 3 below)
DSK	Disk
CRT	Cathode ray tube (not yet implemented)
TAPx	Magnetic tape x is blank or 1-5 (see note on TAP below).

CLEAR deletes the assignments of I/O devices, etc. associated with the first parameter as previously defined by the FILEDEF command. An * in the first parameter erases all assignments made by previous FILEDEF commands.

DUMMY sets up a dummy I/O device. No actual I/O is accomplished during execution. This can be used for program testing or at other times when no I/O is wanted.

def1...defN are file characteristics for disk or tape files.

Usage:

Note that FILEDEF is not currently used by any of the CMS languages.

The FILEDEF command allows the user to specify the I/O devices to be used at program execution time. It is similar to the DD (Data Definition) card in OS/360. Certain file characteristics, such as logical record size, block size, and record format can be specified for disk files, as well as seven-track tape characteristics for tape files.

If no parameters are specified with the FILEDEF command, a list of all DDNAMES and the devices assigned to them by previous FILEDEF commands are typed at the console (See note 1 below). File definitions previously created by FILEDEF can be deleted, either individually, or for all file definitions. A file previously defined by FILEDEF can have its definition modified by specifying the same DDNAME or DSRN with new options.

If RDR, PCH, PTR, CON, DUMMY, or CLEAR has been specified, then def1...defN cannot be specified. RDR, PCH, and CON input default to fixed unblocked, logical record length 80, and blocksize 80. CON output defaults to fixed unblocked, logical record length 130, and blocksize 130. PTR defaults to fixed unblocked logical record length 133 and blocksize 133.

If DSK has been specified, then def1 must specify DSNAME, and def2 must specify DSTYPE. These become the filename and filetype of the file on disk. Filemode defaults to P1 at this time.

The rest of the parameters (definitions) after DSNAME and DSTYPE are in pairs. These pairs may appear in any order or may be omitted, in which case the default value is assigned for the parameters which are not included. These values are used only when they have not been defined during compilation or assembly.

The pairs of parameters are the following:

RECFM F, FB, V, VB, or U specifies the record format for the file

The formats are the following:

Fixed unblocked	F	the default value
Fixed blocked	FB	
Variable unblocked	V	
Variable blocked	VB	
Undefined	U	

LRECL number specifies the logical record length in bytes.

The default is 80 bytes.

BLKSIZ number specifies the block size in bytes. The default is 80 bytes.

If a tape device is being specified, then the above three pairs of parameters (RECFM, LRECL and BLKSIZ) may be specified. Note that DSNAME, DSTYPE, and DSMODE may not be specified. The following pair of parameters may also be specified when seven-track tapes are being used:

MODE number where number is a number from 0 to 15.

The following table shows the number to use for desired tape characteristics. Note that 1-5 are for 800 BPI, 6-10 are for 556 BPI,and 11-15 are for 200 BPI.

MODE NUMBER	800 BPI	556 BPI	200 BPI	PARITY	CONVERTER	TRANSLATOR
1	1	6	11	ODD	ON	OFF
2	2	7	12	ODD	OFF	ON
3	3	8	13	ODD	OFF	OFF
4	4	9	14	EVEN	OFF	ON
5	5	10	15	EVEN	OFF	OFF

Notes.

a. File definitions are already set up for the console. The DDNAMES of these are FT05F001 and FT06F001. These names appear in the list of files whenever FILEDEF is issued without parameters.

b. TAP, with no number following, defaults to TAP1. TAP1 and TAP2 are the only legitimate devices at this time. TAP3 through TAP5 are provided for future expansion.

c. Console input is always uppercase translation with editing and blank filling. Console output is always uppercase.

Examples:

a. FILEDEF
A list of all DDNAMEs and devices previously defined by FILEDEF is printed at the user's console.

b. FILEDEF INFILE RDR
A file definition is created with a DDNAME of INFILE and a Device Type of Card Reader.

c. FILEDEF OUTFILE PTR
A file definition is created with a DDNAME of OUTFILE and a device type of PRINTER.

d. FILEDEF 12 PCH
A file definition is created with a DDNAME of FT12F001 and a device type of PUNCH.

e. FILEDEF TOM CON
A file definition is created with DDNAME of TOM and a device type of console.

f. FILEDEF SYS004 DSK GEORGE DATA RECFM FB BLKSIZ 800
A file description is created with DDNAME of SYS004 and a device type of disk. The filename of the disk file is GEORGE and the filetype is DATA. Record format is fixed blocked, and block size is 800. Logical record length is defaulted to 80.

g. FILEDEF JOANNE TAP MODE 1
A file description is created with a DDNAME of JOANNE and device type TAP1 (defaulted). Mode is 800 BPI, ODD PARITY, CONVERTER-ON, TRANSLATOR-OFF.

h. FILEDEF INFILE CLEAR
This deletes the file definition which has a DDNAME of INFILE. This file definition must already have been defined.

i. FILEDEF * CLEAR
This deletes all file definitions which have been created by the user. FT05F001 and FT06F001 are not deleted.

Error Messages:

E(00001) 1st PARAMETER INVALID
The first parameter is something other than an *, a one or two-digit number, or a DDNAME.

E(00002) BAD RTN CODE FROM INVOKED PROGRAM
The FILEDEF command got a bad return code from its call to SVCFREE or SVCFRET. This is a system problem not correctable by the user. Try the command again.

E(00003) PARAMETERS MISSING AFTER OPERAND xxxxxxxx
A parameter required by the FILEDEF command has been left out of the option list by the user. Retype the command with the proper options.

E(0004) SUPERFLUOUS OR INVALID PARAMETERS AFTER xxxxxxxx
One or more parameters after xxxxxxxx is superfluous or invalid. Retype the command with the proper option list.

E(00005) LRECL OR BLKSIZ VALUES INVALID
One of the following conditions exists:

...ue exceeded the maximum allowable size (61,439 ...imal). This is the maximum positive half-word ...ze.

...CL or BLKSIZ values contained nonnumerics.

...LEGAL xxxxx REQUEST
...ber DUMMY or CLEAR.
...N REQUEST is generated when a CLEAR request is ...le definition not created by the user. ILLEGAL ...T is generated when a DUMMY request is made for ...NN which has already been defined by FILEDEF.

...DNAME AND DSTYPE MUST BE SPECIFIED
...are required when DSK is specified. Retype the ...and with the proper parameters.

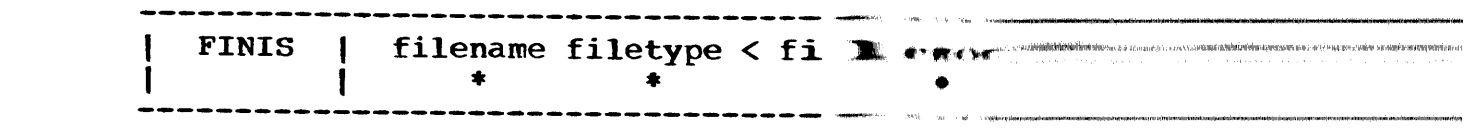
...DNAME NOT SPECIFIED AFTER FILEDEF
...u left out an *, DDNAME or DSRN as his first ...Retype the FILEDEF command with the proper

FINIS

Purpose:

FINIS closes one or more files that ar[illegible]

Format:



filename is the name of the file to be [illegible]
* means all filenames.

filetype is the type of file to be clo[illegible].
* means all filetypes.

filemode is the mode of the file to be [illegible]
* means all filemodes P,T, and [illegible].

Usage:

FINIS closes one or more specified f[illegible]
Closing a file consists of writing o[illegible]
that file on the disk, updating t[illegible]
directory, and removing the entry fo[illegible]
user's active file table. If the fil[illegible]
(delete upon reading), the file is er[illegible].
be used for the filename, filetype, [illegible]
denote the closing of all opened file[illegible]
filenames, filetypes, and/or filemodes. [illegible]
not given, the first file found with t[illegible]
type is closed. The order of sear[illegible]
file(s) is (are) the standard order of [illegible]

The specified file must already be [illegible]
closed. If it is not, an error cod[illegible]
FINIS command.

Note.

FINIS should be issued by the user whe[illegible]
close the files used during the execut[illegible]
Files accessed by CMS commands are clos[illegible]

Responses:

None.

Examples:

a. FINIS DATAOUT CARDS P5
The file whose identifier is DATAOUT CA[illegible]

Value exceeded the maximum allowable size (61,439 decimal). This is the maximum positive half-word size.

LRECL or BLKSIZ values contained nonnumerics.

E(00006) ILLEGAL xxxxx REQUEST
xxxxx is either DUMMY or CLEAR.
ILLEGAL CLEAR REQUEST is generated when a CLEAR request is made for a file definition not created by the user. ILLEGAL DUMMY REQUEST is generated when a DUMMY request is made for DDNAME or DSRN which has already been defined by FILEDEF.

E(00007) DSNAME AND DSTYPE MUST BE SPECIFIED
These items are required when DSK is specified. Retype the FILEDEF command with the proper parameters.

E(00008) DDNAME NOT SPECIFIED AFTER FILEDEF
The user has left out an *, DDNAME or DSRN as his first parameter. Retype the FILEDEF command with the proper parameters.

FINIS

Purpose:

FINIS closes one or more files that are currently open.

Format:

```
-------------------------------------------
| FINIS  | filename filetype < filemode > |
|        |     *        *          *      |
-------------------------------------------
```

filename is the name of the file to be closed.
* means all filenames.

filetype is the type of file to be closed.
* means all filetypes.

filemode is the mode of the file to be closed.
* means all filemodes P,T, and S.

Usage:

FINIS closes one or more specified files that are open. Closing a file consists of writing out the last record of that file on the disk, updating the appropriate file directory, and removing the entry for that file from the user's active file table. If the filemode number is 3 or 4 (delete upon reading), the file is erased. An asterisk may be used for the filename, filetype, and/or filemode to denote the closing of all opened files with the appropriate filenames, filetypes, and/or filemodes. If the filemode is not given, the first file found with the specified name and type is closed. The order of search for the specified file(s) is (are) the standard order of search.

The specified file must already be open in order to be closed. If it is not, an error code is returned by the FINIS command.

Note.

FINIS should be issued by the user when his program does not close the files used during the execution of that program. Files accessed by CMS commands are closed automatically.

Responses:

None.

Examples:

a. FINIS DATAOUT CARDS P5
The file whose identifier is DATAOUT CARDS P5 is closed.

b. FINIS DATAOUT CARDS
The first file found with a filename filetype of DATAOUT CARDS is closed. The permanent disk, the temporary disk, and the system disk are searched in that order for the file.

c. FINIS * FILE1 *
All files that are open and have a filetype of FILE1 are closed.

d. FINIS * * *
All files that are open are closed.

Error Messages:

E(00001)
The specified filename is invalid. It contains leading zeros. The file is not closed.

E(00003)
An error occurred while reading or writing the disk. The command has terminated.

E(00004)
The first character of the mode is illegal.

E(00006)
The specified file is not open and therefore cannot be closed. The command has terminated.

LISTF

Purpose:

LISTF has two purposes: (1) to type at the terminal the name, type, mode, size, date-last-updated, and time-last-updated of specified files or (2) to create a file on the permanent disk containing information similar to that typed at the terminal.

Format:

```
| LISTF | <name   <type  <mode  <(option1...optionN)>>>>|
|       |   *      *     *                              |
```

name is the name of the files to be listed.
* denotes all filenames and is the default value.

type is the type of file to be listed.
* denotes all filetypes and is the default value.

mode is the mode of the files to be listed. If omitted, all read-write disks are searched.
* means all disks.

Note. An asterisk, (*), preceded by any number of characters for name or type searches for the specified characters as the leading characters for that identifier. For example, LISTF ABC* FORTRAN prints the identifiers for all FORTRAN files with filenames beginning ABC.

Options:

EXEC (E) creates a file on the permanent disk containing a list of the specified files.

SORT (S) sorts, or groups together, all similar filetypes.

ITEM (I) types the number of logical items instead of the number of 800-byte physical records.

NAME (N) produces a list of filenames only.

TYPE (TY) causes the list to contain only filename and filetype.

MODE (M) truncates the typed line after filemode.

REC (R) prints filename, filetype, filemode, and number of records.

DATE (D) causes the list to contain name, type, mode, size, and date the file was last written (mm/dd). This is the default line.

YEAR (Y) causes the date to include the year (mm/dd/yy).

TIME (T) causes the time that the file was last updated (hh/mm) to be added to the defaulted line.

Usage:

LISTF either types out the specified files or creates a file containing the information. All operands are optional. If no operand is specified, a complete list of all the files that exist on the user's read-write disks is typed out. The list consists of the name, type, mode, number of records, and date each specified file was last written. The number of records is the number of 800-byte records occupied by the file. The date is typed as month--date (mm/dd). See Figure 6.

```
listf
FILENAME          FILETYPE  MODE  NO.REC     DATE
FORTCLG           EXEC      P1         1     8/29
W                 LISTING   P1         2     7/16
UCC1              MODULE    P1        17     9/08
TRY               TXTLIB    P1         3     8/30
LOAD              MAP       P1        11     8/13
W                 TEXT      P1         5     7/17
UPDATE            MODULE    P1         3     8/30
TEST1             FORTRAN   P1        10     9/03
SUBS              FORTRAN   P1         5     9/02
SUBS              TEXT      P1         6     9/02
R; T=0.05/0.14 15.15.04
```

Figure 6. Output from the LISTF command

If a filename, filetype, and/or filemode other than * is specified, only the file with that identifier is typed out along with its size.

If the filemode is not specified, only the read-write disk directories are examined by LISTF. If a filemode of * is specified, all disks are used. Therefore, in order to have LISTF search read-only disks, a filemode must be specified.

If the (EXEC) parameter is specified, a card-image file is created on the user's permanent disk and assigned the identifier CMS EXEC P1. If a file with the identifier CMS EXEC P1 already exists, it is erased, and a new file is created. This file contains a card image for each of the specified files and the format of each card image is as follows:

Columns	Contents
2-3	&1
5-6	&2
8-15	filename
17-24	filetype
27-28	filemode
31-34	number of records
36-40	date
42-48	time

All other columns are blank. For an example of a CMS EXEC file, see Figure 7, in which the PRINTF command has been used to typeout the contents of the EXEC file.

```
listf * fortran (exec)
R; T=0.20/0.31 15.15.15

printf cms exec

 &1 &2 W          FORTRAN   P1       3       8/12
 &1 &2 SUB2       FORTRAN   P1      12       8/14
 &1 &2 EXAMPLE    FORTRAN   P1       5       7/29
 &1 &2 DRGB       FORTRAN   P1      11       8/30
 &1 &2 GBB        FORTRAN   P1      10       8/30
 &1 &2 SSNSS      FORTRAN   P1       7       8/31
R; T=0.23/0.34 15.18.42
```

Figure 7. Creation and printing of a CMS EXEC file

The CMS EXEC file is like any other user file. It can be printed offline, edited, added to, changed, and so on, but its main purpose is to be used with the EXEC or $ commands. Refer to the EXEC writeup for a description of the usage of a CMS EXEC file.

Responses:

If the (EXEC) option is specified, the file CMS EXEC is generated on the user's permanent disk, and no response is typed at the terminal.

If the (EXEC) option is not given, the list of specified files is typed at the terminal.

As can be seen in the description of the options, only certain quantities of each specified file can be typed by LISTF. Each option has a truncating effect on all options of lower priority.

Examples:

a. LISTF
The name, type, mode, size and date of each file on the read-write disks are typed. See Figure 6.

b. LISTF ABC* FORTRAN (N)
The name of each file that has a filename beginning with the characters ABC, has a filetype of FORTRAN, and exists on the read-write disks is typed.

c. LISTF FILE * * (T)
The name, type, mode, size, date, and time of each file that has a filename of FILE and exists on any disk is typed.

d. LISTF (EXEC)
The file with the identifier CMS EXEC P1 is created on the permanent disk. This file contains the same list of files that was typed in the first example above, but each entry in the list has &1 and &2 placed in front of it.

e. LISTF * FORTRAN (EXEC)
The file with the identifier CMS EXEC P1 is created. This file contains the same list of files that was typed on the terminal in the second example above, but each entry in the list has &1 &2 placed in front of it. See Figure 7.

Error Messages:

E(00001) ERROR IN PARAMETER LIST.
An incorrect form of the command was issued. Check to see if all parameters are valid.

E(00002) NO FILE FOUND.
The specified file does not exist on the disk.

E(00003) * Z (CCU) NOT LOGGED IN **
LISTF was issued with Z as the specified mode, but there is no Z-disk logged in. Either the wrong mode was specified, or LOGIN Z should be issued.

E(00003) NO R/W DISK LOGGED IN
LISTF was issued with no mode specified. The default is all read-write disks, and none are logged in.

OFFLINE

Purpose:

The OFFLINE command controls the unit record input/output devices. Input files may be entered through the card reader. Output files may be printed, with or without automatic carriage control, or punched.

Format:

```
-------------------------------------------------------
| OFFLINE | command   filename  filetype  <filemode>  |
|         |                (*)                        |
-------------------------------------------------------
```

where command is as follows:

READ specifies a deck is to be read from the card reader.

PRINT causes the specified file to be printed on the system printer with automatic single spacing.

PRINTCC causes the specified file to be printed with the first character of each record interpreted as a carriage control character.

PRINTUPC translates to uppercase, and then prints the records of the named file.

PRINTVLR causes the specified, variable-length file produced by an OS/360 language processor to be printed.

PUNCH causes the specified file to be punched onto the system card punch.

PUNCHCC causes an OFFLINE READ filename filetype control card to be inserted as the first card before punching the specified file.

PUNCHDT causes an OFFLINE READ filename filetyped filemode data-last-written time--last-written control card to be inserted as the first card before punching the specified file.

filename filetype <filemode> identify the file to be transferred. If filemode is omitted for a READ, P1 is assumed.

* specifies that filename, filetype, and filemode are found in OFFLINE READ control cards in the input card stream. (Valid only when command = READ)

Usage:

Input

If filename and filetype are specified with the OFFLINE READ command, only one file is read in. Input records of up to 132 characters are accepted. A file that was transferred with the XFER E TO userid command should be read in with the filetype PRINTER. A previously existing file with the same identifiers is erased. If filemode is omitted, P1 is assumed.

If the file designations are to be entered in the card stream, a single asterisk must be specified with the OFFLINE READ command instead of filename and filetype. The deck entered through the card reader may contain any number of files, each immediately preceded by a card containing an OFFLINE READ control card specifying the filename, filetype, and optionally, filemode. The command must start in the first card column. These control cards are typed out at the terminal as they are encountered, and are interpreted by the system just as if they had been entered from the terminal. Any existing file with the same identifiers as those specified on one of the OFFLINE READ cards is erased. Each command card ends the file preceding it, and the last file is ended by the end of the card deck.

If an OFFLINE READ * command is issued, and the first card of the input stream is not of the form OFFLINE READ filename filetype, a file identified as ..NAME.. ..TYPE.. P1 is created containing all cards read in until another OFFLINE READ control card or an end of file is encountered. This temporary file may now be altered to the desired filename and filetype.

When operation is on a virtual machine, user card decks must be read in by CP before an OFFLINE READ command can be issued. The user need not be logged in at the time the decks are read in. Each deck must be entered separately, and each must be preceded by an identification card with CP67USERID punched in the first ten columns, and the user's identification starting in the 13th column, or the characters ID punched in columns 1 and 2 and the userid starting in column 10. CP saves the deck until the user logs in and requests it with an OFFLINE READ command. If more than one deck has been read by CP, they are processed by successive OFFLINE READ commands in the order in which they were entered.

Output

For the OFFLINE PRINT, PRINTCC, PRINTUPC, PRINTVLR, PUNCH, PUNCHDT, and PUNCHCC commands, filename and filetype must be specified. If the filemode field is blank, the P disk is assumed. Asterisks are not permitted in the filename or filetype fields.

The OFFLINE PRINT command prints the specified file with single spacing and CMS page headings containing the file identifiers and a page number. Up to 55 lines are printed on a page. If the file being printed has a filetype of LISTING, it is printed as if PRINTCC were issued.

The OFFLINE PRINTCC command uses the first character of each line in the file as a carriage control code. The maximum line size, including the control character, is 133 characters. A blank (hex'40') in the control position causes the line to be followed by a single space. A zero (hex 'F0') causes a single space before, and after, the printed line. A one (hex 'F1') causes a skip to the top of the next page before the line is printed, and a single space after the line. Another value in the control byte is assumed to be a valid channel command code, and is filled into a CCW. No headings or page numbers are supplied and no automatic skip is performed at the end of the page.

OFFLINE PRINTUPC performs uppercase translation on all records of the specified file. For example, if a file of type SCRIPT or MEMO is to be printed, and the correct printer character chain is not available, PRINTUPC prints the file in uppercase, eliminating all print checks and garbled characters.

OFFLINE PRINTVLR prints variable-length records produced by an OS access method. The printable data of the record is preceded by a four-byte control field which contains the length of the record. This field is discarded, and the record printed under the PRINTCC format.

The OFFLINE PUNCH command accepts records up to 80 characters in length. Shorter records are padded to 80 characters with blanks at the right.

OFFLINE PUNCHCC inserts as the first card of the specified file to be punched, an OFFLINE READ... control card. The punched deck can now be read by means of an OFFLINE READ * command.

OFFLINE PUNCHDT inserts a card with the same information as OFFLINE PUNCHC, but also includes the filemode and date and time last written.

Notes:

a. Files handled by the OFFLINE command must have fixed-length records, except for OFFLINE PRINTVLR, which handles variable-length records.

b. Only the first card of any input deck is checked for CP67USERID or ID. CP processes as a single file all cards following it until a physical end of file is reached.

c. Under CP, printer output is preceded by a single line containing a USERID. PUNCH output is preceded by a card containing a USERID.

d. OFFLINE READ accepts input records up to 132 characters in length, such that LISTING files may be XFER'ed from OS to CMS. The file should be read in with a filetype of PRINTER.

e. If OFFLINE READ was issued, and the file being read in is 132-byte records, CMS types at the finish of the read, RECORD LENGTH = 132 BYTES.

Responses:

a. OFFLINE READ filename filetype filemode
After the command OFFLINE READ * control cards encountered in the input card stream are typed at the terminal.

b. READER EMPTY OR NOT READY.
This response and the Ready message follow an OFFLINE READ command if no card deck has been entered for the user's USERID.

c. R;T=xx.xx/xx.xx xx.xx.xx
The Ready message indicates the command has completed without error. It does not mean that physical output has completed. The output file is held by CP until other users free the output device.

d. "OFFLINE READ..." CONTROL CARD IS MISSING.
THE FOLLOWING ASSUMED:
This response and the assumed control card are typed whenever an OFFLINE READ * command is issued, and whenever the first card of the input stream is not an OFFLINE READ filename filetype <filemode> control card.

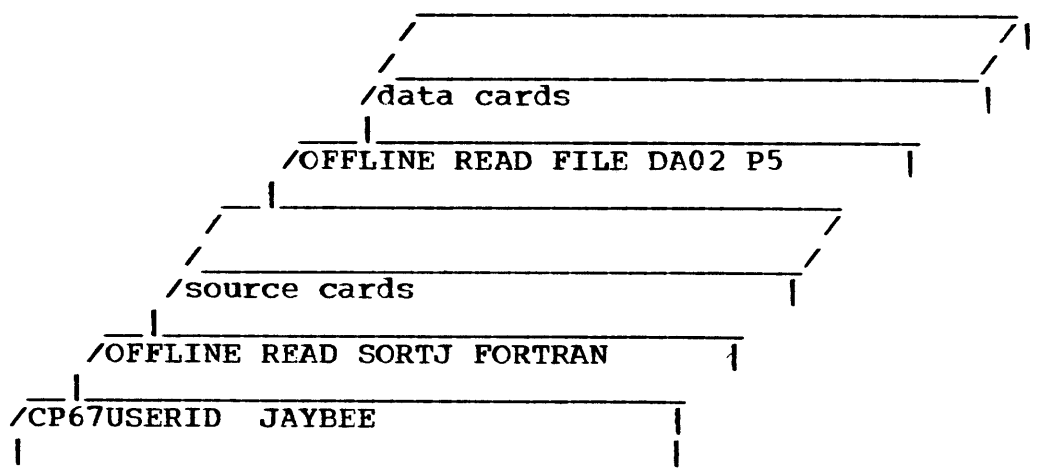
e. (NULL FILE)
Attempt to read a file containing no records was made.

f. SYSTEM I/O ERROR
CP ENTERED, REQUEST PLEASE
This message indicates that an unrecoverable I/O error has occurred on a spooled direct-access device. ReIPL CMS, and issue the OFFLINE command again.

Examples:

a. OFFLINE READ SEC23 SYSIN
Any previous file with the filename and filetype SEC23 SYSIN is erased. All cards following the CP67 identification card are placed in a file on the permanent disk identified as SEC23 SYSIN P1.

b. OFFLINE READ *
Assume the following deck has been entered by the operator:



Any previous files with the identifiers SORTJ FORTRAN or FILE DA02 are erased. The card records are placed in files on the permanent disk under the identifications SORTJ FORTRAN P1 and FILE DA02 P5. The following response is typed:

```
OFFLINE READ SORTJ FORTRAN
OFFLINE READ FILE  DA02 P5
```

c. OFFLINE PRINT FILE DA02
A search is made for FILE DA02, on all three disks, if necessary. When it is located, it is printed out with single spacing and CMS-supplied page headings. The first page of the printout contains only the USERID. The second page starts with the following heading:

```
FILE:FILE DA02 P5  CAMBRIDGE MONITOR SYSTEM PAGE001
```

The heading is followed by a blank line and 55 lines of the file. The heading of the second and subsequent pages is the same, except for the page number.

Error Messages:

E(00001) INVALID OFFLINE FUNCTION OR PARAMETER LIST
There are three possible causes of this message:

a. The operation specified with the command was invalid; it must be one of these: READ, PRINT, PRINTCC, PRINTUPC, PRINTVLR, PUNCH, or PUNCHCC.
b. A filename (or * under the special READ mode) was not specified.
c. Filetype was specified as an *, which is not permitted.

E(00002) FILE NOT FOUND.
The file specified for output does not exist. Check spelling of the filename and filetype.

E(00005)
An attempt has been made to read a variable length file. See

the DISK command to be able to read and punch variable length files.

E(00006) PRINT (max=133) or PUNCH (max=80) RECORD EXCEEDS MAXIMUM LENGTH

The record length of an output file is greater than 132 characters for PRINT, or 133 characters for PRINTCC. This is longer than a printer line. The OFFLINE PUNCH command accepts records of 80 characters or less.

E(00007) PRINT or PUNCH ERROR

System hardware failure has occurred. Retry the operation on the specific unit record device.

E(00008) READ or WRITE DISK ERROR

A disk I/O error has occurred. If reading disk, an illegal mode may have been specified, an attempt may have been made to output an empty file, or there may not be enough core space for the output buffers. If writing disk, the specified mode on the READ command might be illegal, or no more disk space is available.

Purpose:

The PRINTF command types all, or part, of a specified file at the terminal.

Format:

```
 ---------------------------------------------
|PRINTF| filename filetype < n1   n2 < n3 >>  |
|      |                      *    *         |
 ---------------------------------------------
```

filename filetype specify the file to be typed.

n1 is the line number of the first line to be typed.

n2 is the line number of the last line to be typed.

n3 is the maximum number of characters to be typed on a line, if the records are to be truncated.

Usage:

The filename and filetype must be specified. If the first line number and last line number are omitted, or specified with asterisks, the entire file is typed. An asterisk in the first line or end line fields specifies the beginning or the end of the file, respectively.

Typed lines are truncated to the specified limit, if any, or to 113 characters for LISTING files, 120 for SCRIPT files, 80 for MEMO files, or to 72 for all other filetypes. If a limit is specified, the first line number and last line number fields must be filled, either explicitly, or with asterisks.

The standard order of search is used to find the file. In the case of files with duplicate filename and filetype, only the first file found is typed.

Notes:

a. The first line number and last line number must be less than 9999, and may not contain imbedded commas.

b. The first character of each line in a LISTING file is not typed. This is a printer carriage-control character.

c. The KT command overrides any specified last line number or line length.

Examples:

These are given in Figures 8, 9, and 10.

```
printf go exec

LOAD &1
START

R; T=0.27/0.53  10.40.16

printf go exec * * 80

LOAD &1                                                GO 00010
START                                                  GO 00020

R; T=0.27/0.55  10.46.32
```

Figure 8. Two examples of PRINTF commands that type out an entire file

```
printf syslib maclib 157 171 72

         MACRO
&LABEL   MADDPL    &COMM=*,&NAME=*,&TYPE=*,&MODE=P1,&ITNO=0,
                &BUFF=*,&SIX ZE=80,&FV=F,&NOIT=1
&LABEL   DS     0D
&LABEL.COMM DC CL8'&COMM'      COMMAND
&LABEL.NAME DC CL8'&NAME'      FILE-NAME
&LABEL.TYPE DC CL8'&TYPE'      FILE-TYPE
&LABEL.MODE DC CL2'&MODE.      FILE-MODE
&LABEL.ITNO DC H'&ITNO'        ITEM NUMBER
&LABEL.BUFF DC A(&BUFF)        BUFFER AREA
&LABEL.SIZE DC A(&SIZE)        BUFFER SIZE
&LABEL.FV DC   CL2'&FV'        FIXED/VARIABLE FLAG
&LABEL.NOIT DC H'&NOIT'        NUMBER OF ITEMS
&LABEL.NORD DC F'0'            NUMBER OF BYTES ACTUALLY READ
         MEND

R; T=0.50/0.72  10.56.18
```

Figure 9. A PRINTF command that types out a macro definition

```
printf fortj listing 33 * 72

                                FORMAT STATEMENT MAP
 SYMBOL   LOCATION   SYMBOL   LOCATION   SYMBOL   LOCATION
      5      38C         20      392           8      398

     TOTAL MEMORY REQUIREMENTS 00057E BYTES

R; T=0.33/0.47  10.59.42
```

Figure 10. A PRINTF command that types out the bottom of a FORTRAN LISTING file

Error Messages:

E(00001) CORRECT FORM IS: 'PRINTF' FILENAME
FILETYPE STARTLINE ENDLINE LINE-LIMIT,
WHERE 'STARTLINE', 'ENDLINE', AND
'LINE-LIMIT' ARE OPTIONAL.
The filename or filetype was omitted, or one of the optional fields was not valid.

E(00002) DISK ERROR.
An I/O error occurred. It may be necessary to initialize the disk again (see FORMAT).

E(00003) FILE NOT FOUND.
No file with the specified filename and filetype exists.

SCRIPT

Purpose:

The SCRIPT command outputs to a printer, file, or terminal a file of variable-length records in a format specified by included control words.

Format:

```
-------------------------------------------------
|   SCRIPT       | filename (option1...optionN) |
-------------------------------------------------
```

filename specifies a file with a filetype of SCRIPT.

Options:

CENTER (CE) causes offline output to be centered on the printer paper.

FILE (FI) prints the edited and formatted output of SCRIPT into a file named ".filename", instead of at the terminal or offline printer.

NOWAIT (NO) starts SCRIPT output immediately without waiting for the first page to be adjusted.

NUMBER (NU) prints in the left margin the SCRIPT filename and line number corresponding to each line of printed output.

OFFLINE (OF) prints the edited and formatted output of SCRIPT on the offline printer, instead of at the terminal.

PAGExxx causes printout to start at page xxx.

SINGLE (SI) terminates printing after one page, usually used in conjunction with the PAGExxx option.

STOP (ST) causes a pause at the bottom of each page during SCRIPT.

TRANSLATE (TR) translates lowercase letters to uppercase in printout.

UNFORMATTED (UN) prints the inputted SCRIPT file along with the control words; the control words being ignored with no formatting of the output.

Usage:

Filename must be specified with the SCRIPT command. The filetype SCRIPT is assumed.

When the SCRIPT command is issued, the specified SCRIPT file is typed either at the user's terminal, on the offline printer, or into a file. Execution is controlled by format control words included in the specified SCRIPT file. When the file is located, and typing is ready to begin, a response is typed, and execution pauses until a carriage return is entered at the terminal, unless the NOWAIT, OFFLINE, or FILE option has been specified. This pause allows the user to position the output paper at the top of a page. If STOP is specified with the command, the pause is repeated at the bottom of each page, allowing the user to change paper if noncontinuous forms are being used. If STOP is used, the paper should be positioned to the first line to be printed (the heading) rather than to the physical top of the page. Typing resumes when a carriage return is typed.

The TRANSLATE option is needed if output is to be directed to an offline printer that is not equipped with the uppercase and lowercase letters (TN-chain). In conjunction with the UNFORMATTED option, TRANSLATE provides a means of printing the original SCRIPT file on a printer that does not have the TN-chain (this can also be done by the CMS command OFFLINE PRINTUPC).

The PAGExxx option, in conjunction with the SINGLE option, provides a means for selectively formatting and printing portions of a manuscript. The xxx represents a three-digit page number and must include leading zeros (for example, page 12 only should be requested by SINGLE PAGE012). Another means of selectively manipulating a formatted manuscript is to use the FILE option to generate the entire or relevant portion of a manuscript into a file and then use the CMS facilities of EDIT and/or PRINTF to process it.

The FILE option produces an output file in either typewriter format (backspace characters and carriage return characters are used) or printer format (printer control codes are used). The default format is typewriter. The printer format can be specified by the combination of both the FILE and OFFLINE options. A printer format file may be later printed by the CMS command OFFLINE PRINTCC.

Each line read from the disk file by SCRIPT is inspected for a first character of ".", which identifies a format control word. Format control words are not typed, but are interpreted to specify changes in the output format. Control words may be entered in uppercase or lowercase and should be separated from their operands (if any) by one or more blanks.

Control words may appear at the beginning of any line in the file, with changes effective below the points at which they occur. No input data should be included on lines containing control words, since this data could, in some cases, be lost or interpreted as an operand of the control word.

Note.

The TAB key generates an acceptable character in a SCRIPT file, and is transmitted by SCRIPT. The number of spaces actually skipped on print output is dependent on the logical tab setting specified by the .TB command. For indentations, the .IN or .OF control words should be used instead of the TAB key.

References:

For more detailed information on SCRIPT, see CMS SCRIPT User's Manual.

Response:

LOAD PAPER; HIT RETURN
This response is given whenever the SCRIPT command is issued without specifying the NOWAIT option. The carriage return must be hit in order for SCRIPT processing to continue. The paper should be adjusted first.

Example:

An example of SCRIPT input and output is given in Figures 11 and 12, which follow the descriptions of SCRIPT control words.

Error Messages:

E(00004) INCORRECT PARAMETER LIST
An invalid parameter has been specified for a SCRIPT control word, or a required parameter has been omitted. The SCRIPT command has been terminated.

E(00008) FILE xxxxxxxx NOT FOUND.
No SCRIPT version of filename xxxxxxxx was found. The SCRIPT command has been terminated.

E(00012) DISK ERROR WHILE READING
A disk error was incurred by SCRIPT. The SCRIPT command has been terminated, and the disk file being printed remains unchanged.

E(00016) ILLEGAL CONTROL CARD ENCOUNTERED.
An unrecognizable control word was encountered while printing a SCRIPT file. The SCRIPT command has been terminated.

For all of the error messages from SCRIPT, the following message is printed:

ERROR OCCURRED AFTER READING XXXX LINES.

This usually assists in finding the error in the SCRIPT file.

E(00000) *** A TERMINAL ERROR HAS OCCURRED WHILE PROCESSING ON OR AROUND LINE xxxxxxxx *** , ETC.

This message indicates a system error. The appropriate personnel should be informed of the circumstances. Usually this condition can be bypassed by diagnosing the cause of the error and changing the SCRIPT file.

SCRIPT Control Words

SCRIPT control words are interpreted by the SCRIPT command to govern format control as the file is being printed out.

The SCRIPT control words are listed below.

Control Word	Meaning
.AP	Append
.BM	Bottom margin
.BR	Break
.CE	Center
.CM	Comment
.CO	Concatenate mode
.CP	Conditional page
.DS	Double space mode
.FI	Format (old form)
.FO	Format mode
.HE	Heading
.HM	Heading margin
.IM	Imbed
.IN	Indent
.JU	Justification mode
.LL	Line length
.NC	No concatenate mode
.NJ	No justification mode
.OF	Offset
.PA	Page eject
.PL	Page length
.PN	Page numbering mode
.RD	Read from terminal
.SP	Space lines
.SS	Single space mode
.TB	Tab settings
.TM	Top margin
.UN	Undent

APPEND Control

Purpose:

The APPEND control word allows an additional SCRIPT file to be appended to the file just printed.

Format:

```
-------------------------------
|   .AP     |    filename     |
-------------------------------
```

filename specifies the name of the SCRIPT file to be appended to the file which has just been printed.

Usage:

When the .AP control word is encountered, the current file is closed, and the specified SCRIPT file is printed as a continuation of the SCRIPT output from the previous file.

Note.

The .AP control word only allows files to be appended to the end of the current file. If it is desired to insert file contents into the printout of the current file, use the .IM control word.

Example:

```
 .AP  ABC
```

The contents of SCRIPT file ABC are typed immediately following the last line of the current file which precedes the .AP request.

BOTTOM MARGIN Control

Purpose:

The BOTTOM MARGIN control word specifies the number of lines to be skipped at the bottom of output pages, overriding the standard value of three.

Format:

.BM	n

n specifies the number of lines to be skipped at the bottom of output pages. If omitted, 3 is assumed.

Usage:

This control overrides the standard bottom margin size of three lines, and need not be included in the file if that value is satisfactory. It may be included anywhere in the file, and the most recent value set applies on any page.

Note.

The BOTTOM MARGIN control word also acts as a BREAK.

Example:

```
.BM 10
```
Ten lines are left blank at the bottom of the current page (if possible), and on all subsequent pages.

BREAK Control

Purpose:

When CONCATENATE is in effect, BREAK causes the previous line to be typed without filling in words from the next line.

Format:

```
---------------
|     .BR     |
---------------
```

Usage:

BREAK is used to prevent concatenation of lines, such as paragraph headings or the last line of a paragraph. It causes the preceding line to be typed as a short line, if it is shorter than the current line length.

Notes:

a. Many of the other control words have the effect of a BREAK. No BREAK is necessary when one of these is present.

b. A leading blank or tab on a line has the effect of a BREAK.

Example:

```
 Heading:
 .br
 First line of paragraph . . .
```

This part of a file is printed by SCRIPT as

```
     Heading:
     First line of the paragraph . . .
```

If the BREAK control word were not included, it would be typed

```
     Heading:  First line of the paragraph . . .
```

CENTER Control

Purpose:

The line following the CENTER control word is centered between the margins.

Format:

```
---------
|  .ce  |
---------
```

Usage:

The line to be centered is entered on the line following the CENTER control word. It starts at the left margin, and leading or trailing blanks are considered part of its length.

Notes:

a. The CENTER control acts as a BREAK.

b. If the line to be centered exceeds the current line length value, it is truncated.

Example:

```
 .CE
Other Methods
```

When this line of the file is typed, the characters "Other Methods" are centered between the margins.

COMMENT Control

Purpose:

The COMMENT control word causes the remainder of the line to be ignored, allowing comments to be stored within the SCRIPT file.

Format:

```
-------------------------
|   .CM   | comments    |
-------------------------
```

Usage:

The .CM control word allows comments to be stored in the SCRIPT file for future reference. These comments can be seen when editing the file, or printing the file under UNFORMAT mode.

The comments may also be used to store unique identifications that can be useful when attempting to locate a specific region of the file during editing.

Example:

```
 .CM   Remember to change the date.
```

The line above is seen when examining an unformatted listing of the SCRIPT file, and it reminds the user to update the date used in the text.

CONCATENATE Control

Purpose:

CONCATENATE cancels a previous NO CONCATENATE control word, causing output lines to be formed by concatenating input lines and truncating at the nearest word to the specified line length.

Format:

```
------------
|   .CO     |
------------
```

Usage:

The CONCATENATE control specifies that output lines are to be formed by shifting words to or from the next input line. The resulting line is as close to the specified line length as possible without exceeding it or splitting a word; this resembles normal typist output or the MT/ST. This is the normal mode of operation for the SCRIPT command, CONCATENATE is only included to cancel a previous NO CONCATENATE control word.

Note:

This control acts as a BREAK.

Example:

```
 .CO
Output from this point on in the file is formed to approach
the right margin without exceeding it.
```

CONDITIONAL PAGE Control

Purpose:

The CONDITIONAL PAGE control word causes a page eject to occur, if less than the specified number of lines remain on the current page.

Format:

```
-------------------
| .CP |    n    |
-------------------
```

n specifies the number of lines that must remain on the current page for additional lines to be printed on it.

Usage:

The .CP control word causes a page eject to occur if n lines do not remain on the current page. This request is especially meaningful (1) before an .SP control word to guarantee that sufficient space remains on the current page for the number of spaces requested along with any titles, and (2) preceding a section heading to eliminate the possibility of a heading occurring as the last line of a page.

Note:

If no operand is specified with the .CP request, the request is ignored.

Example:

```
 .CP  10
```
If less than ten lines remain on the current page, an eject is issued before printout continues. If ten or more lines remain, printout continues on the current page.

DOUBLE SPACE Control

Purpose:

The DOUBLE SPACE control word causes a line to be skipped between each line of typed output.

Format:

```
---------
|  .DS  |
---------
```

Usage:

DOUBLE SPACE may be included anywhere in the file to force double spaced output.

Note:

This control word has the effect of a BREAK.

Example:

```
 .DS
```
Blank lines are inserted between output lines below this point in the file.

FORMAT Control

Purpose:

The FORMAT control word cancels a previous NO FORMAT control word (or NO CONCATENATE and/or NO JUSTIFY control word), causing concatenation and right justification of output lines to resume.

Format:

```
---------     ---------
| .FI   | or  | .FO   |
---------     ---------
```

Usage:

The FORMAT control word is a shorthand way to specify the two control words: CONCATENATE and JUSTIFY. This control specifies that lines are to be formed by shifting words to or from the next line (concatenate) and padded with extra blanks to produce an even right margin (justify). Since this is the normal mode of operation for the SCRIPT command, FORMAT is only included to cancel a previous NO FORMAT control word.

Notes:

a. This control acts as a BREAK.

b. If a line without any blanks exceeds the current line length, it is truncated.

c. The .FI form of the control word is provided for compatibility with old SCRIPT file and should not be used in new files.

Example:

```
.FO
Output from this point on in the file is padded to produce
an even right margin on the output page.
```

HEADING Control

Purpose:

The HEADING control word specifies a heading line to be typed at the top of subsequent output pages.

Format:

```
---------------------
|  .HE  |   line    |
---------------------
```

line specifies the heading to be printed at the top of subsequent pages.

Usage:

All of the line following the first blank after the HEADING control word is printed at the top of pages starting after the control word is encountered. No heading is typed on the first page of an output file. The heading is typed at the left margin. Its length must be at least ten less than the output line length, to allow for a page number at the right margin. Leading blanks may be used to center the heading. The heading is typed in the line specified by the heading margin and top margin control words. Additional .HE control words may be included at any point in the file to change the heading on subsequent pages.

Note:

If a new heading is to be placed on a page forced with the PAGE control word, the HEADING control must precede the PAGE control.

Examples:

a. .HE CAMBRIDGE MONITOR SYSTEM
The characters CAMBRIDGE MONITOR SYSTEM are typed at the left in the second-last line of the top margin on all pages started after this point in the file:

CAMBRIDGE MONITOR SYSTEM PAGE 7

b. .he CMS
The leading blanks are considered part of the heading, so the characters CMS are centered in the heading line

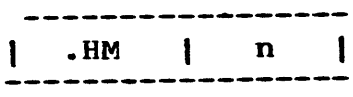
CMS PAGE 8

HEADING MARGIN Control

Purpose:

The HEADING MARGIN control word specifies the number of lines to be skipped between the heading and the first line of text excluding forced space (TOP MARGIN), overriding the standard value of 1.

Format:



n specifies the number of lines to be skipped after the heading line.

Usage:

The heading line is placed a specified number of lines above the top margin. If no HEADING MARGIN control word is included in the file, the default value is 1.

The HEADING MARGIN specified must always be less than the current TOP MARGIN.

Note:

This control word acts as a BREAK.

Examples:

a. .HM 3
Three lines are left between the heading line and the first line of text. If default top margin of 5 is in effect, the heading occurs one line from the top of paper, followed by three more blank lines (the heading margin), and then the text.

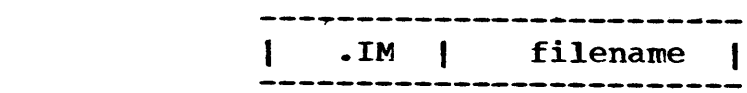
b. .HM 1
The standard heading margin of 1 is set.

IMBED Control

Purpose:

The IMBED control word is used to insert the contents of a specified file into the printout of another SCRIPT file.

Format:



filename specifies the file to be currently formatted into the printout. A filetype of SCRIPT is assumed.

Usage:

The .IM and .AP control words perform similar functions, but .IM allows the contents of a second file to be inserted into the printout of an existing file, rather than appended to the end of it. Imbedding may be used to insert standard sets of control words at desired spots in a file, as well as for many other purposes.

Example:

```
.IM CHAP4
```

The contents of the SCRIPT file whose filename is CHAP4 are inserted in the printout of the current SCRIPT file; when the end of the CHAP4 file is reached, printout of the current file resumes.

INDENT Control

Purpose:

The INDENT control word allows the left side of the SCRIPT printout to be indented.

Format:

.IN	n

n specifies the number of spaces to be indented. If omitted, indentation reverts to the original margin.

Usage:

The .IN control word causes SCRIPT printout to be indented n spaces from the current left margin setting. This indentation remains in effect for all following lines (including new paragraph and pages), until another .IN control word is encountered. ".IN 0" cancels the indentation, and printout continues at the original left margin setting.

Notes:

a. The .IN request acts as a BREAK.

b. The .IN request resets the effective left margin, causing any .OF setting to be cleared. The .OF request may be used alone, or in conjunction with .IN. When the latter is the case, .IN settings take precedence.

Examples:

a. .IN 5
All lines printed after this request are indented five spaces from the current left margin setting. This indentation continues until another .IN control word is encountered.

b. .IN 0
The effect of any current indentation is canceled, and printout continues at the original left margin setting.

JUSTIFY Control

Purpose:

The JUSTIFY control word cancels a previous NO JUSTIFY control word (or part of a NO FORMAT control word), causing right justification of output lines to resume.

Format:

```
---------
|  .JU  |
---------
```

Usage:

This control word specifies that lines are to be justified by padding with extra blanks. If concatenate mode is in effect, the concatenation process occurs before justification. Since this is the normal mode of operation for the SCRIPT command, JUSTIFY is only included to cancel a previous NO JUSTIFY control word, or the NO JUSTIFY part of a NO FORMAT control word.

Notes:

a. This control acts as a BREAK.

b. If a line exceeds the current line length, and CONCATENATE mode is not in effect, the line is printed as is.

c. This control word is seldom used without CONCATENATE mode, therefore, FORMAT should be used to enter JUSTIFY and CONCATENATE mode.

Example:

```
 .JU
Output from this point on in the file is padded to produce
an even right margin on the output page, as long as the
input lines do not exceed the line length.
```

LINE LENGTH Control

Purpose:

The LINE LENGTH control word specifies a line length that is to override the standard line length of 60 characters.

Format:

```
----------------
| .LL   | n  |
----------------
```

n specifies output line length not greater than 120 characters.

Usage:

The LINE LENGTH control sets the length for output lines until the next LINE LENGTH control word is encountered. If no LINE LENGTH control is included in a file, the standard line length of 60 characters is used.

In the JUSTIFY/NO CONCATENATE mode, lines shorter than line length are justified to length by blank padding.

In the CONCATENATE mode, lines longer than line length are spilled into the following line; lines shorter get words from previous, or following lines, to approach line length.

Note:

This control acts as a BREAK.

Example:

```
 .LL 50
Succeeding lines are no more than 50 characters in length.
```

NO CONCATENATE Control

Purpose:

The NO CONCATENATE control stops words from shifting to or from the next line.

Format:

```
----------
|  .NC    |
----------
```

Usage:

The NO CONCATENATE control word stops words from shifting to and from the next line. There is a one-to-one correspondence between the words on the input and output lines. This control word is useful for sections of files containing tabular information, or other special formats.

Note:

This control acts as a BREAK.

Example:

```
 .NC
Concatenation is completed for the preceding line or lines,
but following lines are typed without moving words to and
from lines.
```

NO FORMAT Control

Purpose:

The NO FORMAT control stops the CONCATENATE and JUSTIFY mode, causing lines to be typed just as they appear in the file.

Format:

```
--------
| .NF  |
--------
```

Usage:

The NO FORMAT control is a shorthand way to specify the two control words: NO CONCATENATE and NO JUSTIFY. This stops line justification and concatenation until a FORMAT, JUSTIFY, or CONCATENATE control word is encountered. This control is useful for sections of files containing tabular information or other special formats.

Note:

This control acts as a BREAK.

Example:

```
.NF
```
Justification and concatenation are completed for the preceding line or lines, but following lines typed exactly as they appear in the file.

NO JUSTIFY Control

Purpose:

The NO JUSTIFY control stops padding lines to cause right justification of output lines.

Format:

```
--------
|  .NJ |
--------
```

Usage:

The NO JUSTIFY control word stops the padding of lines with additional blanks. If CONCATENATE mode is in effect, lines are formed that approach the current line length but are not forced to the exact length. The resulting lines resemble the output usually produced by a typist or an MT/ST (Magnetic Tape/Selectric Typewriter).

Note:

This control acts as a BREAK.

Example:

```
 .NJ
```
Justification is completed for the preceding line or lines, but following lines are typed without inserting additional blanks to pad the line.

OFFSET Control

Purpose:

The OFFSET control word provides a technique for indenting all but the first line of a section.

Format:

.OF	n

n specifies the number of spaces to be indented after the next line is printed. If omitted, indentation reverts to the original margin setting.

Usage:

The .OF control word may be used to indent the left side of the printout. Its effect does not take place until after the next line is printed, and the indentation remains in effect until an indent or another offset control word is encountered.

The .OF control may be used within a section which is also indented with the .IN control. Note that .IN settings take precedence over .OF, however, and any .IN request causes a previous offset to be cleared.

If it is desired to start a new section with the same offset as the previous section, it is necessary to repeat the .OF n request.

Notes:

a. This control acts as a BREAK.

b. Two OFFSET control words without an intervening text line constitute an error condition.

Examples:

a. .OF 10
The line immediately following the .OF control word is printed at the current left margin. All lines thereafter (until the next indent or offset request) are indented ten spaces from the current margin setting.

b. .OF
The effect of any previous .OF request is canceled, and all printout after the next line continues at the current left margin setting.

PAGE Control

Purpose:

PAGE causes the output form to be advanced to the next page.

Format:

```
-----------------
| .PA   | n     |
-----------------
```

n specifies the page number of the next page. If n is not specified, sequential page numbering is assumed.

Usage:

Whenever a PAGE control word is encountered, the rest of the current page is skipped. The paper is advanced to the next page, the heading and page number are typed, and output resumes with the line following the PAGE control word. If STOP was specified with the SCRIPT command, a carriage return must be entered when the bottom of the page is reached.

Notes:

a. This control acts as a BREAK.

b. If the heading, line length, or other format parameters are to be different on the new page, the appropriate control words must appear before the PAGE control word.

Examples:

a. .PA
The rest of the current page is skipped. The heading and page number are typed in the top margin of the next page, and output resumes.

b. .PA 5
Regardless of the number of the current page, the rest of that page is skipped, the heading and page number 5 are typed in the top margin of the next page, and output resumes.

PAGE LENGTH Control

Purpose:

The PAGE LENGTH control word specifies the length of output pages in lines. The value specified overrides the standard page length of 66 lines.

Format:

```
-----------------
| .PL  |  n  |
-----------------
```

n specifies the length of output pages in lines.

Usage:

The PAGE LENGTH control word allows varying paper sizes to be used for output. If no PAGE LENGTH control word is included in a file, 66 is the default value. This is the correct size of standard typewriter paper for terminals typing eight lines per inch. Page length may be changed anywhere in a file, with the change effective on the first page started after the control word is encountered.

Note:

This control word acts as a BREAK.

Example:

```
 .PL 51
```
Page length is set to 51 lines. This is the correct size for a terminal typing six lines per inch.

PAGE NUMBER Control

Purpose:

The PAGE NUMBER control word allows the user to control both external and internal page numbering of the file being printed.

Format:

.PN	OFF
	OFFNO
	ON

OFF suppresses external page numbering, although internal page numbering continues.

OFFNO suppresses both external and internal page numbering.

ON causes external page numbering to be resumed.

Usage:

.PN is used to control the page-numbering feature of the system. If the OFF operand is specified, page numbering is discontinued on the printout, although the page numbers continue to be incremented internally. The OFFNO operand discontinues page numbering on the printout and stops the internal incrementation of page numbers. When the ON operand is specified, page numbering resumes from the last internal page number.

Examples:

a. .pn off
No further page numbers will appear on SCRIPT output, although the internal page count continues to be incremented for each page printed.

b. .PN OFFNO
No page numbers will appear on SCRIPT output, and the internal page count remains at its current setting without further incrementation.

c. .PN ON
Page numbering on SCRIPT output resumes using the current internal page count; this count is incremented for each page printed.

READ Control

Purpose:

The READ Control word allows the user to enter a line from the terminal during SCRIPT output.

Format:

```
-------------------
|  .RD   |    n   |
-------------------
```

n specifies the number of lines to be read at the terminal. If omitted, 1 is assumed.

Usage:

When the .RD control word is encountered during SCRIPT output to the terminal, it acts as a BREAK, spins the type head several times, and unlocks the keyboard for a line of input. The line entered is ignored by the program, and no formatting occurs on it. This facility is useful for adding headings to form letters, etc.

As many .RD's may be used as wanted; each results in a separate line accepted at the terminal.

Note:

This control word acts as a BREAK.

Example:

```
 .RD
```
When this control word is encountered during SCRIPT output, the type head rotates and the keyboard is unlocked to allow one line to be typed at the terminal.

SPACE Control

Purpose:

The SPACE control word generates a specified number of blank lines before the next typed line.

Format:

```
----------------
| .SP  |   n   |
----------------
```

n specifies the number of blank lines to be inserted in the output. If omitted, 1 is assumed.

Usage:

The SPACE control word may be used anywhere in the file to generate blank lines. If page end is reached during a SPACE operation, remaining blank lines are inserted after the heading on the following page. If DOUBLE SPACE is in effect, twice as many blank lines are generated as specified.

Note:

This control acts as a BREAK.

Examples:

a. .SP 3
Three blank lines are inserted in the output before the next typed line.

b. .sp
A single blank line is inserted in the output.

SINGLE SPACE Control

Purpose:

The SINGLE SPACE control word cancels a previous DOUBLE SPACE control word, and causes output to be singlespaced.

Format:

```
---------
|  .SS  |
---------
```

Usage:

Output following the SINGLE SPACE control word is singlespaced. Since this is the normal output format, SINGLE SPACE is included in a file only to cancel a previous DOUBLE SPACE control word.

Note:

This control word acts as a BREAK.

Example:

```
 .SS
Singlespacing resumes below this point in the file.
```

TAB SETTING Control

Purpose:

The TAB SETTING control word specifies the tab stops to be assumed for the following lines when converting the TAB character generated by the TAB key into the appropriate number of spaces.

Format:

.TB	n(1) n(2) n(3) n(4) n(5)

n(i) specifies the column location of the (i)th tab stop; the sequence must consist of increasing positive values separated by one or more spaces.

Usage:

TAB characters generated by the TAB key entered into the file during EDIT file creation are expanded by SCRIPT into one or more blanks to simulate the effect of a logical tab stop. The TAB SETTING control word specifies the locations of the logical tab stops, this overrides the default tab stops of 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75.

A TAB SETTING control word without any tab stops specified, results in reversion to the default tab settings. This control word is useful for indenting the beginning of a paragraph (remember a TAB causes a paragraph BREAK), or for tabular information and diagrams.

Notes:

a. This control word acts as a BREAK.

b. The tab settings must be monotonically increasing. Tab settings that are not so ordered result in unpredictable behavior.

Examples:

a. .TB 10 20 30 40
Tab stops are interpreted as columns 10, 20, and 30.

b. .TB
Tab stops revert to default values of 5, 10, 15, etc.

TOP MARGIN (

Purpose:

The TOP MAR(
be skipped
standard va]

Format:

```
-------
|  .TM
-------
```

n specifie:
output pa

Usage:

The specifi
succeeding (
page number
margin. If
file, the de
always be g:

Note:

This control

Example:

```
.TM 3
Three lines
current page
second line
```

```
.tm 10
.ce
SCRIPT Example
.sp 2
.ds
This example will demonstrate some of the capabilitie
of the SCRIPT command.  This file was created by
issuing:
.br
EDIT EXAMPLE SCRIPT
.br
Since the file did not previously exist, the terminal
was placed directly into the input environment.  This
paragraph was double-spaced with the .DS control.
.ss
.sp
No BREAK was needed here, since the .SS (SINGLE-
SPACE) control acts as a break.  Although this is
in FORMAT mode, tabular information can be included:
.sp
        SPACE           .SP     .sp
        SINGLE SPACE    .SS     .ss
        DOUBLE SPACE    .DS     .ds
.sp
The leading blanks caused each line to be handled
separately.
.sp
Use of the LINE LENGTH control allows space
to be left within a page for figures or drawings.
Naturally, it may take some experimentation for
finding how many paragraphs will fit next to a
figure.
.ll 30
.sp
The new line length must take affect
at a paragraph, since it acts as a BREAK.  The
switch back to standard line length, usually 60,
also is a BREAK, and must end a paragraph.  This
works only in FORMAT mode.
.ll 60
.nf
By switching out of FORMAT mode                 CAPTION
and doing some justification
by eye, fancier effects can be obtained.  This also
takes some practice and experimentation.
.fi
.cp 5
```

Figure 11. Contents of a SCRIPT file

```
PARAGRAPHS:
.br
If no space follows a paragraph heading, and if the
paragraphs are not indented, a BREAK is necessary in
FORMAT mode, to keep the heading line from being justified.
        A few leading blanks are the easiest way to force
a BREAK and separate paragraphs.  A line with only a blank
will also force a BREAK and a blank line, if the following
line also begins with a blank, as follows:


        The CENTER control is handy for small figures
included in the text.  A .CE in front of each line of
the figure is necessary, and note that leading or trailing
blanks count for figuring the length to be centered:
.sp
.ce
- - - - - - - - - - - - - - - -
.ce
|  FORMAT  |  EXAMPLE   n  |
.ce
|          |  E            |
.ce
- - - - - - - - - - - - - - - -
.ce
Figure EX.A
.sp
To offset the caption it would be necessary to leave
trailing or leading blanks, which are counted as part
of its length:
.sp
.ce
- - - - - - - - - - - - - - - -
.ce
Figure EX.A
.sp
The above caption has 14 trailing blanks, which move
it to the left.  Leading blanks would move it to the right.
```

Figure 11 (cont.) Contents of a SCRIPT file..

This example will demonstrate some of the capabilities of the SCRIPT command. This file was created by issuing:

```
EDIT EXAMPLE SCRIPT
```

Since the file did not previously exist, the terminal was placed directly into the input environment. This paragraph was double-spaced with the .DS control.

No BREAK was needed here, since the .SS (SINGLE- SPACE) control acts as a break. Although this is in FORMAT mode, tabular information can be included:

```
SPACE           .SP     .sp
SINGLE SPACE    .SS     .ss
DOUBLE SPACE    .DS     .ds
```

The leading blanks caused each line to be handled separately.

Use of the LINE LENGTH control allows space to be left within a page for figures or drawings. Naturally, it may take some experimentation for finding how many paragraphs will fit next to a figure.

The new line length must take affect at a paragraph, since it acts as a BREAK. The switch back to standard line length, usually 60, also is a BREAK, and must end a paragraph. This works only in FORMAT mode.
By switching out of FORMAT mode CAPTION and doing some justification by eye, fancier effects can be obtained. This also takes some practice and experimentation.

Figure 12. SCRIPT output

PARAGRAPHS:
If no space follows a paragraph heading, and if the paragraphs are not indented, a BREAK is necessary in FORMAT mode, to keep the heading line from being justified.

A few leading blanks are the easiest way to force a BREAK and separate paragraphs. A line with only a blank will also force a BREAK and a blank line, if the following line also begins with a blank, as follows:

The CENTER control is handy for small figures included in the text. A .CE in front of each line of the figure is necessary, and note that leading or trailing blanks count for figuring the length to be centered:

```
- - - - - - - - - - - - - - -
|  FORMAT  |  EXAMPLE   n  |
|          |  E            |
- - - - - - - - - - - - - - -
```

Figure EX.A

To offset the caption it would be necessary to leave trailing or leading blanks, which are counted as part of its length:

```
- - - - - - - - - - - - - - -
```

Figure EX.A

The above caption has 14 trailing blanks, which move it to the left. Leading blanks would move it to the right.

Figure 12 (cont.) SCRIPT output

Purpose:

The SPLIT command copies a specified portion of a given file and appends it to a second file or creates a new file.

Format:

```
-------------------------------------------------------------
|          |                                    label1  label2  |
| SPLIT    | fname1 ftype1 fname2 ftype2     n1    <   n2 >   |
|          |                                           eof    |
-------------------------------------------------------------
```

fname1 ftype1 specifies the file from which a portion is copied

fname2 ftype2 specifies the name of the file to which file1 is added

label1 an eight-byte alphameric label with the first character nonnumeric, specifying the first record to be copied

label2 an eight-byte alphameric label with the first character nonnumeric, specifying the item after the last item to be copied

n1 a decimal number specifying the item number of the first item to be copied

n2 a decimal number specifying the number of items to be copied

Usage:

The SPLIT command enables the user to copy a portion of file1 and to append it to file2. File1 and file2 cannot be the same file. If file2 does not exist, it is created. The files may have fixed-length or variable-length length records. If file2 exists, and is a fixed-length record file, file1 must also be a fixed-length record file.

Copying begins at either the first record containing the alphameric string (label1), in the first eight bytes of a record (label field), or at the specified item number if the parameter consists of all numeric characters.

If the last parameter is not provided, copying continues to the end of file. If the last parameter is specified as an alphameric label, copying, once initiated, terminates immediately before the first item having the alphameric string, label2, in the label field of a record. The extent of copying may alternatively be specified by an integer count of the number of items to be copied.

No copying is done if (1) labels are used for both starting and stopping the copying and these two labels are identical, (2) the initial label or item number cannot be found, or (3) the number of items is specified as zero.

SPLIT searches all disks for the file. The new file is placed on the same disk as the original file.

Responses:

WRONG NUMBER OF PARAMETERS
The specified number of parameters given is not five or six.

INVALID LIMIT
One of the limit fields is specified with the first character numeric, and one of the other characters nonnumeric.

EOF REACHED
The end of file1 has been reached with or without copying being initiated.

FILE NOT CHANGED
The command has been completed without any writing of files.

FILE MODIFIED
The command has been successfully completed, and at least one item has been copied.

Any error encountered in the reading of file1 terminates the command after printing one of the following responses:
TYPE NOT FOUND
DISK ERROR
ILLEGAL MODE
NONSTANDARD FILE
OPEN FOR WRITING
OPEN FILE LIMIT

Any error encountered in the writing of file2 terminates the command after printing one of the following responses:
BAD OUTPUT TYPE
ERROR ON DISK
OPEN FOR READ
TOO MANY FILES
DISK FULL
READ ONLY
FILE TYPES INCOMPATIBLE...FILES NOT CHANGED

Examples:

a. SPLIT FILE DATA F1 DATA 45 12
The twelve items beginning with the 45th item are extracted from the FILE DATA file. If the F1 DATA file exists, they are appended to it. If the F1 DATA file does not exist, it is created and they become its contents.

b. SPLIT ABLE SYSIN ABLE1 SYSIN BEG 20
The 20 items beginning with the item which has a label field containing BEG are extracted from the file ABLE SYSIN and appended to the file ABLE1 SYSIN if it exists, or become the contents of the file ABLE1 SYSIN if it doesn't exist and must be created.

c. SPLIT PROG SYSIN PROGEND SYSIN END
If PROGEND SYSIN does not exist, items beginning with the item with END in the label until the end of file PROG SYSIN are used to create a new file called PROGEND SYSIN; if PROGEND SYSIN does exist, those items are appended to it.

Error Messages:

The SPLIT command diagnoses all errors which occur and prints a response message indicating the nature of the error. All returns from SPLIT are with general register 15 equal 0, indicating no error.

STATE

Purpose:

The STATE command tests whether a file exists.

Format:

```
-----------------------------------------
| STATE | filename   filetype  <filemode>|
-----------------------------------------
```

Usage:

When STATE is issued for a file which exists, the command returns with a code of zero. If the file does not exist, a nonzero error code is returned.

Error Codes:

E(00001)
File specified does not exist.

E(00004)
First character of filemode illegal.

UPDATE

Purpose:

The UPDATE command makes changes in a specified file according to control cards in a second file.

Format:

```
|UPDATE| filename1 <filetype1 <filename2 <filetype2>>> <( options )>|
```

filename1 is the name of the file to be changed.

filetype1 is the type of the file to be changed. If omitted, SYSIN is assumed.

filename2 is the filename of the file containing the UPDATE control cards. If omitted, filename1 is assumed.

filetype2 is the filetype of the file containing the UPDATE control cards. If omitted, UPDATE is assumed.

Options:

P specifies that the file incorporating the changes is to replace the original file. If omitted, the old file is retained unchanged, and the new file receives a filename consisting of a period (.), followed by the first seven characters of the original filename.

SEQ8 specifies that sequencing is to be done on all eight characters in columns 73 to 80.

INC specifies that the sequence numbers in columns 73 to 80 of the UPDATE deck are to be placed in the new SYSIN deck.

Control Cards:

Changes are made in the original file according to the UPDATE control cards in the UPDATE file. The format of these cards is shown below:

```
|./ S  segno1  increment  label |
```

S specifies that the new file is to be sequenced in columns 76-80. If this card is included in the UPDATE file, it must be the first card.

segno1 specifies the starting sequence number.

increment specifies the increment to be added to the sequence number for each item.

label is a three-character label to be placed in columns 73-75.

```
-------------------------
| ./  D  segno1  segno2 |
-------------------------
```

D specifies that cards are to be deleted from the original file.

segno1 is the (original) sequence number of the first card to be deleted.

segno2 is the sequence number of the last card to be deleted. If omitted, only one card is deleted.

```
-----------------
| ./  I  segno1 |
-----------------
```

I specifies that cards are to be inserted in the original file. The inserted cards must follow this ./ I card immediately in the UPDATE file. All cards, until the next control card, are inserted.

segno1 specifies the sequence number of the item after which the cards are to be inserted.

```
-------------------------
| ./  R  segno1 segno2  |
-------------------------
```

R specifies that cards are to be inserted in the original file in place of cards now there.

segno1 specifies the first card to be replaced.

segno2 specifies the last card to be replaced. The cards to be inserted in place of those deleted (not necessarily the same number) must follow the ./ R card immediately in the UPDATE file.

Usage:

UPDATE modifies the specified file according to control cards in a second file. The filetype SYSIN is assumed for the file to be modified, if no other is specified. The control-card file normally has the same name as the file to be modified, and has the filetype UPDATE. It is referred to as the UPDATE file, with the understanding that both a different filename and filetype may be specified with the UPDATE command. Note that if different identifiers are

specified, the filetype of the file to be modified must also be included. The options must always be the last arguments specified if they are to be included.

UPDATE generates two files during execution: "filename UPDLOG P5" and "filename INTER P5" where filename is that of the original file in both cases. The UPDLOG file contains a record of the control cards in the UPDATE file, items added to and deleted from the original file, and error messages. A new UPDLOG file is generated on each execution, replacing any existing UPDLOG file with the same filename.

The INTER file receives the records of the original file as changes are made. At the end of execution, the identifiers of the INTER file are changed to one of two formats. If (P) is specified, the original file is erased, and the INTER file receives its filename and filetype. If (P) is not specified, the original file remains unchanged on the permanent disk. new file receives the same filetype and filemode, and a filename composed of a period (.) plus the first seven characters of the original filename.

The control cards of the UPDATE file always refer to the items of the original file by the sequence numbers existing before any changes in columns 76-80. If no sequence numbers exist, issue a preliminary UPDATE command with only the ./ S control card in the UPDATE file. If the SEQ8 option is specified, the sequence numbers referred to are in columns 73 to 80. Sequence numbers will be assigned. The control cards must always be identified by a ./ in columns 1 and 2, but any number of blanks may separate the other fields. Sequence numbers may be expressed with up to five digits, unless SEQ8 is specified. Leading zeros are not necessary. Any sequence numbers in cards to be inserted in the file are ignored unless INC is specified, in which case this number is placed in the new SYSIN. If the ./ S control card is omitted from the UPDATE file, and INC is not specified, asterisks are placed in columns 73-80 of all cards in the new file which were added or replaced, to indicate where changes were made.

Changes are made in order in a single pass through the file. If control cards specify changes that are not in order, an error is recorded, and no changes are made.

Responses:

INTERMEDIATE FILE EXISTS.
The file "filename INTER P5" already exists for the filename specified. ERASE or ALTER this file, and issue the UPDATE command again.

FATAL ERROR 1
A control card was detected in the UPDATE file whose second field was not the character R, I, D, or S.

FATAL ERROR 2
The file to be changed is not on the permanent disk.

READ ERROR or WRITE ERROR
An error occurred while reading or writing to the permanent disk.

PARAMETER ERROR
No parameters were entered with the command.

filename filetype NOT FOUND
The file identified in the response was not found in the user's file directory.

ERRORS ENCOUNTERED. SYSIN REMAINS UNCHANGED.
This response is issued for all of the above error conditions. It indicates control is about to return to the CMS command environment, and that no changes have been made to the files.

Example:

UPDATE RET
Assume that the file RET SYSIN P5 contains these items:

```
RET      CSECT                              RET00010
         BALR   12,0                        RET00020
         USING  *,12                        RET00030
         SR     15,15                       RET00040
         END                                RET00050
```

Assume that the file RET UPDATE P5 contains

```
./  S  100  25  RTN
./  I  10
           ENTRY RETCODE
./  R  40
           L     15,RETCODE
           BR    14
RETCODE    DS    F
```

As the command is executed, the file RET INTER P5 is created. As items are placed into it, RTN is placed in columns 73-75, and sequence numbers, beginning with 00100 and incrementing by 25, are placed in columns 76-80. On completion, the file becomes .RET SYSIN P5, and contains

```
RET      CSECT                              RTN00100
         ENTRY RETCODE                      RTN00125
         BALR  12,0                         RTN00150
         USING *,12                         RTN00175
         L     15,RETCODE                   RTN00200
         BR    14                           RTN00225
RETCODE  DS    F                            RTN00250
         END                                RTN00275
```

RET UPDLOG P5 is also created, containing the control cards, and all items added or deleted.

Error Messages:

E(00002) FATAL ERROR 3
An error occurred while attempting to change the identification of the INTER file. Enter the command

```
ALTER fn1 INTER * fn1 filetype *
```

where fn1 is the filename of the file changed, and filetype is the desired filetype. If another error occurs, reenter the UPDATE command.

EXECUTION CONTROL

Several commands are available to the user for execution control (that is, the loading and running of programs). Files (or programs) which are to be loaded and run under CMS must reside on disk and must be either in relocatable object code form or in core-image form. A program in relocatable object code form is one whose address references can be modified to compensate for the relocation occurring when the program is loaded into core. A program in core-image form is one which represents a copy of the contents of core which would be executable. All of its address references have been resolved and it can no longer be relocated.

Output from the assembler and all compilers supported under CMS is relocatable object code. Unless an option to the contrary is specified by the user, this output is created as a file on the user's permanent disk and assigned a filetype of TEXT. All files, whose filetype is TEXT, are assumed to consist of relocatable object code and are processed accordingly. To load such files into core, either the LOAD, USE, or REUSE commands may be used. The LOAD command reads the specified file(s) from disk and loads them into core, relocating the programs and establishing the proper linkages between program segments. Several options may be specified in the LOAD command, which allow the user to specify text libraries to be searched for missing subroutines, to request that execution of the loaded program(s) begin, etc. USE and/or REUSE should be issued only after a LOAD command has been issued. The purpose of the USE command is to load the specified TEXT file(s) into core, and to establish linkages between these programs and previously loaded programs. The REUSE command performs the same function as the USE command, but has the additional effect of changing the default entry point of these programs to that of the first filename specified in the REUSE command.

A core-image copy of any information currently residing in core may be created by issuing the GENMOD command. This command creates a file on the user's permanent disk that is a copy of the contents of core between the specified locations, and assigns a filetype of MODULE to this file. All files whose filetype is MODULE are assumed to be in core-image form, and are processed accordingly. To load such files into core, the LOADMOD command is used. Since address references do not have to be resolved, the LOADMOD process is faster than the LOAD process for a given program.

After files have been loaded into core by the LOAD, USE, REUSE, or LOADMOD commands, execution may be begun by issuing the START command. Execution may also be initiated by specifying the XEQ option with LOAD.

The $ command is used to load and start a specified file, depending on its filetype, as follows: (1) if a filetype of EXEC is found, the file is assumed to consist of one or more

CMS commands, and the EXEC command is called to execute these commands; (2) if a filetype of MODULE is found, the LOADMOD command is called to load the file into core, and then the START command is called to begin execution; or (3) if a TEXT filetype is found, the file is loaded into core by a LOAD command, and START is called to begin execution.

The function of the GLOBAL command is to specify two types of libraries: libraries containing TEXT files which are to be searched by the LOAD, USE, or REUSE commands for missing subroutines and undefined names; and libraries containing macro definitions, which are to be searched by the assembler for resolving undefined macros. If the GLOBAL command is used, it should be issued before the LOAD, USE, REUSE, or ASSEMBLE commands to which it refers.

Purpose:

EXEC executes one or more CMS commands contained in a specified file, allowing a sequence of commands to be executed by issuing a single command.

Format:

```
-------------------------------------
|   EXEC  |   filename <arg1...argN>|
-------------------------------------
```

filename specifies the filename of a file containing one or more CMS command to be executed. The filetype must be EXEC.

arg1...argN are the arguments to replace the numeric variables in the file "filename EXEC".

Usage:

EXEC executes the sequence of commands that are specified in the file "filename EXEC". This file must be in card-image form, and must consist of one CMS command per card image in the same format as the command is entered at the terminal. The filetype for the specified file must be EXEC. EXEC files can be created by the EDIT or LISTF commands, or by a user's program.

Each CMS command in the EXEC file can have from one to thirty numeric variables. A numeric variable is made up of an ampersand (&) followed by an integer ranging from one to thirty, (that is, &1&2...&30). Before the command is executed, each variable is temporarily replaced by an argument specified when the EXEC command was issued. For example, each time an &1 appears as a variable in an EXEC line, the first argument specified with the EXEC command temporarily replaces the &1, the second argument specified with the EXEC command replaces &2, and so on, to argument N of the EXEC command.

If the double quotation mark (") is used in place of an argument, the corresponding variable (&N) is ignored in all the commands which reference that variable. If the specified EXEC file contains more variables than arguments given with the EXEC command, the higher numbered variables are assumed to be missing, and are ignored when the commands are executed.

Arguments can be concatenated to the right side of any word in an EXEC line. For example, the EXEC line LISTF ABC&1 FORTRAN&2 would result in LISTF ABCXYZ FORTRAN, if arg1 is XYZ and arg2 is unspecified. Use of the double quote (") for arg1 would cause the variable to be ignored leaving

LISTF ABC FORTRAN. If the single quotation mark (') is used in place of an argument, the entire concatenated form is deleted. For example, in the above EXEC line if arg1 is specified with a double quote ("), and arg2 is specified with a single quote ('), the line would be just LISTF ABC.

The EXEC command is completely recursive (that is, an EXEC file can contain other EXEC commands in its sequence of commands). The recursiveness is limited by core size--each level of recursion requiring about 1200 bytes of free storage for data. This limits the depth of recursion to approximately 16.

Notes:

a. Errors resulting from issued commands are not fatal and do not cause the sequence of commands to be terminated. This behavior may be modified by the EXEC control word &ERROR (see "Special Features of EXEC" below).

b. Each EXEC file may contain a maximum of 4095 EXEC lines.

c. This version of the EXEC command is completely compatible with EXEC files created for use with the previous version of the EXEC command, except that in this version only one command is allowed per line. This compatibility may be removed in a later version to save space in the CMS nucleus.

d. If the EXEC command is issued from an EXEC file, EXEC must be specified explicitly, as the search for commands does not include the EXEC filetype.

Response:

As each CMS command in the EXEC file is processed, it is typed at the terminal along with the time, unless the &TYPEOUT OFF control word has been specified (see "Special Features of EXEC" below).

Examples:

a. In Figure 13, the command EXEC FORTCLG LLHS is issued. LLHS is a file whose filetype is FORTRAN, and LLHS replaces the &1 in all CMS commands in the EXEC file. The file LLHS FORTRAN is compiled, and the file LLHS TEXT is loaded and executed. Note that each CMS command is typed before it is executed.

```
printf fortclg exec

FORTRAN &1 &2
LOAD &1 &2 (XEQ)

R; T=0.45/1.23 01.24.45

fortclg llhs
01.29.50 FORTRAN LLHS
01.29.55 LOAD LLHS (XEQ)
EXECUTION BEGINS.
APRIL 1968 DATA 5.320          1.920        5.600
R; T=0.55/1.44 01.30.45
```

Figure 13. Example of an EXEC file to compile, load, and execute a FORTRAN program

b. In Figure 14, the FORT EXEC is created by EDIT. The only command placed in the file is FORTRAN &1 (PRINT). The file CMS EXEC was created earlier with the LISTF command (see LISTF), and contains the sequence of FORTRAN files to be compiled. The file CMS EXEC is typed by issuing the PRINTF command. The EXEC command is issued specifying the filename CMS and the two arguments EXEC and FORT. Each file identifier in CMS EXEC is preceded by two symbolic arguments, &1 and &2. The &1 is replaced by the first argument specified with the EXEC command, which is EXEC, and the &2 is replaced by the second argument specified, which is FORT. The sequence of CMS commands generated in core by EXEC from the file CMS EXEC are then executed, the first of which is

```
EXEC   FORT   W   FORTRAN   P5   001.
```

This command executes the sequence of commands in the file FORT EXEC, and temporarily replaces the numeric variable &1 from FORT EXEC with the argument W. The arguments FORTRAN, P5, and 001 are ignored because there are no variables &2, &3, and &4 for them to replace. As soon as the sequence of commands in FORT EXEC are completed, the next command in the file CMS EXEC is executed. This sequence continues until all commands are executed in the CMS EXEC file.

```
edit fort exec
INPUT:
fortran &1 (print)

EDIT:
file
R; T=0.55/1.43 01.30.50

listf * fortran (exec)
R; T=0.40/0.50 01.31.00

printf cms exec

 &1 &2 W      FORTRAN      P5      001
 &1 &2 SUB2   FORTRAN      P5      001
 &1 &2 A      FORTRAN      P5      001
 &1 &2 SUBB   FORTRAN      P5      001

R; T=0.55/3.21 01.32.15

cms exec fort
01.32.58 EXEC    FORT     W        FORTRAN      P5      001
01.33.00 FORTRAN     W           (PRINT)
01.33.10 EXEC    FORT     SUB2     FORTRAN      P5      001
01.33.12 FORTRAN     SUB2        (PRINT)
01.33.15 EXEC    FORT     A        FORTRAN      P5      001
01.33.17 FORTRAN     A           (PRINT)
01.33.19 EXEC    FORT     SUBB     FORTRAN      P5      001
01.33.23 FORTRAN     SUBB        (PRINT)
R; T=1.50/1.80 01.33.27
```

Figure 14. The **file FORT EXEC** is created, the file CMS EXEC is typed out, **and then an implied** EXEC is issued to nest EXECs

Error Messages:

E(00001) FILE DOES NOT EXIST
The EXEC file does not exist. The EXEC command has terminated. Check to see if the filename specified has a filetype of EXEC.

E(00003) FILE HAS WRONG RECORD SIZE
The specified EXEC file does not contain 80-character records. The command is terminated.

E(00006) WAITRD OR RDBUF ERR
This error would result if an EXEC file was erased after the EXEC command had been successfully begun. For Example, with the procedure shown below the file ABCD EXEC would be erased, and the attempt to read the EXEC line containing PRINTF would result in the error. The EXEC command is terminated.

```
printf abcd exec
ERASE ABCD EXEC
PRINTF XYZ&2
R=0.02/0.13 03.45.14

exec abcd
ERASE ABCD EXEC
WAITRD OR RDBUF ERR
E(00006) T=0.05/0.08 03.46.10
```

||| E(xxxxx) |||
The error code xxxxx was generated by the CMS command issued from the EXEC file. If E(-0003) occurs, the issued command was invalid.

SPECIAL FEATURES OF EXEC

A line of an EXEC file is either a CMS command or an EXEC control line. EXEC control lines control the sequence of commands to be executed, specify what is to be typed on the console during the execution of the EXEC command, or provide input to other command programs, or to the EXEC command itself.

LABELS

EXEC lines, containing either a CMS command or an EXEC control, may be identified with a label. All EXEC labels have a dash as the first character. If the first word of an EXEC line begins with a dash (-) that word is assumed to be a label. Labels are used to control the sequence of EXEC lines executed (see "EXEC Control Words", &GOTO and &LOOP).

EXEC WORDS (&WORDS)

EXEC lines may contain words which begin with an ampersand

(&). A word beginning with an ampersand may be a numeric variable, a keyword (that is, a symbolic variable), or a control word. A numeric variable consists of an ampersand followed by an integer or an asterisk (*). A keyword word consists of an ampersand followed by a string of not more than seven characters, at least one of which is not an integer. Control words have the same form as keywords and are defined under "EXEC Control Words". Numeric variables and keywords are substituted before the EXEC line is interpreted.

Numeric Variables

Numeric variables are substituted for any arguments which are to be specified when the EXEC command is issued. The numeric variable &0 is replaced by the filename of the current EXEC file. The numeric variable &n is ignored when n is negative or greater thatn 30, or when n is greater than the number of arguments supplied when the EXEC command is issued. The variable &* is interpreted to mean all arguments specified. When it is included in a CMS command, the command is executed once for each argument specified. For example, the command line ERASE &* * would cause the erasing of all files whose filename is the same as one of the specified arguments. The variable &* may also be used in an &IF or &LOOP condition (see "EXEC Control Words").

Keyword Variables

The value substituted for a keyword may be one of two types: specified in an EXEC line by the user, or implied if the keyword is a special keyword.

EXEC-Set Keywords

A number of keywords have been defined to have special meaning and have their values set in a special way. These words and their values are described below.

&LINENUM has the value of the current EXEC line number plus one.

&INDEX1...&INDEX9 are used as indices and initially have the value +1. Indices 1-9 may be reset or incremented by an EXEC line. These indices may be set to an integer value in the same way as the value of any keyword is set. An index may be incremented or decremented by specifying the index and the increment in an EXEC line. For example:

&INDEX5 = 30975 sets &INDEX5 to 30975.
&INDEX7 -50 adds -50 to the value of &INDEX7.

Indices are local to the current level of recursion.

&INDEX0 has as its value the return code number in register 15 from the previous CMS command.

&INDEX has as its value the number of arguments given when the EXEC command was issued.

&GLOBALO...&GLOBAL9 are used for communication between levels of EXEC recursion and are set and incremented in the same way as &INDEX1... &INDEX9.

&GLOBAL has as its value the level of recursion.

User-Specified Keywords

The value of a keyword may be specified by an EXEC line of the form

```
&KEYWORD = VALUE
&ABLE = 12345
```

which defines the keyword &KEYWORD to have the value VALUE and &ABLE to have the value 12345.

Keywords can be redefined as often as desired.

EXEC Control Words

The EXEC control words described below can be used to provide a versatile and flexible facility for controlling the execution of commands and for defining a user-oriented command environment. EXEC control words appear in EXEC lines, which can be interspersed with CMS commands.

&ERROR	action
	&CONTINUE

where action is any EXEC line without a statement label. Action is executed immediately upon an error return from a subsequent CMS command. If action is not given, &CONTINUE (see below) is assumed. An error in execution of action, if action is a CMS command, results in an exit from this level of EXEC with error code of 11.

```
-------------------------------
| &IF |  condition   action   |
-------------------------------
```

where condition consists of the three parameters shown below

&*	EQ	&*
&$	NE	&$
anything	GT	anything
	LT	
	GE	
	LE	

and where action is any EXEC line without a label. If the condition is satisfied, the action is executed. The comparison specified by the second argument of condition is made between the first and third arguments. &$ is interpreted as "any of the symbolic arguments". Thus, the EXEC line

```
&IF   &$ EQ XYZ &PRINT HI
```

would cause "HI" to be typed if at least one of the arguments specified when the EXEC command was issued was XYZ. Similarly, &* is interpreted as "all of the supplied arguments". (See below for a description of &PRINT.)

A numerical comparison is made only if both the operands are numeric. For example, the EXEC line

```
&IF 017 EQ 17 &PRINT HI
```

would cause the typing of "HI". Otherwise, the comparison in a condition is a logical comparison.

An &IF can have another &IF as its action; these may be nested to level 3.

```
-------------
| &EXIT | n |
-------------
```

&EXIT causes an EXIT to the next lower level of recursion with an error code of n. If n is not given, a normal exit with a code of 0 results. If n is negative and if EXEC was called from the CMS command level, the absolute value of n is returned. If this EXEC command was called from a previous EXEC command, a negative value of n is returned as the error code in register 15.

&QUIT	n
	ON
	OFF

&QUIT n is similar to &EXIT n, except that &QUIT n returns to level 0, the CMS command level, regardless of the level of recursion of EXEC commands.

&QUIT ON sets the return level for a subsequent &QUIT control to a level of recursion one higher. Thus, if &QUIT ON is issued twice and if the current level of recursion is 5, an &QUIT n would cause a return to level 2 with an error code of n.

&QUIT OFF resets the return level to level 0, the CMS command level.

&SKIP	1
	n

&SKIP causes n lines in the file to be skipped. If n<0, the next EXEC line to be executed will be n lines before the current line. If n>0, the next EXEC line to be executed will be n+1 lines after the current line.

	TOP
&GOTO	label
	EXIT

&GOTO controls the point from which execution will continue. &GOTO TOP causes sequential execution of EXEC lines to be continued at the beginning of the EXEC file.

&GOTO EXIT is identical to &EXIT 0 and causes a return from the current level of EXEC.

&GOTO label searches the EXEC file, starting from the present EXEC line to the end of the file, then going to the beginning of the file, and finally going back to the present line location, looking for the first EXEC line beginning with the specified label. (See label description under "Special Features of EXEC".)

&LOOP	label	condition
	n1	n2

&LOOP causes looping either to and including the labeled

line, or through the number of lines specified by n1, beginning with the next line.

Looping continues either until the condition is satisfied or for n2 times. Condition is specified the same way as with the &IF control word and is tested before looping.

Loops may be nested to a depth of 4. The numbers n1 and n2 must be less than 4096.

```
-------------
| &CONTINUE |
-------------
```

&CONTINUE as an EXEC line is ignored. It may be useful with &GOTO or &LOOP and is the default action for &ERROR.

```
-------------------------------------
| &TYPEOUT | ALL   TIME     PACK    |
|          | ON    NOTIME   NOPACK  |
|          | ERROR                  |
|          | OFF                    |
|          | NOEXEC                 |
-------------------------------------
```

where:

ALL types all CMS command lines and EXEC control lines.

ON types all CMS command lines but suppresses the typing of EXEC control lines.

ERROR types only CMS command lines which result in an error; EXEC control lines are not typed.

OFF suppresses the typing of all EXEC lines.

NOEXEC is the same as OFF and is included for compatibility with EXEC files created with the previous version of the EXEC command.

TIME causes time of day to precede each CMS command line typed.

NOTIME suppresses the typing of the time of day with each CMS command line.

PACK removes excess blanks from typed lines.

NOPACK suppresses the removal of excess blanks from typed lines.

```
-----------------
| &TIME |  TYPE |
|       |  ON   |
|       |  OFF  |
-----------------
```

&TIME TYPE types the time since the previously typed time.

&TIME ON types the time message after typing each CMS command.

&TIME OFF suppresses the typing of the time after each CMS command.

```
-----------------
| &SPACE |  1  |
|        |  n  |
-----------------
```

&SPACE types n carriage returns at the console.

```
-----------------
| &PRINT | line |
-----------------
```

&PRINT prints line on the typewriter console. All keywords, symbolic arguments, etc., are substituted into the line. Any word or words that exceed eight (8) characters are left justified and truncated on the right.

```
-------------------
| &COMMENT | line |
-------------------
```

&COMMENT is used to annotate the EXEC file. It is ignored during execution.

```
------------------------
| &ARGS |<arg1...argN>  |
------------------------
```

&ARGS is used to redefine the numeric variables &1...&n with the values specified by arg1...argN. &INDEX is redefined with the value of the current number of arguments.

```
---------------
| &READ |  1  |
|       |  n  |
|       | args|
---------------
```

&READ causes a read to the typewriter console.

If n is specified, the next n EXEC lines are read from the console and executed immediately. These lines must be entered as commands as if they were included in the EXEC

file, since they are executed in the same way. Reading stops and the next EXEC line is obtained from the EXEC file either when n lines have been read, or when &GOTO, &SKIP, &LOOP, &EXIT, or &QUIT are typed. Reading may be reset by entering &READ.

If ARGS is specified, one line is read from the console. This line will be scanned and used to redefine the numeric variables. &INDEX is redefined to the number of arguments read. This is the only way to read without entering a command.

Only the first 72 characters on a line are read.

&STACK	FIFO LIFO	line

&STACK stacks line in the input buffer, substituting for keywords and variables. Subsequent &READ obtains lines which were stacked in this way.

&STACK can be used to specify input or EDIT requests to EDIT, or DEBUG requests to the Debug environment when it is entered on purpose (that is, by a breakpoint or the DEBUG command). &STACK with a blank line is executed as a null line.

FIFO specifies that the lines are stacked in First-In-First-Out order. LIFO specifies that the lines are stacked in a Last-In-First-Out order.

&BEGSTACK	FIFO LIFO

```
line 1
line 2
 .....
line N
&END STACK
```

&BEGSTACK stacks line 1 through line N, literally without truncation and without substituting for numeric variables or keywords.

This sequence may also be used to specify input or EDIT requests to the EXEC command, where a line of "#file" causes the Edit environment of EDIT to be entered from the Input environment and writes the file on disk. THis sequence may also be used to specify DEBUG requests to the DEBUG environment when that environment has been entered on purpose (that is, via a breakpoint or the DEBUG command).

FIFO and LIFO are as explained under &STACK.

```
---------------
| &SET |action|
---------------
```

&SET has been included for compatibility with old EXEC files that used the control words ERR and TYPEOUT or actions. &SET may later be removed as an EXEC control word.

Notes on EXEC Control Words

a. All numeric variables, keywords, EXEC control settings, and limitations (for example, maximum depth of loop nesting) are local to the current level of EXEC, unless otherwise noted.

b. Any EXEC control word may be abbreviated by a sufficient number of characters to distinguish it from other control words. The following precedence order is observed: ERROR, EXIT, SKIP, SPACE, STACK, SET, TYPEOUT, TIME, other control words, other keywords. Keywords cannot be abbreviated.

c. An error from a CMS command does not cause an exit from the level of EXEC.

d. When EXEC is entered, the assumed state of the controls are &ERROR, &CONTINUE, &TIME OFF, and &TYPEOUT ON TIME PACK.

e. If an EXEC line specifies an invalid CMS command, an error code of E(-0003) is returned. The EXEC command is not terminated.

Errors from EXEC Control Words

E(00001) File does not exist.

E(00002) &SKIP or &GOTO error.

E(00003) File has wrong record size.

E(00004) Keyword or argument error.

E(00005) Exceeded maximum depth of loop nesting.

E(00006) Waitrd or Rdbuf error.

E(00008) Illegal form of condition.

E(00010) Error in &GLOBAL or &INDEX usage.

E(00011) Error occurred in attempt to execute &ERROR's action.

PROFILE EXEC

The PROFILE EXEC feature allows a user to set up his own operating environment within CMS. When CMS is IPL'ed and the first CMS command is entered, an automatic search is made for a file with a filename and filetype of PROFILE EXEC. If such a file exists, it is automatically executed before the first CMS command entered is executed--thereby saving the user from entering any repetitious commands he may be entering each time he uses CMS.

PROFILE EXEC is a standard EXEC file as described in the preceding sections and, as such, may contain any valid EXEC-type statements. Its only difference is in its name, which has a special meaning that causes this automatic execution.

Examples

a. A PL/I user would have to use the GLOBAL T PLILIB statement each time he was on the system so that the PL/I library would be used rather than the FORTRAN libraries. This PROFILE might be created as follows:

```
edit profile exec
NEW FILE.
INPUT:
global t plilib

EDIT:
file
```

b. A user who wanted to redefine his LINEND and BLIP characters each time might set up the following PROFILE EXEC:

```
edit profile exec
NEW FILE.
INPUT:
&typeout off
linend !
blip *

EDIT:
file
```

Note. This automatic execution may be avoided by issuing LOGIN NOPROF as the very first CMS command (see LOGIN under "Control Commands").

GENMOD

Purpose:

The GENMOD command is used to generate non-relocatable core-image files.

Format:

GENMOD	entry1 <entry2> (option1...optionN)

entry1 specifies an entry point or a control section name indicating the starting core location from which the core-image copy is to be generated. It is also the filename assigned to the newly generated file.

entry2 specifies an entry point or a control section name indicating the ending core location from which the core-image copy is to be generated.

Options:

NOMAP specifies that a load map is not to be contained in the core-image file.

P2 specifies that the MODULE file is to have a mode of P2.

Usage:

The GENMOD command causes a file to be created which is a copy of the contents of a specified portion of core. The LOAD, USE, or REUSE commands will have been issued prior to the GENMOD command to load into core the file or files of which a non-relocatable core-image copy is to be created. The newly created file is placed on the user's permanent disk and is assigned a filename of the first operand specified in the GENMOD command, a filetype of MODULE, and a filemode of P1 unless the option P2 was specified, in which case the filemode is P2.

This file is in core-image form and is a copy of the contents of core from the first entry point to the second entry point specified in the GENMOD command. If only one entry point is specified, the core-image file consists of a copy of the contents of core from the first entry point specified to the next available load location. (The next available load location is indicated by a pointer which is updated after each LOAD, LOADMOD, USE, or REUSE command is issued.)

Before the core-image file is written, undefined symbols are defined to location zero and common is initialized. The undefined symbols are not retained in the MODULE file as being unresolved; therefore, once the MODULE is generated,

those references can not be resolved.

Notes:

a. Any files existing on the permanent disk with a filetype of MODULE and the same filename as that specified in the GENMOD command will be erased before the new file is created.

b. To load into core any files which have been created by the GENMOD command, the LOADMOD command should be used. If the MODULE file is to be loaded into core and executed and that MODULE file was generated with the (NOMAP) option, LOADMOD can not be used; instead, the MODULE's filename must be issued as a command.

c. The MODULE file contains a load map of the core-image unless (NOMAP) is specified.

d. A MODULE file without a load map requires less disk space.

Responses:

None.

Examples:

a. GENMOD FIRST
Assuming that a file which containing an entry point FIRST has been loaded into core prior to issuing this command, the above example causes a core-image file to be created on the user's permanent disk. This file consists of the contents of core from entry point FIRST to the next available load location and a load map. It has an identifier of FIRST MODULE P1.

b. GENMOD ABC DEF (NOMAP)
This example creates a file on the user's permanent disk with a filename of ABC, a filetype of MODULE, and a filemode of P1. The file is a copy of the contents of core from entry point ABC to entry point DEF. A load map is not included in the MODULE file.

Error Messages:

E(00001) NO "entry1" MODULE
This message indicates that the entry point(s) specified cannot be located in core. Check to see that these entry points exist and reissue the command.

E(00002) DISK ERROR
An address has been generated outside the bounds of core storage assigned to the user. Reissue the command.

E(00003) DISK ERROR
A disk malfunction has occurred. Reissue the GENMOD command. If the message persists, there is probably a disk hardware problem.

E(00004) DISK ERROR
An attempt to close the file after writing it out has not been successful. Issue FINIS and then reissue the GENMOD command.

E(00005) DISK ERROR
An illegal second character has been encountered for filemode. Reissue the GENMOD command.

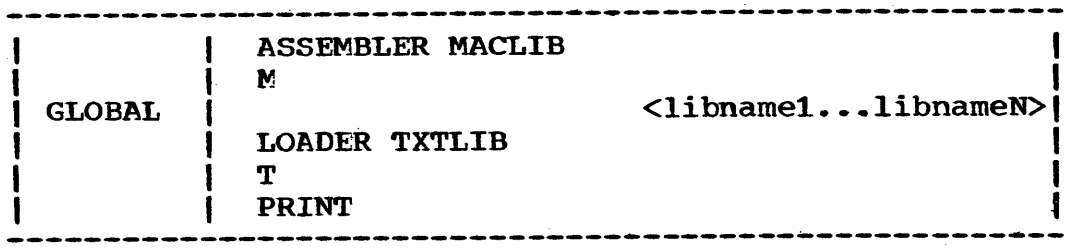
E(00006) DISK ERROR
The system has attempted to close the file prior to opening it. Reissue the GENMOD command.

E(00013) DISK ERROR
The user's disk is full, and the core-image file cannot be created. Erase one or more of the unneeded files and reissue the GENMOD command.

Purpose:

GLOBAL specifies either macro definition libraries to be searched during the ASSEMBLE command, or text libraries to be searched when loading files containing relocatable object code.

Format:



ASSEMBLER MACLIB
M — specifies the library files that are to be searched for macro definitions during subsequent assemblies.

LOADER TXTLIB
T — specifies the library files that are to be searched for missing subroutines during subsequent LOAD, USE, or REUSE operations.

libname1...libnameN — specifies the library files whose filetype is either MACLIB or TXTLIB.

PRINT — specifies that a list of libraries currently in use is to be typed at the terminal.

Usage:

GLOBAL has three forms--the ASSEMBLER form, the LOADER form, and the PRINT form.

ASSEMBLER Form. The ASSEMBLER form of the GLOBAL command allows the user to specify the macro libraries that are to be used during the execution of the ASSEMBLE command. One to five macro libraries may be specified. These macro libraries are searched for macro definitions in the order in which they are named. If the CMS macro library SYSLIB MACLIB and the OS macro library OSMACRO MACLIB are to be searched along with the user's macro libraries, SYSLIB and OSMACRO must be specified as two of the five libraries.

Each macro library specified must have a filetype of MACLIB. For a description of MACLIB files and how to generate them, see the MACLIB command under "Libraries".

If no previous GLOBAL command has been issued, the ASSEMBLE command searches the two macro libraries SYSLIB MACLIB and OSMACRO MACLIB in that order. Both files reside on the CMS system disk; SYSLIB MACLIB contains all of the CMS macros, and OSMACRO MACLIB contains the OS macros. If the user has created a file named SYSLIB MACLIB or OSMACRO MACLIB that resides on a disk which precedes the system disk in the standard order of search, it is used in place of the system file. To terminate the searching of all macro libraries, including SYSLIB MACLIB and OSMACRO MACLIB, the GLOBAL ASSEMBLER command can be issued with no libnames specified.

Once the ASSEMBLER form of the GLOBAL command has been issued, the specified macro libraries are searched for macro definitions during each assembly until either a GLOBAL ASSEMBLER command is reissued, the CMS nucleus is reinitialized, or the user logs out from CP.

For a further discussion of macro libraries, refer to Library Usage under "Operating Considerations".

PRINT Form. The PRINT form of the GLOBAL command types at the terminal a list of the current macro and text libraries that are being searched for that user.

LOADER Form. The LOADER form of the GLOBAL command allows the user to specify text libraries to be searched for missing subroutines and filenames whenever the LOAD, USE, or REUSE commands are issued. One to eight text libraries may be specified. These text libraries are searched in the order in which they are named. If the system text libraries SYSLIB TXTLIB and CMSLIB TXTLIB are to be searched along with the user's text libraries, SYSLIB and CMSLIB must be specified as two of the eight libraries.

Each text library specified must have a filetype of TXTLIB. For a description of TXTLIB files and how to generate them, see the TXTLIB command under "Libraries".

If no GLOBAL has been issued, the LOAD, USE, and REUSE commands search the text library SYSLIB TXTLIB. This file resides on the CMS system disk; SYSLIB TXTLIB contains the Fortran library. If the user has created a file with the identifier SYSLIB TXTLIB that resides on a disk that precedes the system disk in the standard order of search, it is used in place of the system file.

If the GLOBAL LOADER command has been issued and the user wishes to eliminate the searching of the previously specified text libraries, GLOBAL LOADER TXTLIB can be issued specifying no libnames. This terminates all library searching for missing subroutines when files are loaded by LOAD, USE, or REUSE.

Once the LOADER form of the GLOBAL command has been issued,

the specified TXTLIB files are automatically searched for missing subroutines or filenames not found during each LOAD, USE, or REUSE until either a GLOBAL LOADER command is reissued, the option LIBE or SLIBE is specified with LOAD which overrides the GLOBAL LOADER command for the duration of that LOAD and any USE or REUSE commands which follow that LOAD, the CMS nucleus is reinitialized, or the user logs out of CP.

For further discussion on text libraries, refer to Library Usage under "Operating Considerations".

Notes:

a. If the GLOBAL ASSEMBLER command is issued, one to five macro libraries may be specified and each must have a filetype of MACLIB.

b. If the GLOBAL LOADER command is issued, one to eight text libraries may be specified and each must have a filetype of TXTLIB.

c. GLOBAL will verify the existence of the libraries. If a library does not exist, an error message is generated.

d. ASSEMBLER MACLIB and LOADER TXTLIB may be abbreviated by M and T, respectively.

Responses:

THE CURRENT MACRO LIBRARIES (MACLIB) ARE:
xxxxxxxx xxxxxxxx
THE CURRENT TEXT LIBRARIES (TXTLIB) ARE:
yyyyyyyy yyyyyyyy
This is typed in response to the GLOBAL PRINT command where xxxxxxxx and yyyyyyyy are the names of the libraries.

Examples:

a. GLOBAL ASSEMBLER MACLIB NEWLIB MYMAC
The libraries NEWLIB MACLIB and MYMAC MACLIB are searched for macro definitions during the ASSEMBLE command. The order of search for macro definitions is NEWLIB MACLIB, then MYMAC MACLIB. The CMS macro library SYSLIB MACLIB and the OS macro library OSMACRO MACLIB are not searched.

b. GLOBAL ASSEMBLER MACLIB
This example cancels the effect of any previously issued ASSEMBLER form of the GLOBAL command and causes no libraries to be searched for macro definitions during execution of the ASSEMBLE command.

c. GLOBAL LOADER TXTLIB SCOOP OPS SYSLIB
The libraries SCOOP TXTLIB, OPS TXTLIB are searched for missing subroutines during the LOAD, USE, and REUSE commands. The order of search for missing subroutines is

SCOOP TXTLIB, OPS TXTLIB, and SYSLIB TXTLIB.

d. GLOBAL LOADER TXTLIB
This example cancels the effect of any previously issued GLOBAL LOADER command and causes no libraries to be searched for missing subroutines or undefined filenames by subsequent LOAD, USE, or REUSE commands.

Error Messages:

E(00001)
An invalid form of the GLOBAL command has been issued. Reissue the command in its correct format.

E(00002) TOO MANY TXTLIBS (MAX=8) OR MACLIBS (MAX=5) SPECIFIED
Reissue the GLOBAL command reducing the number of libraries specified.

E(00003) "libname" LIBRARY DOES NOT EXIST
Existence of "libname" MACLIB or "libname" TXTLIB has not been verified; "libname" has been omitted from the active list of libraries.

Purpose:

LOAD reads from disk one or more TEXT files containing relocatable object code and loads them into core, establishing the proper linkages between the files. If the specified TEXT files are not found, the appropriate TXTLIB files are searched. Corrections or additions can be made at load time and the user can specify libraries to be searched for missing subroutines. The user can also specify that execution should begin upon successful completion of loading.

Format:

```
|LOAD| fname1...fnameN <(opt1...optN)<libname1...libnameN>>|
```

fname1...fnameN	specify the names of TEXT files to be loaded into core.
opt1...optN	specify the options to be in effect during loading.
libname1...libnameN	specify the names of up to 8 TXTLIB files to be searched for missing subroutines during loading.

Options:

CLEAR	zero the load area before loading
NOCLEAR	do not zero the load area before loading
SLCxxxxx	begin loading the program at hexadecimal location xxxxx
SLC12000	begin loading the program at hexadecimal location 12000
NOMAP	do not create the file LOAD MAP
MAP	create the file LOAD MAP
TYPE	type the LOAD MAP file online
NOTYPE	do not type the LOAD MAP file online
SINV	suppress the printing of invalid card images in the LOAD MAP file
PINV	print invalid card images in the LOAD MAP file
SREP	suppress the print of Replace card images in the LOAD MAP file
PREP	print Replace card images in the LOAD MAP file

LIBE	search only the specified TXTLIB files for missing subroutines
SLIBE	do not search any TXTLIB files for unresolved references
SAUTO	suppress automatic searching for TEXT files
AUTO	search the P, T, and S disks for TEXT files to resolve undefined references (AUTO is the default and can not be specified as an option)
XEQ	execute the loaded files
NOXEQ	do not execute the loaded files

Usage:

The TEXT files specified in the LOAD command must consist of relocatable object code, such as that produced by the ASSEMBLE, FORTRAN, or PLI commands. When LOAD, USE, or REUSE is issued, the standard order of search is used to locate the specified TEXT files. Then, if any unresolved references exist, the search is used again to locate TEXT files corresponding to the unresolved names. If there are still unresolved references, the appropriate TXTLIB files are searched. To suppress the automatic searching of TEXT files for undefined names, specify the SAUTO option. To suppress the library search for unresolved references, specify the SLIBE option.

LOAD assumes the NOCLEAR option as a default, therefore the files that are being loaded are not placed in zeroed core. To zero core before the files are loaded, the option CLEAR must be specified.

LOAD automatically begins loading the specified files into core at hexadecimal location 12000. This load point may be changed by specifying the option SLCxxxxx, where xxxxx is the hexadecimal location at which loading is to begin. The SLCxxxxx option may not appear as the first option in a string unless it is preceded by one or more blanks.

Unless the NOMAP option is specified, a load map is created on the permanent disk each time the LOAD command is issued. A load map is a file that contains the location of control sections and entry points of files loaded into core. It may also contain certain messages and card images of any invalid cards or Replace cards that exist in the loaded files. This load map is normally created as a file with the identifier LOAD MAP P5. Only one such file may exist on the permanent disk. Each time LOAD is issued, a new LOAD MAP replaces any previous LOAD MAP file. To prevent a LOAD MAP file from being created, the option NOMAP must be specified.

Since LOAD assumes a default of NOTYPE, the LOAD MAP file is not automatically typed. To type this, the option TYPE is specified. The LOAD MAP file may also be printed by PRINTF

or OFFLINE PRINT.

If invalid card images exist in the file or files that are being loaded, they are listed with the message INVALID CARD in the LOAD MAP file. To suppress this listing in the LOAD MAP the SINV option must be specified.

```
load w (type)
 W        AT 12000
 W        AT 12000
 IHCFCOMH AT 12148
 SAVAREA  AT 1314C
 IBCOM    AT 12148
 FDIOCS   AT 12204
 IHCFCVTH AT 13198
 ADCON    AT 13198
 FCVEO    AT 13BBA
 FCVLO    AT 13412
 FCVIO    AT 13720
 FCVCO    AT 13DB4
 FCVAO    AT 1338A
 FCVZO    AT 132E4
 IHCFIOSH AT 14188
 FIOCS    AT 14188
 IHCUATBL AT 14D70
 THE FOLLOWING NAMES ARE UNDEFINED:
      SUB1
E(00004); T=0.05/0.20   14.21.33
```

Typeout of the LOAD MAP file during the LOAD command.

If Replace (REP) card images exist in the files being loaded, they are included in the LOAD MAP. To suppress this listing of REP card images the SREP option must be specified. For an explanation of REP card images see Changing Object Programs under "Operating Considerations".

Unless the GLOBAL LOADER command has been issued, LOAD searches only the system text library SYSLIB TXTLIB for subroutines that are missing from the files being loaded. Using GLOBAL, the user can specify from one to eight text libraries to be searched. See the description of GLOBAL for specific details.

If a file exists on a disk preceding the system disk in the standard order of search with the identifier SYSLIB TXTLIB, it is used in place of the system text library.

To prevent LOAD from searching the system text library and the files specified by GLOBAL, the LIBE or SLIBE option can be specified. LIBE terminates the searching of all text libraries except those specified with the LOAD command. If SYSLIB TXTLIB is to be searched along with the specified libnames, SYSLIB must be included as one of the libnames.

The order of search of the specified libnames is the order in which they are named. The maximum number of libraries that can be searched is eight. If LIBE is issued and no TXTLIB files are specified, none are searched for missing subroutines. If SLIBE is specified with LOAD, no TXTLIB files are searched. For a discussion of text library usage, refer to Library Usage under "Operating Considerations".

LOAD assumes NOXEQ as a default option, therefore, LOAD does not normally begin execution of the loaded files. To begin execution immediately upon successful completion of loading, XEQ can be specified. LOAD then transfers control to the default entry point in the program. The default entry point is either the address specified in the operand field of the first END card containing a non-blank operand field or the beginning of the first file loaded if all END card images in the TEXT files contain blank operand fields. In the case of TEXT files that are created by FORTRAN, control is passed to the first main program loaded. If XEQ is not specified, START command must be issued to begin program execution.

Duplicate CSECT's (Control Sections) are bypassed by the loader. Only the first CSECT encountered is physically loaded. The duplicates are not loaded and a warning message is included in the LOAD MAP.

LOAD allows the user to include the following card images in the TEXT files along with the relocatable object code: the Set Location Counter (SLC) card image, the Replace (REP) card image, the Include Control Section (ICS) card image, and the Entry and Library statements. SLC specifies the hexadecimal location at which files are to be loaded. REP specifies corrections to be made to the relocatable object code. ICS specifies additions to be made to the TEXT file. The Entry statement specifies entry points and the Library statement specifies the never-call function. For a description of these card images and their use and placement in a TEXT file, refer to Changing Object Programs under "Operating Considerations".

Notes:

a. To terminate the searching of all text libraries, including SYSLIB, specify LIBE with no libnames, or specify SLIBE.

b. If TEXT files do not exist for the names specified with LOAD, either the specified or default TXTLIB files are searched for the missing TEXT file.

c. If unresolved names occur, the standard order of search is used to locate the TEXT files; if the names are still unresolved, the appropriate TXTLIB files are searched.

Responses:

EXECUTION BEGINS...
XEQ has been specified with LOAD and the loaded program has begun execution. Any further responses are from the program.

INVALID CARD - xxx...xxx
PINV has been specified with LOAD and an invalid card has been found. The message and the contents of the invalid card (xxx...xxx) are listed in the file LOAD MAP. The invalid card is ignored and loading continues.

CONTROL CARD - ...
A loader or library-search control card has been encountered, (that is, ENTRY or LIBRARY).

If TYPE is specified with LOAD, the LOAD MAP file is typed.

Examples:

a. LOAD MAIN SQ3 CALCU
The files MAIN TEXT, SQ3 TEXT, and CALCU TEXT are loaded into core and the linkages resolved. If any subroutines are missing, the loader searches for the corresponding TEXT file. If any references are still unresolved, and neither the SLIBE option nor the GLOBAL LOADER command has been previously issued, the file SYSLIB TXTLIB is searched. If the GLOBAL LOADER command has been previously issued, the libnames specified in that command are searched. The following default options are set: NOCLEAR, NOTYPE, SLC12000, PINV, PREP, MAP, AUTO, and NOXEQ.

b. LOAD MPS67 HOOK (XEQ TYPE CLEAR)
The files MPS67 TEXT and HOOK TEXT are loaded into core and the linkages resolved. Core is zeroed before loading takes place. The LOAD MAP file is typed. Upon successfully loading MPS67 and HOOK, execution begins.

c. LOAD MASS WHATZIT (LIBE) SSP MYLIB SYSLIB
The files MASS TEXT and WHATZIT TEXT are loaded into core and the linkages resolved. If any subroutines are missing, the following libnames will be searched in the order in which they are specified: SSP TXTLIB, MYLIB TXTLIB, and SYSLIB TXTLIB. The remaining options are set to default.

d. LOAD MASSPEC (LIBE)
The file MASSPEC TEXT is loaded into core and the linkages resolved. Since LIBE has been specified without any libnames, no text libraries are searched for missing subroutines. If there are any missing subroutines, an error code is returned.

Error Messages:

E(00001) DEFINED MORE THAN ONCE-xxxxxxxx
The name xxxxxxxx has been defined more than once. Check the files that have been loaded for duplicate entry point names or duplicate control section names. Loading has been completed. Duplicate names are not loaded.

E(00002) OVERLAY ERROR
The files being loaded have run out of core. Specify fewer files or reduce the size of the files. Loading has been completed.

E(00003) REFERENCE TABLE OVERFLOW
There are too many entries for entry points or control section names in the reference table built during loading. Loading has been completed. Reduce the number of entry points and/or control sections in the files.

E(00004) THE FOLLOWING NAMES ARE UNDEFINED-xxxxxxxx
The names xxxxxxxx are referenced in a file and are never defined. If the names are defined in a file with a different name, issue USE for that file. Loading has been completed.

Dynamic Loading

During program execution, another relocatable object deck may be brought into core, external references resolved, and control given to it--that is, dynamic loading. The desired routine must exist on the user's files with a filetype of TEXT, or exist in one of the designated libraries with filetype TXTLIB. The routine may cause other TEXT or library routines to be loaded into core.

The following CMS/OS macros support dynamic loading:

LOAD (SVC 8) causes the object deck containing the specified entry point to be brought into core and the entry point address to be returned in register zero (0).

LINK (SVC 6) calls in and transfers control to the specified entry point.

XCTL (SVC 7) deletes the calling routine, then brings the specified routine in and gives control to it.

RETURN (SVC 3) deletes the called routine and gives control back to the caller.

LOADMOD

Purpose:

LOADMOD loads into core any single file in nonrelocatable core image form.

Format:

```
-------------------------------------
|  LOADMOD  |   filename <filemode>|
-------------------------------------
```

filename is the name of the file to be loaded into core. The filetype must be MODULE.

filemode is the mode of the MODULE file to be loaded.

Usage:

LOADMOD is used to load a file which has been created by the GENMOD command. The filename of the file to be loaded is specified as the operand of the LOADMOD command, and its filetype must be MODULE. If the MODULE file was generated without a load map and the MODULE file is to be read into core and executed, LOADMOD can not be issued; instead, the MODULE's filename must be issued as if it were a command.

When LOADMOD is issued without specifying a filemode, the standard order of search is used to locate a file with the specified filename and a filetype of MODULE. If a filemode is given, only that disk is searched for the MODULE file. If such a file is found, it is assumed to be in non-relocatable core-image form, and is loaded into core.

Responses:

None.

Example:

LOADMOD FILE1
The file FILE1 MODULE is loaded into core. If no such file exists, an error message is returned, and the loading process does not take place.

Error Messages:

E(00001) FILE DOES NOT EXIST
DISK ERROR
Either of the above messages indicates that a file with the specified filename and a filetype of MODULE cannot be located. Check to see that such a file exists and that the filename specified in the LOADMOD command is identical to the filename of the file to be loaded.

E(00002) DISK ERROR
An address has been generated outside the bounds of core storage assigned to the user. Reissue the command.

E(00003) DISK ERROR
A disk malfunction has occurred. Reissue the LOADMOD command. If the message persists, a disk hardware problem has probably been encountered.

E(00004) FILE DOES NOT EXIST
DISK ERROR
Either of the above messages indicates that the filemode of the specified file is invalid. Change the filemode to a valid one and reissue the command.

E(00006) DISK ERROR
Core space assigned to the user is not large enough for loading the specified file or the system has attempted to close the file prior to opening it. Reissue the LOADMOD command.

E(00007) DISK ERROR
The specified file cannot be read from disk. Reissue the LOADMOD command. If this message persists, the file should be recreated using the GENMOD command.

E(00009) DISK ERROR
The specified file is open for writing and cannot be read. Reissue the LOADMOD command.

Purpose:

REUSE reads from disk one or more TEXT files containing relocatable object code and loads them into core, establishing linkages with previously loaded files, and changing the default entry point of these files to that of the first file specified in the REUSE command. If the TEXT files do not exist, the appropriate TXTLIB files are searched.

Format:

```
-------------------------------------------------------------
|REUSE |fname1...fnameN (opt1...optN)<libname1...libnameN>|
-------------------------------------------------------------
```

fname1...fnameN	specify the names of TEXT files to be loaded into core.
opt1...optN	specify the options to be in effect during loading.
libname1...libnameN	specify the names of up to 8 TXTLIB files to be searched for missing routines during loading.

Options:

The options that may be specified with REUSE are the same as those with LOAD.

Usage:

REUSE does not overlay any file that has been previously loaded by a LOAD, USE, or REUSE command. It loads the specified files into higher core from the point at which the previous LOAD, USE, or REUSE command terminated loading. REUSE performs the same function as USE except that REUSE changes the default entry point to that of the first file specified with the REUSE command.

The specified files must have filetypes of TEXT and contain relocatable object code.

If options have been specified with the previous LOAD, USE, or REUSE command, these options remain set unless respecified. The LOAD MAP file is automatically updated to reflect the files loaded by REUSE. Refer to LOAD for a description of the LOAD options, the LOAD MAP file, and how LOAD operates.

Responses:

INVALID CARD - xxx...xxx
PINV has been specified and an invalid card has been found. The message and the contents of the invalid card (xxx...xxx) are listed in the file LOAD MAP. The invalid card is ignored and loading continues.

CONTROL CARD
A loader or library-search control card has been encountered. Normal loading resumes.

If TYPE has been specified with REUSE or has not been reset from the previous LOAD, USE, or REUSE command, the updated portion of the LOAD MAP file is typed prior to the completion of the REUSE command.

Example:

REUSE READIT GAMMA
The TEXT files READIT and GAMMA are loaded into core, linkages resolved with the files previously loaded, and the default entry point is changed to the first entry point in READIT.

Error Messages:

E(00001) DEFINED MORE THAN ONCE - xxxxxxxx
The name xxxxxxxx has been defined more than once. Check the files that have been loaded for duplicate entry point names or duplicate control section names. Loading has been completed. Duplicate names are not loaded.

E(00002) OVERLAY ERROR
The files being loaded have run out of core. Specify fewer files or reduce the size of the files. Loading has been completed.

E(00003) REFERENCE TABLE OVERLAY
There are too many entries for entry points or control section names in the reference table built during loading. Loading has been completed. Reduce the number of entry points or control sections in the files.

E(00004) THE FOLLOWING NAMES ARE UNDEFINED - xxxxxxxx
The names xxxxxxxx are referenced in a file and are never defined. If the names are defined in another file, issue the USE command for that file. Loading has been completed.

E(00005) NAME IS UNDEFINED - xxxxxxxx
The name xxxxxxxx specified as an entry point does not exist. Loading has been completed. Check the name and see if an entry point or a control section exists by that name in the loaded files.

START

Purpose:

START begins execution of programs previously loaded and passes the address of a string of user arguments to that program.

Format:

```
-------------------------------------------
|  START  |<entry <argument1...argumentN>>|
|         |   *                           |
-------------------------------------------
```

entry	specifies the name of a control section or entry point to which control is passed at execution time.
*	specifies that control is to be passed to the default entry point.
argument1...argumentN	specify information to be passed to the started program.

Usage:

START begins execution at one of two entry points. If the entry operand is specified, execution begins at that point in the program. If * or nothing is specified execution begins at the default entry point. The default entry point is either the address specified in the operand field of the first END card containing a non-blank operand field or the beginning of the first file loaded if all END cards in the TEXT files contain blank operand fields. The default entry point can be changed by issuing the REUSE command to continue loading additional files.

Any undefined names or references specified in the files loaded into core are defined to location zero. Thus, if there is a call or branch to a subroutine from a main program and the subroutine has never been loaded, the call or branch transfers control to location zero at execution time.

If arguments are specified with START, they are passed to the program via general-purpose register 1. The entry operand and any arguments are set up as a string of words, one argument per double word, and the address of the parameter is placed in general-purpose register 1. The arguments are accessed with displacements of 8, 16, 24, etc., from the address contained in register 1 when execution of the specified program begins.

Notes:

a. Entry must be a control section name or an entry point name. It may be a filename only if the filename is identical to a control section name or an entry point name.

b. If user arguments are specified, entry or * must be specified; otherwise, the first argument is taken as the entry point.

Responses:

EXECUTION BEGINS...
The program previously loaded into core has begun execution. Further responses are from the executing program.

Examples:

a. START INITIL
The program already loaded into core begins execution at entry point INITIL.

b. START MEGOP 13 ALL 109439
The program already loaded into core begins execution at entry point MEGOP. The three arguments may be accessed in the program by displacements of 8, 16, and 24 from the address in general-purpose register 1.

Error Messages:

E(00004)
The contents at STADDR in NUCON are either 0 or a location that does not contain executable code. Issue LOADMOD and START again.

E(00005) NAME IS UNDEFINED - xxxxxxxx
The name xxxxxxxx specified as the point at which execution is to begin does not exist as an entry point name or a control section name. Execution has not begun. Check the name xxxxxxxx and make sure it is a valid entry point or control section name.

USE

Purpose:

USE reads one or more TEXT files containing relocatable object code from disk and loads them into core, establishing linkages with previously loaded files. If the TEXT files do not exist, the appropriate TXTLIB files are searched.

Format:

```
-------------------------------------------------------------
|USE|fname1...fnameN <(opt1...optN)<libname1...libnameN>> |
-------------------------------------------------------------
```

fname1...fnameN specify the names of TEXT files to be loaded into core.

opt1...optN specify the options to be in effect during loading.

libname1...libnameN specify the names of up to eight TXTLIB files to be searched for missing routines during loading.

Options:

The options that may be specified with USE are the same as those that may be specified for LOAD.

Usage:

USE does not overlay any file previously loaded by a LOAD, USE, or REUSE command. It loads the specified file(s) into higher core from the point at which the previous LOAD, USE, or REUSE command terminated loading. The files specified with USE have filetypes of TEXT and contain relocatable object code.

USE should be preceded by LOAD; it is issued to resolve undefined names when LOAD gives the following error message:

```
E(00004) - THE FOLLOWING NAMES ARE UNDEFINED: xxxxxxxx.
```

USE may be issued repeatedly to resolve linkages and to continue loading more TEXT files. It does not change the default entry point established in a previous LOAD command.

If options are specified with the previous LOAD, USE, or REUSE command, the options remain set unless respecified when USE is issued. The LOAD MAP file is automatically updated to reflect the files loaded by USE. Refer to LOAD for a description of the LOAD options, the LOAD MAP file, and the operation of LOAD.

Responses:

INVALID CARD - xxx...xxx
PINV has been specified with the previous LOAD command and an invalid card has been found. The message and the contents of the invalid card (xxx...xxx) are listed in the file LOAD MAP. The invalid card is ignored and loading continues.

If TYPE has been specified with USE or has not been reset from the previous LOAD, USE, or REUSE command, the updated portion of the LOAD MAP file is typed prior to the completion of the USE command.

Example:

USE MYTEXT1 CALCA WRITE6
MYTEXT1 TEXT, CALCA TEXT, and WRITE6 TEXT are loaded into core. Linkages are resolved between these three files and the file previously loaded into core.

Error Messages:

E(00001) DEFINED MORE THAN ONCE - xxxxxxxx
The name xxxxxxxx has been defined more than once. Check the files that have been loaded for duplicate entry point names or duplicate control section names. Loading has been completed. The duplicates are loaded.

E(00002) OVERLAY ERROR
The files being loaded have run out of core. Specify fewer files or reduce the size of the files. Loading has been completed.

E(00003) REFERENCE TABLE OVERFLOW
There are too many entries for entry points or control section names in the reference table built during loading. Loading has been completed. Reduce the number of entry points or control sections in the files.

E(00004) THE FOLLOWING NAMES ARE UNDEFINED - xxxxxxxx
The names xxxxxxxx are referenced in a file and are never defined. If the names are defined in another file, issue the USE command for that file. Loading has been completed.

$

Purpose:

$ loads and starts the specified file, provided its filetype is EXEC, MODULE, or TEXT.

Format:

```
-------------------------------------
|  $  | filename  <arg1...argN>      |
-------------------------------------
```

filename — is the name of a file whose filetype is EXEC, MODULE, or TEXT.

arg1...argN — are one or more user arguments.

Usage:

The $ command is used to load and start a program. The program exists as a file on one of the user's disks, and its filename must be specified as the first operand of the $ command. The standard order of search is used to locate a file with the specified filename and a filetype of EXEC, MODULE, or TEXT, in that order.

If an EXEC filetype is found for the filename, the file is assumed to contain one or more CMS commands, and EXEC is called to execute this file. If no EXEC filetype exists, but a filetype of MODULE is found, LOADMOD is called by $ to load the program into core and START is called to begin execution of the program. When only a TEXT filetype exists LOAD is called followed by START.

The user may specify as many arguments in the $ command as he wishes, provided they all fit on the same input line. The arguments are set up as a string of double words, one argument per double word. The address of this string is passed to the specified file. Each argument is left-justified, and any argument more than eight characters in length is truncated on the right. With an EXEC file, any arguments specified in the $ command replace the corresponding &n operands in the individual commands of the EXEC file (see EXEC under "Execution Control" for a full explanation of this operand-substitution technique).

With a file whose filetype is MODULE or TEXT, the arguments are placed in a string as described above. The address of the string may be obtained by adding 8 to the address contained in general-purpose register 1 at the time execution of the specified program begins. Additional arguments may be referenced by displacements of 16, 24, 32, etc., from the address thus obtained.

Note:

When a file with a filetype of MODULE or TEXT is used, there must be an entry point in the file that is identical to the filename specified in the $ command. After the file has been loaded, execution begins at this entry point. Such an entry point is created by the Fortran compiler using the filename specified in the FORTRAN command. With Assembler language files, the user should create as an entry point or assign as the name of a control section, the filename by which he wishes to reference the TEXT or MODULE version of that file.

Responses:

For files with filetype EXEC, each command in the EXEC file is typed at the user's terminal prior to its execution unless the &TYPE option is used to suppress the printout.

EXECUTION BEGINS...
This message is typed when a file of filetype MODULE or TEXT has been loaded into core and is about to be started. Output appearing after this message is from the user's program or from a part of CMS called by that program.

DEFINED MORE THAN ONCE - xxxxxxxx
This message is generated by LOAD and indicates that duplicate entry points or control section names (xxxxxxxx) have been found in the TEXT file being loaded. $ is terminated with an error code of 3.

OVERLAY ERROR
There is not enough room in core to hold the TEXT file for which a LOAD has been issued. $ is terminated with an error code of 3.

REFERENCE TABLE OVERFLOW
There are too many entry points or control section names in the TEXT file being loaded. $ is terminated with an error code of 3.

THE FOLLOWING NAMES ARE UNDEFINED - xxxxxxxx
The specified names referenced in the TEXT file being loaded are never defined. $ is terminated with an error code of 3.

DISK ERROR
An error has occurred while reading or closing a file with filetype MODULE. This message is generated by LOADMOD, and terminates with an error code of 3.

Examples:

a. $ MYFILE
The standard order of search is used to locate a file with a filename of MYFILE and a filetype of EXEC, MODULE, or TEXT. If a file exists with filename MYFILE and filetype EXEC, the

commands contained in it are executed. Each command is typed at the user's terminal before it is executed. If filetypes MODULE and TEXT or of MODULE only exist for filename MYFILE, the file with filetype MODULE is loaded into core by LOADMOD and started by START at entry point MYFILE. If only a TEXT filetype exists for filename MYFILE, LOAD is issued to bring MYFILE into core, and START is issued using entry point MYFILE.

b. $ OTHER SAME 1.436 5 A
If filetype EXEC is found for filename OTHER, execution of the EXEC file begins with the argument SAME replacing &1 wherever it appears in the EXEC file, 1.436 replacing &2, 5 replacing &3, and A replacing &4. If no EXEC filetype exists for filename OTHER but a filetype of MODULE or TEXT is found the file is loaded into core and started at entry point OTHER. The four user arguments can be accessed by displacements of 8, 16, 24, and 32, respectively, from the address contained in general-purpose register 1 at the time program OTHER is started.

DEBUGGING FACILITIES

A debugging tool is provided with CMS in the form of the DEBUG command. This command allows the user, while at his terminal, to examine and change the contents of core locations, program status words, general-purpose registers, the channel status word, and the channel address word; to dump portions of core on the offline printer; and to stop and restart programs at any specified point or points. Methods for using these DEBUG facilities are described in this section.

In addition to DEBUG, there are two commands that allow the user to trace supervisor calls (SVC instructions) and therefore, the internal branches which are issued to the various CMS commands and functions. These two commands--SETOVER and SETERR--set certain overrides, or flags, which are checked whenever an SVC instruction is executed and a return is issued from an SVC-called program. Two types of overrides may be set: normal and error. Normal overrides are those which cause trace information to be recorded for SVC-called programs executed without encountering any error conditions. Error overrides are those which record information for SVC-called programs which return with an error code in general-purpose register 15. The SETOVER command sets both types of overrides. The SETERR command sets error overrides only.

To clear overrides which have been set by the SETOVER and/or SETERR commands, the CLROVER command may be issued. In addition to terminating the recording of trace information, CLROVER causes all information recorded up to that point to be printed on the offline printer.

If the user wishes to terminate the recording of trace information during the execution of one of his own programs or of a CMS command (that is, at a point when the CLROVER command cannot be issued) he may do so by hitting ATTN twice (pausing each time for the keyboard to unlock) and typing the letters KO followed by a carriage return. Processing continues as before, but no further information is recorded for SVC's executed. KO terminates overrides and causes recorded trace information to be printed on the offline printer.

If the user has set overrides by issuing either or both the SETOVER and SETERR commands and has failed to clear these overrides, they are cleared automatically and the recorded information is printed on the offline printer when the user logs out from the Control Program or when he issues a RESTART request in the Debug environment.

Purpose:

CLROVER clears overrides set by either or both the SETOVER and SETERR commands. It also causes all trace information recorded up to that point to be printed on the offline printer.

Format:

```
-------------
| CLROVER   |
-------------
```

Usage:

This command terminates the recording of trace information set by the SETOVER and/or SETERR commands and causes that information to be printed on the offline printer. If CLROVER is not issued, the user may clear all currently active overrides by issuing a KO command. Any overrides which have been set but not cleared at the time the user issues a RESTART request in the Debug environment or ends his terminal session by logging out from the Control Program are cleared automatically and the trace information is printed offline.

CLROVER cancels the effect of all SETOVER and SETERR commands issued since the last KO or CLROVER command was issued, or since the user's last CMS login if neither a KO nor a CLROVER has been issued during the terminal session. Once CLROVER is issued, no further trace information is recorded until the user issues another SETOVER or SETERR command.

A sample of the format in which trace information is printed offline is given in Figure 15. A fixed amount of trace information is printed for all error overrides; the amount for normal overrides varies depending on the options specified with the SETOVER command. An explanation of all possible fields which can appear in the printout is given under "Output" in this section.

Notes:

a. If a CLROVER command is issued when no overrides are currently active, it has no effect other than printing the following line on the offline printer:

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****

b. Any operands given with CLROVER are ignored.

Responses:

None.

Output:

An explanation of each field appearing in the printout of trace information is given below.

SETTING ERROR-OVERRIDE TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...
This message appears whenever SETERR is issued.

SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...
This message appears whenever SETOVER is issued.

*****ERROR-OVERRIDE,
This message identifies the first line for each SVC-called program which returns with an error code in general-purpose register 15.

NORMAL-OVERRIDE,
This message identifies the first line for each SVC-called program which issues a normal return.

CALLER = xxxxxxxx
This information appears in the first line for each SVC-called program. It indicates the hexadecimal core location (xxxxxxxx) of the SVC instruction whose execution caused that program to be called.

CALLEE = xxxxxxxx
This information appears in the first line for each SVC-called program. xxxxxxxx is the name of the called program (if a CMS SVC is issued) or the number of the SVC (if an OS SVC is issued).

SVC-OLD-PSW = xxxxxxxxxxxxxxxx
This information, given in the first line for each SVC-called program, gives the contents of the SVC old program status word. For an explanation of the SVC old program status words and its use, see the IBM manual, IBM System/360 Principles of Operation.

NRMRET = xxxxxxxx
This information, given in the first line for each SVC-called program, gives the hexadecimal core location (xxxxxxxx) to which the program returns under normal conditions (that is, when no error code is generated).

ERRET = xxxxxxxx
This information, appearing at the rignt margin of the first line for each SVC-called program, gives the hexadecimal core location (xxxxxxxx) to which the program returns if an error code is generated during its execution.

GPRS BEFORE = xxxxxxxx...xxxxxxxx
Two lines of information appear with this message. The first line consists of the contents of general-purpose registers 0-7; the second line gives the contents of registers 8-15 as they existed when control was passed to the SVC-called program.

FPRS BEFORE = xxxxxxxx...xxxxxxxx
This line of information gives the contents of the four floating-point registers as they existed at the time control was transferred to the SVC-called program.

GPRS AFTER = xxxxxxxx...xxxxxxxx
Two lines of information appear with this message. The first line gives the contents of general-purpose registers 0-7; the second line gives the contents of registers 8-15 as they existed when a return was issued by the SVC-called program.

FPRS AFTER = xxxxxxxx...xxxxxxxx
This line of information gives the contents of the four floating-point registers as they existed when a return was issued by the SVC-called program.

PARM.-LIST = xxxxxxxx...xxxxxxxx
This message is followed by one or two lines of the parameter list which existed at the time the SVC was executed. See SETOVER under "Debugging Facilities" for a discussion of parameter lists and their use.

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
This message appears whenever a CLROVER command is issued.

Example:

CLROVER
This clears all currently active overrides and cause any trace information recorded up to that point to be printed on the offline printer. See Figure 15 for a sample of the type of information printed.

Error Messages:

None.

```
SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=SETOVER  SVC-OLD-PSW=000400CA60009516   NEWRET=0000951A   ERRET=0000951A
    PARM.-LIST =  --SETO--     --VER --   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   FFFFFFFF  FFFFFFFF
                  FFFFFFFF     FFFFFFFF   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA4000057F4   NEWRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2C5E3D6     0000139C   00000000  00005816  00009B10  E2C5E3D6   00050A8   0003E8B8
                  00009386     00012000   00009508  00000001  400057E8  00011B0    40009540  000011B0
    FPRS BEFORE = 00000000     00000000   00000000  00000000  00000000  00000000   00000000  00000000
    GPRS AFTER  = E2C5E3D6     0000139C   0000139C  00000000  00000778  000C29DC   00000A20  00000048
                  00000004     00000100   C6C9D5C9  E2404040  000040A8  00000AA0   000026CE  00000006
    FPRS AFTER  = 00000000     00000000   00000000  00000000  00000000  00CC0000   00000000  00000000
    PARM.-LIST =  --FINI--     --S   --   --*   --  --   --  --*   --  --   --    5C40FFFF  FFFEFFFD
                  FFFF0000     00000000   00000000  00000000  00000000  --OVER--   --RIDE--  --MODU--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA6000955C   NEWRET=00009560   ERRET=0000954A
    PARM.-LIST =  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EFC4BF0   F461F04B  --14 2--
                  --0.50--     4BF1F315   17174040  --   --   40401517  17C54DD5   --UNBR--  5D5E40E3

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA400096DA   NEWRET=000096DE   ERRET=000096DE
    PARM.-LIST =  --WAIT--     --RD  --   01009774  E4000010  --TYPL--  --IN  --   01009386  D2000020
                  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EF04EF0   F461F04E  --14 2--

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA60009516   NEWRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3     00009B10   00000000  00000000  00009B10  E2E3C1E3   000050A8  0003E8B8
                  00009386     00012000   00009508  00000001  40009378  800094C0   60009580  00000001
    FPRS BEFORE = 00000000     00000000   00000000  00000000  00000000  00000000   0000000C  00000000
    GPRS AFTER  = E2E3C1E3     00009B10   00009B10  00000000  00000778  00002A84   00000A20  00000048
                  00000004     00000100   E2E3C1E3  C5404040  00002BD0  00000AA0   000026CE  00000001
    FPRS AFTER  = 00000000     00000000   00000000  00000000  0000000C  00CC0000   0000000C  00000000
    PARM.-LIST =  --STAT--     --E   --   --TEST--  --  --   --SYSI--  --N  --   FFFFFFFF  FFFFFFFF
                  FFFFFFFF     FFFFFFFF   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA40CC57F4   NEWRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3     0000139C   00000000  00005816  00009B10  E2E3C1E3   000050A8  0003E8B8
                  00009386     00012000   00009508  00000001  400057E8  00CC11B0   40009540  000011B0
    FPRS BEFORE = 00000000     00000000   00000000  00000000  0000000C  CCCC0000   0000000C  00000000
    GPRS AFTER  = E2E3C1E3     0000139C   0000139C  00000000  00000778  000029DC   00000A20  00000048
                  00000004     00000100   C6C9D5C9  E2404040  000040A8  CCCC0AA0   000026CE  00000006
    FPRS AFTER  = 00000000     00000000   00000000  00000000  00000000  00000000   00000000  00000000
    PARM.-LIST =  --FINI--     --S   --   --*   --  --   --  --*   --  --   --    5C40FFFF  FFFEFFFD
                  FFFF0000     00000000   00000000  00000000  00000000  --OVER--   --RIDE--  --MODU--

    NORMAL-OVERRIDE,   CALLER=0000972E   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA600C9730   NEWRET=00009734   ERRET=0000972E
    PARM.-LIST =  --TYPL--     --IN  --   01009631  C2000021  00000010  --STAT--   --E TE--  --ST S--
                  --YSIN--     --   --    --   --  --   --  --   --  --   --    --   --  --   --

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA40CC96DA   NEWRET=000096DE   ERRET=000096DE
    PARM.-LIST =  --WAIT--     --RD  --   01009774  E4000014  --TYPL--  --IN  --   01009386  D2000020
                  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EFC4EF0   F461F04B  --11 2--

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA60CC9516   NEWRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3     00009B10   00000000  00000000  00009B10  E2E3C1E3   000050A8  0003E8B8
                  00009386     00012000   00009508  00000001  4000937E  80CC94C0   60C09580  00000001
    FPRS BEFORE = 00000000     00000000   00000000  00000000  00000000  00000000   00000000  00000000
    GPRS AFTER  = E2E3C1E3     00009B10   00009B10  00000000  00000778  000C2A84   00000A20  00000048
                  00000004     00000100   E2E3C1E3  C5404040  00002BD0  C00CCAA0   000026CE  00000001
    FPRS AFTER  = C0000000     00000000   00000000  00000000  00000000  00000000   00000000  00000000
    PARM.-LIST =  --STAT--     --E   --   --SAME--  --LE  --  --PORT--  --RAN --   FFFFFFFF  FFFFFFFF
                  FFFFFFFF     FFFFFFFF   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NEWRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3     0000139C   00000000  00005816  00009B10  E2E3C1E3   000050A8  0003E8B8
                  00009386     00012000   00009508  00000001  400057E8  000011B0   40009540  000011E0
    FPRS BEFORE = 00000000     00000000   00000000  00000000  0000000C  0CCC0000   0000000C  00000000
    GPRS AFTER  = E2E3C1E3     0000139C   0000139C  00000000  00000778  000029DC   00000A20  00000048
                  C0000004     00000100   C6C9D5C9  E2404040  000040A8  00000AA0   000026CE  00000006
    FPRS AFTER  = 00000000     00000000   00000000  00000000  00000000  00CC0000   0000000C  00000000
    PARM.-LIST =  --FINI--     --S   --   --*   --  --   --  --*   --  --   --    5C40FFFF  FFFEFFFD
                  FFFF0000     00000000   00000000  00000000  00000000  --OVER--   --RIDE--  --MODU--

    NORMAL-OVERRIDE,   CALLER=0000972E   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA60CC9730   NEWRET=00009734   ERRET=0000972E
    PARM.-LIST =  --TYPL--     --IN  --   01009631  C2000021  C0000014  --STAT--   --E SA--  --MPLE--
                  -- FOR--     --TRAN--   --   --  --   --  --   --  --   --    --   --  --   --

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA40CC96DA   NEWRET=000096DE   ERRET=000096DE
    PARM.-LIST =  --WAIT--     --RD  --   01009774  E4000013  --TYPL--  --IN  --   01009386  D2000020
                  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EFC4BF0   F561F04B  --11 2--

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA60CC9516   NEWRET=0000951A   ERRET=0000951A
    PARM.-LIST =  --STAT--     --E   --   --FIGU--  --RE  --  --SCRI--  --PT  --   FFFFFFFF  000013E0
                  FFFFFFFF     FFFFFFFF   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA4CCC57F4   NEWRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3     0000139C   00000000  00005816  00009B10  E2E3C1E3   000050A8  0003E8B8
                  00009386     00012000   00009508  00000001  400057E8  000011B0   40009540  000011B0
    FPRS BEFORE = C0000000     00000000   00000000  00000000  00000000  00CC0000   0000000C  00000000
    GPRS AFTER  = E2E3C1E3     0000139C   0000139C  00000000  00000778  000029DC   00000A20  00000048
                  00000004     00000100   C6C9D5C9  E2404040  000040A8  00000AA0   000026CE  00000006
    FPRS AFTER  = 00000000     00000000   00000000  00000000  0000000C  0CCC0000   00C00000  00000000
    PARM.-LIST =  --FINI--     --S   --   --*   --  --   --  --*   --  --   --    5C40FFFF  FFFEFFFD
                  FFFF0000     00000000   00000000  00000000  00000000  --FIGU--   --RE  --  --SCRI--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA60CC955C   NEWRET=00009560   ERRET=0000954A
    PARM.-LIST =  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EFC4BF0   F361F04B  --08 2--
                  --0.50--     4BF4F815   17174040  --   --   40401517  17C54EF0   --0001--  5D5E40E3

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA400096DA   NEWRET=000096DE   ERRET=000096DE
    PARM.-LIST =  --WAIT--     --RD  --   01009774  E4000007  --TYPL--  --IN  --   01009386  D2000020
                  --TYPL--     --IN  --   0100960C  C200001A  D95E40E3  7EFC4EF0   F361F04E  --08 2--

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
```

Figure 15. Sample offline printout of trace information recorded by the SETOVER command

Purpose:

The purpose of the DEBUG command is to provide the user with online facilities for debugging programs and to provide an entry in CMS for handling external and program interrupts and unrecoverable errors.

Format:

```
---------
| DEBUG |
---------
```

Usage:

The facilites of DEBUG are made available to the user when the DEBUG command is issued, an external interrupt occurs, a program interrupt occurs, a breakpoint is encountered during program execution, or an unrecoverable error occurs. Once DEBUG has been entered due to any of the above circumstances, the user is said to be in the Debug environment. The only valid input in this environment is the group of DEBUG requests discussed in this section. Five of the requests--GO, IPL, KX, RETURN, and RESTART--cause the user to leave the Debug environment. Which of these five requests should be issued depends on the circumstances under which DEBUG is entered. Refer to the section dealing with each request for a further discussion of its use.

When the Debug environment is entered, the contents of all general-purpose registers, the channel status word, and the channel address word are saved so they may be examined and changed prior to being restored when leaving the Debug environment. If DEBUG is entered via an interrupt, the old program status word for that interrupt is also saved. The requests which may be issued in the Debug environment allow the user to examine and change the contents of these control words and registers as well as portions of the user's virtual core. Each of these requests is described individually in the following sections.

Notes:

a. KT and KO are not recognized in the Debug environment.

b. The floating-point registers may not currently be examined or changed in the Debug environment. To access the floating-point registers, the CP console functions DISPLAY Yreg and STORE Yreg may be used.

Responses:

DEBUG ENTERED...
This message indicates that DEBUG has been entered in

response to the DEBUG command or due to an unrecoverable error encountered during execution. Any DEBUG request may be issued as soon as the keyboard is unlocked.

DEBUG ENTERED, EXTERNAL INT.
This message indicates that an external interrupt has caused DEBUG to be entered and that the external old program status word is saved. Any DEBUG request may be issued as soon as the keyboard is unlocked.

DEBUG ENTERED PROGRAM INT. PSW = xxxxxxxxxxxxxxxx
This message indicates that a program interrupt has caused DEBUG to be entered. The program old PSW is saved, and its contents typed in hexadecimal representation as indicated by xxxxxxxxxxxxxxxx above. Any DEBUG request may be issued as soon as the keyboard is unlocked.

DEBUG ENTERED BREAKPOINT xx AT xxxxxx
This message is typed when DEBUG is entered due to encountering a breakpoint during the execution of a program. The breakpoint is identified by the number assigned to it (xx) and by the hexadecimal core location (xxxxxx) at which it is encountered. Any DEBUG request may be issued as soon as the keyboard is unlocked.

INVALID DEBUG REQUEST
The user has specified a request which is not valid in the Debug environment or which includes the wrong number of operands. Only the requests discussed in this section are valid.

Requests:

Whenever the keyboard is unlocked in the Debug environment, any DEBUG request may be issued. The following rules apply when issuing DEBUG requests:

(1) The parameters, or operands, of each request must be separated by one or more blanks.

(2) The character-delete symbol, @, may be used to delete individual characters in an input line and n character-delete symbols delete the preceding n characters in the line.

(3) The line-delete symbol, ¢, may be used to delete itself and all preceding characters in an input line. A line-delete symbol cannot be deleted by a character-delete symbol.

(4) All operands longer than eight characters are left-justified and truncated on the right.

(5) All entries in the DEBUG symbol table are created by the DEF request.

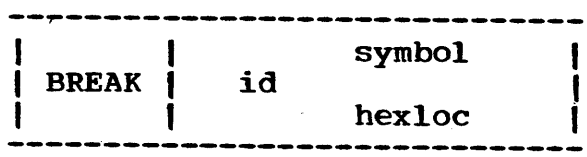
(6) The DEBUG requests can be abbreviated by specifying a minimum of two characters, except for X, RETURN, and RESTART.

Below is a list of all valid DEBUG requests and their minimum abbreviation.

REQUESTS	Minimum Abbreviation
BREAK	BR
CAW	CA
CSW	CS
DEF	DE
DUMP	DU
GO	GO
GPR	GP
IPL	IP
KX	KX
ORIGIN	OR
PSW	PS
RESTART	REST
RETURN	RET
SET	SE
STORE	ST
TIN	TI
X	X

BREAK

Format:



id is any decimal number between 0 and 15 inclusive

symbol is a name which has been assigned (using the DEF request) to the core address at which a breakpoint is to be set

hexloc is the hexadecimal core location (relative to the current origin) at which a breakpoint is to be set

Usage:

This request enables the user to stop execution of a program at specific instruction location called breakpoints. Issuing the BREAK request causes only one breakpoint to be set. Separate BREAK requests must be issued for each breakpoint desired. A maximum of sixteen breakpoints may be in effect at any given time, and any attempt to set more than sixteen is rejected.

The first operand of the BREAK request specifies the identification number assigned to the breakpoint being set. It must be a decimal number between 0 and 15 inclusive. If an identification number is specified which is the same as a currently set breakpoint, the previous breakpoint is cleared and the new one is set.

The second operand of the BREAK request indicates the core address at which the breakpoint is to be set. If this operand contains any non-numeric characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the breakpoint is set at the core address to which the symbol name is assigned provided that address is on an even (halfword) boundary. If no match is found in the DEBUG symbol table, or if the second operand contains only numeric characters, the current origin as established by the ORIGIN request is added to the specified operand and the breakpoint is set at the resulting core address provided that address is on a halfword boundary.

The DEBUG program sets a breakpoint by saving the contents of the byte located at the core address specified by the second operand of the BREAK request. This byte is replaced by the byte EX where x is the hexadecimal equivalent of the breakpoint identifier specified in the first operand. For the breakpoint setting to have meaning, the core address indicated by the second operand must be the location of an

operation code. Thus, when the location is encountered during program execution, a program interrupt occurs because all values E0 through EF are invalid operation codes and control is transferred to the Debug environment. In DEBUG the invalid operation code is recognized as a breakpoint, the original operation code is replaced, and a message is typed identifying the breakpoint encountered.

Figure 16 gives the procedure normally used for setting breakpoints. The program is loaded into core and DEBUG is issued to transfer control to the Debug environment so breakpoints may be set prior to execution. After the desired breakpoint have been set, RETURN should be issued to return control to the CMS Command environment. Issuing the START command causes program execution to begin. Whenever a breakpoint is encountered, a message to that effect is typed, control is returned to the Debug environment, and the keyboard is unlocked to accept any DEBUG request except RETURN. Issuing the GO request causes program execution to continue from a specified location or the location at which the breakpoint had been set.

Notes:

a. A breakpoint is cleared when it is encountered during program execution.

b. To obtain the core addresses of instructions at which breakpoints are to be set, a listing of the program(s) in assembler language mnemonics should be used together with a load map. Assembler language mnemonics are obtained in Fortran listings by specifying the LIST option when the FORTRAN command is issued; see Figure 17. To obtain a load map, the TYPE option is specified in the LOAD command as shown in Figure 15.

c. If the address specified for a breakpoint setting is on a halfword boundary, the byte at that address may not contain an operation code. It is up to the user to make sure that breakpoints are set only at operation code locations. Otherwise, the breakpoint is not recognized during execution and may generate other errors by overlaying data or some part of an instruction other than the operation code.

d. No breakpoints should be set below hexadecimal core location 100 since this area is reserved for hardware control words, and does not contain executable code.

e. If BREAK specifies a core address at which a breakpoint is currently active, the second breakpoint is set at that same location. When encountered during execution, the identification number of the most recently set breakpoint is typed. The second time this core location is reached during program execution, the identifier of the second-most recently set breakpoint is typed, and so on. When DEBUG has

been entered due to a breakpoint, issuing the GO request without an operand causes execution to begin at the location at which the breakpoint was encountered. If more than one breakpoint has been set at this location, the additional breakpoint(s) causes DEBUG to be re-entered.

Responses:

If the BREAK request is correctly issued, the keyboard is unlocked following a carriage return, and the system is ready to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that some number other than two operands have been specified.

INVALID ARGUMENT
This message indicates that the breakpoint identification number specified in the first operand is not a decimal number between 0 and 15 inclusive, or the second operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the second operand is intended to be a symbol, a DEF request must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal core location.

INVALID CORE-ADDRESS
The core location indicated by the second operand is uneven (not on a halfword boundary) or the sum of the second operand and the current origin value is greater than the user's virtual core size. If the current origin value is unknown, it may be reset to the desired value by issuing the ORIGIN request.

REPLACES OLD BREAKPOINT xx AT xxxxxx
This response indicates that the BREAK request just issued specifies a breakpoint identifier (xx) which is assigned to a currently active breakpoint. The old breakpoint (at core location xxxxxx) is cleared and the new breakpoint is set.

DEBUG ENTERED
BREAKPOINT xx AT xxxxxx
This message is given when a breakpoint is encountered during program execution. xx is the breakpoint identifier and xxxxxx is the core address of the breakpoint. After the message is typed, the keyboard is unlocked to accept any DEBUG request except RETURN.

Examples:

a. BREAK 1 18C
The current origin value is added to 18C and the byte at the resulting core address is saved and replaced by the byte 'E1'. Refer to Figure 16 where the origin is set to 12000, and the instruction at breakpoint 1 is set is the second from the last instruction shown in Figure 17. Note that the

load map indicates that program PRIME is loaded at 12000. Setting the origin to 12000, therefore, means that the statement locations shown in the listing of program PRIME may be used in setting breakpoints. When breakpoint 1 is encountered during program execution, the message

```
DEBUG ENTERED
BREAKPOINT 01 AT 01218C
```

is typed.

b. BREAK 3 AAA

The byte at the address assigned to symbol AAA is saved and replaced by the 'E3'. In Figure 16 a DEF request is issued which assigns symbol AAA to location 1A4 relative to the current origin of 12000. Breakpoint 3 therefore sets at core address 121A4, as indicated by the message

```
DEBUG ENTERED
BREAKPOINT 03 AT 0121A4
```

which is typed when the breakpoint is encountered during program execution.

```
printf prime listing
  FORTRAN IV G LEVEL 0, MOD 0        PRIME          DATE = 67080
FILE PRIME                             CAMBRIDGE   MONITOR   SYSTEM
                C      PRIME NUMBER PROBLEM
    0001          100 WRITE (6,8)
    0002            8 FORMAT (27H PRIME NUMBERS FROM 1 TO
                      50/2X,1H1/2X,1H2/2X,1H3)
    0003          101 I=5
    0004            3 A=1
    0005          102 A=SQRT(A)
    0006          103 J=A
    0007          104 DO 1 K=3,J,2
    0008          105 L=I/K
    0009          106 IF(L*K-I)1,2,4
    0010            1 CONTINUE
    0011          107 WRITE (6,5)I
    0012            5 FORMAT (I3)
    0013            2 I=I+2
    0014          108 IF (50-I)7,4,3
    0015            4 WRITE (6,9)
    0016            9 FORMAT (14H PROGRAM ERROR)
    0017            7 WRITE (6,6)
    0018            6 FORMAT (4H END)
    0019          109 STOP
    0020              END
  FORTRAN IV G LEVEL 0, MOD 0        PRIME          DATE = 67080
FILE PRIME                             CAMBRIDGE   MONITOR   SYSTEM
  FORTRAN IV G LEVEL 0, MOD 0          PRIME          DATE = 67080
FILE PRIME                             CAMBRIDGE   MONITOR   SYSTEM
    LOCATION  STA NUM LABEL    OP     OPERAND          BCD OPERAND
    000000                     BC     15,12(0,15)
    000004                     DC     06D7D9C9
    000008                     DC     D4C54040
    00000C                     STM    14,12,12(13)
    000010                     LM     2,3,40(15)
    000014                     LR     4,13
    000016                     L      13,36(0,15)
    00001A                     ST     13,8(0,4)
    00001E                     STM    3,4,0(13)
    000022                     BCR    15,2
    000024                     DC     00000000         A4
    000028                     DC     00000000         A20
    00002C                     DC     00000000         A36
    00016C           A36       L      13,4(0,13)
    000170                     L      14,12(0,13)
    000174                     LM     2,12,28(13)
    000178                     MVI    12(13),255
    00017C                     BCR    15,14
    00017E           A20       L      15,160(0,13)     IBCOM#
    000182                     LR     12,13
    000184                     LR     13,4
    000186                     BAL    14,64(0,15)
```

Figure 17. Sample output created by a FORTRAN command in which the LIST and SOURCE options are specified.

```
load prime (type)
 PRIME#    AT 12000
 PRIME     AT 12000
 IHCFCOMH  AT 122A8
 SAVAREA   AT 13270
 VFIOCS    AT 13110
 IBCOM#    AT 122A8
 FDIOCS#   AT 12364
 IHCSSQRT  AT 132E0
 SQRT      AT 132E0
 IHCFCVTH  AT 13390
 ADCON#    AT 13390
 INT6SW    AT 143F1
 FCVEO     AT 13E3A
 FCVLO     AT 13612
 FCVIO     AT 13948
 FCVCO     AT 1403C
 FCVAO     AT 13582
 FCVZO     AT 134DC
 IHCFIOSH  AT 14410
 FIOCS#    AT 14410
 IHCTRCH   AT 14FA8
 IHCUATBL  AT 15220
R;T=0.39/0.60

debug
DEBUG ENTERED...

origin 12000

break 1 18c

def aaa 1a4

break 3 aaa

return
R;T=0.02

start
EXECUTION BEGINS...
DEBUG ENTERED
BREAKPOINT 01 AT 01218C

go

PRIME NUMBERS FROM 1 TO 50
 1
 2
 3
DEBUG ENTERED
BREAKPOINT 03 AT 0121A4
```

Figure 16. Sample procedure for setting breakpoints

CAW

Format:

```
----------
|  CAW   |
----------
```

Usage:

This request causes the contents of the channel address word which existed at the time DEBUG was entered to be typed at the terminal. The CAW, specifies the storage protection key and the core address of the first channel command word associated with the next or most recent START I/O. The CAW located in hexadecimal core location 48, is saved at the time DEBUG is entered. It has the following format:

Bits	Contents
0-3	Protection key which is matched with a key in storage whenever reference is made to main storage.
4-7	Not implemented; currently set to zeros.
8-31	Command address, indicating the hexadecimal core location of the first channel command word associated with the next or the most recent START I/O.

For a further discussion of the channel address word, refer to the IBM manual, IBM System/360 Principles of Operation.

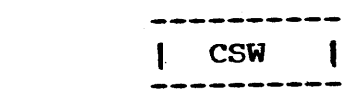
Responses:

If the request is issued correctly, the contents of the channel address word are typed in hexadecimal representation at the terminal and, following a carriage return, the keyboard is unlocked to accept another DEBUG request. See Figure 18 for an example of response to the CAW request.

INVALID DEBUG REQUEST
This response to the CAW request indicates that one or more operands have been specified. Reissue the request in its correct format.

CSW

Format:



Usage:

This request causes the contents of the channel status word which existed at the time DEBUG was entered to be typed at the terminal. The CSW indicates the status of a channel or an input/output device, or the conditions under which an I/O operation has been terminated. The CSW is formed in the channel and stored in hexadecimal core location 40 when an I/O interruption occurs. If I/O interruptions have been suppressed, the CSW is stored when the next START I/O, TEST I/O, or HALT I/O instruction is executed. The CSW is saved when DEBUG is entered and has the following format:

Bits	Contents
0-3	Protection key moved from the CAW and used to indicate the protection key under which I/O was started.
4-7	Not implemented; currently set to zeros.
8-31	Next command address--a pointer to the core location eight bytes greater than the address of the last channel command word executed.
32-47	Status bits indicating the conditions in the device or the channel that caused the CSW to be stored.
48-63	Residual count indicating the difference in the number of bytes specified in the last executed channel command word and the number of bytes which were actually transferred.

Responses:

If the request is issued correctly, the contents of the CSW are typed at the terminal in hexadecimal representation. A carriage return is then issued and the keyboard is unlocked to accept another DEBUG request. For an example of response to the CSW request see Figure 18.

INVALID DEBUG REQUEST
This response to the CSW request indicates that one or more operands have been specified. Reissue the request in its proper format.

DEF

Format:

DEF	symbol hexloc < bytecount >
	4

symbol is the name to be assigned to the core address derived from the second operand, hexloc

hexloc is the hexadecimal core location, relative to the current origin, to which the name specified in the first operand is to be assigned

bytecount is a decimal number between 1 and 56 inclusive which specifies the length attribute (in bytes) of the symbol specified in the first operand.

Usage:

The DEF request allows the user to assign a symbolic name to a specific core address and to refer to that address in other DEBUG requests by the assigned name. The symbol name is specified as the first operand of the DEF request. It may be from one to eight characters in length, must contain at least one non-numeric character, and the first character of the symbol name should not be an asterisk. Any name longer than eight characters will be left-justified and truncated on the right.

The second operand specifies a hexadecimal number which is added to the current origin as established by the ORIGIN request. The sum of these two values is the core address to which the symbol name is assigned.

The third operand is optional and, if given, must specify a decimal number between 1 and 56 inclusive. This number is the length attribute in bytes of the symbol name. If the third operand is omitted, a default attribute of four bytes is assigned.

When DEF is issued, an entry is made in the DEBUG symbol table indicating the symbol name, the core address to which it is assigned, and the length attribute of the symbol. Symbols remain defined until a new DEF request is issued for them or until the user obtains a new copy of CMS by issuing an IPL request in the Debug environment or in the Control Program environment.

If DEF is issued specifying a symbol that has been previously defined, the previous core address is replaced by the more recent core address for that symbol in the DEBUG symbol table. DEF requests specifying additional symbol names for core locations to which a symbol name has already

been assigned cause additional entries in the DEBUG symbol table so that multiple symbols may be assigned to the same core address.

Notes:

a. Only sixteen symbols may be defined in the Debug environment at any given time.

b. Issuing a new ORIGIN request does not affect core addresses to which previously defined symbols are assigned.

c. Symbols assigned using DEF are defined for use only in the Debug environment.

Responses:

If DEF is issued correctly, a carriage return is given and the keyboard is unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response to DEF indicates that less than two or more than three operands have been specified. Reissue the request in its correct format.

INVALID ARGUMENT
This message indicates that the name specified in the first operand contains all numeric characters, the second operand is not a valid hexadecimal number, or the third operand is not a decimal number between 1 and 56 inclusive.

INVALID CORE-ADDRESS
This response is given when the sum of the second operand and the current origin is greater than the user's virtual core size. If the current origin value is unknown, reset it to the desired value by issuing the ORIGIN request and reissue the DEF request.

16 SYMBOLS ALREADY DEFINED
If this message is given, the DEBUG symbol table has been filled and no new symbols may be defined until the current definitions are cleared by obtaining a new copy of CMS. However, existing symbol may be assigned to a new core location by issuing another DEF request for that symbol.

Examples:

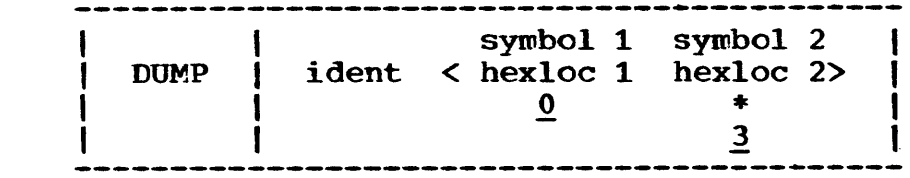
a. DEFINE IN1 12F5A
The current origin value is added to 12F5A and the symbol IN1 is assigned to the resulting hexadecimal core address. The default length attribute of 4 is assigned to symbol IN1, and an entry for IN1 is made in the DEBUG symbol table.

b. DEFINE K 13 12
The currently defined origin is added to the hexadecimal value 13, and the resulting address is assigned the symbol

K. An entry for K is made in the DEBUG symbol table, and its length attribute is 12 bytes.

DUMP

Format:



ident indicates the name by which the printout is identified

symbol1 is a name assigned (using DEF) to the core address at which the dump is to begin

hexloc1 is the hexadecimal core location, relative to the current origin, at which the dump is to begin

symbol2 is a name assigned (using DEF) to the core address at which the dump is to end

hexloc2 is the hexadecimal core location, relative to the current origin, at which the dump is to end

* indicates that the dump is to end at the last address of the user's virtual core

Usage:

This request is used to dump the contents of all or part of the user's virtual core on the offline printer. The information has the heading "ident FROM symbol1 TO symbol2" where ident is the identifier specified as the first operand of the DUMP request.

The second and third operands specify the portion of core which is to be dumped, and are optional. If omitted, the core address specified in the most recent DUMP request is used or, if no previous DUMP request has been issued, one word (four bytes) of core is dumped starting at location 0.

If the second and third operands are specified, the core addresses to which they refer are determined in the following way. If the second operand contains any non-numeric character, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the core address to which that symbol name is assigned is used as the address at which the dump is to begin. If no match is found, or if the operand contains only numeric characters, the current origin as established by the ORIGIN request is added to the specified operand. The resulting core address is used as the beginning address of the dump, provided it is not greater than the user's virtual core size. The core address at which the dump is to end is given by the third operand of the DUMP request. If an asterisk is specified

for this operand, all of core from the starting address to the end of core is dumped. If an asterisk is not specified as the third operand, the same procedure is used to determine the ending address of the dump as that described above for the starting address. Both addresses must be within the address range of the user's virtual core, and the address specified in the third operand must be greater than that specified in the second.

The first three lines of output from the DUMP request give the contents of general-purpose registers 0-7 and 8-15, and the floating-point registers 0-6. Thereafter, the contents of the specified portion of core are given, 32 bytes per line. The core address of the first byte in the line is given in the left-most column of the dump and is always an even doubleword boundary.

The alphameric interpretation for the 32 bytes is printed to the right of the specified hexadecimal locations.

Responses:

A carriage return is issued and the keyboard is unlocked to accept another DEBUG request. The requested information is printed offline as soon as the printer is available.

INVALID DEBUG REQUEST
No, two, or more than three operands have been specified in the DUMP request. Reissue the request, specifying one or three operands.

INVALID ARGUMENT
This message is given if the address specified by the third operand is less than that specified by the second operand or the second and/or third operands cannot be located in the DEBUG symbol table and are not valid hexadecimal numbers. If either operand is intended to be a symbol, a DEF request must previously have been issued for that symbol; if not, the operand must specify a valid hexadecimal core location.

INVALID CORE-ADDRESS
The hexadecimal number specified in the second or third operand, when added to the current origin, is greater than the user's virtual core size. If the current origin value is unknown, reset it to the desired value by issuing ORIGIN and reissue the DUMP request.

Example:

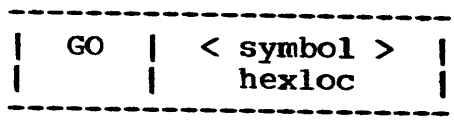
DUMP NEUC 0000 05F0
The contents of core from locations 0000 to 05F0 (each plus the current origin) are printed on the offline printer and identified by the heading NEUC. See Figure 19 for a sample of this output, where the origin value is 0.

```
                                    NEUC:  FROM      000 TO     05F0

GR 0-7  C4C5C2E4  00009B10  00009B10  00000000      00000778  000029AC  000C0A2C  00CC0048
GR 8-F  00000004  00000100  C4C5C2E4  C74C4040      0C00A3B0  00000AA0  0000026E  0000A3B0
FPREGS  00000000  00000000  00000000  00000000      00000000  00000000  00000000  00000000

000000  800094C0  60009580  00000002  60000050      0203FB10  0000A3B0  FF0C0CEC  8003F298  *.............................2.*
000020  000400CA  60009516  00000005  6C00D0C2      CC000000  00000000  FF040009  0000223E  *.....................................*
0CCC4C  0000028E  0C000000  0000A048  00000000      0001F69E  00000000  00000000  0000BD00  *..................6..............*
000060  00040000  00002288  00C40000  0000BB00      00000000  0000BC2C  0004CCCC  00CC05AC  *.....................................*
0C0080  00000000  00000000  00000000  00C00000      C4C5C2E4  C74C4040  00016D1E  00000000  *.................DEBUG  ........*
0C0CA0  00000CCC  00000000  00000000  00000000      00000000  00000000  0000000C  0C000000  *.....................................*
0000C0  00003000  00000000  00000000  00000000      CC000000  00000000  00000000  00000000  *.....................................*
0C0CEC  00000CCC  00000000  00000000  0000CFA2      070058C0  00EC07FC  00000000  00000000  *.....................................*

000100  00000AFC  00040000  00012000  00012000      00000000  00040000  0000CCCC  CC0C00CA  *...0................................*
000120  4000A56A  0000A56A  0000A238  C4C5C2E4      0C00A76C  00009B10  00000000  00000778  * ...........DEBU.................*
0C0140  000C29AC  00000A20  00000009  0000A3B0      00000009  9000A564  C74C4C4C  000CA438  *.........................G    ....*
000160  00003AA0  000026CE  00C00000  00000000      C00400CA  400057F4  000057B8  000057F8  *..........................  ..4...6...8*
000180  C3D3D9D6  0000139C  00000000  00005816      00009E10  C3D3D9D6  C00C5CA8  000E8BB8  *CLRO................CLRO......Y.*
0001A0  00009386  00012000  00C09508  00000001      4C0057E8  000011B0  4000C954  0CCC0006  *.................. ..Y.... .. ....*
0C01CC  00000000  0003F000  00000000  00000000      CC000000  00000000  0003E000  00012000  *.......0............................*
0C01E0  00000190  000001E5  00020000  C3D6D5E2      00000000  00000000  0A00C25E  6CCC0C82  *........V....CONS..................*
000200  03000000  20000001  E6C1C9E3  404C404C      C3D6D5F1  00000000  C3D6E5F1  00000000  *.........WAIT    CON1....CON1....*
0C0220  00000238  00050000  00000000  00000000      0000023C  000026CE  0000CCCC  0A82C4E4  *.....................................DU*
0C0240  D4D740D5  C5E4C340  F0F0F040  F0F5C6F0      00000000  0000000C  00CCCCCC  CCCC0C0C  *HP NEUC 000 05F0...................*
000260  00000000  00000000  00000000  00000000      CC0C0000  0C000000  00000000  00000000  *.....................................*
                  *** DUPLICATE CORE LOCATIONS ***
0002C0  000002C4  891AD95E  40E37EF0  4BF0F161      F04BF0F2  40F2F04B  F5F04EF5  F3151717  *...D..B. T.0.01.0.02 20.50.53...*
0C02E0  F04BF3F6  15171700  00000000  00000000      00000000  00000000  00000000  00000000  *0.36................................*
000300  00000000  00000000  00000000  00000000      00000000  0000000C  C0CCCCCC  CCCC0C00  *.....................................*
                  *** DUPLICATE CORE LOCATIONS ***
0C03EC  00000CCC  00000000  00000000  00000000      00000000  000003F4  0000C3F9  0CC40000  *.....................4...5....*
000400  00090000  C3D6D5F1  0000C2E0  019C00C8      C4E2D2F1  00001050  0191CCC8  C4E2D2F2  *....CON1..B.....DSK1.........DSK2*
0C0420  0C001050  01920000  C4E2D2F3  0C001050      0C0C0000  D9C4D9F1  00000000  000E0000  *........DSK3.........RDR1.........*
0C0440  D7C3C8F1  0000000C  000E0000  D7D9D5F1      00000000  0180000C  E3C1D7F1  00CC0000  *PCH1.........PRN1.........TAP1....*
000460  01810000  E3C1D7F2  00000000  00000000      C4E2D2F4  0C001050  00000000  C4E2D2F5  *....TAP2.........DSK4.........DSK5*
0C0480  00001C50  019C0008  C4E2D2F6  00001050      01060000  C3D9E3F1  C00CCCCC  00000000  *.........DSK6.........CRT1.........*
0004A0  00000000  C3D6D5F1  000011E0  00009774      00000E10  00000400  00002C78  00CC9EC0  *....CON1...........................*
0C04C0  00001EF8  00002C98  00C0BBBC  0C0019C0      CC0C19C0  00009EF6  000014C0  000034C8  *...8.....................6.......H*
0C04E0  00009828  000104D8  00001A28  00000E00      0000049C  00001A9C  000CC4F9  00CC0000  *.......C.....................9....*
000500  00003504  00008EE0  00001EA0  000C0E3C      C000031F0  000032F8  00009E58  000C051D  *.....................0...8........*
0C0520  000019F0  00000525  000093A6  00001060      CC00237C  00000778  00000EA0  00000530  *...0................................*
0C0540  00CC0778  00000545  000057CA  00010FF8      00000000  0000000C  C00CCCCC  000CA078  *.................8..................*
000560  0000CC88  00009B10  0000C569  00009DF0      CCFFFFFF  00003870  00003880  0000057D  *.....................0..............*
0C0580  00001A58  00000585  0000BAEC  0000BAF4      000006A8  00000000  0000BAC8  0CCC0000  *.....................4..............*
0005A0  50C00000  05C0900F  C162D203  C1920000      98030038  41E0C0A0  984BCCF2  144C1259  *........A.K.A.................2. ..*
0C05C0  4786C02C  41F0C05C  49409000  078F879A      CC2241F0  C0884190  C0409180  50020789  *.....0... .........0..... ......*
0C05E0  49405CCC  078F8756  C0345850  C0F64940      5000078F  8756C048  4040CC5A  47FCCCBA  *. ............6. ........  ...0..*
```

Figure 19. Sample DUMP output from the offline printer

GO

Format:



symbol is a name that has been assigned by the DEF request to the core address at which execution is to begin

hexloc is the hexadecimal core location, relative to the current origin, at which execution is to begin

Usage:

The GO request causes the user to leave the Debug environment and begin execution at a user-specified address or at the address contained in bits 40-63 of the old PSW for the interrupt which caused DEBUG to be entered. This PSW is saved when DEBUG is entered, and is loaded as the current PSW when GO is issued. If an operand is specified in the GO request, the instruction address portion of the PSW is altered to contain the address indicated by that operand before the PSW is loaded.

The core address indicated by the GO operand is determined in the following way. If the operand contains any non-numeric characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the core address to which that symbol name is assigned is used as the location at which execution is to begin, and is moved to the saved old PSW. If no match is found in the DEBUG symbol table, or if the operand contains only numeric characters, the current origin (as established by the ORIGIN request) is added to the specified operand and the resulting address moved to the PSW, provided it is not greater than the user's virtual core size.

Prior to loading the PSW, the general-purpose registers, channel address word, and channel status word are restored to their contents as they existed when DEBUG was entered, or, if they have been modified by the user in the Debug environment, to their modified contents. The saved old PSW is then loaded to become the current PSW, and execution begins at the specified instruction address.

Notes:

a. GO should not be issued without an operand unless the Debug environment has been entered due to a breakpoint, external, or program interrupt.

b. When an operand is specified, GO may be issued at any time in the Debug environment, except when DEBUG has been

entered to set breakpoints in a program prior to starting it.

c. If an operand is specified in the GO request, the address to which it refers must be the core location of an operation code.

Responses:

If GO is issued successfully, there is a carriage return and execution coninues from the address contained in the loaded PSW.

INVALID DEBUG REQUEST
This response indicates that more than one operand has been specified in the GO request. Reissue the GO request in its correct format.

INVALID ARGUMENT
An operand specified in the GO request cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the operand is intended to be a symbol, a DEF request must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal core location.

INVALID CORE-ADDRESS
The address at which execution is to begin is not on a halfword boundary (indicating that an operation code is not located at that address) or the sum of the GO operand and the current origin value is greater than the user's virtual core size. If the current origin value is unknown, it may be reset to the desired value by issuing the ORIGIN request.

INCORRECT DEBUG EXIT
The GO request has been issued without an operand when DEBUG had not been entered due to a breakpoint, external, or program interrupt. The IPL, KX, or RESTART requests may be issued, GO may be issued with an operand, or RETURN may be issued if DEBUG had been entered via the DEBUG command.

Examples:

a. GO
The old PSW for the interrupt that caused DEBUG to be entered is loaded as the current PSW, and execution begins at the address specified in bits 40-63 of that PSW.

b. GO INN
The DEBUG symbol table is searched for symbol INN and the address to which that symbol is assigned is loaded into bits 40-63 of the old PSW prior to loading it as the current PSW. Control passes from the Debug environment and execution begins at the core address to which symbol INN refers.

c. GO 12345
The current origin is added to location 12345 and the resulting address placed in bits 40-63 of the old PSW prior to loading it as the current PSW. Control is transferred from the Debug environment and execution begins at the specified address.

GPR

Format:

```
-------------------------
| GPR | reg1 < regN > |
-------------------------
```

reg1 is a decimal number from 0-15 inclusive, indicating the first or only general-purpose register whose contents are to be typed.

regN is a decimal number from 0-15 inclusive, indicating the last general-purpose register whose contents are to be typed.

Usage:

The GPR request is used to inspect the contents of one or more general-purpose registers as they existed when DEBUG was entered. If only one operand is given, the contents of the specified register are typed at the terminal. If two operands are given, the contents of the registers specified by the first through the second operand, inclusive, are typed.

The first and second operands must be decimal numbers from 0-15, and the second operand must be greater than the first.

Responses:

If the request is issued correctly, the contents of the register(s) specified are typed at the terminal, one per line. Following a carriage return, the keyboard is unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This message indicates that none or more than two operands have been specified. Reissue the request in its proper format.

INVALID ARGUMENT
This response is given if the operand(s) specified are not decimal numbers between 0 and 15, or if the second operand is less than the first.

Examples:

a. GPR 8
The contents of general-purpose register 8 as it existed when DEBUG was entered are typed at the terminal. See Figure 20.

b. GPR 5 15
The contents of general-purpose registers 5 through 15, inclusive, are typed as they existed when DEBUG was entered.

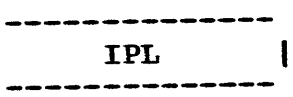
See Figure 20.

```
gpr 8
002A1A88


gpr 5 15
000A5DD8
002A1A88
00009B38
002A1A88
00000100
80009670
80009FBA
600095C2
00011D40
0000550C
000095B8
```

Figure 20. Examples of the GPR request

IPL

Format:



Usage:

The IPL request causes control to transfer from the Debug environment to the CMS Command environment. IPL may be issued any time the keyboard is unlocked in the Debug environment regardless of the circumstances by which DEBUG is entered.

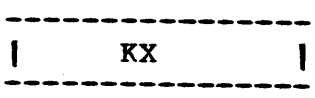
Responses:

CMS...VERSION nn LEVEL mm
This indicates that IPL has successfully executed and control has passed from the Debug environment to the CMS Command environment. The keyboard is unlocked to accept any CMS command.

INVALID DEBUG REQUEST
This indicates that one or more operands have been specified in the IPL request. Reissue the request in its correct format.

KX

Format:



Usage:

The KX request closes all open files and I/O devices, updates the user's file directory, and reIPL's CMS. This request may be issued whenever the keyboard is unlocked in the Debug environment, regardless of the circumstances by which DEBUG is entered.

Responses:

KILLING CMS EXECUTION . . .
CMS...VERSION nn LEVEL mm
This indicates that KX has executed, CMS has reIPL'ed, and and control has passed to the CMS command environment. The keyboard is unlocked to accept any CMS command.

INVALID DEBUG REQUEST
This indicates that one or more operands have been specified in the KX request. Reissue the request in its correct format.

ORIGIN

Format:

```
+----------+----------+
| ORIGIN   | symbol   |
|          | hexloc   |
+----------+----------+
```

symbol is any name that has been assigned to a core address by the DEF request

hexloc is any hexadecimal core location between 0 and the end of the user's virtual core

Usage:

This request allows the user to specify an origin, or base address, which is added to the hexadecimal locations specified in other DEBUG requests. For example, the ORIGIN request enables users to specify instruction addresses relative to program load points, rather than to 0, while operating in the Debug environment. If the ORIGIN request is not issued, all hexadecimal locations specified in DEBUG requests are assumed to be relative to 0.

When ORIGIN is issued, the origin setting is determined in the following way. If the ORIGIN operand contains any non-numeric characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the core address to which that symbol name is assigned becomes the new origin setting. If no match is found in the DEBUG symbol table, or if the operand contains only numeric characters, the address specified in the operand becomes the origin setting.

Any origin set by an ORIGIN request remains in effect until another ORIGIN request or a RESTART request is issued, or until the user obtains a new copy of CMS. Whenever a new ORIGIN request is issued, the value specified in that request overlays the previous origin setting. If the user obtains a new copy of CMS or issues a RESTART request, the origin is set to 0 until a new ORIGIN request is issued.

Responses:

If the request is issued correctly, there is a carriage return and the keyboard is unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that the wrong number of operands have been specified. One and only one operand must be specified.

INVALID ARGUMENT
The operand specified in the ORIGIN request cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the operand is intended to be a symbol, a DEF request must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal core location.

INVALID CORE-ADDRESS
The address specified by the ORIGIN operand is greater than the user's virtual core size.

Examples:

a. ORIGIN 12000
The origin is set to the hexadecimal value of 12000. 12000 is added to all hexadecimal locations specified in other DEBUG requests and the resulting core address is referenced.

b. ORIGIN XYZ5
The absolute address assigned to symbol XYZ5 (provided an entry for XYZ5 exists in the DEBUG symbol table) becomes the new origin setting. This setting is added to all hexadecimal locations specified in other DEBUG requests, and the resulting core address is referenced.

PSW

Format:

```
| PSW |
```

Usage:

This request types the contents of the old program status word for the interrupt that caused DEBUG to be entered. If DEBUG was entered due to an external interrupt, the PSW request causes the contents of the external old PSW to be typed at the terminal. If a program interrupt caused DEBUG to be entered, the contents of the program old PSW are typed. If DEBUG was entered for any other reason, the following is typed in response to the PSW request:

01000000xxxxxxxx

where the 1 in the first byte means that external interrupts are allowed and xxxxxxxx is the hexadecimal core address of the DEBUG program.

The fields of the PSW follow.

Bits	Contents
0-7	System mask, indicating the sources which are allowed to interrupt the CPU.
8-11	Protection key, used to determine if a given core location may be written into.
12	ASCII flag, indicating whether ASCII-8 or EBCDIC code is to be used.
13	Machine check flag, which is set to 1 whenever a machine check occurs.
14	Wait state flag, which is set to 1 when the CPU is in the wait state.
15	Problem state flag, set to 1 when the machine is operating in the problem state rather than the supervisor state.
16-31	Interrupt code, showing the source of the interrupt for external interrupts or the cause of the interrupt for program interrupts.
32-33	Instruction length code, indicating the length, in halfwords, of the instruction being executed when a program interrupt occurred (unpredictable for external interrupts).
34-35	Condition code, which reflects the result of a

previous arithmetic, logical, or I/O operation.

36-39 Program mask, indicating whether or not various program exceptions are allowed to cause program interrupts.

40-63 Instruction address, giving the location of the next instruction to be executed for program interrupts or of the instruction last executed for external interrupts.

For a further discussion of program status words and their use, refer to the IBM manual, IBM System/360 Principles of Operation.

Responses:

If the request is issued correctly, the contents of the appropriate PSW are typed in hexadecimal representation at the terminal, followed by a carriage return and an unlocked keyboard. See Figure 18 for an example of response to the PSW request.

INVALID DEBUG REQUEST
This response indicates that the user has specified one or more operands in the PSW request. Reissue the request in its correct format.

RESTART

Format:

Usage:

The RESTART request is equivalent to the IPL request. RESTART causes control to transfer from the Debug environment to the CMS Command environment. RESTART may be issued any time the keyboard is unlocked in the Debug environment, regardless of the circumstances by which DEBUG has been entered.

Issuing RESTART causes a new copy of the CMS nucleus to be brought into core from the system disk. This new copy overlays the user's former copy of the nucleus, causing all symbols which had been previously defined by DEF requests to be cleared from the DEBUG symbol table.

Responses:

CMS...VERSION nn LEVEL mm
This response indicates that RESTART has successfully executed and control has passed from the Debug environment to the CMS Command environment. The keyboard is unlocked to accept any CMS command.

INVALID DEBUG REQUEST
One or more operands have been specified in the RESTART request. Reissue the request in its correct format.

RETURN

Format:

```
------------
| RETURN   |
------------
```

Usage:

This request is a means of exitting from the Debug environment to the CMS Command environment. It should be used only when DEBUG has been entered by issuing the DEBUG command.

When RETURN is issued, the general-purpose registers are restored with the information they contained at the time DEBUG was entered or, if the user has specified a change to this information while in the Debug environment, with the changed information. In either case, register 15, the error code register, is set to zero. A branch is then made to the address contained in register 14, the normal CMS return register. If DEBUG is entered by issuing the DEBUG command, register 14 contains the address of a central CMS service routine and control transfers directly to the CMS Command environment.

Responses:

R;T=xx.xx/xx.xx
The Ready message followed by a carriage return and an unlocked keyboard indicates that the RETURN request has successfully executed and control has transferred from the Debug environment to the CMS Command environment. After this message is typed, the keyboard is unlocked to accept any CMS command.

INVALID DEBUG REQUEST
This message is given if one or more operands have been specified in the RETURN request. Reissue the request in its correct format.

INCORRECT DEBUG EXIT
If DEBUG is entered due to a program or external interrupt, a breakpoint, or an unrecoverable error, this message is typed in response to the RETURN request. To exit from the Debug environment under the above circumstances, issue GO with an operand, IPL, or RESTART. The GO request may be issued with no operand if DEBUG has not been entered due to an unrecoverable error.

SET

Format:

```
-----------------------------------------------
|       | CAW  hexinfo                        |
|       | CSW  hexinfo  <hexinfo>             |
|  SET  | PSW  hexinfo  <hexinfo>             |
|       | GPR  reg       hexinfo  <hexinfo>   |
-----------------------------------------------
```

CAW the specified information is to be stored in the channel address word that existed at the time DEBUG was entered.

CSW the specified information is to be stored in the channel status word that existed at the time DEBUG was entered.

PSW the specified information is to be stored in the old program status word for the interrupt that caused DEBUG to be entered.

GPR the specified information is to be stored in the general-purpose register given as the second operand.

Usage:

The SET request is used to change the contents of control and general-purpose registers which are moved from their normal locations when the DEBUG environment is entered. The contents of these registers are restored when control transfers from DEBUG to another environment. If register contents have been modified in DEBUG, the changed contents are restored.

The register that is to be modified is specified as the first operand of the SET request, and the information to be inserted in this register is given in hexadecimal format in the hexinfo operand(s). Each hexinfo operand should be from one to four bytes (that is, two to eight hexadecimal digits) in length. If an operand is less than four bytes and contains an uneven number of hexadecimal digits (representing half-byte information) the information is right-justified and the left half of the uneven byte is set to 0, as shown in Example b. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right, as shown in Example d.

The number of bytes that can be stored using the SET request varies depending on the form of the request. With the CAW form, up to four bytes of information may be stored. With the CSW, GPR, and PSW forms, up to eight bytes of information may be stored, but these must be represented in two operands of four bytes each. When two operands of

information are specified, the information is stored in consecutive locations, even if one or both operands contain less than four bytes of information, as shown in Example b.

The contents of registers that have been changed using the SET request are not typed after the request has been issued. To inspect the contents of these registers, the CAW, CSW, PSW, or GPR requests are issued as needed. Figure 17 contains examples of issuing these requests both before and after SET has been issued.

Responses:

If the request is issued correctly, a carriage return is issued and the keyboard is unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that the wrong number of operands have been specified. If the CAW is set, two operands must be given. To set the CSW or the PSW, two or three operands are required. To set a GPR, three or four operands must be given.

INVALID ARGUMENT
This message indicates that either the first operand is not CAW, CSW, PSW or GPR, the first operand is GPR and the second operand is not a decimal number between 0 and 15 inclusive, or one or more of the hexinfo operands does not contain hexadecimal information.

Examples:

a. SET CAW 1100
This example causes the two-bytes 1100 to be placed in the first two bytes of the channel address word that existed when DEBUG was entered. See Figure 18. This new channel address word is restored when an exit is made from the DEBUG environment.

b. SET CSW 001 00FF81
Since an uneven number of bytes is specified in the second operand, a zero is placed in the left-most half-byte, giving 0001. These two bytes, together with the three bytes given in the third operand are placed as a single five-byte field into the CSW that existed when DEBUG was entered. See Figure 18. This new channel status word is restored when leaving the DEBUG environment.

c. SET PSW 01000000 00012036
The contents of the entire program status word for the interrupt that caused DEBUG to be entered are replaced by these eight bytes of information. See Figure 18. This PSW becomes the current PSW when an exit is made from the DEBUG environment.

d. SET GPR 5 000012345

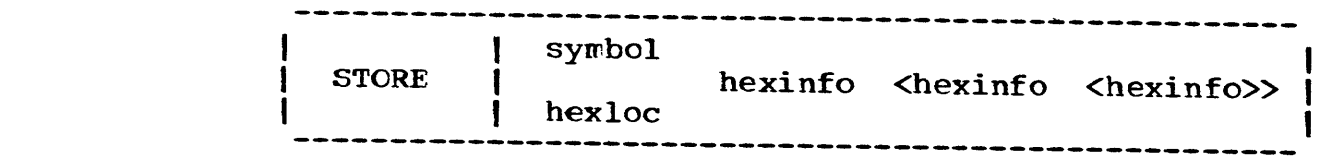
The contents of general-purpose register 5 are replaced with the information given in the third operand. Since this information is greater than eight hexadecimal digits, it is left-justified and truncated on the right, giving 00001234. See Figure 18. General-purpose register 5 contains this new information when the general-purpose registers are restored prior to leaving the DEBUG environment.

```
caw
00010FE8

set caw 1100

caw
11000FF8

csw
00010FF80C000005

set csw 001 00ff81

csw
000100FF81000005

psw
0100000000095B8

set psw 01000000 00012036

psw
0100000000012036

gpr 5
00007F68

set gpr 5 000012345

gpr 5
00001184
```

Figure 18. Examples of the SET request, using other requests as appropriate to inspect contents both before and after SET is issued

STORE

Format:



symbol is a name assigned by the DEF request to the core address at which the first byte of specified information is to be stored

hexloc is the hexadecimal location relative to the current origin where the first byte of information is to be stored

hexinfo is any hexadecimal information, four bytes or less in length, that is to be stored starting at the address specified by the first operand

Usage:

This request allows the user to store information in any virtual core location. The location at which the information is to be stored is specified by the first operand and is determined in the following way. If the first operand contains any non-numeric characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the core address to which that symbol name is assigned is used as the address at which information is to be stored. If no match is found in the DEBUG symbol table, or if the first operand contains only numeric characters, the current origin is added to the specified operand and the resulting core address is used, provided it is not greater than the user's virtual core size.

The information to be stored is given in hexadecimal format and is specified in the second through the fourth operands. Each of these operands is from one to four bytes (that is, two to eight hexadecimal digits) in length. If an operand is less than four bytes in length and contains an uneven number of hexadecimal digits (representing half-byte information), the information is right-justified and the left half of the uneven byte is set to 0 as shown in Example b. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right as shown in Example b.

A maximum of 12 bytes may be stored at one time using the STORE request. This is done by specifying three operands after the location operand, each of which contains four bytes of information. If less than four bytes are specified in any or all of the operands, the information given is arranged into a string of consecutive bytes, and that string

is stored starting at the location given in the first operand. Stored information is not typed at the terminal. To inspect the changed contents of core after a STORE request, issue an X request as shown in Figure 21.

Responses:

If the request is issued correctly, a carriage return is given and the keyboard unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response to STORE indicates that less than two or more than four operands have been specified. Reissue the request in its correct format.

INVALID ARGUMENT
This message is given if the first operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number, or information specified in the second, third, and/or fourth operands is not in hexadecimal format. If the first operand is intended to be a symbol, a DEF request must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal core location.

INVALID CORE-ADDRESS
The current origin value, when added to the hexadecimal number specified as the first operand, gives an address greater than the user's virtual core size. If the origin value is unknown, reset it to the desired value using ORIGIN and reissue the STORE request.

Examples:

a. STORE 12024 0ACA
This causes the two bytes of information 0ACA to be stored at the core address obtained by adding the current origin to location 12024. See Figure 21.

b. STORE XYZ 1341234567890 5CA14 B
Since the second operand in this example contains more than eight digits, it is truncated on the right, giving 13412345. The third and fourth operands, containing an uneven number of digits, become 05CA14 and 0B respectively. This eight-byte string is then stored in the core address indicated by symbol XYZ. See Figure 21.

c. STORE F12 FFFFFFFF FFFFFFFF F0F1F2F3
This example causes the maximum number of bytes, 12, to be stored at location F12. If F12 is a previously-defined symbol, the information is stored starting at the address to which that symbol refers. If no symbol F12 is defined, the current origin is added to the hexadecimal number F12, and the resulting address is used as the starting address into

which the information is stored. See Figure 21.

```
x 12024
E3D640C6

store 12024 0aca

x 12024
0ACA40C6

x xyz 15
C2D8506300044120C2E04800C2E812

store xyz 1341234567890 5ca14 b

x xyz 15
1341234505CA140BC2E04800C2E812

x f12 13
F1EC4370A00141A0A002910FF0

store f12 ffffffff ffffffff f0f1f2f3

x f12 13
FFFFFFFFFFFFFFFF0F1F2F3F0
```

Figure 21. Examples of the STORE request, using the X request to inspect the contents both before and after storing

TIN

Format:

Usage:

The TIN request determines how DEBUG is to handle I/O in the Debug environment, either by the normal CMS I/O routines or by DEBUG itself.

TIN CMS allows CMS to handle the I/O for the Debug environment when that environment is entered either by typing DEBUG or by encountering a breakpoint (that is, when the Debug environment is entered on purpose). This permits input to be entered as normally in CMS, including the ability to have input lines stacked in an EXEC file. TIN CMS is the default mode of I/O operation in the Debug environment.

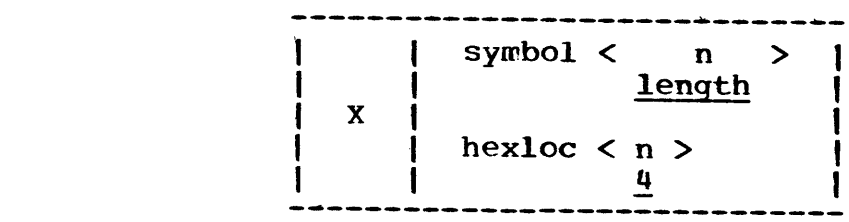
TIN DEB accepts input to the Debug environment only from the terminal as it allows Debug to handle its own I/O. This is the method used in previous versions of CMS.

Responses:

There are no responses to this request other than the unlocking of the keyboard.

X

Format:



symbol is a name assigned by using the DEF request to the core address of the first byte to be examined

hexloc is the hexadecimal core location relative to the currently defined origin of the first byte to be examined

n is a decimal number from 1 to 56 inclusive, that specifies the number of bytes to be examined

length is the length attribute of the symbol specified as the first operand

Usage:

This request is used to examine the contents of specific locations in the user's virtual core, and causes contents to be typed at the terminal in hexadecimal form. The first operand of the request specifies the beginning address of the portion of core to be examined. This address is determined in the following way. If the operand contains any non-numeric characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the core address to which that symbol refers is used as the location of the first byte to be examined. If no match is found, or if the first operand contains only numeric characters, the current origin as established by the ORIGIN request is added to the specified operand and the resulting core address is used as the location of the first byte to be examined, provided that address is not greater than the user's virtual core size.

The second operand of X is optional. If specified, it indicates the number of bytes--up to a maximum of 56--whose contents are to be typed. If the second operand is omitted, a default value of 4 bytes is assumed unless the first operand is a symbol; if it is, the length attribute which is assigned to that symbol in the DEBUG symbol table is used as the number of bytes to be typed.

Responses:

If the X request is correctly issued, the information is typed and, following a carriage return, the keyboard is

unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that no or more than two operands have been specified in the X request. Reissue the request in its correct format.

INVALID ARGUMENT
This message is given when the first operand cannot be located in the DEBUG symbol table and does not constitute a valid hexadecimal number or the second operand is not a decimal number between 1 and 56 inclusive. If the first operand is intended to be a symbol, it must have been defined in a previous DEF request; otherwise, the operand must specify a valid hexadecimal number.

INVALID CORE-ADDRESS
The hexadecimal number specified in the first operand, when added to the current origin, is greater than the core size of the machine being used. If the current origin value is unknown, reset it to the desired value by issuing ORIGIN and reissue the X request.

Examples:

a. X XYZ
The contents of core starting at the address to which symbol XYZ is assigned are typed at the terminal. The number of bytes typed is determined by the length attribute of symbol XYZ as established in the DEF request for that symbol. See Figure 22.

b. X OTHER 12
Twelve bytes of core are typed beginning with the core address to which symbol OTHER has been assigned in a previous DEF request. See Figure 22.

c. X 123
Since no byte count is specified, four bytes of core are typed starting at the core address that is the sum of the current origin value and the hexadecimal number 123. See Figure 22.

d. X 123 32
Thirty-two bytes of core are typed, starting at the address that is the sum of the current origin value and the hexadecimal number 123. See Figure 22.

```
x xyz
00CCD501


def other 120

x other 12
0670897000031A7150170004


x 123
7000031A


x 123 32
7000031A715017000492EE7004D2037000C31C5910C35447B0C0E25010C35458
```

Figure 22. Examples of the X request

Purpose:

The SETERR command is used to trace transfers to and from all SVC-called programs in which error conditions occur.

Format:

```
---------------
|    SETERR    |
---------------
```

Usage:

The SETERR command sets error overrides that cause trace information to be recorded for all SVC-called programs that return with an error code in general-purpose register 15. The following information is recorded.

(1) A basic line consisting of the core locations of the SVC instruction and of the program which it called, the contents of the SVC old program status word, and the core locations to which the SVC-called program would return under both normal and error conditions.

(2) The contents of all general-purpose registers, both before the SVC-called program is given control and after it issues a return.

(3) The contents of all floating-point registers both before the SVC-called program is given control and after it issues a return.

(4) Two lines--16 words--of the parameter list which exists at the time the SVC is executed. See Figure 23 for a sample of the type of information recorded by the SETERR command, and the format in which this information is printed on the offline printer.

The SETERR command causes the above trace information to be recorded for all SVC-called programs that return error codes. This information is not printed until the user issues a CLROVER or a KO command, a RESTART request in the Debug environment, or logs out from the Control Program.

Traces initiated by the SETERR command may be terminated by issuing a CLROVER or a KO command. Both CLROVER and KO cause the trace information recorded up to the time they are issued to be printed on the offline printer.

Notes:

a. Issuing one or more SETERR commands when a SETOVER command is active has no effect, since error as well as normal overrides are set by SETOVER.

b. Issuing more than one SETERR command has no additional effect other than causing the heading given under Output to appear in the trace information printout each time a SETERR is issued.

c. Any operands given in the SETERR command are ignored.

Responses:

None.

Output:

SETTING ERROR-OVERRIDE TO PROVIDE A DYNAMIC
TRACE OF CMS (AND OS) SVC-CALLS . . .
This message appears in the offline printout of the trace information at all points where SETERR commands have been issued.

Example:

SETERR
This causes information such as that given in Figure 23 to be recorded for all SVC-called programs which return an error code in general-purpose register 15.

Error Messages:

None.

```
SETTING ERROR-OVERRIDE TO PROVIDE A DYNAMIC TRACE OF CHS (AND OS) SVC-CALLS...

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2C5E3C5  0000139C  00000000  00005816  00009B10  E2C5E3C5  000050A8  0003E8B8
                  00009386  00012000  C0009508  CC000001  400057E8  000011B0  40009540  000011B0
    FPRS BEFORE = 00000000  00000000  00000000  00000000  00000000  0CCC0000  00000000  00000000
    GPRS AFTER  = E2C5E3C5  0000139C  0000139C  00C00000  00000778  000029DC  00000A20  00000048
                  00000004  00000100  C6C9D5C9  E2404040  000040A8  000C0AA0  000026CE  00000006
    FPRS AFTER  = 00000000  00000000  00000000  CCC00000  00000000  00000000  00000000  00000000
    PARM.-LIST =  --FINI--  --S   --  --*   --  --   --  --*   --  --   --  5C40FFFF  FFFEFFFD
                  FFFF0000  00000000  00000000  00000000  00000000  --OVER--  --RIDE--  --MODU--

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3  0000139C  00000000  00005816  00009B10  E2E3C1E3  000050A8  0003E8B8
                  00009386  00012000  00009508  00000001  400057E8  000C11B0  40C09540  000011B0
    FPRS BEFORE = 00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
    GPRS AFTER  = E2E3C1E3  0000139C  0000139C  00000000  00000778  0C0C29DC  00000A20  00000048
                  00000004  00000100  C6C9D5C9  E2404040  000040A8  0CCC0AA0  000026CE  00000006
    FPRS AFTER  = 00000000  00000000  00000000  00C00000  00000000  00000000  00000000  00000000
    PARM.-LIST =  --FINI--  --S   --  --*   --  --   --  --*   --  --   --  5C40FFFF  FFFEFFFD
                  FFFF0000  00000000  00000000  00000000  00000000  --TEST--  --   --  --FORT--

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA60009516   NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3  00009B10  00000000  00000000  00009B10  E2E3C1E3  000050A8  0003E8B8
                  00009386  00012000  C0009508  CC000001  40009378  800094C0  60009580  00000001
    FPRS BEFORE = 00000000  00000000  00000000  00000000  00000000  0CCC0C00  00000000  00000000
    GPRS AFTER  = E2E3C1E3  00009B10  00009B10  00000000  00000778  00002AB4  00000A20  00000048
                  00000004  00000100  E2E3C1E3  C5404040  00002BD0  00000AA0  000026CE  00000001
    FPRS AFTER  = 00000000  00000000  00000000  00000000  00000000  0CCC0C00  00C00000  00000000
    PARM.-LIST =  --STAT--  --E   --  --TEST--  --   --  --SYSI--  --B   --  FFFFFFFF  FFFFFFFF
                  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3  0000139C  C0000000  00C05816  00009B10  E2E3C1E3  000050A8  0003E8B8
                  00009386  00012000  00009508  00000001  400057E8  0C0C11B0  40009540  000011B0
    FPRS BEFORE = 00000000  00000000  00000000  00000000  0000000C  0CCC0C00  00C00000  00000000
    GPRS AFTER  = E2E3C1E3  0000139C  0000139C  00C00000  00000778  000029DC  00000A20  00000048
                  00000004  00000100  C6C9D5C9  E2404040  000040A8  CCCC0AA0  000026CE  00000006
    FPRS AFTER  = 00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
    PARM.-LIST =  --FINI--  --S   --  --*   --  --   --  --*   --  --   --  5C40FFFF  FFFEFFFD
                  FFFF0000  00000000  00000000  00000000  00000000  --TEST--  --   --  --FORT--

*****ERROR-OVERRIDE,   CALLER=000025C2   CALLEE=LOADMOD  SVC-OLD-PSW=000400CA400025C4   NRMRET=000025C8   ERRET=000025F8
    GPRS BEFORE = 00000000  000009E0  00009B10  00000000  00000778  000009E8  00000A20  00000048
                  00000004  00000100  D3404040  40404040  40009378  4CCC228E  C00007B8  00000000
    FPRS BEFORE = 00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
    GPRS AFTER  = 00000000  000009F4  00005838  00000000  00000778  00CC2A0C  00000AF0  00000044
                  00000004  00000100  000011B0  000026CE  00002F00  CCCCCE7C  000026CE  00000001
    FPRS AFTER  = 00000000  00000000  00000000  00C00000  00000000  00000000  00000000  00000000
    PARM.-LIST =  --LOAD--  --MOD --  --L   --  --   --  FF000000  --CPFU--  --NCTN--  00011000
                  00000000  00000000  00000000  CCC400CA  5000E692  00000000  0000EE58  080CE268

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=L         SVC-OLD-PSW=000400CA50009516   NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = 00000000  00009B10  00000000  00000000  00009B1C  D34C4C40  000050A8  0003E8B8
                  00009386  00012000  C0009508  CC000001  40009378  800094C0  40009500  00000001
    FPRS BEFORE = 00000000  00000000  00000C00  00000000  00000000  0CCC0C00  00C00000  00000000
    GPRS AFTER  = D3C9E2E3  000115E8  00011000  00C00000  00000028  000002F8  00000000  00000000
                  0000F138  00001BA8  D3C9E2E3  00011470  000011B0  00000002  000026CE  00000002
    FPRS AFTER  = 00000000  00000000  00000000  00000000  00000000  CCCCC00C  00C00000  00000000
    PARM.-LIST =  --L   --  --   --  --TEST--  --   --  --SYSI--  --B   --  FFFFFFFF  FFFFFFFF
                  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA4CCC57F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = 00000000  0000139C  C0000000  00C05816  00009B10  D3404040  000050A8  0003E8E8
                  00009386  00012000  00009508  00000001  400057E8  0CCC11B0  40009540  000011B0
    FPRS BEFORE = 00000000  00000000  00000000  03000000  00000000  0CCC0C00  00C00000  00000000
    GPRS AFTER  = 00000000  0000139C  0000139C  00C00000  00000778  000029DC  00000A20  00000048
                  C0000004  00000100  C6C9D5C9  E2404040  000040A8  0CCCCAA0  000026CE  00000006
    FPRS AFTER  = 00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
    PARM.-LIST =  --FINI--  --S   --  --*   --  --   --  --*   --  --   --  5C40FFFF  FFFEFFFD
                  FFFF0000  00000000  00000000  00000000  0000000C  --LIST--  --F   --  --MODU--

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
```

Figure 23. Sample offline printout of trace information recorded by the SETERR command

SETOVER

Purpose:

The SETOVER command is used to trace transfers to and from SVC-called programs which are executed normally as well as those in which error conditions are encountered.

Format:

```
|SETOVER|  <SAMELAST>    <option1...optionN>|
```

SAMELAST leaves all options as set by the user's last SETOVER command provided a CLROVER has not been issued. If SAMELAST is not specified or if CLROVER has been issued, options are reset to their default settings. Any options specified after SAMELAST replace these settings.

option1...optionN are one or more of the options given below.

Options:

GPRS record the contents of the general-purpose registers both before the SVC-called program is given control and after a return from that program

GPRSB record the contents of the general-purpose registers only as they exist before the SVC-called program is given control

GPRSA record the contents of the general-purpose registers only as they exist after a return from the SVC-called program

The default option is that no general-purpose register information is recorded.

FPRS record the contents of the floating-point registers both before the SVC-called program is given control and after a return from that program.

FPRSB record the contents of the floating-point registers only as they exist before the SVC-called program is given control

FPRSA record the contents of the floating-point registers as they exist after a return from the SVC-called program

The default option is that no floating-point register information is recorded.

NOPARM no parameter list information is to be recorded when a program is called by an SVC instruction

PARM1 record one line (8 words) of parameter list information when a program is called by an SVC instruction

The default option is two lines (16 words) of the parameter list recorded when a program is called by an SVC instruction

NOWAIT no information is to be recorded when WAIT is called by an SVC

WAITSAME record the same information for GPR's, FPR's, and parameter lists when WAIT is called as that recorded for all other SVC-called programs

WAIT1 record one line (8 words) of the parameter list when WAIT is called

WAIT2 record two lines of the parameter list when WAIT is called

The default option is that no GPR, FPR, or parameter list information recorded when WAIT is called

DEFAULT cancel all current settings and reset the options to their default settings

Usage:

The SETOVER command traces all internal branches which take place due to SVC, or supervisor call, instructions. The effect of the SETOVER command is to set overrides which cause information to be recorded at the appropriate times. Both normal and error overrides are set by the SETOVER command.

The information recorded will vary, depending on the type of override. For both normal and error overrides, the core location of the calling SVC instruction and the name of the called program or routine are recorded, as well as the contents of the SVC old program status word (that is, that which was stored when the SVC was issued) and the core locations of the normal and error returns from the called program. In addition to this basic line, error overrides record the contents of the general-purpose and floating-point registers before branching to the SVC-called program and returning from it, and 16 words of the parameter list which existed when the SVC was issued. For normal overrides, the additional information recorded depends upon the options specified by the user in the SETOVER command. If no options are specified, the default options are used: no general-purpose or floating-point register information, and 16 words of the parameter list are recorded for all routines except WAIT. For WAIT only the basic line of information, common to both normal and error overrides, is recorded. When the user does specify options, the default settings are assumed initially, and options specified by the user replace the appropriate default settings. If the user specifies SAMELAST as his first option, the options remain as they were set by the last SETOVER command issued provided no CLROVER command has been issued, and any further options given in the SETOVER command replace these previously set options.

It is possible to issue two or more SETOVER commands before

issuing a KO or a CLROVER command. When additional SETOVER commands are issued, the option settings are adjusted to reflect those specified in the most recent SETOVER command. If SAMELAST is not specified as the first option in these additional commands, the options are reset to their default settings, and only those options specified by the user in the new SETOVER command replace the default settings.

To terminate overrides set by the SETOVER command, the user may issue a KO or a CLROVER command. Both CLROVER and KO cause all trace information recorded up to the point they are issued to be printed on the offline printer. CLROVER can be issued only when the keyboard is unlocked to accept input to the CMS Command environment. To clear overrides at any other point in system processing, KO must be issued. If a user issues a RESTART request to the Debug environment or logs out from the Control Program prior to clearing overrides set by SETOVER and/or SETERR, the overrides are cleared automatcially and all recorded trace information is printed on the offline printer.

Notes:

a. If SAMELAST is specified, it must be the first option given and there must be no intervening CLROVER between the two SETOVER commands.

b. If mutually exclusive options are specified, such as NOPARM and PARM1, the last such option appearing in the SETOVER command is used. For options which are not mutually exclusive--GPRS and GPRSA, for example--the logical combination of the options is used. In this case, the option set by GPRS would be used since it includes the GPRSA option.

c. The number of lines of parameter list information recorded for calls to the WAIT routine cannot exceed that recorded for other routines, (that is, the PARM setting governs the WAIT setting when the latter specifies the greater number of lines to be recorded.

Responses:

None.

Output:

SETTING NORMAL - AND ERROR-OVERRIDES TO PROVIDE
A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS. . .
This message appears in the offline printout of trace information at all points where SETOVER commands are issued.

Examples:

a. SETOVER
In addition to the basic information recorded for both

normal and error overrides, this example causes the default information (no general-purpose or floating-point register information and two lines of the parameter list) to be recorded for all normal overrides. For error overrides, the contents of all general-purpose and floating-point registers are recorded both before branching to the SVC-called program and after returning from it, as are two lines of the parameter list information. A sample of the type of information which is recorded is given in Figure 15.

b. SETOVER GPRSB FPRSA NOPARM
In addition to the basic information, this example causes the contents of the general-purpose registers to be recorded before the SVC-called program is given control, and the contents of the floating-point registers to be recorded after a return is issued by that program for all normal overrides. No parameter list information is recorded. For calls to the WAIT routine, only the basic line of information is recorded. The same information is recorded for error overrides as that given in Example a. See Figure 24 for a sample of the type of information which is recorded for this example and the format in which it is printed.

c. SETOVER SAMELAST PARM1 WAIT2
Assume Example b had been issued prior to this example and no CLROVER had been issued between them. Since SAMELAST is specified as the first option, the settings for Example b would be assumed initially. The NOPARM setting, however, is replaced by the PARM1 option specified above. WAIT2 would normally replace the default option of recording no parameter list information for calls to the WAIT routine, but since PARM1 has been specified, only one line of the parameter list for calls to WAIT is recorded. The same information is recorded for error overrides as that given in Example a. Refer to Figure 25 for a sample of the type of information which is recorded by this example and the format in which it is printed.

Error Messages:

E(00001)
The user has specified one or more invalid options. Reissue the SETOVER command making sure that all options are spelled correctly and that SAMELAST, if specified, is the first option given.

```
SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=SETOVER  SVC-OLD-PSW=000400CA60009516   NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2C5E3D6   00009B10   00000000   00009B10   E2C5E3D6   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   40009378   8CCC94C0   60C09580   00000001
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --SETO--   --VER --   --SAME--   --LAST--   --PARM--   --1  --   --WAIT--   --2  --

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2C5E3D6   0000139C   00000000   00005816   00009B10   E2C5E3D6   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   400057E8   0CCC11B0   40009540   000011B0
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2C5E3D6   0000139C   0000139C   00000000   00000778   000029DC   00000A20   00000048
                  00000004   00000100   C6C9D5C9   E2404040   000040A8   0CCCCAA0   C00026CE   00000006
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --FINI--   --S  --   --*  --   --  --   --*  --   --  --   5C40FFFF   FFFEFFFD
                  FFFF0000   00000000   00000000   00000000   00000000   --STAT--   --  --   --MODU--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA60CC955C   NRMRET=00009560   ERRET=0000954A
    GPRS BEFORE = 00000004   000095FC   00000000   00000000   00009B10   E2C5E3D6   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   40009378   8CCC954E   40C096C2   0000960F
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --TYPL--   --IN  --   0100960C   C200001A   D95E40E3   7EFC4EF0   F461FC4B   --11 2--

    NORMAL-OVERRIDE,   CALLER=0000CB4E   CALLEE=ATTN     SVC-OLD-PSW=000400CA6CCCCB5C   NRMRET=0C00CB50   ERRET=0000A238
    GPRS BEFORE = 00000000   0000CB58   00000013   01CC9774   C000023E   00000012   00000251   0000023E
                  CC0001F8   00000009   000095DC   000001E8   0000C658   0000C658   00000001   00000000
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   CCCCCCC0   C0C00C0C   00000000
    PARM.-LIST =  --ATTN--   --   --   --LIFO--   12000251   --NULL--   --   --   000001E8   0000C4CC

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA400096DA   NRMRET=000096DE   ERRET=C0CC96DE
    GPRS BEFORE = CC000000   000095DC   00000000   00009456   CCCCCCC0   C0C050A8   0003E8E8
                  00009386   00012000   C0009508   CCC00001   40009378   800094C0   00C0CEE0   000032F8
    FPRS AFTER  = CC000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --WAIT--   --RD  --   01009774   E4000012   --TYPL--   --IN  --   01009366   D2000020

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA6CCC9516   NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3   00009B10   CCC00000   00009B10   E2E3C1E3   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   40009378   800094C0   60009580   00000001
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   CCCCCCC0   CCC00000   00000000
    PARM.-LIST =  --STAT--   --E   --   --TEST--   --   --   --FORT--   --E2N --   FFFFFFFF   000C13D0

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4   NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3   0000139C   00000000   00005816   00009B1C   E2E3C1E3   C0C050A8   0003E8E8
                  00009386   00012C00   C0C09508   CCC00001   400057E8   000011B0   40009540   000011B0
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2E3C1E3   0000139C   0000139C   00000000   00000778   00CC29DC   C0000A2C   00000048
                  00000004   00000100   C6C9D5C9   E2404040   000040A8   00000AA0   0000026E   00000006
    FPRS AFTER  = CCCC0000   00000000   00000000   00000000   000000CC   0CCCCC00   C0C0C000   00000000
    PARM.-LIST =  --FINI--   --S  --   --*  --   --  --   --*  --   --  --   5C40FFFF   FFFEFFFD
                  FFFF0000   C0000000   00000000   CCC00000   00000000   --TEST--   --  --   --FORT--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA6000955C   NRMRET=00009560   ERRET=0C00954A
    GPRS BEFORE = 00000004   000095FC   00000000   00000000   00009B1C   E2E3C1E3   C0C05CA8   0003E8B8
                  00009386   00012000   C0009508   CCC00001   40009378   8000954E   400096C2   0000960F
    FPRS AFTER  = CC000000   00000000   00000000   00000000   00000000   CCCCCC0C   00C00C0C   00000000
    PARM.-LIST =  --TYPL--   --IN  --   0100960C   C2C0001A   D95E40E3   7EF04EF0   F461F04E   --09 2--

    NORMAL-OVERRIDE,   CALLER=0000CB4E   CALLEE=ATTN     SVC-OLD-PSW=000400CA6000CE50   NRMRET=0000CB50   ERRET=0000A238
    GPRS BEFORE = CC000011   0000CB58   00000002   01009774   0000023E   00000010   0000024F   00000048
                  00000007   C0000009   000095DC   00000 1E8   0000C65E   0CCCC65E   CCC00C01   000032F8
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --ATTN--   --   --   --LIFO--   01000 24F   --NULL--   --   --   C0C0C1E8   00000400

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA4CCC96DA   NRMRET=0C0096DE   ERRET=000096DE
    GPRS BEFORE = C0000000   000095DC   00000000   CCC00000   00009458   00000000   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   40009378   8CCC94C0   C000CEE0   000032F8
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   0CCCCC0C   00000000   00000000
    PARM.-LIST =  --WAIT--   --RD  --   01009774   E4000010   --TYPL--   --IN  --   01009386   D2000020

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA60009516   NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3   00009B10   00000000   00000000   00009B1C   E2E3C1E3   C0C050A8   0003E8B8
                  00009386   00012C00   C0009508   CCC00001   40009378   800094C0   60009580   00000001
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2E3C1E3   00009B10   00009B10   00000000   00000778   00CCC2A84   C0C00A2C   00000048
                  00000004   00000100   E2E3C1E3   C5404040   00002BD0   C0000AA0   0000026E   00000C01
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   C0000000   00000000   00000000
    PARM.-LIST =  --STAT--   --E   --   --TEST--   --   --   --SYSI--   --N   --   FFFFFFFF   FFFFFFFF
                  FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF
```

Figure 24. Offline printout showing trace information recorded by the SETOVER command with the options GPRSB, FPRSA, and NOPARM specified

```
SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=SETOVER  SVC-OLD-PSW=000400CA6CCC9516  NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2C5E3D6   00009B10   00000000   00000000   00009B10   E2C5E3D6   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   4000937E   8CCC94C0   60009580   00000001
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4  NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2C5E3D6   0000139C   00000000   00005816   00009B10   E2C5E3D6   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   400057EE   0CCC11B0   40C0954C   000011B0
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2C5E3D6   0000139C   0000139C   00000000   0000077E   00CC29DC   C0000A20   00000048
                  00000004   00000100   C6C9D5C9   E2404040   000040A8   CCCCCAA0   000026CE   00000006
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --FINI--   --S   --   --*   --   --   --   --*   --   --   --   5C40FFFF   FFFEFFFD
                  FFFF0000   00000000   00000000   00000000   00000000   --CVEE--   --RIDE--   --MOLU--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA6000955C  NRMRET=00009560   ERRET=00CC954A
    GPRS BEFORE = 00000004   000095FC   00000000   00009B10   E2C5E3D6   000050A8   0003E8B8
                  00009386   00012000   00009508   00000001   40009378   8000954E   400096C2   0000960F
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C0C   00C00000   00000000

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA4CCC96DA  NRMRET=000096DE   ERRET=000096DE
    GPRS BEFORE = 00000000   000095DC   00000000   00000000   00009458   00000000   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   4000937E   8CCC94C0   C000CEE0   000032F8
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C0C   00C00000   00000000

    NORMAL-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA6CCC9516  NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3   00009B10   00000000   00000000   00009B10   E2E3C1E3   000050A8   0003E8B8
                  00009386   00012000   00009508   00000001   40009378   8CCC94C0   60009580   00000001
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   00000000   00000000   00000000

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4  NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3   0000139C   00000000   00005816   00009B10   E2E3C1E3   C0CC50A8   0003E8B8
                  00009386   00012000   00009508   00000001   400057E8   00001180   40009540   000011E0
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2E3C1E3   0000139C   0000139C   00000000   0000077E   00CC29DC   C0C00A20   00000048
                  00000004   00000100   C6C9D5C9   E2404040   000040A8   CCCCCAA0   000026CE   00000006
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C00   C0C00000   00000000
    PARM.-LIST =  --FINI--   --S   --   --*   --   --   --   --*   --   --   --   5C40FFFF   FFFEFFFD
                  FFFF0000   00000000   00000000   00000000   00000000   --TEST--   --   --   --PORT--

    NORMAL-OVERRIDE,   CALLER=0000955A   CALLEE=TYPLIN   SVC-OLD-PSW=000400CA6CCC955C  NRMRET=00009560   ERRET=00CC954A
    GPRS BEFORE = 00000004   000095FC   00000000   00009B10   E2E3C1E3   00005CA8   0003E8B8
                  00009386   00012000   00009508   00000001   40009378   8000954E   400096C2   0000960F
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C0C   00C00000   00000000

    NORMAL-OVERRIDE,   CALLER=000096D8   CALLEE=WAITRD   SVC-OLD-PSW=000400CA4CCC96DA  NRMRET=000096DE   ERRET=000096DE
    GPRS BEFORE = 00000000   000095DC   00000000   00000000   00009458   00000000   000050A8   0003E8E8
                  00009386   00012000   00009508   00000001   4000937E   8CCC94C0   C000CEE0   000032F8
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C0C   00C00000   00000000

*****ERROR-OVERRIDE,   CALLER=00009514   CALLEE=STATE    SVC-OLD-PSW=000400CA6CCC9516  NRMRET=0000951A   ERRET=0000951A
    GPRS BEFORE = E2E3C1E3   00009B10   00000000   00000000   00009B10   E2E3C1E3   000050A8   0003E8B8
                  00009386   00012000   00009508   00000001   4000937E   8CCC94C0   60009580   00000001
    FPRS BEFORE = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    GPRS AFTER  = E2E3C1E3   00009B10   00009B10   00000000   0000077E   00CC2A84   C0000A20   00000048
                  00000004   00000100   E2E3C1E3   C5404040   00002BDC   CCCCCAA0   000026CE   00000001
    FPRS AFTER  = 00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
    PARM.-LIST =  --STAT--   --E   --   --TEST--   --   --   --SYSI--   --N   --   FFFFFFFF   FFFFFFFF
                  FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=000057F2   CALLEE=FINIS    SVC-OLD-PSW=000400CA400057F4  NRMRET=000057F8   ERRET=000057F8
    GPRS BEFORE = E2E3C1E3   0000139C   00000000   00005816   00009B10   E2E3C1E3   C0CC50A8   0003E8B8
                  00009386   00012000   00009508   00000001   400057E8   00001180   40009540   000011E0
    FPRS BEFORE = 00000000   00000000   00000000   00000000   0000000C   0CCC0C00   00C00000   00000000
    GPRS AFTER  = E2F3C1E3   0000139C   0000139C   00000000   00000778   000029DC   00000A20   00000048
                  CC000004   00000100   C6C9D5C9   E2404040   000040A8   CCCCCAA0   000026CE   00000006
    FPRS AFTER  = 00000000   00000000   00000000   00000000   0000000C   0CCC0C00   00C00000   00000000
    PARM.-LIST =  --FINI--   --S   --   --*   --   --   --   --*   --   --   --   5C40FFFF   FFFEFFFD
                  FFFF0000   00000000   00000000   00000000   00000000   --TEST--   --   --   --PORT--
```

Figure 25. Offline printout showing trace information recorded by the SETOVER command with the options SAMELAST, PARM1, and WAIT2 specified (SAMELAST in this example refers to the options as set in Figure 24)

LANGUAGE PROCESSORS

The language processors supported in CMS are the same ones used under Operating System/360 (OS); these include Assembler (F), Fortran IV (G), and PL/I (F).

The language processors in CMS and OS are compatible at the source language level as long as the macros and SVC's used in an Assembler program are supported by CMS. The object modules (text decks) that are output from the above Operating System compilers may be executed under either CMS or OS as long as the previous restrictions are adhered to.

There are two additional processors available as Type III programs from the IBM Program Information Department; they are SNOBOL, a string processing language, and BRUIN, an interpretive language. BRUIN, Brown University Interpreter, was adapted from the OS version of BRUIN developed at Brown University, Providence, Rhode Island. BRUIN provides two modes of operation: a desk calculator mode and a stored program mode.

Purpose:

The ASSEMBLE command creates relocatable object programs from programs written in System/360 Assembler Language.

Format:

```
| ASSEMBLE |filename1 <...filenameN><(option1...optionN>) |
|    A     |                                               |
```

filename specifies a SYSIN file to be assembled. Additional filenames specify additional assemblies.

option is one or more of the assembler options listed below.

Options:

DECK creates a TEXT file of the relocatable object program
NODECK suppresses the TEXT file

LIST creates a LISTING file of the assembled program
NOLIST suppresses the LISTING file and online diagnostics

XREF includes a cross-reference symbol table in the LISTING file
NOXREF suppresses the cross-reference symbol table

DIAG types source statements containing errors at the terminal, along with diagnostic and error messages
NODIAG suppresses typing of errors.

LTAPn writes the LISTING file on the tape whose symbolic address is TAPn
LDISK writes the LISTING file on the permanent disk
PRINT writes the LISTING file on the offline printer

RENT checks the program for reentrance
NORENT suppresses the reentrance check

Usage:

The filetype SYSIN is assumed for all input files to the ASSEMBLE command. The file must have fixed-length, 80-character records. More than one assembly may be performed by specifying additional filenames, separated by blanks. Any number of files may be specified, but the command must not exceed a single input line. Each file

named is assembled separately in the order named.

Assembler output is controlled by a set of options. The list of options selected, enclosed in a set of parentheses, follows the last, or only, filename specified with the command. One set of options governs all assemblies performed by one ASSEMBLE command. The options, specified in any order, are separated by at least one blank. A default value is supplied for any option not included. The default option values are:

DECK LIST XREF DIAG NORENT LDISK

Any combination of option values may be specified, but if NOLIST is included, XREF, DIAG, LDISK, PRINT, and LTAPn are ignored.

During an assembly, three work files are created, with the filetypes SYSUT1, SYSUT2, and SYSUT3. Their filenames are the same as the SYSIN file being assembled. According to the options specified, files with filetypes LISTING and TEXT may also be created, with the same filename. At the beginning of each assembly, any pre-existing files with any of the above filetypes and the current filename are deleted, even if the set of options specified means they are not to be replaced by the current assembly. To save old copies of LISTING and TEXT files, use ALTER to change their filenames or the filename of the SYSIN file before assembly. Insufficient space on the permanent disk for any of the assembler files causes termination with an I/O error message.

filename TEXT P1 is the file of machine-language code created by the assembler. This file can be loaded for execution with the LOAD or $ commands, or punched in object deck form on the cardpunch with the OFFLINE PUNCH command. If NODECK is specified, this file is not created.

filename LISTING P1 is the file of source statements and assembled machine code produced by the assembler. This file is not created if NOLIST is specified. An external symbol directory and a cross-reference symbol table are included, unless NOXREF is specified. Diagnostics and error messages appear at the bottom of the LISTING file, and, unless NODIAG is specified, are typed at the terminal.

If PRINT is specified, the LISTING file is printed on the offline printer. If LTAPn is specified, the LISTING file is written on the tape whose symbolic address is TAPn in blocks of ten 121-character records. If LDISK is specified, or no value is specified, the LISTING file is written on the permanent disk.

The SYSUT1, SYSUT2, and SYSUT3 files created by the assembler and used as work files are deleted at the end of each assembly. If they are not deleted because of an abnormal termination, they may be deleted with the ERASE

command, or by reissuing the ASSEMBLE command.

The assembler searches the system macro libraries (SYSLIB MACLIB SY and OSMACRO MACLIB SY) for macro definitions. Names of macros in these libraries may be obtained with the MACLIB LIST command. Additional macro libraries may be created on the permanent disk with the MACLIB GEN command, and up to five of these additional libraries may be included in the assembler search list at one time with the GLOBAL ASSEMBLER MACLIB command. The assembler accepts the first definition for a macro it finds, allowing the user to override system macro definitions. If the user has a file called SYSLIB MACLIB or OSMACRO MACLIB on a disk that precedes the system disk in the order of search, that library is searched in place of the system library. See GLOBAL under "Execution Control" and MACLIB under "Libraries" for further information.

ASSEMBLE uses the standard order of search to locate the SYSIN files.

Reponses:

a. ASSEMBLING: filename
This response is typed for the second and subsequent assemblies performed by a single command. It separates diagnostics for the assemblies.

b. SYMBOLIC TAPE ADDRESS INCORRECT
LISTING FILE WILL BE WRITTEN ON DISK. WRITING ON TAPE IS CANCELLED.
The option LTAPn was specified, and TAPn was not a valid symbolic tape address. The LISTING file is written on the permanent disk.

c. OUTPUT TAPE FULL. CHANGE IT AND HIT CARRIAGE RETURN.
An end of reel condition was detected on the tape unit for the LISTING file. The tape has been rewound. Press ATTN to enter the CP environment, and ask the operator to mount a new tape. When the operator replies that a new tape is mounted, return to CMS with ATTN, and issue a carriage return. The assembly resumes where it was interrupted.

d. READY THE TAPE UNIT AND HIT CARRIAGE RETURN.
The tape unit specified by LTAPn signalled not ready. Go to CP with ATTN and ask the operator to ready the unit. When he replies that the unit is ready, return to CMS with ATTN, and issue a carriage return. The assembly begins or resumes.

e. PERMANENT I/O ERROR ON TAPE
LISTING FILE WILL BE WRITTEN ON DISK. WRITING ON TAPE IS CANCELLED.
All or part of the assembly listing is being written on disk as filename LISTING P5. If any of the records were successfully written on the specified tape before the error,

they are missing from the disk LISTING file.

f. PLEASE READY THE PRINTER
This response should never occur under CP. Notify the responsible system programmer.

g. PERMANENT I/O ERROR ON THE PRINTER, LISTING FILE WILL BE WRITTEN ON DISK.
This response should never occur under CP. Notify the responsible system programmer.

h. PERMANENT I/O ERROR ON DISK, ASSEMBLY CONTINUES WITHOUT WRITING LISTING FILE ON DISK.
The assembly is completing without any LISTING file. If a listing is necessary, execution may be cancelled with the KX command. The error may have resulted from insufficient space on the permanent disk. Use the ERASE command to create more free space and try the assembly again. If the error recurs, notify the operator.

Note:

The error completion code returned on completion of the ASSEMBLE command is the highest severity code returned by the assembler for any of the assemblies performed by that command.

References:

The System/360 instruction set is described in System/360 Principles of Operation. The assembler instructions and macro language are described in IBM System/360 Operating System Assembler Language. Additional information on assembler execution and messages may be found in IBM System/360 Operating System Assembler (F) Programmer's Guide. Note that the execution options in the Programmer's Guide are different than those supported by CMS.

Examples:

a. ASSEMBLE RETURN
The file RETURN SYSIN P1 is assembled. Since no options are specified, the default options govern the assemble. The following files are created:

```
RETURN LISTING P1
RETURN TEXT    P1
```

See Figure 24 for an example showing the on-line diagnostics generated by the assembler.

b. ASSEMBLE RETURN JOBA TEST1 (RENT NOXREF NODIAG)
The file RETURN SYSIN P1 is assembled, and the resulting object code is checked for reenterability. The listing file is printed, and is not saved on disk. The listing does not contain a cross-reference symbol table. On-line diagnostics are suppressed. RETURN TEXT P1 is created on the permanent

disk. When the assembly is complete, the same operations are performed for JOBA SYSIN P5, and then for TEST1 SYSIN P5.

Error Messages:

E(00001) FILE(S) TO ASSEMBLE UNDEFINED.
No filenames were specified with the ASSEMBLE command.

E(00001) AT LEAST ONE OF THE FILES TO ASSEMBLE DOESN'T EXIST OR DOESN'T HAVE A CORRECT 'TYPE' NAME.
Assemblies were started.

E(00001) AT LEAST ONE OF THE FILES TO ASSEMBLE HAS INCORRECT RECORD LENGTH.
Sysin files must have fixed-length, 80-character records.

E(00004)
Minor errors were detected during the assembly, but successful execution of the program is still probable.

E(00008)
Errors were detected in the assembled program, but execution may still be possible.

E(00012)
Serious errors were detected in the assembled program. Successful execution is not probable.

E(00016)
Very serious errors were detected in the assembled program. Execution is impossible.

E(00020) PERMANENT I/O ERROR WHILE READING SYSIN FILE filename UNABLE TO ASSEMBLE ALL THE FILE.
Files specified before the one named in the message have been assembled. The file named and subsequent ones have not been assembled. The file in which the error occurred must be reentered before assembling it.

E(00020)
The assembler detected a catastrophic error such that it could not continue processing. This may be an I/O error, caused by insufficient free space on the permanent disk. Free some space with the ERASE command, and retry the assembly. If the error recurs, notify the operator.

ASSEMBLER LANGUAGE PROGRAMMING

PROGRAM NAMING

The normal entry point name of a program should be the same as the filename of the TEXT file. The program may then be executed by the command

```
$ filename
```

If the filename and entry point name are different, the following sequence must be used:

```
LOAD filename
START entryname
```

PROGRAM ENTRY

When control is received by a user program, the address of the entry point is in register 15, which may be used for immediate addressability. Register 14 contains a return address into the CMS nucleus, and must be saved. Register 13 contains the address of an 18-word save area. Register 1 points to a parameter list, which contains any parameters passed to the program by $ or START. The parameter list is aligned on a double word boundary, and each entry is found in the high-order bytes of successive double words. All data is in EBCDIC. The first entry is always the entry point name of the program being executed. The following entries are the parameters passed to the program. For instance, after the command

```
$ JOB10 5/7/68  21.37
```

register 1 points to a parameter list in the following format.

```
PLIST    DS     0D
         DC     CL8'JOB10'
         DC     CL8'5/7/68'
         DC     CL8'21.37'
         DC     8X'FF'
```

The last entry is always a double word with all bits set to one which serves as a delimiter. Any parameters longer than eight characters are truncated to the eight high-order characters.

PROGRAM EXIT

Return should always be to the address received in register 14. This gives control to CMS service routines which close files, update the user's disk file directory, and calculate and type the time used in execution. CMS also inspects register 15 for an error code. If none is found, the completion message has the form "R; T=n.nn/x.xx xx/xx/xx". If there is a value in register 15, the message is "E(nnnnn); T=n.nn/x.xx xx/xx/xx" where nnnnn is the error code returned in register 15 and n.nn is the CMS CPU time in seconds used for execution, x.xx is the CP and CMS CPU time,

and xx/xx/xx is the time of day in hours/minutes/seconds.

LINKAGE TO CMS COMMANDS AND ROUTINES

With few exeptions, all CMS linkages are made with one supervisor call: SVC X'CA'. The address of a parameter list is placed in register 1 before the call, and the first entry of the list always specifies the CMS command or function being called. All registers are saved and restored by the SVC-handling service routine, except register 15, which is used as an error return register.

The parameter list is always aligned on a double word boundary. If a command is called, the same parameters that would be typed to call the command are placed in successive double words. For instance, a parameter list for ERASE might appear as

```
PLIST   DS  0D
        DC  CL8'ERASE'
        DC  CL8'*'
        DC  CL8'WORKFILE'
```

In this case all files with the filetype WORKFILE are erased during execution of the program by the sequence

```
        LA  1,PLIST
        SVC X'CA'
```

After the files are deleted, CMS returns control to the next instruction after the SVC. Register 15 is set to zero to indicate that no error occurred. Should an error occur, a response is typed and control passes to DEBUG. Parameter lists for CMS routines which are not commands, such as RDBUF or WRBUF, are not uniform. Complete explanations of these parameter lists are found in "CMS Nucleus Routines" later in this section and in the CMS Program Logic Manual.

To avoid going to DEBUG when an error occurs in a called program, the programmer may specify an error return address with each SVC. The address is placed in the four bytes immediately after the SVC. Control goes to the address specified on any error in the called program. For example

```
        LA   1,PIST
        SVC  X'CA'
        DC   AL4(ERROUT)
        LH   3,26(1)
```

Note that the address constant must include a length specification to prevent alignment by the assembler. In this example, if the called program completes normally, control is returned at the LH instruction. If any error occurs, control goes to ERROUT. Errors are ignored by the sequence

```
SVC   X'CA'
DC    AL4(*+4)
```

Linkage Notes:

a. Commands that are not resident in the CMS nucleus may also be called by the CMS SVC, but they are loaded at hexadecimal location 12000. This is also the default load point for user programs. Therefore, a higher load point must be specified when the user's program is loaded. A complete list of disk resident commands may be obtained with a LISTF * MODULE SY, which also gives a rough idea of the command size, expressed in 800-byte records.

b. A few CMS routines may be executed only by branching. The addresses of these programs are listed in a nucleus module under the entry point SYSREF. SYSREF is resolved as an EXTRN in any user program. The displacement from SYSREF to the desired address is obtained with the CMSYSREF macro instruction, which generates a series of EQU statements. The name of each nucleus routine, with a D prefixed to it, is equated to its displacement from SYSREF in the address list. The following example shows how to get to the SCAN routine in the nucleus.

```
USERJOB   CSECT
          EXTRN SYSREF
          .
          L     3,=A(SYSREF)
          L     15,DSCAN(3)
          BALR  14,15
          LTR   15,15
          BNZ   ERR$
          .
          CMSYSREF
          .
```

In the expansion of CMSYSREF, DSCAN is equated to the displacement of the SCAN address in the SYSREF list. The address is loaded into register 15 for the BALR. SCAN is a routine to break up terminal line images in core into a standard CMS parameter list. For further information on SCAN, see the CMS Program Logic Manual.

The macro definitions that are used by the Cambridge Monitor System are contained in the files SYSLIB MACLIB and OSMACRO MACLIB which reside on the system disk. SYSLIB MACLIB contains the CMS macros which provide linkages to the CMS I/O routines, and OSMACRO MACLIB contains the System/360 Operating System macros which have been changed to interface with CMS. Only the CMS macros are discussed in this section. For a discussion of the OS macros supported, see "OS Macros". To obtain a list of the names, size, and location of macro definitions in libname MACLIB, the command MACLIB LIST libname may be issued. To print out the actual macro definitions, refer to the procedure described in MACLIB under "Libraries".

The CMS macros described in this section deal primarily with linkage to the CMS disk and terminal handling routines. TYPE and TYPIN handle terminal I/O, and FCB, STATE, SETUP, RDBUF, WRBUF, CKEOF, ERASE and FINIS handle I/O to the permanent disk. The offline unit record devices may be accessed from an assembler language program by calling the OFFLINE command, as explained previously under "Linkage to CMS Commands and Routines".

The TYPE and TYPIN macros each set up a parameter list in line and issue a CMS supervisor call. For disk I/O, the parameter list is set up in a constant area by the FCB (File Control Block) macro, and the label of that macro is used as a parameter of the WRBUF, RDBUF, SETUP and STATE macros which issue CMS SVC instructions for linkage. If an existing file is to be read, STATE and SETUP must be executed before the first RDBUF to initialize the first File Control Block. Figure 26 shows a typical sequence of macros for writing and reading disk files.

Notes:

a. The TYPE and TYPIN macros generate parameter lists in line with executing code. If either macro is to be used repeatedly, it may be more efficient to set up a single parameter list and issue CMS SVC's to call the routines directly. See "Linkage to CMS Commands and Routines".

b. CMS closes user files on program completion.

c. The CMS macros discussed in this section are in alphabetical order.

```
STMT SOURCE STATEMENT
   1 BEGIN  CSECT
   2        PRINT NOGEN
   3        BALR  12,0    ESTABLISH ADDRESSABILITY
   4        USING *,12
   5        STATE SINPUT,ERR1    INITIALIZE INPUT
   9        SETUP INPUT           FILE CONTROL BLOCK
  15 RD     RDBUF INPUT,ERROR=EOF      READ A RECORD
  20        WRBUF OUTPUT,ERROR=ERR2    WRITE SAME
  25        B     RD                REPEAT TO END
  26 EOF    CKEOF ERR3             REALLY EOF?
  29        SR    15,15    YES,CLEAR ERROR RETURN
  30 DONE   BR    14       RETURN TO CMS
  31 ERR1   TYPE  'FILE NOT FOUND'
  45 ERRET  LA    15,1     INDICATE ERROR RETURN
  46        B     DONE
  47 ERR2   TYPE  'ERROR WRITING'
  61        B      ERRET
  62 ERR3   TYPE   'ERROR READING'
  76        B      ERRET
  77 INPUT  FCB    (FILE1,DCL),BUFF1
  95 OUTPUT FCB    (FILE2,DCL),BUFF1
 113 BUFF1  DS     CL80
 114        END    BEGIN
 115               =CL8'RDBUF'
 116               =CL8'WRBUF'
 117               =F'12'
```

Figure 26

An example of CMS I/O macros in an assembler language program. This program copies the file FILE1 DCL P5, and assigns the new copy the identifiers FILE2 DCL P5. The STATE and SETUP macros are executed for the existing file to initialize the File Control Block. These are not needed for the output file, since it is being created. The program then alternates reading and writing records, until the RDBUF at Statement 15 returns the end of file error code. Control then goes to the CKEOF at Statement 26, which tests whether end of file or some other error has occurred. If it is end of file, the error return register is cleared, and control returned to CMS via Register 14. If an error occurs, an appropriate response is typed, and a value is placed in Register 15 to indicate to CMS that an error condition exists. In any case, CMS closes both files when the program is completed.

Purpose:

The CKEOF macro checks the return code from RDBUF for the end of file indication. Any other value is considered an error.

Format:

```
-----------------------------------
| label | CKEOF | erlabel |
-----------------------------------
```

label specifies a statement label.

erlabel specifies an error-handling routine.

Usage:

The label of the CKEOF macro is normally the ERROR= parameter of the RDBUF macro. CKEOF checks for a value of 12 in register 15, indicating end of file. If any other value is present (including zero), CKEOF branches to the specified error routine.

Example:

```
        RDBUF A17,ERROR=DONE
        .
        .
DONE    CKEOF ERRR
CONT    .
        .
ERRR    TYPE  'ERROR READING A17'
        BR    14
```

This sequence of instructions gives control to CKEOF whenever register 15 is non-zero on return from RDBUF. If the error return indicates end of file, execution continues with CONT. If some other error has occurred, CKEOF branches to ERRR, where a response is typed and control returned to CMS. Note that the value in register 15 is typed by CMS only if it is saved and restored around the TYPE macro. If this is done register 15 is displayed in the error completion response:

```
E(nnnnn); T=n.nn/nn.nn hh.mm.ss
```

CMSREG Macro

Purpose:

The CMSREG macro equates symbolic names to general-purpose registers and floating-point registers.

Format:

```
---------------------
|       CMSREG      |
---------------------
```

Usage:

The CMSREG macro-instruction equates the following symbolic names to the corresponding general-purpose and floating-point registers. This allows the symbolic names to be used in place of the register designations.

General-Purpose Registers

```
R0      EQU     0
R1      EQU     1
R2      EQU     2
R3      EQU     3
R4      EQU     4
R5      EQU     5
R6      EQU     6
R7      EQU     7
R8      EQU     8
R9      EQU     9
R10     EQU     10
R11     EQU     11
R12     EQU     12
R13     EQU     13
R14     EQU     14
R15     EQU     15
```

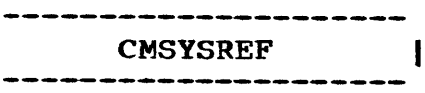
Floating-Point Registers

```
FR0     EQU     0
FR2     EQU     2
FR4     EQU     4
FR6     EQU     6
```

Purpose:

The CMSYSREF macro allows users to branch to routines in the CMS nucleus which cannot be called by CMS SVC's.

Format:



Usage:

The CMSYSREF macro instruction generates a series of EQU statements which provide the addresses of various CMS nucleus routines not available via a supervisor call. The addresses of these routines are listed in a nucleus module whose entry point is SYSREF. The name of each nucleus routine, with a D prefixed to it, is equated to its displacement from SYSREF in the macro expansion.

Note:

In all programs in which the CMSYSREF macro is called, SYSREF must be declared in an EXTRN statement.

Example:

The following example shows how to call the SETCLK function.

```
SAMPROG     CSECT
            EXTRN    SYSREF
            .
            .
            L        5, = A (SYSREF)
            L        15, DSETCLK (5)
            BALR     14,15
            LTR      15,15
            BNZ      ERRET
            .
            .
            CMSYSREF
            .
            .
```

In the expansion of CMSYSREF, DSETCLK is equated to the displacement of the SETCLK routine in the SYSREF list. The address is loaded into register 15 for the BALR, and register 15 is checked for an error code after control returns to SAMPROG. SETCLK is a routine that saves the current value of the timer for subsequent use, as described in the CMS Program Logic Manual.

ERASE Macro

Purpose:

The ERASE macro provides linkage to the ERASE command, which erases the specified file from the user's permanent disk.

Format:

<label>	ERASE	fcb

label is an optional statement label.

fcb is the label of the FCB which identifies the file to be erased.

Usage:

The ERASE macro allows the user to erase any file on his permanent disk.

Note:

To erase a disk file with a filemode other than P, the user may set up a parameter list and call the ERASE command directly, as explained under "Linkage to CMS Commands and Routines".

Example:

```
ERASE INPUT
```

The permanent disk file identified by the FCB labeled INPUT is erased. For example, if the macro

```
     INPUT FCB (INFILE,DATA), BUFF
```

has been issued, file INFILE DATA Pn is erased.

Purpose:

The FCB macro generates a CMS File Control Block, which serves as a parameter list naming and describing disk files for the CMS I/O routines.

Format:

```
-------------------------------------------
| label | FCB | (filename,filetype),area |
-------------------------------------------
```

label — is a statement label of seven or less characters.

(filename,filetype) — specify the name and type of the file. Mode * is assumed.

area — is the label of the input or output area in the program.

Usage:

A File Control Block is needed for each disk file referenced in a program. The label assigned to the FCB macro is a parameter of the SETUP, RDBUF, and WRBUF macros. The FCB label with an S prefixed is a parameter of the STATE macro.

Before the FCB can be used for input files, the STATE and SETUP macros must be executed. No initialization is needed for output files being created.

Included in the parameter list generated by FCB is an item number field. This specifies the record of the file being referenced. If unchanged, records are read sequentially from first to last, and written in the same order. Records are added to the end of existing files. However, if a number is specified in the item number field, the item specified is read or written. The field is a half word at LABEL+26, where LABEL is the FCB statement label.

Example:

```
INPUT FCB (INFILE,DATA),BUFF
```

Expansion of this macro generates a parameter list referencing INFILE DATA. Before a RDBUF macro can use this FCB for input, the following sequence must be executed:

```
     STATE SINPUT,ERROR
     SETUP INPUT
```

No initialization is needed for output. Data is read into the area labelled BUFF unless a different area is specified with RDBUF.

Purpose:

The FINIS macro provides linkage to the FINIS command, which closes the specified user file by clearing its entry from the active file table.

Format:

```
-----------------------------------
| < label > | FINIS | fcb |
-----------------------------------
```

label is an optional statement label

fcb is the label of the FCB naming the file to be closed

Usage:

The FINIS macro closes the permanent disk file identified by the FCB whose label is given as an operand of the macro.

Note:

CMS closes files automatically at program completion.

Example:

FINIS INPUT
The permanent disk file identified by the FCB labeled INPUT is closed. For example, if the macro

```
INPUT FCB (INFILE,DATA),BUFF
```

is issued, file INFILE DATA Pn is closed.

Purpose:

RDBUF reads a record from a disk file.

Format:

```
-------------------------------------------------------
| <label> | RDBUF | fcb <,AREA=alabel><,ERROR=elabel> |
-------------------------------------------------------
```

label is an optional statement label

fcb is the label of the FCB naming the file to be read

alabel is the label of an input buffer. If omitted, the area specified in the FCB is used

elabel is the entry of an error-handling routine. If omitted, errors, including end of file, are ignored.

Usage:

RDBUF returns one item each time it is executed. The item returned is specified by the item number field of the File Control Block at FCB+26. This number is incremented each time RDBUF is executed. It may be set to any value, allowing direct access to files.

The file read may have fixed or variable length records, but the buffer named AREA= must be large enough for the longest record in the file. The buffer named with RDBUF overrides that specified in FCB if it is different.

Each time RDBUF is executed, it links via a CMS SVC instruction to a routine that returns control when the record requested has been read in. If no error occurs, register 15 is set to zero on return. If an error or end of file occurs, register 15 contains a code indicating the type of error. See Figure 27 for a list of RDBUF error returns. If ERROR1 is supplied, control goes to that point on an error. If no label is supplied, control returns immediately following RDBUF, regardless of error. Note that end of file is always considered an error. An error handling routine should start with the CKEOF macro, to detect end of file.

CODE	MEANING
0	no error
1	file type not found
2	buffer area not in core
3	disk error
4	illegal mode letter
5	item number is 0
6	no core available
7	file not in correct format
8	buffer too small (truncated item)
9	file open for writing
11	incorrect number of items
12	end of file
13	mode number invalid

Figure 27. RDBUF error return codes (returned in register 15)

Example:

```
READA  RDBUF INFIL,AREA=BUF,ERROR=EOF
NEXT   .
       .
       B     READA
EOF    CKEOF ERREAD
       .
       .
BUF    DS    CL80
INFIL  FCB   (ELIPR,DATA),BUF
```

Assuming that the RDBUF macro is preceded by a STATE SINFIL and a SETUP INFIL, this sequence reads successive items from the file ELIPR DATA Pn. Items are placed in BUF, and control returned at NEXT. When end of file is reached, control goes to EOF where CKEOF checks for a 12 in register 15. If any other value is in register 15, control goes to ERREAD.

SETUP Macro

Purpose:

SETUP initializes the FCB with information from the user's permanent file directory.

Format:

```
-----------------------------------
| label | SETUP | fcb             |
-----------------------------------
```

label is an optional statement label

fcb is the label of the FCB to be initialized

Usage:

SETUP is executed following STATE to initialize the FCB. Record length and record format (fixed or variable length) are filled in. SETUP should not be executed if STATE returns with an error code in Register 15 indicating the file was not found in the directory.

Example:

SETUP A17
This initializes the File Control Block which has been generated by the macro:

```
A17  FCB  (NAME,TYPE),,BUFFER
```

STATE Macro

Purpose:

The STATE macro provides linkage to a CMS routine which searches the user's permanent disk directory for a specified file.

Format:

```
-----------------------------------
|<label>| STATE | Sfcblabel,error |
-----------------------------------
```

label — is an optional statement label

Sfcblabel — is the label of the FCB being referenced with an S prefixed to it

error — is the label of a routine to handle an error return

Usage:

STATE should be executed before any existing file is referenced for input or output. STATE places the directory address of the specified file in the FCB. From this entry, SETUP initializes the FCB.

If the file specified is not found in the user's file directory, STATE returns control at the point specified by the second parameter.

Note:

STATE is not needed for files being created.

Example:

```
ENTER  STATE  SA17,ERROUT
```

This causes a search of the permanent file directory for the file described by the FCB macro with the label A17. If it is found, the address is filled in the FCB; if not, control goes to ERROUT.

Purpose:

TYPE generates a parameter list and issues a CMS supervisor call to type a line of terminal output.

Format:

```
 ---------------------------------
 |       |    | 'message'  (r)    |
 |<label>|TYPE|                   |
 |       |    | msglabel   n      |
 |       |    |            length |
 ---------------------------------
```

label	an optional statement label
'message'	the message to be typed
msglabel	the label of an area where the message to be typed is located
(r)	a register containing the length of the message area
n	is a self-defining term giving the length of the message area.
length	the label of a full or half word constant containing the length of the message area

Usage:

The maximum message length is 130 characters. If the actual message is specified in single quote marks with the macro, the length parameter is not used. If the label of an output area is specified instead of the message itself, the length must be included. Length may be specified as a self-defining term (a decimal number or an equated symbol), or as the label of a full or half word constant containing the length. If length is not known until execution time, a register containing the length may be specified in parentheses.

Notes:

a. Issuing a TYPE macro specifying the message to be typed in single quotes is equivalent to issuing a CMSTYPE macro specifying the same message.

b. The GEN macro is included in SYSLIB MACLIB for use by the TYPE macro, and is not meaningful for use in a user program.

Examples:

```
a.    ERR1    TYPE 'ERROR WHILE READING'
The message ERROR WHILE READING is typed at the terminal.

b.            TYPE MSG,LTH
      MSG     DC   C'EXECUTION BEGINS...'
      LTH     EQU  *-MSG
The message EXECUTION BEGINS... is typed.
```

Purpose:

TYPIN reads a line of input from the terminal.

Format:

```
-----------------------------------------
|<label>|TYPIN|  area <,length><,ED=c>  |
-----------------------------------------
```

label — an optional statement label

area — the label of a 130-character input buffer

length — an optional label to be assigned to a three-byte field of the parameter list where the number of characters read is placed

c — a one-character code specifying the editing of the input line to be performed by CMS before returning control. If omitted, U is assumed.

Editing Codes

U — interpret delete characters, translate to upper case, pad with blanks to 130 characters.

V — interpret delete characters, translate to upper case.

S — interpret delete characters, pad with blanks to 130 characters.

T — interpret delete characters.

X — do not edit the line.

Usage:

TYPIN generates a parameter list and issues a CMS supervisor call to read a line from the terminal. The input area specified must be 130 characters long. The number of characters read is filled into a three-byte area in the parameter list. This number may be accessed by specifying a label as the second parameter of the macro. This number is always the number of characters read, whether or not the record has been padded to 130 characters with blanks.

The ED=parameter allows the user to select which of the CMS editing functions are to be performed on the line. The delete characters (@ and ¢) have their normal CMS meaning if U, V, S, or T is specified. Alphabetic data is translated to upper case if U or V is specified. The buffer

area following the characters read is set to blanks (hex '40') if U or S is specified. The X option returns the line exactly as typed. U is the default option.

Note:

The number of bytes read field is in the low-order three bytes of a full word. For example, if the label specified is LAB, the number may be obtained by a LOAD HALF instruction from LAB+1.

Examples:

```
a.    RD    TYPIN BUFF1,LNGTH
      NEXT  LH    2,LNGTH+1

      BUFF1 DS    CL130
```

This sequence reads a line from the terminal into BUFF1. Since no ED= option is specified, U is assumed. Delete characters are interpreted, the length adjusted, lowercase letters translated to uppercase, and the buffer behind the line set to blanks before control is returned at NEXT. The LH instruction places the number of characters read into Register 2.

```
b.    GET   TYPIN B,ED=X
            .
            .
      B     DS    33F
```

This macro reads a terminal line into B. The X option indicates that no editing is to be performed on the line.

Purpose:

The WRBUF macro creates, updates, or expands disk files.

Format:

```
-------------------------------------------------
|<label>|WRBUF| fcb <,AREA=alabel><,ERROR=elabel> |
-------------------------------------------------
```

label	an optional statement label
fcb	a label referencing the FCB describing the output file
alabel	the label of an area from which data is to be written
elabel	the label of an error-handling routine

Usage:

WRBUF normally is used to create a new sequential file or to add to the end of an existing file. However, any record of an existing fixed-length record file may be replaced by adjusting the item number field at FCB+26.

The AREA= parameter may be omitted with WRBUF if specified in the FCB. If specified in both places, WRBUF takes precedence.

Control is normally returned with register 15 set to zero, indicating error-free completion. If an error occurs, register 15 contains a code indicating the nature of the error, and control is returned wherever specified by the ERROR= parameter. If no ERROR= is included, errors are ignored. See Figure 28 for a list of WRBUF error returns.

Example:

```
          WRBUF 17,AREA=ITEM,ERROR=QUIT
                .
    QUIT  BR    14
    ITEM  DS    CL80
    A17   FCB   (INTR,JBH),ITEM
```

This sequence writes successive items into INTR JBH Pn. On an error control goes to QUIT, which returns to CMS. The error code is printed in the CMS error completion response.

Code	Meaning
0	no errors
1	filename missing
2	buffer area not in core
3	disk error
4	illegal mode letter
5	illegal mode number
6	no core available
7	skipping variable item
8	length not specified
9	file open for reading
11	fixed or variable not set
12	mode SY illegal
13	disk is full
14	read-only file
15	item wrong length
16	F-V changed
17	65K length limit on item
18	no. items more than 1 for V-format
19	no. items equal zero

Figure 28. WRBUF return codes returned in register 15

OS MACROS

The OS macros that are used by the Cambridge Monitor System are contained in the file OSMACRO MACLIB SY, which resides on the system disk.

To obtain a list of the names, size, and location of the macro definitions in OSMACRO MACLIB, the command MACLIB LIST OSMACRO may be issued. To print the macro definitions, refer to MACLIB under "Libraries".

The OS macros supported under CMS are listed below.

Note that only the forms of the following macros used by the OS language processors are supported under CMS. Programs using OS macros not listed or using unsupported forms of the following macros will not run correctly under CMS.

ABEND
ANALYZ
ASGNBFR
ASMTRTAB
ATTACH
ATTNINQ
BLDL
BREAKOFF
BSP
BUFFER
BUFINO
BUILD
CALL
CAMLST
CANCELM
CATALOG
CHAP
CHECK
CHGNTRY
CHKPT
CHNGP
CHNGT
CIRB
CKREQ
CLOSE
CLOSEMC
CNTRL
COPYP
COPYQ
COPYT
COUNTER
DAR
DATESTMP
DCB
DELETE
DEQ
DETACH
DEVTYPE

DFTRMLST
DIRECT
DLIST
DOM
DUMP
ENDRCU
ENDREADY
ENDSEND
ENQ
EOA
EOB
EOBLC
EOV
ERRMSG
ESETL
EXCP
EXTRACT
FEOV
FIND
FREEBUF
FREEDBUF
FREEMAIN
FREEPOOL
GET
GETBUF
GETMAIN
GETPOOL
IDENTIFY
IECTDECB
IHBERMAC
IHBGAM1
IHBGAM2
IHBGAM3
IHBINNRA
IHBINNRB
IHBOPLST
IHBRDWRD
IHBRDWRK
IHBRDWRS
IHBRDWRT
IHB01
IHB02
INDEX
INTERCPT
IOHALT
LABEL
LERB
LERPRT
LINK
LOAD
LOCATE
LOGSEG
LOPEN
LPSTART
MODE
MSGTYPE
NOTE
OACB
OBTAIN
ONLIST
OPCTL
OPEN
OPTION
PAUSE
POINT
POLL
POLLIMIT
POST
POSTRCV
POSTSEND
PROCESS
PRTOV
PUT
PUTX
RCVEITA2
RCVEZSC3
RCVHDR
RCVSEG
RDJFCB
READ
RELBUF
RELEASEM
RELEX
RELSE
RENAME
REQBUF
REROUTE
RESERVE
RESETPL
RETRIEVE
RETURN
RJELINE
RJETABL
RJETERM
RJEUSER
RLSEBFR
ROUTE
SAEC
SAVE
SCRATCH
SEGLD
SEGWT
SENDHDR
SENDITA2
SENDSEG
SENDZSC3
SEQIN
SEQOUT
SET
SETL
SETPRT
SKIP

SMFWTM
SNAP
SOURCE
SPAR
SPIE
STAE
STARTLN
STIMER
STOPLN
STOW
SYNADAR
SYNADRLS
TERM
TERMTBL
TEST
TIME
TIMESTMP
TRACE
TRUNC
TTIMER
TWAIT
WAIT
WAITR
WRITE
WRU
WTL
WTO
WTOR
XCTL
XDAP

The following OS macros are defined in CMS but do not perform a function as in OS. They are essentially no-ops (that is, they have no meaning in CMS and return an error code when the SVC is issued).

ATTACH
CHKPT
DETACH
DEQ
DELETE
EXTRACT
ENQ
IDENTIFY
STIMER

For a discussion of the OS macros themselves, refer to IBM manual, C28-6647 IBM System/360 Operating System Supervisor and Data Management Macro Instructions. For a description of the use of the OS macros in CMS refer to the CMS Program Logic Manual.

CMS ROUTINES (FUNCTIONS)

Routines that can be called with the SVC X'CA' are called functions. These routines serve as the basic interface between programs and the CMS nucleus.

The calling sequence for all routines is similar. There are two parts of each calling sequence; the code to transfer control to the CMS routine, and the parameter list. The inline code always has the following form

```
LA      1,PLIST     put address of parameter list into GRP1
SVC     X'CA'       transfer control to subroutine
DC      AL4(ERR)    address of error return if desired
normal return
  .
  .
  .
```

Register 15 is the only register that is modified when control is returned to the calling program. On a normal return, register 15 contains zero, on an error return, it contains an error code. The error codes are described in the write-up of each routine. If an error return is specified, the byte after the SVC instruction is zero and the following three bytes are assumed to contain the address of the error return. If the error return address is specified, the normal return is to four bytes after the SVC instruction; otherwise, it is to the location after the SVC instruction. If no error handling is to be provided, it is recommended that DC AL4(*+4) be used, otherwise an error causes DEBUG to be entered.

The PLIST differs for each routine and is described in subsequent sections for each routine. All functions can be called directly as CMS commands although many functions require a parameter list specified in hexadecimal and thus should only be called from an assembly language program. The following CMS commands directly call the equivalent CMS functions and are described under the sections on CMS commands:

ALTER, CLOSIO, ERASE, FINIS, GENMOD, GLOBAL, LOAD, LOADMOD, REUSE, USE, START, $, DEBUG, EXEC, LOGOUT, IPL, BLIP, LINEND

See "Assembly Language Programming" for PLIST format.

The commands ALTER, ERASE, BLIP, and LINEND should be called from an assembly language program as functions if it is desirable to specify a character that cannot be typed from a keyboard.

The functions STATE and TAPEIO are useful as commands as well as functions. When called as a function, STATE returns the address of a copy of the file status table that can be used in further processing. When called as a function,

TAPEIO reads or writes a tape record. The use of the functions as commands is described under the section on CMS commands. The PLIST for the functions below are described in the following section.

ATTN	Stacks a line for terminal input
CARDIO	Reads and/or punches cards
CONWAIT	Waits for terminal I/O to finish
CPFUNCTN	Issues CP-67 console functions from CMS
ERASE	Erases file(s)
FINIS	Closes file(s)
HNDINT	Sets or clears I/O interrupt return addresses
HNDSVC	Sets or clears SVC handling addresses
POINT	Sets read or write pointer
PRINTR	Prints line
RDBUF	Reads item(s) from disk
STATE	Queries file status
TAPEIO	Handles tape I/O
TRAP	Sets external interrupt address
TYPE	Writes to terminal without carriage return
WAIT	Waits for interrupt
WAITRD	Reads terminal
WRBUF	Writes item(s) on disk

ATTN Function

Purpose:

The ATTN function stacks a line into the input buffer.

Calling Sequence:

```
PLIST    DS      0D
         DC      CL8'ATTN'
         DC      CL4'order'      LIFO or FIFO
         DC      AL1(LBUFF)
         DC      AL3(BUFF)
         .
         .
         .
BUFF     DC      C'line'
LBUFF    EQU     * - BUFF
```

Usage:

The line that is stacked is unstacked and used when a call to WAITRD is made to read a line from the typewriter console. Any number of lines may be stacked for subsequent use as terminal input. When the input stack is empty, the keyboard is unlocked to receive typewriter input.

Purpose:

The CARDIO function punches a card from the specified 80 byte area, or reads a card (or prints a line) into the specified 80 (or 132) byte area.

Calling Sequence:

```
PLIST     DC      CL8'              CARDRD or CARDPH
          DC      X'flag'           X'00': standard reader
                                    X'80': non-standard reader or
                                    extended PLIST
          DC      AL3(buffer)
          DC      H'no. bytes to be read'
          DC      H'0'    no. of bytes read
```

Error Codes:

```
E(00001)    End of file
E(00002)    Unit Check
E(00003)    Unknown error
E(00004)    Not operational
```

CONWAIT Function

Purpose:

The CONWAIT function waits for all stacked reads and writes to finish from the console typewriter.

Calling Sequence:

```
PLIST    DS      0D
         DC      CL8'CONWAIT'
         DC      CL4'CON1'
```

Usage:

CONWAIT is called as a standard CMS function. The device id 'CON1' corresponds to the console typewriter.

Error Returns:

None.

CPFUNCTN Function

Purpose:

The CPFUNCTN function transmits console functions to CP-67 without leaving the virtual machine mode.

Calling Sequence:

```
PLIST  DS  0F
       DC  CL8'CPFUNCTN'
       DC  CL8'NOMSG'  This parameter may be omitted
       DC  CLn'CP command string'
       DC  X'FF'  fence
```

Error Codes:

E(00001) No CP command string present

E(00004) INVALID CP REQUEST

E(00008) BAD ARGUMENT

E(xxxxx) Any other error codes are from the CP console function specified.

ERASE Function

Purpose:

The ERASE function erases the specified file(s).

Calling Sequence:

```
PLIST    DS       0F
         DC       CL8'ERASE'      called routine
         DC       CL8'    '       filename or *
         DC       CL8'    '       filetype or *
         DC       CL2'  '         mode or *
```

Error Codes:

```
E(00001)     First character of mode illegal
E(00002)     File not found
E(00004)     Disk error
```

FINIS Function

Purpose:

The FINIS function closes the specified file(s).

Calling Sequence:

```
PLIST     DS        0D
          DC        CL8'FINIS'        called routine
          DC        CL8'      '       filename or *
          DC        CL8'      '       filetype or *
          DC        CL2'  '           mode letter or *
```

Note:

FINIS does not update the directory on the disk.

Error Codes:

```
E(00001)   Invalid filename
E(00002)   Invalid filetype
E(00003)   Disk error
E(00004)   Invalid mode
E(00006)   File not open
```

HNDINT Function

Purpose:

The HNDINT function sets the CMS I/O interrupt handling routines to transfer control to a given location for an I/O device other than those normally handled by CMS, or clears such transfer requests.

Calling Sequence:

```
        DS      0F
PLIST   DC      CL8'HNDINT'             called routine
        DC      CL4'SET' or CL4'CLR'      function
        IODEV   NAME,NUMBER,ADDRESS,
                  ASAP/WAIT-FLAG,KEEP/CLEAR-FLAG
          .                   .
          .                   .
          .                   .
        DC      X'FFFFFFFF'             end of list
```

Macro IODEV:

The IODEV macro sets up the following information in a 12-byte field:

```
NAME = Symbolic device name (1st 4 letters)
NUMBER = Hexadecimal device address
ADDRESS = Symbolic address of interrupt-handler to
          be invoked.  If address = 0, interrupts
          are ignored when received.
ASAP/WAIT-FLAG:
     ASAP = Invoke interrupt-handler immediately
     WAIT = Invoke interrupt-handler only when
            WAIT is called
KEEP/CLEAR-FLAG:
     KEEP = Retain interrupt-handling between CMS
            commands
     CLEAR = Clear interrupt-handling after each
             CMS command.  CLEAR is the default.
```

Example:

```
IODEV  NEWD,387,MYCODE,ASAP,KEEP
```

Usage:

When an interrupt is received and processed by 'IOINT', it passes control to the interrupt-handler as follows:

```
     Register 0,1      I/O OLD PSW
              2,3      CSW
              4        Device address
              14       Return address to IOINT
              15       Address of interrupt-handler
```

When processing is complete, the interrupt-handler must return to IOINT via register 14, with Register 15 as follows:

R15 = 0 means SUCCESSFUL HANDLING
R15 Nonzero means ANOTHER INTERRUPT EXPECTED.

The general procedures for CMS I/O handling using HNDINT follow.

1. The program must initialize handling to be done via HNDINT SET.

2. When I/O to the appropriate device is to be done, the system-mask must be set OFF by SSM instruction and the appropriate SIO given.

3. When SIO is performed satisfactorily, the system-mask can be set to allow all interrupts.

4a. If ASAP is specified, the interrupt-handler is invoked as soon as the interrupt is fielded by CMS IOINT. The interrupt-handler returns to IOINT which returns to users program.

4b. If ASAP is not specified, IOINT retains needed information until the CMS WAIT function is called.

5. When the program needs the interrupt to have been received, the CMS WAIT function is called. If the interrupt has not yet been received, CMS goes into the WAIT state until IOINT fields and processes the interrupt in the normal way.

If the interrupt has been received and processed (for example, on ASAP), WAIT returns to the caller with the necessary internal flags cleared.

If the interrupt has been received but not yet processed as under the WAIT option rather than ASAP, CMS WAIT calls IOINT to invoke the desired interrupt-handler, clears the necessary flags, and returns to the caller.

6. When finished, the user program should clear the interrupt-handling scheme through the HNDINT CLR call. If the KEEP option is used, the interrupt-handler remains intact in core.

Error Code:

E(00001) Incorrect parameter list

HNDSVC Function

Purpose:

The HNDSVC function initializes the SVC-interrupt handler to transfer control to a given location for a specific SVC number--other than X'CA' or 202--or clears such previous handling.

Calling Sequence:

```
            DS      OF
PLIST       DC      CL8'HNDSVC'             called routine
            DC      CL4'SET' or CL4'CLR'    function
            DC      AL1(SVC-number),AL3(SVC-handler)
            .         .
            .         .
            .         .
            DC      X'FFFFFFFF'             end of list
```

Usage:

At entry to a non-CMS SVC-handling routine the following conditions exist:

Registers

0-11 and 15 as they were at SVC time
12 address of SVC-handler routine
13 address of SVC save area
14 return address to SVCINT

The SVC save area has the following format:

Bytes	Contents
0-63	caller's registers 0-15
64-71	SVC-old-PSW
72-95	floating-point registers 2,4,6*
96-175	80 bytes for use by SVC-handler

*FPR 0 is saved elsewhere by CMS.

Notes:

a. For CLR the address fields are irrelevant.

b. Individual SVC-numbers may be added or cleared at different times but should all be cleared before termination of the command.

Error Codes:

E(00001) Incorrect PLIST
E(00002) SVC-number replaced another of the same number
E(00003) SVC-number cleared was not set

Purpose:

The POINT function sets the read pointer and/or the write pointer at a specified item number.

Calling Sequence:

```
PLIST    DS     0D
         DC     CL8'POINT'     called routine
         DC     CL8'     '     filename
         DC     CL8'     '     filetype
         DC     CL2'     '     mode or *
         DC     H'       '     write pointer
         DC     H'       '     read pointer
```

Usage:

This routine sets the read or write pointer in the file status table to a value provided by the caller. Zero leaves the pointer unchanged; a value of H'-1' (X'FFFF') sets the write pointer to the last item number plus one.

Error Codes:

E(00001)	File specified does not exist
E(00002)	First character of mode illegal

PRINTR Function

Purpose:

The PRINTR function writes a line of text on the offline printer.

Calling Sequence:

```
PLIST  DS  0D
       DC  CL8'PRINTR'
       DC  A(bufsiz)    buffer area
       DC  F' '         buffer size
```

Notes:

a. The first byte of the buffer is used for carriage control and is not printed. The carriage controls are:

CL1' '	single space
CL1'0'	double space
CL1'1'	page eject
CL1'-'	triple space
CL1'+'	write, no space
CL1'n'	skip to channel n

b. Machine CCW OP code carriage controls as used by the assembler are also accepted.

c. The buffer size should not exceed 133 bytes. If the buffer size is 1, only the carriage control function will occur.

Error Codes:

E(00001)	Printer unit check
E(00002)	Illegal carriage control
E(00003)	Incorrect parameter list
E(00004)	Not operational
E(00005)	Unknown printer error

Purpose:

The RDBUF function reads an item of information from a disk file.

Calling Sequence:

```
PLIST DS 0D
      DC CL8'RDBUF'    called routine
      DC CL8'  '       filename
      DC CL8'  '       filetype
      DC CL2'  '       mode or *
      DC H'    '       item number
      DC A(userarea)   pointer to input buffer
      DC F'    '       number of bytes in buffer
      DC CL2'F'        fixed-variable flag (F or V)
      DC H'    '       number of items to read
      DC F'    '       number of bytes actually read
```

Notes:

a. All errors except error 8 cause the function call to be aborted. On error code 8 that portion of the item that fits in core is read.

b. The number of bytes in the user area divided by the number of items (that is, the same logical record length) must be constant for a fixed-length-record file.

Error Codes:

E(00001)	File not found
E(00002)	User's memory address is illegal
E(00003)	Disk malfunction has occurred
E(00005)	Number of items equal zero
E(00007)	File not written with WRBUF therefore it cannot be read
E(00008)	User's memory area too small for item
E(00009)	File open for writing and therefore cannot be read
E(00011)	Number of items greater than 1 for variable-length file
E(00012)	Item number specified does not exist (EOF)
E(00013)	Variable file has invalid displacement in Active File Table

STATE Function

Purpose:

The STATE function provides a copy of the FST (file status table) block for the file specified in the parameter list.

Calling Sequence:

```
PLIST     DS    0D
          DC    CL8'STATF'      called routine
          DC    CL8'      '     filename
          DC    CL8'      '     filetype
          DC    CL2'  '         mode letter or *
          DC    CL2'  '         unused
PRETN     DC    A(*)            address of copy of FST
                                block returned
```

Notes:

a. PRETN is set to the location of a copy of the file status table entry if the file is found.

b. Mode * means that the first file found with name and type specified is used. The standard order of search is used.

Error Codes:

E(00001) File specified doesn't exist
E(00004) First character of mode illegal

TAPEIO Function

Purpose:

The TAPEIO function reads or writes a tape record or positions a tape.

Calling Sequence:

```
PLIST     DS     0D
          DC     CL8'TAPEIO'
          DC     CL8'function'
          DC     CL4'deviceid'     Symbolic tape address
          DC     XL1'modeset'      7-track mode set
          DC     AL3(buffer)       Buffer address
          DC     F'size'           Buffer size
COUNT     DS     F                 Number of bytes read
```

Functions are:

BSF	backspace one file
BSR	backspace one record
FSF	forward space one file
FSR	forward space one record
READ	read one record
REWIND	rewind the tape to load point
RUN	rewind and unload the tape
WRITE	write one record
WRITEOF	write a tape mark
WTM	write a tape mark
ERG	erase a gap

Device ID's are:

TAP1 or TAP2 corresponding to 180 or 181

The modeset code is one byte of the form

DDMMM011 with the following interpretation

DD	7 track density
00	200
01	556
10	800
11	800

MMM	Function
000	not used
001	not used
010	set density, odd parity, converter on, translator off
011	not used
100	even parity, converter off, translator off
101	set density, even parity, converter off, translator on
110	set density, odd parity, converter off, translator off
111	set density, odd parity, converter off, translator on

Usage:

The function TAPEIO is used to read, write or move magnetic tape. If the WRITE function is used, the number of bytes indicated is written. If the READ function is used, information is moved into the buffer and the number of bytes read is stored into COUNT. If a tape mark is read, the function returns with error code 2.

Notes:

a. A mode set of X'00' causes the default mode bit of X'B3' to be used.

b. A mode set of X'B3' indicates density 800, parity odd, converter off, and translator off.

c. A mode of X'93' indicates density 800, parity odd, converter on, and translator off.

Responses:

TAPn NOT READY YET
The tape has been attached but it is not in a ready status.

(OK - READY NOW)
The tape is now ready for use.

Error Codes:

E(00001) INVALID 'TAPEIO READ' PARAMETER-LIST
An invalid parameter list was specified for reading tape.

E(00001) INVALID 'TAPEIO WRITE' PARAMETER-LIST
An invalid parameter list was specified for writing tape.

E(00002)
An end of file or end of tape has occurred.

E(00003)
A permanent I/O error has occurred while reading or writing.

E(00004)
An illegal symbolic device id was specified.

E(00005) TAPn NOT ATTACHED
The tape unit has not been attached to the virtual machine. Refer to "Operating Considerations - Tape Procedures".

E(00006) TAPn IS FILE PROTECTED
The tape contains a file-protect ring. Therefore it can not be written on.

E(00007) TAPn - SERIOUS TAPE ERROR ATTEMPTING function
An unrecoverable tape error has occurred while attempting the specified function.

TRAP Function

Purpose:

The TRAP function sets a user's return for an external interrupt. This return overrides the call to DEBUG on an external interrupt.

Calling Sequence:

```
PLIST    DS     0D
         DC     CL8'TRAP'
         DC     A(trapsubr)
```

where trapsubr is the location transferred to on an external interrupt. If the parameter trapsubr is a zero, the return is reset to go to DEBUG on an external interrupt.

Usage:

The user's interrupt routine should set a flag which should be examined by the main line program. After the flag is set, this routine should return to the location specified in GPR14 on entry. All other general registers can be used as desired. The main line program should periodically examine the trap flag to determine whether an external interrupt has occurred.

Error Returns:

None.

Purpose:

The TYPE function types an output message on the console. Terminal blanks (if any) are not deleted, and no carriage return is added.

Calling Sequence:

```
          DS      0F
PLIST     DC      CL8'TYPE'
          DC      AL1(1)           terminal number
          DC      AL3(MSG)         address of output message
          DC      C'code'          code B or K (see below)
          DC      AL3(EMSG-MSG)    message length (in bytes)
          .
          .
          .
MSG       DC      C' message to be typed without carriage
                  return'
EMSG      EQU     *
```

where the write codes are:
B = move line to free storage before typing
K = type line from the specified location

Note:

The output message must be from 1-130 bytes in length.

Error Codes:

E(00001) Invalid terminal number

E(00002) Length of output message not between 1 and 130 bytes

WAIT Function

Purpose:

The WAIT function awaits an interrupt from one of the specified devices.

Calling Sequence:

```
          DS      0F
PLIST     DC      CL8'WAIT'
          DC      CL4'deviceid'
          .
          .
          .
          DC      F'0'
INTDEV    DS      F
```

where the deviceid is CON1, DSK1, PCH1, RDR1,
PRN1, TAP1, or TAP2

Usage:

When one of the specified devices causes an interrupt, the deviceid is stored in the word INTDEV and control returns to the calling program.

Error Returns:

E(00001) Invalid deviceid specified

Purpose:

The WAITRD function reads an input message up to 130 bytes in length into a given buffer from a console and waits for completion of the input message.

Calling Sequence:

```
          DS    0F
PLIST     DC    CL8'WAITRD'
          DC    AL1(1)          terminal number
          DC    AL3(INPBUF)     address of 130-byte
                                 input buffer
          DC    C 'code'        U, V, S, T, or X
                                 (see below)
          DC    AL3(*-*)        byte count of input
                                 message is stored
                                 here
          .
          .
          .
INPBUF    DS    CL130           130 bytes (or more)
                                 input buffer
```

An input line can be edited as follows: Use the 'at sign' @ to delete the previous character, or the 'cent sign' ¢ to delete an entire line up to and including the cent sign.

Conventions for U, V, S, T, or X are:

U performs editing, upper-case translation and
  blank filling
V performs editing and upper case translation
S performs editing and blank filling
T performs editing only
X leaves input line exactly as is

Notes:

a. If the user has stacked input commands, WAITRD accepts the first stacked input message and moves it to the specified buffer.

b. The input buffer is zero-filled before read is initialized, and must be at least 130 bytes long.

Error Codes:

E(00001) Invalid terminal number
E(00002) Read-type invalid (not U, V, S, T, or X)

WRBUF Function

Purpose:

The WRBUF function writes one item of information into the file whose name is specified by the filename and filetype parameters. If the file does not exist when this function is first called, a new file is opened and assigned the given name and type. WRBUF automatically packs fixed-length items into an 800 byte buffer and writes this 800 byte buffer onto the disk when required.

Calling Sequence:

```
PLIST  DC  CL8'WRBUF'
       DC  CL8'    '     filename
       DC  CL8'    '     filetype
       DC  CL2'    '     mode must be specified
                         not *
       DC  H'      '     item number (0 if next item)
       DC  A(      )     user's buffer address
       DC  F'      '     number of bytes
       DC  CL2'    '     fixed-variable flag, F or V
       DC  H'      '     number of items to write
                         (0 treated as 1 item)
```

Note:

The number of bytes in the user area divided by the number of items (that is, the same logical record length) must be constant for a fixed-length-record file.

Error Codes:

```
E(00001)   File name or file type not specified
E(00002)   User memory address not in user area
E(00003)   Disk error. The disk might be read-only.
E(00004)   First character of mode illegal
E(00005)   Second character of mode illegal
E(00006)   Attempted to write item whose number
           exceeds 65533
E(00007)   Attempt to skip over unwritten variable-
           length item
E(00008)   Number of bytes not specified
E(00009)   File already active for reading
E(00010)   Maximum number of CMS files (3500) reached
E(00011)   F-V flag not F or V
E(00012)   Mode S (system) is illegal
E(00013)   Disk already full
           Goes to KILLEXF instead of returning to
           caller
E(00014)   Attempt to write a not yet formatted disk
E(00015)   Length of this item not same as previous
E(00016)   Characteristic (F-V Flag) not same as
           previous
E(00017)   Variable length item greater than 65K bytes
```

E(00018) Number of items greater than 1 for variable-length file

E(00019) Maximum number of data blocks per file (16060) reached.

FORTRAN

Purpose:

The FORTRAN command compiles programs written in FORTRAN IV into machine code, and provides program listings and diagnostics.

Format:

```
-------------------------------------------------------
|FORTRAN | filename1...filenameN <(option1...optionN)>|
|   F    |                                            |
-------------------------------------------------------
```

filename is the name of a FORTRAN file to be compiled. Up to 32 separate compilations may be performed by adding filenames separated by blanks.

option is one or more of the eight compiler options.

Options:

MAP includes tables of FORTRAN variables, NAMELIST, and FORMAT statements in the LISTING file.
NOMAP suppresses the tables of variables.

DECK generates the TEXT file of object code.
NODECK suppresses the TEXT file.

LIST includes a listing of object code in assembler language mnemonics in the LISTING file.
NOLIST suppresses the object code listing.

SOURCE includes the source program in the LISTING file.
NOSOURCE suppresses the listing of the source program.

BCD is used if the source program is punched in Binary Coded Decimal.
EBCDIC is used if the source program is punched in Extended Binary Coded Decimal Interchange Code.

GO forces compiler processing to completion despite source statement errors.
NOGO terminates compiler processing when serious errors are detected.

PRINT prints the LISTING file on the offline printer, and deletes it.
NOPRINT suppresses printing of the LISTING file.

DIAG types source program errors at the terminal.
NODIAG suppresses typing of source program errors.

Usage:

The FORTRAN command compiles files of FORTRAN source language into machine-language object code. Input files must have a filetype of FORTRAN and a record length of 80 characters. Up to 32 files may be compiled by one command by listing the filenames, and each file may contain any number of routines, each delimited by an END statement. Each file processed generates one object deck and one listing, replacing any previous output files for the same program.

FORTRAN uses the standard order of search for locating the source files.

The options governing compiler operation and output are specified in any order in a set of parentheses following the last filename. One set of options governs all compilations performed by one command. Each of the eight options has a default value which is selected when none is specified. The default values are

NOGO DIAG EBCDIC DECK NOSOURCE NOMAP NOLIST NOPRINT

Any combination of options is valid, but the result of specifying more than one value for a single option is unpredictable. Unsupported or misspelled options are ignored. If no options are specified, the parentheses are not necessary. No filenames, options, or comments should be placed following the closing parenthesis.

Diagnostic and error messages produced by the compiler are placed in the LISTING file (see "Output"), and, unless the NODIAG option is specified, are typed out at the terminal. The compiler error messages have two formats, depending on when the error is detected (see Figure 29). Statements in which an error is detected during the statement scan, such as a syntax error, are typed out immediately, followed by a line with a $ beneath each point at which an error was detected. The pointer line is followed by the error codes and explanations, numbered from left to right. If an error, such as an undefined label, is not detected until statement scanning is completed, the error message is typed, followed by a list of the labels or variables in error.

If source statement errors are detected, CMS terminates the compilation with a message and an error code of 32. If the GO option is specified, CMS does not terminate processing, although for some conditions the compiler terminates itself. When processing is completed under the GO option, any error completion code is the greatest error severity code assigned by the compiler (see "Error Messages").

Source files read through the offline card reader for compilation may be punched in either Binary Coded Decimal (BCD), or Extended Binary Coded Decimal Interchange Code

(EBCDIC). If BCD is used, the BCD option must be specified.

FORTRAN Options

The FORTRAN command does not produce a listing file, unless requested. All diagnostics are printed on the console, unless suppressed by the option NODIAG. To obtain a FORTRAN listing file, the options SOURCE, MAP, or LIST must be specified, and only those options requested are included in the listing file. If a listing file is produced, the diagnostics are included. If only the option NODIAG is specified, a listing file is produced containing only the FORTRAN diagnostics. The PRINT option directs the listing to the printer and prints only those parts of the listing requested by SOURCE, MAP, or LIST. If PRINT is the only option specified, only the diagnostics are printed.

TEXT Identification

The characters appearing in columns 73-76 of TEXT files generated by FORTRAN are as shown below. They are followed by a sequence number in columns 77-80:

a. for a subroutine--the first four letters of the subroutine name

b. for a main program--the first four letters of the filename if it is physically the first deck in the file; otherwise, the letters MAIN.

The letters used for columns 73-76 of a TEXT file also appear in the middle of the first line of each LISTING page. In addition, the name of the file appears at the beginning of the second line of each page in the LISTING file.

Output

Files with the designation "filename TEXT P5" and "filename LISTING P5", where "filename" is the name of the FORTRAN input file, are produced for each file compiled. If NODECK is specified, the TEXT file is suppressed.

The object program in the TEXT file is identical to that produced by a compiler under the Operating System, and object decks may be loaded and executed under CMS or OS. The entry point for the first main program in the file is the same as the filename. Subsequent main programs in the same file all have the entry point MAIN. Subroutines have the entry point specified in the SUBROUTINE statement, regardless of their position in the file.

Under the default options (NOSOURCE NOMAP NOLIST) the LISTING file contains diagnostic messages, and a statement of object program size in bytes (see Figure 30). Statements in which errors were detected are always included, with error messages in the same format as they are typed at the

terminal.

If MAP is specified, a table of addresses is generated for each of seven classifications of variables used in the source program. The classifications are COMMON, EQUIVALENCE, NAMELIST, FORMAT, scalar and array variables, and called subprogram names.

If LIST is specified, a listing of the object program is generated, with relative addresses, and instructions translated into assembler language.

The PRINT option causes the LISTING file to be printed on the offline printer, and then deleted. If NOPRINT, the default option, is specified, the LISITING file is saved on the permanent disk, and may be printed with the OFFLINE PRINTCC command or typed out at the terminal with the PRINTF command.

Notes

a. Previous LISTING and TEXT files with the same filename as the current FORTRAN input file are deleted, although in some cases they may not be replaced because of different options or an error termination.

b. If multiple files or a file with multiple routines are being compiled, the GO option should be specified to prevent an error termination of one compilation deleting all compilations requested.

References:

The FORTRAN command executes the System/360 Operating System FORTRAN IV (G) compiler. For information on the FORTRAN IV language, see IBM System/360 FORTRAN IV Language (C28-6515) and IBM System/360 Operating System FORTRAN IV Library: Mathematical and Service Subprograms (C28-6810). For information on compiler operation and messages, see IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide (C28-6817), information in this guide on Operating System job control language and data management is not applicable under CMS; the LOAD, NAME=, and LINECNT= options are not supported.

Responses:

Source statement errors and compiler messages are typed at the terminal unless NODIAG was specified. If GO was specified, or if no errors were detected, there is no response except the Ready message or an error completion code. The following responses should not occur:

```
READY THE PRINTER.

I/O ERROR ON PRINTER. 'PRINT' OPTION CANCELLED.
A 'LISTING' DISK FILE WILL BE CREATED.
```

If either appears at the terminal, notify the responsible system programmer.

Examples:

a. see Figure 29
A file, FAC FORTRAN P5, is to be compiled. No options are specified, so the set of default options govern the compilation. At statement 0041 the compiler detects three errors at the points indicated by the $'s in the succeeding line. Error 01) refers to the left $, error 02) to the $ in the center, error 03) to the $ under the 5. (The number in the '0n)' being the ordinal occurance of the error). Explanations of the error codes are found in the IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide. CMS cancels the compilation and supplies the error code E(00008).

```
fortran fac

   0041                READE (5,177) ANS(I)
                     $           $  $
            01) IEY0021 LABEL       02) IEY0041 COMMA       03)
                                  IEY0221 UNDEFINED LABEL
    777
E(00008); T=5.42/5.98 11.40.56
```

Figure 29. FORTRAN compilation with errors

b. see Figure 31
The same file as in the above example is to be compiled, but the SOURCE, MAP, and NOLIST options are specified. Assume that the errors flagged above have been corrected. The file compiles correctly this time.

```
fortran fac (source map nolist )
R; T=5.75/6.27 11.44.57
```

Figure 31. FORTRAN compilation with options

c. see Figure 32
Three files are to be compiled: FAC FORTRAN P5, TEST FORTRAN P5, and PRACTICE FORTRAN P5. No errors are detected during any of the compilations. Maps of variables are included in each of the three LISTING files, which are automatically printed offline, and erased from the disk.

```
fortran fac test practice
 COMPILING:  TEST
 COMPILING:  PRACTICE
R; T=5.90/6.69 11.44.06
```

Figure 32. Multiple FORTRAN compilations

Error Messages:

F(00001) TOO MANY LEFT PARENTHESIS.
More than one left parenthesis was found in the command. No compilation was started.

E(00001) LEFT PARENTHESIS MISSING.
An unbalanced right parenthesis was detected. No compilation was started.

E(00001) NO FILE TO BE COMPILED IS DEFINED.
No filename was specified in the command. No compilation was started.

E(00001) UNABLE TO COMPILE MORE THAN 32 FILES
IN ONE RUN. PLEASE SPLIT YOUR REQUEST.
More than 32 filenames were specified in the command. Enter in groups of less than 32 in two or more commands. No compilation was started.

E(00001) AT LEAST ONE OF THE FILES TO BE COMPILED
DOESN'T EXIST OR DOESN'T HAVE A 'FORTRAN'
TYPE NAME.
"filename FORTRAN *" was not found in the file directory.

E(00001) AT LEAST ONE OF THE FILES TO BE COMPILED HAS
LOGICAL RECORD LENGTH DIFFERENT OF 80 BYTES.
Recreate the file with 80-byte records. The EDIT command truncates overlength records to 80 bytes. No compilation was started.

E(00004)
Possible errors were detected in the source program, but successful execution is possible.

E(00008)
Errors were detected in the source program. Execution may fail.

E(00012)
Serious errors were detected in the source program. Execution is impossible.

E(00016)
Terminal errors were detected in the source program. Compilation was terminated.

E(00016) ERROR WHEN LOADING THE IEYFORT MODULE.

Loading of the compiler failed; no compilation was started. Retry the command.

E(00032) COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
CMS canceled processing. Correct the source program, or specify the GO option.

SEQUENTIAL I/O

All sequential files used or created by FORTRAN programs have file identifiers in the following format:

Filename	Filetype
FILE	FTxxFyyy

xx is the data set reference number, from 01-14.

yyy is a sequence number, beginning with 001 used to distinguish multiple files under the same data set reference number.

If a file is being created by a FORTRAN program, the filemode is P1. For input to a FORTRAN program, any filemode of P is accepted.

With the exception of terminal input and output, all files are kept on the permanent disk or on tape. Existing input files must conform to the record formats described below. Output files are created by the FORTRAN program, and need not be defined before execution. If output files already exist, the new output is appended to the existing file.

Data set reference number 5 is reserved for terminal input records of 80 characters or less. Number 6 is reserved for terminal output records of 120 characters or less. The terminal is also addressed by statements of the form "READ b,list" and "PRINT b,list" where b is a FORMAT statement number. The FORMAT for a PRINT statement must allow a leading space for a carriage-control character, or the first character of the record is lost. The carriage-control character does not have to be filled in, however. Output records generated by a statement of the form "PUNCH b,list" are placed in a file on the permanent disk under the identifiers FILE FT07F001 P1. Actual punching out of cards may be performed later with the OFFLINE PUNCH command.

Data set reference numbers 11 and 12 are reserved for tape I/O. Number 11 corresponds to TAP1 at virtual address 180, and number 12 corresponds to TAP2 at virtual address 181. Before number 11 and/or 12 are used, virtual 180 and/or 181 must be attached to the user's virtual machine configuration, or I/O errors occur.

With the exception of data set reference numbers 6,8,12, and 14, which allow 133-character records, and data set reference number 9, which allows 140-character records, all files must contain 80-character fixed-length records. The implied record format and device for each data set reference

number is shown in Figure 33.

The sequence field of the filetype is always 001 unless multiple files are referenced under the same data set number. There is no limit to the number of files which may be created or referenced under the same number, but only one may be referenced at a time. The first used must have the filetype FTxxF001, the second FTxxF002, etc. The END FILE statement closes the file currently in use, and the next READ or WRITE specifying the same data set reference number refers to a file with a reference number one larger. The REWIND statement "repositions" the files to the first one used in that program under that data set reference number (see Figure 34). The BACKSPACE statement is supported under CMS for tape data sets only.

A facility for defining a correspondence between a CMS file with unique identifiers and a FORTRAN data set reference number has been provided in the DEFINE subroutine. Refer to the description in SYSLIB TXTLIB.

Data Set Ref. No.	External Filetype	Record Length	Internal Reference
1	FT01Fyyy	80	READ (a,b) list*
2	FT02Fyyy	80	WRITE (a,b) list
3	FT03Fyyy	80	END FILE a
4	FT04Fyyy	80	REWIND a
5	---	80	READ (5,b) list or READ b, list
5	FT05Fyyy	80	WRITE (5,b) list
6	---	120	WRITE (6,b) list or PRINT b,list
7	FT07Fyyy	80	same as 1-4 or PUNCH b,list
8	FT08Fyyy	133	same as 1-4
9	FT09Fyyy	140	same as 1-4
10	FT10Fyyy	80	same as 1-4
11	---	80	same as 1-4
12	---	133	same as 1-4
13	---	80 blocked 10	same as 1-4
14	---	133 blocked 10	same as 1-4

*a is the data set reference number, b is the FORMAT statement number, and list is a series of variable or array names.

See IBM System/360 FORTRAN IV Language for additional acceptable I/O statement formats.

Figure 33. Summary of record formats and I/O statements for sequential FORTRAN files

Notes to Figure 33

a. The file identifiers for each of the files are

```
FILE FTxxFyyy
```

where xx is the data set reference number, and yyy is the sequence number.

b. The sequence number is 001, except when multiple files are referenced under the same data set reference number.

c. No file identifiers are shown for reading data set reference number 5 or writing number 6, since these numbers address the terminal for input and output, respectively. No file identifiers are shown for data set reference numbers 11, 12, 13, and 14, as these numbers address virtual tapes 180-183, respectively.

* Data set reference numbers 13 and 14 should not currently be used, as there are no definitions in CMS for tapes 182 and 183.

FORTRAN STATEMENT	FILE NAME	FILE TYPE	MODE
REAL*8(5)			
100 FORMAT(5(G10.8,6X))			
5 DO 20 I=1,45			
10 READ (4,100) B	FILE	FT04F001	P1
.			
.			
20 WRITE (3,100) B	FILE	FT03F001	P1
30 END FILE 3	FILE	FT03F001	P1
40 READ (4,100,END=50) B	FILE	FT04F001	P1
.			
.			
44 WRITE (3,100) B	FILE	FT03F002	P1
GO TO 40			
50 END FILE 3	FILE	FT03F002	P1
52 REWIND 3			
.			
.			
75 READ (3,100) B	FILE	FT03F001	P1
.			
82 PRINT 200, B	(terminal output)		
.			
STOP			

Figure 34. Files referenced by sequential FORTRAN I/O statements

Notes to Figure 34

Statement 10 reads the first 45 records of the existing file FILE FT04F001 P1. At statement 20, these records are placed in a new file FILE FT03F001 P1. This file is closed by the END FILE statement at 30, and the next time data set reference number 3 is used, at statement 44, a second new file is created: FILE FT03F002. After the REWIND statement, data set reference number 3 is again associated with the first file created: FILE FT03F001. Note that the PRINT statement requires a different FORMAT statement, which allows for a carriage-control character.

DIRECT ACCESS I/O

All direct-access files used or created by FORTRAN programs have file identifiers in the following format:

Filename	Filetype
FILE	DAxx

xx is the data set reference number, from 01 through 08.

If the file is being created by a FORTRAN program, the filemode is P1. For input to a FORTRAN program, any filemode of P is acceptable.

Direct access refers only to those files which are used with the FORTRAN language DEFINE FILE statement. (Note the distinction between the CMS library DEFINE subroutine in SYSLIB, TXTLIB which is referenced by a CALL DEFINE statement, and the FORTRAN language statement DEFINE FILE.) Files used sequentially are not considered direct-access files, even though they reside on disk.

Unlike the sequential data set reference numbers, the direct-access number does not imply any record length. This information is supplied by the DEFINE FILE statement within the FORTRAN program. All files are on the permanent disk. The same data set reference number may not be used for both a sequential and a direct-access file in the same program, nor may a single file be referenced by both methods in the same program. Different access methods may be used for the same file by different programs, provided the file identifiers are changed.

The number of records specified in the DEFINE FILE statement should be realistic. If a new file is being created, the specified number of records are blanked out on the permanent disk before the first record is written. Specifying an unnecessarily large number of records wastes disk space.

Although the FIND statement is supported, there is no need to use it in a time-sharing environment. I/O overlap is achieved through sharing of CPU time among the virtual machines. Use of the FIND statement actually slows down execution of the FORTRAN program slightly, since two operations must be carried out instead of one.

An example of direct-access I/O is shown in Figure 35.

FORTRAN STATEMENT	FILE-NAME	FILE-TYPE	FILE-MODE
100 FORMAT(I6,2X,A20,4G13.6)			
DEFINE FILE 1(200,80,E,R3)			
.			
.			
5 READ(1'J,100) MNNO,NAME,TOTS			
.	FILE	DA01	P1
30 WRITE(1'J,100) MNNO,NAME,NWTOTS			
.			
STOP			

Figure 35. File referenced by direct-access FORTRAN I/O statements

Note to Figure 35

The DEFINE FILE statement describes a file of 200 80-character records. If this file had not existed before program execution, 200 records of 80 blanks would have been written on the permanent disk. Statements 5 and 30 read and write the Jth record, where J has been assigned an integer value less than 201.

TERMINAL OUTPUT

Terminal output can be produced in either of two ways in FORTRAN programs:

(1) WRITE (6,FS) A,B,C
(2) PRINT FS,ABC

where FS is a format statement number.

In versions of CMS, up to 1.58 alphameric information printed at the terminal by using either the A-format or the Hollerith format had the first character of each line blanked out. This has now been modified, and the first character is used as a format control character in a manner analogous to that used in OFFLINE PRINTCC. This control character has the following meaning:

Character	Action
' '	line, carriage return
'+'	line
'0'	carriage return, line, carriage return

Note. Lines not followed by a carriage return have the line that follows them typed immediately behind them. If blanks are required, nonprinting characters should be included.

FORTRAN FILES

The defined FORTRAN logical files are as follows:

Logical file	Record description
1-4	80-character records
5	80-character sysin records
6	130-character sysout records
7	80-character records
8	133-character records with carriage control

(To print logical unit file 8, the OFFLINE PRINTCC command must be issued.)

The subroutine DSDSET can be called to allow a user to change the record format and the logical record length for FORTRAN disk files. Thus, a user is no longer confined to records of 80, 130, or 133 bytes in length (see the DSDSET Subroutine for a description of DSDSET in SYSLIB TXTLIB).

A facility has been added to change the identifier FILE FTxxFyyy required for FORTRAN disk file reads and writes. A call to the DEFINE subroutine is required to change the correspondence between a CMS file and a FORTRAN data set reference number. Refer to the writeup on the DEFINE routine.

FORTRAN write statements can be used to create disk files with a name of FILE FT0xF00y, where x is the logical unit number and y is the logical file number. The first time a write is issued to logical unit x, logical file 1 is written. After an ENDFILE to logical unit x, subsequent writes to logical unit x before a REWIND will write into file FT0xF002. This file is a separate disk file from file FT0xF001. Additional logical files can be written by issuing an END FILE to the previous logical file, and then continuing to write onto logical unit x. If a file already exists when the write statement is issued, the lines written are appended to the existing file. To append information to an existing logical file 2, first

write onto logical file1 and issue an END FILE x, or

read the file until the end file condition is reached,

then write into logical file2, which appends to the data on the end of the existing file.

After a rewind to a logical unit, subsequent writes overwrite the existing file. Only logical file 1 (that is, file FT0xF001) can be overwritten. There is no way to overwrite into logical file 2, etc. An overwrite does not shorten the file length; thus, information in a file which is not overwritten remains in the file. Writing over an

existing file can lengthen the file, and thus eliminate all the original information in the file. To write a completely new file, an existing file must first be erased, either by issuing the ERASE command or by calling the ERASE routine.

A tape facility for FORTRAN programs is available where logical units 11-14 are the standard logical tape units. These units correspond to symbolic devices TAP1-TAP4, and to virtual devices 180-183. A tape file can be written in one of the following five tape formats:

type 1: fixed-record size, unblocked
type 2: fixed-record size, blocked
type 3: variable-record size, unblocked
type 4: variable-record size, blocked
type 5: undefined-record size, no blocking

The default settings are as follows:

Virtual Device	Symbolic Device	Logical Unit	Block Size	Format Type	Logical Record Length
180	TAP1	11	80	1	80
181	TAP2	12	133	1	133
182	TAP3	13	800	2	80
183	TAP4	14	1330	2	133

Unless otherwise set by a call to TAPSET, the mode setting for 7-track tapes is for 800 bpi, odd parity, converter on, and translator off. For 9-track tapes, the mode setting is ignored. Refer to TAPSET Subroutine.

Note. TAP3 and TAP4 are not currently defined in CMS, therefore do not use logical units 13 and 14.

I/O FORMAT CONVERSION

A FORTRAN reread facility is available to perform a core I/O format conversion. To use this facility, a call to the REREAD routine must be made to specify the logical unit number. The logical unit may be any unit from 1 to 99 except unit 5, 6, or 7. A WRITE statement must be issued before the READ statement for rereading the specified logical unit number. The FORTRAN statement

```
CALL REREAD (n)
```

sets the reread unit to logical unit n with a default record size of 140 bytes. This may be changed by specifying the record size as a second parameter in the call to REREAD, for example, CALL REREAD (99,80). Refer to the subroutine description in the REREAD Subroutine writeup. To read the record from the reread unit a second or subsequent time, a REWIND n statement must be executed before the READ statement. If a reread is issued without executing a REWIND statement, an END OF FILE condition results. Any

input/output statements for other logical units can be issued between a write and a read on the reread unit. The reread unit or the blocksize can be changed by another call to the REREAD routine.

Notes:

a. Each FORTRAN file can contain any number of routines to be computed.

b. A CMS namelist facility is available for obtaining input to a FORTRAN program in free format without specifying variable names. Refer to the writeup on NLSTON/NLSTOF for a more detailed description of this namelist facility.

c. The supported data set reference numbers, device assignments, or record formats at a particular installation may vary from those described. Check with the responsible system programmer.

d. Since different FORTRAN programs using the same data set reference number reference the same CMS file identifiers, files should not be left on the permanent disk under FORTRAN format identifiers. If a FORTRAN program is to be executed repeatedly, it is advisable to create an EXEC file renaming execution-time files both before and after execution. An example of such a file might be MYSTRUP EXEC P1, containing

```
ALTER MSTR LIST P1 FILE DA02 P1
ALTER CHANGES LIST P1 FILE FT04R001 P1
$ FORTMSTR
ALTER FILE FT04F001 P1 CHANGES DONE P1
ALTER FILE DA02 P1 MSTR LIST P1
```

The command $ MYSTRUP then renames the files, executes the FORTRAN program, and changes the identifiers again on completion.

e. The FORTRAN subroutines are found in SYSLIB TXTLIB. Descriptions of the subroutines can be found in the section called SYSLIB TXTLIB. The extended error messages in SYSLIB TXTLIB give the user a traceback with registers 14,15,0, and 1 and the entry point. A standard fixup is taken and execution attempts to continue. At program completion a summary of errors is typed to the terminal. The nonerror message subroutines can be found in CMSLIB TXTLIB. These messages give the error number and a brief description. To use this, rather than the standard error-message subroutines, GLOBAL T CMSLIB SYSLIB must be issued.

PLI

Purpose:

The PLI comma
language into
and diagnostic

Format:

```
----------
|   PLI
----------
```

fname1...fname

option 1...opt

Options:

SIZE	nnnK	s
SIZE	128K	(
		p
LC	nn	l
LC	50	
O	nn	l
O	00	s
SM	cc,cc	s
SM	01,72	s
S	includes	
NS	suppress	
L	includes mnemoni	
NL	suppress	
A	includes	
NA	suppress	
E	includes file.	
NE	suppres Dictior	
X	include file.	
NX	suppress	

The following options relate to the CMS control of output. Abbreviation for each option appears in par

PUNCH (PU) outputs the TEXT file onto the offli Option D must be in effect.

PRINT (P) outputs the LISTING file onto the printer. A copy of the LISTING fil placed onto the user's disk.

NOPRINT (NP) no LISTING file is produced.

NODIAG (NDG) compiler diagnostics are not tvped user's terminal.

If neither P nor NP is specified, the LISTING file onto the user's permanent disk.

Usage:

The PLI command compiles files of PL/I source lang machine-language object code. Input files must filetype of PLI and a record length of 80 character

The options governing compiler operation and ou specified in any order in a set of parentheses foll last filename. Any combination of options is val conflicting options are specified, the last specifi is used. Unsupported or misspelled options are One set of options governs all compilations perform command. Each of the options has a default value selected when none is specified. These defa underlined in "Options" above.

For a complete discussion of PL/I usage, refer System/360 Operating System PL/I(F) Programmer (C28-6594) and IBM System/360 Operating Syst Subroutine Library Computational Subroutines (C28- introduction to PL/I is provided in Manual C28-680 Primer.

Responses:

None

Error Messages:

E(0004)
Warning messages have been included in the LISTI Successful execution is probable.

E(0008)
Error messages have been included in the LISTING f compilation was completed with errors, and execu fail.

E(00012)
Severe error messages have been included in the LISTING file. Successful execution is improbable.

E(00016)
Compilation terminated abnormally. Successful execution is impossible.

E(00026) FILE TO BE COMPILED, UNDEFINED.
No file with the specified filename and filetype of PLI can be located on disk. Check to see that such a file exists and then reissue the PLI command.

E(00026) FILE(S) TO BE COMPILED, NOT SPECIFIED.
No filename(s) was(were) specified in the PLI command. Reissue the command in its proper format.

E(00026) FILE HAS INCORRECT RECORD LENGTH.
The source code to be compiled is not in 80-byte record format and cannot be processed.

E(00026) SYNTAX ERROR IN OPTION LIST.
Verify format of option list in COMMAND line.

E(00027) CMS NUCLEUS ERROR.
ReIPL the CMS system disk.

E(00028)
An error was encountered in attempting to punch the TEXT file for the program being compiled.

COMPILATION NOTES:

a. The main procedure must be compiled with the OPTIONS(MAIN) option stated.

b. Compiler diagnostics must be examined carefully -- do not attempt to execute a program that has not compiled successfully.

c. Conversion subroutines are noted as "warnings" in the compiler diagnostics. This does not indicate an error and should not affect the execution of the procedure; it merely notifies the user of costly conversions. The warnings may be suppressed by the FE option.

d. The compiler produces a file "name TEXT P1" from the input file "name PLI P1", except in the event of "terminal" compiler errors.

PL/I LIBRARY

CMS uses the PL/I Version 4 Subroutine Library. The PL/I library is called PLILIB TXTLIB. Three additional subroutines have been added to the library: IHECMS, IHECLOK, and IHEFILE.

LOADING A PL/I PROGRAM

Before loading a PL/I program, it is necessary to designate that the PL/I library is to be used (the default library is the FORTRAN library called SYSLIB). This may be done immediately before loading, or automatically at login via the PROFILE EXEC, by the following CMS command:

```
GLOBAL TXTLIB PLILIB
```

Warning: If the loader indicates that names of the form IHExyz are undefined, the PL/I library was not correctly designated.

EXECUTING A PL/I PROGRAM

A PL/I program may be loaded and executed in a similar manner to all the other language processors. The only restriction is that the START entryname, if used, must be IHECMS which is a special initialization routine that transfers control to the user's main program. This routine is automatically invoked under all other circumstances.

The three common techniques are

```
LOAD prog1 <prog2...> (XEQ)

LOAD prog1 <prog2...>
START
```

```
LOAD prog1 <prog2...>
GENMOD prog1
.
.
.
prog1
```

PASSING PARAMETERS TO A PL/I PROGRAM

The CMS Command Processor automatically converts all parameters on a command line into a series of 8 character fields (left-justified and padded with blanks, or truncated as necessary). These parameters may be passed to the PL/I main procedure that is illustrated in the form below.

```
prog1:  PROCEDURE (PARMS) OPTIONS (MAIN);
        DECLARE PARMS CHAR(*) VARYING;
```

When using parameter passing, only the two loading techniques described below may be used.

```
LOAD prog1 <prog2 ...>
START IHECMS parm1 parm2 parm3 ...

LOAD prog1 <prog2 ...>
GENMOD prog1
.
.
.
prog1 parm1 parm2 parm3 ...
```

Note. The varying character string PARMS contains the concatenation of the specified parameters. Its length is always a multiple of eight (or null, if no parameters are specified).

TERMINAL I/O

Terminal input/output can best be performed by using the PL/I DISPLAY or DISPLAY/REPLY commands. Though the DISPLAY/REPLY facilities deal only with character strings, all possible uses can be accomplished by the following techniques:

1. Character I/O
   For simple character strings, the DISPLAY/REPLY facilities are convenient to use.

   a. Output message only:
      DISPLAY (message);
   Example: DISPLAY ('HELLO THERE');

   b. Output message followed by response:
      DISPLAY (message) REPLY (response);
   Example: DISPLAY ('ENTER YOUR NAME') REPLY (INPUT);

where INPUT is declared to be a character variable.

c. Input only:
```
DISPLAY (' ') REPLY (response);
```
Example:
```
DISPLAY (' ') REPLY (input);
```

where INPUT is declared to be a character variable.

2. Nonformat Simple I/O
For input/output that is not a simple character string, the PL/I conversions often work adequately.

Examples:

a. DISPLAY (VARIABLE);
Where VARIABLE can be any simple variable (that is, integer, floating-point, character string, etc.), it is converted to a character string by the default format, and printed.

b. DISPLAY ('THE VALUE OF N IS' || N);
The value of N is converted to a character string, concatenated with the message "THE VALUE OF N IS", and printed.

c. DISPLAY ('ENTER VALUE OF N') REPLY (INPUT);
N = INPUT;
The number entered is accepted as a character string and converted to a number by the N=INPUT statement. If INPUT contains an illegal representation for a number, a CONVERSION error results.

3. Full PL/I Format Capabilities
The full PL/I format capabilities can be used by creating formatted strings using the GET STRING and PUT STRING commands. Regular PL/I input/output utilizes the LIST or EDIT mode format control.

a. A typical output statement might be
```
PUT FILE(SYSPRINT) EDIT (A, B, C) (F(8,0), F(8,0), F(7,2));
```

To accomplish the same function to the terminal
```
PUT STRING(OUTPUT) EDIT (A, B, C) (F(8,0), F(8,0), F(7,2));
DISPLAY (OUTPUT);
```
where OUTPUT is declared to be a sufficiently long character string.

b. A typical input statement might be
```
GET FILE(SYSIN) EDIT (A, B, C) (F(8,0), F(8,0), F(7,2));
```
To accomplish the same function to the terminal:
```
DISPLAY (' ') REPLY (INPUT);
GET STRING (INPUT) EDIT (A, B, C) (F(8,0), F(8,0), F(7,2));
```
where INPUT is declared to be a sufficiently long character string.

I/O VIA FILES

Extensive input/output file capabilities are available in PL/I. At present, CMS has very limited support for PL/I file I/O. Those capabilities are briefly described here.

1. File Names

   A reference to a file in PL/I requires the use of a filename. A filename of DATA in PL/I corresponds to a CMS file named FILE DATA. Thus, the standard PL/I files SYSPRINT and SYSIN correspond to the CMS files FILE SYSPRINT and FILE SYSIN, respectively.

2. File Declarations

   ALL files must be declared, since PL/I learns the characteristics of files under OS/360 from either PL/I declarations or DD cards, and CMS does not use DD cards. The general form for a declaration is

```
DECLARE filename FILE <STREAM/RECORD> <EXTERNAL/INTERNAL>
                <PRINT> ENVIRONMENT (F(n));
```

   where n is the length of each record in the file.

   Recommended declarations for SYSPRINT and SYSIN are

```
DECLARE SYSPRINT FILE STREAM PRINT ENVIRONMENT (F(121));
DECLARE SYSIN FILE STREAM ENVIRONMENT (F(80));
```

   Since SYSPRINT is used by PL/I to record execution errors, it is necessary that this file be declared even if it is not explicitly used in the program. To insure that this condition is satisfied, the following default declaration is used if there is no user declaration for SYSPRINT:

```
DECLARE SYSPRINT FILE STREAM PRINT ENV (F(80));
```

3. File Operations

   At present only SEQUENTIAL files can be processed under CMS, though either the STREAM or RECORD form can be used. The following commands may be used:

   a. OPEN
      There is no real need for using OPEN, as the first GET/PUT or READ/WRITE to a file causes it to automatically open. Since the OPEN routine is dynamically loaded, there is usually a noticeable pause when an OPEN occurs.

   b. GET/PUT
      All forms of GET/PUT can be used-- of course, the file must have been declared as STREAM.

   c. READ/WRITE
      Only the SEQUENTIAL forms of READ/WRITE can be

used--the file must have been declared as RECORD.

d. CLOSE
Before a file is switched from GET to PUT, PUT to GET, READ to WRITE, or WRITE to READ, it must be CLOSED.

Note. Files are not automatically erased; therefore writing into an already existing file results in appending records to the end of the original file or rewriting the records depending upon the position of the file's Write Pointer as a result of the file's previous usage. The PL/I CLOSE statement resets the Write Pointer to the beginning of the file so that later PUT or WRITE statements will overwrite the previous records. Most CMS commands, such as EDIT, leave the Write Pointer at the end of the file so that later PL/I PUT or WRITE statements will cause records to be appended to the end of the file. There is one exception--the file SYSPRINT is automatically erased at the beginning of execution.

Warning. Although the OPEN statement is not needed, due to the significant overhead associated with implicit OPENs (caused by GET, PUT, READ, WRITE), it is recommended that all files be OPEN'ed explicitly, at the beginning, by a single OPEN statement. The form of the OPEN statement is

```
OPEN FILE(file1), FILE(file2), FILE(file3), ...;
```

Of course, it is not necessary nor advisable, to OPEN any files that are not used in the program, including SYSIN and SYSPRINT.

ERROR RECOVERY

PL/I attempts to catch all execution errors such as invalid conversion, program interrupts, and input/output errors, and prints an appropriate message on SYSPRINT. The message that is placed into the FILE SYSPRINT is also transferred to the user's terminal.

Occasionally the message Interrupt in Error Handler occurs. This means that PL/I has been unable to recover from an error condition. There are three main causes for that phenomenon: (1) the compiler has generated incorrect code or there is a malfunction of a library routine--that is, system error, (2) a subscript has exceeded its bounds and destroyed some arbitrary area of memory--usually a part of the program, or (3) parameters have been passed incorrectly (that is, scalar instead of array) causing incorrect addresses to be used. The second problem can be avoided by enabling the SUBSCRIPTRANGE check by making the first statement of each subroutine

```
(SUBSCRIPTRANGE):
```

PL/I catches all program interrupts, including the breakpoints set by the DEBUG routine. Therefore, if you wish to set breakpoints in a PL/I environment, it is necessary to disable the SPIE that PL/I uses to trap all interrupts. The SPIE can be disabled by using the special CMS library function IHESPOF (SPIE off), and enabled by the function IHESPON (SPIE on). Breakpoints should be placed so that they are triggered after the SPIE has been disabled.

OTHER LIMITATIONS

At present several PL/I system-dependent capabilities malfunction under CMS--the TIME and DATE built-in functions and certain types of ON-conditions do not operate correctly. There may be other similar temporary limitations. The user should try to avoid using these facilities.

SYSIN/SYSPRINT TO USER'S TERMINAL

Occasionally, it is desirable to use the full PL/I GET/PUT LIST, EDIT, and DATA facilities with the user's terminal. Furthermore, it is often desirable to have programs that can use the terminal for testing with small quantities of data, but later use files for large-scale runs. A facility has been added to CMS-PL/I to specify that the SYSIN/SYSPRINT be directed to the user's terminal rather than files. It is still necessary for the user to declare the SYSIN and SYSPRINT files, whether he is using files or the terminal.

There are two mechanisms available for activating this mode of operation.

1. If the first parameter passed to a PL/I program is (TYPE), it is trapped by the interface routine and removed from the parameter list, and the typewriter SYSIN/SYSPRINT mode is turned on. The remaining parameters, if any, are passed to the PL/I main procedure.

   Examples:

   ```
   LOAD prog ....
   START IHECMS (TYPE)
         or
   prog (TYPE)
   ```

   where prog is a MODULE created by GENMOD.

2. It may be desirable to selectively switch SYSIN/SYSPRINT from files and terminal. The CMS-PL/I library routine IHEONNL (online) causes all future SYSIN/SYSPRINT requests to refer to the user's console; the routine IHEOFFL (offline) causes all future SYSIN/SYSPRINT requests to refer to the files FILE SYSIN and FILE SYSPRINT. The most recent IHEONNL or IHEOFFL negates the effect of any previous mode.

Note that the normal precautions concerning GET/PUT apply. In particular, if both SYSIN and SYSPRINT are to be used, it is recommended that they be opened simultaneously by the statement

```
OPEN FILE(SYSIN), FILE(SYSPRINT);
```

Notes:

a. SYSIN and SYSPRINT are STREAM input/output files; as a result, they do not always respond to the terminal as the user might expect. In particular, a PUT statement merely places information into an output buffer; the buffer is not actually printed until the buffer becomes full, or, until a PUT with the SKIP option is encountered. Therefore, the sequence

```
PUT LIST ('READY');
PUT SKIP LIST ('ENTER ARGS');
GET LIST (A,B,C);
PUT SKIP LIST ('THANK YOU');
```

prints 'READY' and then request A,B, and C input before printing 'ENTER ARGS' (which is still in the current print buffer). The 'ENTER ARGS' message is not printed until after reading the arguments!

Ways to overcome this problem are use DISPLAY and DISPLAY/REPLY instead of GET/PUT, or place a PUT SKIP; after each PUT statement.

b. If an explicit OPEN is used for SYSIN, the stream input mechanism of PL/I will request the first input line immediately even though a GET statement has not been encountered yet. That input line is saved and used for later GET statements. This is not usually a serious program but can be confusing if output was expected prior to entering input. Therefore the sequence:

```
OPEN FILE (SYSIN), FILE (SYSPRINT);
PUT LIST ('ENTER ARGS');
PUT SKIP;
GET LIST (A,B,C);
```

would unlock the keyboard to accept an input line before printing 'ENTER ARGS' even though the PUT SKIP was used (see Note 1). This is due to the explicit OPEN for SYSIN.

For very similar reasons, the CALL IHEONNL statement, if used, must precede the OPEN FILE (SYSIN), otherwise the system will attempt to read the first line from FILE SYSIN.

There are several ways to overcome this problem: (1) use DISPLAY and DISPLAY/REPLY instead of GET/PUT, or (2) do not explicitly open SYSIN but let first GET statement cause

implicit open.

PL/I SUBROUTINES

Three subroutines have been added to the PL/I library for use under CMS. Two of these--IHECLOK and IHEFILE---were written to assist users attempting to write monitor-type programs in PL/I.

IHECMS--PL/I Initialization Routine

Purpose:

IHECMS performs the CMS-dependent initialization and passes parameters to the primary PL/I initialization subroutine, IHESAP.

Calling Sequence:

IHECMS is automatically loaded with the PL/I initialization subroutines.

Internal Entries:

The following CMS PL/I library functions are entries within the IHECMS subroutine.

IHESPOF turns PL/I SPIE off
IHESPON turns PL/I SPIE on

IHEONNL SYSIN/SYSPRINT to terminal
IHEOFFL SYSIN/SYSPRINT to file

IHEDBUG transfer control to CMS DEBUG

IHECLOK--PL/I Clock Routine

Purpose:

IHECLOK reads the CP virtual chronolog clock and returns the date, time of day, elapsed virtual and total time.

Calling Sequence:

```
CALL IHECLOK ( CLOCK );
```

where CLOCK is a PL/I structure in the form illustrated below.

```
DECLARE 1 CLOCK,
     2 DATE CHAR(8),          /* date in form 01/21/69 */
     2 TIME CHAR(8),          /* time of day in form 21.14.34 */
     2 VIRTUAL_TIME FIXED BIN(31,0),
                   /* elapsed virtual time in timer units */
     2 TOTAL_TIME FIXED BIN(31,0);
                   /* elapsed total cpu time in timer units */
```

DATE and TIME contain actual slashes (/) and periods (.) as illustrated in the examples above. The PL/I SUBSTR function can be used to rearrange the DATE and TIME and/or remove the punctuation.

VIRTUAL_TIME and TOTAL_TIME are binary integers. They represent elapsed time charged for cpu usage running under problem state (virtual time) and elapsed time charged for total cpu usage (total time). To convert from timer units to hundredths of seconds, divide by 768 or X'300'. Both times are increasing numbers; the normal 360 timer decreases.

Example:

```
TESTCLK:     PROCEDURE OPTIONS (MAIN);                    TES00010

     TESTCLK:     PROCEDURE OPTIONS (MAIN);
             DECLARE IHECLOK ENTRY (1, 2 CHAR(8), 2 CHAR(8),
                  2 FIXED BIN (31,0), 2 FIXED BIN (31,0));
             DECLARE 1 CLOCK STATIC,
                  2 DATE CHAR(8),
                  2 TIME_OF_DAY CHAR(8),
                  2 VIRTUAL_TIME FIXED BIN (31,0),
                  2 TOTAL CPU_TIME FIXED BIN (31,0);

     LOOP:
             DISPLAY ('BEFORE CLOCK.');
             CALL IHECLOK (CLOCK);
             DISPLAY ('AFTER CLOCK.');
             DISPLAY (DATE);DISPLAY (TIME_OF_DAY);
             DISPLAY (VIRTUAL_CPU_TIME);DISPLAY(TOTAL_CPU_TIME);
             GO TO LOOP;
     END;
```

IHEFILE--PL/I File Access Routine

Purpose:

IHEFILE converts PL/I file access requests into the corresponding CMS commands STATE, ERASE, FINIS, RDBUF, and WRBUF.

The user should be familiar with the operations of the above-named CMS file system routines, and the error codes produced by them.

Calling Sequence

```
CALL IHEFILE ( FCB ) ;
```

where FCB (File Control Block) is a PL/I structure in the form illustrated below.

```
DECLARE 1 FCB,

   2 COMMAND CHAR(8), /*CMS command desired */
   2 FILENAME CHAR(8) /* CMS filename */
   2 FILETYPE CHAR(8) /* CMS filetype */
   2 CARD_NUMBER FIXED BIN (31,0),
                 /* record number - RDBUF/WRBUF only */
   2 STATUS FIXED BIN(31,0),
                 /* CMS return code - 0 means OK */
   2 CARD_BUFFER CHAR(80);
                 /* 80 byte record to be read or written */
```

The 80-byte record need not necessarily be a single character string. The next 80 bytes of the structure (after STATUS) are used, whatever they may be. Therefore, binary integers, character strings, floating-point numbers, etc. can be used in the following ways:

(1) adding them to structure in place of CARD_BUFFER,

(2) defining another structure positioned on top of CARD_BUFFER, or

(3) converting all information into a concatenated character string and separating out on input by SUBSTR.

CARD_NUMBER must be set for RDBUF/WRBUF usage; it is ignored for STATE, ERASE, and FINIS. If sequential writing or reading is to be done, the PL/I program must increment the CARD_NUMBER or set it to zero. During writing, if the card already exists, it is replaced by the new card. If the card does not exist (that is, the file is being expanded), the position of the end-of-file is moved and the new card added to the file.

STATUS is the CMS return code for the last IHEFILE issued

using that FCB. For example, return code 12 from RDBUF indicates an attempt to read beyond the current position of the end-of-file (see example that follows).

Example

The example below does the following:

a. Erases the file TEST1 DATA if it already existed.

b. Creates a file TEST1 DATA consisting of 25 records; each record contains a character string representing the square of the card number (that is, the third card in the file contains the number 9).

c. The file is closed via FINIS.

d. Assuming the number of cards in the file is unknown, the program attempts to read through the file until the status indicates that the end-of-file was reached.

e. The file is then read backwards (that is, starting at the last card, then next to last, etc.).

f. The file is closed.

```
printf testfil pli

 TESTFIL:      PROCEDURE OPTIONS(MAIN);

          DECLARE 1 FCB STATIC,
               2 COMMAND CHAR(8),
               2 FILENAME CHAR(8) INITIAL('TEST1'),
               2 FILETYPE CHAR(8) INITIAL('DATA'),
               2 CARD_NUMBER FIXED BIN(31,0),
               2 STATUS FIXED BIN(31,0),
               2 CARD_BUFFER CHAR(80);

          COMMAND='ERASE';
          CALL IHEFILE( FCB );
          COMMAND = 'WRBUF';

          DO J = 1 TO 25;
               CARD_BUFFER = J*J;
               CARD_NUMBER = J;
               CALL IHEFILE( FCB );
               DISPLAY(J||J*J);
          END;

          COMMAND = 'FINIS'; CALL IHEFILE( FCB );
          COMMAND = 'RDBUF';

          DO J = 1 BY 1;
               CARD_NUMBER = J;
               CALL IHEFILE( FCB );
               IF STATUS ¬= 0 THEN GOTO END_OF_FILE;
               DISPLAY(CARD_BUFFER);
          END;

 END_OF_FILE:
          DISPLAY('STATUS = '||STATUS);
          JMAX=J-1;

          DO J = JMAX TO 1 BY -1;
               CARD_NUMBER = 1;
               CALL IHEFILE( FCB );
               DISPLAY(CARD_BUFFER);
          END;

          COMMAND = 'FINIS'; CALL IHEFILE( FCB );
          DISPLAY('THATS ALL.');
 END;

R; T=0.08/0.38 20.04.10
```

SNOBOL **

Purpose:

The SNOBOL command compiles source programs written in SNOBOL into SPL/1, and executes SPL/1 programs.

Format:

```
-------------------------------------------
|SNOBOL| filename  <(option1...optionN)>  |
-------------------------------------------
```

filename — specifies a files with the filetype SNOBOL to be compiled, or with the filetype SPL1 to be executed.

option1...optionN — are one or more of the compiler options described below.

Options:

OFFLINE — specifies that information normally placed in the LISTING file is also to be printed offline.

ONLINE — specifies that information normally placed in the LISTING file is to be typed out at the terminal.

NOLIST — suppresses the LISTING file, but does not affect either of the above options.

SPL1 — specifies that the file named with the command is an SPL1 file to be executed, and not a SNOBOL file to be compiled.

In addition to the options above, execution is also controlled by control cards within the source file. These cards must begin with a hyphen in column 1, and appear as shown below.

-LIST ON — resumes the listing of the SNOBOL source program in the LISTING file.

-LIST OFF — suppresses the listing of the SNOBOL source program in the LISTING file.

-SEQUENCE — causes columns 73-80 of the source file records to be ignored by the compiler, allowing card sequence numbers.

-EJECT — inserts a carriage-control character in the LISTING file to skip to a new page.

-DECK — generates a file filename PUNCH P1 containing the SPL/1 output in a special abbreviated format.

-NOGO suppresses execution of the compiled program.

-MEMBER=name identifies the following cards as an SPL/1 routine.

-DATA marks the end of input to the compiler. This card must be present, whether any data cards follow or not.

-ASSEMBLY OFF suppresses listing of SPL/1 programs in the LISTING file as they are loaded for execution. This is the default value.

-ASSEMBLY ON includes the listing of SPL/1 programs in the LISTING file as they are loaded for execution.

-TRACE causes all strings referenced by the user's program to be listed in the LISTING file.

Usage:

The SNOBOL command uses two separate programs to compile and execute SNOBOL programs. The SNOBOL compiler is itself a SNOBOL program that translates SNOBOL into SPL/1, a more basic string-processing language. The SPL/1 assembler-interpreter executes SPL/1 programs interpretively (performing the requested operations for the user's program, rather than translating his program into machine language). The compiler and assembler-interpreter use several files, described below. In each case, "filename" is the filename specified with the SNOBOL command.

"filename SNOBOL P1" is the input to the SNOBOL compiler. It may consist of a SNOBOL program or program and subroutines, a mixture of SNOBOL and SPL/1 programs, or entirely of SPL/1 programs which have already been compiled. SNOBOL subroutines must begin with a SUBROUTINE statement and end with an END card. SPL/1 programs must be preceded by a -MEMBER= statement to be handled by the compiler. A -DATA card must close input to the compiler, whether data cards follow or not. The SNOBOL file may also contain the other control cards listed above.

"filename SPL1 P1" is the output from the SNOBOL compiler, and is input to the SPL/1 assembler-interpreter. If subroutines are included in the SNOBOL compiler input, each generates a separate SPL1 file, with the subroutine name used as a filename. An SPL1 file may be executed without compilation by specifying the SPL1 option.

"filename LISTING P1" is a listing file created by both the compiler and the assembler-interpreter. According to the options specified and the control cards included, it may contain any, or all, of the following information:

A listing of the SNOBOL source program, including

diagnostic messages immediately following any errors detected

A listing of the SPL/1 program as it was loaded for execution.

Any output generated by a SNOBOL PRINT statement in the program.

A message explaining any error completion.

Output generated by the TRACE subroutine, if the program requested it.

"filename PUNCH P1" is created if a -DECK control card is encountered in the program. This file is similar to the SPL1 file, except that comment cards are deleted and a special abbreviated format is used. It is generally about one-third the size of the SPL1 file for the same program.

"filename anyname P1" is the general identifier used for files referenced by name from a SNOBOL program. "filename" is the name of the program, and anyname is the name used for the file within the program, (see "Input/Output" under "SNOBOL Programming").

If the compiler detects an error, a diagnostic message is placed in the LISTING file, and a HALT instruction is inserted in the SPL1 file. Compilation continues to the -DATA card, but execution is terminated at the HALT instruction.

Notes:

a. CMS SNOBOL differs in some significant ways from other SNOBOL implementations. "SNOBOL Programming" describes briefly the I/O subroutines of CMS SNOBOL, but does not attempt to define the language. The user should be familiar with the manual listed under "References".

b. SNOBOL accepts any of the 256 EBCDIC bit patterns as data, but names and labels are restricted to letters and numbers, and several installation-dependent special characters.

Responses:

The SNOBOL command gives no responses, unless the ONLINE option is specified. In this case, information normally placed in the LISTING file is also typed out at the terminal.

References:

CMS SNOBOL users should have a copy of CMS SNOBOL User's Manual.

Examples:

a. SNOBOL SORT4
The file SORT4 SNOBOL P1 is compiled into SPL/1. A listing is created as SORT4 LISTING P1, and the SPL/1 program is saved as SORT4 SPL1 P1. As soon as compilation is complete, the SPL/1 assembler-interpreter receives control to execute the program.

b. SNOBOL SORT4 (SPL1 ONLINE NOLIST)
This command executes the file SORT4 SPL1 P1, created in the previous example. ONLINE causes any output normally placed in the LISTING file to be typed out. NOLIST suppresses the placing of the same output in the LISTING file.

Error Messages:

E(00008)
This is a general error code returned by the assembler-interpreter for most errors. An explanatory message is found in the LISTING file.

E(00016)
There was insufficient room in core storage for an SPL1 file during loading, or more than three levels of subroutine nesting were attempted.

** SNOBOL is a Type III program available from the IBM Program Information Department.

SUBROUTINES

CMS SNOBOL includes a subroutine feature, which may be used in two ways. System subroutines are provided, and are called by using the name of the subroutine as a SNOBOL statement. User-written subroutines are created and called by the SUBROUTINE, CALL, and RETURN system subroutines. The general format for using system subroutines is shown below.

```
-----------------------------------------------
|  label  subname(arg1,...,argN)   <(where)>  |
-----------------------------------------------
```

label — is an optional statement label.

subname — is the name of the subroutine.

arg1,...,argN — are string names or literals passed to the subroutine.

where — is an optional statement label specifying the subroutine statement at which execution is to begin.

No blanks may separate the subroutine name and the left parenthesis. The system subroutines are described below, grouped according to usage.

INPUT/OUTPUT

PRINT(string) writes the string specified on the LISTING file. A literal of fewer than 63 characters may be specified instead of a string name.

READ(string) reads successive items from the SNOBOL input file into the string specified. These items followed the -DATA card in the input stream.

PUNCH(string) writes the string specified into "filename PUNCH P1", where filename is the name of the program. The string specified must be 80 characters or less in length. A literal of fewer than 63 characters may also be specified.

GET(string1,string2)
PUT(string1,string2)
CLOSE(string1)
EJECT(string1)
are used to access files on disk or the user's terminal. The filename of the file referenced is always the same as the name of the program being executed. "string1" specifies the filetype of the file referenced. This operand may also be expressed as a literal. For example, if TEST1 is a SNOBOL program

being executed, and the string DATA contains SAMPL, either of the following statements would read an item from the file TEST1 SAMPL P1:

```
LABL GET(DATA,NUMS)

LABL GET('SAMPL',NUMS)
```

GET and PUT both open files automatically, if necessary, but if the same file is to be written and read, a CLOSE must be issued between the operations. (This does not apply to the terminal.)

Except for the terminal and the LISTING file, record size is always 80 characters. The LISTING file has 120-character records, plus a carriage control character. Size of a terminal record is always the number of characters written or read. A literal of fewer than 63 characters may replace string2 in a PUT statement.

The terminal is accessed by specifying a filetype consisting of blanks. This may be done in three ways: by specifying a string containing from one to eight blanks, by specifying a literal blank, or by omitting the first operand. The following three statements all read one record from the terminal into the string IN:

```
LAB GET(TERMINAL,IN)
    GET(' ',IN)
    GET(,IN)
```

The first assumes that the string TERMINAL contains only a series of blanks. The CMS delete characters have their usual effect, and all input is translated to uppercase. A line consisting of only a carriage return causes an I/O error and program termination. If a whole line is deleted by ¢, another read is issued.

The EJECT statement causes a carriage-control character to be placed in a file, forcing a page eject when the file is printed. EJECT(LISTING) forces the LISTING file to skip to the top of a page. Any file for which an EJECT is the first statement issued has a carriage-control character prefixed to each item as it is written out. This character is returned as the first character of strings read back in from such a file.

SUBROUTINE GENERATION

SUBROUTINE(arg1,...,argN) must be the first statement of user-written subroutines. An END card must conclude the subroutine. The name by which the subroutine is called must be placed in the label field of the SUBROUTINE statement. This name is also used for the filename of the SPL1 file generated by the compiler for

the subroutine. The SUBROUTINE statement may also include an unconditional branch. An example of a SUBROUTINE statement is

```
SORT SUBROUTINE(STRING,FIELD,NUM)  /(PROCESS)
```

This statement causes the SNOBOL compiler to generate a file SORT SPL1 P1, which is loaded whenever this subroutine is called by the name SORT. On receiving control, execution would begin at the statement labeled PROCESS. If the transfer had been omitted, execution begins at the first statement of the subroutine. STRING, FIELD, and NUM are the strings received as arguments by the subroutine.

RETURN (dummy) is used to return control from a subroutine to the calling program. If RETURN is executed in a main program, all files are closed, and control is returned to the CMS Command evironment. "dummy" is a dummy string name which must be included.

LINKAGES

CALL(string,arg1,...,argN) is used to call a subroutine generated by the SUBROUTINE statement. "string" specifies a string containing the name of the called subroutine, or may be a literal specifying the callee. "arg1,...,argN" are the strings passed to the subroutine. Their names are independent of those used in the SUBROUTINE statement, and matching is done positionally. A CALL statement may also include a statement label, and an unconditional transfer specifying a return point. A call to the SORT routine, described above, might appear as

```
GOSRT CALL('SORT',TEXT,CHAR1,LENGTH)  /(SEM)
```

The strings TEXT1,CHAR1, and LENGTH are passed to the subroutines as STRING, FIELD, and NUM, respectively. On return, control goes to the statement labeled SEM. Only three levels of nested CALLS are permitted, but up to that depth, CALLs may be recursive.

XCTL(name) overlays the currently executing SPL/1 program with the specified SPL1 file, and starts executing the new program. "name" specifies a string containing the name of the called SPL1 file, or it may be a literal. No arguments may be passed.

DEBUGGING AIDS

TRACE_ON(dummy)
TRACE_OFF(dummy)
are used to turn the TRACE option on and off. The TRACE option may also be enabled by the -TRACE control card. "dummy" is a dummy string name which must be

present. When the TRACE option is in effect, the contents of each string are placed in the LISTING file each time it is referenced. The assembler-interpreter location counter value is included with each string. The counter value refers to statement numbers in the assembler-interpreter listing obtained when the -ASSEMBLY ON control card is used.

TIME(name)
returns the current timer value in hundredths of a second in the string specified. Under CP, the timer value is elapsed virtual CPU time since LOGON. It is useful for comparative timings of programs.

BRUIN **

Purpose:

The BRUIN command initiates the Brown University Interpreter.

Format:

```
-------------
|   BRUIN   |
-------------
```

Usage:

The BRUIN environment is entered. When the character > (greater than) is printed and the keyboard unlocks, the interpreter is ready to accept a command. A BRUIN command should then be entered immediately following the > character.

To leave BRUIN and return to the CMS command environment, issue the BRUIN command

CANCEL

References:

BRUIN was developed at Brown University, Providence, Rhode Island. For information on BRUIN and its commands, refer to the document CMS BRUIN User's Manual.

Responses:

BRUIN
As the result of a null line being entered, the interpreter is confirming that the user is in the BRUIN environment.

BRUIN READY
The BRUIN environment has been entered, and it is ready to accept commands.

R;T=xx.xx/xx.xx hh.mm.ss.xx
The CMS command environment has been entered, where the first xx.xx is the CMS CPU time used in seconds, and the second xx.xx is the total CPU time used.

For other responses, refer to the references above.

Error Messages:

None.

** BRUIN is a Type III program available from the IBM Program Information Department.

UTILITIES

This section contains those CMS commands which are utilitarian in function. They include SORT, which sorts a disk file: COMPARE, which compares two disk files, record by record; TPCOPY, which copies a tape; MODMAP, which types out the load map of an existing MODULE file; CNVT26, which converts 026 code to 029; and CVTFV, to convert fixed records to variable.

To dump files to tape or cards from disk, TAPE DUMP or DISK DUMP can be used, respectively. To load tape files selectively, issue TAPE SLOAD; otherwise issue TAPE LOAD to load through n tapemarks. To load card images from a tape created by OS/360, use OSTAPE.

To test terminal transmission, use ECHO. To print or type out disk files in hexadecimal, use DUMPD or DUMPF respectively.

Purpose:

The CNVT26 command converts a BCDIC (026) card-image file to EBCDIC (029), using the scientific character set.

Format:

```
-------------------------------------
| CNVT26    |   filename  filetype  |
-------------------------------------
```

filename filetype is the file to be converted.

Usage:

The specified file is searched for by the standard order of search. It is converted using the scientific character set, the old file is erased, and the new file is given the original name, type, and mode.

CNVT26 creates the work file BCDEBC UTILITY with the same filemode as the file being converted. If a file with the identifier BCDEBC UTILITY already exists, it is erased.

Example:

CNVT26 EASY FORTRAN
The file EASY FORTRAN is converted from BCDIC to EBCDIC, the old file is erased, and the new file is given the original name.

Error Messages:

E(00001) FILE NOT FOUND
The specified file does not exist.

E(00002) FILENAME(S) MISSING
Either the filename and/or the filetype were not specified with the command. Reissue the command correctly.

COMPARE

Purpose

The COMPAR

Format:

```
--------
| COMPAR
--------
```

Usage:

If the fil
correspond
from the f
the second
the end of

The maxim
characters

Responses:

EOF FILE n
The end o
either 1 o

NOT EOF FI
This mess
reached on

FILES HAVE
The files
size or th

PARAMETER
Insufficie

RECORDS AR
The record
cannot be

Error Mess

E(00001) F

E(xxxxx) F

Examples:

a. compare
The files
FORTRAN fi
variable; t

CVTFV

Purpose:

The CVTFV command converts a file with fixed-length r
to variable-length records.

Format:

```
-----------------------------------------
|   CVTFV   | filename filetype <(T)> |
-----------------------------------------
```

filename filetype is the file to be converted.

(T) specifies that all trailing blanks are to be de
Blanks and data in columns 73 to 80 are to be dele

Usage:

CVTFV is particularly useful for incorporation of card
into SCRIPT files. If (T) is specified and the file co
of 80-character records (a normal card file), bytes 73
each card image are deleted, regardless of con
Therefore, to convert card files without loss of infor
in columns 73-80, do not use the (T) option.

The standard order of search is used to locate the fil
old file is erased, and the new file is given the or
name, type, and mode.

CVTFV creates a work file with a filetype of CVTUT1 a
filename and mode of the specified file. If a file a
exists with the same identifier as the work file,
erased.

Example:

CVTFV USERGUID MEMO (T)
The file USERGUID MEMO is converted from fixed-
records to variable-length records. If the file co
80-character records, columns 73-80 are deleted.

Error Messages:

E(00001) FILE NOT FOUND
The specified file does not exist.

E(00002) INCORRECT FORMAT
A T was not specified between the parentheses.

E(00003) INCORRECT FORMAT
Either a filename and/or a filetype was not specifie
the command. Reissue the command correctly.

E(00004) FILE IS ALREADY VARIABLE
The specified file contains variable length records, therefore it cannot be converted.

DISK

Purpose:

The DISK command punches specified disk files on cards in a special format, and can reload these card files onto disk storage.

Format:

DISK	DUMP filename filetype <filemode>
	LOAD

DUMP specifies that a file is to be punched out.

filename filetype <filemode> specify the file to be punched.

LOAD specifies that one or more card files are to be read and saved on the user's permanent disk.

The DISK card format is

Columns	Content
1	a 12-2-9 punch
2-4	CMS
5	blank, or N for EOF
6-55	file data
56-57	blank
58-76	filename,type,mode
77-80	sequence number

Usage:

For a DISK DUMP operation, filename, filetype, and optionally filemode are specified. If the filemode is omitted, all disks are searched. Only one file is punched out by a command. The file may have either fixed or variable-length records. After all data is transferred, an end-of-file card is punched with an N in column 5. This card contains directory information, and must remain in the deck. The original disk file is retained, unless the mode is delete-after-reading.

The DISK LOAD operation reads any number of decks punched by DISK DUMP. File designations are obtained from the card stream, and may not be specified with the command. Any existing file with the same designation as one of these in the card stream is erased and replaced. DISK LOAD loads files only onto the P disk.

Files to be loaded by a DISK LOAD command must be read by

CP-67 as a single deck before the command is issued. An identification card with CP-67USERID in columns 1-10 and the user's identification starting in column 13 must precede the deck.

Notes:

a. Data is punched in Extended BCD Card Code.

b. Each file punched out is preceded by a CP-67 identification card containing the USERID.

Responses:

DISK DUMP gives no response except the Ready message.

filename filetype filemode
DISK LOAD gives this response for each file encountered in the input deck. The Ready message is typed after transfer is completed.

SYSTEM I/O ERROR
CP ENTERED, REQUEST PLEASE
This message indicates that an unrecoverable I/O error has occurred on a spooled direct-access device. ReIPL CMS and issue the DISK command again.

Examples:

a. DISK DUMP PROCS TXTLIB P5
The specified file is punched out. Each card contains CMS in columns 2-4, file data in columns 6-55, and the characters PROCS TXTLIB P5 in columns 58-76. The cards are sequenced 0001, 0002, etc. in columns 77-80. The last card contains an N in column 5.

b. DISK LOAD
If the deck produced in the previous example has been read by CP-67, the disk file PROCS TXTLIB P5 is erased. A new file is created with the data on the cards, and receives the designation PROCS TXTLIB P5. The following response is typed:

```
PROCS TXTLIB P5
```

The Ready message is then issued to indicate completion.

Error Messages:

```
E(00001)   CORRECT FORM IS:
           "DISK DUMP" FILENAME FILETYPE FILEMODE
           OR "DISK LOAD"
```

The format of the command was incorrect. No cards were punched or read. Issue the command again.

E(00002) FATAL PUNCH ERROR.
Indicates an I/O error on the card punch. This message should never occur under CP-67. Notify the responsible

system programmer.

E(00003) FATAL DISK ERROR.
AN I/O error on the disk occurred. A DISK DUMP operation may be retried. For a DISK LOAD operation, notify the operator to enter the card deck again before retrying the command. If the error recurs, notify the operator.

E(00004) FATAL READER ERROR.
Indicates an I/O error on the card reader or an empty reader. If the reader was not empty, notify the responsible system programmer.

E(00005) ILLEGAL CARD IN DISK LOAD DECK.
A card was encountered in the input to DISK LOAD that was not punched by DISK DUMP. A CP-67 output identification card may have been left in the deck, or the wrong deck may have been read. The deck being read at the time of the error has been flushed, and will have to be reread before retrying the command. No permanent file has been created.

E(00006) END CARD MISSING FROM DISK LOAD DECK.
Physical end of data for a DISK LOAD was encountered without reading the end-of-file card (N in column 5). The file created on disk does not have the correct designation. It may be accessed under (DISK) (TFILE) P3. Note that the mode is delete-after-reading, so an ALTER or COMBINE must be issued before inspecting the file.

Purpose:

The DUMPD command prints the contents of one direct-access record, specified by a CCHHRR address, in hexadecimal on the printer.

Format:

```
| UMPD | ccu    cc    <hh   <rr>> |
```

ccu is the device address,

cc hh rr is the cylinder, track, and record number to be printed

Usage:

The contents of the specified record is printed in hexadecimal on the printer. If cc hh rr is not specified, or if one record has been printed, DUMPD requests another record address. To terminate DUMPD, issue a carriage return with no characters on the line.

Responses:

CC HH RR
Specify another cylinder, track (head), and record number. If another record is not desired, issue a carriage return.

CORRECT FORM IS: DUMPD UUU CC HH RR
OR: DUMPD UUU WITH THE CCHHRR ENTERED ON REQUEST
No parameters were specified with DUMPD.

I/O ERROR ON DISK UNIT XXX,
CSW = xxx ... xxx, SENSE = xxx ... xxx
An I/O error occurred. Check to see if the record address is outside the file boundaries.

SIO CONDITION CODE 1
SIO ERROR ON DISK UNIT xxx,
CSW = xxx...xxx, SENSE = xxx...xxx.
An invalid device address was specified.

Purpose:

The DUMPF command types the contents of all or part of a specified file in hexadecimal.

Format:

```
-----------------------------------------------
|DUMPF | filename  filetype  <n1   n2  <n3>>    |
|      |                      *    *   80       |
-----------------------------------------------
```

filename filetype specify the file to be typed

n1 is the record number of the first record to be typed.

n2 is the record number of the last record to be typed.

n3 is the maximum number of characters of a record to be typed, if the records are to be truncated.

Usage:

The filename and filetype must be specified. If the first record number and last record number are omitted, or specified with asterisks, the entire file is typed out. An asterisk in the first record or end record fields specifies the beginning or the end of the file, respectively.

The number of bytes specified is printed or, if the number of bytes is not specified, the full record is printed. If a byte count is specified, the first record number and last record number fields must be filled (either explicitly or with asterisks).

The standard order of search is used to locate the file. In the case of files with duplicate filename and filetype, only the first file found is typed out.

Error Messages:

E(00001) CORRECT FORM IS: 'DUMPF' FILENAME FILETYPE STARTLINE ENDLINE LINE-LIMIT,
WHERE 'STARTLINE', 'ENDLINE', AND 'LINE-LIMIT' ARE OPTIONAL.

E(00002) DISK ERROR

E(00003) FILE NOT FOUND

DUMPREST

Purpose:

DUMPREST dumps the contents of the user's permanent disk onto tape, and can reload a disk from such a tape.

Format:

```
--------------
|  DUMPREST  |
--------------
```

A message is typed, requesting one of the following replies:

DUMP (D) specifies that the contents of the permanent disk are to be copied on tape.

RESTORE (R) specifies that the permanent disk is is to be formatted, and contents of the mounted tape are to be copied onto it.

Usage:

The user's tape must be mounted at virtual device address 181 before the command is issued. The disk address is always virtual 191. For both DUMP and RESTORE operations, the command positions the tape at the load point with a REWIND.

In the DUMP operation, data on the permanent disk is copied to the tape one track at a time, including all directories and pointers. A RESTORE operation copies these records back to a disk in exactly the same locations. The disk is automatically set to CMS format, so a new or different disk pack may be used.

Note:

RESTORE may not be valid if the location of the user's virtual disk 191 has been redefined since the DUMP operation.

Responses:

SPECIFY 'DUMP' OR 'RESTORE :
This message is typed, without a carriage return, as soon as the command is issued. When the keyboard unlocks, enter DUMP or RESTORE, or just D or R, to specify the operation. A reply error causes the response to be repeated.

DUMP/RESTORE MOVED nnn CYLINDERS.
THERE WERE nnn RECOVERABLE TAPE ERRORS.
This message is typed at the completion of the requested transfer. If the number of cylinders specified is not equal to the size of the virtual disk, an irrecoverable I/O error

caused early termination.

Examples:

a.
```
DUMPREST
SPECIFY 'DUMP' OR 'RESTORE': D
DUMP/RESTORE MOVED 005 CYLINDERS.
THERE WERE 000 RECOVERABLE TAPE ERRORS.
```
The contents of the user's permanent disk have been dumped to tape. No errors occurred.

b.
```
DUMPREST
SPECIFY 'DUMP' OR 'RESTORE': restore
DUMP/RESTORE MOVED 005 CYLINDERS.
THERE WERE 000 RECOVERABLE TAPE ERRORS.
```
The contents of the user's permanent disk have been replaced by the contents of the dump tape. No errors occurred.

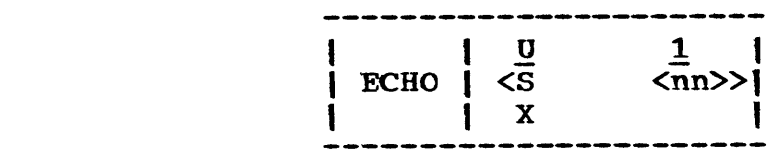
Error Messages:

None. I/O errors are retried ten times. If no recovery is made, the second response shown above is typed.

Purpose:

The ECHO command tests terminal transmission by retyping entered lines.

Format:



U The delete characters (@,¢) are interpreted, and any lowercase letters are changed to uppercase.

S Delete characters are interpreted but no lowercase to uppercase translation is made.

X No change is made in the line.

nn Number of times line is retyped. The default value is 1.

Usage:

When the ECHO command is issued, the Echo environment is entered and each line typed by the user is repeated by the terminal the specified number of times.

If no options, or an invalid option, are specified, U and 1 are assumed. An entered line is interpreted according to the option specified, if any, and then typed out n number of times. The keyboard is then unlocked to accept another line.

Control is returned to the CMS command environment by typing RETURN as the first word of a line with no leading blanks.

Notes:

a. If the X option is in effect, RETURN must be entered without error to be recognized (delete characters and linend are not interpreted).

b. The Ready message, following the RETURN request, indicates that the user has left the Echo environment and entered the CMS command environment.

Responses:

START CONSOLE TEST
The first line may be entered when the keyboard unlocks after this message is typed.

```
END CONSOLE TEST
The RETURN to CMS has been requested and accepted.
```

Examples:

a.
```
echo u 2
START CONSOLE TEST
echo retypes any entered lines,
ECHO RETYPES ANY ENTERED LINES,
ECHO RETYPES ANY ENTERED LINES,
return
END CONSOLE TEST
R; T=0.15/0.32 10.14.40
```

b.
```
echo s
START CONSOLE TEST
incg¢Including Deletionssa, Backspaces, & special characi
Including Deletions, Backspaces, & special characters,
Return
END CONSOLE TEST
R; T=0.35/0.58 10.17.36
```

c.
```
echo x 3
START CONSOLE TEST
Except in the X mode¢
Except in the X mode¢
Except in the X mode¢
Except in the X mode¢
returra
returra
returra
returra
return
END CONSOLE TEST
R; T=0.57/1.02 10.20.30
```

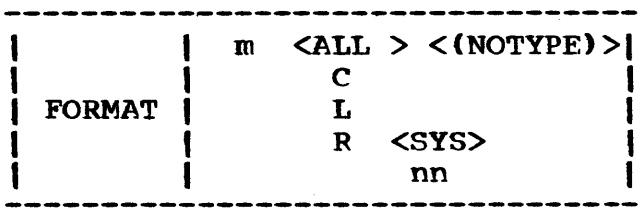
Error Messages:

None.

Purpose:

The FORMAT command is used to initialize a disk area in the CMS format, to count the number of cylinders on a disk, or to write a label in the first 6 bytes of record 3 of the disk.

Format:



m specifies the logical mode letter of the user's disk area that is to be initialized. Any files on this disk are erased.

ALL specifies that all tracks of the specified disk area are to be initialized. If omitted, the first three records of Cylinder 0, Track 0 are skipped.

C specifies that the command is a check only, and cylinders are to be counted, but not erased.

L specifies that a label only is to be written on the disk. No formatting is to take place.

nn specifies that only the first nn cylinders are to be formatted.

R is the same as C, but expands or reduces the bit-mask (PQMSK) to recompute the number of cylinders actually on the disk.

SYS is used with FORMAT P R SYS to truncate disk counts to leave room for the CMS nucleus as written by IPLDISK (on either 54 cylinder 2314 or 203 cylinder 2311).

(NOTYPE) specifies that the format is to continue without requesting any responses from the terminal.

Usage:

The disk mode must be specified. According to the option selected, the user's disk area is initialized by writing a new home address and four records on each 2311 track, or fifteen records on each two 2314 tracks. Any previous data on the disk is erased.

Unless ALL is specified, the first three records of Cylinder 0, Track 0 are skipped. The ALL operand should be specified in the FORMAT command whenever an unformatted disk area is accessed for the first time, and is not normally needed thereafter. A 6-byte label may be written in record 3 of the disk.

If C is specified, data on the disk is not erased, and files are preserved.

If L is specified, a 6-byte label may be written on cylinder 0, track 0, record 3 of the disk, preceded by CMS=. For the T-disk, the label CMS=TDISK is always used.

A read-after-write check is made as the disk is formatted.

If nn is specified, only the first nn cylinders are formatted.

If R is specified, the number of cylinders are recomputed and the PQMSK expanded or reduced accordingly. If SYS is specified with the R, there is room left for the CMS nucleus written by IPLDISK.

If (NOTYPE) is specified, there is no pause for YES or NO to continue formatting (see "Responses").

Note:

If FORMAT is the first command issued after IPL, user files are destroyed, regardless of the options specified. Enter the LOGIN UFD command, or just hit RETURN before entering FORMAT.

Responses:

m(ccu): nnn CYL
The number of cylinders formatted is typed out, indicating which disk was initialized.

** FORMAT WILL ERASE ALL YOUR M-DISK (ccu) FILES **
** DO YOU WISH TO CONTINUE? ENTER "YES" OR "NO":
This is typed out so the user can verify that the command was issued correctly. If so, YES causes the FORMAT to begin; if not, NO causes a return to CMS without altering the contents of the disk. If (NOTYPE) is specified, this response is not made.

FORMATTING M-DISK (ccu)...
This is typed after a YES is entered to inform the user that formatiing is taking place.

m (ccu): n FILES, n REC IN USE, n LEFT (of n), FULL (n CYL)
This is typed at the completion of a FORMAT M C to indicate the counts returned from the check.

Example:

```
format p
P-DISK: 010 CYL.
```

The permanent disk is initialized by writing home addresses and four blank CMS records on every track. Any files are destroyed. The response indicates that the user's permanent disk under CP consists of ten cylinders.

Error Messages:

E(00001) PLEASE SPECIFY DISK: PERMANENT (P) OR TEMPORARY (T)
The first parameter specified was not P or T.

E(00002) CONDITION-CODE 1 ON SIO IN FORMATTING DISK (BAD)
The condition code after Start I/O was issued indicates the operation was not accepted. Notify the operator.

E(00002) CONDITION-CODE 2 ON SIO IN FORMATTING DISK (BAD)
The condition code after Start I/O was issued indicates the channel was busy. This should not occur under CP. Notify the system operator.

E(00002) CONDITION-CODE 3 ON SIO IN FORMATTING DISK (BAD)
The condition code after Start I/O was issued indicates the device addressed is not operational. Notify the operator.

E(00003) UNEXPECTED UNIT-CHECK IN FORMATTING DISK
A unit check occurred. Notify the operator.

E(00004) CE and DE NOT FOUND TOGETHER (VERY STRANGE)
Channel end and device end did not occur correctly. Notify the operator.

E(00005)
Same as E(00002), except that C was specifies with the command.

E(00006) CE and DE NOT TOGETHER CHECKING NO. CYLINDERS
Same as E(00004), except that C was specified with the command.

E(00007) UNEXPECTED UNIT-CHECK NO. CYLINDERS
Same as E(00003), except that C was specified with the command.

E(00008) NO T-DISK AVAILABLE.
The T parameter was specified, but the user does not have a T-disk attached to him.

E(00009) UNRECOGNIZABLE DASD DEVICE
The device to be formatted is not either a 2311 or a 2314.

E(00010) DISK READ-ONLY
The disk is read-only, and therefore cannot be formatted.

E(00011)
YES was not typed in as the response to the *FORMAT WILL ERASE... response.

E(00012) DISK NOT ATTACHED
The disk specified is not attached to the user.

E(00013) FORMAT P R FAILURE
Data-loss would result if the command executed. The counts are unchanged.

E(00014) FORMAT P R SYS FAILURE
The disk is smaller than that required for use of the SYS option. The counts are unchanged, and the user's data is preserved.

MAPPRT

Purpose:

MAPPRT creates, and optionally prints, a file containing a map of entry points in the CMS nucleus.

Format:

```
-----------------------------------------
|              | A       ON              |
|  MAPPRT      | <N      <OFF>>          |
|              | C       NO              |
-----------------------------------------
```

A creates the file CMS-NUC ALPHABET P1, containing the nucleus entry points listed in alphabetic order.

N creates the file CMS-NUC NUMERIC P1, containing the nucleus entry points listed in numerical order.

<u>C</u> creates the file CMS-NUC ALPHANUM P1, containing both the A and N orderings.

ON types the file at the terminal.

OFF prints the file on the offline printer.

<u>NO</u> specifies that no output is requested.

Usage:

The MAPPRT command creates a file on the user's permanent disk containing a list of entry points and their addresses in core. In conjunction with a listing of the CMS routines, this information allows the system programmer to examine or temporarily modify the nucleus in storage.

The first option specifies the format of the list. The entry points are listed in alphabetic order if A is specified. If N is specified, they are listed in the order in which they appear in core. If no option, or C, is specified, both types of list are combined in a single file. The filename of the file created is always CMS-NUC. The filetype is ALPHABET, NUMERIC, or ALPHANUM, depending on the option specified. See Figure 36 for examples of the three types of format.

```
printf cms-nuc alphabet

ABEND      AT 0D5B8
ACTTAB     AT 01208
ACTTA1     AT 01210
ALTER      AT 00000
ASMADDR    AT 00B30
BATCH      AT 00000
BATSWT     AT 0B5DC
BCKSPACE   AT 0D5B8
BDEBUG     AT 0D4E4
BUFFER     AT 0B8F0
CARDPH     AT 0D808


printf cms-nuc numeric

ALTER      AT 00000
BATCH      AT 00000
ERASE      AT 00000
OFFLINE    AT 00000
SYSCTL     AT 00000
TAPEIO     AT 00000
NUCON      AT 00100
USFL       AT 00108
STADDR     AT 00110
TBLNG      AT 00118
LSTADR     AT 001C4
LOCCNT     AT 001C8


printf cms-nuc alphanum

ABEND      AT 0D5B8          ALTER      AT 00000
ACTTAB     AT 01208          BATCH      AT 00000
ACTTA1     AT 01210          ERASE      AT 00000
ALTER      AT 00000          OFFLINE    AT 00000
ASMADDR    AT 00B30          SYSCTL     AT 00000
BATCH      AT 00000          TAPEIO     AT 00000
BATSWT     AT 0B5DC          NUCON      AT 00100
BCKSPACE   AT 0D5B8          USFL       AT 00108
BDEBUG     AT 0D4E4          STADDR     AT 00110
BUFFER     AT 0B8F0          TBLNG      AT 00118
CARDPH     AT 0D808          LSTADR     AT 001C4
```

Figure 36. Samples of the three types of nucleus map files created by MAPPRT. Only the first few entries of each are shown

The second option specifies whether or not the file is to be typed or printed. ON specifies output on the terminal, and OFF specifies output on the offline printer. If no second option, or NO, is specified, no output occurs. The file is always left on the permanent disk, whether or not output is requested.

Notes:

a. If only one or two addresses are needed, the N option avoids the CPU time needed for an alphabetic sort of the entry points. Any address can be obtained by using the LOCATE request in the EDIT environment.

b. MAPPRT should only be issued after CMS has been initialized by IPL ccu, since MAPPRT uses tables set up by the nucleus loader.

Responses:

None, except under the ON option when the file is typed out.

Examples:

a. MAPPRT
With no options specified, both the default options are taken. The file CMS-NUC ALPHANUM P1 is created, containing both an alphabetic and a numeric listing of the nucleus entry points.

b. MAPPRT C OFF
The same action is taken as in the above example, and the file is printed on the offline printer.

Error Message:

None.

Purpose:

The MODMAP command types the load map associated with the specified MODULE file on the console typewriter.

Format:

```
-------------------------
| MODMAP   | filename |
-----+-------------------
```

filename is the filename of a file whose filetype is MODULE

Usage:

The loader table, which was in use at the time the GENMOD was issued to create the MODULE, is typed on the console typewriter.

Notes:

a. Transient area module load maps cannot be printed.

b. Any MODULE files created with the GENMOD (NOMAP) command do not contain a load map.

Responses:

LOAD MAP UNAVAILABLE
The MODULE file was created with the (NOMAP) option of GENMOD, or is a transient area module.

FILE NOT FOUND
The module file cannot be found.

Purpose:

OSTAPE creates CMS files from tapes produced by systems other than CMS---for example, by the OS/360 IEBPTPCH utility.

Format:

```
|OSTAPE| <filetype <filename>>  (option1...optionN  |
|      |   SYSIN     OSTAPE                        |
```

filetype Filetype of the newly created files.
SYSIN The default filetype.

filename Name of the newly created file. Has meaning only for NPDS files.
OSTAPE The default filename.

PDS Indicates that the tape contains many members of a partitioned data set, each preceded by "member name xx".
NPDS Indicates that the tape does not contain many members of a partitioned data set, each preceded by "member name xx".

COL1 Indicates that column 1 contains data. Data is taken from columns 1-80.
NCOL1 Indicates that column 1 contains carriage control information. Data is taken from columns 2-81.

TAPx The x indicates the tape unit number.
TAP2 Tape 2 is the default tape unit number.

END Indicates that the END card means end of the member.
NEND Indicates that the END card does not mean the end of the member.

MAXTEN Indicates that a maximum of ten files is to be read at one time. This is useful for long data sets.
NMAXTEN Indicates that ten is not to be the maximum number of files to be read at one time.

Usage:

OSTAPE creates CMS files from tapes produced by systems other than CMS. The tape must be in unblocked card image format. It may be unlabeled, or may have a standard OS/360 label.

A tape produced by the OS/360 IEBPTPCH utility may be used.

OSTAPE assumes the file is going to the P disk.

Responses:

FILE filename filetype COPIED

TEN FILES PROCESSED
MAXTEN has been specified and ten files have been copied.

Examples:

a. OSTAPE (PDS TAP1 MAXTEN
This reads a tape from drive 180 that was produced by IEBPTPCH of a PDS source file, ten members at a time.

b. OSTAPE DATA PAYROLL (COL1
This reads an unblocked card image tape from drive 181 and creates a single file called PAYROLL DATA.

Error Messages:

E(00002) TWO ADJACENT TAPE MARKS ENCOUNTERED

E(00003) TAPE ERROR
Warning that a parity error ha occurred; the record is processed anyway.

SORT

Purpose:

SORT arranges the records from file 1 into file 2 in ascending order according to specified sort fields.

Format:

```
-------------------------------------------------------
| SORT  |  filename1 filetype1 filename2 filetype2 |
-------------------------------------------------------
```

filename1 filetype1 is the input file to be sorted.

filename2 filetype2 is the output file that contains the sorted records.

Usage:

The records are sorted in EBCDIC order from file1 to file2. If duplicate records are found, they are written onto file2 in the order in which they are encountered in file1. All records must be of fixed length to be sorted.

File1 and file2 must have unique identifiers, as SORT cannot write the output back into the input file (file1). If the output file already exists, the old file can be erased, the sort can be terminated, the existing file can be appended with the output, or a new output file can be specified.

Once the SORT command is issued, it asks for the sort fields to be defined via the following message:

*ENTER SORT FIELD DEFINITIONS

The sort fields are defined by typing pairs of numbers, separated by blanks, where each pair is the starting and ending character position of a sort field within each input record and the leftmost pair is the major sort field. The total number of pairs of sort field definitions that can be entered is limited only by the maximum number of characters that can be typed on one line and by the maximum number of characters in the defined sort fields.

The total number of characters on which the records are to be sorted must not exceed 254 characters.

The sorting operation takes place with two passes of the input file. Pass one creates an ordered pointer table in core storage. Pass two uses the pointer table to read the input file in a random manner and write the output file. Therefore, the size of core storage and the size of the sort field is the limiting factor on the number of records that can be sorted at any one time. An estimate of the maximum number of records that can be sorted can be made by using the following formula:

$$NR = \frac{160000}{14 + NC}$$

where NR is the maximum number of input records and NC is the total number of characters in the defined sort field.

SORT uses the standard order of search to find the file. The sorted file is placed on the same disk from which the file to be sorted was obtained.

Responses:

*ENTER SORT FIELD DEFINITIONS
Define the character positions on which the records are to be sorted. Type pairs of numbers, separated by blanks, where each pair is the starting and ending character position of a sort field within each input record.

*OUTPUT FILE ALREADY EXISTS, TO APPEND IT, HIT RETURN.
*TO ERASE OLD FILE, TYPE 'ERASE'. *TO QUIT, TYPE 'QUIT', ELSE ENTER NEW FILNAM/FILTYP.
File2 already exists. Erase the old file, terminate the sort, or specify a new output file.

XXXX RECORDS READ ON PASS1.
The total number of records read from the input file is XXXX.

Example:

```
SORT  CLASS  MEMO  ALPHCLAS  MEMO
*ENTER SORT FIELD DEFINITIONS
1   10   25   28
   66 RECORDS READ ON PASS1.
R; T=0.30/1.06   15.06.47
```

The 66 records in file CLASS MEMO are sorted on positions 1-10 and 25-28. The sorted output is written into the newly created file ALPHCLAS MEMO.

Error Messages:

E(00001) *INPUT FILENAME/FILETYPE NOT DEFINED.
Both the filename and filetype were not specified for file1.

E(00002) *OUTPUT FILENAME/FILETYPE NOT DEFINED.
Both the filename and filetype were not specified for file 2.

E(00003) *INCOMPLETE SORT FIELD PAIR DEFINITION.
An ending character position was not specified for a sort field.

E(00005) *INPUT FILE DOES NOT EXIST.
The file specified as file1 does not exist.

E(00006) *INPUT FILE IS NOT FIXED LENGTH FORMAT.
The records in file1 are not fixed length, therefore, they cannot be sorted.

E(00010) CANNOT ERASE INPUT FILE
The identifiers for file1 and file2 are the same--they must be unique, as the input file cannot be erased.

Purpose:

The STAT command is used to (1) type out statistics on how many records of the user's disk areas are currently in use, (2) execute a routine to compact information in the disk management tables, (3) indicate whether the disk is a 2311 or 2314, or (4) indicate whether the disk is read-write or read-only.

Format:

STAT	<mode <?>>
	*
	?

mode specifies that only statistics for the disk with the specified mode are to be typed out.

* specifies that statistics on all disks are to be typed out (read-only and read-write).

If no operand is specified, statistics on all read-write disks are typed out.

? specifies that additional information for the requested disks is desired; that is, whether it is a 2311 or 2314 and whether it is read-write or read-only. As the first operand, specifies that a brief status of all logged-in disks is desired.

Usage:

The STAT Command gives a count of the records in use on the specified user's disks, the number of free records remaining, the percentage of assigned records in use, and the number of cylinders assigned to the user. Disk records are always 800 bytes long.

If a ? is specified, the device type (2311 or 2314) and the disk status (read-only or read-write) is typed along with the above information.

If a "?" is specified as the only parameter, a brief status of all logged in disks is typed; ie., ccu, mode, R/O or R/W.

Responses:

X-DISK (CCU): nnnn RECORDS IN USE, nnnn LEFT (of nnnn), n FULL (nnn CYL.)
This response is given for the disk specified.

ccu x status
This response is given for each logged in disk when the brief mode is requested. The x is the disk mode, and the status is either R/O, or blank for R/W.

Example:

stat
P-DISK (191): 0771 RECORDS IN USE, 0229 LEFT (of 1000), 77% FULL (025 CYL.)
T-DISK (192): 0000 RECORDS IN USE, 0040 LEFT (of 0040), 00% FULL (001 CYL.)
Since no options are specified, statistics for all read-write disks are typed out.

stat ?
191 P
193 A,P R/O
196 B,P R/O
19A S R/O
The brief statistics are typed showing the status of all currently logged in disks.

Error Messages:

E(00001) INCORRECT "STAT" PARAMETER-LIST
The command was entered in the incorrect format. Issue it again.

E(00002) ** Z-DISK (ccu) NOT CURRENTLY LOGGED IN **
The device specified is not part of your machine configuration. Issue the console function LINK or the CMS LOGIN Z ccu.

E(00003) ** NO READ-WRITE DISK(S) CURRENTLY LOGGED IN **
The STAT was isssued with no parameters, but no READ-WRITE disks are currently logged in.

Purpose:

The TAPE command copies files from disk to tape, recopies such tapes to the permanent disk, writes tape marks, positions the tape, and scans the tape for file identifiers.

Format:

```
+-------+-------------------------------------------------+
|       | DUMP   filename   filetype   <filemode>         |
|       | LOAD     < n >                                  |
| TAPE  |                                                 |
|       | REWIND                                          |
|       |                                        < TAPn > |
|       | SCAN < n >                               TAP2   |
|       | SKIP     < n >                                  |
|       | SLOAD  filename filetype  < n >                 |
|       | WRITEOF  <n>                                    |
+-------+-------------------------------------------------+
```

DUMP specifies that a disk file is to be written onto the mounted tape.

filename filetype <filemode> specify the file to be written.

* means all filename, all filetypes, or all files of the specified mode.

LOAD n specifies all files on the mounted tape before the next n tapemarks are to be copied on to the permanent disk. n defaults to 1.

REWIND positions the tape at the load point.

SCAN n types out the identifiers of the files that exist on the tape in a tape-dump format until the next n tapemarks are encountered. n defaults to 1.

SKIP n positions the tape after the next n tapemarks. n defaults to 1.

SLOAD n scans the tape between the next n tapemarks for the specified "filename filetype" and loads it onto the permanent disk. n defaults to 1.

WRITEOF n writes n tapemarks at the point where the tape is currently positioned. n defaults to 1.

TAPn specifies the symbolic tape unit of the tape to be written, loaded, or positioned. TAP2 is the default, which corresponds to tape address 181.

Usage:

Filename and type, or asterisk, must be specified for a TAPE DUMP command. Files may contain fixed or variable-length records. The specified file is first copied on to a disk utility file, then to the tape mounted at virtual device address 181. Two end-of-file marks are written after the date transfer for safety, but the tape is backspaced over them for subsequent DUMP operations. The last record of the tape file is flagged and contains directory information for TAPE LOAD.

The TAPE LOAD command copies any number of files from the tape to the permanent disk. The filenames and filetypes are obtained from the tape, and may not be specified with the command. The filemode of all files is P1. Any existing file on the permanent disk with the same designation as one of the tape files is erased and replaced. Whenever a flagged directory record is encountered on the tape, the file being written on disk is closed, and a new file is started. Reading continues to the n tapemark written by TAPE WRITEOF.

The TAPE REWIND command positions the tape reel at the load point.

The TAPE SCAN command reads through n tapemarks and types out the identifier of the files on the tape. If two adjacent tapemarks are encountered, TAPE SCAN terminates.

The TAPE SKIP command positions the tape after the next n tapemarks encountered, skipping the file at the current position. Use of the TAPE WRITEOF and TAPE SKIP allows access to individual files or groups of files.

The TAPE SLOAD command scans the tape between the next n tapemarks for the specified file and loads it onto the permanent disk with mode P1.

The TAPE WRITEOF command writes n tapemarks wherever the tape is currently positioned. The tapemark is recognized by TAPE LOAD as end of file.

Responses:

filename filetype P1 LOADED.
TAPE LOAD gives this response as each file copied from tape is completed. This response is followed by the Ready message after the last file.

DUMPING...
Disk files are being dumped to tape and their identifiers are typed out as they are dumped.

TAPn NOT READY YET
The tape is attached, but not physically, in a ready status.

(OK - READY NOW)
The attached tape has been readied and can now be used.

Notes:

a. The TAPE command can be used with a 7-track tape, density 800 BPI, converter on, and translator off.

b. The (EXEC) form of the LISTF command is useful for copying all files, or groups of related files.

c. The TAPE command does not handle multivolume files.

d. TAPE DUMP and TAPE LOAD are valid even if the size or location of the virtual disk is redefined between the commands.

e. Tape records written by TAPE DUMP are 805 bytes long. The first character is a binary 2 (hex '02'), followed by the characters CMS and a blank (hex '40'), followed by 800 bytes of file data packed without regard for logical record length. In the final record, the character N replaces the blank after CMS, and the data area contains directory information (see program logic manual).

f. Under CP-67, all tape units must be attached to the CMS user before any tape I/O can occur. Refer to "Operating Considerations--Tape Procedures" for information on the attachment of tapes.

Examples:

a.
```
TAPE REWIND
TAPE DUMP FILEA TEXT P5
TAPE DUMP FILEB TEXT P5
TAPE WRITEOF
TAPE REWIND
```
This series of commands copies two user files to tape. (In this and the following examples, the Ready message following each command is omitted.) The initial TAPE REWIND positions the tape at the load point. FILEA is then copied, followed by the flagged directory record. The next record is the first of FILEB. TAPE WRITEOF places a tapemark after the directory record of FILEB, and the final command repositions the tape to the load point again.

b.
```
TAPE SKIP
TAPE DUMP SYSLIB MACLIB SY
TAPE WRITEOF
TAPE REWIND
```
Assuming the same tape is mounted as in the previous example, this series of commands copies SYSLIB MACLIB onto the tape after the tapemark following FILEB. A tapemark is written following the directory record of SYSLIB, and the tape is rewound to the load point.

c. TAPE LOAD

Again using the same tape, this command erases FILEA from the disk, and replace it with the tape copy. When the directory record is processed, the following response is typed:

```
FILEA   TEXT   P1 LOADED.
```

Note that the mode is now P1. Because there is no tapemark following FILEA, FILEB is read and replaces the disk copy of FILEB. The response

```
FILEB   TEXT   P1 LOADED.
```

is followed by the Ready message when the tapemark is encountered.

Another TAPE LOAD command would now copy SYSLIB MACLIB onto the permanent disk as SYSLIB MACLIB P1. SYSLIB MACLIB SY on the system disk would not be erased.

Error Messages:

E(00001) INVALID TAPE COMMAND FUNCTION OR PARAMETER LIST.
The format of the command was incorrect, or the operation specified was unknown. Retry the command. No operation was performed.

E(00007) FILE NOT FOUND.
The file specified with TAPE DUMP was not found. No operation was performed. Check the file designation and retry the command.

E(00002) FATAL TAPE ERROR WHILE WRITING.
An I/O error occurred which could not be corrected by ten retries. Notify the operator to replace the tape. This message is also issued for errors during tape control operations (REWIND, WRITEOF, SKIP). If TAPE SKIP was issued at a point beyond which no tapemarks existed, the tape went to end of reel. A subsequent DUMP, SKIP, or WRITEOF results in this message.

E(00002) TAPn IS FILE PROTECTED
FATAL TAPE ERROR WHILE WRITING.
The ring is not in the tape, therefore writing cannot occur.

E(00002) TAPn NOT ATTACHED
FATAL TAPE ERROR WHILE WRITING.
The tape is not attached, therefore it cannot be used. Refer to "Operating Considerations--Tape Procedures" for information on tapes.

E(00002) DISK ERROR WHILE READING.
An I/O error occurred while the file was being transferred to or from the disk utility file. Retry the command. If the error recurs, notify the operator. If the file was delete-after-reading, see note 2 under COMBINE.

E(00003) FATAL DISK ERROR.
An I/O error occurred, or, for a DISK LOAD, the disk may be

filled. To retry the command, reposition the tape with a TAPE REWIND and the necessary number of TAPE SKIP commands. If the error recurs, notify the operator.

E(00003) DISK ERROR WHILE WRITING.
An I/O error occurred during transfer of the file to or from the disk utility file, or the user's disk is full. Retry the command. If the error recurs, notify the operator. If the filemode was delete-while-reading, see note 2 under COMBINE. Note b.

E(00004) FATAL TAPE ERROR WHILE READING.
An I/O error occurred. A tapemark may not be at the end of the tape. Reposition the tape and retry the command. If the error recurs, and the file has not been loaded, part of the file may be recoverable under the designation (DISK) (TEMP). See E(00006).

E(00005) TAPE IS NOT IN "TAPE LOAD" FORMAT.
A tape record was read which was not written by TAPE DUMP. The wrong tape is mounted, or reading has continued past the end of TAPE DUMP output because no tapemark was read.

E(00006) ENDING RECORD OF FILE MISSING.
A tapemark or end of reel was encountered before the flagged directory record of the current file. Part of the file may be recoverable under the designation (DISK) (TFILE) P3. Note that the mode is delete-after-reading, so an ALTER or COMBINE command must be issued before inspecting this file.

TAPEIO

Purpose:

The TAPEIO command executes appropriate tape I/O commands.

Format:

```
-------------------------------------------------
|           |                        <TAP1>     |
|  TAPEIO   |  function                         |
|           |                         TAP2      |
-------------------------------------------------
```

TAP1 corresponds to device 180.

TAP2 corresponds to device 181. TAP2 is assumed if no tape is specified.

Functions are as follows:

BSF	to backspace one file
BSR	to backspace one record
ERG	to erase a gap
FSF	to forwardspace one file
FSR	to forwardspace one record
REWIND	to rewind the tape to load point
RUN	to rewind and unload the tape
WRITEOF	to write a tape mark
WTM	to write a tape mark

Usage:

The tape on the specified device is moved a single record or file, is rewound and/or unloaded, a tape mark (end of file) is written, or a gap is erased.

Notes:

a. The modeset is set to X'FF', therefore, 9-track tapes with density 800, odd parity, and converter off can be manipulated with the TAPEIO command. If other modes of tapes are to be used, TAPEIO should be called as a function from an Assembly Language program (see "CMS Nucleus Routines--TAPEIO").

b. The REWIND and RUN functions indicate completion before the physical operation is completed. Thus a subsequent operation to the same physical device may encouter a "device busy" situation.

c. The TAPEIO command is identical to the TAPEIO function except that TAPEIO READ and TAPEIO WRITE cannot be executed from the command level at a terminal. If either of these is attempted, an appropriate error and error code are given (see below).

Responses:

TAPn NOT READY YET.
The specified tape has been attached but it is not ready yet. The following message is received when the tape drive is in a ready status.

(OK - READY NOW)
The attached tape is now ready for use.

Error Messages:

E(00001)
An invalid function was specified.

E(00002)
An end of file or end of tape has been reached.

E(00003)
A permanent I/O error has occurred while reading or writing the tape.

E(00004)
An invalid symbolic tape unit was specified.

E(00005) TAPn NOT ATTACHED.
The specified tape is not attached, therefore the function cannot occur.

E(00005)
Same as previous message, but the message only prints the first time this error occurs.

E(00006) TAPn IS FILE PROTECTED.
The specified tape contains no file-protect ring, therefore the tape cannot be erased or written on.

E(00007) TAPn - SERIOUS TAPE ERROR ATTEMPTING function
An unrecoverable tape error has occurred on the tape while attempting the specified function.

TAPRINT

Purpose:

TAPRINT copies files from a specified tape to the offline printer. The files must be LISTING files created by the WRTAPE or ASSEMBLE commands.

Format:

```
------------------
|TAPRINT|  <TAPn> |
------------------
```

TAPn is either TAP1 or TAP2, specifying which tape is to be copied. If omitted, TAP2 is assumed.

Usage:

TAPRINT prints tape files in the special format written by the LTAPn option of the ASSEMBLE command and by the WRTAPE command. Printing starts wherever the tape is positioned when the command is issued, and continues until two consecutive end of file marks are encountered. Single end of file marks are ignored, except for a message. On completion, the tape is rewound.

Responses:

EOF READ ON TAPE
A single end of file was read on the tape. Printing will continue.

PLEASE READY THE PRINTER
This message should not occur under CP.

END OF TAPE
Two consecutive end-of-file marks were encountered. The tape is being rewound, and control is passed to the CMS Command environment.

PERMANENT I/O ERROR ON TAPE.
An I/O error occurred. The command is terminated.

Example:

TAPRINT TAP1
All the files up to the first two consecutive end-of-file marks on the tape mounted on TAP1 (at 180) are assumed to be LISTING files and are printed on the offline printer.

Error Messages:

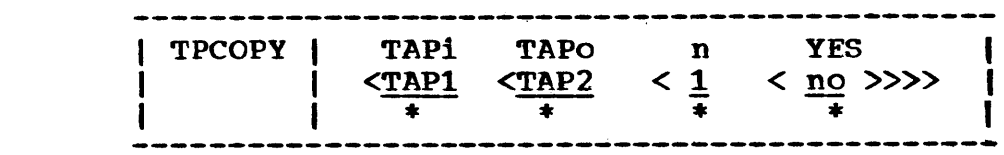
E(00001) SYMBOLIC TAPE ADDRESS INCORRECT.
The symbolic tape address specified was not TAP1 or TAP2. No action was performed.

TPCOPY

Purpose:

The command TPCOPY is used to copy tape files.

Format:



TAPi is the input symbolic tape unit

TAPo is the output symbolic tape unit

n is the number of files to be copied and ranges from 1-9.

YES types out a summary of each file copied. If YES is not specified, no summary is assumed.

* specifies the default is to be taken for each parameter.

Usage:

7-track tapes are read and written at 800 bpi, odd parity, converter on, translator off.

If TAPi and TAPo are not specified, TAP1 and TAP2 are assumed, which correspond to 180 and 181 respectively.

If n is not specified, the default is one file to be copied.

The maximum record size is 4096 characters.

Note:

If TAPE DUMP was issued to write files from disk to tape and that tape is to be copied, the input tape unit is tape 181 or TAP2. Therefore TAP2 should be specified as the first operand with TPCOPY.

Responses:

SUMMARY OF COPY
The YES parameter was specified with TPCOPY, therefore a summary of the copy of each file is typed out.

FILE xxx
The xxx is the sequence number of the file being copied on the tape.

xxxxxxxx RECORDS, LENGTH = xxxxx BYTES
The number of records on TAPi and the maximum length of each

are specified.

xxxxxxxx RECORDS COPIED
The xxxxxxxx is the number of records copied from TAPi to TAPo.

SUMMARY COMPLETE
The summary typeout has terminated.

TAPE READ ERROR; BYPASS? (YES/NO)
An error has been encountered while reading the tape. The keyboard is unlocked for either a YES or NO response. If YES is entered, TPCOPY skips the error record, and continues reading. If NO is entered, error code 1 is returned and TPCOPY terminates.

SUMMARY TABLE OVERFLOW; SUMMARY CANCELLED
The summary has been canceled.

Error Messages:

E(00001) TAPE WRITE ERROR; TERMINATING.

E(00001) ILLEGAL FILE COUNT FIELD:
The number of files specified to be copied was less than one or more than nine.

E(00003) SAME UNIT REQUESTED FOR INPUT AND OUTPUT:

WRTAPE

Purpose:

WRTAPE copies fixed-length format files from disk to tape. If the filetype is LISTING, assembler and compiler carriage-control codes are translated to machine codes.

Format:

```
-------------------------
|WRTAPE| fname ftype     |
-------------------------
```

fname ftype specifies the file to be copied.

Usage:

WRTAPE copies files from any disk to TAP1. Records are blocked in groups of ten, and the record format must be of fixed length less than 256 bytes. No end of file is written on the tape, and the tape is not rewound when the command completes.

Although WRTAPE handles any file of the format described above, it is specially designed for LISTING files created by the ASSEMBLE and FORTRAN commands. These files contain a carriage control code for the offline printer as the first byte of each record. WRTAPE translates the code into a machine code for the printer. Files are written on the tape in the same format as if the ASSEMBLE command had been specified with the (LTAPn) option. This format is acceptable to the TAPRINT command.

Notes:

a. WRTAPE does not write an end of file on the tape on completion, nor does it position the tape past any existing files. Use the facilities of the TAPE command for positioning. To mark an end of file for TAPRINT, write two end-of-file marks.

b. Under CP, the tape must be attached by the operator. WRTAPE expects TAP1, addressed at 180, which should be specified when you request the operator to attach the tape.

c. Tape files written with WRTAPE are not suitable for rereading with TAPE LOAD.

Responses:

None.

Example:

WRTAPE PROG LISTING
The file PROG LISTING P5 is copied to tape, with carriage-control codes translated to machine codes, and records blocked in groups of ten. If this is to serve as input to the TAPRINT command, the following commands should be issued:

```
TAPE WRITEOF
TAPE WRITEOF
TAPE REWIND
```

Error Messages:

E(00002) PARAMETER ERROR
Two parameters (filename and filetype) were not specified.

E(00003) VARIABLE LENGTH FILE
WRTAPE does not handle files with variable record length.

E(00004) FILE NOT FOUND
No file with the specified filename and filetype was found.

E(00005) FATAL ERROR
An error occurred while reading the file from disk.

E(xxxxx) TAPE ERROR
A tape error occurred which could not be recovered. The error code is unpredictable.

CONTROL COMMANDS

This section contains CMS commands that are used to control the user's environment at the console.

The BLIP command changes the character used to notify the user of every two CPU seconds of execution time. The CHARDEF command allows the user to redefine the logical characters, such as the character delete, line delete, logical backspace, and logical tab characters. LINEND redefines the logical carriage return or line-end character. SYN allows a user to define his own synonyms for commands. CPFUNCTN allows CP console functions to be issued from CMS.

IPL loads a new copy of CMS into memory. KO, KT, and KX are used to kill overrides, typing, and execution respectively. RT restores typing once it has been terminated by KT. LOGIN sets up the user's permanent files for this terminal session; LOGOUT updates the files on the P-disk and removes the user from CMS; and RELEASE frees a disk when the user has finished with it.

BLIP

Purpose:

The BLIP command causes a string of from one to eight characters to be typed periodically during CMS operation. The BLIP characters are typed out after every two seconds of CPU execution and give the user an indication of the execution time of his program.

Format:

BLIP	chars	<count>
	(OFF)	1

chars is the character to be typed out, and

count is a digit from one to eight, indicating the number of BLIP characters.

(OFF) sets BLIP to nothing so that there is no indication after each two seconds of CPU execution.

Usage:

The default setting of the BLIP characters is a sequence of nonprinting characters--uppercase shift, lowercase. If it is is desired to have a printed recording of the execution of a program, the BLIP characters should be changed to printing characters (for example, a single dot).

Notes:

a. If the count parameter is defaulted, a count of 1 is assumed.

b. If the first parameter is zero or if both parameters are defaulted, the BLIP characters are reset to their nonprinting default setting.

c. If noncharacter codes are desired, the eight bytes for the BLIP characters can be specified in hexadecimal in an Assembly Language Program.

Error Returns:

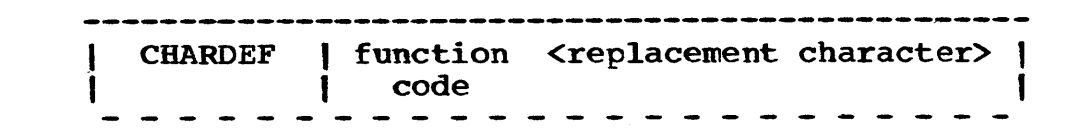
None.

CHARDEF

Purpose:

CHARDEF redefines the logical characters in CMS that correspond to hardware functions.

Format:



function code:

B signifies the EDIT logical backspace character.

C signifies the character-delete symbol.

L denotes the line-delete symbol.

T signifies the EDIT logical tab character.

replacement character is the character to be used for the specified function. If a replacement character is not specified, there is no symbol for that function.

Usage:

The predefined characters in CMS for the character-delete symbol and the line-delete symbol are the @ and ¢ respectively. If a character is not specified with either CHARDEF C or CHARDEF L, there is no delete symbol for a character or a line.

The predefined logical tab character and backspace character in EDIT are the # and the % respectively. If B or T is specified, EDIT and CEDIT commands cause reference to be made to the redefined symbols each time either editor is invoked.

The function symbols that are redefined remain in effect until (1) CMS is reIPL'ed, or (2) another CHARDEF command is issued, or (3) the user logs out of CP.

Note:

CHARDEF does not redefine the logical characters for CP-67.

Examples:

a. CHARDEF C ?
The character-delete symbol is now defined as ?. The @ can be used as a normal character.

b. CHARDEF L
The line-delete symbol no longer exists. The ¢ can be used as a normal character.

Error Message:

E(00001)
Illegal function code was specified.

CPFUNCTN

Purpose:

The CPFUNCTN command transmits console function commands to CP-67 without leaving the virtual mode of operation.

Format:

```
-----------------------------------------
| CPFUNCTN  |  <NOMSG>     string       |
|    CP     |                           |
-----------------------------------------
```

NOMSG is an optional argument to turn off typing of BAD ARGUMENT and INVALID CP REQUEST messages.

string is a CP-67 console function.

Usage:

CPFUNCTN is provided to allow CP-67 console functions to be issued from CMS and to be incorporated in EXEC files.

Examples:

a. CPFUNCTN XFER 00D to CSC1
Executes an XFER console function.

b. CP NOMSG DETACH 00E
Executes a DETACH command. If 00E is not a valid address, the BAD ARGUMENT message is not typed.

Error Messages:

E(00001) FUNCTION MISSING
No string was specified with the command.

E(00004) INVALID CP REQUEST
string is not a proper CP-67 console function.

E(00008) BAD ARGUMENT
The arguments supplied are not proper for the specified console function.

E(xxxxx)
Any other error codes are dependent on the particular console function designated.

IPL

Purpose:

IPL causes a new copy of the CMS nucleus to be initial-program loaded.

Format:

```
------------------
| IPL | <devadd> |
------------------
```

devadd is the address of the disk device that is to be IPL'ed.

Usage:

This command performs the same action as the IPL console function. If a devadd is specified, CMS gives the IPL to CP, which IPL's the specified device.

If no operand is included in the IPL command, IPLDEV in the nucleus is checked to see what copy of CMS the user had been using (that is, if CMS had been IPL'ed by name or device). This same copy of CMS is then brought back into core, and the user receives a new copy of the version of CMS he had been using.

Response:

CMS...VERSION nn LEVEL mm
This response indicates that a new copy of the nucleus has been loaded and that control has transferred to the CMS Command environment.

Example:

ipl
CMS...VERSION nn LEVEL mm
A new copy of the CMS nucleus has been initial-program loaded and the keyboard is unlocked to accept any CMS command. If the user IPL'ed 190 originally, that copy of CMS has been brought back in; if he IPL'ed CMS by name, the "saved" system has been reloaded.

Error Messages:

None.

Purpose:

The KO command may be issued during the execution of a command or user program to stop the recording of trace information.

Format:

```
--------
|  KO  |
--------
```

Usage:

The KO command causes recording of trace information, initiated by the SETOVER or SETERR commands, to be halted. KO differs from the CLROVER command in that it may be used to clear overrides during the execution of a program. In order for the KO command to be recognized, it must be entered after interrupting program execution by hitting ATTN once to transfer to the CP environment, and a second time to transfer control to CMS. Typing KO then causes all overrides to be cleared, and no further trace information is recorded. Program execution continues to its normal completion, and all recorded trace information is printed on the offline printer.

Notes:

a. Entering KO as a normal CMS command has no effect. KO has meaning only after two attention interrupts have been generated during command or program execution.

b. If the keyboard does not unlock following the second attention interrupt, internal processing is probably taking place which does not allow interrupts from the multiplexer channel. As soon as this processing is conpleted, the keyboard is unlocked and KO may be entered.

c. The character-delete symbol and line-delete symbol are not valid when issuing the KO command.

d. Issuing the KO command has no effect unless a SETERR or SETOVER command has been issued previously.

Example:

KO
Assuming that ATTN had been hit twice before typing KO, CMS stops recording trace information, and, at the completion of the currently executing program, all recorded trace information would be recorded on the offline printer.

Error Messages:

None.

Purpose:

The KT command causes all terminal output generated by the CMS command or user program in progress to be suppressed.

Format:

```
---------
|   KT  |
---------
```

Usage:

The KT command is used to stop terminal typeout from an executing CMS command or user program. In order to enter the KT command meaningfully, ATTN must be hit once to interrupt execution and transfer control to the CP environment, and then a second time to transfer control to CMS. Program execution continues but the keyboard is unlocked to accept user input. Typing KT at this point causes all further terminal output from the executing command or user program to be intercepted and suppressed. Execution continues to normal program completion, when the Ready message is typed out and normal terminal output resumes.

Notes:

a. Entering KT as a normal CMS command has no effect. KT has meaning only after two attention interrupts have been generated during program execution.

b. If the keyboard does not unlock following the second attention interrupt, internal processing is probably taking place which does not allow interrupts from the multiplexer channel. As soon as this processing is completed, the keyboard is unlocked and KT may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing a KT command.

Example:

```
KT
```
Assuming that ATTN had been hit twice before typing KT, CMS would intercept and suppress any terminal output generated by the program in progress.

Error Messages:

None.

KX

Purpose:

The KX command causes the execution of any CMS command or user program to stop, closes any open files and I/O devices, reIPL's CMS, and returns the user to the CMS command environment.

Format:

```
----------
|   KX   |
----------
```

Usage:

In order for the KX command to be recognized, it must be issued after interrupting program execution by hitting ATTN once to transfer to the CP environment, and a second time to transfer control to CMS. Issuing the KX command then closes any open user files and signals CP that the user has no more I/O for offline devices. After updating the user's file directory, KX reIPL's CMS from the same device as it was initially IPL'ed, and the user is returned to the CMS command environment.

Notes:

a. Entering KX as a normal CMS command has no effect. KX has meaning only after two attention interrupts have been generated during program execution.

b. If the keyboard does not unlock following the second attention interrupt, internal processing is probably taking place which does not allow interrupts from the multiplexer channel. As soon as this processing is completed the keyboard is unlocked and KX may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing a KX command.

Response:

CMS...VERSION nn LEVEL mm
This response is typed whenever KX is interpreted, as CMS is reIPL'ed. The CMS command environment is then entered.

Error Messages:

None.

Purpose:

The LINEND command defines the logical line-end character to be used, in addition to the carriage return (new line).

Format:

```
----------------
| LINEND | <c>  |
----------------
```

c is the redefined logical line-end character. If c is not specified, there is no line-end character defined, and the carriage return is the only line delimiter.

Usage:

The logical line-end character permits a number of logical input lines (each separated by the line-end character) to be typed on a single physical input line. The physical input line is terminated by a carriage return. Logical input lines are terminated by the line-end character or by the carriage return. Each call to read a line from the terminal returns the logical input line. Subsequent calls to read a line from the terminal returns the logical input line which was given following the previous logical input line. The line-end character can be used to input logical lines whenever a physical line is input from the typewriter, whether to CMS or to a program. In addition, logical lines can be input and stacked by use of attention to CMS (double attention if from CMS and running under CP). See Terminal Usage-"Attention Interrupt".

Notes:

a. The defined line-end character is the #, unless the LINEND command has been issued to redefine the character.

b. If the command LINEND is issued without any parameters, only the carriage return is used as the line delimiter.

c. When a physical input line is read, it is scanned and processed according to the specifications in WAITRD. If lowercase to uppercase conversion is specified, the complete physical input line is translated. Thus, all logical input lines are translated according to the initial specification.

d. The redefined line-end character stays in effect until either CMS is IPL'ed again or LINEND is reissued.

e. LINEND does not redefine the logical line-end character for CP-67.

f. The pound sign (#) is also the logical tab character.

LINEND takes precedence over the logical tab character.

Examples:

a. LINEND !
The line-end character is set to the exclamation mark (!).

b. LINEND
There is no longer a defined line-end character, therefore, only the carriage return is used as a line delimiter in CMS.

Error Messages:

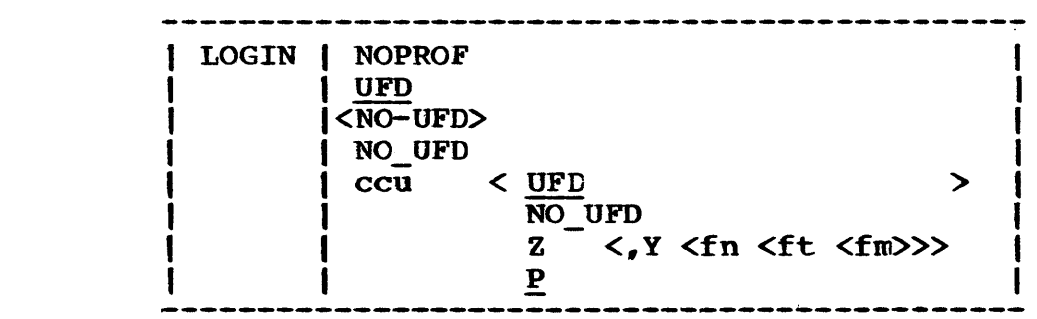
None.

LOGIN

Purpose:

LOGIN causes the user's files on a specified disk to be either saved or deleted, as specified, or bypasses the execution of the PROFILE EXEC file.

Format:



UFD indicates that the user's file directory (and hence all existing disk files) are to be saved. (This is the default parameter.)

NO_UFD indicates that the user's file directory is not to be saved, effectively clearing his disk area and all files stored on it.

NO-UFD has the same effect as NO_UFD.

NOPROF indicates that the user's PROFILE EXEC file is not to be executed.

ccu indicates the address of the disk that is to be used. ccu must be a nonzero, hexadecimal number not to exceed 6FF.

Z indicates the mode of the disk (that is, P,T,A,etc.).

,Y indicates the disk for which this disk is to be a read-only extension.

fn <ft <fm>> indicates on a read-only disk the files whose directories are to be brought into core.

Usage:

The LOGIN command may be issued at any time from the CMS Command environment. If an operand is specified with LOGIN, it must be either UFD, NO_UFD, NO-UFD, NOPROF, or ccu. If UFD is specified, the user's file directory, and hence all of his P-disk files, are maintained as they presently exist. If NO_UFD or NO-UFD is specified, the user file directory is cleared and all P-disk files are, in effect, erased. If NOPROF is specified, the execution of the PROFILE EXEC

feature is inhibited. In particular, NOPROF is used if the PROFILE EXEC is to be bypassed or contains errors. If no operand is specified, either 191 or the last disk that was used as the P-disk is logged in as if LOGIN ccu were specified. If the LOGIN command is not issued as the first CMS command after IPL'ing, LOGIN UFD is assumed and all existing P disk files saved.

LOGIN ccu allows the user to specify the device that he wishes to use. Z, if specified, is the mode of the disk--the default mode is P; if Z is specified, Y may be specified to describe this disk as a read-only extension of the Z-disk. The user may also specify fn and, optionally, ft and fm to allow his use of only that specific file.

Notes:

a. If a user links to another Z-Disk and he already has a Z-Disk, he must either reIPL CMS to read in the new file directory, or issue LOGIN ccu. Otherwise, the old file directory exists in core and the file directory of the new disk is overwritten.

b. LOGIN NO_UFD or NO-UFD performs the same functions as FORMAT P, that is, it obtains a "clean" file directory; however, it does not physically format the disk. Therefore, if the disk is already formatted correctly, this is a faster method of clearing the file directory.

c. If LOGIN ccu is issued as the first command and ccu is not 191, the normal 191 P-disk is not logged in. Therefore, if 191 and ccu are both needed, LOGIN 191 first, then issue LOGIN ccu.

d. If a user wishes to login a disk which is normally READ-WRITE in a READ-ONLY status, he can do so by making the disk a READ-ONLY extension of itself (for example, LOGIN 191 P,P).

Responses:

** Z (CCU) IS READ-ONLY **
The device specified as CCU is a read-only disk, therefore writing is not allowed on the device. This response is only a warning.

** CCU REPLACES Z (CCU) **
This response is typed following a LOGIN ccu command. It indicates that the device specified by the first ccu has replaced that specified by the second ccu.

** CCU ALSO = Z-DISK **
This response informs the user that the device ccu for which he has just issued a LOGIN, is also defined as his Z-Disk. This response is only a warning.

Examples:

a. LOGIN
The user's file directory for 191 is read into core.

b. LOGIN NOPROF
The user's 191 file directory is read into core; however, the file PROFILE EXEC is not automatically executed.

c. LOGIN 195 A,P
Disk 195 is logged in as A-disk, and this is defined as a READ-ONLY extension of the user's P-disk.

d. LOGIN 197
Disk 197 is logged in as the user's P-disk--it replaces the disk previously defined as the P-disk.

Error Messages:

E(00001) **Z (CCU) FILE DIRECTORY UNREADABLE **
Check that the correct disk is mounted. If so, issue the FORMAT P command.

E(00002) **Z (CCU) NOT ATTACHED **
No Z-disk is attached to the user's virtual machine. Notify system operator.

E(00003) ** Z (CCU) NOT RECOGNIZABLE DEVICE-TYPE**
The attached ccu device is not a 2311, 2314, or 2303. Contact system operator.

E(00004) ** LOGIN NO_UFD FAILED BECAUSE Z (CCU) IS READ-ONLY **
A LOGIN NO_UFD erases files, therefore it may not be issued for a READ-ONLY disk.

E(00005) **Z(CCU) OLD (1967 ERA) 2311 DEVICE ADDRESSES**
The disk specified is of the pre-1967 type that is no longer supported by CMS.

E(00007) **CCU ALREADY ACCESSED AS Z-DISK (READ-WRITE)**
"RELEASE CCU Z" IF DESIRED & RETRY
The disk that you have just logged in as ccu is already attached to you as a Z-disk. These two descriptions are incompatible.

** LOGIN NO_UFD FAILED; TRY "FORMAT Z ALL" **
This message may appear as a second line to errors 1,2,3, or 5. The disk needs to be formatted.

LOGOUT

Purpose:

The LOGOUT command compacts the user's file directory, executes any CMS command specified as an operand, and logs out of CMS, transferring control to the CP environment.

Format:

```
-------------------------------
|   LOGOUT   |      <anycom>    |
-------------------------------
```

anycom is any CMS command

Usage:

The LOGOUT command is not required when use of CMS has been completed. If issued, however, it causes a compacting routine (identical to that which is called when the STAT C command is issued) to be called. This routine reorganizes the user's file directory, eliminating blank entries resulting from files which have been erased during the current terminal session. In addition, LOGOUT executes any CMS command specified as its operand (LISTF, for example, to obtain a list of the newly organized file directory).

Finally, the Ready message is typed, indicating in seconds the CPU time used during this terminal session for CMS execution as well as CMS and CP execution. The revised user-file directory is written out to disk, and control is transferred to the CP environment.

Notes:

a. Given the fact that the user-file directory is updated on disk after the completion of each CMS command, it is not necessary to LOGOUT from CMS to ensure that the user's files be saved.

b. In order to log out from the CP-67/CMS system without using the LOGOUT command, hit ATTN once to transfer control to CP and type LOGOUT to log out from the Control Program.

c. Even if the LOGOUT command is issued in CMS, it is also necessary to log out from the Control Program. The CMS LOGOUT command can be bypassed, but the CP logout command must be issued to log off the system and disconnect the telephone line.

Responses:

R; T=xx.xx/xx.xx hh.mm.ss
CP ENTERED, REQUEST, PLEASE.
This message is typed whenever the LOGOUT command is issued

in the CMS Command environment. The first xx.xx is the total CPU time in seconds for CMS execution. The second xx.xx represents the total CPU time in seconds for CMS and CP. These times are total times for the terminal session. The keyboard is then unlocked to accept any CP console function.

Examples:

a. logout

```
R; T=18.32/43.21  12.15.28
CP ENTERED, REQUEST, PLEASE.
```

The compacting routine is called, the indicated message is typed at the terminal, and control transfers to the CP environment, where the keyboard is unlocked to accept any CP console function.

b. logout listf

The compacting routine is called, the LISTF command types out the contents of the reorganized user-file directory, the logout message is typed, and control transfers to the CP environment, where the keyboard is unlocked to accept any console function.

Error Messages:

None.

RELEASE

Purpose:

The RELEASE command frees an active disk when the user no longer needs it.

Format:

```
-------------------------------------
| RELEASE | ccu  mode <(DETACH)>  |
-------------------------------------
```

ccu is the device address of the disk that is to be released.

mode is the mode of the disk to be released.

(DETACH) means that the released disk is also to be DETACHED from user's virtual machine (the equivalent of the console function DETACH).

Usage:

The RELEASE command allows a user to release an active disk when he no longer needs it. By specifying the (DETACH) option, the disk may also be removed from the user's virtual machine configuration in the same way as by the console function DETACH.

RELEASE is useful in conjunction with the console function LINK and the CMS command LOGIN, when multiple disks are used and/or users share disks on a READ-ONLY basis.

Responses:

None.

Examples:

a. RELEASE 195 T
This command causes T-Disk 195 to be released by the user.

b. RELEASE 197 P (DETACH)
This command causes the P-Disk 197 to be released by the user. It also detaches it from the user's virtual machine configuration so that it is no longer available to him.

Error Messages:

E(00001) INVALID "RELEASE" PARAMETER-LIST
The command was entered in incorrect form. Check the command format and reenter it.

E(00002) NO ACTIVE-DISK-TABLE FOUND FOR GIVEN MODE
The disk mode specified is incorrect. Reissue the command

with the correct mode.

E(00003) GIVEN DISK-NUMBER DOESN'T MATCH DEVICE-TABLE
The disk specified is incorrect. Reissue the command.

Purpose:

The RT command restores the typeout at the console previously suppressed by the KT command.

Format:

```
----------
|   RT   |
----------
```

Usage:

The RT command restores console typeout from an executing CMS command or user program that was previously suppressed by the KT command. In order to enter the RT command meaningfully, ATTN must be hit once to interrupt execution and transfer control to the CP environment, and then hit a second time to transfer control to CMS. Program execution continues but the keyboard is unlocked to accept user input. Typing RT at this point causes all further console output from the executing command or user program to be typed out. Execution continues to normal program completion.

Notes:

a. Entering RT as a normal CMS command has no effect. RT has meaning only after two attention interrupts have been generated during program execution.

b. If the keyboard does not unlock following the second attention interrupt, internal processing is probably taking place which does not allow interrupts from the multiplexer channel. As soon as this processing is completed, the keyboard is unlocked and RT may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing an RT command.

Example:

```
RT
```
Assuming that ATTN had been hit twice before typing RT, CMS restores any console output generated by the program in progress.

Error Messages:

None.

Purpose:

The SYN command allows the user to specify his own command names to be used with or in place of the standard system command names.

Format:

SYN	<filename <filetype <filemode>><(option1...optionN)>>
	SYN *

filename, filetype, and filemode are the identification of the file containing the user-defined synonyms to be used by CMS.

Options:

P	Prints the standard system abbreviations and user synonyms currently defined.
PUSER	Prints only the user synonyms currently defined.
STD	Standard system abbreviations are to be used. This is the default value.
NOSTD	Standard system abbreviations are not to be used.
MIN	Use minimum number of characters specified to identify commands. The default value.
EXACT	Use exact number of characters specified to identify commands.
CLEAR	Clears any previously defined synonym table set up by SYN.

Usage:

The SYN command permits user-defined names to be used either alone, or in conjunction with the standard CMS system abbreviations--that is, it permits the user to modify the command names acceptable to his own environment.

User-defined synonyms are located in a file identified as "filename filetype filemode" in the format shown in note 2 and the first example below. If filetype is omitted, a filetype of SYN is assumed; if filemode is omitted, a mode of * is assumed, meaning any disk. If no file is specified, no user-defined synonyms are set up, and the system abbreviations are used in the manner defined by the specified options.

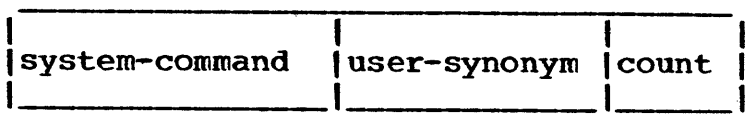
All options are specified between a pair of parentheses. The default options are STD (use standard abbreviations) and MIN (allow a minimum number of characters to represent a command). NOSTD flags the standard system abbreviations as unusable; MIN accepts abbreviations as long as the minimum number of characters specified in the abbreviation table are present; EXACT accepts only the entry as specified.

SYN can also be used to print out the list of synonyms and abbreviations currently acceptable.

Notes:

a. SYN with no additional parameter is the same as SYN (P), that is, it types a listing of system and user abbreviations currently in effect.

b. The user synonym file "filename filetype filemode" consists of 80-byte fixed-length records in freeform format with columns 73-80 ignored. The format for each record is



where count is the number of characters necessary for the synonym to be accepted. If omitted, the entire synonym must be entered (see the first example below). SYN builds a table from the contents of this file to use for command synonyms.

Responses:

SYSTEM ABBREVIATIONS FLAGGED "NOT IN USE"
A request has been made to print the system abbreviations while a previous NOSTD is in effect.

NO USER SYNONYM TABLE CURRENTLY IN USE
A request has been made to print the user-defined synonym table while no such table has been defined by a SYN command.

Examples:

a.
```
edit abb syn
NEW FILE
INPUT:
erase delete 3
combine copy

EDIT:
file
```

This creates a file ABB SYN which is a synonym table. In the first record of the file the count field is 3. This means that DEL, DELE, DELET, or DELETE are acceptable as the ERASE command. However, with no count specified in the next line, only COPY is acceptable as the COMBINE command.

b. SYN
Types a list of the system abbreviations and user synonyms currently in effect.

c. SYN ABB (P)
Sets the synonym list found in file ABB SYN as well as the standard system abbreviations as acceptable -- and prints these at the console.

d. SYN (PUSER)
Prints the contents of the user synonym table currently acceptable to CMS.

e. SYN ABB (NOSTD)
Allows the synonyms in file ABB SYN to be accepted by the system, and flags the standard system abbreviations as unacceptable.

f. SYN ABB (NOSTD MIN)
Only the synonyms specified in ABB SYN are acceptable. However, a DELE is valid for ERASE as the minimum DEL is present.

g. SYN ABB (NOSTD EXACT)
Only the synonyms specified in ABB SYN are acceptable. However, DELETE must be entered to be accepted as ERASE.

Error Messages:

E(00001) INCORRECT 'SYN' PARAMETER-LIST
Invalid or mutually exclusive options were specified, (that is, P and PUSER, MIN and EXACT, STD and NOSTD).

E(00002) NO "ABBREV"IATIONS AT ALL (ABBREV NOT IN NUCLEUS)
The system being used does not support synonyms.

E(00003) GIVEN USER SYNONYM FILE NOT FOUND
File "filename filetype filemode" was not found.

E(00004) USER SYNONYM FILE BAD (MUST BE 80-BYTE FIXED RECORDS)

E(00005) FAULTY DATA IN USER SYNONYM FILE

E(xxxxx) DISK ERROR READING USER SYNONYM FILE The error code from RDBUF is returned as xxxxx.

LIBRARIES

CMS provides two types of libraries--macro and text (subroutine). Macro libraries are searched for missing macros during assemblies. The system macro libraries are called SYSLIB MACLIB SY and OSMACRO MACLIB SY. SYSLIB MACLIB contains the CMS macros, and OSMACRO MACLIB contains the OS macros (from SYS1.MACLIB, Rel. 15/16).

Text libraries are searched for missing subroutines or undefined filenames during the LOAD, USE, or REUSE command. The system text libraries are called SYSLIB TXTLIB SY, CMSLIB TXTLIB SY, PLILIB TXTLIB SY, and SSPLIB TXTLIB SY. SYSLIB TXTLIB contains the FORTRAN library, CMSLIB TXTLIB contains the nonerror message FORTRAN subroutines, PLILIB TXTLIB contains the PL1 library and PL1 subroutines, and SSPLIB TXTLIB contains the FORTRAN Scientific Subroutine Package. TXTLIB allows up to 1000 entries per library.

To generate, add to, delete, or replace in macro or text libraries, use MACLIB or TXTLIB respectively. To specify that certain libraries are to be used in addition to or in place of the system libraries, use GLOBAL, which allows up to eight TXTLIBs to be specified.

MACLIB

Purpose:

The MACLIB command (1) generates a macro library, (2) adds, deletes, or replaces macros in an existing library, (3) lists the name, size, and location of macro definitions in a macro library, or (4) compacts a macro library.

Format:

MACLIB	GEN libname filename1...filenameN
	ADD libname filename1...filenameN
	REP libname filename1...filenameN
	DEL libname macroname1...macronameN
	COMP libname
	PRINT libname
	LIST libname

GEN generates the macro library "libname" from the macro definitions in the specified file(s).

ADD adds the macro definitions from the specified file(s) to the existing macro library "libname".

REP deletes the existing macro and adds the new copy

DEL deletes the specified macro entry from the dictionary.

COMP compacts the macro library and removes the macros that have been deleted.

PRINT creates a file called "libname MAP P1" containing the name, size, and location of the macros in "libname", and offline prints the MAP file.

LIST types out the dictionary of the macro library specified by "libname".

libname specifies the filename of a macro library.

filename1...filenameN specify the macro definition files to be used in generating, adding, or replacing in the macro library "libname". Their filetype must be ASP360 or COPY. Replace requires the filename to be identical to the macro in the library.

macroname1...macronameN specify the macros to be DELeted.

Usage:

A macro library is a file that has a filetype of MACLIB and that contains macro definitions and a dictionary. A macro

definition is a group of OS/360 assembler language statements identified by a unique name and used as an expansion for a source statement in an OS/360 assembler language program. The dictionary is generated by the MACLIB command and is made up of macro definition names, the indexes or locations of the macro definitions within the library, and the size or number of card images in each macro definition.

To be accessible to the MACLIB command, a macro definition must first exist on disk in a macro definition file. A macro definition file is a file that contains card images of macro definitions written in the System/360 Assembler language, according to the rules for defining macros. A macro definition file must have a filetype of ASP360.

The MACLIB Command functions are: GEN, ADD, LIST, DEL, REP, PRINT, and COMP.

The GEN form of the MACLIB command creates a macro library and assigns the identifier "libname MACLIB P1" to it. This macro library is generated from the macro definitions in the specified macro definition file(s) having a filetype of ASP360 or COPY.

The ADD form of the MACLIB command appends the macro definitions from the specified macro definition files to the end of the existing macro library. As in the GEN form, the specified macro definition files must have a filetype of ASP360 or COPY, and the specified libname must have a filetype of MACLIB.

The LIST form of the MACLIB command types out the name, index, and the size of each macro definition in the specified library. For an example of the output from the MACLIB LIST command see Figure 37, which lists the CMS system macro library SYSLIB MACLIB.

```
maclib list syslib

MACRO              INDEX      SIZE

CMSTYPE              2          10
CMSAVE              12          42
CMSREG              54          29
MADCALL             83          21
MADDPL             104          15
MADTYPE            119          11
   .                 .           .
   .                 .           .
   .                 .           .
TSTPA              812          11
CMS                823          17
CMLST              840          14
DJCB               854          23
BRETURN            877          14
BCLOSE             891           7
$EXECSWT           898          10
SYSOUT             908          19
R; T=0.08/0.59   11.23.41
```

Figure 37. Example of MACLIB List command

The DEL function removes from the MACRO library directory the name of the specified macro.

The COMP function removes any previously deleted macros from the library.

The PRINT function creates the file "libname MAP P1" from the specified library and prints it on the offline printer.

The REP function replaces the existing code of the specified macro, with the new code from the ASP360 or COPY file of the same name.

Notes:

a. The MACLIB command does not check the macro definitions for errors; it assumes that the definitions are correct and that the card images conform to the assembler language input format.

b. Each specified ASP360 file may contain one or more macro definitions.

c. If a MACLIB file is being generated and a macro library already exists with the same identifier ("libname" MACLIB P1), it is erased and replaced by the library generated with the MACLIB GEN command.

d. No checking for duplicate macro names between the ASP360 files and the existing MACLIB file is performed. The ASP360

files are added to the end of the existing MACLIB file.

e. To print the contents of a macro definition, use the PRINTF command and specify the index number as the starting location and (index+size-1) as the ending location of the macro definition. For an example of PRINTF that prints the CMSTYPE macro definition, see Figure 38.

```
printf syslib maclib 73 86

          MACRO
&LABEL    CMSTYPE &MESSAGE
&LABEL    LA      1,TYP&SYSNDX      SET PARAMETER LIST
          SVC     X'CA'             ISSUE CALL TO TYPLIN
          BC      15,TRA&SYSNDX     BRANCH AROUND PARAMETER LIST
          DS      OF                (ALIGNMENT)
TYP&SYSNDX DC     CL8'TYPLIN'       COMMAND NAME
          DC      AL1(1)            CONSOLE NO.
          DC      AL3(MES&SYSNDX)       LOC. OF MESSAGE
          DC      C'D'              RED INK,FIXED-LOCATION-MESSAGE
          DC      AL3(L'MES&SYSNDX)     LENGTH OF MESSAGE
MES&SYSNDX DC     C&MESSAGE         ACTUAL MESSAGE
TRA&SYSNDX DS     0H
          MEND

R; T=0.05/0.48  11.30.52
```

Figure 38. Printout of the CMSTYPE macro definition using PRINTF

f. A macro can be replaced by "filename ASP360", where filename is the same as the macro name being replaced. The replacement can also contain one or more macros none of which have the same name as the macro being replaced.

Responses:

None.

Examples:

a. MACLIB GEN LSLIB MAC1 MAC2
The macro library LSLIB MACLIB is generated from the macro definitions in the MAC1 ASP360 and MAC2 ASP360 files. If the file LSLIB MACLIB P1 already exists, it is erased and the new file is generated.

b. MACLIB ADD LSLIB IOSUB
The macro definitions in the file IOSUB ASP360 are added to the end of the existing macro library LSLIB MACLIB.

c. MACLIB LIST SYSLIB
The names of the macro definitions in SYSLIB MACLIB are printed out along with each macro definition's index and size (see Figure 37).

Error Messages:

E(00001) INVALID MACLIB COMMAND FUNCTION OR PARAMETER LIST.
An incorrect form of the MACLIB command was given, or, the parameter list is invalid. Reissue the command with the correct format.

E(00002) ERROR WHILE WRITING
An error occurred while writing the library on disk. The library is closed and the command terminated. All of the input card images that were processed before the error occurred are included in the file "libname" MACLIB.

E(00003) ERROR WHILE READING
An error occurred while reading the disk. The library is closed and the command terminated. If the GEN or ADD form of MACLIB had been issued, all of the input card images that were processed before the error occurred are included in the file "libname" MACLIB.

E(00004) INPUT FILE NOT IN CORRECT MACRO FORMAT.
The ASP360 file is not in correct macro format--the card images are out of order, the macro name is longer than eight characters, there is a blank in column 10, or a MEND card is missing from a macro definition. The library is closed and the command terminated. All of the input card images that were processed before the error occurred are included in the file "libname" MACLIB.

E(00005) xxxxxxxx DOES NOT EXIST. IGNORED.
An ASP360 or COPY input file does not exist. Check to see that the specified file has a filetype of ASP360 or COPY.

E(00006) MACLIB FILE SPECIFIED DOES NOT EXIST.
The specified library file does not exist--check to see that the specified libname has a filetype of MACLIB.

E(00007) MACLIB FILE SPECIFIED NOT IN CORRECT FORM.
The specified libname is not a valid MACLIB file as created by the MACLIB command.

E(00009) xxxxxxxx NOT FOUND IN DICTIONARY.
The macro xxxxxxxx does not exist in the specified macro library. The MACLIB command is terminated. The macros in the ASP360 files that were in front of the macro xxxxxxxx have been replaced in the macro library.

E(00010) REPLACEMENT FILE NOT FOUND.
The specified filename does not exist with a filetype of ASP360 or COPY.

E(00011) MACRO ALREADY EXISTS IN MACLIB
The specified macro already exists in the macro library, it is not added again.

TXTLIB

Purpose:

The TXTLIB command (1) generates a text library, (2) adds to an existing text library, (3) deletes from an existing text library, or (4) lists the entry points and control section names, the location, and the size of the TEXT files included in the text library.

Format:

TXTLIB	GENERATE G	libname filename1...filenameN
	ADD A	libname filename1...filenameN
	DELETE D	libname csectname1...csectnameN
	PRINT P	libname
	LIST L	libname

GENERATE
G — creates the text library "libname" from the specified file(s).

ADD
A — adds the contents of the specified file(s) to the existing text library "libname".

DELETE
D — deletes from the text library "libname" the specified control sections (csects).

PRINT
P — creates the file "libname MAP P1" containing a list of the entry points and control section names of the TEXT files in the library, their location, and the size of the TEXT file.

LIST
L — provides the same information as PRINT, but the information is typed out at the terminal instead of being generated into a MAP file.

libname — is the name of the text library to be generated, added to, printed, or listed. The filetype of of libname must be TXTLIB.

filename1...filenameN — specify the file(s) to be used in either generating or adding to a TXTLIB file. Their filetype must be TEXT.

csectname1...csectnameN — specify the csect(s) to be deleted

from a TXTLIB file. Entry points which are not csectnames are ignored.

Usage:

A text library is a file that has a filetype of TXTLIB and that contains a dictionary and the relocatable object code from TEXT files. The dictionary is created by the TXTLIB command which contains the entry points and control section names, their location, and the size of each TEXT file included in the text library. The relocatable object code in TEXT files can be created by the Assembler, FORTRAN, or PLI compiler.

There are five forms of the TXTLIB command: GENERATE, ADD, DELETE, PRINT, and LIST.

The GENERATE form of the TXTLIB command generates a text library from the specified file(s) and assigns to that library the identifier "libname TXTLIB P1". If a file already exists with the identifier "libname TXTLIB P1", the existing file is erased, and the new file is created.

The ADD form of the TXTLIB command appends the contents of the specified TEXT files to the end of the existing text library.

The DELETE form of the TXTLIB command removes the specified control sections from the file "libname TXTLIB". If a TXTLIB file has two control sections with the same name, only the first one is deleted (unless the csectname is given twice in the argument list). The order of the csectnames given in the command argument is immaterial.

The PRINT form of the TXTLIB command generates the file "libname MAP P1" on the permanent disk. If a file already exists with the same identifier, it is erased and the new file created. The "libname MAP" file contains the same information as that in the dictionary of the specified text library and is in the format of a list of entry points and control section names that reside in the text library, their location or index in the file, and their size in number of card images.

The LIST form of the TXTLIB command types out the contents of the dictionary at the terminal and does not generate the file "libname MAP P1". For an example of the TXTLIB LIST command, see "Libraries". The pound sign (#) indicates the first control section name in the file.

Both the PRINT and LIST forms of TXTLIB type out a statement indicating the total number of entry points and control section names that currently exist in the TXTLIB file.

For the usage of text libraries, see the GLOBAL command and "Operating Considerations--Library Usage".

Notes:

a. Each TEXT file that is to be included in the TXTLIB file must consist of one or more control sections with an END card image following each control section. This format is automatically generated by the ASSEMBLE, PL/I, and FORTRAN commands.

b. With the TXTLIB ADD command, the TEXT files are added to the end of the existing TXTLIB file, and no checking for duplicate entry points or control section names is performed.

c. The total number of control section names and entry points in the TXTLIB file cannot exceed 1000. When this maximum number is reached, an error message is typed out. The text library created includes all the text files entered up to (but not including) the one that caused the overflow.

Responses:

xxx ENTRYS IN LIBE
When TXTLIB is issued, the contents of the dictionary of the specified text library are typed out (see Figure 39). The number of entries xxx in the text library are typed out when either TXTLIB LIST or TXTLIB PRINT is issued.

```
txtlib list a
     ENTRY  INDEX  SIZE
  LLHS#          2    21
  LLHS
  SUBD#         23    24
  SUBD
  SUB1
  W#            47    17
  W
       7 ENTRYS IN LIBE
R; T=0.28/0.41 14.13.12
```

Figure 39. Example of the TXTLIB LIST command

Examples:

a. TXTLIB G SSP MEGOP MEG OPS
The file SSP TXTLIB P1 is generated from the files MEGOP TEXT, MEG TEXT, and OPS TEXT. If SSP TXTLIB P1 already exists, it is erased and the new file created.

b. TXTLIB ADD SSP MYROUT
The contents of the file MYROUT TEXT are added to the existing file SSP TXTLIB. No checking for duplicate entries is performed.

c. TXTLIB DEL A SUBD#
The control section SUBD# in text library A is deleted.

TEXT LIBRARIES

This section covers the libraries that are searched for missing subroutines or undefined filenames during the execution of LOAD, USE, or REUSE.

The libraries and their contents are outlined below.

TXTLIB	CONTENTS
SYSLIB	FORTRAN library
CMSLIB	Nonerror message FORTRAN subroutines
PLILIB	PL/I library and subroutines
SSPLIB	FORTRAN scientific subroutines

SYSLIB TXTLIB

The library SYSLIB TXTLIB contains the FORTRAN Library as well as a number of CMS library subroutines. Refer to Form C28-6596, OS/360 FORTRAN IV Library Subprograms for a description of the routines from the FORTRAN library. This section contains the descriptions of the subroutines written for use under CMS.

The subroutines described are:

Entry Point	Description
BLIP	Notifies user of executing program
TRAP	Sets a user's return from an external interrupt
NLSTON/NLSTOF CPNMON/CPNMOF	Allows free format I/O with NAMELIST
DEFINE	Equates a data set reference number with a CMS file and allows that file to be accessed at random
DSDSET	Changes record format and length for FORTRAN disk files
GETPAR	Obtains parameters from CMS commands
ERASE	Removes a specified file from the disk
LOGDSK	Closes all open files and updates the P-disk
RENAME	Changes the identifiers of disk files
TAPSET	Changes default settings of FORTRAN logical tape units
REREAD REREADV	Rereads a record with different formats

This library provides extended error messages. These include a traceback with registers 15,14,0, and 1 and the entry point location. A standard fixup is taken, and a summary of errors is typed at program completion.

BLIP Subroutine

Purpose:

The BLIP subroutine causes a string of from one to eight characters to be typed on the console periodically during CMS operation. The BLIP characters are typed out after every two seconds of CPU execution and give the user an indication of the execution time of his program.

Calling Sequence:

CALL BLIP ('character', count)
The count parameter must be an integer from one to eight indicating the number of BLIP characters.

CALL BLIP ('')
If the count is defaulted, a count of one is assumed.

CALL BLIP (0)
If the first parameter is a zero, the BLIP characters are reset to their nonprinting default setting (a space followed by a backspace).

CALL BLIP (OFF)
If the first parameter is (OFF), the BLIP function is turned off--that is, there is no indication every two seconds of CPU execution.

Usage:

The default setting of the BLIP characters is a sequence of nonprinting characters. If it is desired to have a printed recording of the execution of a program, the BLIP characters should be changed to printing characters (for example, a single dot).

Error Messages:

None.

NLSTON/NLSTOF Subroutines
CPNMON/CPNMOF Subroutines

Purpose:

The NLSTON/NLSTOF subroutines allow a user to input variables to a FORTRAN program in a free format mode without specifying the variable names, as is required with the normal NAMELIST feature of FORTRAN.

Calling Sequence:

CALL NLSTON
To set the namelist feature to the free format mode.

CALL NLSTOF
To set the namelist feature to the normal FORTRAN mode.

Usage:

Under the normal namelist mode, the variable name and an equal sign must be specified with the value for each variable. In addition, the name of the list preceded by an ampersand (&) must be specified before the values for the variables are specified, and the terminating marker &END must be specified after the values for the variables are specified (see Figure 40).

Under the free format namelist mode, each variable is specified in the same order as indicated in the namelist statement, where each variable is delimited by a comma or a tab (see Figure 41).

To use this namelist feature, the routine NLSTON must be called. This causes the free format mode to be in effect. Read-and-write statements are then issued in the standard way.

To return to the normal mode, a call to NLSTOF should be made. Thus, one can go from one to the other as desired.

Note:

CPNMON/CPNMOF are equivalent to NLSTON/NLSTOF.

Examples:

```
&name  a=1., b=2., c=3.
&end
```
Data specified under the standard namelist mode.

```
1., 2., 3.
```
Data specified under the free format namelist mode.

```
printf normal fortran * * 72

REAL*4A
INTEGER B
DIMENSION C(3)
NAMELIST/LIST1/A,B,C
READ(5,LIST1)
WRITE(6,LIST1)
STOP
END
R; T=0.05/0.75  10.02.33

$ normal

EXECUTION BEGINS...

&list1
a=1.,
b=2., c=3., 4.,
5.,
&end
&LIST1
A= 1.0000000 ,B=        2,C= 3.0000000 ,4.0000000 ,5.0000
&END
R; T=0.32/1.02  10.10.44
```

Figure 40. Example of a normal NAMELIST function

```
printf freefrm fortran * * 72

REAL*4A
INTEGER B
DIMENSION C(3)
NAMELIST /LIST1/ A,B,C
CALL NLSTON
READ(5,LIST1)
WRITE(6,LIST1)
STOP
END
R; T=0.03/0.48  10.11.56

$ freefrm
EXECUTION BEGINS...
1,2,3,4,5
&LIST1
&END
A=1.000000 ,B=      2,C=3.0000000 ,4.0000000 ,5.0000
R; T=0.28/0.58  10.12.22
```

Figure 41. Example of a freeform NAMELIST function

DEFINE Subroutine

Purpose:

The DEFINE subroutine defines Fortran disk files that may be accessed randomly and makes a correspondence between a CMS file and a Fortran logical unit number.

Calling Sequence:

CALL DEFINE (dsrn, name, type, recno, recsiz, <&n>)
where

dsrn is the FORTRAN data set reference number to be associated with the defined CMS file. This parameter must be a four-byte integer or integer variable.

name is the filename of the CMS file and it must be eight-bytes in length.

type is the filetype of the CMS file and it must be eight-bytes in length.

recno must be a four-byte integer variable. Each file defined should use a different variable for recno. This parameter indicates which record of the file is to be read or written.

recsiz is a four-byte integer or integer variable that contains the record length. All records must be the same length. If the file already exists, the value of this parameter is reset by the DEFINE routine. The maximum recsiz is 256K.

Usage:

DEFINE is used to make a correspondence between a CMS file and a FORTRAN data set reference number, such that the identifiers FILE FTxxFyyy are not required for FORTRAN disk file reads and writes. Records may then be read or written using a standard FORTRAN statement. Records can be accessed either sequentially or randomly using the sequential I/O statements. Before each READ or WRITE statement the record number must be set to the record wanted. After the execution of the READ or WRITE statement, the record number is automatically incremented to point to the next record in the file. Thus, to READ or WRITE a file sequentially, the variable for recno must be initially set to one before any READ or WRITE statement is executed. All control operations are ignored; however, resetting the variable for recno to one is equivalent to rewinding the file, and setting the variable for recno to recno+1 is equivalent to skipping one record.

Records may be read and written without closing the file.

Thus updating in place is easily accomplished.

For accessing FORTRAN disk files randomly, the normal FORTRAN IV direct access I/O statements should be used instead of the DEFINE subroutine.

Notes:

a. The user should remember that each READ or WRITE statement increments the record number by at least one. If the format statement indicates several records to be read or written, the record number is incremented accordingly.

b. Unformatted READ or WRITE statements may read or write only one record per statement.

c. Each different file should use a different variable for recno.

d. A second call to DEFINE with the same dsrn undefines the old data set and associates the dsrn with the new data set.

e. A maximum of 20 files may be defined for random access.

f. Programs using DEFINE should not use the unformatted PUNCH, PRINT, or READ statements.

Error Messages from the CALL to DEFINE:

ILLEGAL DSRN SPECIFIED FOR DEFINE
This means that dsrn is specified as 0, 5, 6, 7, or greater than 99.

ILLEGAL PARAMETER FOR DEFINE
This means that name or type was not specified, or, that the first byte of this filename is X'00'.

TOO MANY DEFINE FILES
Maximum number of different files is 20.

DATASET DOES NOT HAVE FIXED LENGTH RECORDS

Error Messages during program execution:

DIRECT ACCESS RECORD NO.=0
This means that the record number variable for recno was set to zero.

FATAL ERROR DURING DIRECT ACCESS READ (WRITE)
A fatal disk error occurred during the reading or writing of a defined file and no ERR exit was specified in the user's program.

EOF DURING DIRECT ACCESS READ
The record pointer variable for recno was set to a number larger than the number of records in the file, and no END

exit was specified in the user's program.

Example:

The program below uses CALL DEFINE to access DSRN2 sequentially and DSRN4 directly.

```
            NUM=1
            J=1
            CALL DEFINE(2,'X       ','Y       ',J,80)
            CALL DEFINE(4,'ABC     ','DEFGHIJK',NUM,80)
            READ(4,10) K
            WRITE(6,10) K
            WRITE(2,10) K
10          FORMAT(1H ,14)
            NUM=3
            J=2
            READ (4,10) K
            WRITE (6,10) K
            WRITE (2,10) K
            NUM=2
            READ(4,10) K
            J=3
            WRITE(6,10) K
            WRITE (2,10) K
            STOP
            END
```

DSDSET Subroutine

Purpose:

The DSDSET subroutine enables a user to change the record format and logical record length for FORTRAN disk files.

Calling Sequence:

CALL DSDSET (dsrn, blksize, recfm, lrecl)

where

dsrn is the FORTRAN data set reference number that is used to reference the specified logical unit. This number must be from 1-14 with exception of 5, 6, and 7.

blksize is the byte count for the maximum size of the physical records to be read or written on the specified unit.

recfm is a type number from 1-5 indicating the record format. The type numbers are:
1) fixed record size, unblocked
2) fixed record size, blocked
3) variable record size, unblocked
4) variable record size, blocked
5) undefined record size, no blocking

lrecl is the byte count of the logical record to be read or written. It is used for record format types 1-4. For type 2 records, blksize must be an integral multiple of lrecl; refer to IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide (C28-6817) for a discussion of lrecl and blksize for variable type records.

Usage:

A call to DSDSET is made to change the default settings for data reference numbers 1-14 with the exception of 5, 6, and 7. If it is not required to change the association of a data set reference number with a symbolic tape unit, a call to DSDSET can be made instead of a call to TAPSET. The default settings for FORTRAN disk files are as follows:

Data Set Reference Number	Block Size	Format Type	Logical Record Length
1	80	1	80
2	80	1	80
3	80	1	80
4	80	1	80
8	133	1	133
9	80	1	80
10	80	1	80

Notes:

The default settings for FORTRAN logical units 5, 6, and 7 cannot be changed. Logical unit 5 is the sysin device to read from the terminal, and logical unit 6 is the sysout device to write on the terminal. The default settings are as follows:

Data Set Reference Number	Block Size	Format Type	Logical Record Length
5	80	1	80
6	120	1	120
7	80	1	80

ERASE Subroutine

Purpose:

The ERASE subroutine removes a file from the user's file directory.

Calling Sequence:

CALL ERASE (fname,ftype,<fmode>)
where

fname is the filename of the file to be erased.

ftype is the filetype of the file to be erased.

fmode is the filemode of the file to be erased.

Usage:

ERASE is used to erase a file (that is, to remove it from the file directory).

Examples:

a.
```
CALL ERASE ('ABC     ','DEF     ')
CALL EXIT
END
```
This sequence causes the reference to ABC DEF to be removed from the file directory.

b.
```
REAL*8 NAME,TYPE
DATA NAME/'ABC     '/,TYPE/'DEF     '/
CALL ERASE (NAME,TYPE)
CALL EXIT
END
```
This sequence causes the file ABC DEF to be erased from the file directory.

GETPAR Subroutine

Purpose:

The GETPAR subroutine obtains the parameters specified when the command or program was called from CMS.

Calling Sequence:

CALL GETPAR (name, number <, 'RITE' > <,&last>)

where

name is the variable (real*8) that is to take the value of the parameter indicated by the argument "number".

number is the parameter number (inter*4) in the initial parameter list of the parameter desired; it may be any nonnegative value, with zero indicating the command or program name on initial entry. If number exceeds the number of parameters in the string, no parameter is passed, and control passes to statement 'last' if &last is specified.

'RITE' is an optional literal which causes the current parameter to be right-justified in its doubleword field, with leading blanks supplied. This is useful in reading numeric parameters.

&last is an optional statement label to which control is passed if "number" exceeds the number of parameters specified.

Usage:

If GETPAR is called with item number equal to zero, the command name or program entry is returned. If the error return is not specified and the parameter corresponding to the item number was not specified, the routine returns without storing any parameter. Note that blanks are not returned for an unspecified parameter.

Example:

CALL GETPAR (PARAM (I), I, &100)
Parameter I is stored in the array PARAM if it was specified; otherwise, control passes to statement 100.

LOGDSK Subroutine

Purpose:

The LOGDSK subroutine closes all open files and writes the file directory onto the permanent disk.

Calling Sequence:

CALL LOGDSK

Usage:

The user should call LOGDSK during the execution of his program if he wants to cause new or modified files to be permanently written onto the permanent disk before the completion of the program and the return to the CMS command environment.

Error Exit:

If the directory cannot be written out, the virtual machine is put in disabled wait state and control is passed to CP.

RENAME Subroutine

Purpose:

The RENAME subroutine allows a file identifier to be changed from within a FORTRAN program.

Calling Sequence:

CALL RENAME (oldfn, oldft, newfn, newft)
where

oldfn is the filename of the file to be changed.

oldft is the filetype of the file to be changed.

newfn is the filename to be given to the file.

newft is the filetype to be given to the file.

Usage:

RENAME is used to change the file identifiers of a file from within a FORTRAN program.

Examples:

a.
```
CALL RENAME ('ABC     ','DEF     ','NEW     ','DEF     ')
CALL EXIT
END
```
This program takes the file ABC DEF and changes its identifiers to NEW DEF.

b.
```
REAL*8 NAME,TYPE,NNAME,NTYPE
DATA NAME/'ABC     '/,TYPE/'DEF     '/
DATA NNAME/'NEW     '/,NTYPE/'DEF     '/
CALL RENAME (NAME,TYPE,NNAME,NTYPE)
CALL EXIT
END
```
This program also changes file ABC DEF to NEW DEF.

REREAD Subroutine

Purpose:

The REREAD subroutine provides a facility for rereading a record with different format statements without performing any input/output operations.

Calling Sequence:

CALL REREAD (dsrn, <blksize>)

where

dsrn specifies any data set reference number from 1 to 99, except unit 5, 6, or 7.

blksize specifies the blocksize for the reread record. If the blocksize is omitted, it is defaulted to 140 bytes.

Usage:

A call to REREAD is made when it is desired to specify a reread unit or to change the data set reference number or the blocksize. To read a record from the reread unit a second or subsequent time, a REWIND n statement must be executed before the READ statement; if a subsequent reread is issued without executing a REWIND statement, an END OF FILE condition results. Any input/output statements can be issued between a write and a read on the reread unit.

Error Returns:

None.

TAPSET Subroutine

Purpose:

The TAPSET subroutine changes the default settings for the FORTRAN logical tape units.

Calling Sequence:

```
CALL TAPSET (tapno, dsrn, <blksize>, <modeset>,
             <recfm>, <lrecl>)
```

where

tapno is an integer from 1-4 indicating the virtual tape unit TAP1, TAP2, TAP3, or TAP4 corresponding to device address 180, 181, 182, or 183.

dsrn is the FORTRAN data set reference number that is used to reference the specified tape unit. This number must be from 1-14 with the exceptions of 5, 6, and 7.

blksize is the byte count for the maximum size of the physical records to be read or written on the defined unit.

modeset is a code number for setting the mode mask for 7-track operations. The mask is ignored on all 9-track operations.

The code numbers range from 1-15 in groups of 5. Codes 1-5 are for 800 bpi, 6-10 for 556 bpi, and 11-15 for 200 bpi. Within each group of 5, the mode setting is as follows:

1) odd parity, converter on, translator off
2) odd parity, converter off, translator on
3) odd parity, converter off, translator off
4) even parity, converter off, translator on
5) even parity, converter off, translator off

recfm is a type number from 1-5 indicating the record format. The type numbers are:

1) fixed-record size, unblocked
2) fixed-record size, blocked
3) variable record size, unblocked
4) variable record size, blocked
5) undefined record size, no blocking

lrecl is the byte count of the logical record to be read or written. It is used for record format types 1-4. For type 2 records, blksize must be an integral multiple of lrecl; refer to IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide (C28-6817) for a discussion of lrecl and blksize for variable type

records.

If the parameter blksize, modeset, recfm, or lrecl is zero or omitted, the parameter is defaulted to that which was previously set, or defaulted for the tape unit.

Usage:

FORTRAN data set reference numbers 11-14 are the standard logical tape units. These units correspond to symbolic devices TAP1-TAP4, and to virtual devices 180-183. The default settings are as follows:

Vir Dev	Sym Dev	Data Set Ref	Blk Size	Modeset Code	Format Type	lrecl
180	TAP1	11	80	1	1	80
181	TAP2	12	133	1	1	133
182	TAP3	13	800	1	2	80
183	TAP4	14	1330	1	2	133

The parameters blksize, modeset, recfm, and lrecl are associated with the dsrn. If a call to tapset is made with these settings defaulted, the settings previously associated with the dsrn will still be in effect. If a call to TAPSET associates a tape unit with a dsrn which is already associated with a tape, the tape previously associated with that dsrn will not be associated with any logical unit. If a call to TAPSET associates a tape unit with a dsrn different from that with which it was previously associated, the previous dsrn will be associated with disk files. If a call to TAPSET associates a tape unit with a dsrn which was previously associated with a disk file, a read or write to that dsrn will be directed to the tape unit.

If the blocksize is defaulted, the following message is typed at the terminal or on sysout.

TAPSET-BLKSIZE IS ZERO OR NEGATIVE; DEFAULT USED.

If the logical record length parameter is defaulted, the following message is typed at the terminal or on sysout:

TAPSET-LRL IS ZERO OR NEGATIVE: DEFAULT USED.

Note:

In the current CMS, TAP3 and TAP4 are not defined, therefore data set reference numbers 13 and 14 should not be used.

Return Message:

TAPSET-ONLY ONE ARGUMENT; CALL IGNORED.
The tape number, tapno, and the data set reference number, dsrn, are required parameters. If a call to TAPSET is made

with only one argument, the routine returns without altering the settings for FORTRAN logical tape units.

Error Exits: (program terminates)

ABEND 1 TAPESET-INVALID TAPE NUMBER; TERMINATING.
Only tape numbers 1-4 are legal parameters.

ABEND 2 TAPSET-DSRN IS GREATER THAN 14; TERMINATING.

ABEND 3 TAPSET-DSRN OF ZERO IS ILLEGAL; TERMINATING.

ABEND 4 TAPSET-MODESET CODE IS GREATER THAN 15; TERMINATING.
Only the modeset codes from 1-15 are legal.

ABEND 5 TAPSET-RECFM CODE IS GREATER THAN 5; TERMINATING.
Only the record format type numbers from 1-5 are legal.

ABEND 6 TAPSET-DSRN5, 6, AND 7 ARE ILLEGAL FOR TAPE UNITS.

TRAP Subroutine

Purpose:

The TRAP subroutine sets a user's return for an external interrupt. This return overrides the call to DEBUG on an external interrupt.

Calling Sequence:

CALL TRAP (extrap)
where extrap is the name of an external routine which is transferred to on an external interrupt.

CALL TRAP (0)
If the argument to TRAP is zero, the external interrupt return is reset to go to DEBUG.

Usage:

The interrupt routine should set a flag which should be examined by the main line program. After the flag is set, the interrupt routine should issue a RETURN to the executing program. The main line program should periodically examine the trap flag to determine whether an external interrupt has occurred.

Example:

The following sample program illustrates the use of the TRAP procedure, using a FORTRAN pseudo sense switch as the flag for communicating between the interrupt routine and the main line program.

```
          EXTERNAL EXTRAP
          CALL TRAP (EXTRAP)
          CALL SLITE (0)
          DO 10 K=1,10000
          CALL SLITET(2,1)
          IF (1.EQ.1)GOTO 20
          IF (1.NE.2) GOTO 30
          IF (MOD(K,50) .EQ0)WRITE(6,100)K
 10       CONTINUE
 20       WRITE (6,200)
          STOP
 30       WRITE (6,300)
          STOP
 100      FORMAT(1X,115',TIMES'),
 200      FORMAT('END OF RUN')
 300      FORMAT('SENSE LITE ERROR')
          END
          SUBROUTINE EXTRAP
          CALL SLITE(2)
          RETURN
          END
```

```
fortran testtrap
T=2.23 15:26:58

$ testtrap
EXECUTION BEGINS...
          50 TIMES
      100 TIMES
      150 TIMES
      200 TIMES
      250 TIMES
      300 TIMES
e
END OF RUN
IHC0021 STOP        0
```

CMSLIB (Non-Error-Message FORTRAN Library)

The library CMSLIB TXTLIB contains the non-error message FORTRAN subroutines. To specify that these subroutines are to be used, issue the command

```
GLOBAL T CMSLIB SYSLIB
```

The routines contained in CMSLIB TXTLIB are listed below.

```
IHCFCOMH
IBCOM#
FDIOCS#
INTSWTCH
IHCCOMH2
SEQDASD
IHCDIOSE
DIOCS#
IHDFIOSH
FIOCS#
IHCFINTH
ARITH#
ADJSWTCH
IHCTRCH
IHCERRM
DEFINE
REREAD
REREADV
IXCCMS
CMSFORTR
```

The entries DEFINE, REREAD, REREADV, IXCCMS, and CMSFORTR are included in both SYSLIB TXTLIB and CMSLIB TXTLIB, since they are dependent on whether the extended error message or non-error message subroutines are used.

PLILIB--PL/I Library

The library needed for PL/I execution is PLILIB. It resides on the system disk. To specify that this library is to be used, issue the command

```
GLOBAL TXTLIB PLILIB
```

SSPLIB--FORTRAN Scientific Subroutine Library

SSPLIB is a text library residing on the system disk that contains Version 2 of the FORTRAN Scientific Subroutine Package. For further information on these subroutines, refer to System/360 Scientific Subroutine Package--Application Description (H20-0205).

SYSLIB--System Macro Library

The system macro library of CMS is called SYSLIB MACLIB and it resides on the system disk. This macro library is used by the ASSEMBLE command to expand undefined macros.

For a discussion of the CMS macros in SYSLIB MACLIB, refer to "ASSEMBLE--CMS Macros".

OSMACRO--OS Macro Library

OSMACRO MACLIB contains the OS macros. For a discussion of these, refer to "ASSEMBLE--OS Macros".

To generate additional macro libraries and modify them, see the MACLIB command. To specify that only certain macro libraries are to be used during ASSEMBLE, see the GLOBAL command.

CP-67 CONSOLE FUNCTIONS

To communicate with the Control Program (CP) and to simulate the computer console, the user can issue CP console functions. These console functions allow the user to perform such operations as: initially loading the desired system (that is, CMS), dumping core selectively, closing out files on unit record devices, displaying core and machine conditions, communicating with the operator in the computer room, as well as with other users, querying the number of users on the system as well as other parameters, controlling messages typed on his terminal, and beginning program execution at a certain core location. The user also has the ability to simulate the System/360 console system reset key, to cause the Control Program to simulate an external interrupt, to replace specified parts of core, to ready specified I/O devices, and to release the virtual machine. Additional I/O devices can also be added to a specific virtual machine and then be released for use by another virtual machine. The user can direct his spooled output to a particular device, he can purge spooled input and output files, and he can place his terminal in a dormant state, so as to receive messages when he is not actively using the terminal.

The Control Program environment is entered on the completion of the login procedure (see "Terminal Usage-Logging Procedures"). After the message READY AT xx.xx.xx ON xx/xx/xx is typed at the user's terminal, where xx.xx.xx is the time of day and xx/xx/xx is the date, the keyboard is unlocked and the Control Program is then ready to accept console functions. By typing IPL CMS or IPL 190, the user loads the Cambridge Monitor System and he can begin issuing CMS commands. To load other operating systems, type IPL xxx, where xxx is the address of that operating system's systems residence device; the terminal then becomes the operator's console for that operating system.

To reenter the Control Program from CMS, or other system environment, the user must hit ATTN once. This causes the keyboard to unlock on the 2741, 1050, and TTY 33 or 35, and permits the Control Program to accept console functions. If ATTN is hit again while in the Control Program, the keyboard unlocks for an input line to be entered, such as a CMS command or a line of data for the program being loaded and executed; control is then transferred to the environment from which the control program was entered. For instance, if the CMS command ASSEMBLE LOOPX had been issued, then ATTN was hit twice and PRINTF LOOPX LISTING entered, the assembly would continue, and as soon as it terminated, the LISTING file would be typed out. If there is no desire to enter an input line once the keyboard has unlocked, hit carriage return and then control is transferred (as above), to the environment from which the Control Program was entered. The console function BEGIN is also used to leave the Control Program; it causes control to transfer either to the

environment from which CP was entered, or to a specific core location, if one is specified. See the CP BEGIN command for more details.

CONSOLE FUNCTION DESCRIPTIONS

CP console functions are issued in the Control Program environment. Input can be entered in either uppercase or lowercase. Entering input in lowercase enables the user to distinguish the input from the output, since all output is typed in uppercase.

The Control Program has a character-delete symbol and a line-delete symbol, both of which are the same as the default characters in CMS. The AT character (@) deletes the immediately-preceding character from the input line and itself. It may be used repetitively to delete a string of characters. The cent character (¢) or shift K (left bracket) on the teletype, deletes all preceding characters in the line and itself. A character-delete symbol (@) cannot be used to delete a line-delete symbol (¢). Note that if the character-delete and line-delete symbols are redefined in CMS, they are <u>not</u> redefined in CP.

The Control Program also has a logical carriage return or line-end character to allow multiple console functions to be typed on one line. The logical line-end character is the # and it cannot be changed. If the logical line-end character is redefined in CMS via the LINEND command, it does <u>not</u> redefine it for CP-67.

One or more blanks must be used to delimit operands or arguments specified with the console functions.

A console function is accepted and executed if a carriage return occurs and the keyboard is unlocked, unless the console function caused control to pass to another environment, in which case the response is that of the new environment. If a console function is rejected, the message INVALID CP REQUEST is typed. If invalid operands are specified with a console function, the message BAD ARGUMENT xx is typed, where xx is the argument number. If only one operand is specified and it is invalid, the console function is not executed. If multiple operands are specified, and part of them are invalid, the valid operands specified before the first invalid operand are executed properly. If a null line is entered to CP (that is, a carriage return with no previous characters in the line) the confirming message CP is typed.

The console functions that are available for the user are listed below.

CONSOLE FUNCTIONS

BEGIN initiates execution at either a specified core location or at the location specified on the current PSW (which is normally the location from which the Control Program was last entered)

CLOSE releases the spooling areas from the specified multiplexer devices, and the actual I/O operation is performed for any output

DETACH removes the attached device whose address is specified from the virtual machine configuration

DISCONN causes the terminal to be disconnected from the virtual machine, while the virtual machine continues to run

DISPLAY types out in hexadecimal the contents of the virtual machine's core storage, general-purpose registers, floating-point registers, and/or PSW

DUMP prints in hexadecimal on the virtual printer the contents of core storage, general-purpose registers, floating-point registers, and/or the PSW

EXTERNAL simulates an external interrupt on the virtual machine and returns control to that machine

IPL causes the Control Program to simulate an Initial Program Load sequence on a specified device, and clears all virtual storage to zeros.

IPLSAVE causes the Control Program to simulate an Initial Program Load sequence on a specified device without first zeroing core.

LINK attaches the specified device to the requester's virtual machine, based on information contained in the CP-67 User Directory

LOGOUT removes the user from the system by releasing the virtual machine and any attached devices, by clearing the temporary disk area, and by closing the spooling areas

MSG sends a specified message to a specific user

PURGE allows the user to delete <u>all</u> spooled files (not currently processed for output) from his virtual printer or punch, or to delete <u>all</u> his spooled input for his virtual card readers without reading the data

QUERY types out the number of users logged on or dialed,

the identification and terminal address of the users logged on, the terminal address of a specific logged-on user (or a message that he is not logged on), the log message sent by the operator, the number of spooled input and output files currently held by CP for the user, or the amount of connect, virtual CPU, and total CPU times used since login.

READY simulates a device end for the specified virtual address

RESET simulates the system reset key on the system control panel

SET allows the user to save the card file in his virtual card reader, to control the messages and warnings typed at his terminal, and to control his running status

SLEEP allows the user to place his terminal in a dormant CP mode such that he may receive messages without hitting carriage return

SPOOL allows the user to direct his spooled output to a specific unit record device at the computing facility and to control the nature of reading spooled input files

STORE replaces the contents of specified locations in core storage, general-purpose registers, floating-point registers, and the PSW

XFER controls the passing of files between users

BEGIN

Purpose:

BEGIN initiates execution at either a specified core location or at the location specified in the current PSW (which is normally the location from which the Control Program was last entered).

Format:

BEGIN	<hexloc>
B	

hexloc is the hexadecimal core location at which execution is to begin.

Usage:

BEGIN transfers control to the specified hexadecimal core location. If a hexadecimal core location is not specified, execution resumes from the location contained in the current program status word (PSW)--this is normally from the location at which the Control Program was last entered, unless the PSW was previously changed. For example, if the Console Program had been entered from CMS by hitting ATTN, and BEGIN is issued without specifying a core location, CMS is entered at the location from which it was left.

Responses:

BAD ARGUMENT xx
An invalid hexadecimal core location was specified in argument number xx. Correct the request and issue again.

Any other response will be that of the new environment.

Examples:

a. b
Execution resumes at the location from which CP was last entered.

b. b 15000
Control is transferred to hexadecimal core location 15000.

CLOSE

Purpose:

CLOSE releases the spooling areas from the specified multiplexer devices, and the actual I/O operation is performed for any output.

Format:

CLOSE	devadd
C	

devadd is the device address of the virtual card reader, card punch, and printer, which are normally 00C, 00D, and 00E, respectively, for CMS.

Usage:

In releasing the spooling areas for the virtual card punch or printer, CLOSE punches or prints the contents of the spooling area on the real device when that device becomes available. The printed output is preceded by one page which contains the user's identification, the date, and the time of day.

The punched output is preceded by one card which contains the user's identification, the date, and the time of day, in the following format:

columns 1-10	CP67USERID
columns 13-20	user's id, left-justified
columns 25-32	date (mm/dd/yy)
columns 37-44	time (hh.mm.ss)
columns 49-80	asterisks (*)

This punched ID card is in the correct format to be used as the input identification card required by CP for spooled card reader input for users.

In releasing the spooling area for the virtual card reader, CLOSE assumes that the user has read all the cards he wants from the current card deck and frees the spooling area. If another command to read a card is then executed, CP returns either the first card of the next deck it has read for that user (if any), or reader-not-ready.

Notes:

a. CLOSE 00E must be given after the DUMP console function has been issued.

b. Logging out from CP automatically issues a CLOSE for the virtual card reader, printer, and punch and releases all

spooling areas.

Responses:

BAD ARGUMENT xx
An invalid device was specified in argument number xx.

Examples:

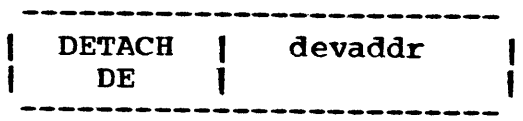
c e
The spooling area for the virtual printer is printed and released.

DETACH

Purpose:

DETACH removes the specified attached device from the virtual machine configuration.

Format:



devaddr specifies the address of a device that had previously been attached to the virtual machine.

Usage:

DETACH removes the specified device from the user's virtual machine configuration for the current session. DETACH is also used in conjunction with the console function ATTACH, which can only be issued by the CP operator. Once a device has been ATTACH'ed to a virtual machine (see "Console Function Applications" for a discussion of attaching I/O devices), the responsibility is left to the user to remove or detach the specified device from his configuration. As long as the device is attached to him, it is unavailable for use by any other user.

DETACH removes the specified device from the configuration. As soon as the device is detached, the message DEV devadd DETACHED is typed out at the terminal and the device is free for use by other users. Also, a message is automatically typed out to the CP operator specifying the device that is free. If the device address detached is that of a tape drive, the tape is rewound and unloaded.

Responses:

a. BAD ARGUMENT xx
An invalid argument was specified.

b. DEV devadd DETACHED
The specified device address is no longer usable. The device must be attached again for further use.

c. NONEXISTENT UNIT
The specified unit does not exist in the virtual machine configuration. Either it has been detached or it never was attached.

Examples:

a. DETACH 181
```
  console function:     detach 181
  response:             DEV 181 DETACHED
```
Unit 181 is detached from the virtual machine configuration.

b. DET 00C
The spool reader 00C is removed from the users virtual machine.

DISCONN

Purpose:

DISCONN causes the terminal to be disconnected from the virtual machine, and the virtual machine continues to run.

Format:

```
-------------------------
| DISCONN |  <anything> |
-------------------------
```

anything is any nonblank character string that signifies the phone line is not to be hung up upon DISCONN'ecting

Usage:

DISCONN causes the terminal to be disconnected from the virtual machine. The virtual machine continues to run as though the user had issued BEGIN. Any "writes" to the terminal (virtual console) are ignored. The virtual machine is automatically logged out if a "read" is attempted from the terminal, or if the virtual machine goes into the disabled wait state. The DISCONN'ected virtual machine runs with low priority in Q2 of the dispatcher.

If the <anything> option is selected, the line is not disabled. The terminal can then be used to LOGIN with another userid, or to DIAL into a multiaccess system.

At a later time, a user can "reconnect" with his DISCONN'ected machine simply by following the normal login procedure. This reconnection can be made from any terminal. The message:

```
RECONNECT AT xx/xx/xx ON xx/xx/xx
```

is printed upon reconnection and the terminal is placed in console function mode. To continue running the virtual machine, a BEGIN is required.

DISPLAY

Purpose:

DISPLAY types in hexadecimal the contents of the virtual machine's core storage, general-purpose registers, floating-point registers, and/or program status word. For virtual 360/67 only, DISPLAY also types the contents of the control registers.

Format:

DISPLAY	<hexloc> <hexloc1-hexloc2> <Lhexloc>
D	<Lhexloc1-hexloc2> <Greg> <Greg1-reg2>
	<Yreg> <Yreg1-reg2>
	<Xreg> <Xreg1-reg2>
	<PSW>

hexloc is interpreted the same as Lhexloc.

hexloc1-hexloc2 is interpreted the same as Lhexloc1-hexloc2.

Lhexloc displays in hexadecimal the contents of specified hexadecimal location hexloc.

Lhexloc1-hexloc2 displays in hexadecimal the contents of the specified hexadecimal locations hexloc1-hexloc2.

Greg displays in hexadecimal the contents of general-purpose register reg, where reg is an integer ranging from 0-15.

Greg1-reg2 displays in hexadecimal the contents of genreal-purpose registers reg1-reg2, where reg1 must be less than reg2, and each reg must be an integer ranging from 0-15.

Yreg displays the contents of the floating-point register reg, where reg is an integer ranging from 0-7. If reg is odd, it is adjusted to the preceding even integer. The contents are displayed in hexadecimal in two forms: the internal format and the E format.

Yreg1-reg2 displays the contents of the floating-point registers reg1-reg2, where reg1 must be less than reg2 and each reg must be an integer ranging from 0-7. If reg1 and/of reg2 is an odd integer, each is adjusted to the preceding even integer. The contents are displayed in hexadecimal in two forms: the internal format and the E format.

Xreg	displays in hexadecimal the contents of the control register reg, where reg is an integer ranging from 0-15. Applies to virtual 360/67 only.
Xreg-reg2	displays in hexadecimal the contents of control registers reg1-reg2, where reg1 must be less than reg2, and each reg must be an integer ranging from 0-15. Applies to virtual 360/67 only.
PSW	displays as two hexadecimal words the contents of the program status word (PSW).

Usage:

Before the specified contents of core storage is displayed, alignment is made to the nearest fullword boundary. The output is typed in multiples of a fullword (eight hexadecimal characters) and all information is displayed in hexadecimal.

When DISPLAY is issued, the arguments can be combined in any order desired, separated by one or more blanks, and up to one input line in length. If neither G, Y, X, nor L precedes the specified number, L is assumed. If an invalid argument is specified, a message is typed out and the console function terminates. If any valid arguments were specified before the invalid one, they are executed properly.

Notes:

a. The contents of core, the registers, and the PSW are not altered by issuing DISPLAY.

b. No imbedded blanks are permitted within an argument.

c. Xreg or Xreg1-reg2 is an invalid argument if the virtual machine is not a System 360/Model 67.

Responses:

The specified information is typed out.

BAD ARGUMENT xx
An invalid argument was specified. If valid arguments were specified before the first invalid one, they were executed properly.

Examples:

a. display 112402
   L 12400 = 9208F17E
The core storage location 12403 is adjusted to the nearest fullword boundary, 12400, and one fullword is displayed in

hexadecimal.

b. display 12000-12010
```
   L 12000 = 05C050E0 C7EE5830 CBB65833 00385030 CB0E1821
```
The contents of hexadecimal location 12000-12010 is displayed in hexadecimal in multiples of fullwords.

c. display g1
```
   G   1 = 00011C98
```
The contents of general-purpose register 1 is displayed in hexadecimal.

d. d g1-5
```
   G   1 = 00011C98 00008990 00008B40 00000082 00000400
```
The contents of genreal-purpose registers 1-5 are displayed in hexadecimal.

e. d x0
```
       X   0=00016C80
```
The contents fo control register 0 are displayed in hexadecimal.

f. d x4-6
```
       X   4  FF000000  00000000  F08000FF
```
The contents fo control registers 4-6 are displayed in hexadecimal.

g. d x2
```
   BAD ARGUMENT 01
```
The virtual machine is not a 360/67, therefore it has no control registers.

h. d y2
```
   Y   2 = 0004560000000000     .14627299646232350 E-78
```
The contents of floating-point register 2 are displayed in hexadecimal in two forms: the internal format and the E format.

i. d y1-6
```
   Y   0 = 1230000000000000     .76468411734357709 E-56
   Y   2 = 0004560000000000     .14627299646232350 E-78
   Y   4 = 000000AB11100000     .88057543452313567 E-82
   Y   6 = 0099999999990000     .51817011330519540 E-77
```
The contents of floating-point registers 0, 2, 3, 6 are displayed in hexadecimal in two forms: the internal format and E format, after the specified register 1 is adjusted to the preceding even integer 0.

j. d psw
```
   PSW   =  00000000 80001374
```
The contents of the program status word (PSW) are displayed in hexadecimal.

k. d g1-4 140-50 y4 psw x2-4
```
   G    1 = 00011C98 00008990 00008B40 00000082
   L   40 = 00011CC0 0C000000 00011CB0 FFFFFFFF 7FFFDCCE
   Y    4 = 0000000000000000      .00000000000000000 E 00
```

```
   PSW    = 00000000 80001374
   X    2 = 0003AC28 00000000 FF000000
```

The contents of general-purpose registers 1-4, hexadecimal locations 40-50, floating-point register 4, the program status word, and control registers 2-4 are all displayed in hexadecimal with one console function.

DUMP

Purpose:

DUMP prints in hexadecimal on the virtual printer the contents of core storage, general-purpose registers, floating-point registers, and/or the program status word. For virtual 360/67 only, DUMP also prints the contents of the control registers.

Format:

DUMP DU	<hexloc> <hexloc1-hexloc2> <Lhexloc> <Lhexloc1-hexloc2> <Greg> <Greg1-reg2> <Yreg> <Yreg1-reg2> <Xreg> <Xreg1-reg2> <PSW>

hexloc is interpreted the same as Lhexloc.

hexloc1-hexloc2 is interpreted the same as Lhexloc1-hexloc2.

Lhexloc dumps in hexadecimal the contents of the hexadecimal location hexloc.

Greg dumps in hexadecimal the contents of the general-purpose register reg, where reg is an integer ranging from 0-15.

Greg1-reg2 dumps in hexadecimal the contents of the general-purpose registers reg1-reg2, where reg1 must be less than reg2, and each reg must be an integer ranging from 0-15.

Yreg dumps the contents of the floating-point register reg, where reg is an integer ranging from 0-7. If reg is odd, it is adjusted to the preceding even integer. The contents are dumped in hexadecimal in two forms: the internal format and the E format.

Yreg1-reg2 dumps the contents of the floating-point registers reg1-reg2, where reg1 must be less than reg2, and each reg must be an integer ranging from 0-7. If reg1 and/or reg2 is an odd integer, each is adjusted to the preceding even integer. The contents are dumped in hexadecimal in two forms: the internal format and the E format.

Xreg dumps in hexadecimal the contents of the control register reg, where reg is an integer ranging from 0-15. Applies to

virtual 67 only.

Xreg1-reg2 — dumps in hexadecimal the contents of the control registers reg1-reg2, where reg1 must be less than reg2, and each reg must be an integer ranging from 0-15. Applies to virtual 67 only.

PSW — dumps as two hexadecimal words the contents of the program status word (PSW).

Usage:

Before the specified contents of core storage is dumped, alignment is made to the nearest fullword boundary. The output is printed in multiples of a fullword (eight hexadecimal characters) and all information is dumped in hexadecimal.

DUMP prints the specified information on the virtual printer. In order for CP to close the virtual printer (to release the spooling area) and print the dumped information on the real printer, the console function CLOSE 00E must be issued. CLOSE need only be issued once for the printer after all the variations of DUMP have been given.

When DUMP is issued, the arguments can be combined in any order desired, separated by one or more blanks, and up to one input line in length. If an invalid argument is specified, a message is typed out and the console function terminates. If any valid arguments were specified before the invalid one, they are executed properly.

Notes:

a. The contents of core storage, the registers, and the PSW are not altered by issuing any form of DUMP.

b. No imbedded blanks are permitted within an argument.

c. Xreg or Xreg1-reg2 is an invalid argument if the virtual machine is not allowed to run in Model 67 mode.

Responses:

After the completion of valid DUMP console functions, the keyboard is unlocked.

BAD ARGUMENT xx
An invalid argument was specified. If valid arguments were specified before the invalid one, they were executed properly.

Examples:

a. dump g1-15
dump y0-6
dump 0-80
dump psw
close 00e

General-purpose registers 1-15, floating-point registers 0, 2, 4, and 6, core storage locations 0-80, and the program status word are dumped on the virtual printer. To close the virtual printer and print on the real device, "close 00e" was issued. The dump output is shown in Figure 42.

b. dump x0-15
BAD ARGUMENT 01

The virtual machine is not a System 360/Model 67; therefore it has no control registers.

```
**************************************************************************************************************************
**************************************************************************************************************************
**************************************************************************************************************************
**************************************************************************************************************************
**************************************************************************************************************************

CP67USERID  BOYD        09/30/70    21.04.42

G      0 = 00000000   00000218   C3D6D5F1   0000C748   00000000   00000000   0CCC001C   CC000000
G      8 = 000004A4   00000400   0000000C   00000490   C000C658   0000C658   4000C742   400021A4
Y      C = 0000000000000000       .0000000000000000 E 00
Y      2 = 0000000000000000       .0000C0000000000000 E 00
Y      4 = 000000C000000000       .0000003000000000 E 00
Y      6 = 0000000000000000       .0000000C00000000C E 00
    0 = 0000C658   0000CEE0   000032F8   60000050   0203FB10   0000A3B0   FF000080   8003F298    *..F........8-...&......I.......2Q*
   20 = 000400CA   400096DA   00000005   60C0D0C2   C0000000   00000000   FF040009   0000223E    *.... .0.....-.....................*
   40 = 00000208   0C000000   000001F8   00000000   0000ED1C   00000000   00000000   0000BD00    *.............8.....................*
   60 = 00040000   00002288   00040000   0000BB00   00000000   0000BC2C   00040CCC   000C05AC    *........H.........................*
   80 = 00000000   00002288   00040000   0000BB00   00000000   0000BC2C   00040000   000005A0    *.................IDLE    ..0.....*
PSW     = FF060000 0000223E
```

Figure 42. Output from DUMP console function

EXTERNAL

Purpose:

EXTERNAL simulates an external interrupt on the virtual machine and returns control to that machine.

Format:

Usage:

On a real machine, an external interrupt is a means by which the central processing unit responds to signals from the interrupt key on the computer console. EXTERNAL simulates this external interrupt, which is then reflected to the virtual machine, and control is returned to that machine. If the user had previously IPL'ed CMS, the Debug environment of CMS is entered from the Control Program.

Responses:

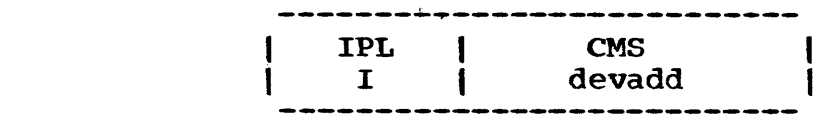
DEBUG ENTERED, EXTERNAL INT.
CMS was previously IPL'ed. When the external interrupt occurred, the Debug environment was entered.

IPL

Purpose:

IPL causes the Control Program to simulate an Initial Program Load (IPL) sequence on a specified device. Core is zeroed before the IPL, and all pages of the virtual machine are freed.

Format:



CMS specifies that a saved copy of the Cambridge Monitor System is to be brought into core.

devadd specifies the address of the device to be IPL'ed.

Usage:

IPL simulates the IPL button on the computer console by zeroing core on the virtual machine and causing a record to be read from the specified I/O device and executed.

During the login procedure of CP when the message

READY AT xx.xx.xx ON xx/xx/xx

is typed out, where xx.xx.xx is the time of day and xx/xx/xx is the date, CP is ready to accept any console function. By issuing IPL CMS or IPL 190, the Cambridge Monitor System (CMS) is loaded, and the CMS command environment is entered.

The IPL 190 console function loads in a copy of the nucleus which resides on disk 190. Periodically, a copy of this nucleus is saved by a CP utility called SAVESYS. This saves the nucleus at a point at the end of the IPL sequence, thereby allowing shared pages of the nucleus and faster response time. The IPL CMS console function loads in this saved copy of the CMS nucleus. This means that if the nucleus on 190 has been modified since the last copy was saved, the versions referenced by IPL 190 and IPL CMS are different.

Responses:

BAD ARGUMENT xx
An invalid device address was specified.

ERROR DURING IPL SIO
CP is unable to load from the specified device. If the card reader had been specified, check to see that cards had been read into the virtual card reader by the computer operator.

UNABLE TO IPL SPECIFIED UNIT TYPE
The specified unit could not be IPL'ed. Check to see if the specified-device address is included in the virtual machine configuration.

Any other responses are those of the new environment that is being loaded.

IPLSAVE

Purpose:

Same as IPL except the existing core is not cleared. See the console function IPL for further information.

Format:

```
----------------
| IPLSAVE  |ccu|
----------------
```

Usage:

IPLSAVE is used to perform an Initial Program Load sequence without clearing core. For example, it would be used to bring a program such as a stand-alone dump into high core without disturbing the current contents of core.

Note:

For examples on IPLSAVE, see IPL.

LINK

Purpose:

LINK attaches the specified device to the requestor's virtual machine, based on information contained in the CP-67 User Directory.

Format:

```
-------------------------------------------
| LINK | userid xxx yyy <W,R>  <(NOPASS)>   |
|      |   *                  PASS= password|
-------------------------------------------
```

userid The name of the user "owning" the device xxx. An * denotes that you are linking to yourself.

xxx The "owner's" virtual device address in hexadecimal

yyy The hexadecimal address to be assigned in the requestor's virtual machine

W Write access mode is requested.

R Read access mode only is requested. This is the default value.

(NOPASS) Used only if no special password is required for the desired access mode (that is, a "public" device).

PASS= password this form can only be use when LINK is executed by a virtual console function (that is, from CMS). The password is not print suppressed.

Usage:

LINK allows attachment of directory-defined virtual disks to the requestor's virtual machine. The device must be described in the CP-67 User Directory under the name and device address (userid,xxx) specified.

If the userid is that of the requestor, no password is required, and rules governing access are the same as prevail at LOGIN time. If you are linking to yourself, the default access is read-write; if you are linking to another user, the default access is read-only.

If the LINK is to some other userid, a password for the desired access must be provided (see Responses).

In general, any number of users can link simultaneously to a device in read-only mode. Only one user can have access to a device if the first link has write access. If a read-only

link exists, and a write request is issued, the link is made in read-only mode.

Note:

If a user links to another disk as ccu and he already has a disk ccu, he must either reIPL CMS to read in the new file directory, or issue LOGIN ccu. Otherwise, the old file directory exists in core and the new disk is clobbered.

Responses:

ENTER PASSWORD:
Type the required password. By convention, devices specified as public have the password ALL. A null line defaults to ALL.

BAD ARGUMENT
Missing or invalid arguments. Error code is 08.

SET TO READ ONLY
The user has requested and been granted read-only access. Error code is 00.

SET TO WRITE
The user has requested and been granted read-write access. Error code is 00.

FORCED READ ONLY
The user has requested write access but because read-only links exist, read-only access is forced. Error code is 36.

NONEXISTENT UNIT
Device xxx not found in the specified directory. Error code is 12.

ALREADY ATTACHED
Requestor already has a device yyy. Error code is 16.

BAD PASSWORD
The supplied password is not valid, or the device is not sharable. Error code is 20.

USER LOGGING IN
userid is in process of logging in. Try again later. Error code is 24.

DEVICE IN USE
A write link exists. LINK denied. Error code is 28.

UNIT NOT READY
The required physical volume is not mounted. Notify system operator. Error code is 32.

UNIT NOT DASDI
Device xxx must be a DASDI device for LINK. Error code is

44.

USER NOT ON SYSTEM
userid is not in the directory. Error code is 40.

UPDATING DIRECT
The system DIRECTORY is being modified; attempt a LINK again when the DIRECTORY is completed. Error code is 48.

Example:

CP LINK USERA 195 196 R PASS= READPASS
The user links to USERA's 195 disk as his own 196 in a read-only mode. The password is given on the same line, so it is not print suppressed.

LOGOUT

Purpose:

LOGOUT removes the user from the system by releasing the virtual machine and any attached devices, clearing the temporary disk area, and closing the spooling areas.

Format:

anything If any nonblank character string is specified with LOGOUT, the telephone line is not hung up and the terminal remains connected to CP for another user to login or dial in.

Usage:

LOGOUT logs the user off the system. The temporary disk area is cleared and all spooling areas are closed. If there is output in the spooling areas, it is printed or punched on a real device when that device becomes available, as in CLOSE.

Because of the finality of this command, the only abbreviation accepted is LOG.

If "anything" is specified, a LOGOUT occurs in the normal manner but the communication line is not disabled. Upon completion of the LOGOUT procedure, the "CP-67 online" message indicates that another login or dial can proceed.

Responses:

LOGOUT AT xx.xx.xx ON xx/xx/xx
The user is removed from the system. The xx.xx.xx is the time of day and the xx/xx/xx is the date.

CP-67 ONLINE xxxxxxxxxxxx
xxxxxxxxxxxx CP-67 ONLINE
CP-67 ONLINE
If one of the above messages is typed out after the LOGOUT message, a nonblank character string was specified with LOGOUT; therefore the terminal remains connected to CP for another user.

Purpose:

MSG sends a specified message to a specific user.

Format:

userid specifies to whom the message should be sent.
CP specifies that the message is sent to the systems operator whatever his userid might be. For this reason no user should have the id of CP; CPxxxxxx is acceptable though.

line is the message to be sent to the specified user.

Usage:

MSG allows the users to communicate with the CP operator in the computer room as well as with other users. The specified user must be logged on the system before a message can be sent to him. If a specified user is not logged on, the message USER NOT ON SYSTEM is typed out but is not held until he logs on.

A message that is sent by MSG is in this format

FROM userid: line

The message is typed out at the specified user's terminal when the terminal is not ready for input. If the terminal is waiting for input, the message is held until a carriage return occurs.

Responses:

userid NOT RECEIVING
The user has suppressed his receiving of messages (see Console Function SET).

USER NOT ON SYSTEM
The specified user was not logged on the system so the message was not sent to him. The message is not saved.

Example:

msg CP please attach a tape drive to 181
The message "please attach a tape drive to 181" is typed out at the system operator's terminal.

PURGE

Purpose:

PURGE allows the user to delete all spooled files still in the spooling area from his virtual printer or punch, or to delete all his spooled input for his virtual card readers without reading the data.

Format:

PURGE P	READER R PRINTER P PUNCH PU

Usage:

If the user determines that he does not require either his spooled print output or his punch output, or if he wishes to purge all his input reader files, the PURGE command can be issued.

Responses:

BAD ARGUMENT xx
An invalid device type was specified.

xx FILES PURGED
This is the normal response, where xx is the number of files actually purged or the word NO, if there were none to purge.

Examples:

a. purge pun
NO FILES PURGED
There were no punch files awaiting spooled output for the user.

b. purge r
02 FILE PURGED
The two spooled input files for the user have been deleted.

QUERY

Purpose:

QUERY types out the number of users logged on or dialing, the identification and terminal address of the users logged on, the terminal address of a specific logged on user (or a message that he is not logged on), the log messages set by the CP operator, the number of spooled input and output files currently held by CP for the user, or the amount of connect, virtual CPU, and total CPU time used since login.

Format:

QUERY Q	FILES LOGMSG NAMES USER userid TIME

FILE (F) types the number of spooled input and output files currently held by CP for the user.

LOGMSG (L) types out the message of the day set by the operator.

NAMES (N) types out the userid and terminal address of all users on CP.

USER (U) types out the number of virtual machine users, and the number of users attached to virtual machines using the CP-67 DIAL facility.

userid types out the userid and the terminal address where the user is logged in, or gives USER NOT ON SYSTEM.

TIME types out the CONNECT,VIRTCPU, and TOTCPU time used so far in this terminal session for this user.

Usage:

QUERY is used to determine the number of users logged on or dialing, who they are, and what their terminal address is, what the log message is, the number of current spooled files for this user, and the amount of time (connect and CPU) that has elapsed for this user since login. This information gives an idea of the system load, as well as any pertinent information about the system. The log message is normally the message that types out once the user has logged on under CP.

Responses:

BAD ARGUMENT xx
An invalid argument was specified.

nnUSERS; mm DIALED
If the USER argument was specified, this message indicates that nn virtual machines are logged in and that mm users have dialed other virtual machines.

userid - xxx, userid - xxx . . .
If the NAMES argument was specified, current users are displayed four to a line.

userid - xxx
This is the response to "Q userid", if that user is logged on.

USER NOT ON SYSTEM
This is the response to "Q userid", if that user is not logged on to CP-67.

FILES: xx RDR, xx PRT, xx PUN
The number of spooled files for this user is printed.

xxxxx ... xxxxx
This is the log message obtained as a result of specifying the LOGMSG argument.

Examples:

a. query user
04 USERS; 06 DIALED
Four users are logged on the system. Six users have DIALED other virtual machines.

b. q names
OPERATOR - 009, APL - 023, CJOHNSON - 024
A list of current users logged on the system is typed out on the terminal.

c. q logmsg
TS DOWN AT 12 SHARP
The log message TS DOWN AT 12 SHARP, set by the operator, is typed out.

d. q files
FILES: 01 RDR, 06 PRT, NO PUN
The number of separate spooled files for the user is shown. It is a total for all virtual card reader, printers or punches.

e. q temp21
TEMP21 - 032

q temp4
USER NOT ON SYSTEM

READY

Purpose:

READY simulates a device end for the specified virtual address.

Format:

READY	devadd

devadd specifies a virtual device address, such as 191.

Usage:

On a real machine a device end is caused by the completion of an I/O operation at a device or, on some devices, by manually changing the device from the not-ready to ready state. A device end normally indicates that the I/O device has become available for another operation. READY simulates this device end without having to complete an I/O operation or without making a device not-ready.

Responses:

BAD ARGUMENT xx
An invalid device address was specified.

NONEXISTENT UNIT
The specified unit does not exist in the virtual machine configuration.

Example:

READY 191
A device end is simulated for the device whose address is 191.

RESET

Purpose:

RESET simulates the system reset key on the system control panel.

Format:

```
-----------------
|     RESET     |
|     RE        |
-----------------
```

Usage:

RESET places the virtual machine in a stopped state and resets all pending I/O interrupts. All error conditions are reset. The system is automatically reset by IPL.

Responses:

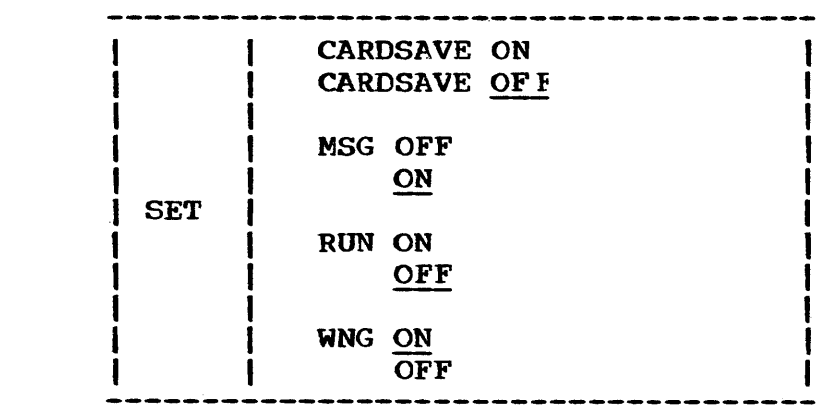
None.

SET

Purpose:

SET allows the user to save the card file in his virtual card reader, to control the messages and warnings typed at his terminal and to control his running status.

Format:



CARDSAVE ON saves the card file in the virtual card reader so that it can be reread.
CARDSAVE OFF erases the card file in the virtual card reader once the reader has been closed.

MSG OFF specifies that no messages are to be typed at the terminal.
MSG ON specifies that all message are to be typed at the terminal.

RUN ON allows the user to activate the ATTN button causing a read of a CP console function without stopping his virtual machine. When the CP function is typed in, it is immediately executed and the virtual machine resumes execution.
RUN OFF places the user in the normal CP environment such that when ATTN is hit, the virtual machine stops.

WNG ON specifies that all warnings are to be typed at the terminal.
WNG OFF specifies that no warnings or messages are to be typed at the terminal.

Usage:

SET CARDSAVE ON does not erase the cards in the virtual card reader once the reader has been closed. Therefore, the SET CARDSAVE ON allows the same virtual card input to be read repeatedly from the beginning of the file. The operation is effective for all the user's virtual card readers.

SET CARDSAVE OFF is the normal mode of operation for the card reader. Once a virtual input file has been closed, it is lost. To reread the file, it must be physically read into CP-67 again by the operator or via XFER.

The normal mode of operation for messages and warnings typed at the terminal is MSG ON and WNG ON, respectively. Any messages directed to a user or broadcast by the operator are typed at the terminal whenever the terminal is not ready for input. A warning sent by the CP operator prints immediately, regardless of what is occurring at the terminal. If MSG OFF is specified, only warnings from the operator type at the terminal, and any messages sent by another user or the operator are lost.

If either a MSG OFF or WNG OFF has been specified, and the user wishes to resume receiving either messages or warnings, the MSG ON and/or WNG ON command must be issued.

If WNG OFF is specified, no warnings or messages will ever type on the terminal. If WNG OFF and MSG ON are both set, only messages will type on the terminal. SET WNG OFF should be used with discretion.

The normal mode of operation for running is RUN OFF. When ATTN is hit, the virtual machine stops running, and the terminal waits for a CP console function. In a multiaccess virtual machine, this is an unacceptable method of operation; therefore SET RUN ON can be issued to CP to allow the virtual machine to continue running when ATTN is hit. When RUN OFF is issued after RUN ON had been previously set, the virtual machine continues to run until ATTN is hit; this causes the machine to stop for input. The SET RUN OFF mode is automatically set if the user gets an "idle" virtual machine, that is, with the message

CP ENTERED, REQUEST, PLEASE.

Responses:

BAD ARGUMENT xx
An invalid argument was specified.

Examples:

a. SET CARDSAVE ON
The virtual input file is saved and not erased, once the reader is closed.

b. SET MSG OFF
No messages will type at the terminal, only warnings from the operator.

c. SET RUN ON
The virtual machine immediately continues execution. If the

user now activates the ATTN key, a CP read occurs, but his virtual machine continues to run.

SLEEP

Purpose:

SLEEP allows the user to place his terminal in a dormant CP mode such that he may receive messages without hitting carriage return.

Format:

```
---------------
|    SLEEP    |
---------------
```

Usage:

If the user does not expect to be using the terminal for a while, the SLEEP console function places the terminal in a state to receive messages. The user's virtual machine is not run, but he is still accounted for connect time. The terminal can be "awakened" by activating the ATTN key which returns the user to CP for more input.

Responses:

None.

Examples:

sleep
The terminal is placed in a dormant state to receive messages. "Warnings" set by the operator are not affected. If the user has done a SET MSGOFF he does not, of course, receive messages, only warnings, if they are issued.

SPOOL

Purpose:

The SPOOL console function allows the user to direct his spooled output to a specific unit record device at the computing facility and to control the nature of reading spooled input files.

Format:

SPOOL SP	xxx	<ON yyy> <OFF>
	ccc	<CONT> <OFF>

xxx is the virtual device address from which output is to be directed to the specific real device, yyy.

yyy is the real device address of the desired output unit. It can only be a printer or punch address.

ccc is the virtual device address of the card reader that is to have continuous spooled input.

Usage:

When printing or punching files from virtual devices, the output is written to disk or "spooled", until the physical device is available for use. If there are multiple printers or punches, the first available device is used for the output. To control the spooled output direction, SPOOL can be issued specifying the virtual device address and the real device to which the output should go. The virtual and real device types must be the same; in other words the virtual punch cannot be directed to the real printer.

To discontinue the directed spooling and return to normal spooling, issue SPOOL, specifying the virtual device address and OFF.

SPOOL also controls the virtual card reading characteristics. When multiple physical card decks have been spooled onto disk by CP, a user must close each file before the next file can be read. For continuous virtual card reader input such that the card reader does not have to be closed between each file, the SPOOL command can be issued specifying the virtual card reader address and CONT. The virtual machine receives an end-of-file indication only after the last card of the last spooled input file has been read.

To terminate the continuous reading of files, issue SPOOL,

specifying the virtual card reader address and OFF.

Notes:

a. Continuous reading of input files should not be in effect with SET CARDSAVE ON, as an unending "wrap-around" input file will exist.

b. Directed output is useful--and necessary if, for instance, an installation has two printers, one with a PN train and another with a TN train. If the user has script output to produce, he may specify the desired output printer, perform his printing, and then return his printer to normal spooling.

Reponses:

BAD ARGUMENT xx
This indicates that an error in specification has been made, such as invalid virtual address, invalid real address, or conflicting device types.

Examples:

a.
```
spool e on 30
begin
CMS
script report (offline center)
R;T=05.21/06.03 12.22.13
     <-----ATTN key hit
cp   <-----confirmation of environment
sp e off
```
The SPOOL console function is issued to CP to direct the virtual printer 00E output to the real printer 030. BEGIN returns control to CMS, where REPORT SCRIPT is formatted offline. ATTN is hit to go to CP, and the directed output of printer 00E is terminated. Normal spooling now occurs on 00E.

b.
```
spool c cont
begin
CMS
offline read * *
OFFLINE READ A FORTRAN
OFFLINE READ B FORTRAN
OFFLINE READ CALC FORTRAN
R;T=02.01/02.78 12.34.56
     <-----ATTN key hit
cp
query files
FILE: - NO RDR, 00 PRT, NO PCH
spool c off
```
The SPOOL command is issued to read multiple spooled input files as if they were continuous. Control is transferred to CMS by BEGIN, and the spooled input files are read. ATTN is hit to return to CP and the spooled files are queried.

Input spooling is then returned to normal for noncontinuous reading.

STORE

Purpose:

STORE replaces the contents of specified locations in core storage, general-purpose registers, floating-point registers, and the program status word, and for virtual 360/67, in control registers 0,2,4,and 6.

Format:

STORE ST	<Lhexloc hexinfo1...hexinfoN> <Greg hexinfo1...hexinfoN> <Yreg hexinfo1...hexinfoN> <Xreg hexinfo1...hexinfoN> <PSW<hexinfo1> hexinfo2>

Lhexloc hexinfo1...hexinfoN stores the hexadecimal values hexinfo1... hexinfoN in successive fullword locations starting at hexadecimal location hexloc.

Greg hexinfo1...hexinfN stores the hexadecimal values hexinfo1... hexinfoN in successive general-purpose registers, starting at the register specified by reg. The parameter reg must be an integer ranging from 0-15, and successive values of reg cannot exceed 15.

Yreg hexinfo1...hexinfoN stores the hexadecimal values hexinfo1... hexinfoN in sucessive floating-point registers, starting at the register specified by reg. The parameter reg must be an integer between 0 and 6, and successive values of reg cannot exceed 6. If reg is an odd integer, it is adjusted to the preceding even integer.

Xreg hexinfo1...hexinfoN stores the hexadecimal values hexinfo1... hexinfoN in successive control registers starting at the register specified by reg. The parameter reg must be an integer ranging from 0-15, and successive values of reg cannot exceed 15. Only control

registers 0-2-4-6 can be modified by STORE; all the other control registers are supposed to have a zero value. Applies to virtual 360/67 only.

PSW <hexinfo1> hexinfo2 stores the hexadecimal values hexinfo1 and hexinfo2 in the first and second words of the program status word. If only hexinfo2 is specified, it is stored in the second word of the PSW. The interrupt code is set to zero. The hexinfo1 and hexinfo2 must be separated by one or more blanks.

Usage:

The smallest group of hexadecimal values that can be stored is one full word and alignment is made to the nearest fullword boundary. If the hexadecimal value being stored is less than a fullword (eight hexadecimal characters), it is right-adjusted in that word and filled in with high order zeros.

The options can be combined in any order wanted, separated by one or more blanks, up to one full line in length, and issued in a single STORE console function. L, G, or Y must be specified or the options are invalid.

If invalid arguments are specified, a message is typed out and the console function terminates. If there are any valid arguments before the invalid one, they are executed properly.

Responses:

BAD ARGUMENT xx
An invalid argument was specified. If there were any valid arguments before the invalid one, they were executed properly. The console function terminated on the invalid argument.

ILLEGAL CREG 0
An attempt to load control register 0, with bits 26-31 nonzero; the register is not loaded.

ILLEGAL PSW
An attempt to load the PSW with bits 0-4 nonzero, when the virtual machine is a System 360/Model 67 running in extended PSW mode (bit 8 of control register 6 set to 1). This message is just a warning; the PSW is modified and a program interrupt occurs with "specification exception" on the execution of the next instruction.

Examples:

a.
```
d 12011
L  12010 = 4110D01C

st 112011 579

d 12011
L  12010 = 00000579
```
A full word at core storage location 12011 is displayed after alignment is made to the nearest fullword boundary 12010. The hexadecimal number 579 is right-justified, filled in with zeros, and then stored in a full word, beginning at location 12010. That word is displayed again to reflect the changed value.

b.
```
d g5-8
G  5=00000400   0000000C   0000049C   00000004

st g5 123 456 aa

d g5-8
G  5=00000123   00000456   000000AA   00000004
```
The contents of general-purpose registers 5-8 are displayed. The hexadecimal numbers 123, 456, and AA are right-justified in separate words, filled in with zeros and then stored in general-purpose registers 5, 6, and 7. The contents of registers 5-8 are then displayed to show their stored contents.

c.
```
d psw
PSW        = 00000000   80001374

st psw 55555

d psw
PSW        = 00000000   00055555
```
The contents of the PSW are displayed. The hexadecimal number 55555 is right-justified, filled with leading zeros, then stored in the second word of the PSW, and the interrupt code is set to zero. The PSW is displayed to reflect the change.

d.
```
d psw
PSW        = 00000000   80001374

st psw 111111 12000

d psw
PSW        = 00111111   00012000
```
The contents of the PSW are displayed. The hexadecimal numbers 111111 and 12000 are stored right-justified in the first and second words of the PSW, and the interrupt code in the first word is set to zero. The PSW is then displayed to reflect the change.

e.
```
d x0
X       0 = 00016C80

st x0 123456
ILLEGAL CREG 0

d x0
X       0 = 00016C80
```
The contents of control register 0 are displayed; an attempt is made to store a wrong value in it; the contents of the register are not modified.

f.
```
d psw x6
PSW       = 03060000 00000000
X       6 = F08000FF

st psw 12345678 12345678
ILLEGAL PSW

d psw
PSW       = 12345678 12345678
```
The contents of the PSW and control register 6 are displayed, the hexadecimal numbers 12345678 12345678 are stored in the PSW, and since the virtual machine is in extended mode, the warning ILLEGAL PSW is issued.

g.
```
st psw 3060000 0
st x6 f00000ff

d psw x6
PSW      = 03060000 00000000
X      6 = F00000FF

st psw 12345678 12345678

d psw
PSW      = 12340000 12345678
```
Now the virtual machine is not in extended mode, therefore, the PSW is modified without warning.

h.
```
d x0
BAD ARGUMENT 01
```
The virtual machine is not a System 360/Model 67.

Purpose:

XFER controls the passing of files between users.

Format:

devadd specifies the address of the device from which all succeeding output is to be either transferred or normally written out. A virtual punch or printer output may be XFER'ed.

To userid turns on the transfer mode. Userid is the eight-character user identification of the user that is to receive the transferred file.

TO * specifies that the recipient of the transfer is the user issuing the command.

OFF terminates the transferring of all further output to userid.

Usage:

XFER controls the passing of files between users. When the transfer mode is turned on by specifying TO userid, any succeeding output written to devadd is placed in the card reader of userid. For userid to receive the transferred files on his disk, userid must read them onto his files, that is, by issuing commands such as OFFLINE READ filename filetype, OFFLINE READ *, or DISK LOAD.

To turn the transfer mode off, the option OFF must be specified. Any succeeding output to the devadd is written onto disk as in normal spooling, and put on the real device when that device is free.

A second XFER command issued to change userid automatically turns off the XFER to the previous userid, even if the userid is invalid.

If the user XFER'ed to is currently logged on to CP-67, he receives the following message:

** CARDS XFERED BY userid **

where userid is the transferring user.

The user XFER'ing the file receives the message:

** CARDS XFER'D TO userid **

where userid is the receiving user.

Notes:

A user may do an XFER to himself to allow him to read a card file he created, (using OFFLINE PUNCH or DISK DUMP) without requiring operator intervention and with a considerable saving in card handling.

Responses:

BAD ARGUMENT xx
An invalid argument was specified.

NONEXISTENT UNIT
The specified device does not exist and is invalid.

DEVICE BUSY
Spooled output is in operation for the punch. A close should be issued to clear the status.

UPDATING DIRECT
The system DIRECTORY is being modified; attempt the XFER after the DIRECTORY is created. The XFER command has not been accepted.

Examples:

a. xfer d to user1
The transfer mode is turned on. Any succeeding output to device 00D is placed in the card reader of USER1.

b. xfer d off
The transfer mode is turned off. Any succeeding output to virtual 00D is written on the real device.

c. xfer d to userA4
USERA4 NOT IN DIRECTORY
USERA4 does not exist; therefore cards are not transferred to him. Any active transfer mode is turned off.

d. CP <-----environment confirmation
x d to user20
begin
CMS
disk dump document script
R;T=02.19
** CARDS XFER'D BY USER4 **
offline read prog2 datain
Output device 00D is transferred to USER20. Control is returned to CMS by issuing BEGIN to CP, and the file DOCUMENT SCRIPT is transferred to the card reader of USER20 in disk dump format. Meanwhile, cards were XFER'ed to this user by USER4, and he reads them onto his disk.

CONSOLE FUNCTION APPLICATIONS

Console functions give the facilities of a computer console to each virtual machine, and they should be used accordingly. It is through these functions that a virtual machine is loaded, displayed, dumped, altered, and controlled by the user.

Console functions have various uses. Some of the application areas are described below, such as debugging, initializing CMS, and attaching/detaching additional I/O devices.

Debugging. Console functions are very useful for debugging purposes. The CP environment can be entered at any time and the contents of core storage, the registers, and the PSW displayed, dumped, or altered. Execution can be started again by issuing BEGIN, with or without a specified hexadecimal location.

If the CP environment has been entered from CMS and the user desires to enter the Debug environment of CMS, he can issue EXTERNAL. The EXTERNAL console function generates an external interrupt and causes DEBUG to be entered immediately.

Initializing CMS. If at any time the user destroys his copy of CMS, a new copy can be loaded again by issuing IPL CMS or IPL 190. All pending interrupts are reset and CMS is started anew.

If I/O errors occur on the disk when logging in to CMS, the file directory from the permanent disk has probably been read into core incorrectly. DO NOT LOG OUT. Enter CP by hitting ATTN once and issue IPL CMS or IPL 190 to initialize CMS.

Attaching/Detaching Additional I/O Devices. If an I/O device such as a tape drive, is needed that does not belong to the virtual machine configuration, the user must communicate with the CP operator that he wants a device attached with a specified address, such as 181. As soon as the device is attached by the operator, the message DEV devadd READY is typed out at the terminal. The specified device address can now be used.

The user can continue using the terminal while he is waiting for a device to be attached, as long as he does not address that device. Once the device is attached and the appropriate message is typed out, the specified device is dedicated to the one user.

There is no tape-label checking performed by CP. It is up to the operator to mount the correct tape.

The responsibility is left to the user to detach or remove

the specified device from his configuration. As long as the device is attached to him, it is unavailable for use by any other user. By issuing DETACH devadd, the device is removed from the virtual machine configuration and the message

DEV devadd DETACHED

is typed out at the terminal. A message is automatically typed out to the operator specifying the device that is free. If the detached device address is that of a tape drive, the tape is rewound and unloaded.

Transferring Files. Any files up to 132 characters in length, fixed or variable, can be transferred between users by issuing the XFER console function. Output written to the XFER'ed device is placed in the card reader of the specified user. CMS commands, such as OFFLINE PUNCH for card-image records, OFFLINE PRINT for printer files, and DISK DUMP for variable-length records, can be used to write output to the XFER'ed device.

Controlling Spooled Input/Output. Spooled input and output remain on the disk until the user reads or closes the input file, or until the physical device is available for output. To remove all spooling areas for a particular device, issue the PURGE console function. To determine how many spooled files currently exist, issue--QUERY FILE. To direct spooled output to a particular device rather than the first available one, issue SPOOL. To concatenate spooled input files so that many physical decks can be read continuously as if they were one, issue the SPOOL console function, specifying CONT for the card reader.

CP-67 MESSAGES

The following CP-67 messages are directed to a user in response to a machine malfunction or operator intervention.

When the operator attaches a device to a user, the message:

```
DEV XXX ATTACHED
```

is typed at the user's terminal.

If the user has placed the terminal in sleep state the message is typed immediately, otherwise it is typed following the next carriage return.

Should the operator detach a device, the message

```
DEV XXX DETACHED
```

is similarly typed.

Should a machine check occur and the user's machine is enabled for machine checks, the message

```
**MACHINE CHECK** - CP ENTERED, REQUEST PLEASE
```

is printed at the user's terminal and the user's machine is placed in console function mode. One must reIPL should this occur.

If the user's machine is not enabled for interrupts, the message:

```
** MACHINE CHECK **
```

is printed at the user's terminal and the user's machine remains unaltered.

If an error is encountered on a system disk, such as a T-disk, the message:

```
**SYSTEM IOERROR**  CP ENTERED, REQUEST PLEASE
```

is typed at the user's terminal and the user's machine is placed in console function mode.

The message

```
CONNECT=xx.xx.xx  VIRTCPU=xxx.xx.xx  TOTCPU=xxx.xx.xx
```

is typed at the user's terminal if the operator does an ACNT CP console function. No change occurs in the user's machine status except that connect, virtual, and total times are reset to zero.

OPERATING CONSIDERATIONS

OFFLINE PROCEDURES

To read cards into a user's file, the OFFLINE READ command is used. The first card in a deck to be read must be the control card

CP67USERID userid

where the user's ID is punched beginning in column 13 (that is, two spaces between CP67USERID and the user's ID).

Cards are read by the OFFLINE READ command in CMS and a file is formed which is referred to by its filename and filetype. The control card

OFFLINE READ name type mode

specifying the filename and filetype, should be placed before each set of cards which is to form a file where "name" is the filename, "type" is the filetype, and "mode" is the filemode. If the "mode" is not specified, each file is entered into the user's file directory with mode P1.

When a user wishes to have a deck of cards read in offline, he should give the deck to the CP operator and request that his deck be read into CP. If the user was logged in before the deck was read by CP, the following message is typed at the console:

CARDS HAVE BEEN READ

If the deck was read by CP before the user logged in, the following message is typed at the console, as soon as the user logs on:

FILES: - xx RDR, xx PRT, xx PCH

When the deck has been read by CP, the user should issue the CMS command

OFFLINE READ *

to cause CMS to read the cards and form the desired files. If the deck is read by CP, but the CMS command OFFLINE READ is not issued, the files are not written onto the user's disk.

TAPE PROCEDURES

A facility is available for attaching magnetic tapes to a virtual machine. The selection of a physical tape drive is made by the CP operator, and the virtual device address is chosen by the terminal user. The CMS TAPE command requires that the tape have address 181. The TAPEIO function and the TPCOPY command associate the device names TAP1 and TAP2 with the device addresses 180 and 181, respectively.

To have a tape attached to your virtual machine, send a message to the CP operator asking him to mount a tape for you as a specified device address. When the tape is mounted and ready for use, the system automatically types the message

DEVICE xxx ATTACHED

Since there are only a limited number of tape drives available for use by CMS users, a user should not keep a tape attached to his machine any longer than he is using it. To release a tape from a virtual machine, enter the CP environment and type the CP console function

DETACH devadd

When a user logs out of CP, all attached devices are automatically detached.

LIBRARY USAGE

CMS has two types of libraries--macro libraries and text libraries.

Macro Libraries.

A macro library is created by the MACLIB command; it is a file that has a filetype of MACLIB and that contains macro definitions and a dictionary. A macro definition is a group of assembler-language statements identified by a unique name and used as an expansion of a source statement in an assembler-language program. The dictionary generated by the MACLIB command is made up of macro definition names, indices or locations of the macro definitions within the library, and the size in number of card images in each macro definition.

Both the user and the system may have macro libraries. CMS provides two macro libraries that reside on the systems disk; SYSLIB MACLIB and OSMACRO MACLIB. SYSLIB MACLIB contains the CMS macros, and OSMACRO MACLIB contains the OS macros supported under CMS. The user can generate his own libraries on his permanent disk by issuing the MACLIB command.

Macro libraries are searched during the ASSEMBLE command for missing macro definitions. Normally the only macro libraries searched are SYSLIB MACLIB and OSMACRO MACLIB, in that order. Once macro libraries have been generated by the user, there are two methods by which the user's MACLIB files can be searched for missing macro definitions.

The first method is by issuing the GLOBAL ASSEMBLER command. This command names from one to five MACLIB files that are to be searched for missing macro definitions. If the files SYSLIB MACLIB and OSMACRO MACLIB are to be searched in addition to the user's MACLIB files, SYSLIB and OSMACRO must be specified as two of the five libnames in the GLOBAL ASSEMBLER command. The MACLIB files are searched in the order in which they are named. The GLOBAL macro libraries remain in effect until another GLOBAL ASSEMBLER command is issued, or the CMS nucleus is reinitialized, or the user logs out of CP.

The second method is by generating a SYSLIB MACLIB file or an OSMACRO MACLIB file on a disk that precedes the systems disk in the standard order of search. The ASSEMBLE command normally searches for the file SYSLIB MACLIB and OSMACRO MACLIB for missing macro definitions.

To terminate the searching of all MACLIB files, including SYSLIB MACLIB and OSMACRO MACLIB, the GLOBAL ASSEMBLER command can be issued with no libnames specified.

Text Libraries

A text library, created by the TXTLIB command, is a file that has a filetype of TXTLIB and that contains a dictionary and the relocatable object code from TEXT files. The dictionary is created by the TXTLIB command and contains control section names, the entry points, their location, and the size of each TEXT file included in the text library.

Both the user and the system may have text libraries. CMS provides four text libraries which reside on the systems disk; they are SYSLIB TXTLIB, CMSLIB TXTLIB, PLILIB TXTLIB, and SSPLIB TXTLIB. SYSLIB TXTLIB contains the standard IBM OS/360 FORTRAN G library routines; CMSLIB TXTLIB contains the nonerror message FORTRAN subroutines; PLILIB TXTLIB contains the standard PL/I library; and SSPLIB TXTLIB contains the Scientific Subroutine Package. The user can generate his own text libraries by issuing the TXTLIB command.

Text libraries are searched by the LOAD, USE, and REUSE commands to find missing subroutines, undefined names, and TEXT files not found. Normally the only text library searched is SYSLIB TXTLIB. To search either CMSLIB TXTLIB, PLILIB TXTLIB, or SSPLIB TXTLIB, the GLOBAL command should be issued. If text libraries have been generated by the user, there are three methods by which the user's text libraries can be searched during the LOAD, USE, and REUSE commands.

The first method is by issuing the GLOBAL LOADER command. This command names from one to eight TXTLIB files that are to be searched for missing subroutines. If the file SYSLIB TXTLIB is to be searched in addition to the user's TXTLIB files, SYSLIB must be specified as one of the eight libnames in the GLOBAL LOADER command. The TXTLIB files are searched in the order in which they are named. If the GLOBAL LOADER command has been issued and the user wishes to terminate the use of his TXTLIB files, the GLOBAL LOADER command can be issued specifying SYSLIB. To terminate the searching of all TXTLIB files, issue GLOBAL without specifying any TXTLIB files. The GLOBAL text libraries remain in effect until another GLOBAL LOADER command is issued, the LOAD command is issued with the LIBE or SLIBE option, the CMS nucleus is reinitialized, or the user logs out of CP.

The second method is by issuing the LOAD command with the LIBE option and specifying the text libraries to be used. If the file SYSLIB TXTLIB is to be searched along with the user's text libraries, SYSLIB must be specified as one of the libnames. The LIBE option with the LOAD command overrides the effect of the GLOBAL LOADER command for that LOAD and any following USE or REUSE commands.

The third method is by generating a SYSLIB TXTLIB file on a disk that precedes the systems disk in the standard order of

search. LOAD, USE, and REUSE normally search for the file SYSLIB TXTLIB for missing subroutines or TEXT files not found. If a user file has the identifier SYSLIB TXTLIB it is used in place of the system text library.

RECOVERY PROCEDURES

The recovery procedures discussed here deal with error recovery as well as general recovery from user problems. The recovery procedures are as follows: errors during LOGIN, errors specified by the E(xxxxx) message, recovering from the system going down, reinitializing CMS, file space full, and general recovery procedures.

Errors during CMS Login

CMS LOGIN is defined as reading into core the user-file-directory from the specified disk. Login is initiated after the CMS startup message by: 1) entering a null line, 2) issuing the CMS command LOGIN, 3) invoking any valid CMS command or function. Errors that might occur during the login procedure are: a) P-disk not attached--contact CP systems operator; b) file-directory unreadable--total catastrophe, issue FORMAT P ALL; c) invalid device type--only 2311 or 2314 devices are allowed; d) hardware I/O error attempting to read the file-directory--real hardware problems.

If I/O errors occur during the CMS LOGIN NO_UFD command, issue FORMAT P to reinitialize the permanent disk. If errors occur during FORMAT P, issue FORMAT P ALL.

Errors Specified by the E(xxxxx) Message

Any error conditions that occur during a CMS command are typed out with an E(xxxxx) message when the command is terminated. The xxxxx is the error code number and it is explained in that command's description. If files were permanently written out or updated before the error condition occurred, the most current files are reflected in the user-file-directory, when the time (T=xx.xx) is typed out, unless otherwise stated by that command's description.

Recovering from the System's Going Down

If the system goes down while using CMS, the files on the temporary disk are lost, and the files on the permanent disk are as current as they were when the last Ready message (R;T=xx.xx) or error message (E(xxxxx);T=xx.xx) was typed out, with one exception. If an EXEC command had been issued, the files used by the CMS commands that had finished execution before the system went down are reflected in the current user-file-directory, even though no time (T=xx.xx) was specified between the commands. Except in the case of EXEC, the user-file-directory is always updated on disk whenever the time (T=xx.xx) is typed at the terminal.

If a file is being edited or created by EDIT or CEDIT when the system goes down, it may not be completely lost. Issue a LISTF, and see if the EDIT or CEDIT work files (INPUT1) FILE or (INPUT2) FILE exist. If they both exist, take the

longer of the two if you are creating or adding to the file, or take the shorter file if you are deleting many lines, and then proceed as below; if they both exist and are the same length, issue a PRINTF or OFFLINE PRINT for both work files to see which work file has the latest copy of the file being edited, and then proceed as below; if only one work file exists then proceed as below. Note that EDIT only uses one work file--(INPUT1) FILE.

ALTER the filename and/or filetype of the appropriate work file. Then issue EDIT for the ALTER'ed file and begin editing, as this file contains the latest copy of the file that was being edited when the system went down.

If no work file exists, all input and changes made since the last FILE or SAVE request have been lost. To prevent the updated or new file from being lost, issue the FILE or SAVE request frequently.

Reinitializing CMS

If DEBUG is entered, CMS can be reinitialized as follows: (1) the CMS IPL command can be issued from the DEBUG environment or the CMS command environment, (2) ATTN can be hit and IPL CMS or IPL 190 issued to CP, (3) RESTART can be issued from the DEBUG environment to reIPL the CMS nucleus, or (4) KX issued in DEBUG, which updates the user directory and IPL's the CMS nucleus. Issuing KX to DEBUG is the only way among the previous three methods of reinitializing CMS to permanently update files that have been modified or created since the last T=xx.xx message.

If CMS does not work as it should, it could be that the copy of CMS in the virtual machine has been destroyed by the user (no user can get to or alter another user's virtual machine or core storage). The user should either (1) enter the Control Program by hitting ATTN and issue IPL CMS to reload CMS, or (2) issue IPL to the CMS command environment.

The command IPL, LOGIN, and FORMAT can be issued at any time in the CMS command enviroment and not just at the beginning.

File Space Full

If 99% of the user's filespace is full, an error message is typed out and the user is logged out of CMS. Issue IPL CMS to CP and, if at all possible, erase some files. If there is still not sufficient space available on the disk, dump all or part of the files onto tape with the CMS commands TAPE DUMP, and then erase those files on disk. Whenever the files on tape are needed, the commands TAPE LOAD (if TAPE DUMP created the tape) could read them back onto disk. If the user still needs more filespace, he should contact the system administrator for more disk space.

General Recovery Procedures

If it is not clear which environment has been entered, hit carriage return and the response confirms which environment has been entered.

To kill execution of a command or program in CMS, ATTN should be hit twice and KX entered. To kill a type out in CMS, ATTN should be hit twice and KT entered. To kill overrides in CMS, ATTN should be hit twice and KO entered.

If all of the files on the permanent disk are to be erased, the command LOGIN NO_UFD should be issued instead of ERASE * *.

If errors persist in an executing program, use the DEBUG command or, if the program is written in FORTRAN, use the FORTRAN G Debug package (see IBM System/360: FORTRAN IV Language, C28-6515).

If transmission errors occur, issue the ECHO command, using each of the options U, S, and X to test the transmission of data between the terminal and the computer.

CHANGING OBJECT PROGRAMS

Files which contain relocatable object code and have a filetype of TEXT can be read into core storage and have linkages resolved by the LOAD, USE, and REUSE commands.

Five types of cards can be added to a TEXT file. These are the Set Location Counter (SLC), the Include Control Section (ICS), the Replace (REP), the ENTRY, and the LIBRARY cards. These are used to set the core location where LOAD begins placing the file in core, to make corrections and additions to the relocatable object code in core once the file is loaded, to specify entry points, and to specify references that are not to be resolved. These cards can be added to the TEXT file(s) which have been OFFLINE PUNCH'ed and can then be read back in, or they can be added using the SPLIT and COMBINE commands or the EDIT command.

The filetype REPS automatically places a 12-2-9 punch in column 1 and begins placing the user's data in column 7 of the file.

Set Location Counter (SLC) Card

The Set Location Counter Card sets the location counter used with the loader. The file loaded in after the SLC card is placed in core beginning at the address set by this SLC card. The SLC card has the format shown in Figure 43. It sets the location counter in one of three ways:

a. With the absolute address specified as a hexadecimal number in columns 7-12.

b. With the symbolic address already defined as a program name or entry point. This is specified by a symbolic name punched in columns 17-22.

c. If both a hexadecimal address and a symbolic name are specified, the absolute address is converted to binary and added to the address assigned to the symbolic name; the resulting sum is the address to which the loader's location counter is set. For example, if 0000F8 was specified in columns 7-12 of the SLC card image and GAMMA was specified in columns 17-21, where GAMMA has an assigned address of 006100 (hexadecimal), the absolute address in columns 7-12 is added to the address assigned to GAMMA giving a total of 0061F8. Thus, the location counter would be set to 0061F8.

If there are blanks in both columns 7-12 and 17-22, or the symbolic name has not yet been defined, the response INVALID CARD xxx...xxx is typed out--or, depending on whether the LOAD option SINV or PINV was specified, is written in the file LOAD MAP. If only the symbolic address is to be used, columns 7-12 must be left blank or all zeros. If only the absolute address is to be used, columns 17-22 must be left blank.

COLUMN	CONTENTS
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	SLC Identifies the type of load card.
5-6	BLANK
7-12	HEXADECIMAL ADDRESS to be added to the value of the symbol, if any, in columns 17-22. Must be right-justified in these columns, with unused leading columns filled in with zeros.
13-16	BLANK
17-22	SYMBOLIC NAME whose assigned location is used by the loader. Must be left-justified in these columns. If blank, the address in the absolute field is used.
23	BLANK
24-72	May be used for comments or left blank.
73-80	Not used by the loader. The user may leave these blank or insert program identification for his own convenience.

Figure 43. Format of an SLC card

Include Control Section (ICS) Card

The ICS card changes the length of a specified control section or defines a new control section. It should be used only when REPLACE cards cause a control section to be increased in length. The format of an ICS card is shown in Figure 44. An ICS card must be placed at the front of the card deck or TEXT file.

COLUMN	CONTENTS
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ICS Identifies the type of load card.
5-16	BLANK
17-22	CONTROL SECTION NAME--left-justified in these columns.
23	BLANK
24	, (comma)
25-28	HEXADECIMAL LENGTH IN BYTES of the control section. This must not be less than the actual length of the previously specified control section. Must be right-justified in these columns with unused leading columns filled in with zeros.
29	BLANK
30-72	May be used for comments or left blank.
73-80	Not used by the loader. The user may leave these blank or insert program identification for his own convenience.

Figure 44. Format of an ICS card

Replace (REP) Card

A REPLACE card allows instructions and constants to be changed and/or additions made. The REP card must be punched in hexadecimal code. The format of a REP card is shown in Figure 45. The data in columns 17-70 (excluding the commas) replaces what has already been loaded into core, beginning at the address specified in columns 7-12. REP cards are placed in the card deck either (1) immediately preceding the last card (END card) if the text deck does not contain relocatable data such as address constants, or (2) immediately preceding the first RLD (relocatable dictionary) card if there is relocatable data in the text deck. If additions made by REP cards increase the length of a control section, an ICS card, which defines the total length of the control section, must be placed at the front of the deck.

COLUMN	CONTENTS
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	REP Identifies the type of load card.
5-6	BLANK
7-12	HEXADECIMAL STARTING ADDRESS of the area to be replaced as assigned by the assembler. Must be right-justified in these columns with unused leading columns filled in with zeros.
13-14	BLANK
15-16	ESID --EXTERNAL SYMBOL IDENTIFICATION-- the hexadecimal number assigned to the control section in which replacement is to be made. The LISTING file produced by the compiler indicates this number.
17-70	A maximum of 11 four-digit hexadecimal fields, separated by commas, each replacing one previously loaded halfword (two bytes). The last field must not be followed by a comma.
71-72	BLANK
73-80	Not used by the loader. The user may leave these blank or insert program identification for his own convenience.

Figure 45. Format of a REP card

Entry Card

The ENTRY statement specifies the first instruction to be executed. It can be placed before, between, or after object modules or other control statements. The format of the ENTRY statement can be seen in Figure 46. The "externalname" is the name of a control section or an entry name in the input deck. It must be the name of an instruction, not of data.

ENTRY	externalname

Figure 46. Format of an ENTRY card

The loader selects the entry point for the loaded program in the following ways:

a. From the paramter list on the START command.

b. From the last ENTRY statement in the input.

c. From the first assembler- or compiler-produced END statement that specifies an entry point if no ENTRY statement is in the input.

d. From the first byte of the first control section of the loaded program if no ENTRY statement and there is no assembler- or compiler-produced END statement specifying an entry point.

Example:

```
ENTRY GO
```

where GO is defined as the external name of the first instruction to be executed when the program is loaded.

The address of the instruction, indicated by the symbolic name GO, is specified by the loader as the starting point of the program when it is executed.

Library Card

The LIBRARY statement can be used to specify the never-call function. The never-call specifies those external references that are not to be resolved by the automatic library call during any loader step. It is negated when a deck containing the external name referred to is part of the input to the loader. The format of the LIBRARY card can be seen in Figure 47. The "*" specifies the never-call function; the "external reference" refers to an external reference that may be unresolved after input processing. It is not to be resolved. Any additional external references within the subscript must be preceded by a comma. The LIBRARY statement can be placed before, between, or after object decks or other control statements.

LIBRARY	* (external reference)

Figure 47. Format of the LIBRARY card

Example:

```
LIBRARY * (SINE)
* specifies the never-call function.
SINE is an external reference in the output.
```

As a result, if SINE is unresolved after input processing, no automatic library call is made.

CMS BATCH MONITOR

The Batch Monitor can be run from a terminal like conversational CMS. The Batch Monitor permits use of the FORTRAN (G), ASSEMBLER (F), and PL/I (F) compilers. Temporary work files, referred to as FORTRAN logical files, are stored on disk and should be of limited size. The Batch Monitor's permanent disk is erased and CMS is re'IPLed whenever a JOB card is processed.

SAMPLE BATCH JOB

Source, object, or data cards included in the job stream are stored as separate CMS files. Each file is identified by a filename and filetype. FORTRAN, ASSEMBLE, and TEXT control cards indicate the filetypes FORTRAN, SYSIN, and TEXT, respectively. If unspecified, the filename is chosen by the system.

When a FORTRAN file is compiled or a SYSIN file is assembled, an object module is generated as a file with a filetype of TEXT and the same filename as the source file.

Data files to be read as a FORTRAN logical unit must be identified as FILE FT0nF001 where n is 1, 2, 3, 4, or 8. Files created by writing on one of the above FORTRAN logical units are identified, in the same way. These files may be PRINT'ed or PUNCH'ed using the appropriate control card and specifying the filename and filetype.

Execution of object modules (TEXT files) is initiated by the GO control card. The control card causes all TEXT files to be loaded into core. Execution begins at the entry point specified in the GO control card or at the first program encountered in the job stream. If a program interruption occurs during program execution, the programmer gets a complete core dump and the job is terminated. Data cards following the GO control card can be read as FORTRAN logical unit 5 (sysin). Any data cards following the GO control card which are not read are ignored. An example of a CMS Batch job deck is shown in Figure 48.

```
//jobname JOB                                    Standard Job Card
// userid acctno.
// FORTRAN
     .
     .
   Fortran Program 1 Source Code
     .
     .
   Fortran END card
     .
     .
   Fortran Program 2 Source Code
     .
     .
   Fortran END card
// ASSEMBLE ABLE
     .
     .
   Assemble Program 1 Source Code
     .
     .
   Assembler END card
     .
     .
   Assembly Program 2 Source Code
     .
     .
   Assembler END card
// PUNCH ABLE SYSIN
// TEXT
     .
     .
   Object Module
     .
     .
// GO
     .
     .
   Data Cards
     .
     .
// PRINT FILE FT01F001
// ASSEMBLE ABLE (NOLIST)
     .
     .
   Second Version of Assembly Program ABLE
     .
     .
   Assembler END card
// GO ABLE
```

Figure 48. Sample CMS Batch stream

CMS BATCH CONTROL CARDS

A deck to be submitted for CMS Batch must have an OS/360 standard job card. The second card must be // userid acctno. The control cards for CMS Batch specify a procedure name together with optional parameters. Control cards must have // in columns 1 and 2, and column 3 must be blank. Procedure names and parameters are each separated by one or more spaces. The CMS Batch control cards are described in subsequent sections.

To simplify the transition from an OS/360 job to a CMS Batch job, selected OS/360 control cards are translated into CMS Batch control cards to perform the corresponding functions. The OS/360 procedure names which are recognized, and the CMS Batch control cards they are equivalent to, are specified below.

OS Procedure Name	Batch Control Card
FORTRAN	// FORTRAN
LARGFORT	// FORTRAN
LINKEDIT	// TEXT
LINKOBJ	// TEXT
LINKSRC	// TEXT
EXECUTE	// GO
DISKEXEC	// GO
ASSEMBLE	// ASSEMBLE
ASEMLINK	// ASSEMBLE
PUNCH	// PUNCH
TPPCHOBJ	// PUNCH

Notes:

a. The only OS/360 // EXEC cards that are recognized by CMS Batch are the ones specified above; all other OS/360 control cards are ignored.

b. Only one job step per procedure call is allowed.

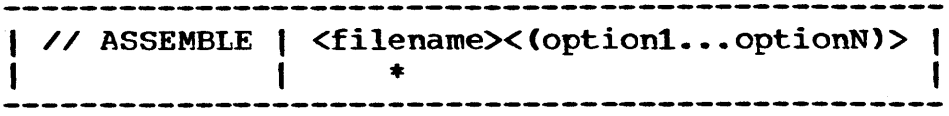
c. A list of CMS Batch control cards can be found in Figure 49 (see following sections for specific formats).

```
// ASSEMBLE
// COMMAND
// CP
// DATASET
// FORTRAN
// GO
// PRINT
// PUNCH
// TEXT
```

Figure 49. CMS Batch control cards

// ASSEMBLE

Format:



filename specifies the name of the file which contains the cards following the // ASSEMBLE control card up to the next control card. The filetype is SYSIN.

* If no parameters are specified, a filename is chosen by the system and the default options are taken.

Options:

LIST creates the LISTING file.
NOLIST suppresses creation of the LISTING file.

XREF creates the cross-reference symbol table within the LISTING file.
NOXREF suppresses the creation of the symbol table. This option is ignored if the NOLIST option was specified.

RENT causes the assembler to generate messages in the LISTING file if nonreentrant coding is found in the assembled program.
NORENT suppresses the generation of messages in the LISTING file if nonreentrant coding is found in the assembled program.

Usage:

All cards following the // ASSEMBLE control card up to the next control card are assumed to be Assembly Language source cards. These cards are read and stored as a file and translated using the F-level Assembler and the specified options. The source cards may contain any number of routines, each delimited by an END statement. If a filename is specified, the source cards are stored as a file with this filename and the filetype of SYSIN. If filename is not specified, the system chooses a unique name. The options specified govern the translation of the source cards. Any combination of options may be specified in any order.

An assembly generates a LISTING and a TEXT file. The LISTING is automatically written out on sysout. The TEXT file (that is, object module) can be punched using a // PUNCH control card. The filename of the TEXT file is the same as the filename of the Assembly Language source file.

Object modules are loaded into core and executed when the //

GO control card is specified.

Output:

LIST provides a listing of the assembled machine-language code, together with the source statements and an external symbol directory.

XREF includes a cross-reference symbol table with the listing.

RENT includes messages in the listing for nonreentrant coding found in the assembled program.

Diagnostic and error messages appear at the end of the listing.

Note:

The assembler searches the CMS macro libraries, SYSLIB MACLIB and OSMACRO MACLIB, for macro definitions. Additional macros may be included with the source program.

References:

For information on the Assembly Language, see IBM OS/360 Assembler Language (C28-6514) and Assembler F Programmer's Guide (C26-3756). For information on the System/360 machine, see IBM System/360 Principles of Operation (A22-6821).

// COMMAND

Format:

```
-----------------------------
| // COMMAND | anycmscommand|
-----------------------------
```

anycmscommand any valid CMS command may be executed except CPFUNCTN.

Usage:

The // COMMAND enables any CMS command to be entered and processed within the BATCH job stream, thereby enabling the user to perform operations not directly possible under Batch. For example, he could create EXEC files and execute them, or use the other CMS language processors in addition to FORTRAN and ASSEMBLER.

Examples:

a. // COMMAND PLI PROGA
This causes the compilation of the file PROGA PLI by the PL/1 (F) compiler.

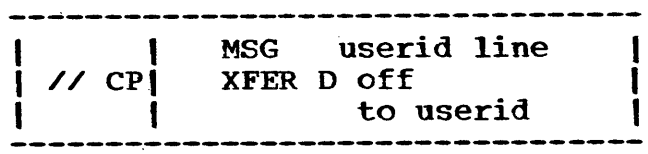
b. //COMMAND EXEC FORTCLG TEST1.
This causes the file FORTCLG EXEC to be executed, and would pass the argument TEST1 to it. The file FORTCLG EXEC would have been previously entered via the // DATASET FORTCLG EXEC card.

Note:

The CPFUNCTN command cannot be issued.

// CP

Format:



Usage:

The // CP card enables the user either to send messages or to transfer his punched output. If Batch is running in a disconnect mode, the user may send decks to be processed, and, via the // CP XFER, he may receive his punched decks back at his own virtual machine.

Examples:

a. // CP XFER D TO MINE
This causes all subsequent punched output to be sent to the user MINE.

b. // CP MSG CP YOU MAY DISCARD MY OUTPUT
The message YOU MAY DISCARD MY OUTPUT is typed on the operator's console.

// DATASET

Format:

```
-------------------------------------------------
| // DATASET          |   filename        filetype|
-------------------------------------------------
```

filename filetype specifies the identifier to be given to the file which contains the cards following the // DATASET control card to the next control card.

Usage:

The cards following the // DATASET control card up to the next control card are read and stored as a file. If the file identifier is of the form FILE FT0nF001, the file can be read as a FORTRAN logical unit.

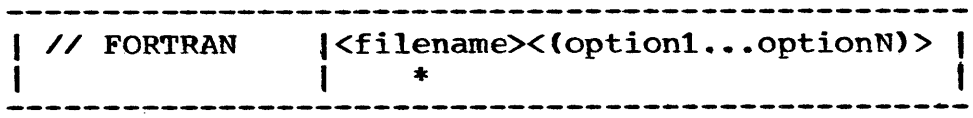
Examples:

a. // DATASET FILE FT01F001

b. // DATASET QUIET EXEC

// FORTRAN

Format:



filename specifies the name of the file which contains the cards following the // FORTRAN control card up to the next control card. The filetype is FORTRAN.

* If no parameters are specified, a filename is chosen by the system and the default options are taken.

Options:

MAP includes tables of addresses of FORTRAN variables in the LISTING file.
NOMAP suppresses the tables of variables.

LIST includes a listing of object code in Assembly language mnemonics in the LISTING file.
NOLIST suppresses the object code listing.

BCD causes the source program to be interpreted using the Binary Coded Decimal code.
EBCDIC causes the source program to be interpreted using the Extended Binary Coded Interchange Code.

Usage:

All cards following the // FORTRAN control card up to the next control card are assumed to be FORTRAN source code. These cards are read and stored as a file and compiled using the FORTRAN-G compiler under the specified options. The source cards may contain any number of routines, each delimited by an END statement. If a filename is specified, the source cards are stored as a file with this filename and a filetype of FORTRAN. If filename is not specified, the system chooses a unique name. Any combination of options may be specified in any order.

A compilation generates a LISTING and a TEXT file. The LISTING file is automatically written on sysout. The TEXT file can be punched using a // PUNCH control card. The filename of the TEXT file is the same as the filename of the FORTRAN source file.

TEXT files are loaded into core and executed when the // GO control card is specified.

Output:

MAP produces a table of addresses for each of the classifications of variables in the source program -- COMMON, EQUIVALENCE, NAMELIST, FORMAT, scalar variables, and called subprogram names.

LIST provides a listing of the object program with relative addresses and instructions translated into assembly language.

Diagnostic and error messages produced by the compiler are placed in the LISTING file and printed on sysout.

Notes:

a. Source cards can be punched in either Binary Coded Decimal (BCD) or Extended Binary Coded Decimal Interchange Code (EBCDIC). If BCD code is used, the BCD option must be specified for the compilation.

b. The entry point for the first main program is the same as the filename. Subsequent main programs have the entry point MAIN. The entry point for subroutines are specified in the SUBROUTINE statement.

c. Data can be read from sysin using FORTRAN logical unit 5 and written onto sysout using FORTRAN logical unit 6.

d. Only FORTRAN logical units 1-6 and 8 may be used. The FORTRAN logical files are defined as follows:

Logical File	Record Length
1-4	80-character records
5	80-character sysin records
6	130-character sysout records
8	133-character records with carriage control

e. An ID is punched in columns 73-80 of object decks. For subroutines, the first four letters of the subroutine name are used in columns 73-76. For main programs the first four letters of the filename are used if it is the first deck in the file, otherwise MAIN is used. Columns 77-80 contain a sequence number.

f. Information in the IBM System/360 Operating System Job Control Language is not applicable under CMS. The LOAD, NAME=, and LINECNT= options are not supported.

References:

For information on the FORTRAN IV language, see IBM System/360 FORTRAN IV Language (C28-6515), and IBM

System/360 Operating System FORTRAN IV Library: Mathematical and Service Subprograms. For information on compiler operation and messages, see IBM System/360 Operating System FORTRAN IV (G and H) Programmer's Guide.

// GO

Format:

// GO	< entry point >
	default entry point

entry point specifies the name of a control section or entry point to which control is passed at execution time.

default entry point is the beginning of the first program encountered in the job stream.

Usage:

// GO causes all TEXT files to be loaded into core, together with the required modules from SYSLIB TXTLIB, and proper linkages to be established between the program modules. Programs are loaded at X'12000' and may extend to X'3D000'.

A load map containing the location of control sections and entry points of programs loaded is created and printed on sysout. After loading, execution is begun by transferring control to the entry point.

Data cards following the // GO control card can be read from FORTRAN programs using logical unit 5. Data cards not read are ignored. Output written onto FORTRAN logical unit 6 is written onto sysout and printed.

Note:

An entry point must be either a control section name or an entry point name. It may not be a filename unless the filename is identical to a control section name or an entry point name.

Examples:

a. // GO

b. // GO ENTRY1

Error Messages:

E(00001) DEFINED MORE THAN ONCE -xxxxxxxx

E(00002) OVERLAY ERROR
The files being loaded have run out of core.

E(00003) REFERENCE TABLE OVERFLOW
There are too many entries for the entry points or control section names in the reference table.

E(00004) THE FOLLOWING NAMES ARE UNDEFINED -xxxxxxxx
The names xxxxxxxx are referred to and have never been defined.

E(00005) NAME IS UNDEFINED - xxxxxxxx
The name xxxxxxxx specified as an entry point does not exist.

// PRINT

Format:

```
---------------------------------
| // PRINT | filename filetype |
---------------------------------
```

filename filetype specifies the name of the file to be printed on sysout.

Usage:

Each line of the specified file is truncated to 130 characters and printed on sysout. The first character of each line of a LISTING file is used as a printer carriage control character.

These carriage control characters are interpreted as follows:

CHARACTER	EBCDIC	ACTION
" "	40	print line and space 1
"0"	F0	space 1, print line, space 1
"1"	F1	skip to new page, print line, space 1
	xx	any character other than the above is used as a CCW

Example:

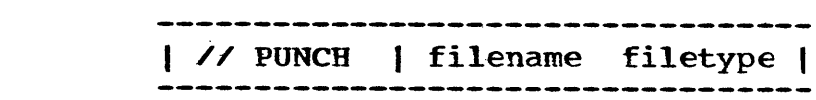
```
// PRINT FILE FT01F001
```

Error Message:

```
E(00003)  FILE NOT FOUND
```

// PUNCH

Format:



filename filetype the identifier of the file to be punched

Usage:

Files with records up to 80 characters in length are punched into an 80-column card.

Examples:

a. // PUNCH PROG1 TEXT
The file PROG1 TEXT is punched.

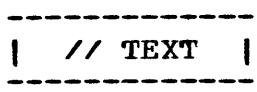
b. // PUNCH FILE FT01FT001
The FORTRAN logical file 1 is punched.

Error Message:

E(00002) FILE NOT FOUND

// TEXT

Format:



Usage:

The cards following the // TEXT control card up to the next control card are read and stored as a file. A unique filename is chosen by the system and the filetype is TEXT.

Example:

```
// TEXT
```

RUNNING THE CMS BATCH MONITOR

Setup

CMS Batch accepts an input stream from either the card reader or from tape. The punched output can go to tape or to cards and the printed output can go to tape or to the printer. The tape assignments are as follows:

Symbolic Address	Virtual Address	Contents	Track
TAP5	185	SYSIN	9
Tap6	186	SYSOUT	9
TAP7	187	PUNCH	9

If the SYSIN tape is not attached to the BATCH machine, the system assumes that the input stream comes from the card reader. If the PUNCH tape is not attached, the system writes the PUNCH'ed output to the online punch. If the SYSOUT tape is not attached, the printer is used.

Unlabeled tapes are used. The SYSIN tape consists of unblocked card images. The PUNCH tape consists of unblocked card images, but each job is a separate file where the first record contains the job number and the programmer's name. The SYSOUT print tape consists of unblocked 133-character records.

Each job is limited in number of minutes of execution time and number of lines of printed output. These limits may be reset by the system programmer. The default settings are five minutes and 5000 lines.

Using CMS BATCH

To run the CMS BATCH stream,

a. LOGIN to CP as the BATCH user,

b. IPL the Batch Monitor's Nucleus,

c. Issue the START command.

d. Optionally, ATTN may be hit to enter CP, and DISCONN may be entered to run BATCH in the disconnect mode.

When a job is canceled, a message is written on SYSOUT indicating this to the programmer, along with an explanation as to why this was done. The following messages may be printed:

a. TIME LIMIT EXCEEDED
The running time of the job exceeded the allowable execution

time.

b. OUTPUT LIMIT EXCEEDED
The number of output lines exceeded the allowable length of output.

Notes:

a. Each job is limited to minutes of execution time and lines of printed output. If either of these limits is exceeded, the job is canceled. These limits may be set by the system programmer at your installation.

b. When an end of file is encountered on the input device, the message END OF INPUT STREAM is typed on the console. An enabled-WAIT PSW is loaded, and a DEVICE END interrupt on the input device reinitiates batch operations. Thus, a job may be XFER'ed to BATCH, and BATCH processes it.

c. An accounting card is punched for every job that Batch runs. This card is spooled to the virtual machine whose userid is OPERATOR, and is punched out in the CP account card format.

Example:

The login procedure for using the Batch Monitor is

```
cp-67 online      xd.65 qsyosu
1 cmsbatch
ENTER PASSWORD:

READY AT 13.32.38 ON 09/08/69
CP
ipl ccu
READY
start
```

where ccu is the address of the disk containing an IPL'able copy of the BATCH nucleus.

Optionally, the ATTN key may be hit once to enter CP, and DISCONN may be typed in. The terminal is then free for use by another virtual user, while BATCH continues to process its job stream, or waits for a stream to be XFER'ed to it.

GLOSSARY

Terms with specific meanings in CP and CMS are described below in alphabetical order.

ACTIVE DISK TABLE. A table residing in the user's copy of the CMS nucleus which contains an entry for each of the six disks that the user may access with CMS.

ACTIVE FILE TABLE. A table residing in the user's copy of the CMS nucleus which contains an entry for each of that user's currently opened files.

ARGUMENT. Any alphameric information, not exceeding eight bytes in length, the address of which is to be passed to a program at the time it begins executing or to a CMS command.

ATTENTION INTERRUPT. A signal to the system which effects a transfer of control between the Control Program and other environments. The terminal keyboard is unlocked, regardless of current processing, and the input line is processed by the environment in control.

ATTENTION KEY. A key on the terminal which, when hit, causes an attention interrupt. This key is labeled ATTN on the 2741, and RESET LINE on the 1050.

CARD IMAGE. An 80-character logical record in which each character corresponds positionally to the columns of a punched card.

CARRIAGE RETURN. The signal which indicates to the system the termination of a line of input from the terminal. This signal is transmitted on the 2741 by hitting the key labeled RETURN; on the 1050, it is transmitted either by holding down the ALTN CODING key while hitting the 5 key, or (if the 1050 is equipped with the Automatic EOB special feature) by hitting the RETURN key.

CHARACTER-DELETE SYMBOL. A character appearing on the terminal keyboard which, when hit n times, deletes the preceding n characters and itself from the input line. Currently defined as the @ character, but can be redefined in CMS by the CHARDEF command.

CMS FUNCTION. A routine available to the CMS command programs for the handling of internal processing, such as accessing and updating disk file directories or handling disk and terminal I/O.

CMS NUCLEUS. The core-resident portion of CMS of which each user receives a copy at the time he issues an IPL 190 or IPL CMS console function.

CONSOLE FUNCTION. A software facility whereby the user, at his terminal, can simulate a function he would normally be able to perform at a 360 console. The command facilities of the Control Program are referred to collectively as CP console functions.

CONTROL SECTION. A block of coding that can be relocated, independent of other coding, without altering or imparing the operating logic of the other coding.

CP/CMS SYSTEM. A time-sharing system in which the Cambridge Monitor System (CMS) runs as the operating system of a virtual machine created by the Control Program (CP).

CPU TIME. The period of time during which the central processing unit of the computer is actively engaged in the processing of instructions.

DEFAULT ENTRY POINT. The core location at which execution begins if no starting location is specified; either that given in the first nonblank operand of an END card image or, if all END operands are blank, the beginning of the first labeled control section of the loaded program(s).

ENTRY POINT. Any symbol in a control section which can be used by other control sections to effect a branch operation or a data reference.

ENVIRONMENT. That portion of the CP/CMS system which has control at the time an input line is transmitted from the terminal, and which processes that input line to determine its acceptability. Only a subset of all possible input is acceptable in any given environment.

ERROR MESSAGE. The message "E(xxxxx);T=xx.xx/xx.xx hh.mm.ss", where xxxxx is the error code returned in general-purpose register 15, the first xx.xx is the virtual CPU time in seconds, and the second xx.xx is the total CPU time in seconds used since the last Ready or Error message. Any information typed at the terminal which explains the meaning of the error code may also be considered part of the error message.

FILE DIRECTORY. A table for each disk file storage area which indicates the file identifier, file size, and location of each file stored in that area. For example, the system file directory contains information for each file stored on the system disk.

FILE IDENTIFIER. A three-part designation which uniquely identifies each file stored on disk. This identifier consists of a filename (any descriptive term), a filetype (indicating file contents), and a filemode (indicating file location).

INPUT LINE. All information, up to a maximum of 130 characters in length, typed by a user between the time the typing element of his terminal comes to rest following a carriage return until another carriage return is issued.

LINE-DELETE SYMBOL. A character appearing on the terminal keyboard which, when hit, deletes all preceding characters in the input line and itself. Currently defined as the ¢ character, but can be redefined by the CHARDEF command in CMS.

LINE-END CHARACTER. A character appearing on the terminal keyboard which, when hit, causes a logical end of the input line so that information typed before and after this character is interpreted as if entered on separate lines (that is, a logical carriage return). Currently defined as the # character, but can be redefined by the LINEND command.

LINKAGE. The resolving of external references between control sections at load time.

LOAD MAP. A file containing the core locations of control sections and entry points of programs loaded into core.

MACRO LIBRARY. A disk file (whose filetype is MACLIB) containing macro definitions in assembler language source code and a dictionary of the name, size, and location of each macro definition within the file.

NULL LINE. An input line consisting of a carriage return issued as the first and only information after the typing element of the terminal has come to rest following a previous carriage return.

OFFLINE DEVICE. A device whose I/O is temporarily stored in a spooling area by the Control Program; namely, the card reader, printer, and card punch.

ONLINE. Any operation performed at a terminal which is actively connected to the computer.

OPERAND. Any field, delimited by one or more blanks, which may be specified in a command, request, or console function. The operands are distinct from the command, request, or console function name, which is always the first field specified.

OUTPUT. Any message or information typed by the system (as opposed to the user) at the terminal. This term is also used to refer to information punched onto cards, printed on the printer, or written out on magnetic tape.

OVERRIDE. A flag set internally to indicate whether or not

the user has requested the recording of trace information.

PAGING AREA. A secondary storage area on disk which is assigned to a particular virtual machine, and is used by the Control Program for temporary storage of portions of core belonging to that virtual machine, in order to allocate main storage dynamically among the various users.

PARAMETER LIST. A string of doublewords used whenever a CMS command or function is called by an SVC instruction. The format of the parameter list varies depending on the command or function being called, but always contains the name and operands of that command or function.

PERMANENT DISK. A disk area allocated to each user (at the time he is authorized to use the CP/CMS system) on which stored files are retained until the user requests that they be deleted.

READ-ONLY. An indication that the user may have access to the contents of a disk, but may perform no WRITE operations to that area (that is, he may only READ it).

READY MESSAGE. The message "R;T=xx.xx/xx.xx hh.mm.ss" which is typed as a response indicating the successful completion of a CMS command and a return to the CMS command environment. The first xx.xx is the virtual CPU time and the second xx.xx is the total CPU time in seconds used since the last ready or error message.

REQUEST. Input acceptable only to an environment which is unique to a specific CMS command.

RESPONSE. Any nonerror message typed out by the system at the terminal.

SPOOLING AREA. Any disk area used by the Control Program to temporarily hold input from the offline card reader or output to the offline card punch or printer.

SYSTEM DISK. A disk area containing the CMS nucleus, of which each user receives his own copy, and the disk-resident portion of CMS, which is shared by all users.

SYSTEM FILE. Any file residing on the system disk as opposed to the user's permanent or temporary disks.

TEMPORARY DISK. A disk area allocated to the user at the time he logs into the Control Program, on which stored files are retained only for the duration of the terminal session.

TERMINAL SESSION. The period between a user's completed login to CP until he logs out from CP. (Note that new copies of the CMS nucleus may be obtained during a terminal session.)

TEXT LIBRARY. A user or system file, whose filetype is TXTLIB, which is composed of TEXT files containing relocatable object code and a dictionary indicating the location and size of each of these TEXT files within the library.

TOTAL CPU TIME. The time required to execute the virtual machine's instructions plus the overhead of CP for running that virtual machine.

TRACE INFORMATION. Data (such as the contents of various registers and parameter lists) recorded by the system to enable the user to trace transfers to and from SVC-called programs.

UNIT RECORD DEVICE. A card-reader, card punch, or printer.

USER FILE. A file residing on the user's permanent or temporary disk as opposed to the system disk.

USERID. Any combination of from one to eight characters which uniquely identifies a user to the Control Program.

VIRTUAL CPU TIME. The time required to execute the virtual machine's instructions. This time includes no overhead of CP.

VIRTUAL MACHINE. A functional simulation of a computer and its associated devices. The Control Program, by creating several virtual machines and allocating the hardware facilities of a single computer among them, creates an atmosphere in which the users of the virtual machines may each function independently and with different operating systems.

VIRTUAL MEMORY. The amount of core that the virtual machine has--this need not be the same as the memory of the real machine or the memory of any or all other virtual machines.

APPENDIX A: CONTROL PROGRAM CONSOLE FUNCTIONS

Each of the CP-67 console functions that can be issued by the terminal user is described below.

BEGIN — begins execution at the specified address or, if no address is given, at the location at which execution was interrupted.

CLOSE — releases the spooling areas containing input from the card reader or output to the printer or card punch.

DETACH — removes the specified device from the user's virtual machine configuration.

DIAL — is used in place of LOGIN to connect a user's terminal with a virtual telecommunications operating system or a virtual time-sharing system.

DISCONN — allows a user to disable the terminal and leave his virtual machine running.

DISPLAY — types at the terminal the contents of the specified register(s), core location(s), or program status word.

DUMP — prints the contents of the specified register(s), core location(s), or program status word on the offline printer.

EXTERNAL — simulates an external interrupt to the virtual machine, causing control to pass to the CMS DEBUG command.

IPL — simulates the Initial Program Load sequence on the specified unit.

IPLSAVE — simulates the Initial Program Load sequence on the specified unit, but does not zero core first.

LINK — attaches virtual disks dynamically.

LOGOUT — releases the user's virtual machine, including his temporary disk area, and closes any spooling areas which have not been released.

MSG — types the specified message at the terminal of the person whose USERID is specified.

PURGE — erases spooled input or output files by device.

QUERY — types out either the number of logged-in and dialed users, the names of logged-in users, the

number of current spooled input and output files, or the current clock time and CPU time used.

READY — simulates a device end for the specified unit.

RESET — simulates the system reset key on the 360 console by resetting any pending I/O interrupts.

SET — controls the saving of virtual card reader files, the operation of the virtual machine even when in the CP environment, and the typing of messages at the terminal.

SLEEP — places the terminal in a dormant state to receive messages.

SPOOL — directs spooled output and controls the reading of spooled input.

STORE — replaces the contents of the specified register(s), core location(s), or program atus word with the specified information.

XFER — controls the passing of files between users.

APPENDIX B: CMS Commands

The commands that can be issued by a user to CMS are described below.

ALTER changes all or part of the identifier (filename, filetype, and filemode) of a file stored on the user's permanent or temporary disk without altering the contents of the file.

ASSEMBLE converts assembler language source code into relocatable object code using the OS/360 F level assembler.

BLIP causes a specified string of characters to be typed at the console every two seconds of CPU execution.

BRUIN invokes the Brown University Interpreter, in which the user has a desk-calculator mode and a stored-program mode. BRUIN is available as a Type III program.

CHARDEF redefines the character-delete symbol, the line-delete symbol, the logical tab character, and the logical backspace character, which default to the @, ¢, #, and % respectively.

CLOSIO signals the Control Program that I/O to offline unit record equipment has been completed and that the spooling areas for this I/O may be processed. CLOSIO is generally issued automatically by the commands which access unit record equipment.

CLROVER clears overrides set by the SETERR and/or SETOVER commands, and causes all recorded trace information to be printed on the offline printer.

COMBINE copies the specified file(s), concatenating them in the order given, into a new file, which is placed on the user's permanent or temporary disk and assigned the specified identifier.

COMPARE compares two disk files.

CPFUNCTN issues CP console functions without leaving CMS.

CNVT26 converts BCDIC files to EBCDIC.

CVTFV converts files of fixed-length records to variable-length records.

DEBUG allows the user to stop and restart programs at specified points, and to inspect and change the

contents of registers, core locations, and hardware control words online.

DISK causes a CMS disk file to be punched out or read in from cards which are in CMS card format.

DUMPD prints the contents of a direct-access record, specified by a CCHHRR address, in hexadecimal on the printer.

DUMPF types the contents of all, or part, of a specified file in hexadecimal on the console.

DUMPREST dumps the contents of an entire disk to magnetic tape, or restores the contents of an entire disk from magnetic tape.

ECHO tests terminal line transmission by repeating as typeout whatever is typed in by the user.

EDIT allows the user to create files on disk, to make changes to existing files from his terminal, and to peruse the contents of files.

ERASE deletes the entry for a specified file (or files) from the appropriate directory, rendering the file inaccessible to the user, and freeing the disk area containing that file.

EXEC executes a file containing one or more CMS commands, allowing a sequence of commands to be executed by issuing a single command.

FILEDEF allows the user to specify the I/O devices which are used by his program. This command is not currently supported by the CMS language processors.

FINIS closes the specified file (or files) by writing the last record of that file on disk, updating the user-file-directory and removing the entry for that file from the user's table of active files.

FORMAT prepares the user's permanent or temporary disk area for CMS use by writing blank records over the currently stored information.

FORTRAN converts FORTRAN language source code into relocatable object code using the OS/360 FORTRAN G compiler.

GENMOD creates a nonrelocatable core-image file on the user's permanent disk, which is a copy of the contents of core between two given locations.

GLOBAL specifies macro definition libraries to be searched during the assembly process or text libraries to be searched when loading files containing relocatable object code.

KO clears overrides previously set by the SETOVER or SETERR commands, and causes all trace information recorded by these commands to be printed on the offline printer.

KT stops typeout at the terminal for the duration of the currently executing command or user program.

KX terminates the currently executing program, updates the user-file-directory, reIPL's CMS and returns control to the CMS command environment.

LINEND redefines the logical line-end character which allows multiple commands to be entered on one line. The default is the #.

LISTF either types out at the terminal the identifier, size and creation date or change date of the specified disk file(s), or creates a file on the user's permanent disk containing information for use by the EXEC and/or $ commands.

LOAD reads the specified TEXT file(s)--containing relocatable object code--from disk or a library, loads them into core, and establishes the proper linkages.

LOADMOD reads a MODULE file--which is in nonrelocatable core-image form--from disk, and loads it into core.

LOGIN causes the user's permanent disk files to be either saved or deleted, as specified, and allows a specified disk to be logged-in as the P-disk. If LOGIN is not issued, the files are saved.

LOGOUT compacts the user-file-directory, executes any CMS command specified as an operand, and logs out of CMS, transferring control to the Control Program.

MACLIB generates, adds to, replaces, deletes, or compacts macros in a specified macro library, or types out the contents of the dictionary of that library.

MAPPRT types or creates a file containing the load map associated with the CMS nucleus.

MODMAP types the load map associated with a core-image MODULE file on the console.

OFFLINE creates disk files from card input, prints a disk file on the offline printer, or punches a disk file on cards.

OSTAPE creates disk files from card images on tape.

PLI converts PL/I language source code into relocatable object code, using the OS/360 PL/I F compiler.

PRINTF types at the terminal the contents of all, or part, of a specified disk file.

RELEASE releases a disk from a user's virtual machine when he has finished using it.

REUSE reads the specified TEXT file(s)--containing relocatable object code--from disk, and loads them into core, establishing linkages with previously loaded files and changing the default entry point of these files to that of the first file specified in the REUSE command.

RT restores typing at the terminal that was previously suppressed by KT.

SCRIPT types or prints out the contents of the specified file, formatting it, as indicated, by control words contained in the text.

SETERR sets error overrides which cause trace information to be recorded for each SVC-called program which returns with an error code in general-purpose register 15.

SETOVER sets normal and error overrides which cause trace information to be recorded for all SVC-called programs--both those which are executed normally, and those which return an error code in general-purpose register 15.

SNOBOL converts a card-image file in Snobol source language into SPL/1 interpreter language and executes SPL/1 programs. SNOBOL is available as a Type III program.

SORT arranges records of a disk file in ascending order and writes the sorted output into a new file.

SPLIT copies the specified portion of a card-image file, and either creates a new file or appends it to a second specified card-image file.

START	begins execution of the loaded program(s) at the specified or default entry point and passes the address of a string of user arguments to the program(s).
STAT	types statistics regarding the amount of permanent and/or temporary disk space used, or compacts the user-file-directory.
STATE	tests whether a file exists.
SYN	allows the user to specify his own command abbreviations to be used with, or in place of, the standard system abbreviations.
TAPE	writes the contents of CMS disk files of any type or size onto magnetic tape, or restores these files by writing them from tape onto disk.
TAPEIO	writes a tape mark on magnetic tape, erases a gap on the tape, or moves the tape.
TAPRINT	copies LISTING files from tape to printer.
TPCOPY	copies tape files.
TXTLIB	generates, adds to, or deletes modules in a specified text library, types out the contents of the dictionary for that library, or creates a file containing a list of entry points and control section names contained in that library.
UPDATE	updates the specified disk file with a file containing control cards, where each control card indicates whether the information immediately following it is to be resequenced, inserted, replaced, or deleted.
USE	reads the specified TEXT file(s)--containing relocatable object code--from disk and loads them into core, establishing linkages with previously loaded files.
WRTAPE	copies files of fixed-length records from disk to tape.
$	executes a file containing one or more CMS commands, or loads into core a file which is in either core-image form or relocatable object code and begins execution of that file.

APPENDIX C: DEBUG REQUESTS

BREAK	specifies a core location at which a program currently loaded into core is to be stopped during its execution.
CAW	types out the contents of the channel address word as it existed when DEBUG was entered.
CSW	types out the contents of the channel status word as it existed when DEBUG was entered.
DEF	enters the specified symbol in the DEBUG symbol table, allowing it to be used thereafter in other DEBUG requests to refer to a specific core location.
DUMP	prints out the contents of the specified portion of core on the offline printer.
GO	begins execution either at a specified location or at the point where execution was interrupted when the DEBUG environment was entered.
GPR	types out the contents of the specified general-purpose register(s) as they existed at the time DEBUG was entered.
IPL	performs an initial program load sequence on the version of the CMS nucleus which has been saved by CP.
KX	terminates the currently-executing program; and logs out from CMS, transferring control to the CP environment.
ORIGIN	establishes a "base" address which is added to all hexadecimal locations specified in other DEBUG requests.
PSW	types out the contents of the old program status word which was saved at the time DEBUG was entered.
RESTART	reinitializes the CMS system, leaving the user in the CMS Command environment.
RETURN	returns the user from the DEBUG environment to the CMS Command environment.
SET	changes the contents of the specified general-purpose register, channel address word, channel status word, or program status word by replacing them with the specified information.
STORE	changes the contents of the specified core

location by replacing them with specified information.

TIN — Selects which environment is to handle terminal I/O, CMS or DEBUG.

X — types at the terminal the contents of a specified or assumed number of bytes of core, starting at the specified location.

APPENDIX D: EDIT REQUESTS

The EDIT requests are described below in alphabetical order. These are the only valid input to the Edit environment, although some of the requests affect the format of the input to the Input environment.

BACKSPACE defines a logical backspace character for use in both the Edit and Input environments. The default character is %.

BLANK places blanks in the indicated columns of the line at which the internal pointer is currently positioned.

BOTTOM positions the pointer after the last line of the file.

BRIEF causes the brief mode of the Edit environment to be entered, in which lines found or altered by Edit requests are not automaticallly typed out.

CHANGE replaces a specified string of information currently in the file with another specified string of information.

DELETE deletes the specified number of lines from the file, starting with the line at which the pointer is currently positioned.

FILE writes the edited file on the user's permanent disk and transfers from the Edit environment to the CMS Command environment.

FIND scans each line of the file, starting at the line immediately following the one at which the pointer is currently positioned, for a column-dependent match with the specified information.

INPUT transfers the user from the Edit environment to the Input environment.

INSERT inserts the specified line of information into the file immediately after the line at which the pointer is currently positioned.

LOCATE scans the contents of the file, starting at the line immediately following the one at which the pointer is currently positioned, for the specified string of information.

NEXT moves the pointer forward in the file for the number of lines specified.

OVERLAY replaces characters in the line at which the

pointer is currently positioned with the nonblank character(s) specified.

PRINT — types out the contents of a specified number of lines, starting with the line at which the pointer is currently positioned.

QUIT — transfers from the Edit environment to the CMS Command environment without saving the edited file.

REPEAT — repeats the following BLANK or OVERLAY request the specified number of times.

RETYPE — replaces the contents of the line at which the pointer is currently positioned with the line of specified information.

SAVE — writes the edited file on the user's permanent or temporary disk and returns to the Input environment.

SERIAL — establishes whether identification information is to be placed in each line of the file and, if so, specifies that identification information.

TABDEF — defines a character which is to be recognized as the logical tab character in the Edit and Input environments. The default character is #.

TABSET — establishes the internal or logical tab settings which are to be used in both the Edit and Input environments.

TOP — positions the pointer to a null line which precedes the first line of information at the beginning of the file.

UP — repositions the pointer the specified number of lines above the current line.

VERIFY — causes the verify mode of the Edit environment to be entered, in which the contents of lines found or altered by Edit requests are automatically typed out.

ZONE — specifies the columns to be scanned by LOCATE and CHANGE.

APPENDIX E: SCRIPT CONTROL WORDS

Page Layout

.BM n — specifies the number of lines to be allowed for the bottom margin of each page (the default value is 3).

.HM n — specifies the number of lines to be skipped between the heading and the first line of text, excluding the forced space (.TM), (the default value is 1).

.PL n — specifies the number of lines to be typed on a page, (the default value is 66).

.TM n — specifies the number of lines, including the header line, to be allowed for the top margin of each page, (the default value is 5).

Page Control

.CP n — causes a page eject to occur if less than the specified number of lines remain on the current page.

.HE heading — causes the next line to be used as a header line, and to be typed at the top of each subsequent page.

.PA n — starts the next line on a new page, with n as the page number. If n is omitted, sequential numbering of pages is assumed.

.PN — controls both external and internal page numbering of the file being printed.

Spacing

.DS — doublespaces the information being typed out.

.SP n — inserts the specified number of carriage returns before typing the next line.

.SS — singlespaces the information being typed out.

Paragraph Layout

.IN n — indents the left side of the printout n number of spaces.

.LL n — specifies the line length in characters, (the default value is 60).

.OF n — indents the following lines n spaces from the left margin, except for the first line which

follows this command.

.TB specifies the tab stops to be assumed for the following lines when converting the TAB character into the appropriate number of spaces. (The default values are 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75.)

.UN n forces the immediately following line to start n spaces further left than the position currently set by .IN.

Formatting Modes

.BR causes a break, meaning that information appearing before and after the .BR request is typed on separate lines.

.CE centers the next line between the left and right margins.

.CO cancels a previous .NC, causing output lines to be formed by concatenating input lines and truncating at the nearest word to the specified line length. It does this by shifting words to or from the next input line. The resulting line is as close to .LL as possible without exceeding it.

.FI cancels a previous .NF (or .NC and/or .NJ), causing concatenation and right justification of output lines.

.FO same as .FI

.JU cancels a previous .NJ (or part of a .NF) causing right justification of output lines. It does this by padding lines with extra blanks to have an even right margin.

.NC stops words from shifting to or from the next line.

.NF causes lines to be typed exactly as they appear in the file. It is the same as issuing both .NC and .NJ.

.NJ stops padding lines with blanks to cause right justification.

Special Features

.CM causes the remainder of the line to be ignored, allowing comments to be stored within the SCRIPT file.

.RD n issues the specified number of reads to the terminal to allow user input to be inserted in the printout.

Manuscript Layout

.AP name appends the contents of the specified SCRIPT file to the file just printed.

.IM name inserts the contents of the specified SCRIPT file into the printout of another SCRIPT file.

APPENDIX F: CMS FUNCTIONS

ATTN	stacks a line into the input buffer.
CARDIO	reads or punches a card into or from the specified 80-byte area.
CONWAIT	waits for all stacked reads and writes to "finish" from the console typewriter.
CPFUNCTN	transmits console functions to CP-67 without leaving the virtual machine mode.
ERASE	erases the specified file(s).
FINIS	closes one or more specified files.
HNDINT	sets the CMS I/O interrupt handling routines to transfer control to a given location for an I/O device other than those normally handled by CMS, or clears such transfer requests.
HNDSVC	initializes the SVC interrupt handler to transfer control to a given location for a specific SVC number (other than X'CA' or 202), or clears such previous handling.
POINT	sets the read pointer and/or the write pointer at a specified item number.
PRINTR	outputs lines on a printer.
RDBUF	reads an item of information from a disk file
STATE	provides a copy of the FST (file status table) entry for the file specified.
TAPEIO	reads, writes, or moves magnetic tape.
TRAP	sets a user's return for an external interrupt. This return overrides the call to DEBUG on an external interrupt.
TYPE	types an output message on the console. Terminal blanks (if any) are not deleted and no carriage return is added.
WAIT	wait state is entered to await an interrupt from one of the devices specified.
WAITRD	reads an input message up to 130 bytes in length into a given buffer from a console and waits for completion of the input message.
WRBUF	writes one item of information into the file specified. If the file does not exist, a new

file is opened and given the specified identifier. It automatically packs fixed-length items into an 800-byte buffer and writes this 800-byte block onto the disk.

APPENDIX G: FORMAT OF COMMANDS, REQUESTS, AND CONSOLE FUNCTIONS

Below is a key to the symbols used to represent command formats in this appendix:

UPPERCASE	information given in capitals must be typed exactly as shown, although it may be entered in either uppercase or lowercase.
lowercase	lowercase information designates the contents of a field, and does not in itself constitute meaningful input.
( )	parentheses must be typed as shown when any of the information appearing within them is specified
.	a period designates the beginning of a Script Print control word, and must be typed as shown
-	a hyphen must be typed where shown, and must not be offset by blanks
/	a slash denotes any string delimiter, which does not appear in the string, other than blank.
*	an asterisk, specified where shown, indicates the universality of an item or items

The following are logical symbols only, and should not be typed:

< >	brackets indicate information which may be omitted
< >< >	successive brackets enclose items which, if specified, may appear in any order
<< >>	nested brackets indicate items which, if specified, must appear in the order shown
...	an ellipsis indicates that the preceding item(s) may be repeated more than once in succession
1 2 3 N	these suffixes indicate first, second, third, and Nth items respectively

underlining indicates the value which is assumed if none is specified. When no underlined item appears in bracketed < > information, the default value is none.

stacked items, not enclosed in anything, indicate that only one item may be specified.

All lowercase symbols used in this appendix are described in alphabetical order below.

anycom	any CMS command
arg	any user argument to be passed to a program
c	any text character appearing on the 2741 keyboard
ccu	device address
defent	default entry point
defset	default option settings
editfn	filename specified in EDIT command
entry	an entry point or control section name
eof	end of file
first	first occurrence of the specified item found
fm	filemode
fn	filename
ft	filetype
hexinfo	a fullword or less of hexadecimal information
hexloc	hexadecimal core location
id	an alphameric identifier
intloc	core location at which processing was interrupted
lastset	core location specified in last DUMP request in the Debug environment
length	inherent length attribute
libname	the filename of a library
line	a group of position-dependent 2741 characters
n	a decimal number
name	saved system name
nolineno	line number not typed
norec	no formatting of first three disk records
newfm	new filemode
newfn	new filename
newft	new filetype
nextloc	next available load location
oldfm	old filemode
oldfn	old filename
oldft	old filetype
option	an option of the indicated command
reg	the number of a register (in decimal)
seq	sequential page numbering
setmar	original margin or tab settings
string	any group of 2741 characters
symbol	a name for which a DEF request has been issued in the Debug environment
syslib	system text library, unless otherwise

	specified by an option or previous command
typeout	type out information at the terminal
userid	identifier by which user logs in to CP

Note. For abbreviations, see "CMS Commands".

** BRUIN and SNOBOL are available as Type III programs.

CMS COMMANDS

```
ALTER          oldfn  oldft  oldfm  newfn  newft  newfm
                 *      *      *      *      *      *

ASSEMBLE       fn1 ... fnN  <(option1 ... optionN)>

BLIP           <char> <n>
                        1
               <(OFF)>

BRUIN
  **

CEDIT          <fn>  ft

               E
CHARDEF        C     <Char>
               L
               T

               READER
               RDR
CLOSIO         PRINTER      < OFF >
               PRT            ON
               PUNCH
               PU

CLROVER

CNVT26         fn     ft

COMBINE        newfn  newft  newfm  oldfn1 oldft1  oldfm1
                    ... oldfnN  oldftN  oldfmN

COMPARE        fn1  ft1  fn2  ft2

CPFUNCTN       <NOMSG>    cpcommand

CVTFV          fn   ft   <(T)>
```

```
DEBUG

DISK            DUMP fn ft fm
                LOAD

DUMPD           ccu  cc  <hh  <rr>>

                          n1   n2
DUMPF           fn   ft << *     *    n3 >>
                          beg  end  80

DUMPREST

                   S
ECHO            <  X  >   < nn >
                   U         1

EDIT            <fn>   ft

ERASE           fn   ft   < fm >
                *    *      *

EXEC            fn  <arg1 ... argN>

                  ddname   device
FILEDEF         < dsrn     CLEAR    <def1 ... defN>>
                   *       DUMMY

FINIS           fn   ft     fm
                *    *   <   *     >
                           first

                P       ALL     <(NOTYPE)>
FORMAT          T     <  C   >
                         L
                         nn
                         R <SYS>

FORTRAN         fn1 ... fnN   <(option1 ... optionN)>
```

```
GENMOD          entry 1 <<entry2 > <NOMAP)>>
                        nextloc   MAP

GLOBAL          ASSEMBLER MACLIB
                      (m)       <libname1 ... libnameN>
                LOADER TXTLIB           SYSLIB
                      (T)
                PRINT

IPL

KO

KT

KX

                <<fn   ft   < fm >>>
LISTF             *    *      *     <(option1 ... optionN) >
                                P
                                T

LOAD            fn1 ... fnN  < (option1 ... optionN)

                             <libname1 ... libnameN>>
                                      syslib

LOADMOD         fn   <fm>

LOGIN              NOPROF
                <  UFD  >
                   NO_UFD
                   NO-UFD
                   ccu      < UFD                      >
                               NO_UFD
                               Z  <,Y <fn <ft <fm >>>
                               P

LOGOUT          <anycom>
```

```
MACLIB          GEN  libname  fn1 ... fnN
                ADD  libname  fn1 ... fnN
                LIST libname
                REP  libname  fn1 ... fnN
                DEL  libname  macroname1 ... macronameN
                COMPACT  libname
                PRINT  libname

                  A     ON
MAPPRT          < N    <OFF  >>
                  C     NO

MODMAP          fn

OFFLINE         PUNCH
                PUNCHCC
                PUNCHDT
                PRINT
                PRINTCC           fn  ft   <fm>
                PRINTUPC                   first
                PRINTVLR
                PRINTPLI
                READ              fn  ft   < fm >
                                      *      P1

OSTAPE          < filetype  <filename>> (option1 ... optionN)
                  SYSIN       OSTAPE

PRINTF          fn  ft  < n1   n2  >   < n3   >
                          *    *        length

RELEASE         ccu fmode < (DETACH) >

REUSE           fn1 ... fnN  <(option1 ... optionN)

                             <libname1 ... libnameN>>

RT

SCRIPT          fn  <(option1 ... optionN)>

SETERR
```

```
SETOVER         <SAMELAST  <option1 ... optionN>>
                   defset

SNOBOL          fn  <(option1 ... optionN)>

SORT            fn1   ft1   fn2   ft2

SPLIT           fn1  ft1  fn2  ft2  id1  <id2 >
                                     n1    n2
                                           eof

START           <entry      <arg1 ... argN>>
                 *
                 defent

                  P
STAT            <  T  >
                  C
                P and T

STATE           fn  ft   < fm >
                           P

SYN             < fn  ft  fm >  < (option1 ... optionN) >
                  *  syn

TAPE            DUMP     fn  ft  <fm>
                         *   *    P1
                LOAD     <n>
                          1
                REWIND                      <  TAPn  >
                SCAN   <n>                     TAP2
                SKIP   <n>
                SLOAD   fn  ft   <n>
                WRITEOF <n>

TAPEIO          BSF
                BSR
                ERG             < TAP1 >
                FSF               TAP2
                FSR
                REWIND
                RUN
                WRITEOF
                WTM
```

```
TAPRINT        < TAP1 >
                 TAP2

TPCOPY         < TAPi > < TAPo >  < n >  < YES >
                 TAP1     TAP2      1      NO
                   *        *       *

TXTLIB           GENERATE      libname fn1 ... fnN
                  G
                 ADD           libname fn1 ... fnN
                  A
                 PRINT         libname
                  P
                 LIST          libname
                  L
                 DELETE        libname  CSect 1 ... CSectN
                  D

UPDATE         fn1  <<  ft1   <  fn2   <  ft2  >>>  (P)  >
                      SYSIN      fn1    UPDATE     .fn1

USE            fn1 ... fnN  <(option1 ... optionN)

                            <libname1 ... libnameN>>

WRTAPE         fn  ft

$              fn  <arg1 ... argN>
```

DEBUG REQUESTS

```
BREAK          n   symbol
                   hexloc

CAW

CSW

DEF            symbol  hexloc  < n >
                                 4

DUMP                     symbol1   symbol2
                 id  < hexloc1   hexloc2 >
                         0         *
                                   3

GO               symbol
                < hexloc >
                 intloc

GPR            reg1  <reg N>

IPL

KX

ORIGIN          symbol
                hexloc

PSW

RESTART

RETURN

SET            CAW   hexinfo
               CSW   hexinfo  <hexinfo>
               PSW   hexinfo  <hexinfo>
               GPR   reg       hexinfo  <hexinfo>
```

STORE

symbol
hexloc hexinfo <hexinfo <hexinfo>>

TIN

< CMS >
  DEB

X

symbol < n >
hexloc < length / n / 4 >

EDIT REQUESTS

```
BACK              < c >
                    1

BACKUP            < N >
UP                  1
U

BLANK             line
B

BOTTOM
BO

BRIEF
BR

CHANGE             /string1/string2/      *     *
C                                       < n   G     >>
                                          1   first

DELETE               n
D                 < /string/ >
                     1

FILE              <  fn  >
                    editfn

FIND               line
F

INPUT
I

INSERT             line
I

LOCATE             /string/
L

NEXT              < n >
N                   1
```

```
OVERLAY          line
O

PRINT              n      LINENO
P               < 1    <    L    >>
                          nolineno

QUIT
Q

REPEAT          < N >
                  1

RETYPE           line
R

SAVE            < fn      >
                  editfn

SERIAL           id    <  n  >
SER              (NO)    10

TABDEF          <  c  >
TABD               #

TABSET          < n1 ... nN >
TABS               setmar

TOP
T

UP
U               < n >
BACKUP            1

VERIFY          < n >
VER               72

ZONE             n1    n2
Z                1     trunc01
```

CONTROL PROGRAM CONSOLE FUNCTIONS

BEGIN B	<hexloc> intloc
CLOSE C	ccu
DETACH DE	ccu
DIAL D	System
DISCONN	<anychar>
DISPLAY D	<hexloc><hexloc1-hexloc2><Lhexloc> <Lhexloc1-hexloc2><Greg><Greg1-reg2> <Xreg><Xreg1-reg2> <Yreg><Yreg1-reg2><PSW>
DUMP DU	<hexloc><hexloc1-hexloc2><Lhexloc> <Lhexloc1-hexloc2><Greg><Greg1-reg2> <Xreg><Xreg1-reg2> <Yreg><Yreg1-reg2><PSW>
EXTERNAL E	
IPL I	name ccu
IPLSAVE	name ccu

```
LINK            userid  xxx  yyy  W   <(NOPASS)>
LI              *                 R

LOGOUT          <anychar>
LOG

MSG             userid      line
M               CP

PURGE           READER
P               R
                PRINTER
                P
                PUNCH
                PU

QUERY           FILES
Q               F
                LOGMSG
                L
                NAMES
                N
                TIME
                T
                USERS       <userid>
                U
                userid

READY           ccu
R

RESET
RE

SET             CARDSAVE      <ON>
                               OFF
                MSG OFF
                MSG ON

                RUN ON
                RUN OFF

SLEEP
```

```
SPOOL           xxx  <ON  yyy>
                      OFF
SP              ccc  <CONT>
                      OFF


STORE           <Lhexloc hexinfo1 ... hexinfoN>
ST
                <Greg hexinfo1 ... hexinfoN>

                <Xreg hexinfo1 ... hexinfoN>

                <Yreg hexinfo1 ... hexinfoN>

                <PSW hexinfo1  hexinfo2>


XFER            ccu TO userid
X                           *
                ccu OFF
```

.AP	fn	appends file fn
.BM	<n> 1	bottom margin set to n
.BR		causes a break
.CE		centers following line
.CM	line	"line" is a comment
.CO		concatenates lines
.CP	n	new page if n lines are not left on current page
.DS		doublespacing
.FI		concatenates and right-justifies each line
.FO		same as .FI
.HE	line	"line" becomes the heading on subsequent pages
.HM	n	heading margin set to n
.IM	fn	imbeds file fn in the file at point specified
.IN	<n >	indents lines n spaces starting in the following line
	setmar	
.JU		causes right justification of lines
.LL	<n > 60	line length set to n

.NC		turns off concatenation
.NF		turns off concatenation and right justification
.NJ		turns off right justification
.OF	<n > setmar	offsets lines n spaces to the right starting with second line following
.PA	<n > seq	page eject with n as next page number
.PL	<n> 66	sets length of output pages to n
.PN	ON OFF OFFNO	determines if external and/or internal page numbering is to occur
.RD	<n> 1	allows n lines to be input from the terminal during output
.SP	<n> 1	causes n carriage returns
.SS		singlespacing
.TB	<n,...nN > 5, 10, 15 ... 75	sets physical tabs that correspond to logical tabs
.TM	<n> 5	top margin set to n
.UN	n	starts following line n spaces to the left of current margin

APPENDIX H: CP-67 MACHINE CONFIGURATION

CP-67 is structured to run on an IBM System/360 Model 67.

CP-67 MINIMUM CONFIGURATION

2067-1 or 2067-2 Processing Unit

Recommended feature:

#4434 Floating Storage Addressing (Model 1 only)

2365 Processor Storage

2860 Selector Channel

2870 Multiplexer Channel

1052 Printer-Keyboard

1403 Printer

2540 Card Read Punch

Three 2311 Disk Storage Drives or 2314 Direct Access Storage Facility

(2 Disk Storage Modules minimum)

2400 Nine-Track Magnetic Tape Unit, 800 or 1600 bpi

2702 or 2703 Transmission Control or

2701 Data Adapter Unit

TERMINALS SUPPORTED BY CP-67 AS OPERATOR's CONSOLE

1051/1052 Model 2 Data Communication System
Features and Specifications:
Data Set Attachment (#9114)
IBM Line Adapter (#4647)
Receive Interrupt (#6100 or RPQ E27428) required
Transmit Interrupt (#7900 or RPQ E26903) required
Text Time-out Suppression (#9698) required

1056 Card Reader

2741 (-1,-2) Communication Terminals
Features and Specifications:
Data Set Attachment (#9114)
Data Set Attachment (#9115)
IBM Line Adapter (#4635, #4647)
Dial-Up (#3255)
Receive Interrupt (#4708) required
Transmit Interrupt (#7900) or Transmit Interrupt Control (RPQ E40681) required
Print Inhibit (#5501) desirable

Line control for teletypewriter terminals* compatible with the IBM Telegraph Terminal Control Type II Adapter (8-level ASCII code at 110 bps).

* The customer is responsible for terminal compatibility with this program. IBM assumes no responsibility for the impact that any changes to the IBM supplied products or programs may have on terminals provided by others.

TRANSMISSION CONTROL UNITS SUPPORTED BY CP-67

2701 Data Adapter Unit

Terminals	2701 Adapter
8-level ASCII, 110 bps*	7885

* The customer is responsible for terminal compatibility with this program. IBM assumes no responsibility for the impact that any changes to the IBM supplied products or programs may have on terminals provided by others.

2702 Transmission Control

Terminals	Terminal Control Base	Terminal Control	Line Adapter
2741s, 1050	9696 or 7935	4615, 9684, 8200**	3233
8-level ASCII 110 bps*	9697 or 7935	7912	3233

** Feature 8200 on the 2702 is equivalent to the 2741 Break feature #8055 and the Type 1 Break RPQ E46765 on the 2702.

2703 Transmission Control

Terminals	Line Speed Option	Line Set	Terminal Control	Line Bases
2741s, 1050	4878	3205/6	4619, 4696, 8200***	7505
8-level ASCII, 110 bps*	4877	3205/6	7905, 7912	7505

*** Feature 8200 on the 2703 is equivalent to the 2741 Break feature #8055 and the Type I Break RPQ E53715 on the 2703.

OTHER DEVICES SUPPORTED BY CP-67

Additional Devices Utilized by CP-67

2301 Drum Storage
2303 Drum Storage

2870 Multiplexer Channel with #6990, #6991, #6992
1, 2, 3 Selector Subchannels

DEVICES USED ONLY BY AN OPERATING SYSTEM IN A VIRTUAL MACHINE AND NOT BY CP-67

2321 Data Cell Drive

2400 Magnetic Tape Units

2250 Display Unit
2260 Display Station

2860 Selector Channel with
#1850 Channel-to-Channel Adapter

2780 Data Transmission Terminal
1130 Computing System

APPENDIX I: DEVICES SUPPORTED BY CMS

Core size: Minimum 256K or up in multiples of 8K (up to 16M virtual)

1052 Printer-Keyboard

Six 2311 Disk Storage Drives or

2314 Direct Access Storage Facility

(2 disk storage modules minimum)

2540 Card Reader/Punch

1403 Printer

Two 2400 series tape drives, nine or seven track

200, 556, 800 or 1600 bpi

(one nine track, 800 or 1600 bpi required for installation)

GH20-0859-0

Control Program-67/Cambridge Monitor System UG Printed in U.S.A. GH20-0859-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

Control Program-67/Cambridge Monitor System
User's Guide

GH20-0859-0

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold fold

fold fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, STAPLE AND MAIL.

GH20-0859-0

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Technical Publications

fold fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]