9999-0015-01
d1

320-2015

CP-67/CMS
USER'S GUIDE

IBM Cambridge Scientific Center Report

International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts
October, 1967
Revised July , 1968

320-2015
October, 1967
Revised July, 1968
Scientific Center Report

CP-67/CMS
USER'S GUIDE

International Business
Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts

Abstract

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

Index Terms for the IBM Subject Index

Computer Systems
IBM 036-67
Time-Sharing
Virtual Systems
Conversational Computing
Interactive Computing
Remote Computing
On-Line Debugging

07-Computers
21-Programming

CP-67/CMS USER'S GUIDE

Table of Contents

Date

Date

Date

INTRODUCTION

Components of the System

The CP/CMS time-sharing system is composed of two independent components: the Control Program (CP) and the Cambridge Monitor System (CMS).

CP builds and maintains for each user a "virtual machine". The virtual machine is indistinguishable to the user and his programs from a real machine, but is really one of many CP 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.

Since the virtual machines are simulated, their configurations may differ from each other and from the real machine. Each user controls his virtual machine from his terminal, which is, effectively, his console keyboard.

Like real machines, virtual machines will 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 virtual machine, at the 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 "in it", nothing he does can affect any other user. In addition, since users cannot get "outside" their virtual machines, CP is protected from any user error.

The Control Program: Virtual Machines

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 a table describing the virtual machine assigned to that user. Whenever he logs in, CP sets up this virtual machine for him. Although all the virtual machines may be different, most will be set up with the configuration expected by CMS, the most commonly used operating system. They will include at least 256K bytes of core storage, two disk drives, a console (the terminal), a card-read-punch unit, and a printer - another disk drive and two tape drives are optional (tapes must be enabled on request, by the system operator). The real system will usually have a larger number of disk drives and/or a drum, more 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; as active and inactive pages change status they are "paged" in and out of real core. While the paging operation is being performed for one virtual machine, another virtual machine can be operating. The paging operation

and resultant allocation of real core to a given user's pages, appear random to the system. 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 randomly located pages. This is called "dynamic address translation"; it is transparent to the user.

Because of the virtual machine concept employed by this system, only the Control Program (CP) may operate in the supervisor state on the real machine. All programs other than CP -- all programs executed on virtual machines -- operate in the problem state on the real machine. By a special interrupt-handling procedure, however, CP supports what amounts to a virtual supervisor state on the virtual machine. All user interrupts, including those caused by attempted privileged operations, are handled by CP, which reflect to the user program only those the user program would expect from a real machine. The user may expect his programs to execute on his virtual machine in a manner identical to their execution on a real machine.

Since many users, through their virtual machines, require system resources that together can total many times the resources of the real system, the system operator must occasionally take steps to maintain adequate response time for all users. CP recognizes that certain kinds of jobs on virtual machines -- large assemblies or compilations, for example -- can cause heavy paging loads or otherwise tax the Control Program. CP will attempt to schedule the different jobs so as to maximize the total throughput of the system while maintaining a fast response time for those users whose demands are lighter but whose responses must be quicker. The operator can also limit the total number of users supported at any one time. Operator-imposed parameters may be obtained by issuing the QUERY console function.

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 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 CCW string for the user, and issues SIO when the channel is free. When CP receives an interrupt indicating I/O completion, it sets a "ready-to-run" flag in the user's virtual machine status table; when control is returned to the virtual machine, the proper I/O interrupt is simulated. The virtual machine is not given control from the time it issues a SIO until CP delivers the simulated I/O interrupt. In the meantime, another virtual machine(s) may be operating.

All virtual machine unit record I/O is spooled on disk by CP. Thus, any card deck to be "read" by a virtual machine must have been read by CP prior to the user's call for it on his virtual machine; the physical deck must have been preceded by a card containing the USERID, so CP knows who the card-image file is for. 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 CLOSE console function specifies the (virtual) address of the device to be closed; or CP detects an invalid CCW address 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, printed and punched output, files are always preceded by a record (supplied by CP) that contains 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. The user can stop his virtual machine at any time (by depressing the ATTN key) and 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 QUERY function mentioned above, which are not normal console functio

The 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 (in this case, the user's remote terminal).

Whether running on a real or a virtual machine, CMS expects the following machine configuration:

device	virtual address	symbolic name	
1052	009	CON1	console
2311	190	DSK1	system disk (read-only)
2311	191	DSK2	permanent disk (user files)
2311	192	DSK3	temporary disk (work space)
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.			

Note that the above devices represent the minimum configuration for CMS except for the following three devices, which are optional: DSK3 at virtual address 192, TAP1 at virtual address 180, and TAP2 at 181.

Under CP, of course, these devices are simulated and remapped 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.

Under CP, all CMS users share the read-only system disk; on it reside the CMS nucleus routines, which the user IPLs, and the other routines and libraries that CMS calls as needed. Since each virtual machine maintains its own copy of CMS, users may modify it as they wish without affecting either the system disk version or the copies in other virtual machines.

CMS Commands:

CMS commands fall naturally into four categories: file manipulation, compilation, execution control, and debugging aids.

The file handling commands allow the user to create, copy, move, combine and erase disk files. Other commands provide access to the tape units, printer, and card-read-punch. Under the CMS linkage scheme, all of these commands are available to executing programs as well as to the user at the terminal.

The CMS language processors are the same ones used under Operating System/360 (OS); these include Assembler (F), FORTRAN IV (G), and PL/1 (F). The Assembler produces object programs that may be executed under either CMS or OS, depending on the macros used in the source program. Special file-handling routines for macro libraries are included. The FORTRAN and PL/1 compilers also produce OS-compatible object programs. (The FORTRAN execution-time support programs have been modified for CMS.) The SNOBOL compiler and assembler-interpreter were adapted from programs designed to execute under OS.

The execution control commands allow the user to load his programs from single object decks (the filetype TEXT is reserved for relocatable object programs) or from a library of programs. He can pass a list of parameters to his program from the terminal, and specify the point at which execution is to begin. To avoid relocation (bypass the relocating loader) he can create a file consisting of an image of the portion of core storage containing his program, and load that non-relocatable copy back at any time. Since the loading commands can be accessed by executing programs, overlay structures may be set up. The user can also create a file which is a series of commands, and then execute these commands by typing a single line (loading and executing that file).

The debugging facility of CMS allows the user to stop his programs at predetermined points and examine his registers, PSW, and storage, and modify these he desires. This information may be typed out at this terminal or printed off-line. A program interrupt gives control to DEBUG, as does the external interrupt caused by the EXTERNAL console function. The user may also employ the program tracing routines, which record all SVC transfers, or will record just those in which an error return is made.

2.0.0 CP/CMS CONVENTIONS

Various "rules" or conventions must be observed when using the CP/CMS system. These are discussed in the following sections. Included are a discussion of the file facilities available to the CMS user and the methods of identifying these files, a description of the 2741 and 1050 terminals and their use, and a description of the procedures to be used when logging in and out of the system. Also given are rules for typing input to the system from a terminal, and a description of each of the environments to which this input may be entered.

2.1.0 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 in Sections 3.1.0.

Conventions given in this section apply to disk files only.

Disk Facilities

Two disk areas are available to each CMS user for storing information. These are the permanent disk area, where stored information is retained until the user requests that all or part of it be deleted, and the temporary disk area, where information is retained only from the time it is created until the user ends his terminal session. For convenience, these two areas are often referred to as the user's "permanent disk" and his "temporary disk" respectively, although the size of each area seldom constitutes an entire physical disk. The sizes of a user's permanent and temporary disks are assigned by the system administrator at the time he establishes that person as an authorized user of the CP/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.

A third disk area accessed by each user is the system disk, which is composed of (1) the CMS nucleus, of which each user receives a copy, and (2) the disk-resident portion of CMS, which is shared by all users. The system disk is read-only; any attempt to write on it will be denied and will cause an error message to be typed to the user.

Information stored on the permanent, temporary, and system disks is organized into files. Files on the permanent and/or temporary disks are referred to as "user files"; those 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 will be erased.

The filename may be any combination of from 1 to 8 non-blank EBCDIC characters provided the first character is not a zero, an asterisk, a period, or a left parenthesis. With system files, the filename is the name which is issued by the user in calling a specific command, and 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 1 to 8 non-blank EBCDIC characters provided the first character is not a zero, an asterisk, or a left parenthesis. Certain filetypes imply specific file characteristics to CMS. With system files, filetype is used to indicate whether the file resides in the nucleus or in the disk-resident portion of CMS. Filetypes which have specific implications for user files are given in Figure 2.1.0-A. 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 will not be successfully executed if the contents of the file are in any form other than that which the assigned filetype implies.

Several of the CMS commands create files for their own use on the user's permanent disk 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
INPUT	FILE	EDIT
INPUT1	FILE	EDIT
(TEMP)	(FILE)	COMBINE
(DISK)	(TFILE)	DISK, TAPE
SNOBOL	TEMPFILE	SNOWBALL, SPL1
LISTF	EXEC	LISTF
LOAD	MAP	LOAD, $, USE, REUSE
(INPUT1)	FILE	SCRIPT
(INPUT2)	FILE	SCRIPT

In addition to the above, it should be noted that if a user file is created whose filename and filetype are identical to those of a file on the disk-resident portion of CMS, the user's file will be accessed in place of the system file. Reasons for this are described in Section 3.0.0.

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. For

These COMMANDS assume that files with these FILETYPES contain information in this FORMAT	
Assemble, Edit, Update	SYSIN — Assembler language source code
Fortran, Edit	FORTRAN — Fortran source code
PLI, Edit	PLI — PL/1 source code
Spl4, Snowball	SNOBOL — Snobol source code
Spl4	SPL4 — Spl4 source code
Script	SCRIPT — Script input
Load, Use, Reuse, $, Txtlib	TEXT — Relocatable object code
Txtlib, Global	TXTLIB — Relocatable object code library
Loadmod, $	MODULE — Non-relocatable object code
Exec, $	EXEC — Any combination of CMS commands
Update	UPDATE — Update control and replacement card images
Maclib, Global	MACLIB — Assembler language macro definition library
Maclib, Edit	ASP360 — Assembler language macro definitions
Assemble, PLI	SYSUT1, SYSUT2, SYSUT3 — Assembler or PLI utility files
Assemble, Fortran, Printf, PLI	LISTING — Assembler or compiler source statements and corresponding machine code
Assemble	DIAG — Assembler diagnostics
Txtlib	MAP — Library map file

FIGURE 2.1.0-A Filetype Implications

system files, the second character is always a "Y". For user files, the second character is a number from 1 to 6. 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 after the first read.

4 - file is read-only and is erased after the first read.

File Sizes

Files stored on disk will be formatted into records 800 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 12.848 million bytes, or 16,060 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 12.848 million bytes allowed by CMS.

Although there is no inherent limitation to the number of files a user may create, he is practically limited by the sizes of his permanent and temporary disk areas. When a user has filled either of these areas, a message to this effect will be typed out at his terminal. Refer to Section 5.1.0 for recovery procedures which should be used 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 will be automatically opened. "Opening" in this case consists of making an entry into the user's active file table. CMS can open an unlimited number of files, but the language processors impose certain restrictions on this number. For example, only 30 data set reference numbers are currently defined in FORTRAN. CMS commands likely to access more than eight files will close files (i.e., remove their entries from the active file table by executing a FINIS command) as necessary. Also, all open files are closed by CMS after the successful execution of any CMS command.

system files, the second character is always a "Y". For user files, the second character is a number from 1 to 6. 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 after the first read.
4	-	file is read-only and is erased after the first read.

File Sizes

Files stored on disk will be formatted into records 800 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 12.848 million bytes, or 16,050 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 12.848 million bytes allowed by CMS.

Although there is no inherent limitation to the number of files a user may create, he is practically limited by the sizes of his permanent and temporary disk areas. When a user has filled either of these areas, a message to this effect will be typed out at his terminal. Refer to Section 5.1.0 for recovery procedures which should be used 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 will be automatically opened. "Opening" in this case consists of making an entry into the user's active file table. Only eight entries may exist in this table at any given time. CMS commands likely to access more than eight files will close files (i.e., remove their entries from the active file table by executing a FINIS command) as necessary. Also, 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 except when he desires to access more than eight disk files with one of his own programs.

2.2.C 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".

Either of four terminals may be used to access the CP-67/CMS system. These are (1) the IBM 2741 Communications Terminal as shown in Figure 2.2.0-A, (2) the IBM 1050 Data Communications System terminal shown in Figure 2.2.0-B, (3) the type 33 teletype terminal, and (4) 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 (similar to that shown in Figure 2.2.0-C) will be 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 Section 2.2.2.

18

2.2.1 TERMINAL CHARACTERISTICS

2741 Characteristics

The IBM 2741 Communications Terminal consists of an IBM Selectric typewriter mounted on a typewriter stand (see Figure 2.2.0-A). 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. To be used with the CP/CMS system, the 2741 should be equipped with the Transmit Interrupt special feature and the Receive Interrupt RPQ.

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 in this section 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 2.2.1-A) and the standard Selectric configuration (shown in Figure 2.2.1-B). On either keyboard the alphanumeric 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 following keys have special significance on the 2741 keyboard:

RETURN - The RETURN key is hit to signal the termination of each input line. When the RETURN key is hit, control is transferred to the system and the keyboard is locked until the system is ready to accept another input line.

ATTN - The ATTN key is used to generate an attention interrupt. It may be hit at any time (since it is never locked out) and will cause the keyboard to be unlocked to accept an input line. Refer to Section 2.3.0 for a discussion of the transfer between environments which 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.

19

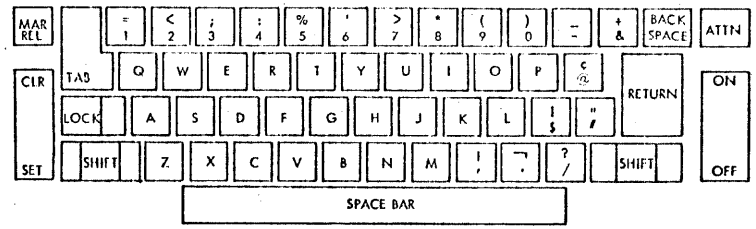


FIGURE 2.2.1-A IBM 2741 Keyboard (PTTC/EBCD Configuration)

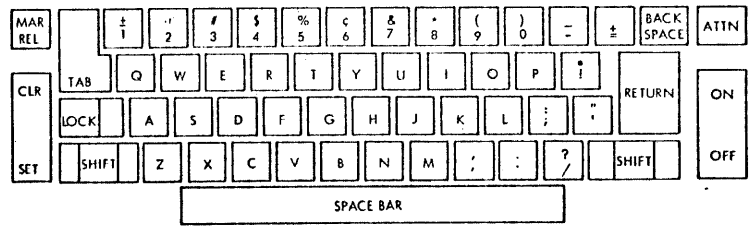


FIGURE 2.2.1-B IBM 2741 Keyboard (Standard Selectric Configuration)

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) will appear in terminal output as a vertical bar (|). For a further discussion of 2741 characteristics, refer to IBM Manual A24-3415.

2741 Initiation Procedures

The steps for readying the 2741 for use are described below. After these steps have been performed, proceed with the Login procedure described in Section 2.2.2.

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

2. After making sure that the terminal is plugged in, turn the power on by pressing down on 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 Section 3.1.6. 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 will clear all previous tab settings. New settings may be made by spacing the typing element to the desired location(s) and then pressing the SET portion of the tab control key. After tab stops have been set for the entire line, hit the RETURN key 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 as described in Section 2.2.2.

1050 Characteristics

The IBM 1050 terminal is composed of the 1051 Control Unit and a 1052 Printer-Keyboard, as illustrated in Figure 2.2.0-B. 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 special feature and the Transmit Interrupt and Receive Interrupt RPQs. The 1052 keyboard is similar in appearance to the standard IBM typewriter keyboard. Figures 2.2.1 C, D and E illustrate the 1050 paper controls, switch panel, and keyboard. The alphanumeric and special character keys, the space bar, the 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 the RETURN key is hit will be considered part of the input line.

ALTN CODING-this key, when pressed and held while one of the other keys is hit, will originate 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 will cause a carriage return at the terminal.

RESET LINE-hitting this key, at the left side of the keyboard, will cause an attnetion interrupt (provided the terminal is equipped with the Transmit Interrupt special feature.) The RESET LINE key may be hit at anytime, since it is never locked out, and will cause the keyboard to be unlocked to accept an input line. Refer to Section 2.3.0 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, the RESEND key may be hit to turn off the light, and the previous input line may then be re-entered. While the light is on, no input will be accepted from the keyboard.

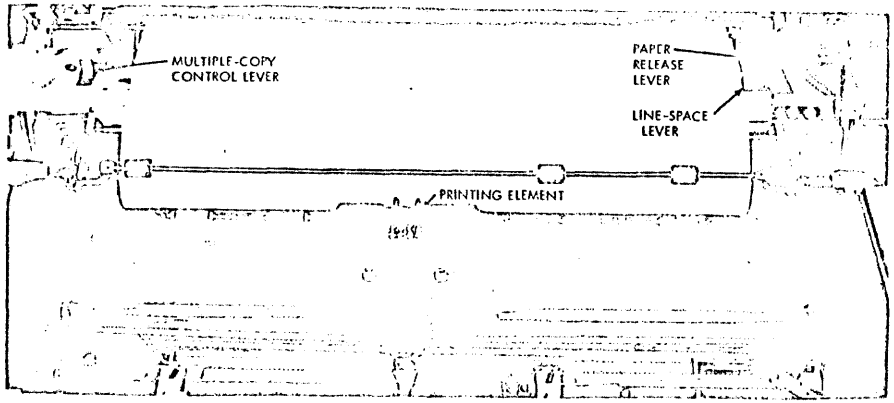


FIGURE 2.2.1-G IBM 1052 Paper Controls

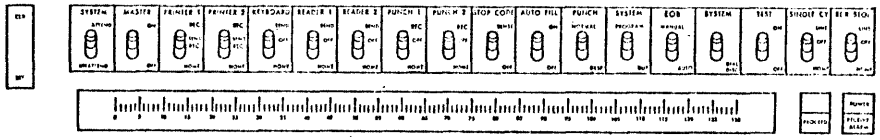


FIGURE 2.2.1-D IBM 1052 Switch Panel

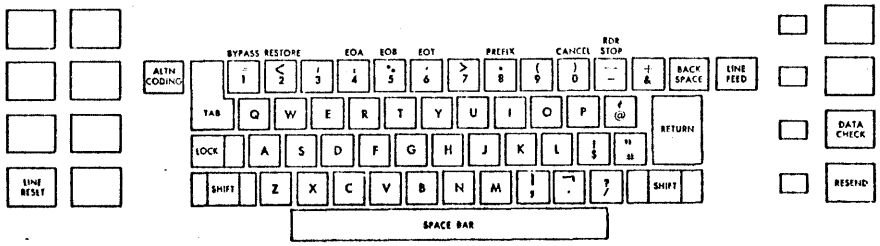


FIGURE 2.2.1-E IBM 1052 Keyboard

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 whenver 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) will appear in terminal output as a vertical bar (|).For further information on the characteristics and handling of the 1050 terminal, refer to IBM Manual A24-3020.

1050 Initiation Procedures

The procedure for readying the 1050 for use are described below. When these steps have been performed, proceed with the Login procedure described in Section 2.2.2.

1. After making sure that the terminal is plugged in, set the panel switches, shown in Figure 2.2.1-D. 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 visible 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 main-line switch shown in Figure 2.2.0-B 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 Section 3.1.6. 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 will clear all previous tab settings. New settings may be made by spacing the typing element to the desired location(s) 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 main-line switch to POWER ON and continue with the Login procedure discussed in Section 2.2.2.

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. Figure 2.2.1-F illustrates the keyboard and control panel. 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 - this key 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 over-printing of the next line.

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

REPT - this key repeats the action of any key depressed.

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

CNTRL - this key 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-F 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 it 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 LINE 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.

2.2.2 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 multi-access system such as RAX or APL, refer to Section 2.2.3, "Dialing a Multi-Access 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.

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 readied for use as described in section 2.2.1, the following procedure must be performed in order to gain access to the CP-67/CMS system. (Note that input may be entered in either upper or lower case. Upper case is used below to indicate words which must be typed as they are shown; lower case indicates fields whose contents may vary.)

1. First, 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, the CP Login procedure must be repeated. Otherwise, a Data-Phone similar to that shown in Figure 2.2.C-C will be 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 will remain 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 will acknowledge 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 will be 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 the attention key once. On the teletype 33 or 35, hit BREAK and then ERR-RLS.

4. The system will respond by either (1) unlocking the keyboard on a 2741 or 1052 or waiting for input on the teletype 33 or 35, or by (2) typing one of the following messages:

MAX NO. OF USERS EXCEEDED
If the keyboard unlocks or CP-67 waits for input, proceed to Step 5. If the message is typed, the system is already servicing the maximum number of users and the login procedure will be terminated. In this case wait for a few minutes and then try again by returning to Step 1.

SHUTDOWN IN PROGRESS
CP-67 is in the process of being taken down. When it is brought up again, return to Step 1.

5. Identify yourself to the system by typing:

LOGIN userid
followed by a carriage return, where "userid" is your user identification.

6. The system will respond with one of the following messages:

ENTER PASSWORD:
This message indicates that your user identification has been accepted. Proceed to Step 7.

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

USERID ON xxx
REPEAT PROCEDURE

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 login with the same userid until the other user has logged off.

```
USER NOT IN DIRECTORY.
REPEAT PROCEDURE
```

If this message is typed, an invalid userid has been specified. Return to Step 5 and login again.

7. Type your password, followed by a carriage return. If the 2741 terminal is equipped with the Print Inhibit feature, the password will not be 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 PCH
```

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

9. The system will respond 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 login 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.
REPEAT PROCEDURE
```

If this message is typed, an invalid password has been specified and the login procedure will be repeated. Return to Step 5.

10. At this point in time, any operating system can 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 OOC. If the device that is IPLed contains an operating system, such as OS/360, of that operating system and your terminal is effectively 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, February 1969.

CMS Initialization

11. To initialize CMS, issue the following console function:

```
IPL 190
```

or

```
IPL CMS
```

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

12. The message:

```
CMS...VERSION x.x   xx/xx/xx
```

will be typed, and the keyboard will be unlocked. The x.x is the version number and xx/xx/xx is the version date. The CMS Command environment will have 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 logout, he may enter this environment by hitting the attention key once. The keyboard will be unlocked and the user should type

```
LOGOUT
```

followed by a carriage return. The system will respond with

```
LOGOFF AT xx.xx.xx ON xx/xx/xx
```

and the connection to the computer will be lost. 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 such that another user may login from the terminal, the user should type

```
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 login, as in Step 2.

CP/67 ONLINE

The first message will be typed if the terminal is a 1052 or a 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, CP-67 must be notified that someone wishes to use the terminal. To do this, hit the attention key once. On the teletype 33 or 35, hit BREAK and then ERX-RLS.

4. The system will respond by either (1) unlocking the keyboard on a 2741 or 1052 or waiting for input on the teletype 33 or 35 or by (2) typing one of the following messages:

MAX. NO. OF USERS EXCEEDED
LOGGED OFF AT xx.xx.xx ON xx/xx/xx

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

SHUTDOWN IN PROGRESS
LOGGED OFF AT xx.xx.xx ON xx/xx/xx.

CP-67 is in the process of being taken down. When it is brought up again, return to Step 1.

5. Specify the multi-access system to which you wish to gain access by typing:

DIAL system

followed by a carriage return, where "system" is the userid of the multi-access system.

6. The system will then respond with one of the following messages:

...connected...

This message indicates a connection has been made between the terminal and the multi-access system and the terminal is now under control of that system. Only further responses will be those of the multi-access system, as the user can not get to CP-67 to issue console functions.

system ALL LINES BUSY
REPEAT PROCEDURE

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
REPEAT PROCEDURE

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

system LINES NOT READY
REPEAT PROCEDURE

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

system NO DIAL LINES
REPEAT PROCEDURE

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

Disconnecting

The 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.

or

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

or

(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 login in CP-67 as in Section 2.2.2 or to dial into a multi-access system again.

2.2.3 DIALING a Multi-Access System

This section describes the procedures which must be performed to connect a user to a system that provides multi-terminal facilities, such as APL or RAX. The process of placing a user into a multi-access system is referred to as "dialing". The system to be dialed into must be logged on to CP-67, as in Section 2.2.2, 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 multi-access system, he should log out of that system in the appropriate manner; when that multi-access 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 multi-access system.

Dialing

After the terminal has been readied for use as described in Section 2.2.1, the following procedure must be performed to gain access to a multi-access system. (Note that input may be entered in either upper or lower case. Upper case is used below to indicate words which must be typed as they are shown; lower case indicates fields whose contents may vary.)

1. First, 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, Step 1 must be repeated. Otherwise, a Data-Phone similar to that shown in Figure 2.2.0-C will be 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 will remain lighted as long as the terminal remains connected to the computer. If this light goes out at any point during the terminal session, Step 1 must be repeated.

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

```
CP/67 ONLINE-  xxxxxxxxxxx
xxxxxxxxxxx    CP/67 ONLINE
CP/67 ONLINE
```

The first message will be typed if the terminal is a 1052 or a 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 xxxxxxxxxxx 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, CP-67 must be notified that someone wishes to use the terminal. To do this, hit the attention key once. On the teletype 33 or 35, hit BREAK and then BRK-RLS.

4. The system will respond by either (1) unlocking the keyboard on a 2741 or 1052 or waiting for input on the teletype 33 or 35 or by (2) typing one of the following messages:

```
MAX. NO. OF USERS EXCEEDED
LOGGED OFF AT xx.xx.xx ON xx/xx/xx
```

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

```
SHUTDOWN IN PROGRESS
LOGGED OFF AT xx.xx.xx ON xx/xx/xx.
```

CP-67 is in the process of being taken down. When it is brought up again, return to Step 1.

5. Specify the multi-access system to which you wish to gain access by typing:

```
DIAL system
```

followed by a carriage return, where "system" is the userid of the multi-access system.

6. The system will then respond with one of the

following messages:

...connected...

This message indicates a connection has been made between the terminal and the multi-access system and the terminal is now under control of that system. Only further responses will be those of the multi-access system, as the user can not get to CP-67 to issue console functions.

system ALL LINES BUSY
REPEAT PROCEDURE

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
REPEAT PROCEDURE

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

system LINES NOT READY
REPEAT PROCEDURE

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

system NO DIAL LINES
REPEAT PROCEDURE

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

Disconnecting

The 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.

or

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

or

(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 login in CP-67 as in Section 2.2.2 or to dial into a multi-access system again.

2.2.4 GENERAL TYPING CONVENTIONS

The following typing conventions should be observed when entering input to the CP-67/CMS system from either a 2741, 1050, or a teletype 33 or 35 terminal:

a. Input may be entered in either upper or lower case.

b. 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- hand 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.

c. 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.

d. 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.

e. 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 - will be truncated to 130 characters. On the teletype 33, an input line can only contain a maximum of 72 characters.

f. An input line from the 2741 or teletype is terminated by hitting the RETURN key. 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 ECB special feature. If this special feature is available and the ECB panel switch is set to AUTO, input lines may be terminated by hitting the RETURN key.

g. Input lines can contain a number of logical input lines which are separated by line-end character (the # sign). Each call to read a line from the terminal will return a logical input line. Subsequent calls to read a line from the terminal will return the logical input line which was typed following the previous logical input line. For example, the single input line -


```
FORTRAN ABLE # $ ABLE
```

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

```
T # 1 /BUFT/ # C /FI/FFER/
```

will cause the characters BUFT to be located and changed to BUFFER. The line-end character can be changed with the LINEND command (See 3.5.1).

h. An output line on the 2741 and 1050 may be a maximum of 130 characters. Any line longer than 130 characters will cause 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 uparrow is printed at the end of the line to indicate continuation.

i. Illegal output characters will 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.

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

2.2.5 ATTENTION INTERRUPT

After making a phone connection with the computer and the message 'CP/67 Online' is printed, an attention causes the CP login procedure to begin or the dialing of a multi-access 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 the ATTN key) causing control to pass from CMS to CP, a subsequent attention will pass control back to CMS. If CMS previously had control and was reading a line from the terminal, i.e., the keyboard was unlocked, the CMS program will be restarted and the keyboard will be unlocked again. If CMS previously had control and was not reading a line from the terminal, i.e., the keyboard was locked, the interruption permits a single line of input to be entered and stacked. Stacked lines of input will be used on successive calls to read a line from the terminal until there are no more stacked lines. Input will then be 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, the ATTN key 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 will be ignored.

2.3.0 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 will be 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
Script Edit environment
Script Input environment
Echo environment

In addition to these eight specific environments, input may be entered to any other executing program which expects terminal input. These other input processing programs are grouped into a ninth, "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, Script, Edit, Script 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 360 console. Input to the CMS Command environment may be any CMS "command".

A certain number of the CMS commands cause environments of their own to be entered. These are the DEBUG, EDIT, SCRIPT, 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 and SCRIPT commands will cause either of two environments to be entered: If either command is issued for a file which already exists, the appropriate editing environment (Edit or Script Edit) will be entered, allowing the contents of the existing file to be examined and modified. If an EDIT or SCRIPT command is issued for a file which does not currently exist, the appropriate input environment (Input or Script Input) will be entered, allowing the file to be created. Both the Input and Script Input environments will accept any input typed at the terminal, and this input will become a part of the file being

created. The Echo environment will also accept any input line, but will repeat that line as output in order to test terminal transmission. Because no check is made to determine the acceptability of input to these three environments, lines which are acceptable are termed merely "input". The Debug, Edit, and Script Edit environments, on the other hand, accept only specific input lines which are referred to as "requests".

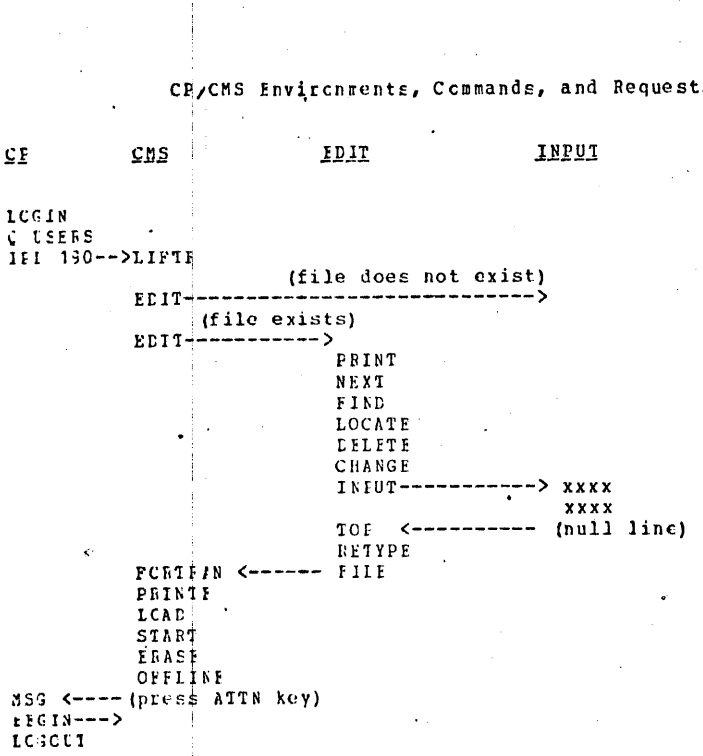
To verify at any time which environment the user is in, the RETURN key can be hit.

Various actions by the user will cause control to pass from one environment to another. These actions are summarized in Figure 2.3.0-A, and are specified in detail throughout this guide. Note that the attention key can always be used to transfer control to the CP environment from any of the other environments. Hitting the attention key while in the CP environment will cause 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, KC or KE) or the RT command, it is executed immediately and control will return 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 will return 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.

To (Entry) ...	CP Environment	CMS Command Environment	DEBUG Environment	Edit Environment	Input Environment	Script Edit Environment	Script Input Environment	Echo Environment	Program Environment
CP Environment	Completion of processing of non-transfer console functions	IPL CMS or IPL CMS console functions Attention key (CP entered previously from CMS Command)	EXTERNAL or BEGIN console functions Attention key (CP entered previously from DEBUG)	Attention key (CP entered previously from Edit)		Attention key (CP entered previously from Script Edit)	Attention key (CP entered previously from Script Input)	Attention key (CP entered previously from Echo)	Attention key (CP entered previously from Program)
CMS Command Environment	Attention key	Completion of processing of non-transfer command	DEBUG command	EDIT command (file exists)	EDIT command (file does not exist)	SCRIPT command (file exists)	SCRIPT command (file does not exist)	ECHO command	LOAD command with XEQ option LOAD or LOADMOD and START commands $ command
Debug Environment	Attention key RESTART request RETURN request (Debug entered by issuing DEBUG command)	Completion of processing of non-transfer request	None	Completion of processing of non-transfer request	INPUT request	None	None	None	GO request RETURN request (Debug entered from Program via SVC)
Edit Environment	Attention key QUIT or FILE request (Edit entered by issuing EDIT command)	None	None	Null line	Completion of processing of non-transfer input	None	None	None	QUIT or FILE request (Edit entered from Program via SVC)
Input Environment	Attention key	None	None	None	None	Completion of processing of non-transfer request	RNRT request	None	QUIT or FILE request (Script Edit entered from Program via SVC)
Script Edit Environment	Attention key QUIT or FILE request (Script Edit entered by issuing SCRIPT command)	None	None	None	None	Null line	Completion of processing of non-transfer input	None	None
Script Input Environment	Attention key	None	None	None	None	None	None	None	None
Echo Environment	Attention key RETURN request (Echo entered by issuing ECHO command)	None	None	None	None	None	None	Completion of processing of non-transfer input	RETURN request (Echo entered from Program via SVC)
Program Environment	Attention key Return on GPR 14	Program Interrupt Break point setting Call to DEBUG command via SVC	Call to EDIT command via SVC (file exists)	Call to EDIT command via SVC (file does not exist)	Call to EDIT command via SVC (file does not exist)	Call to SCRIPT command via SVC (file exists)	Call to SCRIPT command via SVC (file does not exist)	Call to ECHO command via SVC	Normal instruction execution

FIGURE 2.3.0-A Methods of Transfer Between Environments

CP/CMS Environments, Commands, and Requests



3.0.0 CMS COMMANDS

The CMS commands provide user facilties for file maintenance and manipulation, execution control, debugging, language processing, and various utility operations.

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 the number of characters of each command command name which serve to 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. "L", for example, is the abbreviation for the LISTF command, and "LO" for LOAD. Any number of additional characters beyond the minimum may be specified in the command name. For this reason, "LO", "LOA", and "LOAD" all identify the LOAD command whereas "LOADM" is the minimum abbreviation for the LOADMOD command. If the user desires to change the abbreviations or use synonyms, the SYN command can be used to create the command names.

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 the *Format* section of 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 # . 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 either 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 left-most input on the command line.

When a command is issued, the directories 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 directories, it is assumed to be in core, and control will be transferred to it directly by a BALR instruction. If not, a LOADMOD will be issued to bring the command program into core. In attempting to locate this program on disk, the user's permanent and temporary directories and the system directory are searched, in that order, 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 will be loaded into core and control will be transferred to it. If the MODULE file is not found, a check is made for abbreviations by checking for user-defined synonyms (see SYN) and other 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 performed again.

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, i.e., one whose command program does not reside in the CMS nucleus or transient-area and for which a core-image module cannot be located, will be ignored and the message "INVALID CMS COMMAND" will be typed at the terminal. Operand processing is handled by the individual command programs and these programs provide all messages dealing with command format.

A format and usage of each of the CMS commands are described in detail in the following sections. 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 will terminate command execution. All responses and error messages which may be issued appear in alphabetical order in Appendix C, with references to the section(s) where they are described further.

3.1.0 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 either the EDIT or SCRIPT EDIT commands for a disk file which does not currently exist will allow the specified file to be created from terminal input. EDIT creates card-image files; SCRIPT EDIT accepts any terminal input up to 120 characters in length. To create a disk file from card input, the OFFLINE READ or DISK LOAD commands may be used. OFFLINE READ accepts card input in any format. DISK LOAD accepts only input in the CMS card format. The DUMPREST and TAPE LOAD commands create disk files from magnetic tape input. DUMPREST restores the entire tape contents to the user's permanent disk area, whereas TAPE LOAD may be used to transfer one or more specific files to disk. The MACLIB GEN and TXTLIB GEN commands create macro and text library files on disk using the contents of existing disk files. The TXTLIB PRINT command will create a file containing the entry points of the specified text library. Other commands which create disk files from disk input are discussed under File Manipulation.

Card files are created in CMS by requesting that the contents of disk files be punched out. The OFFLINE PUNCH command will punch out any disk file whose records are 80 characters or less in length. DISK DUMP will punch any disk file onto cards, but will first convert its contents into CMS card format.

Two of the commands, TAPE DUMP and DUMPREST, cause tape files to be created from disk files. TAPE DUMP copies single disk file to tape. DUMPREST will copy the contents of the user's entire permanent disk area to tape.

File Maintenance

Several commands provide facilities for maintaining disk files. UPDATE, EDIT, and SCRIPT 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. Both EDIT and SCRIPT EDIT allow 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. Additions to existing macro and text library files are made using the MACLIB ADD and TXTLIB ADD commands, respectively. A file or group of files can be deleted from disk by issuing the ERASE command. The FINIS command must be issued by any user program which accesses more than eight disk files, and 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, spliting, and listing disk files. To copy a disk file, the EDIT or COMBINE commands can be used. COMBINE also will create a new disk file from the contents of two or more existing disk files and may be used to transfer a file between the permanent and temporary disk areas. The SPLIT command creates a new disk file composed of the specified portions of an existing disk file or files.

LISTF, PRINTF, MACLIB LIST, and TXTLIB LIST cause file information to be typed at the terminal. The LISTF command will type out 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. The MACLIB LIST and TXTLIB LIST commands type out the directories of the specified library files.

To print the contents of a disk file on the offline printer, the OFFLINE PRINT or OFFLINE PRINTCC commands can be issued. OFFLINE PRINT will print the file with single spacing and CMS headings, whereas the PRINTCC version will use the first character of each record as a printer carriage control character.

3.1.1 ALTER

Purpose:

The ALTER command changes the name, type, or mode number of user files.

Format:

ALTER or AL	ofn	oft	ofm	nfn	nft	nfm
			*	*	*	*

ofn oft ofm is the original filename, filetype, and filemode.

nfn nft nfm is the new filename, filetype, and filemode.

Usage:

The name, type, or mode number of any file on the permanent or temporary disk may be changed with the ALTER command. All fields of the command must be specified. Any field not changed may be represented by an * in the new file designation. If the mode number is not changed, the original mode may be represented by an *.

Notes:

a. ALTER does not move files between disks. The mode letter, P or T, may not be changed. See the COMBINE command.

b. ALTER may not be used for files on the system (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-or-write.

b. ALTER JBS LISTING * JLEVEL3 * *

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

Error Messages:

E(00001) OLD FILE NOT FOUND
The file to be re-designated is not in the file directory. Check the spelling of the filename and type.

E(00002) NEW 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 desired, use the COMBINE command.

E(00003) OLD MODE ILLEGAL
The original mode specified does not begin with P or T, or does not end with a number from 1 to 6.

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

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

E(00006) NEW MODE ILLEGAL
The new mode specified does not begin with P or T, or the mode number is not between 1 and 6.

E(00007) INSUFFICIENT PARAMETERS
A name, type, and mode were not specified for both files.

3.1.2 CLOSIO

Purpose:

CLOSIO notifies CP/67 that I/O operations to an offline unit record device are complete.

Format:

```
{CLOSIO}  [READER] [PRINTER] [PUNCH]
{CLO   }
```

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 including unit record I/O routines terminate abnormally or do not include a call to CLOSIO.

CLOSIO notifies CP/67 of an end-of-file condition for the devices specified. 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. Cardreader input files are erased.

All CMS unit record commands (OFFLINE, DISK, DUMP), whether issued from the terminal or called from within a program, have already performed a CLOSIO when control is returned to the issuer.

Notes:

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

b. CP/67 assumes CLOSIO for all unit record devices when the user logs out.

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

Responses:

None.

Example:

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

Error Messages:

None.

3.1.2CLOSIO

Additional Notes:

1. CLOSIO is generally called to signal the end of an offline operation by sending an end-of-file condition to CP. When CP detects the end-of-file condition, the disk spooling areas are closed enabling the printer or card punch files to be queued for actual output.

   If it is desired to insure that a number of offline punch or print files are output together, the CLOSIO function can be disabled by the following command:

   CLOSIO {PUNCH / PRINTER} OFF

   After a number of offline punch or print files have been output, the CLOSIO function can be re-enabled by the following command:

   CLOSIO {PUNCH / PRINTER} ON

   In this way offline output can be collected before it is actually output to the physical device.

3.1.3 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 / C}	nfn nft nfm ofn1 oft1 ofm1 ... ofnN oftN ofmN

nfn nft 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 will be created on the permanent or temporary 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 "(TEMP) (FILE) mm", where "mm" is the specified mode of the output file, is created. 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 permanent disk, containing 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 permanent disk file JOBS is made on the temporary 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) CORRECT FORM IS: COMBINE N1 T1 M1 N2 T2 M2 ...
WHERE N1, T1, M1 ARE THE NAME, TYPE, AND MODE OF THE FILE TO BE CREATED.
AND N2 T2 M2,ETC. ARE THE FILES TO BE COMBINED.
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 SYSTEM DISK ILLEGAL.
The system (SY) disk is read-only.

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

3.1.4 DISK

Purpose:

The DISK command punches specified permanent disk files on cards in a special format, and can reload these card files onto disk storage.

Format:

```
{DISK}  {DUMP  filename filetype filemode}
{DI  }  {LOAD                            }
```

DUMP — specifies 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 filemode must be specified. Only one file is punched out by a command. The file may have either fixed or variable length records, but must be on the permanent (mode P) disk. After all data is transferred, and 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 will read any number of deck 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 those in the card stream is erased and replaced.

Files to be loaded by a DISK LOAD command must be read by CP67 as a single deck before the command is issued. An identification card with CP67USERID 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 CP67 identification card containing the USERID. This card must be removed before reading the deck.

Responses:

a. DISK DUMP gives no response except the Ready message.

b. filename filetype filemode LOADED.

DISK LOAD gives this response for each file encountered in the input deck. The Ready message is typed after transfer is completed.

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 CP67, 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 LOADED.
```

The Ready message will then be 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 CP67. 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. This message should never occur under CP67. 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 CP67 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 re-read before retrying the command. No permanent file has been created.

E(00006) END CARD MISSING FROM DISK LOAD DECK.

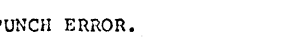
Physical end-of-data for a DISK LOAD was encountered without reading the end-of-file card ('N' in column 5). The file created on disk does not have the correct designation. It may be accessed under (DISK) (TFILE) P3. Note that the mode is delete-after-reading, so an ALTER or COMBINE must be issued before inspecting the file.

3.1.5 DUMPREST

Purpose:

DUMPREST dumps the contents of the user's permanent disk onto tape, and can reload a disk from such a tape.

Format:



A message is typed requesting one of the following replies:

{DUMP / D}	specifies the contents of the permanent disk are to be copied to tape.
{RESTORE / R}	specifies the permanent disk 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 a track 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:

a. RESTORE may not be valid if the location of the user's virtual disk 191 has been redefined since the DUMP operation.

Responses:

a. 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.

b. 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 10 times. If no recovery is made, Response (b) is typed.

3.1.6 EDIT

Purpose:

EDIT has two purposes: (1) to create card image files on the permanent disk and (2) to make changes to existing card-image files on the permanent or temporary disk.

Format:

EDIT	filename	filetype
E		

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 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.

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.

For files that have a filetype of ASP360, COPY, UPDATE or SYSIN, each input line is truncated after column 72. In the Input environment, if such a line contains a non-blank character in column 72, the message "STATEMENT CONTINUES:" will be typed. For files that have the filetype LISTING, the truncation column is column 132. For files that have a filetype of MEMO, the truncation column is column 80. For all 80 character files that have a filetype other than ASP360, SYSIN, COPY, UPDATE or MEMO, each input line is truncated after column 72. Files with record length other than 80 or 132, will be truncated at the full line length.

Normally, an identifier is placed in columns 73-80 of each card image. The identifier consists of a three character identification and a five digit sequence number. The identification is taken from the first three characters of the filename; 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 in the Edit invironment.

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 other identifier assigned to each card image. A file whose filetype is SCRIPT, LISTING or MEMO automatically defaults to no serialization.

Internal or logical tab settings indicate a column position that defines the beginning of a field within a card image. 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 indicate the column of the card image in which input is to begin.

Filetype Specified in EDIT Command	Assumed Tab Settings
FORTRAN	1,7,10,15,20,25,30
SYSIN	1,10,16,31,36,41,46,51,56,72
ASP360	Same as SYSIN
COPY	Same as SYSIN
FLOW	Same as SYSIN
UPDATE	Same as SYSIN
PLI	2,10,15,20,25,30,35,40, 45,50,55,60
SPL1	1,7,17,30,40,50,60
SNOBOL	1,10,20,25,30,35,40,45, 50,55,60
EXEC	1,5,8,17,27,31
REPS	2,7,15,24,25,30,73
TEXT	Same as REPS
default settings	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.

There is a character used in conjunction with the logical tab settings and it is called the logical tab character. This character causes blanks to be inserted into the card image from the column position at which this character is input up to 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 will be inserted as the first character of the next field. The standard logical tab character is the exclamation point (!). The logical tab character can be redefined by the TABDEF request from the Edit environment to allow the character ! 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 when the tab key is depressed; whereas, the logical tab character prints when the character is depressed and the typing element does not move to the next physical tab stop.

There is a character used to backspace one column position in a card image and it is called a logical backspace character. For "n" logical backspace characters, "n" column positions are backspaced in the card image 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 will not be overlaid. The backspacing is performed for the column positions of a card image and not for the characters of an input line.

The standard logical backspace character is the percent character (%). The logical backspace character can be redefined by the BACKSPACE request from the Edit environment to allow the character % to be used as a normal input character. The logical backspace character should not be confused with the physical backspace key on the terminal. The physical backspace key moves the typing element back one position and generates a valid input character that takes up one column in the card image for each time the key is depressed. The character generated by the backspace key does not print when entered on the terminal and does not print on the offline printer but it backspaces the typing element one position per character when the card image is printed out at the terminal. Thus, the backspace key allows underscoring and overprinting at the terminal. The logical backspace character prints only when entered and does not take up a column in the card image; it logically backspaces one column in the card image.

The Edit environment is entered from the Input environment by typing a null line, i.e., 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; this allows recursive modifications to be made to a file.

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 card image 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.

There are two modes in which to operate in the Edit environment: "verify" mode and "brief" mode. Verify mode is the normal mode. Verify mode is the normal mode and causes an automatic typeout of each line that has been changed or searched for as the result of a request. The brief mode does not type out the specified lines, and thus the user must issue a PRINT request to get the typeout if it is desired.

The input environment is entered from the Edit environment by typing the INPUT request, and a carriage return.

Notes:

a. New files created by the EDIT command are written on the permanent disk with mode P1.

b. Existing files being edited can be read in from either the permanent disk or the temporary disk. If the FILE or SAVE request is issued, the final copy of the file is written out to the same disk with the same filemode as it previously existed.

c. The final copy of a file being edited or the first copy of a newly created file will not be permanently written on disk until the FILE or SAVE request is issued.

d. 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. When the Edit environment is entered from the Input environment, the pointer is positioned to the last input line typed by the user. When the Input environment is entered from the Edit environment, the pointer is positioned at the last line edited by the user. When the EDIT command begins in the Edit environment, the pointer is positioned before the first line in the file.

e. A null line is automatically placed in front of the first line of a file being created or edited 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. A null line is also placed after the last line of the file to permit additions at the end. When an end-of-file condition occurs, the pointer is positioned at this null line after the last line. The two null lines never get written on disk.

f. If the end-of-file is reached by an EDIT request, a message is typed out that states the request that caused the end-of-file to be reached and the pointer is positioned after the last line of the file.

g. 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.

h. To insert a blank line in a file, type at least one space and hit carriage return.

i. If a new file is being created, the Input environment is entered first. If no serialization is desired or it is desired to input 80 characters per card image, the Edit environment must be entered and a SERIAL (NO) request issued before any card images are created.

j. If a file already exists with no identifiers, the SERIAL (NO) request must be given each time the EDIT command is issued in order to maintain the data that exists in columns 73-80 of the file.

k. Logical tab settings that 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.

l. If the logical tab character is redefined, the TABDEF request must be issued each time the EDIT command is issued in order to use the same redefined logical tab character.

m. If the logical backspace character is redefined, the BACKSPACE request must be issued each time the EDIT command is issued in order to use the same redefined backspace character.

n. If extensive input and/or changes are being made to a file, it is a good time-sharing practice to make few additions and/or changes at a time, issue the FILE or 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.

o. EDIT uses an intermediate work file during execution which has the identifier (INPUT1) FILE. This work file is automatically allocated on the user's permanent disk with a filemode of P1 if the file being edited does not exist. If the file being edited does exist, the work files are allocated with the same modes as the file being edited. If the work file exists when EDIT is entered, the message '"(INPUT1) FILE P1" EXISTS. ALTER TO SAVE LAST FILE EDITED OR ERASE.' The size of the work file will be the size of the file being edited plus changes and additions. When EDIT is terminated with either the FILE or QUIT request, the work file is erased.

p. If previous EDIT terminated abnormally, the current changes or the newly created file may not be completely lost. After CMS has been initialized, issue a LISTF to verify whether (INPUT1) FILE P1 exists. If it does, ALTER its filename and filetype. Then issue the EDIT command for that ALTERed file. Issuing EDIT before ALTERing the filename and filetype, results in the message in o. (above) being typed. EDIT will then terminate with an error code.

q. If the file being edited has logical records larger than 80 characters, each record will be truncated to its line length when read. When the FILE or SAVE request is issued, the full logical record will be written on the disk.

r. All input to the file being edited will be converted from lower case to upper case unless a filetype of MEMO or SCRIPT has been specified. A MEMO file or SCRIPT file accepts the characters as is without translation.

s. If a filetype of REPS or TEXT is EDITed, tab settings will be assumed and a "12-2-9" character will be inserted into column 1 if it appears blank.

t. 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.

Responses:

NEW FILE

The specified file does not exist, thus the Input environment will be 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.

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. Edit request may now be issued.

TRUNCATED

The input line was too long. If the request "SERIAL (NO)" had been previously issued or the filetype was MEMO, the input line was truncated after 80 columns; if the filetype was ASP360 or SYSIN, it was truncated after column 71; otherwise, it was truncated after column 72. Continue typing more input.

INVALID EDIT REQUEST: xxx...xxx

The invalid request xxx...xxx was issued to the Edit environment. Check to see if too many arguements were specified with the request or if a non-numeric field was specified for a numeric argument. Correct the request and issue it again.

EOF:

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

STATEMENT CONTINUES:

A non-blank character has been entered as Input into column 72 of a SYSIN, ASP360, UPDATE or COPY file. If this was intentional, proceed. Otherwise, leave the Input environment and correct the line.

Error Message:

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

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.

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. If the tab key or the logical tab character is used and the request has "line" as the operand, the "line" will be placed in the card image as if the tab key or logical tab character was the first character in "line".

f. If the end-of-file is reached by an EDIT request, a message is typed out that states the request that caused the end-of-file to be reached and the pointer is positioned after the last line of the file.

g. 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.

h. To insert a blank line in a file, type at least one space and hit carriage return.

i. If a new file is being created, the Input environment is entered first. If no serialization is desired or it is desired to input 80 characters per card image, the Edit environment must be entered and a SERIAL (NO) request issued before any card images are created.

j. If a file already exists with no identifiers, the SERIAL (NO) request must be given each time the EDIT command is issued in order to maintain the data that exists in columns 73-80 of the file.

k. Logical tab settings that 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.

l. If the logical tab character is redefined, the TABDEF request must be issued each time the EDIT command is issued in order to use the same redefined logical tab character.

m. If the logical backspace character is redefined, the BACKSPACE request must be issued each time the EDIT command is issued in order to use the same redefined logical backspace character.

n. If extensive input and/or changes are being made to a file, it is a good time-sharing practice to make few additions and/or changes at a time, issue the FILE or 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.

o. EDIT uses two intermediate work files during execution which have the identifiers INPUT FILE and INPUT1 FILE. These work files are automatically allocated on the user's permanent disk with a filemode of P1 if the file being edited does not exist. If the file being edited does exist, the work files are allocated with the same modes as the file being edited. If files already exist with the same identifiers as the work files, they are erased. The size of each work file will be the size of the file being edited plus changes and additions. When EDIT is terminated with either the FILE or QUIT request, the work files are erased.

p. If the system goes down during the EDIT command, the current changes or the newly created file may not be completely lost. After CMS has been initialized, issue a LISTF and note the size of the two work files (if both exist). If both work files do exist, ALTER the filename and filetype of the larger of the work files. If only one work file exists, ALTER its filename and filetype. Then issue the EDIT command for that ALTERed file. Do not issue EDIT before ALTERing the filename and filetype, as EDIT erases existing work files.

q. EDIT performs input/output with 80 character logical records. If the file being edited has logical records larger than 80 characters, each record will be truncated to 80 characters when read. When the FILE or SAVE request is issued, 80 character logical records will be written on the disk.

r. All input to the file being edited will be converted from lower case to upper case unless a filetype of MEMO has been specified. A MEMO file accepts the characters as is without translation.

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

t. 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.

Responses:

NO PRIMARY NAME SPECIFIED.
When the EDIT command was issued, only the filetype was specified. The Input environment will be entered and all subsequent input lines will be accepted as input to file. The user must specify a filename with the FILE or SAVE request when he is ready to save the file.

FILE DOES NOT EXIST; WILL BE CREATED.
The specified file does not exist, thus the Input environment will be 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.

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. Edit request may now be issued.

TRUNCATED.
The input line was too long. If the request "SERIAL (NO)" had been previously issued or the filetype was MEMO, the input line was truncated after 80 columns; if the filetype was ASP360 or SYSIN, it was truncated after column 71; otherwise, it was truncated after column 72. Continue typing more input.

INVALID REQUEST: xxx...xxx
The invalid request xxx...xxx was issued to the Edit environment. Check to see if too many arguments were specified with the request or if a non-numeric field was specified for a numeric argument. Correct the request and issue it again.

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

Error Message:

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

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.

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. If the tab key or the logical tab character is used and the request has "line" as the operand, the "line" will be placed in the card image as if the tab key or logical tab character was the first character in "line".

The tab settings which are discussed are internal or logical tab settings, not the external or physical tab settings on the terminal.

The requests are listed below and are described on the indicated pages:

3.1.6 EDIT

Additional Notes:

1. The truncation columns used for EDIT requests are as follows:

   INPUT and REPLACE - column 72 for SYSIN, ASP360, COPY and UPDATE, with continuation punch checking
   Column 72 for FORTRAN, PLI, SPL1, SNOBOL AED, FLOW, EXEC, MAD, AS1130, and REPS files
   column 132 for SCRIPT, and listing 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

2. An asterisk (*) can be used to mean "toEOF" or "GLOBAL" in a change request as follows:

   C /a/b/**

3. A filetype of MEMO is used to input 80 character card images containing both upper and lower case letters.

4. The EDIT command allows processing of SCRIPT files in a form compatible with the SCRIPT PRINT command. If the filetype is SCRIPT, all input lines introduced under INPUT and strings introduced through use of CHANGE, OVERLAY, RETYPE, and INSERT are interpreted without converting lower-case characters to upper-case. The character-de;ete and line-delete symbols have the usual effect; all other characters are stored without modification.

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

5. Input lines containing a backspace character are converted into canonical form such that only one backspace follows any character and precedes the character which will overprint the character preceding the backspace.

6. EDIT recognizes the file type of MEMO, SCRIPT, LISTING, COPY, SYSIN, UPDATE, AED, MAD, and AS1130, in addition to those mentioned on page 3.1.6-2.

7. If no line is specified with the RETYPE request, the current line is deleted and the INPUT environment is entered.

   The assumed tab settings are as follows:

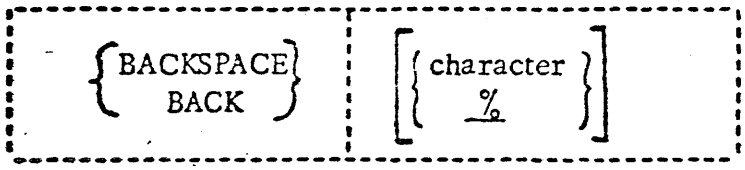
MEMO	default settings
SCRIPT	default settings
LISTING	default settings
COPY	same as SYSIN
UPDATE	same as SYSIN
AED	1, 10, 15, 20, 25, 30, 35, 40, 45
MAD	same as AED
AS1130	21, 27, 32, 35, 45, 50, 55, 60

8. 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, setting in continuation characters, searching on the serialization field, etc.

9. Serialization is suppressed for all filetypes except FORTRAN, PLI, SYSIN, UPDATE, COPY, SPL1, SNOBOL, AED, FLOW, EXEC, MAD, AS1130, and REPS. If serialization is turned on for other filetypes, and either ZONE 1 71 or SERIAL id is not issued during subsequent Edits of the file, serialization characters may be inadvertantly shifted into column 72 or columns 73-80 may be overwritten.

BACKSPACE

Format:



character is any valid character. The default **character** is " %."

Usage:

The BACKSPACE request defines the character to be used as the logical backspace **character** in the Input environment. 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 backspace 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.

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.

Responses:

The keyboard is unlocked.

Examples:

a. BACK $

The character $ will be defined as the logical backspace character. The character % may now be used as a normal input character.

3.1.6 EDIT

Additional Notes:

1. The truncation columns used for EDIT requests are as follows:

   INPUT and REPLACE - column 71 for SYSIN, ASP 360, COPY, and UPDATE
   column 72 for FORTRAN, PLI, SPL1, SNOBOL, AED, FLOW, EXEC, MAD, AS1130, and REPS files
   column 132 for SCRIPT, and LISTING 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

2. An asterisk (*) can be used to mean "to EOF" or "GLOBAL" in a change request as follows:

   C /a/b/ * *

3. A filetype of MEMO is used to input 80 character card images containing both upper and lower case letters.

4. The EDIT command allows processing of SCRIPT files in a form compatible with the SCRIPT PRINT command. If the filetype is SCRIPT, all input lines introduced under INPUT and strings introduced through use of CHANGE, OVERLAY, RETYPE, and INSERT are interpreted without converting lower-case characters to upper-case. The character-delete and line-delete symbols have the usual effect; all other characters are stored without modification.

   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.

5. Input lines containing a backspace character are converted into canonical form such that only one backspace follows any character

and precedes the character which will overprint the character preceding the backspace.

6. The logical tab character is defined as the # sign. The # sign is also the logical carriage return character or line-end character and allows for multiple commands per line of input. The logical line-end character takes precedence over the tab character, therefore, the LINEND command must be issued to allow the # to be used as the tab character.

7. EDIT recognizes the file type of MEMO, SCRIPT, LISTING, COPY, SYSIN, UPDATE, AED, MAD, and AS1130, in addition to those mentioned on page 3.1.6 - 2.

8. If no line is specified with the RETYPE request, the current line is deleted and the INPUT environment is entered.

   The assumed tab settings are as follows:

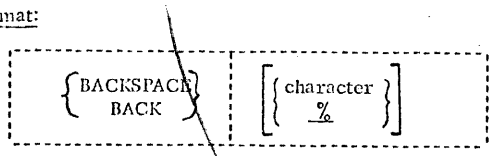
MEMO	default settings
SCRIPT	default settings
LISTING	default settings
COPY	same as SYSIN
UPDATE	same as SYSIN
AED	1, 10, 15, 20, 25, 30, 35, 40, 45
MAD	same as AED
AS1130	21, 27, 32, 35, 45, 50, 55, 60

9. 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, setting in continuation characters, searching on the serialization field, etc.

10. Serialization is suppressed for all filetypes except FORTRAN, PLI, SYSIN, UPDATE, COPY, SPL1, SNOBOL, AED, FLOW, EXEC, MAD, AS1130, and REPS. If serialization is turned on for other filetypes, and either ZONE 1 71 or SERIAL id is not issued during subsequent Edits of the file, serialization characters may be inadvertantly shifted into column 72 or columns 73-80 may be overwritten.

BACKSPACE

Format:



character is any valid character. The default character is " %."

Usage:

The BACKSPACE request defines the character to be used as the logical backspace character in the Input environment. 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 backspace 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.

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:         ! #%5'page')
card image in file:       5'PAGE')
```

This places "5'PAGE')" beginning in column 6 of the card image.

Responses:

The keyboard is unlocked.

Examples:

a. BACK $
The character $ will be defined as the logical backspace character. The character % may now be used as a normal input character.

BACKUP

Format:

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

Usage:

The BACKUP 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 will function 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.

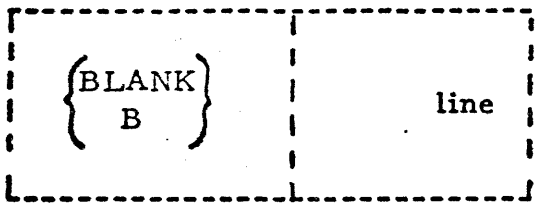
Examples:

a. U 9

This request repositions the pointer 9 lines before the current line.

BLANK

Format:



line is any valid input line

Usage:

The BLANK request places blanks in the current line wherever non-blank characters occur in "line".

The tab key and 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.
Brief Mode: The keyboard is unlocked.

EOF:

The end of file was reached by the request. For BLANK to reach the end of file, a REPEAT request had to be issued prior to the BLANK.

Example:

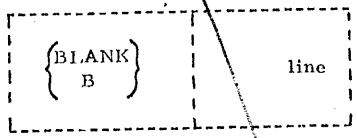
a. BLANK AAAAAA A

	column 1 6 8
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.

BLANK

Format:



line is any valid input line

Usage:

The BLANK request places blanks in the current line wherever non-blank characters occur in "line".

The tab key and 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.
Brief Mode: The keyboard is unlocked.

EOF REACHED BY: xxx...xxx

The end of file was reached by the request xxx...xxx. For BLANK to reach the end of file, a REPEAT request had to be issued prior to the BLANK.

Example:

a. BLANK AAAAAA A

```
                                  column
                                  1     6 8
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

Format:

Usage:

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

Example:

a. BO

The pointer is positioned at the bottom of the file.

BRIEF

Format:

{BRIEF / BR}

Usage:

The BRIEF request turns off the verify mode (see the VERIFY request) and turns on the brief mode in the Edit environment. In the brief mode lines that are changed in the file will not be typed out automatically. The requests that are affected by BRIEF are BACKUP, BLANK, CHANGE, FIND, LOCATE, and OVERLAY.

Example:

a. BR
This request turns on the brief mode.

CHANGE

Format:

{CHANGE / C} /stringl/string2/ [{n / * / 1}] [G]

/	is any unique delimiting character that does not appear in stringl or string2.
stringl	is the group of characters to be replaced.
string2	is the group of characters that replace stringl.
n	specifies the number of lines to be searched for stringl. The default is 1 line.
G	signifies that the change is to be applied to every occurrence of stringl in the lines.
*	signifies that all lines from the current line to the end-of-file are to be searched.

Usage:

The CHANGE request replaces the occurrence of stringl in "n" lines by string 2. If G is specified, every occurrence of stringl in "n" or "*" lines is changed; if G is not specified, only the first occurrence of stringl is changed. If neither "n" nor "*" is specified, only the current line will be searched for string 1.

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

Stringl and string2 can be of different lengths. Each of the "n" or "*" lines will be expanded or compressed accordingly.

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

CHANGE

Format:

CHANGE C	/string1/string2/ [{ n * 1 }] [{ GLOBAL G * }]

/	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 that replace string1.
n	specifies the number of lines to be searched for string1. The default is 1 line.
n or *	signifies that n or all lines from the current line to the end-of-file are to searched.
GLOBAL or G or *	signifies that the change is to be applied to every occurrence of string1 in the lines.

Usage:

The CHANGE request replaces the occurrence of string1 in "n" lines by string 2. If G 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 will be searched for string 1.

If the occurrence of string1 is not found, the line(s) is 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 will be 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 string 1 from the current line to the end will be changed. The logical backspace character and the logical tab character cannot be used in string1 or string2, as they will be considered normal input characters in either string.

c. In a SYSIN or ASP360 file, only 71 characters of an 80 character line can be changed. That is, only the first 71 characters are scanned for string1. In a MEMO file or one that has requested no serialization, all 80 characters can be changed. In all other filetypes, only 72 characters can be changed.

Responses:

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

EOF:
The end of file was reached by the request xxx...xxx. 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. C /ALPHA/DELTA/

Column
1 7 10 15

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.

b. C *ALPHA*DELTA* 1 G

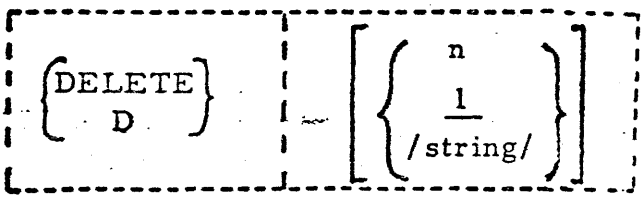
Column
1 7 10 15

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.

DELETE

Format:



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

/ is any unique delimiting character that does not appear in the string.

string specifies the character string to be located before deleting lines from the current line to the line containing "string."

Usage:

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, only the current line is deleted.

If /string/ is specified, DELETE first issues a "LOCATE /string/" command. If "string" exists, all lines from the current line to, but not including, the line containing "string" are deleted. The pointer will be positioned at the line containing "string."

If "string" is not located, no lines will be deleted, the end-of-file message is typed out, and the pointer is positioned after the last line.

If neither "n" nor "string" is specified, only the current line is deleted.

Response:

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.

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 string 1 from the current line to the end will be changed. The logical backspace character and the logical tab character cannot be used in string1 or string2, as they will be considered normal input characters in either string.

c. In a SYSIN or ASP360 file, only 71 characters of an 80 character line can be changed. That is, only the first 71 characters are scanned for string1. In a MEMO file or one that has requested no serialization, all 80 characters can be changed. In all other filetypes, only 72 characters can be changed.

Responses:

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

EOF REACHED BY: xxx...xxx
The end of file was reached by the request xxx...xxx. 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. C /ALPHA/DELTA/

```
                                Column
                          1     7 10   15
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.

b. C *ALPHA*DELTA* 1 G

```
                                Column
                          1     7 10   15
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.

DELETE

Format:

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

/ — is any unique delimiting character that does not appear in the string.

string — specifies the character string to be located before deleting lines from the current line to the line containing "string."

Usage:

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, only the current line is deleted.

If /string/ is specified, DELETE first issues a "LOCATE /string/" command. If "string" exists, all lines from the current line to, but not including, the line containing "string" are deleted. The pointer will be positioned at the line containing "string."

If "string" is not located, no lines will be deleted, the end-of-file message is typed out, and the pointer is positioned after the last line.

If neither "n" nor "string" is specified, only the current line is deleted.

Response:

The keyboard is unlocked.

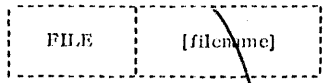
EOF REACHED BY: xxx...xxx
The end of file was reached by the request xxx...xxx. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

a. D 5
This request deletes the current line plus the next four lines.

b. D /SUBROUTINE WRITEX/
This request scans for "SUBROUTINE WRITEX" from the current line to end-of-file. Once the string is located, all lines from the current line to the line containing "SUBROUTINE WRITEX" are deleted. If the string is not located, no lines are deleted.

Format:



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 on 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 on the permanent disk with mode P1. 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. If "filename" is not specified at either time, a message is typed out.

After the file has been written on disk, control is returned to the CMS environment.

Responses:

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

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

NO PRIMARY NAME SPECIFIED - RETRY
A filename was not specified for this file, so it cannot be stored. Re-issue the FILE request and give the filename.

Examples:

a. FILE

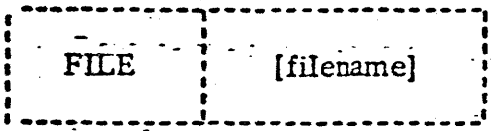
```
request:              file
response from CMS:    R; T=0.85
```

This request will write the latest copy of the file on disk.

3/9/70

FILE

Format:



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 on 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 on the permanent disk with mode P1. 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. If "filename" is not specified at either time, a message is typed out.

After the file has been written on disk, control is returned to the CMS environment.

Responses:

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

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

Examples:

a. FILE

This request will write the latest copy of the file on disk.

Examples: (Cont.)

b. FILE RECALC

This request will write the latest copy of the file on disk and RECALC will be its filename.

FIND

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 non-blank 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 card 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 card image are the ones specified by non-blank 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.

The "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 keyboard is unlocked.
Brief Mode: The keyboard is unlocked

EOF:
The end of file was reached by the request xxx...xxx. 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 searches for "90" in columns 1 and 2. The first line found is typed out in the verify mode.

b. FIND # # SUMX

	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.

Examples: (Cont.)

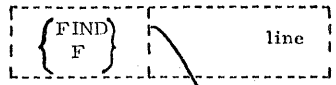
b. FILE RECALC

request: file recalc
response from CMS: R; T=0.63

This request will write the latest copy of the file on disk and RECALC will be its filename.

FIND

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 non-blank 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 card 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 card image are the ones specified by non-blank 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.

The "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.

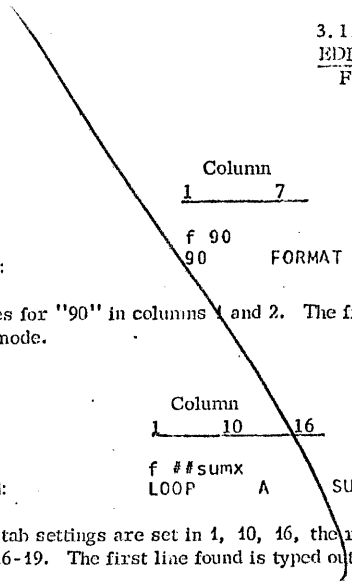
Responses:

Verify Mode: The keyboard is unlocked.
Brief Mode: The keyboard is unlocked

EOF REACHED BY: xxx...xxx
The end of file was reached by the request xxx...xxx. 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



request:
line found:

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

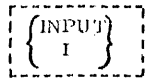
b. FIND # # SUMX

request:
line found:

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

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 last line of input or 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.

Responses:

INPUT:

The Input environment is entered.

Examples:

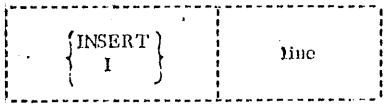
a. I

request: i
response: INPUT:

INPUT enters the Input environment and allows the user to create more lines of input to his file.

INSERT

Format:



line is the exact input line to be inserted into the file. It can obtain 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	
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.

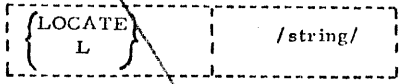
c. INSERT

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

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

LOCATE

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 card images.

Usage:

LOCATE scans all 80 characters in each card image for the character 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 the "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, as all 80 characters are scanned. The logical tab character and the tab key cannot be used to generate blanks in the string. The logical tab character and the logical backspace character can be used as a normal input character in "string".

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

Responses:

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

EOF REACHED BY: xxx...xxx
The end of file was reached by the request xxx...xxx. 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 for "FORMAT" in columns 1-80 of each line. 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 columns 1-80 of each line. In the verify mode the first line found is typed out.

LOCATE

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 card images.

Usage:

LOCATE scans all 80 characters in each card image for the character 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 the "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, as all 80 characters are scanned. The logical tab character and the tab key cannot be used to generate blanks in the string. The logical tab character and the logical backspace character can be used as a normal input character in "string".

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

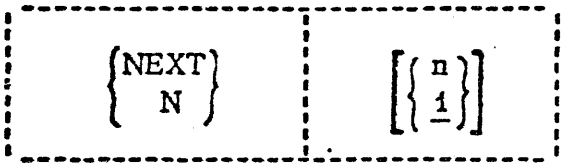
Responses:

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

EOF;
The end of file was reached by the request xxx...xxx. 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.

NEXT

Format:



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

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:

The keyboard is unlocked.

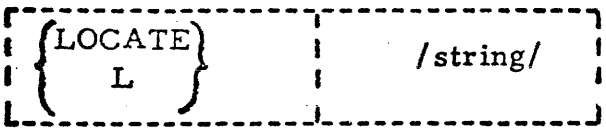
EOF:
The end of file was reached by the request xxx...xxx. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

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

LOCATE

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 card images.

Usage:

LOCATE scans all 80 characters in each card image for the character 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 the "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, as all 80 characters are scanned. The logical tab character and the tab key cannot be used to generate blanks in the string. The logical tab character and the logical backspace character can be used as a normal input character in "string".

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

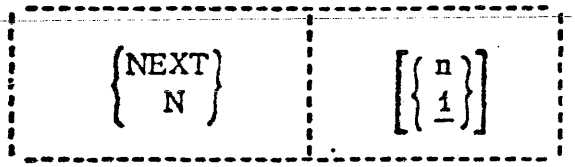
Responses:

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

EOF;
The end of file was reached by the request xxx...xxx. 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.

NEXT

Format:



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

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:

The keyboard is unlocked.

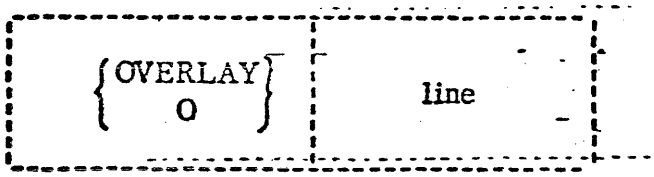
EOF:
The end of file was reached by the request xxx...xxx. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

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

OVERLAY

Format:



line is an input line that overlays the current line.

Usage:

This request takes the non-blank 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 will be considered as part of "line". The logical tab character, the tab key, and the logical backspace character can be used in specifying "line".

Responses:

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

EOF;
The end of file was reached by the request xxx...xxx. For OVERLAY to reach the end of file, the REPEAT request had to be issued prior to 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 non-blank characters in "line". The letter "b" is used here to indicate a single space.

LOCATE

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 card images.

Usage:

LOCATE scans all 80 characters in each card image for the character 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 the "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, as all 80 characters are scanned. The logical tab character and the tab key cannot be used to generate blanks in the string. The logical tab character and the logical backspace character can be used as a normal input character in "string".

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

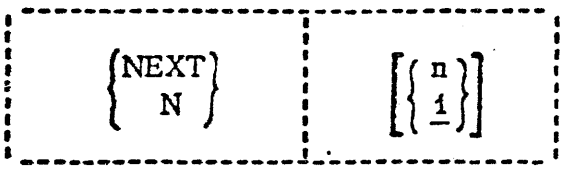
Responses:

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

EOF;
The end of file was reached by the request xxx...xxx. 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.

NEXT

Format:



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

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:

The keyboard is unlocked.

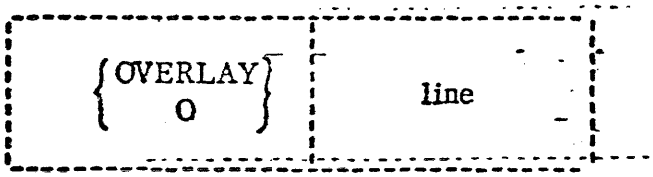
EOF:
The end of file was reached by the request xxx...xxx. To position the pointer at the top of the file, a TOP request must be issued.

Examples:

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

OVERLAY

Format:



line is an input line that overlays the current line.

Usage:

This request takes the non-blank 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 will be considered as part of "line". The logical tab character, the tab key, and the logical backspace character can be used in specifying "line".

Responses:

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

EOF:
The end of file was reached by the request xxx...xxx. For OVERLAY to reach the end of file, the REPEAT request had to be issued prior to 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 non-blank 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 non-blank characters in "line". The letter "b" is used here to indicate a single space.

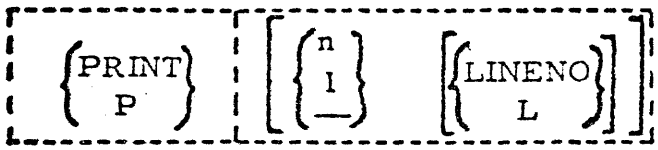
c. Ob!%C

	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

Format:



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

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 out. If the pointer is positioned at the top of the file, the null line is included in the number of lines 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 non-blank characters in columns 1-72 of each line are typed out.

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

Responses:

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

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

Examples:

a. 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 out 5 lines. The line identifier is not included. The pointer is positioned at the last line typed.

QUIT

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.

Responses:

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

94

QUIT

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.

Responses:

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

Examples:

a. Q

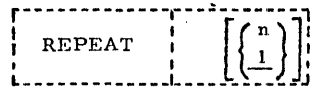
```
request:                Q
response from CMS:      R; T=3.00
```

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

95

REPEAT

Format:



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 will be in effect until the end of the file. Thus, the REPEAT request can provide globol BLANK and OVERLAY requests.

Responses:

The keyboard is unlocked.

Examples:

a. REPEAT 25

The following BLANK or OVERLAY request will be executed 25 times.

96

RETYPE

Format:

{RETYPE / R}	line

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

The pointer is not advanced by this request.

Responses:

The keyboard is unlocked.

Examples:

a. R#IREG = J + K**2

```
                          Columns
                     1    7       14
line before request:      IRE55 = I - 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.

97

SAVE

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 on 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 is being stored on the disk for the first time, it is written on the permanent disk with mode P1. 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. 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 will not be written on the disk and the Input environment will be entered.

NO PRIMARY NAME SPECIFIED - RETRY
A filename was not specified for this file, so it cannot be saved. Re-issue the SAVE request, specifying a filename.

SAVE

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 on 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 is being stored on the disk for the first time, it is written on the permanent disk with mode P1. 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. 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 will not be written on the disk and the Input environment will be entered.

Examples:

a. SAVE filename

request: save my
response: INPUT:

SAVE MY was issued to write the latest copy of the file on disk and give it the filename of MY. The Input environment of EDIT is then entered.

98

Examples:

a. SAVE

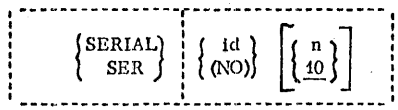
```
request:    save
response:   NO PRIMARY FILE NAME SPECIFIED-RETRY.
request:    save my
response:   INPUT:
```

SAVE was issued for a file that did not have a filename specified with the EDIT command, as the first response indicates. SAVE MY was then issued to write the latest copy of the file on disk and give it the filename of MY. The Input environment of EDIT is then entered.

99

SERIAL

Format:



id — specifies the 3 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. This number also becomes the first line number. The default value is 10.

Usage:

This request allows the user to specify the 3 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. Upon issuing the EDIT command for a new file, 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 will be 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 will replace the contents of columns 73-80; the replacement will not occur until the current pointer is positioned at the top of the file or until a FILE or SAVE request is issued. The entire file will be resequenced with the new identifier.

SERIAL

Usage: (Cont.)

If a file already exists with identifiers or if input lines have been entered which were serialized, the SERIAL (NO) request will have no effect and the identifiers will not be changed. Once serialization has begun, it cannot be nullified.

Note:

a. For a filetype of MEMO, the default option for serialization will be SERIAL (NO). That is, if serialization is desired, it must be explicitly stated.

Responses:

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 will be placed in columns 73-80.

If the file already exists without identifiers, the data in columns 73-80 will be maintained. If a file already exists with identifiers or if input lines have been entered which were serialized, the SERIAL (NO) request will have no effect until the current pointer is positioned at the top of the file after which no new serialization will take place.

TABDEF

Format:

character — is any valid character to be used as the logical tab character. The default character is the #.

Usage:

TABDEF redefines the character to be recognized as the logical tab character. If TABDEF is not issued, the character # will be the assumed character.

If the character # is to be used as a valid input character, the logical tab character must be redefined using this request.

If TABDEF is issued without specifying any character, the logical tab character is reset to the character #.

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
card image in file:	X = A + B

If the logical tab setting is set in column 7, the expression "X=A+B" will begin in column 7 of the card image.

Responses:

The keyboard is unlocked.

Example:

a. TABD $

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

TABDEF

Format:

character is any valid character to be used as the logical tab character. The default character is the !.

Usage:

TABDEF redefines the character to be recognized as the logical tab character. If TABDEF is not issued, the character ! will be the assumed character.

If the character ! is to be used as a valid input character, the logical tab character must be redefined using this request.

If TABDEF is issued without specifying any character, the logical tab character is reset to the character !.

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
card image in file:	X = A + B

If the logical tab setting is set in column 7, the expression "X=A+B" will begin in column 7 of the card image.

Responses:

The keyboard is unlocked.

Example:

a. TABD $

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

TABSET

Format:

{TABSET / TABS}	[n1...nN]

n1...nN are column positions for logical tab settings. If omitted, the default tab settings will be used.

Usage:

This request enables the user to establish his own internal or logical tab settings for a card image. The request is followed by from 1 to 11 numbers that do not exceed the value of 80. The first number indicates the column in which the card image begins, and the following 10 numbers specify the logical tab settings.

If more than 11 numbers are specified, only the first 11 are used. If a number greater than 80 is specified and no serialization is being performed, input lines will be truncated to 80 characters, when tabbing indicates a column position above 80. If an identifier is being placed in columns 73-80, input lines will be truncated to 71 or 72 characters when tabbing indicates a column position above 71.

If the first number, specifying the column in which the card image begins, is not equal to 1, all input to the file will start at the specified column in the card image and all previous columns are set to blanks. The EDIT requests BLANK, FIND, and OVERLAY consider each card image 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 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 will be in effect until TABSET is again issued.

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.

Responses:

The keyboard is unlocked.

Examples:

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

TOP

Format:

Usage:

This request repositions the pointer to the top of the file, i.e., 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 condition immediately preceded the FIND, LOCATE, or CHANGE request.

Responses:

The keyboard is unlocked.

Examples:

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

UP

Format:

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

Usage:

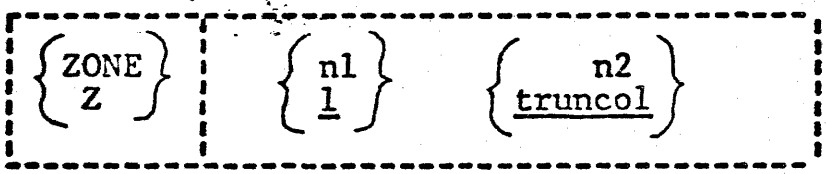
The UP request is the same as the BACKUP request. See BACKUP.

Responses:

See BACKUP.

ZONE

Format:



n1 specifies the initial columns 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, EXEC, MAD, AS1130, and REPS files. If serialization is suppressed, the default value is 80.

Usage:

This request will cause subsequent LOCATE and CHANGE requests to apply only to the zone of the card-image records specified by the integer values for starting and ending columns.

The initial zone column, n1, must be less than or equal to 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.

Note:

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

Responses:

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.

VERIFY

Format:

nn specifies the number of columns to verify in each card-image. The default is 72.

Usage:

This request turns off the brief mode (see the BRIEF request) and turns on the verify mode, causing the automatic typeout of lines changed or searched for by other EDIT requests. If "nn" is specified, "nn" columns will be verified. If "nn" is not specified, 72 columns will be verified; this is the normal mode of operation in the Edit environment. If a MEMO file is being edited, 80 columns will automatically be verified.

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

Responses:

The keyboard is unlocked.

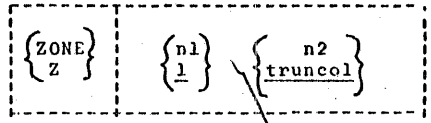
Examples:

a. VERIFY
The verify mode is turned on and 72 columns will be verified unless the file being edited has a MEMO filetype, in which case 80 columns will be verified.

b. VERIFY 50
The first 50 columns will be verified in each card-image.

ZONE

Format:



n1 specifies the initial columns 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, EXEC, MAD, AS1130, and REPS files. If serialization is suppressed, the default value is 80.

Usage:

This request will cause subsequent LOCATE and CHANGE requests to apply only to the zone of the card-image 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.

Note:

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

Responses:

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:

a. 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.

3.1.7 ERASE

Purpose:

The ERASE command deletes a file or a related group of files from the permanent or temporary disks.

Format:

Usage:

Filename and filetype must be specified in the ERASE command, either by name or with an asterisk. If the filemode is omitted, an asterisk is assumed meaning that the permanent and temporary directories (but not the system directory) will be 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 identifier are deleted from the appropriate directory(s), and disk space occupied by these files is made available for new files.

Notes:

a. ERASE will delete read-only files.

Responses:

ERASE gives no response except the Ready message or an error code. See 3.1.7-3.

Examples:

a.

```
ERASE DLFAC MODULE P5
```

The file specified is deleted from the file directory, and its space on the permanent disk is freed.

b.

```
ERASE * LISTING
```

All files with the type LISTING are deleted from the permanent and temporary disks.

c.

```
ERASE * *
```

All user files are deleted and the directory is cleared.

Error Messages:

E(00001) CORRECT FORM: ERASE FILENAME FILETYPE FILEMODE, NAME AND/OR TYPE MAY BE *, AND MODE MAY BE * OR BLANK.

The filename or type was omitted, or the filemode was incorrect. Correct the command.

E(00002)

The file specified was not found in the user's file directory.

E(00004)

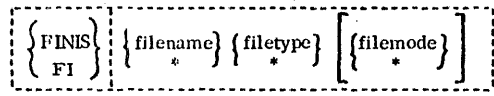
An I/O error occurred. Processing may not have completed. It may be necessary to initialize the disk again. (See FORMAT)

3.1.8 FINIS

Purpose:

FINIS closes one or more files that are currently open.

Format:



{filename / *} is the name of the file to be closed. An * means all filenames.

{filetype / *} is the type of file to be closed. An * means all filetypes.

{filemode / *} is the mode of the file to be closed. An * 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 will be 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 the permanent disk, temporary disk, and system disk, in that order.

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

Notes:

a. 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 will be closed automatically.

b. Only eight files may be open in CMS at one time. Therefore, the FINIS command should be called from user programs in which more than eight files are accessed.

Responses:

None.

Examples:

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

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

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

d. FINIS * * *
All files that are open will be 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. It is not either P, T, S, or *. The command has terminated.

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

3.1.9 LISTF

Purpose:

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

Format:

	filename	filetype	filemode	(option1...optionN)
LISTF L	*	*	*	

filename * — is the name cf the files tc be listed. An * denotes all filenames and is the default value.

filetype * — is the type cf file tc be listed. An * denotes all filetypes and is the default value.

filemode * — is the mcde cf the files to be listed. It may be either P, T, S, or * . The * denotes all filemcdes, P, T, and S. The default is P and T.

Options:

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

SORT (S) — will "sort" or group-together all similar filetypes

NAME (N) — will prcduce a list cf only filenames

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

MODE — trancates the typed line after

(M)	filemode
REC (R)	filename,-type,-mode, and number of records will be printed
DATE (D)	the list will contain name, type, mode, size and the date the file was last written (mm/dd). This is the default line.
TIME (I)	added to the defaulted line will be the time that the file was last updated (hh/mm)

Usage:

LISTF either types out the of 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 in the user's permanent and temporary file directories is typed at the terminal. The list consists of the name, type, mode, number of records, and date the file was last written in each file specified. 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 3.1.9-A.

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

If the filemode is not specified, only the permanent and temporary disk directories will be examined by LISTF. If a filemode of * is specified, the permanent, temporary, and system disks will be used.

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 will be erased and a new file will be created. This file will contain 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 3.1.9-B, in which the PRINTF command has been used to typeout the contents of the EXEC file.

The CMS EXEC file is like any other user file. It can be printed offline, edited, added to, changed, etc., but its main purpose is to be used with the EXEC or $ commands. Refer to Section 3.5.3 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 will be 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 permanent and temporary disks are typed out. See FIGURE 3.1.9-A.

b. LISTF * FORTRAN (R)
The name of each file that has a filetype of FORTRAN and exists on the permanent or temporary disks is typed out.

c. LISTF FILE * * (T)
The name, type, mode, size, date and time of each file that has a filename of FILE and exists on the permanent, temporary, or system disk will be typed out.

d. LISTF (EXEC)
The file with the identifier LIST EXEC P1 is created on the permanent disk. This file will contain the same list of files that was typed out in Example a, but each entry in the list will have &1 and &2 placed in front of it.

e. LISTF * FORTRAN (EXEC)
The file with the identifier LIST EXEC P1 is created. This file will contain the same list of files that was typed on the

terminal in Example b, but each entry in the list will have &1 &2 placed in front of it. See FIGURE 3.1.9-B.

Error Messages:

E(CCCC1) INVALID LISTF PARAMETER LIST.
An incorrect form of the command was issued. Check to see if all parameters are valid.

E(CCCC2) FILE NOT FOUND.
The specified file does not exist on the disk.

3.1.10 MACLIB

Purpose:

The MACLIB command either (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 macroname1...macronameN DEL libname macroname1...macronameN LIST libname COMPACT 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 — will compact the macro library and remove the macroes that have been deleted.

PRINT — offline prints the macro library dictionary.

LIST — types out the dictionary of the macro library specified by "libname".

libname — specifies the filename of a macro library

to be generated, added to, or listed.

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.

macroname1...macronameN specifies the macroes to be DELeted or REPlaced.

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 /360 assembler language statements identified by a unique name and used as an expansion for a source statement in a /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.

maclib list syslib

MACRO	INDEX	SIZE
CMSTYPE	2	10
CMSAVE	12	42
CMSREG	54	29
MADCALL	83	21
MADDEL	104	15
MADTYPE	119	11
CMSYSREF	130	78
FINIS	208	7
RDBUF	215	12
GEN	227	11
TYPE	238	49
FCB	287	20
STATE	307	9
CKEOF	316	5
RDTYP	321	14
WRBUF	335	12
SETUP	347	8
FRASE	355	7
TYPIN	362	14
WAITR	376	20
RDJFCB	396	11
CLOSE	407	158
DCB	565	518
DCBD	1083	534
FREEMAIN	1617	223
GET	1840	9
FREEPOOL	1849	18
GETMAIN	1867	255
GETPOOL	2122	49
LINK	2171	23
OPEN	2194	207
PUT	2401	9
READ	2410	34
SAVE	2444	84
SPIE	2528	143
TIME	2671	17
WAIT	2688	19
WRITE	2707	34

R; T=1.02

FIGURE 3.1.10-A. Example of MACLIB List command.

```
printf syslib maclib 73 86
        MACRO.                                                          SYS00730
&LABEL  CMSTYPE &MESSAGE                                                SYS00740
&LABEL  LA      1,TYP&SYSNDX    SET PARAMETER LIST                      SYS00750
        SVC     X'CA'           ISSUE CALL TO TYPLIN                    SYS00760
        BC      15,TRA&SYSNDX   BRANCH AROUND PARAMETER LIST            SYS00770
        DS      CF              (ALIGNMENT)                             SYS00780
TYP&SYSNDX DC   CL8'TYPLIN'     COMMAND NAME                            SYS00790
        DC      AL1(1)          CONSOLE NO.                             SYS00800
        DC      AL3(MES&SYSNDX)       LOC. OF MESSAGE                   SYS00810
        DC      C'D'            RED INK,FIXED-LOCATION-MESSAGE          SYS00820
        DC      AL3(L'MES&SYSNDX)     LENGTH OF MESSAGE                 SYS00830
MES&SYSNDX DC   C&MESSAGE       ACTUAL MESSAGE                          SYS00840
TRA&SYSNDX DS   CH                                                      SYS00850
        MEND                                                            SYS00860
R; T=0.57
```

FIGURE 3.1.10-B. Printout of the CMSTYPE macro definition Using PRINTF.

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 /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 COPY.

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.

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 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 form the MACLIB LIST command see FIGURE 3.1.10-A which lists the CMS system macro library SYSLIB MACLIB.

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 will create a MAP of the specified library and print it on the offline printer. "libname MAP P1" will also remain on the user's files.

The REP function will replace the existing code of the specified macro, with the new code from the ASP360 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 (see Section 3.4.1).

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 will be 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 will be added to the end of the existing MACLIB file.

e. No more than 100 macro definitions can reside in a MACLIB file. If an attempt is made to include more than 100, an error message will be typed out, and any macro definitions beyond the first hundred will not be included in the MACLIB file.

f. 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 3.1.10-B.

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 IBLIB 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 3.1.10-A.

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. Re-issue the command with the correct format.

E(00002) ERROR WHILE WRITING
An error occurred while writing the library on disk. The library will be closed and the command terminated. All of the input card images that were processed before the error occurred will be included in the file "libname" MACLIB.

E(00003) ERROR WHILE READING
An error occurred while reading the disk. The library will be 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 will be included in the file "libname" MACLIB.

E(CCC04) INPUT FILE NOT IN CORRECT MACRO FORMAT.
The ASP360 file is not in correct macro format--either the card images are out of order, the macro name is longer than 8 characters, there is a blank in column 10, or a MEND card is missing from a macro definition. The library will be closed and the command terminated. All of the input card images that were processed before the error occurred will be included in the file "libname" MACLIB.

"libname" DOES NOT EXIST. IGNORED.
An ASP360 input file does not exist--check to see that the specified libname has a filetype of MACLIB.

E(CC006) 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(CC007) MACLIB FILE SPECIFIED NOT IN CORRECT FORM.
The specified libname is not a valid MACLIB file as created by the MACLIB command.

E(CCC08) MACLIB DICTIONARY OVERFLOW -- TOO MANY MACROS. SORRY.
The MACLIB dictionary contains 100 macro names. No more can be added. The MACLIB file will be closed and the command terminated. All of the input card images that were processed before the error occurred will be included in the file "libname" MACLIB.

E(CC009) xxxxxxxx NOT FOUND IN DICTIONARY.
The macro xxxxxxxx does not exist in the specified macro library. The MACLIB command is terminated. On macros in the ASP360 files that were in front of the macro xxxxxxxx have been replaced in the macro library.

E(CC01C) REPLACEMENT FILE NOT FOUND.

3.1.11 - CFFLINE

Purpose:

The CFFLINE 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>
  (0)                (*)
```

where command = :

READ	specifies a deck is to be read from the card reader.
PRINT	the specified file will be printed onto the system printer with automatic single spacing.
PRINTCC	the specified file is to be printed with the first character of each record interpreted as a carriage control character.
PRINTUPC	the records of the named file will be translated to upper case, then printed.
PRINTVLR	the specified, variable-length file will be printed.
PUNCH	the specified file will be punched onto the system card punch.
PUNCHCC	and "CFFLINE READ filename filetype" control card will be inserted as the first card prior to punching the specified file.

filename filetype <filemode>
identify the file to be transferred. If filemode is omitted for a READ, P1 is assumed. For output operations, if filemode is omitted, the permanent, temporary, and system disks will be searched for the file.

* specifies that filename, filetype, and filemode will be found in OFFLINE READ control cards in the input card stream.

Usage:

Input:
If filename and filetype are specified with the OFFLINE READ command, only one file will be read in. A previously existing file with the same identifiers is erased. If filemode is omitted, P1 is assumed. If filemode is not specified, the field must be left blank, an asterisk is not accepted.

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 " will be 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 operating 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 on 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 10 columns, and the user's identification starting in the 13th column. CP will save the deck until the user logs on 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 they were entered.

Output:
For the OFFLINE PRINT, PRINTCC, PRINTUPC, PRINTVLR, PUNCH and PUNCHCC commands, filename and filetype must be specified. If the filemode field is blank, the permanent, temporary, and system disks are searched, in that order. Asterisks are not permitted in any field.

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 will perform upper case 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 will print the file in upper case, eliminating all print checks and garbeled characters.

OFFLINE PRINTVLR will print 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 disgarded 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 command will insert as the first card of the specified file to be punched, a "OFFLINE READ..." control card. The punched deck can now be read by means of a "OFFLINE READ *" command.

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. CP processes all cards following it until a physical end-of-file as a single file.

c. If the file being printed with OFFLINE PRINT has a filetype of LISTING, it is printed as if PRINTCC were specified.

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

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.
The Ready message indicates the command has completed without error. It does not mean that physical output has completed. The output file may be 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 the first card of the input stream is not an "OFFLINE READ

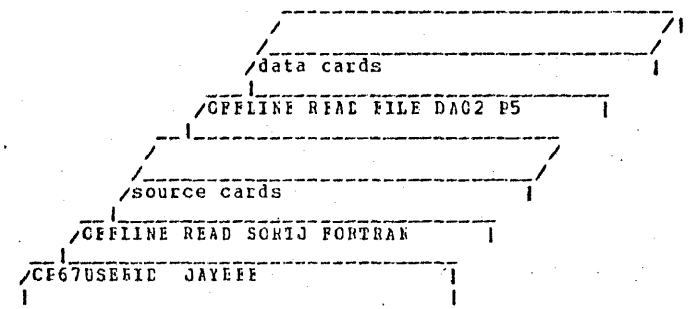
filename filetype <filemode>" control card.

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

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
```

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 will contain only the USERID. The second page will start with the following heading:

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

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.

d. CFFLINE PRINTCC EX FILE P5
See Figure 3.1.11-A below, which shows the contents of the file EX FILE P5, and the printed output produced by this command.

File Contents	Notes	Output
	The USERID always precedes printed output. It is followed by an automatic page eject.	USERID
TITLE---------->	A "1" here would have an extra blank page.	TITLE
0)		
0 (---------->	Each "0" generates two blank lines, or a line followed by a blank line.	
0 HEADING)		HEADING
LINE		
LINE---------->	No carriage-control character causes single spacing.	
CLINE		
CSUB-HEADING		LINE
LINE		LINE
LINE		LINE
1PAGE TWO		SUB-HEADING
0		
CHEADING		
LINE		LINE
LINE		LINE
0LINE		
C EXAMPLE		
LINE		
LINE	The "1" forces a skip to the next page before the line is printed.	PAGE TWO
		HEADING
		LINE
		LINE
		LINE
	The carriage-control character must be in column 1, regardless of the position of the text.	EXAMPLE
		LINE
		LINE

Figure 3.1.11-A. Output produced by an CFFLINE PRINTCC command.

Error Messages:

E(CCC01) INCORRECT COMMAND LIST.
a. The operation specified with the command was invalid; it must be one of these: READ, PRINT, PRINTCC, PRINTUPC, PRINTVLR, PUNCH, or PUNCHCC.
b. No filename (nor "*" under the special READ mode) was specified.
c. Filetype was specified as an "*". That's a no-no!

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

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 CFFLINE PUNCH command accepts records of 80 characters or less. Issuing an EDIT command for the file and re-filing it truncates records to 80 characters.

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, attempt to output an empty file, attempt to open more than eight files at one time, or not having 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.

Additional Notes:

1. With the CP capability of "XFER 00E" a user can receive files in his virtual card reader which have a logical record length of 132 bytes. If the file is transferred via OFFLINE PRINTCC or if its FILETYPE is LISTING, the first character of each record is assumed to be a carriage control character. Otherwise, an extra character (blank) is added to the beginning of each record,i.e., CMS supplies the carriage control character. When the file is read in, the first character is ignored.

   In order to preserve the carriage control in a LISTING file, the user must first alter the FILETYPE to other than LISTING, and then issue an OFFLINE PRINT, not PRINTCC. The first character of each record is then treated as a part of the file.

130

3.1.12 PRINTF

Purpose:

The PRINTF command types all or part of a specified file at the terminal.

Format:

{PRINTF / P} 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 will be typed out. 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 if the filetype is LISTING, to 80 characters if the filetype is MEMO, or to 72 characters if the filetype is not any of the above. If a limit is specified, the first line number and last line number fields must be filled, either explicitly or with asterisks.

The search for the file is made on the permanent, temporary, and system disks in that order. In the case of files with duplicate filename and filetype, only the first file found will be typed out.

Notes:

a. The first line number and last line number must not be greater 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.

131

c. The KT and KE commands override any specified last line number or line length.

Responses:

The specified lines are typed out at the terminal, followed by the Ready message. If the Ready message appears with no typed lines, the specified first line number is beyond the end of the file.

Examples:

a. printf go exec * * 80
The entire file "GO EXEC P5" is typed out. Lines are truncated to 80 characters. As shown in Figure 3.1.12-A, the command "printf go exec" produces the same output except that lines are truncated to 72 characters.

b. printf syslib maclib 104 118 72
A macro definition from "SYSLIB MACLIB SY" is typed out by specifying its first and last line numbers in the file (obtained with the MACLIB LIST command). The line limit of 72 characters truncates the serial number in columns 73-80 of each line. See Figure 3.1.12-B.

c. printf fortj listing 33 * 72
The 33rd through the last line of the file "FORTJ LISTING P5" are typed out, truncated to 72 characters. See Figure 3.1.12-C.

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.

```
printf go exec

LOAD %1                                                    GO 00010
START                                                      GO 00020

R; T=0.03
```

FIGURE 3.1.12-A. Two examples of PRINTF commands which type out an entire file.

```
printf syslib maclib 104 118 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=0.07
```

FIGURE 3.1.12-B. A PRINTF command which 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
```

FIGURE 3.1.12-C. A PRINTF command which types out the bottom of a FORTRAN LISTING file.

3.1.13 SCRIPT

Purpose:
The SCRIPT command performs three functions: (1) it creates a file on disk of variable length records from successive terminal input lines, (2) it modifies these files, and (3) it types these files in a format specified by included control words.

Format:

SCRIPT SC	EDIT filename PRINT filename	(option1...optionN)

EDIT — specifies that a file is to be created or modified. If the file exists, the user is placed in the SCRIPT EDIT environment. If it does not exist, the user is placed in the SCRIPT INPUT environment, so that the file may be created.

PRINT
P — formats and outputs the specified manuscript file onto the terminal, the offline printer, or into another file.

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 PRINT into a file named: ".filename" instead of at the terminal or offline printer.

NOWAIT
NO — starts SCRIPT PRINT 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 PRINT on the offline printer instead of at the terminal.

PAGExxx — causes printout to start at page xxx.

SINGLE
SI — after one page of printed output SCRIPT PRINT terminates, usually used in conjunction with the PAGExxx option.

STOP
ST — causes a pause at the bottom of each page during SCRIPT PRINT.

TRANSLATE
TR — translates lower case letters to upper case in printout.

UNFORMATTED
UN — the input SCRIPT file is printed along with control words, the control words are ignored and formatting does not occur.

Usage:

Filename must be specified with the SCRIPT command. The filetype SCRIPT is assumed. Files created by the SCRIPT command have the filemode P1, but existing input files to SCRIPT EDIT may have mode letter P or T, and mode number 1 or 5. SCRIPT PRINT accepts any filemode.

The SCRIPT EDIT version of this command places the user in one of two environments: The Script Input environment or the Script Edit environment. In the Script Input environment, a file is created or expanded by adding successive terminal input lines. In the Script Edit environment, existing files whose filetype is SCRIPT can be examined and modified. Multiple passes can be made through the file, adding, deleting, or replacing lines. The Script Edit environment accepts typed requests that specify an operation on a line, or a move to another position in the file.

The SCRIPT PRINT version of the SCRIPT command examines each line of the file for format control words, which are identified as lines which begin with a period, and types out the file according to the format specified.

The SCRIPT EDIT requests are described individually beginning on Page 3.1.13-8 and SCRIPT PRINT format control words are described individually beginning on Page 3.1.13-24. Examples of a SCRIPT file and SCRIPT PRINT output are shown in Figures 3.1.13-A and

3.1.13-B, beginning on Page 3.1.13-44.

Script Input Environment:

The Script Input environment is entered when a SCRIPT EDIT command is issued specifying a file that does not exist. It may also be entered from the Script Edit environment by issuing the INPUT or the BOTTOM requests. Whenever this environment is entered, the response "INPUT:" is typed and the keyboard is unlocked to accept an input line. As each input line is entered, it is added to the file, and the keyboard is unlocked to accept the next line.

The @ and ¢ symbols are deletion characters in this environment. An @ symbol deletes itself and the preceding character. Several consecutive @ symbols delete an equal number of preceding characters. The ¢ symbol deletes itself and all characters preceding it on the line. The character immediately following the ¢ symbol is effectively at the left margin. Any other character on the keyboard is a valid character, including the TAB key and the BACKSPACE key.

The maximum length of an input line is 120 characters. If deletion symbols are included in the line, up to 132 characters may be entered, so long as the effective length after deletions is 120 characters or less (including backspaces). Any line longer than 120 characters will be truncated.

Note: Backspaces and underlines are included in the 120 character limit (a CP/CMS limitation, not a SCRIPT limitation). A line consisting of 40 underlined characters occupies 120 characters. A longer line would be truncated with no warning message by SCRIPT since it operates independently of the CP/CMS truncation.

The only exit from the Script Input environment is to the Script Edit environment. This exit is taken when a null line (a line of length 0) is entered. A null line is generated whenever a carriage return is typed while the carriage is already at the left margin. The null line does not become part of the file. Because the null line is the only execution control in the Script Input environment, it is always necessary to go to Script Edit to save a file or to return to the CMS Command environment.

Script Edit Environment:

This environment is entered from the Script Input environment by typing a null line, or directly from the CMS Command environment by specifying an existing SCRIPT file with the SCRIPT EDIT command. In either case, the response "EDIT:" is typed, and the keyboard is unlocked to accept a SCRIPT EDIT request. When a

request is entered, the keyboard is locked until the action specified is completed. As soon as the keys are freed, another request may be entered.

Whenever the user is in the Script Edit environment, a pointer is maintained specifying his current position in the file, or the "current line". SCRIPT EDIT requests perform operations on the current line, on the pointer, or on both.

When the Script Edit environment is entered from the Script Input environment, the current line will be the last input line entered. When the Script Edit environment is entered from the CMS Command environment, the pointer is positioned at a blank line preceding the first line of the file. This line is part of the file only in the Script Edit environment, and is not stored with the file. It may not be modified or deleted, but it allows lines to be inserted at the top of the file.

The pointer always moves forward through the file. Successive passes through the file can be made by using the TOP request, which repositions the pointer at the blank line preceding the file. See the individual request descriptions for the action of each on the pointer.

Requests may be entered in upper or lower case. The first letter of a request is sufficient to uniquely identify the request. For example, a single "t" or any line beginning with "T" would specify the TOP request. Requests which may include numbers must be separated from the number by at least one blank.

Three of the SCRIPT EDIT requests provide exits from the Script Edit environment. FILE stores the current copy of the file on disk, and returns to the CMS Command environment. QUIT returns to the CMS Command environment without saving the current copy of the file. INPUT exits to the Script Input environment.

SCRIPT PRINT:

When the SCRIPT PRINT command is issued, the specified SCRIPT file will be 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 non-continuous 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 will resume 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 upper and lower case 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 cannot be done by the regular "OFFLINE PRINT" CMS command).

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 3-digit page number and must include leading zeros (i.e., 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.

Each line read from the disk file by SCRIPT PRINT 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 upper or lower case 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.

Notes:

a. The Script Edit and Script Input environments should not be confused with the Edit and Input environments accessed by the EDIT command. Although many of the SCRIPT EDIT and EDIT requests are similar, the action taken is not always the same.

b. A ¢ symbol or a symbol immediately followed by a carriage return in the Script Input environment does not generate a null line, and no exit to the Script Edit environment will be taken. The carriage must be physically at the left margin for a carriage return to generate a null line.

c. The TAB key generates an acceptable character in the Script Input environment, and is retransmitted by SCRIPT PRINT. 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 .CP control words should be used instead of the TAB key.

d. The SCRIPT command employs two work files during execution. They are (INPUT1) FILE P1 and (INPUT2) FILE P1. If a previously existing file is being processed, the mode P1 is replaced by the mode of the input file. These files are normally erased on command termination. One of them replaces the previously existing file, if any, just before control is returned to the CMS Command environment. In cases of a machine or line failure causing an unusual termination, all or part of an edited file may be salvaged from (INPUT1) or (INPUT2). If a previously existing file was being processed, it still exists as though no changes had been made to it.

e. Input lines to SCRIPT EDIT are scanned and canonicalized with respect to underlined and overprinted characters. This process is explained in detail in CSC Report 320-2023, "SCRIPT: An Online Manuscript Processing System".

139

Responses:

SEM --- SEPT. 1968 (2)

This response is typed whenever the SCRIPT command receives control. (SEM is an acronym for SCRIPT ENVIRONMENT MODULE.) With SCRIPT PRINT a pause follows this response until a carriage return is typed unless the NOWAIT option has been specified.

INPUT:

This response is typed whenever the Script Input environment is entered.

EDIT:

This response is typed whenever the Script Edit environment is entered.

INVALID SCRIPT EDIT COMMAND

The request just specified was either not a valid request word or did not contain a valid operand. Refer to the description of the individual request for its proper format.

LOAD PAPER; HIT RETURN

This response is given whenever the SCRIPT PRINT command is issued without specifying the NOWAIT option. The carriage return must be hit in order for SCRIPT PRINT processing to continue. The paper should be adjusted as described above.

Error Messages:

E(00001) CORRECT FORM IS:"SCRIPT" "EDIT"FILENAME.
No filename was specified with the SCRIPT EDIT command. No action was taken.

E(00002) DISK ERROR.
An I/O error occurred. If the file existed before the command was issued, it has not been changed by any editing requests. See Note d.

E(00004) INCORRECT PARAMETER LIST
An invalid parameter has been specified for a SCRIPT PRINT control word, or a required parameter has been omitted. The SCRIPT PRINT command has been terminated.

E(00008) FILE xxxxxxxx NOT FOUND.

140

No SCRIPT version of filename xxxxxxxx was found. The SCRIPT PRINT command has been terminated.

E(00012) DISK ERROR WHILE READING
A disk error was incurred by SCRIPT PRINT. The SCRIPT PRINT command has been terminated, and the disk file being printed will remain unchanged.

E(00016) ILLEGAL CONTROL CARD ENCOUNTERED.
An unrecognizable control word was encountered while printing a SCRIPT file. The SCRIPT PRINT command has been terminated.

E(00024) MORE THAN EIGHT ACTIVE FILES - REDUCE NESTING.
This message will be given by SCRIPT PRINT when more than seven IMBED control words are encountered in the SCRIPT file being printed. All open files will be closed and the SCRIPT PRINT command will be terminated.

E(00069) CORRECT FORM IS: 'SCRIPT' COMMAND NAME
WHERE COMMAND IS 'EDIT' OR 'PRINT'.
The command was not SCRIPT EDIT or SCRIPT PRINT. No action was taken.

For all of the error messages from SCRIPT PRINT, the following message is printed:

ERROR OCCURRED AFTER READING XXXX LINES.

This usually assists in finding offending error in SCRIPT file.

E(CCCCC) *** 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.

Requests:

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

Each request is separated from its operands, if any, by one or more blanks. The SCRIPT EDIT requests are listed below and are

3.1.13-7 SCRIPT

RESPONSES:

INPUT:

This response is typed whenever the Script Input environment is entered.

EDIT:

This response is typed whenever the Script Edit environment is entered.

INVALID SCRIPT EDIT COMMAND

The request just specified was either not a valid request word or did not contain a valid operand. Refer to the description of the individual request for its proper format.

LOAD PAPER; HIT RETURN

This response is given whenever the SCRIPT PRINT command is issued without specifying the NOWAIT option. The carriage return must be hit in order for SCRIPT PRINT processing to continue. The paper should be adjusted as described above.

ERROR MESSAGES:

E(00001) CORRECT FORM IS: "SCRIPT" "EDIT" FILENAME.
No filename was specified with the SCRIPT EDIT command. No action was taken.

E(00002) DISK ERROR.
An I/O error occurred. If the file existed before the command was issued, it has not been changed by any editing requests. See Note d.

E(00004) INCORRECT PARAMETER LIST
An invalid parameter has been specified for a SCRIPT PRINT control word, or a required parameter has been omitted. The SCRIPT PRINT command has been terminated.

E(00008) FILE xxxxxxxx NOT FOUND.

described on the indicated pages:

Request	Page Ref.
BOTTOM	3.1.13-8
CHANGE	3.1.13-9
DELETE	3.1.13-11
FILE	3.1.13-12
INPUT	3.1.13-13
INSERT	3.1.13-14
KEEP	3.1.13-15
LOCATE	3.1.13-16
NEXT	3.1.13-17
PRINT	3.1.13-18
QUIT	3.1.13-19
REPLACE	3.1.13-20
SEEK	3.1.13-22
TOP	3.1.13-23

SCRIPT PRINT Control Words:

The SCRIPT PRINT Control words are issued as input lines to a file being created in the Script Input Environment. These words will later be interpreted by the SCRIPT PRINT command to govern format control as the file is being printed out.

The SCRIPT PRINT control words are listed below and are described on the indicated pages:

Control Word	Meaning	Page Ref.
.AE	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	

BOTTOM Request

Purpose:

BOTTOM prints the last line of the file being edited, positions the pointer after this line,and enters the Script Input environment.

Format:

BOTTOM
B

Usage:

This request may be issued whenever the keyboard is unlocked in the Script Edit environment. It is particularly useful when a large SCRIPT file is being entered in short sessions. BOTTOM prints the last line of the file to remind the user where he left off, positions the pointer after this line, and enters the Script Input environment to allow lines to be added to the end of the file.

Responses:

xxx...xxx
INPUT:

This is the normal response to the BOTTOM request where "xxx...xxx" is the last line of the file being edited and the pointer is positioned after the last line of the file prior to transferring control to Script Input.

NO CURRENT LINE.
INPUT:

This message will be given if a BOTTOM request is issued when the pointer is positioned after the last line of the file (i.e., as the next request after the message "EOF REACHED." has been typed.) The pointer will not be moved, and control will transfer to the Script Input environment.

Example:

a. b
This is the last line.
INPUT:

The last line of the file being edited, "This is the last line." is typed out and control is transferred to the Script Input environment. Any lines entered into Script Input will be appended immediately after this last line.

CHANGE Request

Purpose:

CHANGE replaces a specified string of information in the current line with another specified string of information.

Format:

CHANGE	/string1 / string2/
C	

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

string1 is the group of characters to be replaced. If string1 is not specified string2 will be placed at the beginning of the current line.

string2 is the group of characters which will replace or precede string1. If string2 is not specified, string1 will be deleted from the current line.

Usage:

When the CHANGE request is issued, the current line will be scanned for a match on the characters specified as string1. If a match is found, the characters in string1 will be replaced by the characters in string2. If no match is found, the current line will remain unchanged. The strings do not have to be the same length; the current line will be expanded or compressed as required.

If string1 is not specified, the contents of string2 will be appended to the beginning of the current line (see Example C). If string2 is not specified, string1 will be deleted from the current line (see Example d).

Notes:

a. No verification is provided when a line is changed. To examine the changed line, issue the PRINT request.

b. Only the first occurrence of string1 in the current line will be changed; no global change is provided by SCRIPT EDIT.

Responses:

TRUNCATED
The requested change has extended the current line to more than 120 characters and the line has been truncated.

NO CHANGE.
No occurrence of string1 was found in the current line. The pointer is not moved and the line remains unchanged.

Examples:

a.
```
p
of shoes and ships and sealing wax
c /en/an/
p
of shoes and ships and sealing wax
```
This change request will cause the first occurrence of "en" in the current line to be changed to "an".

b.
```
c ?oes?ares?
p
of shares and ships and sealing wax
```
The first occurrence of "oes" in the current line will be changed to "ares" and the line will be expanded accordingly.

c.
```
c >>to speak>
p
to speak of shares and ships and sealing wax
```
The string "to speak" will be appended to the beginning of the current line.

d.
```
c /sealing//
p
to speak of shares and ships and wax
```
The string "sealing" will be deleted from the current line.

DELETE Request

Purpose:

DELETE erases a specified number of lines from the file and positions the pointer at the location of the last deleted line.

Format:

DELETE D	n

n specifies the number of lines to be erased. If omitted, 1 is assumed.

Usage:

Starting with the current line, the specified number of lines is deleted from the file. If no number is specified, only the current line is deleted. If the DELETE request is issued when the pointer is positioned at the null line before the beginning of the file, this null line will be included in the count of lines deleted, but a null line will remain at the top of the file.

Responses:

EOF REACHED.
The end of file was reached before deleting the specified number of lines. The pointer will be at a position immediately following the last line not deleted.

Examples:

a.
```
DELETE 5
```
Five lines of the file are erased, beginning with the current line.

b.
```
D
```
The current line is deleted. The pointer will remain positioned at the point of deletion.

FILE Request

Purpose:

The FILE request stores the current copy of the file on disk, and exits to the CMS Command environment.

Format:

FILE F		filename

filename specifies the name to be assigned to the created file. If this operand is omitted, the filename specified when the SCRIPT Command was issued will be used.

Usage:

FILE is issued to save a new or edited copy of a file on the user's permanent disk. It may be issued at any time in the Script Edit environment. The position of the pointer is not considered, and the whole file is always stored.

Example:

a. FILE
The file currently being edited is stored on the user's permanent disk, replacing any previously existing copy of the file.

INPUT Request

Purpose:

INPUT transfers the user from the Script Edit environment to the Script Input environment.

Format:

INPUT I

Usage:

INPUT allows the user to add a line or lines to a file following the current line specified by the Script Edit environment pointer. The new lines do not replace any already existing lines, but are inserted between what was the current line and the line that formerly followed it.

Note:

If the pointer was at the top of the file, that is, at the blank line preceding the first line, lines entered after an INPUT precede the former first line of the file.

Response:

a. INPUT:
This response signals entry to the Script Input environment.

Examples:

a. PRINT 1
the following part numbers:
INPUT
The PRINT request types the current line. All lines entered after the response issued by the INPUT request will be inserted in the file following the line "the following part numbers:", until a null line is typed to return to the Script Edit environment.

INSERT Request

Purpose:

INSERT allows a single line of input to be added to the file without transferring to the Script Input environment.

Format:

INSERT I	line

line is the exact input line to be inserted into the file.

Usage:

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

Note:

a. If the INSERT request is issued without an operand, it is interpreted as the INPUT request and the Script Input environment will be entered.

Example:

a. I INPUT Line to be added to file.
The line "INPUT Line to be added to file." would be inserted after the line at which the pointer is currently positioned. The keyboard would then be unlocked to accept another SCRIPT EDIT request, and the pointer would be positioned at the input line just entered.

KEEP Request

Purpose:
KEEP creates a new file on disk consisting of all material from the beginning of the file being edited to the line above that at which the pointer is currently positioned.

Format:

KEEP K	filename

filename specifies the filename to be assigned to the new file.

Usage:

This request allows users to create disk files which consist of portions of existing SCRIPT files. When the KEEP request is issued, the contents of the current file from the first line to the line preceding the one where the pointer is positioned will be saved on the user's permanent disk. This file will have the filename specified in the KEEP request, a filetype of SCRIPT and a filemode of P1. This saved portion will then be erased from the current file and the line at which the pointer is positioned will become the first line of the file being edited.

Responses:

OLD FILE NOT FOUND
The KEEP request was specified with the pointer positioned at the first line of the file. No "old file" exists above the pointer, and therefore no action is taken.

NEW FILE ALREADY EXISTS
The filename specified in the KEEP request has already been assigned to a SCRIPT file. No file has been created by the KEEP request.

INVALID SCRIPT EDIT COMMAND
No filename was specified with the KEEP request. A filename must be specified and must be separated from the request by at least one blank.

Example:

a. KEEP CHAPT3
A file will be created on disk with an identifier of CHAPT3 SCRIPT P1. This file will contain all lines from the beginning of the current file to the line above that where the pointer is positioned. This portion will be erased from the file and the line where the pointer is positioned will become the first line of the file being edited.

LOCATE Request

Purpose:

LOCATE searches the SCRIPT file for the first occurrence of the specified string.

Format:

LOCATE	/string/
L	

/ 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 each line of the file for an exact match of the character string specified between the two delimiters. The scan begins on the line immediately following that at which the pointer is currently positioned and continues until the string is found or until the end of file is reached. When a match is found, the keyboard will be unlocked and the pointer will be positioned at the line in which the first occurrence of a match was found.

Responses:

EOF REACHED.
This message is typed whenever the end-of-file is reached before a match of the string is found.

Example:

a. locate /abcd/
The file will be scanned, starting with the line immediately following that at which the pointer is currently positioned, for an occurrence of the characters "abcd". If a match is found, the keyboard will be unlocked with the pointer positioned at the matching line.

NEXT Request

Purposes:

NEXT moves the current line pointer forward a specified number of lines in the file.

Format:

NEXT N	n

Usage:

The current line pointer is advanced the specified number of lines, or to the next line if no number is specified. If end-of-file is reached, the pointer remains positioned after the last line of the file.

Responses:

EOF REACHED.
The end-of-file was reached before advancing the pointer n lines.

Examples:

a. NEXT 15
The pointer is advanced 15 lines, counting the next line as one.

b. n
The pointer is advanced to the next line.

PRINT Request

Purpose:

The PRINT request types the specified number of lines, beginning with the current line.

Format:

PRINT P	n

n specifies the number of lines to type.
If omitted, 1 is assumed.

Usage:

The PRINT request is used to examine the contents of lines in the file while in the Script Edit environment. The current line is always the first line typed. The pointer is advanced to the last line typed. If only one line is specified, the pointer does not move. If end-of-file is reached, the pointer will be positioned after the last line of the file.

Responses:

NO CURRENT LINE
This message indicates that the pointer is positioned (1) before the first line of the file (2) after the last line of the file, or (3) at a line which has just been deleted.

EOF REACHED.
This message is typed whenever the end of file is reached before n lines have been printed. The pointer will be positioned after the last line of the file.

Examples:

a. PRINT 3
Three lines are typed out at the terminal, beginning with the current line. The pointer is advanced two lines.

b. P
The current line is typed, and the pointer is not moved.

QUIT Request

Purpose:

The QUIT request returns the user to the CMS Command environment, without saving the current copy of the file being edited on disk.

Format:

```
+-----------+
|           |
|   QUIT    |
|   Q       |
+-----------+
```

Usage:

This request provides an exit from the Script Edit environment without saving the copy of the file being edited. If the file existed on disk previously, it still exists without any of the changes made by the current SCRIPT EDIT command. If the file was created by the current command, it is erased.

Example:

a. QUIT
The current copy of the file is erased. Control is returned to the CMS Command environment. If this was a previously existing file, it remains unchanged on the user's permanent disk.

157

REPLACE Request

Purpose:

The REPLACE request erases the current line of a file, and replaces it with a specified line.

Format:

REPLACE R	newline

newline is the line which replaces the current line. This line starts in the first position following the first blank after the request. If no operand is specified with the REPLACE request, the current line will be deleted and the Script Input environment will be entered.

Usage:

The first blank following the request serves as a delimiter. Additional blanks are considered part of the new line. The new line must conform to the requirements of the Script Input environment, except that the request words and the first blank are not counted in determining length.

Notes:

a. If no operand is specified with the REPLACE request, the current line will be deleted and the Script Input environment will be entered. This is especially convenient when it is necessary to replace one line by more than one line.

Responses:

INPUT:
This response indicates that the Script Input environment has been entered. It will be given whenever the REPLACE request is issued without an operand. The keyboard will be unlocked to allow any number of lines to be inserted in the file in place of the current line.

158

Examples:

a. REPLACE even though the hippogrif was
The current line is erased and the line "even though the hippogrif was" is inserted in its place.

b. r In view of these results, it
The current line is deleted, and the line "In view of these results, it" is inserted in its place. Note that five of the six blanks between "r" and "In" are retained as part of the line. The other blank serves as a delimiter between the request word and the newline.

c. r
The current line will be erased and the Script Input environment will be entered to allow any number of lines to be inserted in its place.

SEEK Request

Purpose:

The SEEK request advances the current line pointer to the line whose leading characters are specified by the operand, or to end-of-file if no matching line is found.

Format:

SEEK	string
S	

string — is any number of leading characters from the line to which the pointer is to be advanced.

Usage:

Starting with the line following the current line, the leading characters of each line are compared to the specified string. The comparison is column-dependent; the character following the first blank after the request words will be compared to the first character of the line, and so on. Blanks are significant, and must be included. The first line whose leading characters match the specified characters becomes the current line.

Responses:

a. EOF REACHED.
No matching line was found. The pointer is positioned after the last line of the file.

Examples:

a. SEEK west
Beginning with the line following the current line, the first four characters of each line are compared to the characters "west". The following line would become the current line, if it were in the file below the current line.

west of the Monongahela.

These lines would be skipped:

West of the Moon. (Upper case "W")
west, young man. (Wrong columns)

b. Seek west
This request would search for a line whose first eight characters were " west". The last line of the previous example would become the current line, if it were in the file.

TOP Request

Purpose:

TOP places the pointer at the blank line preceding the first line of the file.

Format:

```
+-------+
|  TOP  |
+-------+
```

Usage:

TOP may be issued at any time to place the pointer in front of the first line of the file. It allows recursive passes to be made through a file with a single SCRIPT EDIT command.

Note:

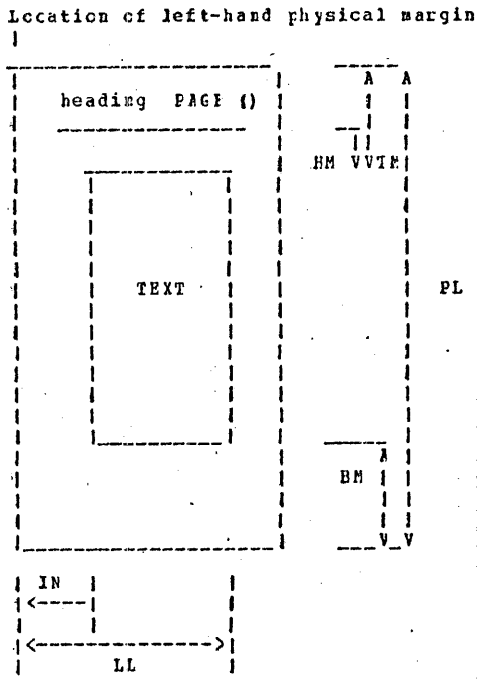
The blank line that becomes the current line after a TOP request is always present in the Script Edit environment. If replaced or deleted, another blank line is supplied. It allows additions to be made at the top of the file.

Examples:

a. TOP
The current line pointer is moved to a position in front of the first line of the file.

Paper Layout

Many of the Script Print control words affect parameters of the paper layout. The diagram below indicates certain of the important paper layout parameters: PAGE LENGTH (PL), TOP MARGIN (TM), BOTTOM MARGIN (BM), HEADING MARGIN (HM), LINE LENGTH (LL), and INDENT (IN).



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 will be closed, and the specified SCRIPT file will be printed as a continuation of the SCRIPT PRINT output from the previous file.

Notes:

a. 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.

Examples:

a. .AP ABC
The contents of SCRIPT file ABC will be 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, 1 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.

Examples:

a. .BM 10
Ten lines will be 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.

Examples:

a.
```
Heading:
First line of the paragraph . . .
```

This part of a file will be printed by SCRIPT PRINT 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 will be 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 will be 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:

a.
```
.CE
Other Methods
```

When this line of the file is typed, the characters "Other Methods" will be 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 in SCRIPT EDIT 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:

a. .CM Remember to change the date.
The line above will be seen when examining a unformatted listing of the SCRIPT file and remind 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 will be 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 PRINT command, CONCATENATE is only included to cancel a previous NO CONCATENATE control word.

Notes:

a. This control acts as a BREAK.

Example:

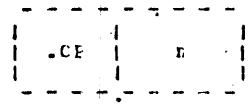
a. .CO
Output from this point on in the file will be formed to approach the right margin without exceeding it.

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:



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 will cause 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) preceeding a section heading to eliminate the possibility of a heading occurring as the last line of a page.

Note:

a. If no operand is specified with .CP request, the request will be ignored.

Example:

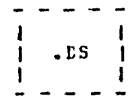
a. .CP 10
If less than 10 lines remain on the current page, an eject will be issued before printout continues. If 10 or more lines remain, printout will continue on the current page.

DOUBLE SPACE Control

Purpose:

The DOUBLE SPACE control word causes a line to be skipped between each line of typed output.

Format:



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:

a. .DS
Blank lines will be 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

```
- - - - -
|       |
|  .FC  |
|       |
- - - - -
```

Usage:

The FORMAT control word is a short-hand 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 lines (concatenate) and then padded with extra blanks to produce an even right margin (justify). This is the normal mode of operation for the SCRIPT PRINT 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:

a. .FC
Output from this point on in the file will be 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 started 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 10 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.

Example:

a. .HE CAMBRIDGE MONITOR SYSTEM
The characters "CAMBRIDGE MONITOR SYSTEM" will be typed at the left in the second-last line of the top margin on all pages started after this point in the file:

b. .he CMS
The leading blanks are considered part of the heading, so the characters "CMS" will be centered in the heading line:

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 spaced (TOP MARGIN), overriding the standard value of one.

Format:

.HM	n

n specifies the number of lines to be skipped after the heading line.

Usage:

The heading line will be 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 one.

The HEADING MARGIN specified must always be less than the current TOP MARGIN.

Note:

This control word acts as a BREAK.

Example:

a. .HM 3
Three lines will be left between the heading line and the first line of text. If default top margin of 5 is in effect, the heading will occur 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 one is set.

Purpose:

The IMBED control word is used to insert the contents of a specified file into the printout of another SCRIPT file.

Format:

.IM	filename

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. Imbedded files may specify additional imbedding to a depth of seven levels. Imbedding may be used to insert standard sets of control words at desired spots in a file as well as many other uses.

Example:

.IM CHAP4
The contents of the SCRIPT file whose filename is CHAP4 will be inserted in the printout of the current SCRIPT file, when the end of the CHAP4 file is reached, printout of the current file will resume.

Error Messages:

E(CCC24) MORE THAN EIGHT ACTIVE FILES - REDUCE NESTING.
More than seven files have been imbedded in the SCRIPT file being printed. Only eight files may be open in CMS at once. All open files will be closed, and the SCRIPT PRINT command will be terminated.

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 will revert to the original margin.

Usage:

The .IN control word causes SCRIPT printout to be idented "n" spaces from the current left margin setting. This indentation remains in effect for all following lines, including new paragraphs and pages, until another .IN control word is encountered. ".IN 0" will cancel the indentation, and printout will continue at the original left margin setting.

Notes:

a. The .IN request acts as a BREAK.

b. The .IN request will reset 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 will take precedence.

Examples:

a. .IN 5
All lines printed after this request will be idented 5 spaces from the current left margin setting. This indentation will continue until another .IN control word is encountered.

b. .IN 0
The effect of any current indentation will be canceled and printout will continue 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. This is the normal mode of operation for the SCRIPT PRINT 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:

a. .JU
Output from this point on in the file will be 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:

a. .LL 50
Succeeding lines will be no more than 50 characters in length.

NC CONCATENATE Control

181

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 lines and output lines. It is useful for sections of files containing tabular information or other special formats.

Notes:

a. This control acts as a BREAK.

Example:

a. .NC
Concatenation will be completed for the preceding line or lines, but following lines will be typed without words being moved to and from lines.

NO FORMAT Control

182

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 short-hand 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. It is useful for sections of files containing tabular information or other special formats.

Notes:

a. This control acts as a BREAK.

Example:

a. .NF
Justification and concatenation will be completed for the preceding line or lines, but following lines will be typed exactly as they appear in the file.

NC 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 blank padding of lines. If concatenate mode is in effect, lines will be formed that approach the current line length but will not be forced to the exact length. The resulting lines resemble the output usually produced by a typist or a MT/ST (Magnetic Tape/Selectric Typewriter).

Notes:

a. This control acts as a BREAK.

Example:

a. .NJ
Justification will be completed for the preceding line or lines, but following lines will be typed without additional blanks inserted 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 will revert to the original margin setting.

Usage:

The .OF control word may be used to indent the left side of the printout. It's effect does not take place until after next line is printed, and the indentation will remain in effect until a break or until another .OF 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 will cause 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 is considered an error condition.

185

Examples:

a. .OF 10
The line immediately following the .OF control word will be printed at the current left margin. All lines thereafter (until the next break or .OF request) will be indented 10 spaces from the current margin setting.

b. .CF
The effect of any previous .OF request will be canceled, and all printout after the next line will continue at the current left margin setting.

186

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 PRINT 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.

Example:

a. .PA
The rest of the current page will be skipped. The heading and page number will be typed in the top margin of the next page, and output will resume.

b. .PA 5
regardless of the number of the current page, the rest of that page will be skipped, the heading and page number 5 will be typed in the top margin of the next page, and output will resume.

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.

Notes:

This control word acts as a BREAK.

Example:

a. .PL 51
Page length is set to 51 lines. This is the correct size for 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 will continue.

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 will discontinued on the printout, although the page numbers will continue to be incremented internally. The OFFNO operand will also discontinue page numbering on the printout and will cause the page numbers not to be incremented internally. When the ON operand is specified, page numbering will resume from the last internal page number.

Example:

a. page off
no further page numbers will appear on SCRIPT PRINT output, although the internal page count will continue to be incremented for each page printed.

b. PAGE OFFNO
No page numbers will appear on SCRIPT PRINT output and the internal page count will remain at its current setting without being further incremented.

c. PAGE ON
Page numbering on SCRIPT PRINT output will resume using the current internal page count, and this count will be incremented for each page printed.

READ Control

Purpose:

The READ Control word allows the user to enter a line from the terminal during SCRIPT PRINT 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 PRINT output to the terminal, it will act as a BREAK, spin the type head several times, and unlock 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 desired; each will result in a separate line accepted at the terminal.

Notes:

This control word acts as a BREAK.

Example:

a. .RD
When this control word is encountered during SCRIPT PRINT output, the type head will rotate and the keyboard will be 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 single-spaced.

Format:

```
.SS
```

Usage:

Output following the SINGLE SPACE control word is single-spaced. Since this is the normal output format, SINGLE SPACE is only included in a file to cancel a previous DOUBLE SPACE control word.

Note:

This control word acts as a BREAK.

Example:

```
a.    .SS
Single-spacing will resume 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 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 strictly monotonically increasing positive values separated by one or more spaces.

Usage:

TAB characters entered into the file during SCRIPT EDIT file creation are expanded by SCRIPT PRINT into one or more blanks to simulate the affect of a logical tab spot. The TAB SETTING control word specifies the locations of the logical tab stops, this overrides the default tab stops of 5, 10, 15, 10, 15, 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 ordered will result in unpredicatable behavior.

c. The right margin is considered to be column 0.

Example:

a. .TB 10 20 30 40
Tab stops are interpreted as columns 10, 20, and 30.

b. .TB
Tab stops will revert to default values of 5, 10, 15, etc.

TOP MARGIN Control

Purpose:

The TOP MARGIN control word specifies the number of lines to be skipped above the text on output pages, overriding the standard value of five.

Format:

.TM	n

n specifies the number of lines to be skipped at the top of output pages. n must be three or greater.

Usage:

The specified number of lines will be left at the top of succeeding output pages before the first line of text. The page number and heading, if any, are placed above the top margin. If no TOP MARGIN control word is included in the file, the default value is five. The TOP MARGIN specified must always be greater than the current HEADING MARGIN.

Note:

This control word acts as a BREAK.

Example:

a. .TM 3
Three lines will be left at the top of pages started after the current page. The heading and page number will be typed in the second line under default heading margin.

UNDENT Control

Purpose:

The UNDENT control word forces the immediately following line to start further left than the position indicated by the current indent.

Format:

```
+-----------+
|           |
| .UN   n   |
|           |
+-----------+
```

n specifies the number of spaces to be "undented" (negative indent) for the next line only, it must be less than or equal to the amount of indent currently in effect.

Usage:

The UNDENT control word serves the same purpose as the OFFSET control word but in a different manner. It is usually used to make the first line of a paragraph or section extend further to the left than the body of the paragraph.

The choice between using UNDENT and OFFSET is usually a matter of personal preference. In general, UNDENT is more convenient once one becomes familiar with its usage.

Notes:

a. This control acts as a BREAK.

b. It is a very severe error to specify an undentation amount that exceeds the current indentation amount.

Examples:

a. .UN 10
If an indentation of 10 is in effect, the next line will start at the left margin, all following lines will occur at normal indent position, 10 spaces from left margin.


```
.tm 10
.he SCRIPT Example
.ce
SCRIPT Example
.sp 2
.ds
This example will demonstrate some of the capabilities
of the SCRIPT command.  This file was created by issuing:
.br
SCRIPT EDIT EXAMPLE
.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.
```

Figure 3.1.13-A. Contents of a SCRIPT file.

```
PARAGRAPHS:
.br
If not 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 3.1.13-A (cont.) Contents of a SCRIPT file.

SCRIPT Example

This example will demonstrate some of the capabilities of the SCRIPT command. This file was created by issuing:

SCRIPT EDIT EXAMPLE

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 and doing some justification by eye, fancier effects can be obtained. This also takes some practice and experimentation.

CAPTION

Figure3.1.13-A. Contents of a SCRIPT file.

SCRIPT Example

PARAGRAPHS:
If not 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.

Figure3.1.13-A (cont.) Contents of a SCRIPT file.

3.1.14 SPLIT

Purpose:

The SPLIT command copies a specified portion of a card image file and appends it to a second card image file or creates a new file.

Format:

SPLIT filename1 filetype1 filename2 filetype2 { label1 / n1 } [{ label2 / n2 / end-of-file }]

filename1 filetype1	specifies the file from which a portion will be copied.
filename2 filetype2	specifies the name of the file to which file1 will be added.
label1	An 8 byte alphanumeric label with the first character non-numeric specifying the first record to be copied.
label2	An 8 byte alphanumeric label with the first character non-numeric 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 append it to file2. File1 and file2 cannot be the same file. If file2 does not exist, it will be created. The files must have fixed-length records of 80 bytes per record.

Copying begins at either the first record containing the alphanumeric 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 alphanumeric label, copying, once initiated, terminates immediately before the first item having the alphanumeric 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.

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 non-numeric.

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 will terminate the command after printing one of the following responses:

TYPE NOT FOUND
DISK ERROR
ILLEGAL MODE
NONSTANDARD FILE
BUFFER TOO SMALL
OPEN FOR WRITING
OPEN FILE LIMIT
F/V FLAG WRONG (file of variable-length items)

Any error encountered in the writing of file2 will terminate the command after printing one of the following responses:

BAD OUTPUT TYPE
ERROR ON DISK
ITEM SIZE? (items vary in length)
OPEN FOR READ
TOO MANY FILES
DISK FULL
READ ONLY

Examples:

a) SPLIT FILE DATA F1 DATA 45 12
The 12 items beginning with the 45th item are extracted from the file FILE DATA and added to the end of the file F1 DATA if F1 DATA exists or creates the file F1 DATA if it does not already exist.

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 are appended to the file ABLE1 SYSIN if it exists or creates the file ABLE1 SYSIN if it doesn't exist.

c) SPLIT PROG SYSIN PROGEND SYSIN END
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 not exist; 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.

3.1.15 TAPE

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    *          *
          1
TAPE
        REWIND                                    TAP n
        SCAN                                      TAP 2
        SKIP   n
               1
        SLOAD  filename filetype
        WRITEOF
```

DUMP specifies a disk file is to be copied to the mounted tape.

filename filetype <filemode> specify the file to be copied. The filemode is optional and defaults to P1.

* means either 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 to the permanent disk. "n" defaults to 1.

REWIND positions the tape at the load point.

SCAN types out the identifiers of the files that exist on the tape in a tape-dump format.

SKIP n positions the tape after the next "n" tapemarks. "n" defaults to 1.

SLOAD filename filetype scans the tape for "filename filetype" and loads it onto the permanent disk.

TAP n 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. If filemode is omitted, P1 is assumed. Files may contain fixed or variable length records. The specified file is first copies to a disk utility file, then to the tape mounted at virtual device address 181. No end-of-file mark is written after the data transfer unless a TAPE WRITEOF command is issued. 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 tapemarks are encountered, TAPE SCAN terminates. The TAPE SKIP positions the tape after the next "n" tapemark 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 for the specified file and loads it into the permanent disk with mode P1.

The TAPE WRITEOF command writes a tapemark wherever the tape is currently positioned. The tapemark is recognized by TAPE LOAD as end-of-file.

Responses:

a. 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.

207

b. DUMPING...

Disk files are being dumped to tape and their identifiers are typed out as they are dumped.

c. TAPE n NOT READY YET

The tape is attached but not physically in a ready status.

d. (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. Failure to issue a TAPE WRITEOF after the last file copied to tape will cause an error on a subsequent TAPE LOAD.

c. The (EXEC) form of the LISTF command is useful for copying all files, or groups of related files.

d. TAPE DUMP and TAPE LOAD are valid even if the size of 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. The data area contain directory information (see Program Logic Manual).

f. Under OS-67, all tape units must be attached to the CMS user before any tape I/O can occur. Refer to Section 5.5.0 "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
```


208

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 S1
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 will erase 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 S1 on the system disk would not be erased.

Error Messages:

E(CCC01) INVALID COMMAND FORMAT
The format of the command was incorrect, or the operation specified was unknown. Retry the command. No operation was performed.

E(CCCC1) FILE filename filetype filemode NOT FOUND.
The file specified with TAPE DUMP was not found. No operations was performed. Check the file designation and retry the command.

E(CCCC2) FATAL TAPE ERROR WHILE WRITING.
An I/O error occurred which could not be corrected by 10 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(CCCC2) TAP n IS FILE PROTECTED
FATAL TAPE ERROR WHILE WRITING.
The ring is not in the tape, therefore writing can not occur.

E(CCCC2) TAP n NOT ATTACHED
FATAL TAPE ERROR WHILE WRITING.
The tape is not attached, therefore it can not be used. Refer to Tape Procedures in Section 5.5.0 for information on tapes.

E(CCCC2) 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 COMBINE, Note b.

E(CCCC3) 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(CCCC3) 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 COMBINE, Note b.

E(CCCC4) 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(CCCC5) 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(CCCC6) 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.

3.1.16 TXTLIB

Purpose:

The TXTLIB command either (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 / TX}	{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 libname must be TXTLIB.

filename1...filenameN specify the file(s) to be used in the 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 PL/I 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 will be erased, and the new file will be 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 will be 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 FIGURE 3.1.16-A. 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, refer to Sections 3.2.2 and 5.4.0.

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 will be automatically generated by the ASSEMBLE 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 will be performed.

c. The total number of control section names and entry points in the TXTLIB file cannot exceed 256. When this maximum number is reached, an error message will be typed out. The text library created will include 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 will be typed out. See FIGURE 3.1.16-A. The number of entries xxx in the text library will be typed out when either TXTLIB LIST or TXTLIB PRINT is issued.

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 will be erased and the new file created.

b. TXTLIB ADD SSP MYROUT
The contents of the file MYROUT TEXT will be 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 will be deleted. This includes entry points SUBD and SUB1 (see FIGURE 3.1.16-A).

d. TXTLIB PRINT ABC
The file "ABC MAP P1" is generated on the user's permanent disk containing the dictionary information from the file "ABC TXTLIB". The number of entries in "ABC TXTLIB" is typed out.

e. TXTLIB LIST SSP
The dictionary of the file SSP TXTLIB as well as the total number of entry points and control section names will be typed at the terminal.

Error Messages:

(E00001) A FILE IS MISSING
Either the filetype of "libname" is not TXTLIB or the filetype of "filename1...filenameN" is not TEXT.

(E00002) A FILE HAS WRONG RECORD LENGTH.
One of the TEXT files does not have 80 character records and, therefore, it has an incorrect format. Re-compile or re-assemble the source language file and then issue the TXTLIB command again.

(E00003) ERROR WHILE READING.
An error occurred while either the TXTLIB file or a TEXT file was being read. The command has terminated.

(E00004) ERROR WHILE WRITING.
An error occurred while writing the TXTLIB file. The command has terminated.

(E00005) LIBRARY NOT FOUND.
The file "libname TXTLIB" does not exist. The command has terminated.

(E00006) xxxxxxx CAUSED OVERFLOW --- OK UP TO THERE
The maximum number of entry points and control section names (256) in the TXTLIB file has been reached. No additional TEXT files may be added. The control section that caused the overflow is not included in the text library created.

(E00007) ERROR, FILENAME MAY BE .DUMMY
ALTER failed. This means that your original text library is lost but the new version may exist as ..DUMMY TXTLIB.

(E00008) xxxxxx, xxxxxx NOT FOUND .... OTHERWISE OK
A list of csectnames not found in the specified text library is printed above this comment. The presence of such csectnames does not interfere with the deletion of ones that are found.

(E00009) WIPEOUT!!!
All csects have been deleted. No text library is left.

(E000010) CSECTNAMES MISSING
No csect names were specified with TXTLIB DELETE.

(E000011) BAD FORM
Command forms must start with G, A, D, L, or P.

UPDATE

Purpose:

The UPDATE command makes changes in a specified file according to control cards in a second file.

Format:

```
{UPDATE}
{  U   }  filename1 [filetype1 [filename2 [filetype2]]] [(P)]
```

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.
(P)	specified 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.

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  seqno1  increment  label
```

S	specified the new file is to be sequenced in columns 76 through 80. If this card is included in the UPDATE file, it must be the first card.
seqno1	specified 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 through 75.

```
./ D seqno1 seqno2
```

D — specifies cards are to be deleted from the original file.

seqno1 — is the (original) sequence number of the first card to be deleted.

seqno2 — is the sequence number of the last card to be deleted. If omitted, only one card is deleted.

```
./ I seqno1
```

I — specifies cards are to be inserted in the original file. The inserted cards must follow the ./ I card immediately in the UPDATE file. All cards until the next control card are inserted.

seqno1 — specifies the sequence number of the item after which the cards are to be inserted.

```
./ R seqno1 seqno2
```

R — specifies cards are to be inserted in the original file in place of cards now there.

seqno1 — specifies the first card to be replaced.

seqno2 — 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 the filetype UPDATE. It will be 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. "(P)" must always be the last argument, if it is 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 and deleted from the original file, items added 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, depending on whether (P) is specified with the command. 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 on the permanent disk unchanged. The 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. Sequence numbers will be assigned. The control cards must always be identified by a "./" in columns one and two, but any number of blanks may separate the other fields. Sequence numbers may be expressed with up to five digits, but leading zeroes are not necessary. Any sequence numbers in cards to be inserted in the file are ignored. If the ./ S control card is omitted from the UPDATE file, asterisks will be 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 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 resonse 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:

a. 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
       D  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 colums 73 through 75, and sequence numbers beginning with 00100 and incrementing by 25 are placed in columns 76 through 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, enter the UPDATE command again.

3.1.18 TAPRINT

Purpose:

TAPRINT copies files from a specified tape to the offline printer. The files must be LISTING files created by the WRTTAP or ASSEMBLE commands.

Format:

{TAPRINT / TAPR} [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 WRTTAP 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 will be passed to the CMS Command environment.

PERMANENT I/O ERROR ON TAPE.
An I/O error occurred. The command is terminated.

Examples:

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.

3.1.19 WRTTAP

Purpose:

WRTTAP 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:

```
{WRTTAP}  filename filetype
{  W   }
```

filename specifies the file to be copied.
filetype

Usage:

WRTTAP copies files from any disk to TAP1. Records are blocked in groups of ten, and the record format must be 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 WRTTAP 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. WRTTAP 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. WRTTAP 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. WRTTAP expects TAP1, addressed at 180, which should be specified when you request the operator to attach the tape.

c. Tape files written with WRTTAP are not suitable for re-reading with TAPE LOAD.

Responses:

None.

Example:

a. WRTTAP 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 10. 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

WRTTAP 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 reading the file from disk.

E(xxxxx) TAPE ERROR

A tape error occurred which could not be recovered. The error code is unpredictable.

3.1.20 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 non-zero error code will be returned. The mode must be specified as P, T, or S.

Error Codes:

E(00001)	File specified does not exist.
E(00004)	First character of filemode illegal.

3.1.21 TAPEIO

Purpose:

The TAPEIO command either writes a tape mark on a magnetic tape, erases a gap, or moves the tape.

Format:

```
| TAPEIO | function  TAP1, TAP2 > |
```

TAP1 corresponds to device 180.
TAP2 corresponds to device 181. TAP2 is assumed if no tape is specified.

Functions Are:

BSF	to backspace one file
BSR	to backspace on record
ERG	to erase a gap
FSF	to forward space one file
FSR	to forward space 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 as indicated or is rewound and/or unloaded or a tape mark (End-of-file) written for a gap is erased as indicated.

Notes:

1. The modeset is set to X'FF', therefore, either 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 3.4.1.4.11).

3.4.1.4.11-1

2. The REWIND and RUN functions indicate completion before the

physical operation is completed. Thus a subsequent operation to the same physical device may encounter a device busy situation.

3. The TAPEIO command is identical to the TAPEIO function (section 3.4.1.4.-11) 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 message 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 will be received when the tape drive is in a ready status.

(CK - 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 can not 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 a file-protect ring, therefore the tape can not be erased or written on.

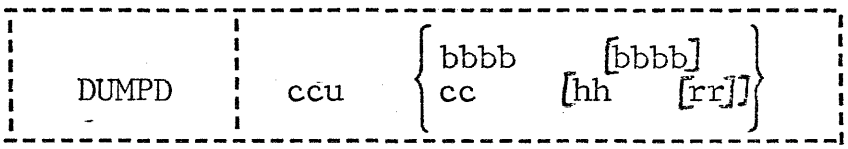
E(00007) TAPn - SERIOUS TAPE ERROR ATTEMPTING function
An unrecoverable tape error has occurred on the tape while attempting the specified function.

3.1.22 DUMPD

PURPOSE:

The DUMPD command prints the contents of one or more direct access records, specified by either a CCHHRR address or a BBBB number, in hexadecimal on the printer.

FORMAT:



ccu	is the device address
cc hh rr	is the cylinder, track, and record number to be printed
bbbb	is the data block number

USAGE:

The contents of the specified record is printed in hexadecimal on the printer. If cc hh rr or bbbb 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. For more than one record:

1. Omit rr for printout of given track (head);
2. Omit hh rr for printout of given cylinder;
3. Specify bbbb bbbb as the beginning and end of the data blocks to be printed.

RESPONSES:

CC HH RR
BBBB

Prompting will depend on the previous mode of input. Specify another cylinder, track (head), and record number or data block number. User may change input mode at any time input is requested. The change will be reflected on the next input prompt.

CORRECT FORM IS: DUMPD UUU CC HH R
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.

```
ERROR IN SPECIFYING RANGE OF BLOCK NUMBERS
CORRECT FORM IS FIRST LAST OF THE RANGE
```

First block number was larger than the second. First parameter must be less than the second parameter.

EXAMPLE:

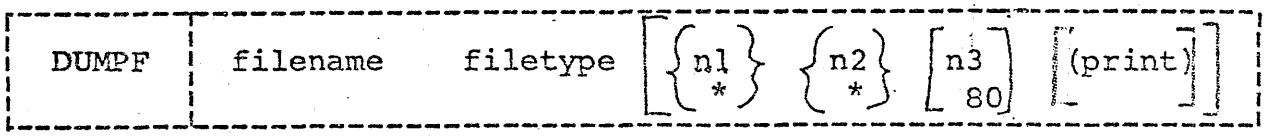
```
dumpd  191  01  02  05
CC HH RR
002C
BBB
03  05  0D
CC HH RR
02  02
CC HH RR
0020  0022
BBBB
cr

R; 10.37.11  T = 0.81/257
```

3.1.23 DUMPF

Purpose:

The DUMPF command dumps the contents of all or part of a specified file in hexadecimal and EBCDIC.

Format:



filename filetype — specify the file to be dumped.

n1 — is the line number of the first line to be output.

n2 — is the line number of the last line to be output.

n3 — is the maximum number of characters to be output on a line, if the records are to be truncated.

print — the output will be sent to the system printer.

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 will be output. An asterisk in the first line or end line fields specifies the beginning or the end of the file, respectively.

Output lines are truncated to the specified limit, if any, or to 80 characters. If a limit is specified, the first line number and last line number fields must be filled, either explicitly or with asterisks.

Output appears at the terminal unless the print option is used.

The search for the file is made on the permanent, temporary, and system disks in that order. In the case of files with duplicate filename and filetype, only the first file found will be typed out.

3.1.24-1 REMOTE

Purpose:

To print or punch CMS files on an OS controlled remote terminal (2780/1130).

Format:

```
REMOTE "command" "destination" filename filetype <filemode>
```

Where command = :

Print:

The specified file will be printed on the remote printer with automatic single spacing.

Printcc:

The specified file will be printed on the remote printer with the first character of each record interpreted as ASA carriage control.

Punch:

The specified file will be punched on the remote punch.

Where destination = :

NY	2780 in New York City-VCS Marketing Office
DC	2780 in Washington DC-SBC,

Filename filetype <filemode>
identify the file to be transferred.
If filemode is omitted, PL is assumed.

Notes:

a. Only fixed length records can be handled
b. Machine carriage control characters are not valid.
c. Punched output will be preceeded by an OS header card. This should be disregarded.

3.1.24-2 REMOTE

Responses:

filename filetype filemode NOW SET UP FOR REMOTE "command" IN "destination". This indicates normal termination of REMOTE

Examples:

a. REMOTE PRINT NY FILE FTO1FOO1. The file FILE FTO1FOO1 P1 will be set up for remote printing on the NY 2780.

b. REMOTE PUNCH DC DATA DA01 P5. The file DATA DA01 P5 will be set up for remote punching on the 2780 in Washington, DC.

Error Messages:

<REMOTE> REMOTE FUNCTION NOT VALID Command specified was not PRINT, PRINTCC, or PUNCH.

filename filetype filemode DOES NOT EXIST ON YOUR FILES

INVALID REMOTE COMMAND

Wrong number of arguments specified

"destination" NOT A VALID LOCATION VALID ONES ARE NY AND DC

YOU MAY NOT PUNCH filename filetype filemode BECAUSE RECORD LENGTH TOO BIG. Punch files must have LRECL = 80 or less.

THIS FILE CANNOT BE REMOTE "command" ED

Record length or other error encountered.

BAD REMOTE MODULE . . .

Error was encountered in the REMOTE module. Contact your marketing support representative.

3.2.C EXECUTION CONTROL

Several commands are available to the user for execution control, i.e. 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 in either relocatable object code form or incore-image form. A program in relocatable object code form is one whose address references can be modified to compensate for the relocation which occurs 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 will be created as a file on the user's permanent disk, and will be 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 will read the specified file(s) from disk and load 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 will create a file on the user's permanent disk which is a copy of the contents of core between the specified locations, and will assign 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 either 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 will be assumed to consist of one or more CMS commands and the EXEC command will be called to execute these commands; (2) if a filetype of MODULE is found, the LOADMOD command will be called to load the file into core and then the START command will be called to begin execution; or (3) if a TEXT filetype is found, the file will be

loaded into core by a LOAD command and then START will be called to begin execution.

The function of the GLOBAL command is to specify two types of libraries: (1) libraries containing TEXT files which are to be searched by the LOAD, USE, or REUSE commands for missing subroutines and undefined names, and (2) 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 prior to the LOAD, USE, REUSE, or ASSEMBLE commands to which it refers.

3.2.1 GENMOD

Purpose:

The GENMOD command is ued to generate non-relocatable core-image files.

Format:

GENMOD G	entry1	<entry2>	(NOMAP) N

entry1 is an entry point or a control section name indating the starting core location from which the core-image copy is to be generated. It will also be the filename assigned to the newly generated file.

entry2 is an entry point or a control section name indicating the ending core location from which the core-image copy is to be generated.

(NOMAP) specifies that a load map is not to be contained in the core-image file.

Usage:

The GENMOD command will cause a file to be created which is a copy of the contents of a specified portion of core. Normally, the LOAD, TSE, 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 will be placed on the user's permanent disk and will be assigned a filename of the first operand specified in the GENMOD command, a filetype of MODULE, and a filemode of P1.

This file will be in core-image form and will be 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 if specified, the core- image file will consist 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, TSE, or REUSE command is issued.)

Before the core-image file is written out, all undefined symbols are defined to location zero and common is initialized.

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 by the GENMOD command.

b. To load into core any files which have been created by the GENMOD command, the LOADMOD command should be used.

c. The MODULE file will contain a load map of the core-image unless (NOMAP) is specified.

d. A MODULE file without a load map normally requires less disk space.

Responses:

None.

Examples:

a. GNEMOD FIRST
Assuming that a file which contains an entry point FIRST has been loaded into core prior to issuing this command, the above example would cause a core-image file to be created on the user's permanent disk. This file would consist of the contents of core from entry point FIRST to the next available load location and a load map and it would have an identifier of FIRST MODULE P1.

b. GENMOD ABC DEF (NOMAP)
This example would create a file on the user's permanent disk with a filename of ABC, a filetype of MODULE, and a filemode of P1. The file will be a copy of the contents of core from entry point ABC to entry point DEF and a load map will not be included in the MODULE file.

Error Messages:

E(CCCC1) NO "entry1" MODULE
This message indicates that the entry point(s) specified cannot be located in core. Check to see that these entry

3.2.2 GLOBAL

Purpose:

GLOBAL specifies either (1) macro definition libraries to be searched during the ASSEMBLE command or (2) text libraries to be searched when loading files containing relocatable object code.

Format:

GLOBAL	ASSEMBLER MACLIB (M) LOADER TXTLIB (T)	<libname1...libnameN>

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.

Usage:

GLOBAL has two forms--the ASSEMBLER form and the LOADER form.

ASSEMBLER Form. The ASSEMBLER form of the GLOBAL command allows the user to specify the macro libraries to be used during the execution of the ASSEMBLE command. One to five macro libraries may be specified. These macro libraries will be searched for macro definitions in the order in which they are named. If the system macro library SYSLIB MACLIB is to be searched along with the user's macro library, SYSLIB must be specified as one 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 (Section 3.1.10).

Unless a previous GLOBAL command has been issued, the ASSEMBLE command will search only one macro library: SYSLIB MACLIB. This file resides on the CMS system disk and contains many macro definitions of common use to CMS users. If the user has created a file named SYSLIB MACLIB that resides on either his permanent or temporary disk, it will be used in place of the system file, since ASSEMBLE searches first the permanent disk, then the temporary disk, and finally the system disk for the specified library file.
To terminate the searching of all macro libraries, including SYSLIB 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

(1) a GLOBAL ASSEMBLER command is re-issued, or

(2) the CMS nucleus is re-initialized, or

(3) the user logs out from CP.

For a further discussion of macro libraries, refer to Section 6.4.0.

LOADER Form. The LOADER form of the GLOBAL command allows the user to specify text libraries to be searched for missing subroutines whenever the LOAD, USE, or REUSE commands are issued. One to four text libraries may be specified. These text libraries will be searched in the order in which they are named. If the system text library SYSLIB TXTLIB is to be searched along with the user's text libraries, SYSLIB must be specified as one of the four 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. (Section 3.1.16).

Without the use of GLOBAL, the LOAD, USE, and REUSE commands search only the text library SYSLIB TXTLIB. This file resides on the CMS system disk and contains many subroutines of use to the CMS user. If the user has created a file with the identifier SYSLIB TXTLIB that resides on either his permanent or temporary disk, it will be used in place of the system file, as the loading process searches first the user's permanent disk, then his temporary disk, and finally the system disk for a specified library file.

If the GLOBAL LOADER command has been issued and the user wishes to eliminate the searching of the previously specified text libraries, the command GLOBAL LOADER TXTLIB can be issued specifying no libnames. This will terminate all library search for the missing subroutines when files are loaded by the LOAD, USE, or REUSE commands.

Once the LOADER form of the GLOBAL command has been issued, the specified TXTLIB files will be automatically searched for missing subroutines during each LOAD, USE, or REUSE until either

(1) a GLOBAL LOADER command is re-issued, or

(2) the option LIBE 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, or

(3) the CMS nucleus is re-initialized, or

(4) the user logs out of CP.

For further discussion on text libraries, refer to Section 6.4.0.

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 four 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 will be generated.

d. "ASSEMBLER MACLIB" and "LOADER TXTLIB" may be abbreviated by "M" and "T", respectively.

Responses:

None.

Examples:

a. GLOBAL ASSEMBLER MACLIB NEWLIB MYMAC
The libraries NEWLIB MACLIB and MYMAC MACLIB will be searched for macro definitions during the ASSEMBLE command. TThe order of search for macro definitions will be NEWLIB MACLIB and then MYMAC MACLIB. The system macro library SYSLIB MACLIB will not be searched.

b. GLOBAL ASSEMBLER MACLIB
This example cancels the effect of any previously issued ASSEMBLER term of the GLOBAL command and causes no library 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 will be searched for missing subroutines during the LOAD, USE, and REUSE commands. The order of search for missing subroutines will be 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 library to be searched for missing subroutines by subsequent LOAD, USE, or REUSE commands.

Error Messages:

E(00001)	An invalid form of the GLOBAL command was issued. Re-issue the command in its correct format.
E(00002)	TOO MANY TXTLIBS (max=4) OR MACLIBS (max=5) SPECIFIED. Re-issue 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.

3.2.3 LOAD

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. 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 | filename1...filenameN <(option1...optionN  libname1...libnameN>) |
```

filename...filenameN	specify the names of TEXT files to be loaded into core.
option1...optionN	specify the options to be in effect during loading.
libname1...libnameN	specify the names of 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 file LOAD MAP online.
NOTYPE	do not type the file LOAD MAP online.
SINV	suppress the printing of invalid card images in the file LOAD MAP
PINV	print invalid card images in the file LOAD MAP.

SREP	suppress the print of Replace card images in the file LOAD MAP.
PREP	print Replace card images in the file LOAD MAP.
LIBE	search only the specified TXTLIB files for missing subroutines.
SYSLIB	search the SYSLIB TXTLIB file for missing subroutines; or, if the GLOBAL LOADER command was previously issued, search the libraries that were specified in the GLOBAL command list.
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 PL/I commands.

Since LOAD assumes the NOCLEAR option as a default, the files that are being loaded will not be 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 has been specified, a load map will be 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 the 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 will normally be 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 will replace any previous LOAD MAP file that exists. To prevent a LOAD MAP file from being created, the option NOMAP must be specified when the LOAD command is issued.

Since LOAD assumes a default of NOTYPE, the LOAD MAP file is not automatically typed out unless option TYPE is specified when LOAD is issued. See FIGURE 3.2.3-A. The LOAD MAP file may also be printed out by using the PRINTF or OFFLINE PRINT command.

If invalid card images exist in the file or files that are being

loaded, they will be listed with the message "INVALID CARD" in the LOAD MAP file. To suppress the listing of invalid card images in the LOAD MAP, the option SINV 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
 FCVLC     AT 13412
 FCV1O     AT 1372C
 FCVCC     AT 13EE4
 FCVAO     AT 1336A
 FCVZC     AT 132E4
 IHCFIOSH  AT 14188
 FICCS     AT 1418E
 IHCUATBL  AT 14D7C
 THE FOLLOWING NAMES ARE UNDEFINED:
      SUB1
E(00004); T=2.33
```

FIGURE 3.2.3-A. Typecut of the LOAD MAP file during the LOAD

If Replace (REP) card images exist in the files being loaded, they will be included in the LOAD MAP. To suppress this listing of REP card images, the option SREP must be specified. For an explanation of REP card images see Section 5.3.0.

Unless the GLOBAL LOADER command has been issued, LOAD searches only the system text library for subroutines that are missing from the files being loaded. The system text library is the file that contains TEXT files and has a filetype of TXTLIB. From one to four text libraries can be searched during loading by the user having previously issued the LOADER form of the GLOBAL command. If the system text library is to be searched in addition to the text libraries specified with the GLOBAL LOADER command, SYSLIB must be included as one of the text libraries in the GLOBAL LOADER command. See the description of the GLOBAL command for the specific details.

If a file has been created on the permanent or

Notes:

temporary disk with the identifier SYSLIB TXTLIB, it will be used in place of the system text library because the order of search for the SYSLIB TXTLIB file is first the permanent disk, then the temporary disk, and finally the system disk.

To prevent LOAD from searching the system text library and the files specified by GLOBAL, the LIBE option can be specified. The LIBE option terminates the searching of all text libraries except for 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 names and they must have TXTLIB filetypes. If the LIBE option is issued andnc TXTLIB files are specified, no TXTLIB files will be searched for missing subroutines--not even SYSLIB TXTLIB. For a discussion of text library usage, refer to Section 5.4.C.

Since LOAD assumes NOXEQ as a default option, LOAD does not normally begin execution of the loaded files. To cause execution to begin immediately upon successful completion of loading, the option XEQ can be specified. LOAD will then transfer control to the default entry point in the program. The default entry point is either (1) the address as specified in the operand field of the first END card image containing a non-blank operand field or (2) the beginning of the first file loaded if all END card images in the TEXT files contained blank operand fields. In the case of TEXT files that are created by FORTRAN, control will be passed to the first main program loaded. If the XEQ option is not specified, the START command must be issued to begin program execution.

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, and the Include Control Section (ICS) card image. The SLC card image specifies the hexadecimal location at which files are to be loaded. The REP card image specifies corrections to be made to the relocatable object code. The ICS card image specifies additions to be made to the TEXT file. For a description of these card images and their use and placement in a TEXT file, refer to Section 5.3.0.

b. The filenames specified must have TEXT filetypes; the libnames must have TXTLIB filetypes.

b. If the LIBE option is specified and the system text library is to be searched, SYSLIB must be specified as one of the libnames.

c. To terminate the searching of all test libraries, including SYSLIB, specify the LIBE option and do not specify any libnames.

Responses:

EXECUTION BEGINS...
The XEQ option was specified with the LOAD command and the loaded program has begun execution. Any further responses will be from the program.

INVALID CARD - xxx...xxx.
The PINV option was 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 was encountered; e.g., ENTRY, LIBRARY.

If the TYPE option is specified with the LOAD command, the LOAD MAP file will be typed out (see FIGURE 3.2.3-A).

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 and the GLOBAL LOADER command has not been previously issued, the file SYSLIB TXTLIB will be searched. If the GLOBAL LOADER command has been previously issued, the libnames specified in that command will be searched. The following default options are set: NOCLEAR, NOTYPE, SIG12000, PINV, PREP, MAP, SYSLIB, 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 and the LOAD MAP file is typed out. Upon the successful completion of loading MPS67 and HOOK, the loaded files will begin execution.

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 the default options.

d. LOAD MASSPEC (LIBE)
The file MASSPEC TEXT is loaded into core and the linkages resolved. If any subroutines are missing, no text libraries will be searched, since the LIBE option was specified without giving any libnames and an error code will be 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 either duplicate entry point names of duplicate control section names. Loading has been completed.

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 that is built during loading. Loading has been completed. Reduce the number of entry points and 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 subroutine, issue the USE command for that file. Loading has been completed.

E(00006) FILE NOT FOUND-xxxxxxxx
A file with a filename of xxxxxxxx does not exist with a filetype of TEXT.

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, (2) or exist in one of the designated libraries with filetype of TXTLIB. The routine may cause other TEXT or library routines to also be loaded into core.

Three CMS/OS macroes may be used to initiate dynamic loading:

ICAD (SVC 8) - will cause 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) - will call in and transfer control to the specified entry point.

XCTL (SVC 7) - will delete the "calling" routine, then the specified routine will be brought in and given control.

RETURN (SVC 3) used to DELETE (SVC 9) the "called" routine and gives control back to the "caller".

3.2.4 LOADMOD

Purpose:

The LOADMOD command loads into core any single file which is in nonrelocatable core image form.

Format:

LOADMOD ICADM	filename <filemode>

filename is the name of the file to be loaded into core, whose 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.

When the LOADMOD command is issued without specifying a filemode, the user's permanent and temporary file directories and the system file directory are searched, in that order, for 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 will be assumed to be in non-relocatable core-image form, and will be loaded into core.

Responses:

None.

Example:

a. LOADMOD FILE1

The file whose filename is FILE1 will be loaded into core, provided it has a filetype of MODULE. If it does not, an error message will be returned, and the loading process will 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 in the user's permanent or temporary file directories or in the system file directory. 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. Attempt to re-issue the command.

E(00003) DISK ERROR.
A disk malfunction has occurred. Re-issue 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 does not begin with P, T, or S. Change the filemode to a valid one, if necessary, and re-issue the command.

E(00006) DISK ERROR.
Either 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. Re-issue the LOADMOD command.

E(00007) DISK ERROR.
The specified file cannot be read from disk. Re-issue the LOADMOD command. If this message persists, the file has probably been written out incorrectly, and should be re-created using the GENMOD command.

E(00009) DISK ERROR.
The specified file is open for writing and cannot currently be read. Re-issue the LOADMOD command.

E(0001C) DISK ERROR.
Eight files are already open, and the specified file cannot be opened. Issue a FINIS command, and then re-issue the LOADMOD command.

points exist and re-issue the command.

E(00002) DISK ERROR.
An address has been generated outside the bounds of core storage assigned to the user. Attempt to re-issue the command.

E(00003) DISK ERROR.
A disk malfunction has occurred. Re-issue the GENMOD command. If the message persists, a disk hardware problem has probably been encountered.

E(00004) DISK ERROR.
An attempt to close the file after writing it out has not been successful. Issue a FINIS and then re-issue the GENMOD command.

E(00005) DISK ERROR.
An illegal second character has been encountered for filemode. Re-issue the GENMOD command.

E(00006) DISK ERROR.
The system has attempted to close the file prior to opening it. Re-issue the GENMOD command.

E(00010) DISK ERROR.
Eight files are already opened, and the command cannot be executed. Issue a FINIS command and then re-issue 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 re-issue the GENMOD command.

3.2.5 REUSE

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.

Format:

REUSE	filename1...filenameN	(option1...optionN) <libname1...libnameN>

filename1...filenameN	specify the names of TEXT files to be loaded into core.
option1...optionN	specify the options to be in effect during loading.
libname1...libnameN	specify the names of 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. Refer to Section 3.2.3.

Usage:

REUSE does not overlay any file that was previously loaded by a LOAD, USE, or REUSE command. It loads the specified files into higher core from the point of 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 REUSE.

The specified files are required to have filetypes of TEXT and should contain relocatable object code.

If options were specified with the previous LOAD, USE, or REUSE command, all the options remain set for REUSE unless each option is specified when REUSE is issued. The LOAD MAP files is automatically updated to reflect the files being loaded by REUSE. Refer to Section 3.2.3 for a description of the LOAD options and the LOAD MAP file.

Responses:

INVALID CARD - xxx...xxx.
The INV option was 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.

CONTROL CARD
A loader or library-search control card was encountered. Normal loading resumes.

If the TYPE option was specified with REUSE or was not reset from the previous LOAD, USE, or REUSE command, the updated portion of the LOAD MAP file will be typed out prior to the completion of the REUSE command.

Example:

1. REUSE READIT GAMMA
The TEXT files of READIT and GAMMA will be loaded into core, linkages will be resolved with the files previously loaded, and the default entry point will be changed to the first entry point in READIT.

Error Messages:

E(00001) DEFINED MORE THAN ONCE - xxxxxxxx
The name has been defined more than once. Check the files that have been loaded for either duplicate entry point names or duplicate control section names. Loading has been completed.

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 that is built during

loading. Loading has been completed. Reduce the number of entry points or control sections in the files.

E(CCC04) 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(CCC05) 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.

E(CCC06) FILE NOT FOUND - xxxxxxxx
A file with a filename of xxxxxxxx does not exist with a filetype of TEXT.

3.2.6 START

Purpose:

START begins execution of programs previously loaded and passes the address of a string of user agruments to that program..

Format:

```
+---------+-------------------------------+
|         |                               |
|  START  |<entry  argument1...argumentN> |
|         |                               |
+---------+-------------------------------+
```

entry specifies the name of a control section or entry point to which control will be passed at execution time.

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 "entry" is not specified, execution begins at the default entry point. The default entry point is either (1) the address as specified in the operand field of the first END card image containing a non-blank operand field or (2) the beginning of the first file loaded if all END card images in the TEXT files contained blank operand fields. The default entry point can be changed by issuing the REUSE command to continue loading additional files. Refer to REUSE in Section 3.2.5.

Any undefined names or references that are specified in the files loaded into core will be defined to location zero. Thus, if there is a call or branch to a subroutine from a main program dn the subroutine was never loaded, the call or branch would transfer 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 in a parameter that as a string of words, on argument per double word, and the address of the parameter that 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 either a control section name or an entry point name. It may not be a filename if the filename is not identical to either a control section name or an entry point name.

b. If user arguments are specified, "entry" must be specified, otherwise, first argument will be taken as the entry point.

Responses:

EXECUTION BEGINS...
The program that had been previously loaded into core has begun execution. The responses now will be from the program that is executing.

Examples:

a. START INITIL
The program that has already been loaded into core will begin executing at the entry point INITIL.

b. START MEGOP 13 A11 109439
The program that has already been loaded into core will begin executing at the entry point MEGOP. The three arguments may be accessed in the program by using displacements of 8, 16, and 24 from the address in general purpose register 1.

Error Messages:

E(COCO5) 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.

3.2.7 USE

Purpose:

USE reads from disk one or more TEXT files -- containing relocatable object code -- and loads them into core, establishing linkages with previously loaded files.

Format:

```
| USE | filename1...filenameN <(option1...optionN)<libname1...libnameN>> |
```

filename1...filenameN	specify the names of TEXT files to be loaded into core.
option1...optionN	specify the options to be in effect during loading.
libname1...libnameN	specify the names of TXTLIB files to be searched for missing routines during loading.

Options:

The options that may be specified with USE are the same as those with LOAD. Refer to Section 3.2.3.

Usage:

USE does not overlay any file that was 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 are required to have filetypes of TEXT and should contain relocatable object code.

USE should be preceded by ICAD; it is normally issued to resolve undefined names when LOAD gives the following error message:

E(00CC4) - 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 were specified with the previous LOAD, USE, or REUSE command, all the options remain set for USE unless each option is respecified when USE is issued. The LOAD MAP file is automatically updated to reflect the files being loaded by USE. Refer to Section 3.2.3 for a description of the LOAD options and the LOAD MAP file.

Messages:

INVALID CARD - xxx...xxx.
The INFO options was 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 the TYPE option was specified with USE or was not reset from the previous LOAD,USE, or REUSE command, the updated portion of the LOAD MAP file will be typed out prior to the completion of the USE command.

Example:

1. USE MYTEXT1 CALCA WRITE6
The files MYTEXT1 TEXT, CALCA TEXT, and WRITE6 TEXT will be loaded into core and the linkages will be resolved between these three files and the files that have previously been loaded into core.

Error Messages:

E(CCC01) DEFINED MORE THAN ONCE - xxxxxxxx
The name xxxxxxxx has been defined more than once. Check the files that have been loaded for either duplicate entry point names or duplicate control section names. Loading has been completed.

E(CCC02) 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(CCC03) REFERENCE TABLE OVERFLOW
There are too many entries for entry points or control section names in the reference table that is build during loading. Loading has been completed. Reduce the number of entry points or control sections in the files.

E(CCC04) 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(CCC06) FILE NOT FOUND - xxxxxxxx
A file with a filename of xxxxxxxx does not exist with a filetype of TEXT.

2.8 $

Purpose:

The $ command will load and start the specified file, provided its filetype is EXEC, MODULE, or TEXT.

Format:

$	filename	[arg1 ... argN]

filename — is the name of a file whose filetype must be either 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 must exist as a file on the user's permanent or temporary disk or on the system disk, and its filename must be specified as the first operand of the $ command. When this command is issued, the user's permanent file directory will be searched for a file with the specified filename and a filetype of either EXEC, MODULE, or TEXT, in that order. If the file is not found in the user's permanent file directory, his temporary file directory and finally the system file directory will be searched.

If an EXEC filetype is found for the filename, the file will be assumed to consist of one or more CMS commands, and the EXEC command will be called to execute this file. If no EXEC filetype exists, but a filetype of MODULE is found, the LOADMOD command will be called by $ to load the program into core and then the START command will be called to begin execution of the program. When only a TEXT filetype exists, the LOAD command will be called, and then the START command.

Optionally, the user may specify as many arguments in the $ command as he wishes, provided they all fit on the same input line. The arguments will be set up as a string of double words, one argument per double word, and the address of this string will be passed to the specified file. Each argument will be left adjusted, and any argument more than eight characters in length will be truncated on the right. With a file whose filetype is EXEC, any arguments specified in the $ command will replace the corresponding &n operands in the individual commands of the EXEC file (see the EXEC command, Section 3.5.3, for a full explanation of this operand-substitution technique).

With a file whose filetype is either MODULE or TEXT, the arguments will be placed in a string as mentioned above, and 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.

Notes:

a. Issuing the $ command for any filename for which an EXEC filetype exists is the same as issuing the EXEC command for that file.

b. When a file whose filetype is MODULE or TEXT is used, there must be an entry point in that file which is identical to the filename specified in the $ command. After the file has been loaded, execution will begin at this entry point. Such an entry point will normally be created by the Fortran compiler using the filename specified in the FORTRAN command (see Output under Section 3.4.2 for exceptions). 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.

Reponses:

With files whose filetype is EXEC, each command in the EXEC file will be typed out at the user's terminal prior to its execution.

EXECUTION BEGINS ...
This message will be typed out after a file with a filetype of MODULE or TEXT has been loaded into core and just before it is started. Any output appearing after this message is given will be from the user's program or from a part of CMS called by that program, and not from the $ command.

DEFINED MORE THAN ONCE - xxxxxxxx
This message, generated by LOAD, indicates that duplicate entry points or control section names (xxxxxxxx) have been found in the TEXT file being loaded. The $ command will be 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. $ will be 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. $ will be terminated with an error code of 3.

THE FOLLOWING NAMES ARE UNDEFINED - xxxxxxxx ... xxxxxxxx
The specified names are referenced in the TEXT file being loaded and are never defined. $ will be terminated with an error code of 3.

DISK ERROR
An error occurred while either reading or closing a file whose filetype is MODULE. This message is generated by LOADMOD, and will cause an error code of 3 to be returned by the $ command.

Examples:

a. $ MYFILE
First the user's permanent file directory, and then the temporary and system file directories, will be searched for a file with a filename of MYFILE and a filetype of EXEC, MODULE, or TEXT.

If a file exists with a filename of MYFILE and a filetype of EXEC, the commands which that file contains will be executed, and each command will be typed out at the user's terminal before it is executed.

If filetypes of both MODULE and TEXT or of MODULE only exist for filename MYFILE, the file whose filetype is MODULE will be loaded into core with a LOADMOD command and started with a START command at entry point MYFILE.

If only a TEXT filetype exists for filename MYFILE, a LOAD will be issued to bring MYFILE into core, and then a START will be issued, using an entry point of MYFILE.

b. $ OTHER SAME 1.436 5 A
If a filetype of EXEC is found on the user's permanent disk for filename OTHER, execution of the EXEC file will take place 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 in the user's permanent directory, but a filetype of either MODULE or TEXT is found, the file will be 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.

If none of the three filetypes is found for filename OTHER in the user's permanent directory, the user's temporary file directory and finally the system file directory will be searched, using the procedure discussed above.

Error Messages:

E(00001) FILENAME NOT GIVEN. COMMAND FORMAT IS:
$ FILENAME OPTIONAL ARGS
The user has issued the $ command without specifying a filename. Re-issue the $ command in the correct format.

E(00002) NO EXEC, MODULE, OR TEXT VERSION OF FILE xxxxxxxx FOUND. $ COMMAND CANNOT BE EXECUTED.
The user has specified a filename which does not have a filetype of EXEC, MODULE, or TEST, where xxxxxxxx is the filename which has been specified. The LISTF command may be used to verify the existence of the specified file and to determine its filetype.

E(00003) LOAD FAILED.
An error code was returned to the $ command by either LOAD or LOADMOD. Check the file for duplicate or undefined symbols, overlay errors, or reference table overflow.

E(00004)
An illegal filemode has been generated by the system when searching for the specified file. Attempt to re-issue the $ command.

E(00005) NAME IS UNDEFINED - xxxxxxxx
The filename specified in the $ command is not an entry point or a control section name in the file which has been loaded. Alter the name of the file or add an entry point so that filename xxxxxxxx will also be a valid entry point, and re-issue the $ command.

3.3.0 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 at his terminal or 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 Section 3.3.2.

In addition to DEBUG, two commands allow the user to trace supervisor calls (SVC instructions) and hence the internal branches which are issued to the various CMS commands and functions. These two commands -- SETOVER and SETERR -- will set certain "overrides", or flags, which will be checked each time an SVC instruction is executed and each time 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 which are 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 causes both types of overrides to be set. The SETERR command will set 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 will cause 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 -- i.e., at a point when the CLROVER command cannot be issued -- he may do so by hitting the ATTN key twice (pausing each time for the keyboard to unlock) and then typing the letters KO followed by a carriage return. Processing will continue as before, but no further information will be recorded for SVC's executed after that point. The KO command will terminate overrides, and will also cause 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 will be cleared automatically, and the recorded information will be 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.

3.3.1 CLROVER

Purpose:

The CLROVER command 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:

Usage:

This command terminates the recording of trace information begun 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 as described in Section 3.3.0. Also, 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 will be cleared automatically and their trace information will be printed offline.

CLROVER will cancel 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 a CLROVER command has been issued, no further trace information will be recorded until the user issues another SETOVER or SETERR command.

A sample of the format in which trace information will be printed offline is given in Figure 3.3.1-A. A fixed amount of trace information is printed for all error overrides; the amount for normal overrides will vary depending on the options specified by the user in the SETOVER command. An explanation of all possible fields which could 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 will have no effect other than causing the following line to be printed on the offline printer:

```
****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW
BEEN CLEARED****
```

b. Any operands given in the CLROVER command will be ignored.

Responses:

None.

Output:

An explanation of each field which can appear 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 will appear in the printout whenever a SETERR command is issued.

SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS. . .

This message will appear in the printout whenever a SETOVER command is issued.

****ERROR-OVERRIDE,

This message identifies the first line of printout for each SVC-called program which returns with an error code in general purpose register 15.

NORMAL-OVERRIDE,

This message identifies the first line of printout for each SVC-called program which issues a normal return.

CALLER = xxxxxxxx

This information will appear in the first line of printout 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 will appear in the first line of printout for each SVC-called program. xxxxxxxx will be either 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 of printout for each SVC-called program, gives the contents of the SVC old program status word. See IBM manual A22-6821, Principles of Operation, for an explanation of the SVC old program status word and its use.

NRMRET = xxxxxxxx

This information, given in the first line of printout for each SVC-called program, gives the hexadecimal core location (xxxxxxxx) to which the program will return under normal conditions; i.e., when no error code is generated.


```
SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

NORMAL-OVERRIDE,   CALLER=00011F30  CALLEE=SETOVER  SVC-OLD-PSW=FF0C0000 00000392  NRMRET=00000A392  ERRET=00000A392
PARM.-LIST =  --SETO--  --VER --  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF
              FFFFFFFF  FFFFFFFF  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF

*****ERROR-OVERRIDE,   CALLER=00011F30  CALLEE=FINIS    SVC-OLD-PSW=FF0C0000 00001F36  NRMRET=00001F36  ERRET=00001F36
GPRS BEFORE = 00000000 00011F30 00001D52 00000000 00000000 00000000 00000000 00000000
              0000A376 00000020 00000000 00000001 00000000 00000000 00000000 00000000
GPRS BEFORE = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
              00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
GPRS AFTER  = 00000000 C5400400 00011F30 00000000 00000000 00000000 00000000 00000000
              C5400400 00011F30 00000000 00000000 00000000 00000000 00000000 00000000
FPRS AFTER  =  --FINI--  --S  --  00000000 --  --  --*  --  01000000 00009A00
PARM.-LIST =  20304170  00054110  00000000 00000000 00000000 00000000 00000000 00000000
              20304170  00054110  00000000 00000000 00000000 00000000 00000000 00000000

NORMAL-OVERRIDE,   CALLER=00009A36  CALLEE=TYPLIN   SVC-OLD-PSW=FF0C0000 00000000  NRMRET=00009A3CC  ERRET=00009A38E
PARM.-LIST =  --TYPL--  --IN  --  20245BF0  23B805EF  54F02230  54F02230  01000000 00001D38
              --LOGI--  --N  --  00000000 00000000 00000000 00000000 00000000 00000000
              --UFD --  --  --  00000000 00000000 00000000 00000000 00000000 00000000

NORMAL-OVERRIDE,   CALLER=00009A3C  CALLEE=WAIT     SVC-OLD-PSW=FF0C0000 00009A3C  NRMRET=00009A3CC  ERRET=00009A38E
              01000A98  C2000010  --  --  --CLOS--  --IO  --
              --UFD --  --  --  --  F6151740  --  --

NORMAL-OVERRIDE,   CALLER=00009A2E  CALLEE=WAIT     SVC-OLD-PSW=61 00009A30  NRMRET=00009A2AE  ERRET=00009A2AE
              01000A98  C2000010  --  --  --CLOS--  --IO  --

NORMAL-OVERRIDE,   CALLER=00009A2E  CALLEE=WAITRD   SVC-OLD-PSW=FF0C0000 00009A30  NRMRET=00009A2E4  ERRET=00009A2F4
PARM.-LIST =  --WAIT--  --RD  --  E5000000  --TYPL--  --TYPL--  01000A5F8  01000A5D1  D2000003
              --TYPE--  --  --  01000A5F1  D2000010  --TYPL--  01000A498  C2000010

NORMAL-OVERRIDE,   CALLER=00011276  CALLEE=TYPLIN   SVC-OLD-PSW=FF0C0000 00011276  NRMRET=00011276  ERRET=00000C108
PARM.-LIST =  --TYPL--  --IN  --  01011418  D2000018  --TYPL--  01011418  C2000018
              --ERAS--  --E  --  --LIST--  --E  --  --ERAS--  D7660000  00000000

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
```

FIGURE 3.3.1-A. Sample offline printout of trace information recorded by the SETOVER command

ERRET = xxxxxxxx
This information, appearing at the right margin of the first line of printout for each SVC-called program, gives the hexadecimal core location (xxxxxxxx) to which the program will return if an error code is generated during its execution.

GPRS BEFORE = xxxxxxxx ... xxxxxxxx
Two lines of information will be given when this message appears in the printout. The first line will consist of the contents of general purpose registers 0-7 and the second line will give 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 will give 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 will be given when this message appears in the printout. The first line will give the contents of general purpose registers 0-7 and the second line will give 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 will give 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 will be followed by either one or two lines of the parameter list which existed at the time the SVC was executed. Refer to Section 5.1.0 for a discussion of parameter lists and their use.

****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
This message will appear in the printout whenever a CLROVER command is issued.

Examples:

a. CLROVER
This example would clear all currently-active overrides and cause any trace information recorded up to that point to be printed on the offline printer. See Figure 3.3.1-A for a sample of the type of information which would be printed.

Error Messages:

None.

DEBUG

Purpose:

The purpose of the DEBUG command is to provide the user with online facilities for debugging his programs and to provide an entry in CMS for handling external and program interrupts and unrecoverable errors.

Format:

Usage:

The facilities of DEBUG are made available to the user when (1) the DEBUG command is issued, (2) an external interrupt occurs, (3) a program interrupt occurs, (4) a breakpoint is encountered during program execution, or (5) 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 has been 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 will be saved so that they may be examined and changed, if desired, prior to being restored when leaving the Debug environment. If DEBUG is entered via an interrupt, the old program status word for that interrupt will also be 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. The KT, KE, and KO commands are not recognized in the Debug environment.

b. Currently the floating point registers may not 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 as described in Sections 4.1.4 and 4.1.13.

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 has been 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 has been saved, and its contents will be typed out 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 will be typed when DEBUG is entered due to a previously-set breakpoint which has been encountered during the execution of a program. The breakpoint will be identified by the number assigned to it (xx) and by the hexadecimal core location (xxxxxx) at which it has been 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, and they must be issued, one per line, in the correct format.

Requests:

Whenever the keyboard is unlocked in the Debug environment, any DEBUG request may be issued. The following rules apply generally 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 will be left adjusted and truncated on the right.

(5) All entries in the DEBUG symbol table are created by issuing the DEF request. Below is a list of all valid DEBUG requests:

BREAK

Format:

BREAK	id	{ symbol / hexloc }

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 the execution of a program at specific instruction locations, called "breakpoints." Issuing the BREAK request will cause 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 will be rejected.

The first operand of the BREAK request specifies the identification number to be assigned to the breakpoint being set, and must be a decimal number between 0 and 15 inclusive. If an identification number is specified which is the same as that of a currently set breakpoint, the previous breakpoint will be cleared and the new one will be 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 will be 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) will be added to the specified operand and the breakpoint will be 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 then replaced by the byte 'Ex', where x is the hexadecimal equivalent of the breakpoint identifier specified in the first operand. In order 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 will occur (since all values E0 through EF are invalid operation codes) and control will be transferred to the Debug environment. In DEBUG the invalid operation code will be recognized as a breakpoint, the original operation code will be replaced, and a message will be typed out identifying the breakpoint which has been encountered.

Figure 3. 3. 2-B indicates the procedure normally used for setting breakpoints. First the program is loaded into core. The DEBUG command is then issued to transfer control to the Debug environment so that breakpoints may be set prior to beginning execution of the program. After the desired breakpoints have been set, a RETURN request should be issued to return control to the CMS Command environment. Issuing the START command will cause program execution to begin. Whenever a breakpoint is encountered, a message to that effect will be typed out, control will return to the Debug environment, and the keyboard will be unlocked to accept any DEBUG request except RETURN. Issuing the GO request will cause program execution to continue from either a specified location or the location at which the breakpoint had been set.

Notes:

a. A breakpoint is cleared whenever it is encountered during program execution.

b. To determine 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 may be obtained in Fortran listings by specifying the LIST option when the FORTRAN command is issued. See Figure 3. 3. 2-A. To obtain a load map, the TYPE option should be specified in the LOAD command, as shown in Figure 3. 3. 2-B.

c. Even 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 will not be 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 will not contain executable code.

c. If a BREAK request is issued which specifies a core address at which a breakpoint is currently active, the second breakpoint will also be set at that same location. When this breakpoint is encountered during execution, the identification number of the most recently set breakpoint will be typed out. The second time this core location is reached during program execution, the identifier of the second-most recently set breakpoint will be typed, and so on. Whenever DEBUG has been entered due to a breakpoint interrupt, issuing the GO request without an operand will cause execution to begin at the location where the breakpoint was encountered. If more than one breakpoint has been set at this location, the additional breakpoint(s) will cause DEBUG to be re-entered.

f. Issuing a RESTART request will cause any breakpoints set at locations greater than hexadecimal 12000 to be cleared; those below 12000 will remain set.

Responses:

If the BREAK request has been correctly issued, the keyboard will be unlocked following a carriage return, and the system will be ready to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that the wrong number of operands have been specified in the BREAK request. Two and only two operands must be specified.

INVALID ARGUMENT
This message indicates that either (1) the breakpoint identification number specified in the first operand is not a decimal number between 0 and 15 inclusive, or (2) 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 either 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 has already been assigned to a currrently active breakpoint. The old breakpoint (at core location xxxxxx) will be cleared and the new breakpoint will be set.

DEBUG ENTERED
BREAKPOINT xx AT xxxxxx
This message will be given whenever a breakpoint is encountered during program execution. xx is the breakpoint identifier and xxxxxx is the core address at which the breakpoint has been encountered. After the message is typed, the keyboard will be unlocked to accept any DEBUG request except RETURN.

Examples:

a. BREAK 1 18C
The current origin value will be added to 18C and the byte at the resulting core address will be saved and replaced by the byte 'E1'. Refer to Figure 3. 3. 2-B where the origin was set to 12000, and the instruction at which breakpoint 1 was set is the second from the last instruction shown in Figure 3. 3. 2-A. Note that the load map indicates that program PRIME was 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 out.

b. BREAK 3 AAA
The byte at the address assigned to symbol AAA will be saved and replaced by the 'E3'. In Figure 3. 3. 2-B, a DEF request is issued which assigns symbol AAA to location 1A4, relative to the current origin of 12000. Breakpoint 3 will therefore set at core address 121A4, as indicated by the message

DEBUG ENTERED
BREAKPOINT 03 AT 0121A4

which is typed out when the breakpoint is encountered during program execution.

pr' f prime listing

```
 FORTRAN IV G LEVEL 0, MOD 0            PRIME              DATE = 67080          1
             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=I
  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          1
 LOCATION   STA NUM     LABEL    OP        OPERAND                  BCD OPERAN
  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
  000168                 A36     L         13,4(0,13)
  00016C                         L         14,12(0,13)
  000170                         LM        2,12,28(13)
  000174                         MVI       12(13),255
  000178                         BCR       15,14
  00017A                 A20     L         15,160(0,13)             IBCOM#
  00017E                         LR        12,13
  000180                         LR        13,4
  000182                         BAL       14,64(0,15)
  000186                         LR        13,12
  000188         1       100     L         15,160(0,13)             IBCOM#
  00018C                         BAL       14,4(0,15)
   00190                         DC        00000006
     :                            :          :
     :                            :          :
```

Figure 3.3.2-A. Sample output created by a FORTRAN command in which the LIST option was specified.


```
load prime (type)
 PRIME#    AT 12000
 PRIME     AT 12000
 IHCFCOMH  AT 122C0
 SAVAREA   AT 13288
 VFIOCS    AT 13128
 IBCOM#    AT 122C0
 FDIOCS#   AT 1237C
 IHCSSQRT  AT 132F8
 SQRT      AT 132F8
 IHCFCVTH  AT 133A8
 ADCON#    AT 133A8
 INTGSW    AT 14409
 FCVEO     AT 13E52
 FCVLO     AT 1362A
 FCVIO     AT 13960
 FCVCO     AT 14054
 FCVAO     AT 1359A
 FCVZO     AT 134F4
 IHCFIOSH  AT 14428
 FIOCS#    AT 14428
 IHCTRCH   AT 14FC0
 IHCUATBL  AT 15238
R; T=0.37

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 3.3.2-B. Sample procedure for setting breakpoints.

CAW

Format:

CAW

Usage:

This request will cause the contents of the channel address word which existed at the time DEBUG was entered to be typed out at the terminal. The channel address word (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 channel address word, located in hexadecimal core location 48, is saved at the time DEBUG is entered. The CAW 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 IBM Manual A22-6821, Principles of Operation.

Responses:

If the request has been correctly issued, the contents of the channel address word will be typed in hexadecimal representation at the terminal and, following a carriage return, the keyboard will be unlocked to accept another DEBUG request. See Figure 3.2.2-F 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. Re-issue the request in its correct format.

CSW

Format:

CSW

Usage:

This request will cause the contents of the channel status word which existed at the time DEBUG was entered to be typed out at the terminal. The channel status word (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 will be stored in hexadecimal core location 40 when an I/O interruption occurs. If I/O interruptions have been suppressed, the CSW will generally be 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 zeroes.
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.

For a further discussion of the channel status word and its use, refer to IBM manual A22-6821, Principles of Operation.

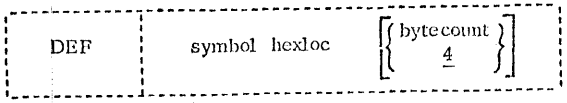
Responses:

If the request has been correctly issued, the contents of the CSW will be typed out at the terminal in hexadecimal representation. A carriage return will then be issued and the keyboard will be unlocked to accept another DEBUG request. For an example of response to the CSW request, see Figure 3.3.2-F.

INVALID DEBUG REQUEST
This response to the CSW request indicates that one or more operands have been specified. Re-issue the request in its proper format.

DEF

Format:



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 symbol 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 and must contain at least one non-numeric character. Also, the first character of the symbol name should not be an asterisk. Any name longer than eight characters will be left-adjusted and truncated on the right.

The second operand of the DEF request specifies a hexadecimal number which will be added to the current origin (as established by the ORIGIN request). The sum of these two values will be the core address to which the symbol name will be assigned.

The third operand of the DEF request 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 will be assigned.

When the DEF request 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 will 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 a DEF request is issued which specifies a symbol that has been previously defined, the previous core address will be replaced by the more recent core address for that symbol in the DEBUG symbol table. DEF requests which specify additional symbol names for core locations to which a symbol name has already been assigned will cause additional entries in the DEBUG symbol table, so that multiple symbols may be assigned to the same core address.

Notes:

a. Only 16 symbols may be defined in the Debug environment at any given time.

b. Issuing a new ORIGIN request will not affect the core address to which an already-defined symbol is assigned.

c. Symbols assigned using the DEF request are defined for use only in the Debug environment. These symbol definitions will not be cleared when a RESTART request is issued.

Responses:

If the DEF request has been correctly issued, a carriage return will be given and the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response to the DEF request indicates that either less than two or more than three operands have been specified. Re-issue the request in its correct format.

INVALID ARGUMENT
This message indicates that either (1) the name specified in the first operand contains all numeric characters, (2) the second operand is not a valid hexadecimal number, or (3) the third operand is not a decimal number between 1 and 56 inclusive.

INVALID CORE-ADDRESS
This response will be 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 then re-issue 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. An existing symbol may be assigned to a new core location, however, by issuing another DEF request for that symbol.

Examples:

a. DEFINE IN1 12F5A

The current origin value will be added to 12F5A and the symbol name IN1 will be assigned to the resulting hexadecimal core address. A length attribute of 4 (the default value) will be assigned to symbol IN1, and an entry for IN1 will be made in the DEBUG symbol table.

b. DEFINE K 13 12

The currently defined origin will be added to the hexadecimal value 13, and the resulting address will be assigned the symbol name K. An entry for K will be made in the DEBUG symbol table, and its length attribute will be 12 bytes, as specified.

.DUMP

Format:

```
DUMP  {ONLINE}  [{symbol 1}  {symbol 2}]
      {ident }  [{hexloc 1 }  {hexloc 2 }]
                [{ 0       }  {  *     }]
                [{         }  {  3     }]
```

ONLINE	indicates that output from the DUMP request is to be typed at the terminal.
ident	indicates that DUMP output is to be printed on the offline printer, and is the name by which that printout will be identified.
symbol 1	is a name assigned (using the DEF request) to the core address at which the dump is to begin.
hexloc 1	is the hexadecimal core location, relative to the current origin, at which the dump is to begin.
symbol 2	is a name assigned (using the DEF request) to the core address at which the dump is to end.
hexloc 2	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 either on the offline printer or at the terminal. If the user specifies "ONLINE" as the first operand of the DUMP request, the contents of the specified core locations will be typed at his terminal; otherwise the information will be printed offline and will have the heading DUMP --- xxxxxxxx, where xxxxxxxx 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 they are not given, the core address specified in the most recent DUMP request will be used or, if no previous DUMP request has been issued, one word (four bytes) of core will be dumped starting at location 0.

If the second and third operands are specified, the core addresses to which they refer are determined as follows: If the second 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 will be 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) will be added to the specified operand. The resulting core address will be 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 will be 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 two lines of output from the DUMP request will give the contents of general purpose registers 0-7 and 8-15 respectively. Thereafter, the contents of the specified portion of core will be given, 32 bytes per line. The core address of the first byte in the line will be given in the left-most column of the dump and will always be on a fullword boundary. This means that the contents of the complete fullword in which the starting (and ending) addresses occur will be included in the dump. All information given in the dump, including the core addresses, will be in hexadecimal format.

Notes:

a. To stop the typeout of an online dump, hit the attention key once to enter the CP environment, and then issue the EXTERNAL console function. This will simulate an external interrupt and will transfer control to DEBUG. The dump will have been canceled and the keyboard will be unlocked to accept any DEBUG request except RETURN.

b. Hitting two attention interrupts during an online dump will not cause the keyboard to unlock. Rather, the dump typeout will continue and the spacing of the interrupted line and the line immediately following it will be incorrect. To stop the typeout of an online dump, use the procedure described in Note a.

Responses:

If ONLINE has been specified, the contents of the general purpose registers and the requested portion of core will be typed out at the terminal, followed by a carriage return, and the keyboard will be unlocked to accept another DEBUG request.

If ONLINE has not been specified, a carriage return will be issued and the keyboard will be unlocked to accept another DEBUG request. The requested information will be printed offline as soon as the printer is free and ready.

INVALID DEBUG REQUEST
Either no, two, or more than three operands have been specified in the DUMP request. Re-issue the request, specifying either one or three operands.

INVALID ARGUMENT
This message will be given if either (1) the address specified by the third operand is less than that specified by the second operand or (2) 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 the ORIGIN request and then re-issue the DUMP request.

Examples:

a. DUMP ONLINE
Since the portion of core to be dumped has not been specified in this request, the last settings specified in a DUMP request would be used or, if no settings have been specified during the current terminal session, four bytes of core will be typed out, starting with location 0. See Figure 3.3.2-C.

b. DUMP ONLINE 12000 END

The contents of core from the address which is the sum of 12000 and the current origin value up to the address identified by the symbol END will be typed out at the terminal. Refer to Figure 3.3.2-C, where the origin value is 0 and the address to which symbol END is assigned is 12093.

c. DUMP NEUC 3456 3548

The contents of core from locations 3456 to 3548 (each plus the current origin) will be printed on the offline printer, and will be identified by the heading DUMP ID --- NEUC. See Figure 3.3.2-D for a sample of this output, where the origin value is 0.

```
dump online
GR 0-7  00000000 00000100 00000100 00000100 00000000 00000520 00005890 00010CC0 00000048
GR 8-F  00000004 00000100 C4C5C2E4 C7404040 0000A012 00010D40 00005454 0000B118

000000  010400CA

def end 12093

dump online 12000 end
GR 0-7  0000000C 00000100 00000100 00000100 00000000 0000C320 00005890 00010CC0 00000048
GR 8-F  00000004 00000100 C4C5C2E4 C7404040 0000A012 00010D40 00005454 0000B118

012000  05C006C0 06C08000 C53850E0 C54CD207 C5500078 58A0C548 48AA0300 50A0C560
012020  58400538 47F0C028 9200C4E2 4190C3D0 4570C124 9640C492 9630C4E2 95C4C492
012040  47800058 943FC4E2 9640C4E2 95D9C492 4780C058 47F0C028 4190C3E0 4570C124
012060  F233C4F0 C4925830 C4E08830 00045030 C558D707 C4F0C4F0 4190C3F0 4570C124
012080  F233C4F0 C4925820 C4F08820 00045020 C55C47F0
```

Figure 3.3.2-C. Examples of the DUMP request using the ONLINE operand.

```
                          DUMP ID ---      NEUC

GR 0-7   0000000C 00000158 00000158 00000000     00000328 00007E68 00011CC0 00000248
GR 8-F   00000004 00000100 C4C6C254 07404040     0000A192 00011040 00007405 00000020

003454   0000047E0 A45641E0 000047E0 A45641E0    000547E0 A45641E0 000047E0 A45641E0
003474   000047E0 A4560200 10008008 0200101F     80240201 10180018 00011014 48700201
003494   1016A870 58F00020 18EF4800 30261002     50F01020 0201101A 80141022 02011024
0034B4   A34645F0 A1CA9200 90601318 58F04886     05EF12FF 4780A136 55F0A84A 4780A7FE
0034D4   47F0A1AA 4010C010 40100000 41200001     40200002 47F0A210 41F00018 47F0A4F6
0034F4   41F0000A 47F0A4F6 41F00011 47F0A4F6     41F00012 47F0A4F6 50200028 10225880
003514   A3924170 80091807 41000322 58800000     91007260 4780A1FE 41200001 8770A1F2
003534   47F0A182 18974220 C0189200 30000766     90019000 50C00060
```

FIGURE 3.3.2-D. Sample DUMP output from the offline printer.

GO

Format:

GO	[{ symbol / hexloc }]

symbol — is a name which has been assigned (using 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 to begin execution at either 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 to become the current PSW when a GO request is issued. If an operand is specified in the GO request, the instruction address portion of the PSW will be altered to contain the address indicated by that operand before the PSW is loaded.

The core address indicated by the GO operand is determined as follows: 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 will be used as the location at which execution is to begin, and will be 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) will be added to the specified operand and the resulting address will be 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 will be restored 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 will then be loaded by DEBUG to become the current PSW, and execution will begin at the specified instruction address.

Notes:

a. The GO request should not be issued without an operand unless the Debug environment has been entered due to a breakpoint interrupt or to an external or program interrupt.

b. When an operand is specified, the GO request 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 should be the core location of an operation code.

Responses:

If the GO request has been issued successfully, a carriage return will be issued and execution will continue from the address contained in the PSW which has been loaded.

INVALID DEBUG REQUEST
This response indicates that more than one operand has been specified in the GO request. Re-issue the GO request in its correct format.

INVALID ARGUMENT
An operand has been specified in the GO request which cannot be located in the DEBUG symbol table and which 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 either 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 of 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 in this case; GO may be issued with an operand; or the RETURN request may be issued if DEBUG had been entered via the DEBUG command.

Examples:

a. GO
The old program status word for the interrupt which caused DEBUG to be entered will be loaded as the current PSW, and execution will begin at the address specified in bits 40-63 of that PSW.

b. GO INN
The DEBUG symbol table will be searched for symbol INN and the address to which that symbol is assigned will be loaded into bits 40-63 of the old PSW prior to loading it as the current PSW. Control will then pass from the Debug environment and execution will begin at the core address to which symbol INN refers.

c. GO 12345
The current origin will be added to location 12345 and the resulting address will be placed in bits 40-63 of the old PSW prior to loading it as the current PSW. Control will then be transferred from the Debug environment and execution will begin at the specified address.

Format:

GPR regl [regN]

regl — is a decimal number from 0-15 inclusive, indicating the first or only general purpose register whose contents is to be typed out.

regN — is a decimal number from 0-15 inclusive, indicating the last general purpose register whose contents is to be typed out.

Usage:

The GPR request may be 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 will be typed out at the terminal. If two operands are given, the contents of the registers specified by the first through the second operand, inclusive, will be typed out.

Both 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 has been correctly issued, the contents of the register(s) specified will be typed at the terminal -- one per line -- and, following a carriage return, the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This message indicates that either none, or more than two, operands have been specified. Re-issue the request in its proper format.

INVALID ARGUMENT
This response will be 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 will be typed out at the terminal. (See Figure 3.3.2-E).

b. GPR 5 15
The contents of general purpose registers 5 through 15, inclusive, will be typed out as they existed when DEBUG was entered. (See Figure 3.3.2-E).


```
gpr 8
002A1A88


gpr 5 15
000A5DD8
002A1A88
00009B38
002A1A88
00000100
80009670
80009FBA
600095C2
00011D40
0000550C
000095B8
```

FIGURE 3.3.2-E. Examples of the GPR request.

IPL

Format:

```
IPL
```

Usage:

The IPL request will cause control to transfer from the Debug environment to the CMS Command environment. IPL may be issued at any time when the keyboard is unlocked in the Debug environment, regardless of the circumstances in which DEBUG has been entered.

Issuing the IPL request will cause a new copy of the CMS nucleus to be brought into core from the system disk. This new copy will overlay 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 INITIALIZED

This response indicates that the IPL request has been successfully executed and that control has passed from the Debug environment to the CMS Command environment. The keyboard will be unlocked, following a carriage return, to accept any CMS command.

INVALID DEBUG REQUEST

This response indicates that one or more operands have been specified in the IPL request. Re-issue the request in its correct format.

KX

Format:

```
KX
```

Usage:

The KX request will close all open files and I/O devices, update the user's file directory, and transfer control from the Debug environment to the CP environment. This request may be issued whenever the keyboard is unlocked in the Debug environment, regardless of the circumstances in which DEBUG has been entered. In order to obtain a new copy of CMS once the KX request has been issued, an IPL console function must be issued in the CP environment.

Responses:

```
KILLING CMS EXECUTION . . .
P-DISK: xxxx RECORDS IN USE,  xxxx  LEFT (of xxxx),
        xx%  FULL (of xxx CYL.)
```

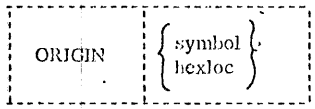
This response indicates that the KX request has been executed and that control has passed to the CP environment. The keyboard will be unlocked to accept any CP console function. The three xxxx fields in the above message indicate the number of 800 byte records on the user's permanent disk area which are in use, free, and assigned, respectively. xx specifies the percent of assigned records in use, and xxx is the number of cylinders assigned to the user.

INVALID DEBUG REQUEST

This response indicates that one or more operands have been specified in the KX request. Re-issue the request in its correct format.

ORIGIN

Format:



symbol	is any name which has been assigned to a core address using 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 will be 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 an ORIGIN request is issued, the origin setting is determined as follows: 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 will become 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 will become the origin setting.

Any origin set by an ORIGIN request will remain 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 will overlay the previous origin setting. If the user obtains a new copy of CMS or issues a RESTART request, the origin will be set to 0 until a new ORIGIN request is issued.

Responses:

If this request has been correctly issued, a carriage return will be given and the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that the wrong number of operands have been specified in the ORIGIN request. 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 will be set to the hexadecimal value of 12000. Thereafter, 12000 will be added to all hexadecimal locations specified in other DEBUG requests and the resulting core address will be referenced.

b. ORIGIN XYZ5
The absolute address assigned to symbol XYZ5 (provided an entry for XYZ5 exists in the DEBUG symbol table) will become the new origin setting. This setting will be added to all hexadecimal locations specified in other DEBUG requests thereafter, and the resulting core address will be referenced.

PSW

Format:

```
PSW
```

Usage:

This request will type out the contents of the old program status word for the interrupt which caused DEBUG to be entered. If DEBUG was entered due to an external interrupt, the PSW request will cause 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 will be typed. If DEBUG was entered for any other reason, the following will be 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 program status word are as follows:

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 rather than running.

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 (unpredicatable 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 IBM manual A22-6821, Principles of Operation.

Responses:

If the request has been correctly issued, the contents of the appropriate program status word will be typed in hexadecimal representation at the terminal, followed by a carriage return and an unlocked keyboard. See Figure 3.3.2-F 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. Re-issue the request in its correct format.

RESTART

Format:

```
RESTART
```

Usage:

The RESTART request is intended for use when an error has been encountered which the user cannot or does not wish to correct, but when the user wishes to retain his current copy of the CMS nucleus and all currently defined symbols and breakpoints below hexadecimal core location 12000. Successful execution of the RESTART request will transfer the user from the Debug environment to the CMS Command environment.

When the RESTART request is issued, an attempt will be made to re-initialize the CMS nucleus in core, i.e., without bringing in a new copy of the nucleus from the system disk. When RESTART is issued the following will occur: (1) the origin will be set to 0 (2) all breakpoints above hexadecimal core location 12000 will be cleared (3) all overrides will be cleared (4) the active file table will be cleared and (5) the last user file directory written to disk will be brought in to replace that currently residing in core.

If RESTART cannot be successfully executed, i.e., if no response is given to the RESTART request and the keyboard remains locked, the user may obtain a new copy of CMS by hitting the attention key once and typing "IPL 190".

Responses:

T = xx.xx
This mesage, followed by a carriage return and an unlocked keyboard, indicates that the RESTART request has been successfully executed, and that control has transferred from DEBUG to the CMS Command environment. xx.xx in the above message is the CPU time (in seconds) which has been used since the last CMS command was issued.

INVALID DEBUG REQUEST
One or more operands have been specified in the RESTART request. Re-issue the request in its correct format.

RETURN

Format:

```
RETURN
```

Usage:

This request is a means of exiting 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 the RETURN request is issued, the general purpose registers are restored with either 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, will be set to zero. A branch is then made to the address contained in register 14, the normal CMS return register. If DEBUG has been entered by issuing the DEBUG command, register 14 will contain the address of a central CMS service routine and control will transfer directly to the CMS Command environment.

Responses:

R; T = xx.xx
The Ready message, followed by a carriage return and an unlocked keyboard, indicates that the RETURN request has been successfully executed and that control has transferred from the Debug environment to the CMS Command environment. The xx.xx portion of the Ready message gives the CPU time, in seconds, used while in the Debug environment. After this message is typed, the keyboard will be unlocked to accept any CMS command.

INVALID DEBUG REQUEST
This message will be given if one or more operands have been specified in the RETURN request. Re-issue the request in its correct format.

INCORRECT DEBUG EXIT
If DEBUG had been entered due to a program or external interrupt or the encounter of a breakpoint or an unrecoverable error, this message will be typed in response to the RETURN request. To exit from the Debug environment under the above circumstances, issue either the GO request with an operand, or the IPL or RESTART requests. The GO request may be issued without an operand if DEBUG has not been entered due to an unrecoverable error.

SET

Format:

SET	CAW	hexinfo			
	CSW	hexinfo	[hexinfo]		
	PSW	hexinfo	[hexinfo]		
	GPR	reg	hexinfo	[hexinfo]	

CAW — indicates that the specified information is to be stored in the channel address word which existed at the time DEBUG was entered.

CSW — indicates that the specified information is to be stored in the channel status word which existed at the time DEBUG was entered.

PSW — indicates that the specified information is to be stored in the old program status word for the interrupt which caused DEBUG to be entered.

GPR — indicates that 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 will be restored when control transfers from DEBUG to another environment; if the register contents have been modified in DEBUG, the changed contents will be restored.

The register which 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 (i.e., 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 will be right-adjusted and the left half of the uneven byte will be set to 0, as shown in Example b. If more than eight hexadecimal digits are specified in a single operand, the information will be left-adjusted, and the additional digits will be truncated on the right and lost. (See Example d).

The number of bytes which 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 of SET, up to eight bytes of information may be stored, but these must be represented in two operands of four bytes each. Whenever two operands of information are specified, the information will be 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 which have been changed using the SET request will not be typed out after the request has been issued. In order to inspect the contents of these registers, the user should issue CAW, CSW, PSW, or GPR requests, as needed. Figure 3.3.2-A contains examples of issuing these requests both before and after the SET request has been issued.

Responses:

If the request has been issued correctly, a carriage return will be issued and the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that the wrong number of operands have been specified in the SET request. If the CAW is being set, two operands must be given. To set the CSW or the PSW, either two or three operands are required. To set a GPR, either three or four operands must be given.

INVALID ARGUMENT
This message indicates that either (1) the first operand is not CAW, CSW, PSW or GPR, (2) the first operand is GPR and the second operand is not a decimal number between 0 and 15 inclusive, or (3) one or more of the "hexinfo" operands does not contain hexadecimal information.

Examples:

a. SET CAW 1100
This example would cause the two-bytes "1100" to be placed in the first two bytes of the channel address word which existed when DEBUG was entered. See Figure 3.3.2-F. This new channel address word will be 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 will be placed in the left-most half-byte, giving 0001. These two bytes, together with the three bytes given in the second operand will be placed as a single five-byte field into the CSW which existed when DEBUG was entered. See Figure 3.3.2-F. This new channel status word will be restored when leaving the DEBUG environment.

c. SET PSW 01000000 00012036

The contents of the entire program status word for the interrupt which caused DEBUG to be entered would be replaced by these eight bytes of information. See Figure 3.3.2-F. This PSW would become the current PSW when an exit is made from the DEBUG environment.

d. SET GPR 5 000012345

In this example, the contents of general purpose register 5 are to be replaced by the information given in the third operand. Since this information is greater than eight hexadecimal digits, it will be left-adjusted and the extra digits will be truncated on the right, giving 00001234. See Figure 3.3.2-F. General purpose register 5 will contain this new information when the general purpose registers are restored prior to leaving the DEBUG environment.


```
caw
00010FE8

set caw 1100

caw
11000FE8



csw
00010FF80C000005

set csw 001 00ff81

csw
000100FF81000005



psw
0100000000009508

set psw 01000000 00012036

psw
0100000000012036



gpr 5
00007F68

set gpr 5 000012345

gpr 5
00001234
```

FIGURE 3.3.2-F. Examples of the SET request, using other requests as appropriate to inspect contents both before and after SET is issued.

STORE

Format:

STORE	{symbol / hexloc}	hexinfo	[hexinfo	[hexinfo]]

symbol — is a name assigned (using 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, which 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 as follows: 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 will be 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 will be added to the specified operand and the resulting core address will be used, provided it is not greater than the user's virtual core size.

The information to be stored must be given in hexadecimal format and is specified in the second through the fourth operands. Each of these operands should be from one to four bytes (i.e., 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 will be right-adjusted and the left half of the uneven byte will be set to 0 as shown in Example b. If more than eight hexadecimal digits are specified in a single operand, the information will be left-adjusted, and the additional digits will be truncated on the right and lost. (See 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 contain four bytes of information. If less than four bytes are specified in any or all of the operands, the information given will be arranged into a string of consecutive bytes, and that string will be stored starting at the location given in the first operand. Stored information will not be typed out at the terminal. To inspect the changed contents of core after a STORE request, the user may issue an X request, as shown in Figure 3.3.2-G.

Responses:

If the request has been correctly issued, a carriage return will be given and the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response to the STORE request indicates that either less than two or more than four operands have been specified. Re-issue the request in its correct format.

INVALID ARGUMENT
This message will be given if either (1) the first operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number, or (2) 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 the ORIGIN request and then re-issue the STORE request.

Examples:

a. STORE 12024 0ACA
This will cause the two bytes of information "0ACA" to be stored at the core address obtained by adding the current origin to location 12024. See Figure 3.3.2-G.

b. STORE XYZ 1341234567890 5CA44 B
Since the second operand in this example contains more than eight digits, it will be truncated on the right, giving 13412345. The third and fourth operands, containing an uneven number of digits, will become 05CA44 and 0B respectively. This eight-byte string will then be stored in the core address indicated by symbol XYZ. See Figure 3.3.2-G.

c. STORE F12 FFFFFFFF FFFFFFFF F0F1F2F3
This example would cause the maximum number of bytes, 12, to be stored at location F12. If F12 is a previously-defined symbol, the information will be stored starting at the address to which that symbol refers. If no symbol F12 has been defined, the current origin will be added to the hexadecimal number F12, and the resulting address will be used as the starting address into which the information will be stored. See Figure 3.3.2-G.

305


```
x 12024
C30640C6

store 12024 0aca

x 12024
0ACA40C6
```

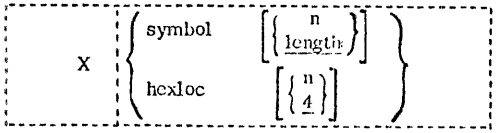
```
x xyz 15
D7D9C9D4C5404090ECD00C9823F028

store xyz 1341234567890 5ca14 h

x xyz 15
13412345O5CA140ECD00C9823F028
```

```
x f12 13
F1FC4370A00141A0A002910FF0

store f12 ffffffff ffffffff f0f1f2f3

x f12 13
FFFFFFFFFFFFFFFFF0F1F2F3F0.
```

FIGURE 3.3.2-G. Examples of the STORE request, using the X request to inspect the contents both before and after storing.

X

Format:



symbol — is a name assigned (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, which 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 will cause contents to be typed out 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 as follows: 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 will be 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) will be added to the specified operand, and the resulting core address will be 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 the X request is optional. If specified, it indicates the number of bytes -- up to a maximum of 56 -- whose contents are to be typed out. If the second operand is omitted, a default value of 4 bytes will be 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 will be used as the number of bytes to be typed out.

Responses:

If the X request has been correctly issued, the information will be typed out and, following a carriage return, the keyboard will be unlocked to accept another DEBUG request.

INVALID DEBUG REQUEST
This response indicates that either no, or more two, operands have been specified in the X request. Re-issue the request in its correct format.

INVALID ARGUMENT
This message is given when either (1) the first operand cannot be located in the DEBUG symbol table and does not constitute a valid hexadecimal number or (2) 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 the ORIGIN request and then re-issue the X request.

Examples:

a. X XYZ
The contents of core starting at the address to which symbol XYZ is assigned will be typed out at the terminal. The number of bytes typed will be determined by the length attribute of symbol XYZ as established in the DEF request for that symbol. See Figure 3.3.2-H.

b. X OTHER 12
Twelve bytes of core will be typed out beginning with the core address to which symbol OTHER has been assigned in a previous DEF request. See Figure 3.3.2-H.

c. X 123
Since no byte count is specified, four bytes of core will be typed out starting at the core address which is the sum of the current origin value and the hexadecimal number 123. See Figure 3.3.2-H.

d. X 123 32
Thirty-two bytes of core will be typed out, starting at the address which is the sum of the current origin value and the hexadecimal number 123. See Figure 3.3.2-H.

```
x xyz
00014020


def other 120

x other 12
D6D922021A0440C5D5C42200


x 123
021A0440


x 123 32
021A0440C5D5C42200000000000000000000000000000000000000000004E0000
```

FIGURE 3.3.2-II. Examples of the X request.

3.3.3 SETERR

Purpose:

The SETERR command is used to trace transfers to and from all SVC-called programs in which error conditions occur.

Format:

```
{SETERR}
{SETE  }
```

Usage:

The SETERR command will "set error" overrides which 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 will be 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, or 16 words, of the parameter list which exists at the time the SVC is executed. See Figure 3.3.3-A for a sample of the type of information which will be recorded by the SETERR command, and the format in which this information would be printed on the offline printer.

The SETERR command will cause the above trace information to be recorded for all SVC-called programs that return error codes, but this information will not be printed out until the user issues a CLROVER or a KO command, issues 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, as noted, will cause the trace information recorded up to the time they are issued to be printed on the offline printer.

```
SETTING ERROR-OVERRIDE TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

*****ERROR-OVERRIDE,  CALLER=00001F30   CALLEE=FINIS    SVC-OLD-PSW=FF  NRMRET=00001F36   ERRET=00001F36
     GPRS BEFORE = 40000000  00001D58  00000000  00009A00  00001D58  00000000  00000000  00000006
     GPRS AFTER  = 00000000  00000000  00000000  00011000  E2E3C1E3  00000006  00000000  FFFE5830
     PARM.-LIST =  --FINI--  --*  --  --  54F023  40103000  50302040

*****ERROR-OVERRIDE,  CALLER=00001F30   CALLEE=FINIS    SVC-OLD-PSW=FF  NRMRET=00001F36   ERRET=00001F36
     GPRS BEFORE = 00000000  00001D58  00000000  00009A00  00001D58  00000000  00000000  00000006
     GPRS AFTER  = 00000000  00000000  00000000  00011000  E2E3C1E3  00000006  00000000  FFFE5830
     PARM.-LIST =  --FINI--  --*  --  --  54F023  40103000  50302040

*****ERROR-OVERRIDE,  CALLER=00001F30   CALLEE=FINIS    SVC-OLD-PSW=FF  NRMRET=00001F36   ERRET=00001F36
     GPRS BEFORE = 00000000  00001D58  00000000  00009A00  00001D58  00000000  00000000  00000006
     GPRS AFTER  = 00000000  00000000  00000000  00011000  E2E3C1E3  00000006  00000000  FFFE5830
     PARM.-LIST =  --FINI--  --5  --  --  54F023  40103000  50302040

*****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED*****
```

FIGURE 3.3.3-A. Sample offline printout of trace information recorded by the SETERR command.

Notes:

a. Issuing one or more SETERR commands when a SETOVER command is active will have no effect, since error as well as normal overrides are set by the SETOVER command.

b. Issuing more than one SETERR command will have 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 will be 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 were issued.

Example:

a. SETERR

This example will cause information such as that given in Figure 3.3.3-A to be recorded for all SVC-called programs which return an error code in general purpose register 15. See Section 3.3.1 for an explanation of each line of the information as it will appear in the offline printout.

Error Messages:

None.

3.3.4 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]  [option 1 ... option N]
{SE      }
```

SAMELAST — leaves all options as set by the user's last SETOVER command; if SAMELAST is not specified, the options will be reset to their default settings. Any options specified after SAMELAST will replace these settings.

option 1 ... option N — 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 is issued.

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 is issued from the SVC-called program.

The default option is that no general purpose register information will be 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 is issued.

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 only as they exist after a return is issued from the SVC-called program.

The default option is that no floating point register information will be 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 that two lines (16 words) of the parameter list will be 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 will be 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 will cause information to be recorded at the appropriate times. Both normal and error overrides, as discussed in Section 3.3.0, will be set by the SETOVER command.

The information that is 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 will be recorded, as well as the contents of the SVC old program status word (i.e., 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 will record the contents of the general purpose and floating point registers, before branching to the SVC-called program and after returning from it, and 16 words of the parameter list which existed when the SVC was issued. (Parameter lists are discussed in Section 3.4.1.1). For normal overrides, the additional information will

depend upon the options, if any, specified by the user in the SETOVER command. If no options are specified, the default options will be used: no general purpose or floating point register information, and 16 words of the parameter list will be recorded for all routines except WAIT. For WAIT only the basic line of information, common to both normal and error overrides, will be recorded. When the user does specify options, the default settings are assumed initially, and options specified by the user will replace the appropriate default settings. If the user specifies SAMELAST as his first option, the options will remain as they were set by the last SETOVER command issued during the terminal session, and any further options given in the SETOVER command will 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 will be adjusted to reflect those specified in the most recent SETOVER command. If WAITSAME is not specified as the first option in these additional commands, the options will be reset to their default settings, and only those options specified by the user in the new SETOVER command will 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 will 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 should be issued as described in Section 3.3.0. If a user issues a RESTART request to the Debug environment or logs out from the Control Program prior to clearing overrides set by issuing the SETOVER and/or SETERR commands, the overrides will be cleared automatically and all recorded trace information will be printed on the offline printer.

Notes:

a. If the SAMELAST option is specified, it must be the first option given.

b. If mutually exclusive options are specified, such as NOPARM and PARM1, the last such option which appears in the SETOVER command will be used. For options which are not mutually exclusive - GPRS and GPRSA, for example - the logical combination of the options will be 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, i.e., the "PARM" setting will govern the "WAIT" setting whenever 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 were issued.

Examples:

a. SETOVER
In addition to the basic information recorded for both normal and error overrides, the above example would cause 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 would be recorded both before branching to the SVC-called program and after returning from it, as would two lines of the parameter list information. (A sample of the type of information which would be recorded is given in Figure 3.3.1-A).

b. SETOVER GPRSB FPRSA NOPARM
For all normal overrides, in addition to the basic information, this example would cause 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. No parameter list information would be recorded. For calls to the WAIT routine, only the basic line of information would be recorded. The same information would be recorded for error overrides as that given in Example a. See Figure 3.3.4-A for a sample of the type of information which would be recorded for this example and the format in which it would be printed.

c. SETOVER SAMELAST PARM1 WAIT2
Assume Example b had been issued prior to this example. Since SAMELAST is specified as the first option, the settings for Example b would be assumed initially. The NOPARM setting, however, would be 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 will be recorded. The same information would be recorded for error overrides as that given in Example a. Refer to Figure 3.3.4-B for a sample of the type of information which would be recorded by this example and the format in which it would be printed.

```
SETTING NORMAL- AND ERROR-OVERRIDES TO PROVIDE A DYNAMIC TRACE OF CMS (AND OS) SVC-CALLS...

    NORMAL-OVERRIDE,   CALLER=0000A38C    CALLEE=SETOVER  SVC-OLD-PSW=FF0000  NRMRET=0000A392   ERRET=0000A392
    GPRS BEFORE = 00000024   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   CCCCCC20    00000000   00000001   0000A012   00000400   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --SETO--   --VER --    --SAME--   --LAST--   --PARM--   --WAIT--   --2  --

*****ERROR-OVERRIDE,   CALLER=0001F38C    CALLEE=FINIS    SVC-OLD-PSW=FF0000  NRMRET=00001F36   ERRET=00001F36
    GPRS BEFORE = 00000024   00001E38    00001D58   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000001   0000A384   00001D58   00000000
    FPRS BEFORE = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    GPRS AFTER  = 00000018   00001F94    0000002C   0002FFFF   FFFF0000   00011000   E2E3C1E3
                  C5404040   00000100    00000F30   0456C440   00005818   00005454   00000006
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --FINI--   --S  --    --*  --   --  --   --*  --   5C40FFFF   FFFE5830
                  20304170   00054110    202456F0   238805EF   54F023A0   40103000   50302048

    NORMAL-OVERRIDE,   CALLER=0000A3C6    CALLEE=TYPLIN   SVC-OLD-PSW=FF0000  NRMRET=0000A3CC   ERRET=0000A3BE
    GPRS BEFORE = 00000000   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000006   0000A012   4000A52A   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --TYPL--   --IN --    01004498   02000010   995E40     F6151740   --  --

    NORMAL-OVERRIDE,   CALLER=0000029AC    CALLEE=WAIT     SVC-OLD-PSW=6  NRMRET=0000029AE   ERRET=0000C108
    PARM.-LIST =  --WAIT--   --  --    --CON1--   00000000   --CON2--   00010C80   00000000

    NORMAL-OVERRIDE,   CALLER=0000029AC    CALLEE=WAIT     SVC-OLD-PSW=61  NRMRET=0000029AE   ERRET=0000C108
    PARM.-LIST =  --WAIT--   --  --    --CON1--   00000000   --CON2--   00010C88   00000000

    NORMAL-OVERRIDE,   CALLER=0000A2EE    CALLEE=WAITRD   SVC-OLD-PSW=FF00  NRMRET=0000A2F4   ERRET=0000A2F4
    GPRS BEFORE = 00000024   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000006   0000A012   4000A52A   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --WAIT--   --2  --    0100A5F2   5800080F   --  --   01003401   02000003

***NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
```

FIGURE 3.3.4-A. 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=0000A38C    CALLEE=SETOVER  SVC-OLD-PSW=FF0000  NRMRET=0000A392   ERRET=0000A392
    GPRS BEFORE = 00000024   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   CCCCCC20    00000000   00000001   0000A012   00000400   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --SETO--   --VER --    --SAME--   --LAST--   --PARM--   --WAIT--   --2  --

*****ERROR-OVERRIDE,   CALLER=0001F38C    CALLEE=FINIS    SVC-OLD-PSW=FF0000  NRMRET=00001F36   ERRET=00001F36
    GPRS BEFORE = 00000024   00001E38    00001D58   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000001   0000A384   00001D58   00000000
    FPRS BEFORE = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    GPRS AFTER  = 00000018   00001F94    0000002C   0002FFFF   FFFF0000   00011000   E2E3C1E3
                  C5404040   00000100    00000F30   0456C440   00005818   00005454   00000006
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --FINI--   --S  --    --*  --   --  --   --*  --   5C40FFFF   FFFE5830
                  20304170   00054110    202456F0   238805EF   54F023A0   40103000   50302048

    NORMAL-OVERRIDE,   CALLER=0000A3C6    CALLEE=TYPLIN   SVC-OLD-PSW=FF0000  NRMRET=0000A3CC   ERRET=0000A38E
    GPRS BEFORE = 00000000   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000006   0000A012   4000A52A   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --TYPL--   --IN --    01004498   02000010   995E40     F6151740   --  --

    NORMAL-OVERRIDE,   CALLER=0000029AC    CALLEE=WAIT     SVC-OLD-PSW=6  NRMRET=0000029AE   ERRET=0000C108
    PARM.-LIST =  --WAIT--   --  --    --CON1--   00000000   --CON2--   00010C80   00000000

    NORMAL-OVERRIDE,   CALLER=0000029AC    CALLEE=WAIT     SVC-OLD-PSW=61  NRMRET=0000029AE   ERRET=0000C108
    PARM.-LIST =  --WAIT--   --  --    --CON1--   00000000   --CON2--   00010C88   00000000

    NORMAL-OVERRIDE,   CALLER=0000A2EE    CALLEE=WAITRD   SVC-OLD-PSW=FF00  NRMRET=0000A2F4   ERRET=0000A2F4
    GPRS BEFORE = 00000024   00001E38    00000000   00000000   00000006   00000028   00009A00
                  0000A376   00000020    00000000   00000006   0000A012   4000A52A   00000000
    FPRS AFTER  = 00000000   00000000    00000000   00000000   01000000   00000000   00000000
    PARM.-LIST =  --WAIT--   --2  --    0100A5F2   5800080F   --  --   01003401   02000003

***NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****
```

FIGURE 3.3.4-B. Offline printout showing trace information recorded by the SETOVER command with the options GAMELAST, PARMIL, and WAIT2 specified. (GAMELAST in this example refers to the options as set in FIGURE 3.3.4-A).

Error Messages:

E(00001) The user has specified one or more invalid options. Re-issue the SETOVER command making sure that all options are spelled correctly and that SAMELAST, if specified, is the first option given.

3.4.0 LANGUAGE PROCESSORS

The language processors supported in CMS are the same ones used under the 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 BACKSPACE statement is not used in CMS Fortran and 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.

The SNOBOL compiler and assembler-interpreter are also included with CMS. They were adapted from programs designed to execute under OS.

321

3.4.1 ASSEMBLE

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 statement containing errors at the terminal, along with diagnostic and error messages.

NODIAG — suppresses typing of errors.

322

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.

REN — checks the program for re-entrance.

NORENT — suppresses the re-entrance check.

Note:

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 will be assembled separately in the order named.

Assembler output is controlled by a set of options. The list of option values 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, LPRINT, PRINT, and LTAPn will be 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 will not be replaced by the current assembly. To save old copies of LISTING and TEXT files, use ALTER to change either their filenames, or the filename of the SYSIN file before assembly. Insufficient space on the permanent disk for any of the assembler files will cause termination with an I/O error message.

"Filename TEXT F1" 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 out in object deck form on the cardpunch with the OFFLINE PUNCH command. If NODECK was specified, this file is not created.

"Filename LISTING F1" 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 was specified. Diagnostics and error messages appear at the bottom of the LISTING file, and, unless NODIAG was specified, are typed at the terminal.

If PRINT or LPRINT was specified, the LISTING file is printed on the offline printer. If LTAPn was specified, the LISTING file is written on the tape whose symbolic address is TAPn, in blocks of ten 121-character records. If LDISK was specified, or if no value was 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 re-issuing the ASSEMBLE command.

The assembler searches the system macro library (SYSLIB MACLIB S1) for macro definitions. Names of macros in this library 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. User-defined libraries are searched before the system library, and the assembler accepts the first definition for a macro it finds, allowing the user to override system macro definitions. If one of the user libraries in the search list has the filename "SYSLIB", the real system library is not searched. See the MACLIB and GLOBAL commands for further information.

Note:

The error completion code returned on termination of an ASSEMBLE command is the highest severity code assigned by the assembler for any of the assemblies performed by that command.

Responses:

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 value LTAPn was specified, and TAPn was not a valid symbolic tape address. The LISTING file is being 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 the ATTN key 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 the ATTN key, and type a carriage return. The assembly will resume 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 the ATTN key and ask the operator to ready the unit. When he replies that the unit is ready, return to CMS with the ATTN key, and type a carriage return. The assembly will begin or resume.

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 F5". If any of the records were successfully written on the specified tape before the error, they will be 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.

Notes:

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", Form A22-6821. The assembler instructions and macro language are described in "Assembler Language", Form C28-6514. Additional information on assembler execution and messages may be found in "Assembler (F) Programmer's Guide", Form C26-3756. 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. No options are specified, so the set of default options governs the assembly. The following files are created:

```
RETURN LISTING P1
RETURN TEXT    P1
```

See Figure 3.4.1-A for an example of this command, showing the on-line diagnostics generated by the assembler. Figure 3.4.1-B shows the LISTING file created.

b. ASSEMBLE RETURN JOBA TEST1 (RENT LEDIT NOXREF NODIAG)
The file "RETURN SYSIN P1" is assembled, and the resulting object code is checked for re-enterability. The listing file is printed out, 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 completed, the same operations will be 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(CCC01) AT LEAST ONE OF THE FILES TO ASSEMBLE HAS INCORRECT RECORD LENGTH.
SYSIN files must have fixed-length, 80-character records.

E(CCC04)
Minor errors were detected during the assembly, but successful execution of the program is still probable.

E(C0008)
Errors were detected in the assembled program, but execution may still be possible.

E(C0012)
Serious errors were detected in the assembled program. Execution is not probably.

E(C0016)
Very serious errors were detected in the assembled program. Execution is impossible.

E(CCC20) 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 will have to be re-entered before assembling it.

E(CCC20)
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.

3.4.1.1 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

```
$ JCB1C 5/7/68  21.37
```

register 1 will point to a parameter list in the following format:

```
PLIST   DS    0D
        DC    CL8'JCB10'
        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 out 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 exceptions, 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 re-stored 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 could be erased during execution of the program by the sequence:

```
        LA    1,PLIST
        SVC   X'CA'
```

After the files were deleted, CMS would return control to the next instruction after the SVC. Register 15 would be set to zero to indicate that no error occurred. Should an error occur, a response would be typed and control passed to DEBUG. Parameter lists for CMS routines which are not commands, such as READF or TYPIN, are not uniform. Complete explanations of these parameter lists may be found in Section 3.4.1.14 CMS Nucleus Routines and in the CMS Program Logic Manual.

To avoid going to DEBUG whenever 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(FERROUT)
        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 FERROUT. Errors may be ignored by the sequence:

```
        SVC   X'CA'
        DC    AL4(*+4)
```

329

Linkage Notes:

a. Commands that are not resident in the CMS nucleus may also be called by the CMS SVC, but they will be loaded at hexadecimal location 12CCC. This is also the default load point for user programs, so a higher load point must be specified when the user's program is loaded. A complete list of the 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 only be executed by branching. The addresses of these programs are listed in a nucleus module under the entry point SYSREF. SYSREF will be 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.

```
USERPGM  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 a terminal line image in core into a standard CMS parameter list. See the CMS Program Logic Manual.)

3.4.1.1 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 will point 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 out 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". If there is a value in register 15, the message is "E(nnnnn); T=n.nn" where nnnnn is the error code returned in register 15 and n.nn is the CPU time in seconds used for execution.

```
JOB13    CSECT
         ENTRY JOB10
         .
JOB10    LR    12,15        NORMAL ENTRY POINT.
         USING JOB10,12     ADDRESSABILITY.
         ST    14,RETNSV    SAVE CMS RETURN.
         LA    1,8(1)       SKIP NAME PARAMETER.
         IC    3,0(1)       PICK UP CODE PARAMETER.
         .
         .
         WRBUF OPUT,ERROR=ERR2
         .                  .
ALLDONE  L     15,ERRCD     GET ERROR CODE, IF ANY.
         L     14,RETNSV    GET RETURN ADDRESS.
         BR    14           BACK TO CMS.
ERR1     LA    15,7
         ST    15,ERRCD     SET ERROR CODE.
         TYPE  'INCORRECT CODE LETTER.'
         B     ALLDONE
ERR2     ST    15,ERRCD     SAVE WRBUF ERROR CODE.
         TYPE  'ERROR WRITING.'
         B     ALLDONE
ERRCD    DC    F'0'
RETNSV   DS    F
```

Figure 3.4.1.1-A. Sample entry and exit sequences in a user program. This program would be kept on disk as JOB10 TEXT P5, and executed by the command $ JOB10 X. On entry, register 1 points to a parameter list with the format:

```
         DC    CL8'JOB10'
         DC    CL8'X'
```

The exit sequence allows for two types of errors. If the parameter supplied were not valid, the following response would be typed:

```
INCORRECT CODE LETTER.
E(00007); T=0.02
```

If an error occurred in WRBUF, the "ERROR WRITING" message would be typed, and the error code would be that returned by WRBUF.

Linkage to CMS Commands and Routines:

With few exceptions, 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:

```
PLST    DS    0D
        DC    CL8'ERASE'
        DC    CL8'*'
        DC    CL8'WORKFILE'
```

In this case, all files with the filetype WORKFILE could be erased during execution of the program by the sequence:

```
        LA    1,PLST
        SVC   X'CA'
```

After the files were deleted, CMS would return control to the next instruction after the SVC. Register 15 would be set to zero to indicate that no error occurred. Should an error occur, a response would be typed and control passed to DEBUG. Parameter lists for CMS routines which are not commands, such as RDBUF or TYPLIN, are not uniform. Complete explanations of these parameter lists may be found in the CMS Program Logic Manual.

To avoid going to DEBUG whenever 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,PLST
        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 may be 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 will be loaded at hexadecimal location 12000. This is also the default load point for user programs, so a higher load point must be specified when the user's program is loaded. A complete list of the 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 only be executed by branching. The addresses of these programs are listed in a nucleus module under the entry point SYSREF. SYSREF will be 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 a terminal line image in core into a standard CMS parameter list. See the CMS Program Logic Manual.)

3.4.1.2 CMS MACROS

The macro definitions that are used by the Cambridge Monitor System are contained in the file SYSLIB MACLIB S1, which resides on the system disk and is called the system macro library. This library contains (1) CMS macros which provide linkages to the CMS I/O routines and (2) System 360 Operating System (OS) 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 which are supported, refer to Section 3.4.1.3. To obtain a list of the names, size, and location of the macro definitions in SYSLIB MACLIB, the command "MACLIB LIST SYSLIB" may be issued. To print out the actual macro definitions, refer to the procedure described in Section 3.1.10.

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 under "Linkage to CMS Commands in Section 3.4.1.1."

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 3.4.1.2-A 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 under Section 3.4.1.1.

b. CMS closes user files on program completion. A user may have only 8 files open at any given time. If it is necessary to use more than 8 files, the FINIS macro may be issued to close some files during program execution. The CMS macros discussed in this section are listed below in alphabetical order.

```
                        IFIQUE  DE
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 3.4.1.2-A. 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 excuted 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-fil 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 out, 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.

FCB

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 fewer characters.
(filename,filetype)	specify the name and type of the file. Mode P5 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 serves as a parameter of the SETUP, RDBUF, and WRBUF macros, and 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 which are being created.

Included in the parameter list generated by FCB is an "item number" field. This specifies which record of the file is being referenced. If not changed, records will be read sequentially from first to last, and written sequentially in 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.

Examples:

a. `INPUT  FCB  (INFILE,DATA),BUFF`

Expansion of this macro generates a parameter list referencing INFILE DATA P5. Before a RDBUF macro can use this FCB for input, the following sequence must be executed:

```
STATE SINPUT,ERROR
SETUP INPUT
```

No initialization would be needed for output. Data will be read into the area labelled BUFF, unless a different area is specified with RDBUF.

339

STATE

Purpose:

The STATE macro provides linkage to a CMS routine which searched 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 will initialize 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 which are being created.

Example:

a. ENTER STATE SA17,ERROUT
This will cause a search of the permanent file directory for the file described by the FCB macro whose label is A17. If it is found, the address will be filled in the FCB, if not control will go to ERROUT.

SETUP

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:

a. SETUP A17
This initializes the File Control Block which has been generated by the macro:

```
A17   FCB  (NAME,TYPE),BUFFER
```

3.5.4 RDBUF

Purpose:

RDBUF is used to read 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 either 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 will override that specified in FCB, if it is different.

Each time RDBUF is executed, it links via a CMS SVC instruction to a routine which will return control when the record requested has been read in. If no error has occurred, register 15 is set to zero on return. If an error, or end-of-file, occurs, register 15 will contain a code indicating the type of error. See Figure 3.4.1.2-B for a list of RDBUF error returns. If the ERROR= was supplied, control goes to that point on an error. If no label was 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
10	eight files open already
11	incorrect number of items
12	end-of-file
13	mode number invalid

Figure 3.4.1.2-B. RDBUF error return codes. Returned in register 15.

Example:

```
a.  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 was preceded by a STATE SINFIL and a SETUP INFIL, this sequence reads successive items from the file ELIPR DATA P5. 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 will go to ERREAD.

CKEOF

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	is 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 whenenver 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 will be typed by CMS only if it is saved and restored around the TYPE macro. If this is done register 15 will be displayed in the error completion response:
E(nnnnn); T=n.nn

WRBUF

Purpose:

The WRBUF macro is used to create, update, or expand disk files.

Format:

```
[label] | WRBUF | fcb [,AREA=alabel] [,ERROR=elabel]
```

label	is an optional statement label.
fcb	is a label referencing the FCB describing the output file.
alabel	is the label of an area from which data is to be written.
elabel	is 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 will contain a code indicating the nature of the error, and control will be returned wherever specified by the ERROR= parameter. If no ERROR= is included, errors are ignored. See Figure 3.4.1.2-C for a list of WRBUF error returns.

Example:

a.
```
        WRBUF A17,AREA=ITEM,ERROR=QUIT
        .
QUIT    BR    14
ITEM    DS    CL80
A17     FCB   (INTR,JBH),BUFF1
```
This sequence writes successive items into INTR JBH P5. On any error, control goes to QUIT, which returns to CMS. The error code will be printed in the CMS error completion response.

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 3.4.1.2-B. RDBUF error return codes. Returned in register 15.

Example:

```
a.  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 was preceded by a STATE SINFIL and a SETUP INFIL, this sequence reads successive items the file ELIPR DATA P5. 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 will go to ERREAD.

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 3.4.1.2-C. WRBUF return codes. Returned in register 15.

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
10	eight files already open
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 3.4.1.2-C. WRBUF return codes. Returned in register 15.

ERASE

Purpose:

The ERASE macro provides linkage to the ERASE command, which will erase 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 which has a filemode of P5.

Notes:

a. To erase a disk file which has a filemode other than P5, the user may set up a parameter list and call the ERASE command directly, as explained under Linkage to CMS Commands in Section 3.4.1.1.

Example:

a. ERASE INPUT
The permanent disk file identified by the FCB labeled INPUT would be erased. For example, if the macro
INPUT FCB (INFILE, DATA), BUFF
had been issued, file INFILE DATA P5 would be erased.

FINIS

Purpose:

The FINIS macro provides linkage to the FINIS command, which will close the specified user file, 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 will close the permanent disk file identified by the FCB whose label is given as an operand of the macro. Since CMS allows only 8 files to be open at one time, this macro must be issued whenever it is necessary to access more than 8 files in a single user program.

Note:

The CMS closes files automatically at program completion. It is only necessary to use the FINIS macro, therefore, when one program accesses more than 8 files.

Example:

a. FINIS INPUT

The permanent disk file identified by the FCB labeled INPUT would be closed. For example, if the macro

INPUT FCB (INFILE, DATA), BUFF

had been issued, file INFILE DATA P5 would be closed.

3/7

TYPE

Purpose:

TYPE generates a parameter list and issues a CMS supervisor call to type a line of terminal output.

Format:

[label]	TYPE	'message' / msglabel (r) / n / length

label is an optional statement label.

'message' is the message to be typed.

msglabel is the label of an area where the message to be typed is located.

(r) is a register containing the length of the message area.

n is a self-defining term giving the length of the message area.

length is 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 will not be known until execution time, a register containing the length may be specified in parentheses.

Notes:

a. Issuing a TYPE macro which specifies 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 out at the terminal.

```
b.          TYPE MSG,LTH
            .
      MSG   DC   C'EXECUTION BEGINS...'
      LTH   EQU  *-MSG
```

The message EXECUTION BEGINS... is typed out.

TYPIN

Purpose:

TYPIN reads a line of input from the terminal.

Format:

```
[label] TYPIN area [,length] [,EDbc].
```

label	is an optional statement label.
area	is the label of a 130-character input buffer.
length	is an optional label to be assigned to a three-byte field of the parameter list where the number of characters read is placed.
c	is 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 in 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 the record has been padded to 130 characters with blanks or not.

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 will be set to blanks (hex '40') if U or S is specified. The X option returns the line exactly as it was typed. U is the default option.

Note:

a. 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.

Example:

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 will be interpreted (and the length adjusted), lower case letters will be translated to upper case, and the buffer behind the line will be 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 means no editing will be performed on the line.

C. REG

Purpose:

The CMSREG macro-instruction 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
```

CMSYSREF

Purpose:

The CMSYSREF macro allows users to branch to routines in the CMS nucleus which cannot be called by CMS SVC's.

Format:

```
CMSYSREF
```

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.

Notes:

a. 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 which saves the current value of the timer for subsequent use, as described in the CMS Program Logic Manual.)

3.4.1.3 OS Macros

The OS macros that are used by the Cambridge Monitor System are contained in the file SYSLIB MACLIB S1, which resides on the system disk and is called the system macro library. The CMS macros are also contained in the library. For a discussion of the CMS macros, refer to Section 3.4.1.2.

To obtain a list of the names, size, and location of the macro definitions in SYSLIB MACLIB, the command "MACLIB LIST SYSLIB" may be issued. To print out the macro definitions, refer to Section 3.1.10.

The OS macros which are supported under CMS are as follows:

ABEND
BLDL
BSP
CHECK
CLOSE
DCB
DCBD
DELETE
DEVTYPE
FIND
FREEMAIN
FREEPOOL
GET
GETMAIN
GETPOOL
LINK
LOAD
NOTE
OPEN
POINT
POST
PUT
PUTX
RD-JFCB
READ
RETURN
SAVE
SPIE
TIME
WAIT
WAITB
WRITE
WTO
WTOB
XCTL

Note that only the forms of the above macros which are used by the OS language processors are supported under CMS. Programs which use OS macros not listed above or use unsupported forms of the above macros will not run correctly under CMS.

The following OS macros are defined in CMS but they do not perform a function as in OS. They are essentially a no-op; they have no meaning in CMS and they 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, "Operation 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".

3.4.1.4 CMS Nucleus Routines

Routines which are contained in the CMS nucleus and 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 are of similar format. 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 GPR1
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. Upon a normal return, register 15 will contain zero. Upon an error return, register 15 will contain an integer 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 will be zero and the following three bytes will be assumed to contain the address of the error return. If the error return address is specified, the normal return will be to four bytes after the SVC instruction; otherwise, it will be 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 will cause 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.

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 which 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 which can be used in further processing. When called as a function, TAPEIO can be used to read or write a tape record. The use of the functions as commands is described under the section on CMS commands.

The PLIST for the following functions are described below:

ATTN	Stack a line for terminal input
CARDPE	Punch cards
CARDRE	Read cards
CCNWAIT	Wait for terminal I/O to finish
CPFUNCTN	Issues CP-67 console functions from CMS
DESBUF	Clear terminal input stack
ERASE	Erase file(s)
FINIS	Close file(s)
HNDINT	Set or clear I/O Interrupt Return Addresses
HNDSVC	Set or clear SVC handling addresses
LOGDISK	Update file directory
PCINT	Set Read or Write pointer
PRINTE	Print line
RDBUF	Read item(s) from disk
STATE	Query file status
TAPEIO	Tape I/O handling
TRAP	Set external interrupt address
TYPE	Write to terminal without carriage return
TYPLIN	Write to terminal with carriage return
WAIT	Wait for interrupt
WAITRE	Read terminal
WRBUF	Write item(s) on disk

3.4.1.4.1 ATTN

Purpose:

The ATTN function stacks a line into the input buffer.

Calling Sequence:

```
PLIST   IS      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 will be 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 instead of terminal input. When the input stack is empty, the keyboard will be unlocked to receive typewriter input.

3.4.1.4.2 CARDPE

Purpose:

The CARDPE function punches a card from the specified 80 byte area.

Calling Sequence:

```
PLIST   IS      0D
        DC      CL8'CARDPE'     routine
        DC      A(buffer)       80 byte area
```

Error Codes:

E(00001)	End of file
E(00002)	Unit Check
E(00003)	Unknown error
E(00004)	Not operational

3.4.1.4.3 CARDRD

Purpose:

The CARDRD function reads a card into the 80 byte area specified.

Calling Sequence:

```
FLIST    IS    0D
         IC    CL8'CARDRD'      routine
         IC    A(buffer)        80 byte area
```

Error Codes:

E(00001)	End of file
E(00002)	Unit check
E(00003)	Unknown error
E(00004)	Not operational

3.4.1.4.4 CONWAIT

Purpose:

The CONWAIT function waits for all stacked reads and writes to "finish" from the console typewriter.

Calling Sequence:

```
FLIST    IS    0D
         IC    CL8'CONWAIT'
         IC    CL4'CON1'
```

Usage:

CONWAIT is called as a standard CMS function. The device id 'CON1' corresponds to the console typewriter.

Error Returns:

None

362

3.4.1.4.5 CPFUNCTN

Purpose:

The CPFUNCTN function transmits console functions to CP-67 without leaving the virtual machine mode.

Calling Sequence:

```
FLIST  DS  0F
       DC  CL8'CPFUNCTN'
       DC  CL8'NO MSG'  This parameter may be omitted
       DC  Cln'CP command string'

       DC  X'FFFFFFFF'   fence
```

Error Codes:

E(00001) No CP command string present

E(00004) INVALID CP REQUEST

E(CCC08) BAD ARGUMENT

E(xxxxx) Any other error codes are from the CP console function specified.

363

3.4.1.6 DESBUF

Purpose:

The DESBUF function clears the input buffer of stacked lines.

Calling Sequence:

```
FLIST     DS      0D
          DC      CL8'DESBUF'
```

Usage:

When DESBUF is called, all input lines previously stacked are deleted, leaving an empty input buffer.

3.4.1.4.7 ERASE

Purpose:

The ERASE function erases the specified file(s).

Calling Sequence:

```
ELIST   IC      CL8'FINIS'      called routine
        IC      CL8'    '       filename or *
        IC      CL8'    '       filetype or *
        IC      CL2'    '       mode or *
```

Error Codes:

E(00001) First character of mode illegal
E(00002) File not found
E(00004) Disk error

3.4.1.4.8 FINIS

Purpose:

The FINIS function closes the specified file(s).

Calling Sequence:

```
ELIST   IC      CL8'FINIS'      called routine
        IC      CL8'    '       filename or *
        IC      CL8'    '       filetype or *
        IC      CL2'    '       mode letter or *
```

Note:

FINIS does not cause the directory to be updated 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

366

3.4.1.4.9 HNDINT

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 to clear such transfer requests.

Calling Sequence:

```
            ES      OP
PLIST       IC      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
```

Error Codes:

B(00001) Incorrect parameter 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 will be 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'=DEFAULT OPTION

Example: IODEV NEWD,387,MYCODE,ASAP,KEEP

Usage:

367

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' are as follows:

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 appropriate 'SIO' given.

3. When 'SIO' is performed satisfactorily, the system-mask can be set to allow all interrupts.

4a. If 'ASAP' was 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 user's program.

4b. If 'ASAP' was not specified, 'IOINT' retains needed information until CMS 'WAIT' function is called.

5. When program 'needs' the interrupt to have been received, CMS 'WAIT' function is called. If interrupt has not yet been received, CMS goes in 'WAIT' state until 'IOINT' fields, and processes the interrupt in normal way.

   If the interrupt has been received and processed (e.g. on 'ASAP'), 'WAIT' returns to caller with necessary internal flags cleared.

   If the interrupt has been received but not yet processed (as under 'WAIT' option instead of 'ASAP'), CMS 'WAIT' now calls IOINT to invoke desired interrupt-handler, then clears needed flags and returns to caller.

6. When finished, using program should normally clear the interrupt-handling scheme thru 'HNDINT CLR' call (unless 'KEEP' option is used and the interrupt-handler remains intact in core).

Purpose:

The HNDSVC function initializes the SVC-Interrupt Handler to transfer control to a given location for a specific SVC number (other that X'CA' or 202), or to clear such previous handling.

Calling Sequence:

```
          DS      0F
PLIST     DC      CL8'HNDSVC'          called routine
          DC      CL4'SET' or CL4'CLR'  function
          DC      AL1(SVC-number),AL3(address  argument)
          .         .
          .         .
          .         .
          DC      X'FFFFFFFF'          end of list
```

Use:

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 by CMS elsewhere.

Notes:

1. For CLR, the address fields are irrelevant.

2. 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 replaces another of the same number
E(00003)	SVC-number clearing one which wasn't set

3.4.1.4.[11] LCGDISK

Purpose:

The LCGDISK function closes all open files and writes the file directory onto the disk.

Calling Sequence:

```
PLIST    DS    CD
         DS    CL8'LOGDISK'
```

Usage:

The user should call LCGDISK during the execution of his program if he wants to cause new or modified files to be permanently written onto the disk before the completion of the program and the return to the CMS command environment.

Error Returns:

None.
If the directory cannot bewritten out, the virtual machine is put in disabled wait state and control will be passed to CP.

3.4.1.4.[12] POINT

Purpose:

The point function sets the read pointer and/or the write pointer at a specified item number.

Calling Sequence:

```
PLIST    DC    CL8'POINT'    called routine
         DC    CL8'      '   filename
         DC    CL8'      '   filetype
         DC    CL2'  '       mode or *
         DC    H'    '       write pointer
         DC    H'    '       read pointer
```

Note:

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 all bits sets it to the last item number plus one.

Error Codes:

```
R(00001)    File specified does not exist
R(00002)    First character mode illegal
```

3.4.1.4.13 PRINTR

Purpose:

The PRINTF function outputs lines on a printer.

Calling Sequence:

```
IC      C18'PRINTR' DC      A(buffer)     buffer area
DC      A(bufsiz)     buffer size
```

Notes:

1. 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

2. Machine CCW OP code carriage controls (as used by the assembler) are also accepted.

3. The buffer size should not exceed 133 bytes. If buffer size is 1, only the carriage control 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

3.4.1.4.14 RDBUF

Purpose:

The RDBUF function reads an item of information from a disk file.

Calling Sequence:

```
PLIST   IC   CL8'RDBUF'     called routine
        IC   CL8' '         filename
        IC   CL8' '         filetype
        IC   CL2' '         mode may be *
        IC   H' '           item number
        IC   A(userarea)    pointer to input buffer
        IC   F' '           number of bytes in buffer
        IC   CL2'F'         fixed-variable flag (F or V)
        IC   H' '           number of items to read
        IC   F' '           number of bytes actually read
```

Error Codes:

E(00001)	File type does not exist
E(00002)	User's memory address is not in his user area
E(00003)	Disk malfunction has occurred
E(00004)	First character of mode illegal
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(00010)	Eight files already open
E(00011)	Number of items greater than 1 for variable-length file
E(00012)	Item number specified does not exist (EOF)
E(00013)	Characteristic not correct

Notes:

1. All errors except error 8 cause the function call to be aborted. On error code 8 that portion of the item that will fit in core is read.
2. The number of bytes in the user area divided by the number of items must be constant (i.e., the same logical record length) for a fixed-length-record file.

3.4.1.4.15 STATE

Purpose:

The STATE function provides a copy of the FST (file status table) entry for the file specified in the parameter list.

Calling Sequence:

```
PLIST    DC   CL8'STATE'     called routine
         DC   CL8'      '     filename
         DC   CL8'      '     filetype
         DC   CL2' '          mode letter or *
         DC   CL2' '          unused
PRETN    DC   A(*)            address of FST copy
                              returned
```

Notes:

1. PRETN will be set to the location of a copy of the file status table entry desired if the file is found.

2. Mode of * means that the first file found with name and type specified will be used. The order of search is permanent, temporary, and system files.

Error Codes:

E(00001) File specified doesn't exist
E(00CC4) First character of mode illegal

3.4.1.4.16 TAPEIO

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	to backspace one file
BSR	to backspace one record
FSF	to forward space one file
FSR	to forward space one record
READ	to read one record
REWIND	to rewind the tape to load point
RUN	to rewind and unload the tape
WRITE	to write one record
WRITEOF	to write a tape mark
WTM	to write a tape mark
ERG	to erase a gap

Device IDs are:

TAP1 or TAP2 corresponding to 180 or 181

The modeset code is one byte of the form:

DDMMM011 with the following interpretation

DD	track density
00	200
01	556
10	800
11	800

MMM		
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 can be used to read, write or move magnetic tape. If the WRITE function is used, the number of bytes indicated will be written. If the READ function is used, information will be moved into the buffer and the number of bytes read will be stored into COUNT. If a tape mark is read, the function will return with an error code of 2.

Notes:

1. A mode set of X'0C' causes the default mode bit of X'B3' to be used.

2. A mode set of X'E3' indicates density 800, parity odd, converter off, and translator off.

3. A mode of X'93' indicates density 800, parity odd, converter on, and translator off.

Messages:

TAPn NOT READY YET
The tape has been attached but it is not in a ready status yet.

(OK - READY NOW)
The tape is now in a ready status 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 Section 5.5.0 Tape Procedures.

E(00006) TAPEn IS FILE PROTECTED .
The tape contains a file-protect ring, therefore the tape can not be written upon.

E(00007) TAPn - SERIOUS TAPE ERROR ATTEMPTING function
An unrecoverable tape error has occurred while attempting the specified function.

3.4.1.4.17 TRAP

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" will be the location transferred to on an external interrupt. If the parameter "trapsubr" is a zero, the return will be 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.

3.4.1.4.18 TYPE

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 E 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:
E = move line to free storage before typing.
K = type line from the specified location.

Notes:

1. The output message must be from 1 through 130 bytes in length.

Error Codes:

E(00001) Invalid terminal number
E(00002) Length of output message not between 1 and 130 bytes

380

3.4.1.4.19 TYPLIN

Purpose:

The TYPLIN function types an output line on the console. Terminal blanks (if any) are deleted, and an automatic carriage return is added.

Calling Sequence:

```
        DS    0F
PLIST   DC    CL8'TYPLIN'
        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'actual message to be typed with
                carriage return added'
EMSG    EQU   *
```

where the write codes are:

B = move line to free storage before typing.
K = type line from the specified location.

Notes:

1. Output line must be 1 through 130 bytes in length (unless byte count is 0, which forces typing a single carriage return).
2. No differentiation is made between red-and-black-ink typeouts.

Error Code:

E(00001) Invalid terminal number

381

3.4.1.4.20 WAIT

Purpose:

The WAIT function awaits an interrupt from one of the specified devices.

Calling Sequence:

```
        DS    0F
PLIST   DC    CL8'WAIT'
        DC    CL4'device id'
        .
        .
        .
        DC    F'0'
INTDEV  DS    F
```

where the device ids are: CON1, DSK1, PCH1, RDR1, PRN1, TAP1, and TAP2

Usage:

When one of the specified devices causes an interrupt, the device id will be stored in the word INTDEV and control will return to the calling program.

Error Returns:

E(00001) Invalid device id specified.

3.4.1.4.21 WAITRD

382

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:

```
            IS      0F
PLIST       IC      CL8'WAITRD'
            IC      AL1(1)          terminal number
            IC      AL3(INPBUF)     address of 130-byte
                                     input buffer
            IC      C 'code'        U, V, S, T, or X
                                     (see below)
            IC      AL3(*-*)        byte count of input
                                     message is stored
                                     here
            .
            .
            .
INPBUF      DS      130C            130 byte (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 (there is no upper case translation)
'T' performs editing only
'X' leaves input line exactly as is

Notes:

1. If the user has 'stacked' input commands, WAITRD will accept the first 'stacked' input message, and move it to the specified buffer.

2. The input buffer is zero-filled before read is initialized, and must be at least 130 bytes long.

383

Error Codes:

E(00001) Invalid terminal number
E(00002) 'Read-type' invalid (not U, V, S, T, or X)

3.4.1.4.22 WRBUF

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 will be opened and assigned the given name and type. WRBUF will automatically pack fixed length items in an 800 byte buffer, and write this 800 byte block onto the disk.

Calling Sequence:

```
ELIST   DC    CL8'WRBUF'
        DC    CL8'    '       filename
        DC    CL8'    '       filetype
        DC    CL2'    '       mode must be specified
                                not *
        DC    H'      '       item number
        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
```

Error Codes:

E(00001) File name not specified
E(00002) User memory address not in user area
E(00003) Disk malfunction
E(00004) First character mode illegal
E(00005) Second character mode illegal
E(00006) Core space not available
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) Eight files already open
E(00011) F-V flag not F or V
E(00012) Mode SY(system) is illegal
E(00013) Disk already full
E(00014) Attempt to write on read-only file
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) Number of items equal zero

Note:

The number of bytes in the user area divided by the number of items must be constant (i.e., the same logical record length) for a fixed-length-record file.

3.4.2 FORTRAN

Purpose:

The FORTRAN command compiles programs written in FORTRAN IV into machine code, and provides program listings and diagnostics.

Format:

```
{FORTRAN}
{   F   } filename1...filenameN [(option1...optionN)]
```

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 will generate one object deck and one listing, replacing any previous output files for the same program.

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 SOURCE 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, typed out at the terminal. The compiler error messages have two formats, depending on when the error is detected (see Figure 3.4.2-B). 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.

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 (SOURCE NOMAP NOLIST) the LISTING file contains the source program statements, diagnostic messages, and a statement of object program size in bytes (see Figure 3.4.2-D). If NOSOURCE is specified, the source program statements are suppressed. However, 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 LISTING 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 is being compiled, the GO option should be specified to prevent an error termination of one compilation deleting all compilations requested.

1 erences:

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, Form C28-6515, and FORTRAN IV Library Subprograms, Form C28-6596. For information on compiler operation and messages, see FORTRAN IV (G) Programmer's Guide, Form C28-6639. Information in the Programmer's 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 3.4.2-A
A file, FAC FORTRAN P5, is to be compiled. No options are specified, so the set of default options governs the compilation. At statement 0011, the compiler detects two errors, at the points indicated by the $ in the succeeding line. Error 01) refers to the left $, error 02) to the $ at the end of the line. Explanations of the error codes are found in the FORTRAN Programmer's Guide. CMS cancels the compilation and supplies the error code E(00032).

b. see Figure 3.4.2-B
The same file as in (a) is to be compiled, but the GO option is specified. The same errors in statement 11 are detected, but processing continues. Another error, an undefined label (40), is found before processing is completed. The completion code, E(00004), is the error severity code assigned by the compiler.

c. see Figure 3.4.2-C
Three files are to be compiled: FAC FORTRAN P5, SUB1 FORTRAN P5, and SUB2 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.

d. see Figure 3.4.2-D
An example of a LISTING file generated under the SOURCE MAP NOLIST options.

```
fortran fac
   0011              IF(J) 300 200 100                                        FAC00110
                                    $                                                $
          01)  ERR 05  ILLEGAL LABEL          02)  ERR 13  SYNTAX
COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
E(00032); T=4.17
```

Figure 3.4.2-A

```
fortran fac (go)
   0011              IF(J) 300 200 100                                        FAC00110
                                    $                                                $
          01)  ERR 05  ILLEGAL LABEL          02)  ERR 13  SYNTAX
                       ERR 22       UNDEFINED LABEL
     40
E(00004); T=6.07
```

Figure 3.4.2-B

```
fortran fac sub1 sub2 (go map print)
R; T=6.25
```

Figure 3.4.2-C

```
printf fac listing
  FORTRAN IV G LEVEL 0, MOD 0          FAC              DATE = 67080        14/39/57          PAGE 000
FILE FAC                                    CAMBRIDGE  MONITOR  SYSTEM
   0001              REAL*8 X,Z/.9D74/                                          FAC00010
   0002            1 FORMAT (G7.4)                                              FAC00020
   0003            2 FORMAT (' FACTORIAL IS ',G16.9)                            FAC00030
   0004            3 FORMAT (' INVALID ARGUMENT')                               FAC00040
   0005            4 FORMAT (' OVERFLOW ON',I4,'TH ITERATION.')                 FAC00050
   0006            5 FORMAT (' ENTER REAL NUMBERS SINGLY, OR 0 TO QUIT.')       FAC00060
   0007          100 WRITE (6,5)                                                FAC00070
   0008          105 READ (5,1) A                                               FAC00080
   0009              J=A                                                        FAC00090
   0010              X=.1D1                                                     FAC00100
   0011              IF (J) 300,200,110                                         FAC00110
   0012          110 DO 120 I=1,J                                               FAC00120
   0013              X=X*I                                                      FAC00130
   0014              IF (Z.LE.X) GOTO 400                                       FAC00140
   0015          120 CONTINUE                                                   FAC00150
   0016              WRITE (6,2) X                                              FAC00160
   0017              GOTO 105                                                   FAC00170
   0018          200 STOP                                                       FAC00180
   0019          300 PRINT 3                                                    FAC00190
   0020              GOTO 100                                                   FAC00200
   0021          400 WRITE (6,4) I                                              FAC00210
   0022              GOTO 100                                                   FAC00220
   0023              END                                                        FAC00230
  FORTRAN IV G LEVEL 0, MOD 0          FAC              DATE = 67080        14/39/57          PAGE 000
FILE FAC                                    CAMBRIDGE  MONITOR  SYSTEM

                                SCALAR MAP
SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LO
Z           C8        X           D0        A           D8        J           DC        I

                            SUBPROGRAMS CALLED
SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LO
IBCOM#      E4

                            FORMAT STATEMENT MAP
SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LOCATION    SYMBOL    LO
  1         E8          2         ED          3         102         4         117         5

   TOTAL MEMORY REQUIREMENTS 0002E6 BYTES
R; T=1.30
```

Figure 3.4.2-D

Error Messages:

E(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 will truncate 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(00012) COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
CMS cancelled processing. Correct the source program, or specify the GO option.

FORTRAN PROGRAMMING

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 through 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 will be P5. For input to a FORTRAN program, any appropriate filemode 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 will be created by the FORTRAN program, and need not be defined before execution.

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 will be 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 P5". 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 will 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 3.4.2-E.

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 will refer to a file with a reference number one larger. If the latest operation on a logical unit was a read (write), then an END FILE to that logical unit will cause all records to be erased on the current logical file of that unit that are beyond the last one read (written). The REWIND statement "repositions" the files to the first one used in that program under that data set reference number. See Figure 3.4.2-F. The BACKSPACE statement is not supported under CMS.

	Data Set Ref. No.	External Filetype	Record Length	Internal Reference
	1	FT01Fyyy	80	
	2	FT02Fyyy	80	READ (a,b) list*
	3	FT03Fyyy	80	WRITE (a,b) list 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	---	133	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
	15	FT15Fyyy	80	same as 1-4
	16	FT16Fyyy	80	same as 1-4
	17	FT17Fyyy	80	same as 1-4
	18	FT18Fyyy	80	same as 1-4
	19	FT19Fyyy	80	same as 1-4
	20	FT20Fyyy	80	same as 1-4

Data Set Ref. No.	External Filetype	Record Length	Internal Reference
21	FT21Fyyy	80	same as 1-4
22	FT22Fyyy	80	same as 1-4
23	FT23Fyyy	80	same as 1-4
24	FT24Fyyy	80	same as 1-4
25	FT25Fyyy	80	same as 1-4
26	FT26Fyyy	80	same as 1-4
27	FT27Fyyy	80	same as 1-4
28	FT28Fyyy	80	same as 1-4
29	FT29Fyyy	80	same as 1-4
30	FT30Fyyy	80	same as 1-4

*a is the data set reference number.

b is the FORMAT statement number.

list is a series of variable or array names.

See the FORTRAN IV Language publication for additional acceptable I/O statement formats

Figure 3.4.2-E. Summary of record formats and I/O statements for sequential FORTRAN files.

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 will 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 3.4.2-E.

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 will refer 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 3.4.2-F. The BACKSPACE statement is not supported under CMS.

	Data Set Ref. No.	External Filetype	Record Length	Internal Reference
	1	FT01Fyyy	80	
	2	FT02Fyyy	80	READ (a, b) list*
	3	FT03Fyyy	80	WRITE (a, b) list 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	--	133	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.

list is a series of variable or array names.

See the FORTRAN IV Language publication for additional acceptable I/O statement formats

Figure 3.4.2-E. Summary of record formats and I/O statements for sequential FORTRAN files.

Notes:

1. 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.

2. The sequence number is "001" except when multiple files are referenced under the same data set reference number.

3. 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 TYPE MODE
REAL*8 B(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 P5
30 END FILE 3	FILE FT03F001 P5
40 READ(4,100,END=50) B	FILE FT04F001 P1
.	
.	
44 WRITE(3,100) B	FILE FT03F002 P5
GO TO 40	
50 END FILE 3	FILE FT03F002 P5
52 REWIND 3	
.	
.	
75 READ(3,100) B	FILE FT03F001 P5
.	
82 PRINT 200,B	(terminal output)
.	
STOP	

Figure 3.4.2-F. Files referenced by sequential FORTRAN I/O statements.

Note:

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 will allow 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 will be P5. For input to a FORTRAN program, any appropriate filemode is acceptable.

"Direct access" refers only to those files which are used with the DEFINE FILE statement. 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 will be 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 will actually slow 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 3.4.2-G.

FORTRAN STATEMENT	FILE- NAME TYPE MODE
100 FORMAT(I6,2X,A20,4G13.6)	
DEFINE FILE 1(200,80,E,R3)	
.	
.	
5 READ(1'J,100) MNNO,NAME,TOTS	
.	
30 WRITE(1'J,100) MNNO,NAME,NWTOTS	FILE DA01 P1
.	
STOP	

Figure 3.4.2-G. File referenced by direct access FORTRAN I/O statements.

Note:
The DEFINE FILE statement describes a file of 200 80-character records. If this file had not existed prior to program execution, 200 records of 80 blanks would have been written on the permanent disk. Statements 5 and 30 read and write the J'th record, where J has been assigned an integer value less than 201.

Additional Notes:

1. 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.

2. Since different FORTRAN programs using the same data set reference number will 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 before and after execution. An example of such a file might be MYSTRUP EXEC P5, containing:

```
ALTER MSTR LIST P1 FILE DA02 P1
ALTER CHANGES LIST P1 FILE FT04F001 P1
$ FORTMSTR
ALTER FILE FT04F001 P1 CHANGES DONE P1
ALTER FILE DA02 P1 MSTR LIST P1
```

The command "$ MYSTRUP" will then rename the files, execute the FORTRAN program, and change the identifiers again on completion.

Additional Notes (continued)

2. 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 will be included in the listing file. If a listing file is produced, the diagnostics will be included. If only the option NODIAG is specified, a listing file will be produced containing only the FORTRAN diagnostics. The PRINT option directs the listing to the printer and will print only those parts of the listing requested by SOURCE, MAP, or LIST. If PRINT is the only option specified, only the diagnostics will be printed.

3. Each FORTRAN file can contain any number of routines to be computed.

4. The characters appearing in columns 73-76 of TEXT files generated by FORTRAN are as follows and 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.

5. The letters used for columns 73-76 of a TEXT file will also appear in the middle of the first line of each LISTING page. In addition, the name of the file will appear at the beginning of the second line of each page in the LISTING file.

6. The FORTRAN logical files which are defined are as follows:

Logical Files	Record Length
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 the logical unit 8 file generated as FILE FT 08F001, the OFFLINE PRINTCC command must be used.

7. The subroutine DSDSET can be called to enable a user to change the record format and the logical record length for FORTRAN disk files. Thus, a user will no longer be confined to records of 80 or 133 bytes in length. Refer to section 3.7.1.7 for a description of the DSDSET routine.

8. A facility for accessing disk files from a FORTRAN program as a direct access file has been provided. This provided an efficient way to update records in place and to access records randomly. A call to the DEFINE routine is required to define the correspondence between a CMS file and a FORTRAN data set reference number. Refer to the description of the DEFINE routine (3.7.1.6).

11. 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. 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, e.g. CALL REREAD (99,80). Refer to the subroutine description in section 3.7.1.5. To read the record from the reread unit a second or subsequent time, a REWIND n or BACKSPACE n statement must be executed before the READ statement. If a reread is issued without executing a REWIND or BACKSPACE statement, an END OF FILE condition will result. 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.

12. A CMS namelist facility is available for obtaining input to a FORTRAN program in free format without specifying variable names. Refer to section 3.7.1.9 CPNMON/CPNMOF for a more detailed description of this namelist facility.

13. 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 section 3.7.1.6 for the DEFINE subroutine.

14. FORTRAN has the ability to read a file from the T-disk during execution of a program if the file does not exist on the P-disk.

9. 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

   a) write onto logical file 1 and issue an END FILE x, or

   b) read the file until the end file condition is reached

   then write into logical file 2 which will append to the data on the end of the existing file.

   After a rewind to a logical unit, subsequent writes will overwrite the existing file. Only logical file 1, i.e., 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 will remain in the file. Writing over an existing file can lengthen the file and thus eliminates 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.

10. 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 section 3.7.1.1 for a description of the TAPSET routine.

   Note: TAP3 and TAP4 are not currently defined in CMS, therefore do not use logical units 13 and 14.

11. A FORTRAN reread facility is available to perform an 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. 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, e.g., CALL REREAD(99, 80). Refer to the subroutine description in section 3.7.1.5. To read the record from the reread unit a second or subsequent time, a REWIND n or BACKSPACE n statement must be executed before the READ statement. If a reread is issued without executing a REWIND or BACKSPACE statement, an END OF FILE condition will result. 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.

12. A CMS namelist facility is available for obtaining input to a FORTRAN program in free format without specifying variable names. Refer to section 3.7.1.9 CPNMON/CPNMOF for a more detailed description of this namelist facility.

13. 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 section 3.7.1.6 for the DEFINE subroutine.

Comma-delimited Input:

Under standard Fortran I/O using numerical formats, variables must appear within the field specified by a FORMAT statement in order to be read in correctly. However, under CMS, the user can effectively override the data-positioning requirement by separating the data items with commas. Comma-delimited input may be used with I, E, F, G, D, and Z format specifications.

Comma-delimited input is processed in the following manner: Fortran begins a character scan on the numerical field. Scanning continues until either a comma is encountered or the number of characters scanned is equal to the field width specified in the FORMAT statement for that variable. If a comma is encountered, the characters examined from the beginning of the scan operation, not including the comma, are read into the associated variables. If the end of the field is encountered, the data item is read in as usual. Scanning resumes at the character following the comma or immediately after the last character of the preceeding field. The entire operation is repeated until the I/O list has been satisfied.

The user is cautioned to allow a field width specification large enough to contain the value and the comma. A field width of zero causes nothing to be read into the variable, i.e., its value remains unchanged. If the first character examined is a comma, Fortran under CMS interprets the preceeding field as having width zero.

Notes: Comma-delimited input may not be used with A or H format specifications.

Examples:

1.

```
      .
      .
      I=2
      J=4
      K=6
      .
      .
   READ (5, 100) I,J,K
100 FORMAT (3I5)
      .
      .
   END
```


If the following line is entered as input:

```
17, 43, 81,
```

I=17, J=43, and K=81 after the READ statement is executed.

2.

```
      .
      .
      A=3.4
      B=62.5
      C=19.0
      D=37.8
      .
   READ (5, 100) A,B,C,D
100 FORMAT (4F5.1)
      .
      .
   END
```

If the following line is entered as input:

```
121,6483,33454,98,
```

A=12.1, B=648.3, C=3345.4, and D=37.8 after the READ statement is executed. Note that the third value has a field width of 5, and the next character examined for variable, D, is a comma. Fortran assumes a null field and reads nothing for D.

403

3.4.3 PLI

Purpose:

The PLI command compiles programs written in PL/I source language into machine code, and provides program listings and diagnostics.

Format:

PLI	filename 1...filename N	(option 1...option N)

filename 1...filename N — are the names of PLI files to be compiled. A separate compilation will be performed for each filename specified.

option 1...option N — are one or more of the compiler options described below.

Options:

S — includes the source program in the LISTING file.
NS — suppresses the listing of the source program.

L — includes a listing of object code in assembly language mnemonics in the LISTING file.
NL — suppresses the object code listing.

A — includes an Attribute Listing in the LISTING file.
NA — suppresses the object code listing.

E — includes an External Symbol Dictionary in the LISTING file.
NE — suppresses the listing of the External Symbol Dictionary.

X — includes a Cross Reference Listing in the LISTING file.
NX — suppresses the Cross Reference Listing.

T — creates the TEXT file of object code on the user's

4 of 7

permanent disk.
ND — suppresses the TEXT file.

C — causes compilation to proceed after compile-time processing has been completed.
NC — suppresses compilation.

M — indicates that compile-time processing is required.
NM — causes the compile-time processor to be bypassed.

S2 — includes input to the compile-time processor in the LISTING file.
NS2 — suppresses listing of input to the compile-time processor.

ST — produces extra code allowing execution-time diagnostic messages to contain statement numbers in addition to offsets relative to entry points.
NST — includes only offsets in execution-time diagnostic messages.

EF — indicates that source program is in Extended Binary Coded Decimal Interchange Code.
B — indicates that source program is punched in Binary Coded Decimal.

C60 — specifies that a 60-character set is available on the offline printer
C48 — specifies that a 48-character set is available on the offline printer.

FW — specifies that warning, error, and severe error messages are to appear in the LISTING file.
FE — specifies that only error and severe error messages are to appear in the LISTING file.
FS — specifies that only severe error messages are to appear in the LISTING file.

E — provides a larger external dictionary
NED — uses normal size external dictionary

NT — a block level and iterative DO level are printed after the appropriate statement.
NNT — suppresses printing of block and DO levels.

CL — the options in effect for this compilation are printed in the LISTING file.
NCL — no option list is printed.

The following options relate to the CMS control of compiler output.

Abbreviations for each option appears in parenthesis.

PUNCH (PU) — outputs the TEXT file onto the offline punch. Option "D" must be in effect.

PRINT (P) — outputs the LISTING file onto the offline printer. A copy of the LISTING file is not placed onto the user's disk.

NOPRINT (NP) — no LISTING file will be produced

NODIAG (NDG) — compiler diagnostics are not typed on the user's terminal.

If neither P nor NP is specified, the LISTING file will be placed onto the user's permanent disk.

Usage:

The PLI command compiles files of PL/I source language into machine-language object code. Input files must have a filetype of PLI and a record length of 80 characters.

The options governing compiler operation and output are specified in any order in a set of parenthese following the last filename. Any combination of options is valid. When conflicting options are specified, the last specified option will be used. Unsupported or misspelled options are ignored. One set of options governs all compilations performed by one command. Each of the options has a default value which is selected when none is specified. These defaults are underlined in the section on Options above.

Set values are specified for the following options, and these may not be changed by the user:

SIZE = 131072
This option specifies the amount of core, in bytes, available to the compiler for use during a compilation.

LC = 50
This option specifies that listings are to be printed with 50 lines per page.

O = 00
The level of optimization for the compiler is set at 0, meaning that the object-time storage requirement will be kept to a minimum.

SM = (1, 72)
The delimiting margins for scanning source statements will be in columns 1 and 72, meaning that only code within these margins will be processed.

For a complete discussion of PL/I usage, refer to IBM Manuals C28-6594, "PL/I Programmer's Guide" and C28-6590, "PL/I Subroutine library". An introduction to PL/I is provided in Manual C28-6806, "PL/I Primer".

Responses:

None

Error Messages:

E(00004) — warning messages have been included in the LISTING file, successful execution is probable.

E(00008) — Error messages have been included in the LISTING file. The compilation was completed with errors, and execution may 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 re-issue the PLI command.

E(00026) — FILE(S) TO BE COMPILED, no specified.
No filename(s) were specified in the PLI command. Re-issue 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. Re-ipl 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 "PF" parameter.

d. The compiler will produce 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. Because the total library is composed of a very large number of subroutines, it has been necessary to divide it into three separate files: "PLILIB1 TXTLIB", "PLILIB2 TXTLIB", and "PLILIB3 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 LOADER TXTLIB PLILIB1 PLILIB2 PLILIB3

Warning: If the loader indicates that names of the form "IHExyz" are undefined the PL/I libraries were not correctly designated.

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/1 main procedure with the main procedure in the form illustrated.

```
proc1:  PROCEDURE (PARMS) OPTIONS (MAIN);
        DECLARE PARMS CHAR(*) VARYING;
```

when using parameter passing only the two loading techniques described below may be used.

1. LOAD proc1 <proc2 ...>
   START IHECMS parm1 parm2 parm3 ...

2. LOAD proc1 <proc2 ...>
   GENMOD proc1
   .
   .
   .
   proc1 parm1 parm2 parm3 ...

Note: The varying character string PARMS will contain the concatenation of the specified parameters. Its length will always be a multiple of 8 (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. Non-format Simple I/O
   For input/output that is not a simple character string, the PL/I conversions will often work adequately.

   Examples:

   a. DISPLAY (VARIABLE);
      where VARIABLE can be any simple variable (i.e. integer, floating point, character string,etc.), it will be converted to a character string by 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 will result.

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));
      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. Presently, CMS has very limited support for PL/I File I/O. Those capabilities will be briefly described here, although the user should note that most of the hundred of possible options and special cases have not been fully tested yet.

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 PI/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(120));

DECLARE SYSIN FILE STREAM ENVIRONMENT (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 will cause it to automatically open. Since the OPEN routine is dynamically loaded there is usually a noticeable pause when an OPEN occurs. At present the TITLE option does not appear to work under CMS.

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 will result in appending records to the end of the original 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) as well as periodic problems with the dynamic loading of the routines, it is recommended that all files be OPENed explicitly at the beginning by a single OPEN statement. The form of the OPEN statement is:

```
OPEN FILE(file1), FILE(file2), FILE(file3), ...;
```

Error Recover

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" will occur. 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 malfunction of library routine - i.e. 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 (i.e. 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 will catch 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 will be trapped by the interface routine and removed from the parameter list, the typewriter SYSIN/SYSPRINT mode will be turned on. The remaining parameters, if any, will be passed to the PL/I main procedure.

2. It may be desirable to selectively switch SYSIN/SYSPRINT from files and terminal. The CMS-PL/I library routine IHEONNL (online) will cause all future SYSIN/SYSPRINT requests to refer to the user's console, the routine IHEOFFL (offline) will cause all future SYSIN/SYSPRINT requests to refer to the files FILE SYSIN and FILE SYSPRINT. The most recent IHEONNL or IHEOFFL negates the affect 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);
```

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

Purpose

IHECMS performs the CMS-dpendent initialization and passes parameters to the primary PL/I initialization subroutine, IHESAP.

Calling Sequence

IHECMS is automatically loaded with the PL/I initialization subroutines.

IHECLCK - 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 IHECLCK ( 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 "/" and "." 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.

The following is a very simple IHECLOK test program.

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 TOTAL CPU_TIME FIXED BIN (31,0);
```



```
LOOP:
          DISPLAY ('BEFORE CLOCK.');
          CAIL IHECLCK (CLOCK);
          DISPLAY ('AFTER CLOCK.');
          DISPLAY (DATE);DISPLAY (TIME_OF_DAY);
          DISPLAY (VIRTUAL_CPU_TIME);DISPLAY(TOTAL_CPU_TIME);
          GO TO LOOP;
END;
```

INEFILE - PL/I File Access Routine

Purpose

INEFILE 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 INEFILE ( 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. When writing, if the card already exists, it will be replaced by the new card. If the card did not exist (i.e., the file is being expanded), the postion of the "end-of-file" is moved and the new card added to the file.

STATUS is the CMS return code for the last INEFILE 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 that follows does the following things:

(1) Erases the file "TEST1 DATA" if it already existed.

(2) Creates a file "TEST1 DATA" consisting of 25 records, each record contains a character string representing the square of the card number (i.e., the third card in the file contains the number "9").

(3) The file is closed via FINIS.

(4) 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.

(5) The file is then read backwards (i.e. starting at last card, then next to last, etc.).

(6) The file is closed.

```
TESTFIL:    PROCEDURE OPTIONS(MAIN);                          TES00010
TESTFIL:    PROCEDURE OPTIONS(MAIN);
```

4/19

```
        DECLARE
             1 FCB STATIC,
                  2 COMMAND CHAR(8),
                  2 FILENAME CHAR(8),
                  2 FILETYPE CHAR(8),
                  2 CARD_NUMBER FIXED BIN(31,0),
                  2 STATUS FIXED BIN(31,0),
                  2 CARD_BUFFER CHAR(80);


        COMMAND='ERASE';
        FILENAME='TEST1';
        FILETYPE='DATA';
        CALL IHEFILE(FCB);
        COMMAND='WRBUF';


        DO I = 1 to 25;
             CARD_BUFFER=I*I;
             CARD_NUMBER=I;
             CALL IHEFILE(FCB);
             DISPLAY(I*I);
        END;


        COMMAND='FINIS';
        CALL IHEFILE(FCB);
        COMMAND='RDBUF';


        DO I = 1 BY 1 WHILE(STATUS=0);
             CARD_NUMBER=I;
             CALL IHEFILE(FCB);
             DISPLAY(CARD_BUFFER);
        END;


        DISPLAY('STATUS ='||STATUS);
        IMAX=I-1;


        DO I=IMAX TO 1 BY -1;
             CARD_NUMBER=I;
             CALL IHEFILE(FCB);
             DISPLAY(CARD_BUFFER);
        END;


        COMMAND='FINIS';
        CALL IHEFILE(FCB);
        DISPLAY('THATS ALL.');
```

3.4.4 SNOBOL

Purpose:

The SNOBOL command compiles source programs written in SNOBOL into SPL/1, and executes SPL/1 programs.

Format:

```
{SNOBOL}
{ SN   }  filename [(option1...optionN)]
```

filename	specifies a file 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 PRINT	either of these options specifies that information normally placed in the LISTING file is also to be printed offline.
ONLINE	specifies 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 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 exactly as shown below.

-LIST ON	resumes the listing of the SNOBOL source program in the LISTING file. This is the default value.
-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 will generate 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:

1) A listing of the SNOBOL source program, including diagnostic messages immediately following any errors detected.

2) A listing of the SPL/1 program as it was loaded for execution.

3) Any output generated by a SNOBOL PRINT statement in the program.

4) A message explaining any error completion.

5) 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 will be terminated at the HALT instruction.

Notes:

a. CMS SNOBOL differs in some significant ways from other SNOBOL implementations. The "SNOBOL Programming" section of this manual 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/360 SNOBOL USER'S MANUAL, by Stuart E. Madnick, available from the IBM Cambridge Scientific Center, Cambridge, Mass. Those interested in the implementation of the language will also be interested in SPL/1: A String Processing Language, from the same source.

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.

3.4.4.1 SNOBOL Programming

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.) Only eight files may be open at any one time, including the system files such as the LISTING file.

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 upper case. 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 will be 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 will have 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 will be called must be placed in the label field of the SUBROUTINE statement. This name will also be 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 labelled 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 environment. "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 TEXT, CHAR1, and LENGTH are passed to the subroutine as STRING, FIELD, and NUM, respectively. On return, control goes to the statement labelled 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 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.

3.4.5 BRUIN

Purpose:

The BRUIN command initiates the Brown University Interpreter.

Format:

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 BRUIN (Brown University Interpreter) for the Cambridge Monitor System (CMS).

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

The CMS command environment has been entered, where the xx.xx is the CPU time used in seconds.

For other responses, refer to the references above.

Error Messages:

None.

3.5.0 Miscellaneous

The miscellaneous commands in CMS include ECHO, which is used to test terminal transmission; EXEC, which executes a sequence of CMS commands from a disk file; FORMAT, which initializes a user's file space; and KO, KT, and KX, which clears overrides, kills typing, and terminates execution, respectively. LOGIN either saves or deletes a user's files. LOGOUT compacts the user's file directory and logs the user out of CMS. STAT provides statistics on the user's file space, MAPPRT creates a load map of the CMS nucleus, and IPL reloads a copy of the CMS nucleus. DIVERTSW diverts output from Fortran and PL/I programs directly to the offline printer. TIMELIM sets a time limit for the execution of a particular CMS command. GETLIB allows the user to access files on a library disk. QUICK provides the capability of doing operations in a file without continually referencing the file identifier. DOSBATCH transfers files to the VCS DOS BATCH machine.

3.5.0 Miscellaneous

The miscellaneous commands in CMS include ECHO, which is used to test terminal transmission; EXEC, which executes a sequence of CMS commands from a disk file; FORMAT, which initializes a user's file space; and KO, KT, and KX, which clears overrides, kills typing, and terminates execution, respectively. LOGIN either saves or deletes a user's files. LOGOUT compacts the user's file directory and logs the user out of CMS. STAT provides statistics on the user's file space, MAPPRT creates a load map of the CMS nucleus, and IPL reloads a copy of the CMS nucleus. DIVERTSW diverts output from Fortran and PL/I programs directly to the offline printer. TIMELIM sets a time limit for the execution of a particular CMS command. GETLIB allows the user to access files on a library disk.

3.5.1 LINEND

Purpose:

The LINEND command defines the logical line-end character to be used in addition to the carriage return (new line).

Format:

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 to be typed on a single physical input line separated by the line-end character. 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 will return the logical input line. Subsequent calls to read a line from the terminal will return 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). Refer to section 2.2.3.

Note:

1. The defined line-end character is the # unless the LINEND command has been issued to redefine the character.

2. If the command LINEND is issued without any parameters, only the carriage return will be used as the line delimiter.

3. When a physical input line is read, it is scanned and processed according to the specifications in WAITRD. If lower case to upper case conversion is specified, the

complete physical input line will be translated. Thus all logical input lines are translated according to the initial specification.

4. The redefined line-end character stays in effect until either CMS is IPLed again or LINEND is reissued.

Examples:

1. LINEND !
   The line-end character is set to the exclamation mark (!)

2. LINEND
   There is no longer a defined line-end character, therefore, only the carriage return will be used as a line delimiter.

Error Messages:

None.

3.5.2 ECHO

Purpose:

The ECHO command tests terminal transmission by retyping entered lines.

Format:

option — one of the three ECHO options, or if not specified, the default option U is assumed.

Options:

U The delete characters (¢, @) are interpreted, and any lower-case letters are changed to upper-case.

S Delete characters are interpreted but no lower-case to upper-case translation is made.

X No change is made in the line.

Usage:

When the ECHO command is issued, the Echo environment will be entered and each line typed by the user is repeated by the terminal.

If no option, or an invalid option, is specified, U is assumed. An entered line is interpreted according to the option specified, if any, and then typed out. 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 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 TERMINAL TEST
The first line may be entered when the keyboard unlocks after this message is typed.

Examples:

a.
```
echo u
START CONSOLE TEST
echo retypes any entered lines,
ECHO RETYPES ANY ENTERED LINES,
return
RETURN
R; T=0.03
```

b.
```
echo s
START CONSOLE TEST
incg¢Including Deletionss@, Backspaces, & special characters,
Including Deletions, Backspaces, & special characters,
Return
Return
R; T=0.03
```

c.
```
echo x
START CONSOLE TEST
Except in the X mode¢
Except in the X mode¢
returr@n
returr@n
return
return
R; T=0.03
```

Error Messages:

None.

135

3.5.3 EXEC

Contents:

3.5.3.1 Basic Usage

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	[argl...argN]

filename — specifies the filename of a file containing one or more CMS commands to be executed. The filetype must be EXEC.

argl...argN — are the arguments to replace the numeric variables in the file "filename EXEC."

Usage:

EXEC executes the sequence of commands that are specified by EXEC lines contained 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, i.e., &1&2...&30. Before the command is executed, each variable will be 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 will temporarily replace the &1, the second argument specified with the EXEC command will replace &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) will be 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 will be ignored when the commands are executed.

Arguments can be concatenated to the right-hand 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 argl is XYZ and arg2 is unspecified. Use of the double quote (") for argl 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 will be deleted. For example, in the above EXEC line if argl 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, i.e., 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:

1. 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 3.5.3.3).

2. Each EXEC file may contain a maximum of 4095 EXEC lines.

3. This version of the EXEC command is completely compatible with EXEC files created for use with the previous version of 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.

Responses:

As each CMS command in the EXEC file is issued, it is typed at the terminal.

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 in Figure 3.5.3-A, 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
T=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
```

Figure 3.5.3-A

!!!E(xxxxx)!!!
The error code xxxxx was generated by the CMS command issued from the EXEC file. Check the appropriate command in the appendix for this error message.

Examples:

a. In FIGURE 3.5.3-B the command EXEC FORTCLG LLHS is issued. LLHS is a file whose filetype is FORTRAN and LLHS will replace 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 out before it is executed.

b. In FIGURE 3.5.3-C the FORT EXEC is created by EDIT. The only command placed in the file is "FORTRAN&1(PRINT)". The file LISTF EXEC is then typed out by issuing the PRINTF command. The EXEC command is then issued specifying the filename LISTF and the two arguments EXEC and FORT. The file LISTF EXEC was created earlier with the LISTF command (see Section 3.1.9) and contains the sequence of FORTRAN files to be compiled. Each file identifier in LISTF 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 LISTF 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 LISTF EXEC is executed. This sequence continues until all commands are executed in the LISTF EXEC file.


```
exec fortclg llhs
FORTRAN LLHS
LOAD LLHS (XEQ)
EXECUTION BEGINS.
APRIL 1968 DATA 5.320                1.920    5.600
T=0.55/1.44 01.30.45
```

FIGURE 3.5.3-B Example of an EXEC file to compile, load, and execute a FORTRAN program.

```
edit fort exec
INPUT:
fortran &1 (print)

EDIT:
file
T=0.55/1.43 01.30.50

printf listf 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
T=0.55/3.21 01.32.15

exec listf exec fort
EXEC      FORT    W       FORTRAN          P5    001
FORTRAN   W       (PRINT)
EXEC      FORT    SUB2    FORTRAN          P5    001
FORTRAN   SUB2    (PRINT)
EXEC      FORT    A       FORTRAN          P5    001
FORTRAN   A       (PRINT)
EXEC      FORT    SUBB    FORTRAN          P5    001
FORTRAN   SUBB    (PRINT)
T=1.50/1.80 01.33.27
```

FIGURE 3.5.3-C The file FORT EXEC is created, the file LISTF EXEC is typed out, and then an EXEC command is issued to nest EXEC's.

3.5.3.2 Special Features

A line of an EXEC file specifies either a CMS command or is 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.

EXEC lines, containing either a CMS command or an EXEC control, may be identified with a label. If the first word of an EXEC line begins with a dash (-) it is assumed to be a label of an EXEC line. Labels are used in the control of the sequence of EXEC lines to be executed (see 3.5.3.3 - &GOTO and &LOOP).

EXEC lines may contain words which begin with an ampersand(&). A word beginning with an ampersand may be either a) a numeric variable, b) a keyword, i.e., a symbolic variable, or c) 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 7 characters at least one of which is not an integer. Control words have the same form as keywords and are defined in section 3.5.3.3. Numeric variables and keywords are substituted for before the EXEC line is interpreted.

Numeric variables are substituted for by arguments which are specified when the EXEC command is issued (see 3.5.3.1). 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 than 30, or 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 the variable &* 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 3.5.3.3).

The value substituted for a keyword may be
i) specified in an EXEC line, or
ii) implied if the keyword is a special keyword.

The value of a keyword may be specified by an EXEC line of the form:
&KEYWORD = VALUE
which defines the keyword &KEYWORD to have the value VALUE.

Keywords can be redefined as often as desired.

A number of keywords have been defined to have special meaning and have their values set in a special way.

1. &LINENUM - has the value of the current EXEC line number plus one.

2. &INDEX1...&INDEX9 - are used as indices and initially have the value +1. Indices 1 through 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 will set &INDEX5 to 30975.
   &INDEX7 - 50 will add -50 to the value of &INDEX7.
   Indices are local to the current level of recursion.

3. &INDEX0 - has as its value the code number in register 15 on return from the previous CMS command.

4. &INDEX - has as its value the number of arguments given when the EXEC command was issued.

5. &GLOBAL0...&GLOBAL9 - are used for communication between levels of EXEC recursion and are set and incremented in the same way as &INDEX1...&INDEX9.

6. &GLOBAL - has as its value the level of recursion.

3.5.3.3 EXEC Control Words

The following EXEC control words 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 in an EXEC file.

1) &ERROR [action / &CONTINUE]

Action is any exec line without a statement label. Action will be 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, will result in an exit from this level of EXEC with error code of 11.

2) &IF condition action

where condition consists of the three parameters:

[&* / &$ / anything] [EQ / NE / GT / LT / GE / LE] [&* / &$ / anything]

and action is any EXEC line without a label. If the condition is satisfied, the action is executed. The comparison specified by the second argument is made between the first and third arguments. The &$ 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 will be made only if both the comparands are numerical. 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.

3) &EXIT [n / 0]

An exit from EXEC is taken 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 will be returned as the error code in register 15.

4) &QUIT [n / ON / OFF]

&QUIT n is similar to &EXIT n except that &QUIT n will return to level 0, the CMS command level, regardless of the level of recursion of EXEC commands; whereas, &EXIT exits to the next lower level of recursion.

&QUIT ON sets the return level for a subsequent &QUIT control to one level higher of recursion. 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 will reset the return level to level 0, the CMS command level.

5) &SKIP [1 / n]

If $n<0$, the next EXEC line to be executed will be n lines before the current line. If $n \geq 0$, the next EXEC line to be executed will be n+1 lines after the current line, i.e., n lines are skipped.

6) &GOTO [TOP / label / EXIT]

&GOTO TOP will cause 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 command.

&GOTO label will search the EXEC file starting from the present EXEC line and proceeding to the end of the file, then to the beginning of the file, and finally back to the present line location, looking for the first EXEC line beginning with the specified label.

7) &LOOP [label / n1] [condition / n2]

This EXEC control line causes looping either 1) to and including the labeled line, or 2) through the number of lines specified by n1, beginning with the next line. Looping continues either 1) until the condition is satisfied, or 2) for n2 times. The condition is specified the same as with the &IF control word and is first tested before looping. Loops may be nested to a depth of 4.

The numbers n1 and n2 must be less than 4096.

8) &CONTINUE

This EXEC line is ignored. It may be useful with &GOTO or &LOOP and is the default action for &ERROR.

9) &TYPEOUT [ALL / ON / ERROR / OFF / NOEXEC] [TIME / NOTIME] [PACK / NOPACK]

ALL causes all CMS command lines and EXEC control lines to be typed.

ON causes all CMS command lines to be typed but suppresses the typing of EXEC control lines.

ERROR causes only the CMS command lines which result in an error to be typed. EXEC control lines are not typed.

OFF suppresses the typing of all EXEC lines.

NOEXEC is the same as OFF and is permitted for compatability with EXEC files created for use with the previous version of the EXEC command.

TIME causes the time of day to be typed preceding each CMS command line typed.

NOTIME suppresses the typing of the time of day with each CMS command line.

PACK causes excess blanks to be removed from a line to be typed resulting in only one space between words in an EXEC line.

NOPACK suppresses the removal of excess blanks from a line to be typed.

10) &TIME [type / ON / OFF]

&TIME will cause typing of the time since the last time the time was typed. &TIME ON will cause typing of the time message after each CMS command is typed.

&TIME OFF suppresses the typing of the time after each CMS command.

11) &SPACE [1 / n]

This control word causes n carriage returns to be typed at the console.

12) &PRINT line

"Line" will be printed on the typewriter console. All keywords, symbolic arguments, etc., will be substituted into the line.

13) &COMMENT line

This EXEC line is ignored and can be used to annotate the EXEC file.

14) &ARGS argl,....argN

The parameters argn are used to redefine the numeric variables &n and the special keyword &INDEX is redefined to have the value of n.

15) &READ [1 / n / ARGS]

If n is specified, the next n EXEC lines will be read from the typewriter console instead of from the EXEC file. As each line is read it is executed as if it were read from the EXEC file.

If n is ≤ 0, reading will stop and the next EXEC line will be obtained from the EXEC file. Reading is also terminated by typing &GOTO, &SKIP, &LOOP, &EXIT, or &QUIT or can be reset by typing another &READ control line.

If ARGS is specified, one line will be read from the console, the line will be scanned and used to redefine the numeric variables. The special keyword &INDEX will be redefined to be the number of arguments read. The line read will not be executed as an EXEC line.

Only the first 72 characters on a line will be read.

16) &STACK [FIFO / LIFO] line

All numeric variables and keywords in a line are substituted for, and the line is "stacked" in the input buffer. Subsequent calls to read from the console will obtain lines which were stacked in the input buffer.

If the first word after the control word is FIFO the line is stacked in a First In First Out order. If the first word after the control word is LIFO the line is stacked in a Last In First Out order. If not specified, FIFO is assumed.

17) &BEGSTACK [FIFO / LIFO]

line 1
line 2
.....
line N
&END STACK

This sequence will stack lines line 1 through lineN, literally without truncation and will not substitute for numeric variables or keywords. This sequence may be used to specify input or EDIT request to the EDIT command.

18) &SET action

This control word has been defined for compatibility with old EXEC files which used the control words ERR and TYPEOUT or actions. &SET may later be removed as an EXEC control word.

Notes:

1) All numeric variables, keywords, EXEC control settings, and limitations (e.g., max. depth of loop nesting) are local to the current level of EXEC unless otherwise noted.

2) Any EXEC control word may be abbreviated by a sufficient number of characters to distinguish the control word from other control words. The following precedence order is observed: ERROR, EXIT; SKIP, SPACE; STACK, SET; TYPEOUT, TIME; other control words, followed by other keywords. Keywords cannot be abbreviated.

3) An error from a CMS command does not cause an exit from the level of EXEC.

4) When EXEC is entered, the assumed state of the controls are &ERROR, &CONTINUE, &TIME OFF, and &TYPEOUT ON TIME PACK.

5) If an EXEC line specifies an invalid CMS command, an error code of E(-0007) will be returned. The EXEC command is not terminated.

Errors:

E(00001)	File does not exist.
E(00002)	&SKIP or &GOTO error.
E(00003)	Files 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.

3.5.4 FORMAT

Purpose:

The FORMAT command is used to initialize a disk area in the CMS format, or to count the number of cylinders on a disk.

Format:

P specifies that the user's permanent disk area is to be initialized, and any files on it erased.

T specifies that the user's temporary disk area is to be initialized, and any files on it 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.

Usage:

P or T must be specified. According to the option selected, the user's permanent or temporary disk area is initialized by writing a new home address and four records on each track. 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 will not normally be needed thereafter.

If C is specified, data on the disk is not erased, and any files are preserved.

Response:

x-DISK: nnn CYL.

The number of cylinders formatted is typed out, indicating which disk was initialized. If C was specified, this number of cylinders was counted, but not erased.

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 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 C was specified with the command.

E(00006) CE and DE NOT TOGETHER CHECKING NO. CYLINDERS
Same as E(00004), except C was specified with the command.

E(00007) UNEXPECTED UNIT-CHECK NO. CYLINDERS
Same as E(00003), except C was specified with the command.

3.5.5 KE

Purpose:

The KE command causes terminal output lines to be truncated to 80 characters.

Format:

KE

Usage:

The KE command is used to truncate terminal output from an executing CMS command or user program. In order for the KE command to be recognized, it must be entered after interrupting program execution by hitting the attention key once to transfer to the CP environment and a second time to transfer control to CMS. Typing KE at this point will cause terminal output to be truncated to 80 characters until the command or program completes. Data beyond 80 characters is lost. Normal terminal output will resume after the executing program has been completed.

Notes:

a. Entering KE as a normal CMS command will have no effect. KE 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 first allow interrupts from the multiplexor channel. As soon as this processing is completed, the keyboard will be unlocked and KE may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing a KE command.

Example:

a. KE
Assuming that the attention key had been hit twice prior to typing KE, CMS would intercept and truncate to 80 characters any terminal output from the command or program in progress.

Error Messages:

None.

3.5.6 KO

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 the attention key once to transfer to the CP environment and a second time to transfer control to CMS. Typing KO will then cause all overrides to be cleared, and no further trace information will be recorded. Program execution will continue to its normal completion, and all recorded trace information will be printed on the offline printer.

Notes:

a. Entering KO as a normal CMS command will have no effect. KO 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 multiplexor channel. As soon as this processing is completed, the keyboard will be 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 will have no effect unless a SETERR or SETOVER command has been issued previously.

Example:

a. KO
Assuming that the attention key had been hit twice prior to typing KO, CMS will stop 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.

3.5.7 KT

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, the attention key 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 will continue but the keyboard will be unlocked to accept user input. Typing KT at this point will cause all further terminal output from the executing command or user program to be intercepted and suppressed. Execution will continue to normal program completion, when the Ready message will be typed out and normal terminal output will resume.

Notes:

a. Entering KT as a normal CMS command will have no effect. KT will have 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 multiplexor channel. As soon as this processing is completed, the keyboard will be unlocked and KT may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing a KT command.

Example:

a. KT
Assuming that the attention key had been hit twice prior to typing KT, CMS would intercept and suppress any terminal output generated by the program in progress.

Error Messages:

None.

3.5.8 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, re-IPL'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 the attention key once to transfer to the CE environment and a second time to transfer control to CMS. Issuing the KX command will then close any open user files and signal CP that the user has no more I/O for offline devices. After updating the user's file directory, KX re-IPL's CMS from the same device as it was initially IPLed, and the user is returned to the CMS command environment.

Notes:

a. Entering KX as a normal CMS command will have 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 multiplexor channel. As soon as this processing is completed the keyboard will be 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 xx.xx xx/xx/xx
```

This response is typed whenever KX is interpreted, as CMS is re-IPLed. The CMS command environment is then entered.

Error Messages:

None.

3.5.9 LOGIN

Purpose:

LOGIN causes the user's files to be either saved or deleted, as specified.

Format:

UFD	indicates that the user's file directory (and hence all existing disk files) are to be saved.
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.

Usage:

The LOGIN command, if used, should be issued immediately after the CMS Command environment has been entered in response to an IPL command, console function, or request. LOGIN must have an operand, which is either UFD of NO_UFD. If UFD is specified, the user's file directory, and hence all of his disk files, will be maintained as they presently exist. If NO_UFD is specified, the user file directory will be cleared and all disk files will, in effect, be erased. If the LOGIN command is not issued when the user first enters the CMS Command environment, LOGIN UFD will be assumed and all existing disk files will be saved.

Examples:

a. LOGIN UFD
The present user file directory will be maintained as it currently exists and a compacting routine will be called to eliminate any blank entries caused by files which were erased during the last terminal session.

b. LOGIN NO_UFD
The user's file directory will be cleared, in effect erasing all of his files currently stored on disk.

Error Messages

E(00001)	LOGIN UFD FAILED The user's file directory has not been successfully brought into core. If the command fails a second time, issue the NO_UFD command.
E(00002)	LOGIN NO_UFD FAILED; TRY FORMAT P The LOGIN NO_UFD command has not been executed correctly. Issue the FORMAT P command.
E(00003)	PLEASE 'LOGIN UFD' OR 'LOGIN NO_UFD' An invalid operand was specified with the LOGIN command. Reissue the LOGIN command in its correct format.

3.5.10 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 LOG	<anycom>

anycom is any CMS command

Usage:

The LOGOUT command is not required when use of CMS has been completed. If issued, however, it will cause 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 will execute any CMS command specified as its operand (LISTF, for example, to obtain a list of the newly organized file directory).

Finally, the Ready message will be 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 will be written out to disk and control will be 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 insure 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 the attention key once to transfer control to CP and type LOGOUT to logout 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 will be 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 will then be unlocked to accept any CP console function.

Example:

a. logout

```
R;T=18.32/43.21  12.15.28
CP ENTERED, REQUEST, PLEASE.
```

The compacting routine will be called, the indicated message will be typed at the terminal, and control will transfer to the CP environment where the keyboard will be unlocked to accept any CP console function.

b. logout listf

The compacting routine will be called, the LISTF command will type out the contents of the reorganized user file directory, the logout message will be typed and control will transfer to the CP environment where the keyboard will be unlocked to accept any console function.

Error Messages:

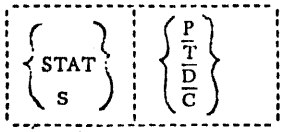
None.

3.5.11 STAT

Purpose:

The STAT command is used to (1) type out statistics on how may records of the user's permanent and/or temporary disk areas are currently in use, (2) type out the current user-defined commands, or (3) execute a routine to compact information in the disk management tables.

Format:



P specifies that only statistics for the permanent disk are to be typed out.

T specifies that only statistics for the temporary disk are to be typed out.

D specifies that only the user-defined commands are to be listed.

C specifies that the compacting routine is to be executed.

If no operand is specified, P, T, and D are assumed.

Usage:

The STAT Command will give a count of the records in use on the user's permanent and/or temporary 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.

Any user-defined commands may also be listed using STAT. These are commands which have been written by the user and entered into the system file directory by the DEFINE command.

If C is specified in the STAT command, none of the above actions are performed. Instead, a routine to eliminate unnecessary space in the disk management tables will be executed. This routine is always called automatically when CMS is initial program loaded and when the user issues a LOGOUT command to CMS.

58

Responses:

```
     P-DISK:  nnnn RECORDS IN USE, nnnn LEFT (of nnnn),
              nn% FULL (of nnn CYL.)
```

This response is given for the permanent disk, and then for the temporary disk, unless P or T was specified, in which case information will be given only for the specified disk.

```
     DEFINED COMMAND(S) = command1...commandN
```

Up to five defined commands are listed by name.

```
     NO DEFINED COMMANDS
```

This response is given if the user has no defined commands.

```
     NO KNOWN T-DISK
```

This response is given when T is specified, but no temporary disk is configured on the user's virtual machine.

Example:

```
stat
P-DISK: 0771 RECORDS IN USE, 0229 LEFT (of 1000), 77% FULL (of 025 CYL.)
T-DISK: 0000 RECORDS IN USE, 0040 LEFT (of 0040), 00% FULL (of 001 CYL.)
DEFINED COMMAND(S) = EMSPRIME  SNAPR
```

Since no options are specified, P, T, and D are assumed. Statistics for the permanent and temporary disks are followed by the list of the user's defined commands.

Error Messages:

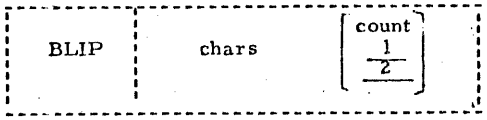
None.

3.5.12 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:



where:

chars	is the character to be typed out, and
count	is a digit from 1 to 8 indicating the number of BLIP characters

Usage:

The default setting of the BLIP characters is a sequence of non-printing characters - a shifting of the typing element from lower case to upper case. If it is desired to have a printed recording of the execution of a program, the BLIP characters should be changed to printing characters, e.g., a single dot.

Note:

1. If the count parameter is defaulted a count of 1 is assumed.
2. If the first parameter is zero or if both parameters are defaulted, the BLIP characters are reset to their non-printing default setting.
3. 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.

3.5.13 MAPPRT

Purpose:

MAPPRT creates, and optionally prints, a file containing a map of entry points in the CMS nucleus.

Format:

A	creates the file CMS-NUC ALPHABET P1, containing the nucleus entry points listed in alphabetical order.
N	creates the file CMS-NUC NUMERIC P1, containing the nucleus entry points listed in numerical order.
C	creates the file CMS-NUC ALPHANUM P1, containing both the A and N orderings.
ON	types out the file at the terminal.
OFF	prints the file on the offline printer.
NO	specifies 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 GMS-NUC. The filetype is ALPHABET, NUMERIC, or ALPHANUM, depending on the option specified. See Figure 3.5.13-A for examples of the three types of format.

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.

Note:

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 th EDIT environment.

b. MAPPRT should only be issued after CMS has been initialized (by IPL 190), as MAPPRT uses tables set up by the nucleus loader.

Responses:

None, except under the ON option, 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 Example a, and the file is printed on the offline printer.

Error Messages:

None

printf cms-nuc alphabet

```
ABEND    AT 0C068
ACTLKP   AT 03C0E
ACTTAB   AT 011E8
ACTTA1   AT 011F0
ALTER    AT 00000
ASMADDR  AT 00B20
BATCH    AT 00000
BATSWT   AT 0A2CA
BCKSPACE AT 0C068
BDEBUG   AT 0BF84
BUFFER   AT 0A5F8
CARDPU   AT 0C2B8
```

printf cms-nuc numeric

```
ALTER    AT 00000
BATCH    AT 00000
ERASE    AT 00000
OFFLINE  AT 00000
SYSCTL   AT 00000
TAPEIO   AT 00000
UFDFLG   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 0C068
ACTLKP   AT 03C0E
ACTTAB   AT 011E8
ACTTA1   AT 011F0
ALTER    AT 00000
ASMADDR  AT 00B20
BATCH    AT 00000
BATSWT   AT 0A2CA
BCKSPACE AT 0C068
BDEBUG   AT 0BF84
BUFFER   AT 0A5F8
```

```
ALTER    AT 00000
BATCH    AT 00000
ERASE    AT 00000
OFFLINE  AT 00000
SYSCTL   AT 00000
TAPEIO   AT 00000
UFDFLG   AT 00000
NUCON    AT 00100
USFL     AT 00108
STADDR   AT 00110
TBLNG    AT 00118
```

Figure 3.5.13-A. Samples of the three types of nucleus map files created by MAPPRT. Only the first few entries of each are shown.

3.5.14 IPL

Purpose:

IPL causes a new copy of the CMS nucleus to be initial program loaded.

Format:

```
IPL
 I
```

Usage:

This command performs the same action as the IPL console function. Both will cause the CMS nucleus to be initial program loaded, although the copies of the nucleus which are brought in will not necessarily be the same in both cases.

The IPL command and the "IPL 190" console function will load in the copy of the nucleus which resides on disk 190. Periodically, a copy of this nucleus is saved by a CP utility. The "IPL CMS" console function will load 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 the IPL command and the IPL console function will be different.

Response:

```
CMS . . . VERSION I.0 - 05/01/68
```

This response indicates that a new copy of the nucleus has been loaded and that control has transferred to the CMS Command environment.

Example:

a.
```
ipl
CMS . . . VERSION I.0 - 05/01/68
```

A new copy of the CMS nucleus has been initial program loaded and the keyboard will be unlocked to accept any CMS command.

Error Messages:

None.

3.5.15 RT

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, the attention key 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 will continue but the keyboard will be unlocked to accept user input. Typing RT at this point will cause all further console output from the executing command or user program to be typed out. Execution will continue to normal program completion.

Notes:

a. Entering RT as a normal CMS command will have 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 multiplexor channel. As soon as this processing is completed, the keyboard will be unlocked and RT may be entered.

c. The character-delete symbol (@) and line-delete symbol (¢) are not valid when issuing an RT command.

Example:

a. RT

Assuming that the attention key had been hit twice prior to typing RT, CMS will restore any console output generated by the program in progress.

Error Messages:

None.

3.5.16 CHARDEF

Purpose:

CHARDEF redefines the logical characters that correspond to hardware functions.

Format:

CHARDEF	function code	<replacement character>

function code
- E 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 will be 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 E or T is specified, EDIT and CEDIT commands will 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 either (1) CMS is re-IPLed, (2) another CHARDEF command is issued, or (3) the user logs out of CP.

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.

3.5.17 CPFUNCTN

Purpose:

The CPFUNCTN command transmits console function commands to CP-67 without leaving the virtual mode of operation.

Format:

CPFUNCTN	<NOMSG>	string
CPF		

NOMSG — optional argument to turn off typing of 'BAD ARGUMENT' and 'INVALID CP REQUEST' messages.

string — 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.

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.

Examples:

1. CPFUNCTN XFER 00D to CSC1
   Executes an XFER console function.

2. CPF NOMSG DETACH 00E
   Executes a DETACH command. If 00E is not a valid address, the BAD ARGUMENT message will not be typed.

DIVERTSW

Purpose: The DIVERTSW command will set a switch either on or off for diverting certain terminal output to the offline printer. Output for Fortran logical unit 6 and PL/I FILE SYSPRINT can be diverted to the printer without an intermediate file being created on the P-disk.

Format: DIVERTSW [ON / OFF]

Usage: DIVERTSW ON sets an indicator which is later tested during either Fortran or Pl/I execution. Subsequently, all output destined for the terminal will be diverted directly to the offline printer. This command can be convenient whenever large amounts of output are expected and P-disk space is too small to accommodate the output.

DIVERTSW OFF will reset the indicator to output to the terminal.

Responses: NONE.

Notes:

a. DIVERTSW does not close the printer automatically. Therefore, the user must close the printer file after the program completes execution. This is done in CP with the CLOSE function CLOSE 00E, or with the CMS command CLOSIO PRINTER.

TIMELIM

Purpose: The TIMELIM command will set a time limit for the execution of any command and terminate execution if the time limit is exceeded.

Format: TIMELIM [n1 [n2] / 10 / OFF]

where n1 = number of minutes
n2 = number of seconds

Usage: TIMELIM n1 n2 will establish a time limit of n1 minutes and n2 seconds for the command which immediately follows TIMELIM. If the execution of the subsequent command exceeds the time limit, execution will be terminated. If the job is completed within the time limit specified, the time limit is no longer in effect. TIMELIM with no parameters sets the time limit to a default value of ten minutes.

TIMELIM OFF nullifies the effect of a previous TIMELIM command.

Responses: TIME LIMIT EXCEEDED.....EXECUTION TERMINATED. This message is printed immediately before a KX is issued.

Note: If a TIMELIM command has been specified, no "blips" will occur during the execution of the timed job.

Examples:

a.
```
TIMELIM 14
$ FORTPROG
```

b.
```
LOAD ITERATE SUB1 SUB2
TIMELIM 35 30
START
```

Errors: none

3.5.20 GETLIB

Purpose: The GETLIB command permits access to files on a library disk.

Format:

```
        [userid devadd [PASS=  xxxxxxxx]]
GETLIB  [OFF                           ]
```

Userid — specifies the owner of the disk to be accessed

Devadd — the owner's virtual device address in hexadecimal

xxxxxxxx — read-share password of the library disk

OFF — discontinues the use of the library disk

Usage: GETLIB is provided so that many users may share a common library of files. It is a space-saving mechanism designed to eliminate the need for storing duplicate copies of certain seldom-modified files on several users' P-disks.

Issuing a "GETLIB userid devadd" command will provide access to a library disk via the following sequence of events: Device 192, if it exists, is detached; GETLIB then LINKS to the userid's devadd as 192; and finally the tables which reside on the library disk and describe its contents are read into core. GETLIB OFF detaches device 192 and clears the core-resident tables associated with that device. GETLIB with no arguments requires that device 192 is already attached. It then makes the linked disk accessible as a library disk.

Responses: DEV 192 DETACHED
Either a "GETLIB OFF" command has been executed, or "GETLIB userid devadd" has been issued subsequent to other GETLIB commands.

Notes:

a. In accessing files. the P-disk will be searched before the library disk.

b. All files which reside on a library disk are temporarily assigned the mode letter, T, by CMS.

c. All library files are read-only.

d. The ability to specify a password in the GETLIB command itself is provided for added flexibility. It is specifically designed for use in EXEC procedures. However, the user is cautioned to set type-out of CMS commands off (&TYPEOUT OFF) to prevent the printing of passwords on the terminal sheet.

Error messages:

E(00001) An invalid GETLIB argument has been detected.

E(00002) GETLIB has been issued with no arguments, and no disk has been attached as device 192.

Examples:

1. GETLIB LIBMACH 191
   Device 191 of LIBMACH is attached and its files are made available for use.

2. GETLIB OFF
   The library disk is removed.

3. GETLIB
   User's device 192 now becomes a library disk.

3.5.21 QUICK

Purpose:

The use of the QUICK environment allows the user to create, edit, compile (or assemble) and execute files without continually referencing the file by its identifier.

Format:

```
QUICK    filename    [ SYSIN | PL/1 | FORTRAN ]
```

filename	specifies the filename of the file on which operations are to be performed.
filetype	specifies the filetype of the file on which operations are to be performed. If the second argument is omitted, a filetype of FORTRAN is assumed.

Usage:

The QUICK exec procedure is entered. Filename is the name of the program to be created, edited, compiled, or executed. SYSIN indicates the program is written in assembly language, PLI indicates a PL/1 program, and FORTRAN indicates a FORTRAN program.

An asterisk is typed at the terminal upon successful entry into QUICK and after the completion of subsequent QUICK requests.

QUICK Requests: The requests described on the following pages are valid in the QUICK environment. Some of the requests, however, are not valid for all three languages. QUICK responds to invalid requests with the message: INVALID QUICK REQUEST

EXAMPLES:

1) QUICK TESTFILE
   QUICK entered-operations to be performed on the file, TESTFILE FORTRAN.

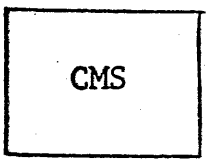
2) Q EXAMPLE SYSIN
   QUICK entered-operations to be performed on the file, EXAMPLE SYSIN.

ERROR MESSAGES:

1) WRONG ARGUMENTS-CMS ENTERED
Retry with valid arguments.

CMS

Format:



Usage:

The CMS request unlocks the keyboard so that a CMS command can be entered; this command is executed immediately and control is then returned to the QUICK environment.

Note: Any CMS command may be entered following the CMS request.

Error Messages: none

COPY

Format:

```
COPY
```

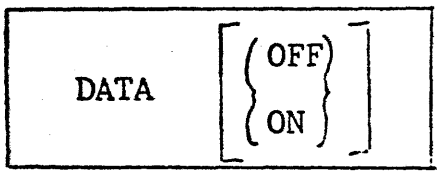
Languages: Fortran, Assembler, Pl/1

Usage:

The COPY request saves a copy of the current file as filename EXTRA*** Pl. If filename EXTRA*** Pl exists, it is erased before the fresh copy of the current file is created.

Error Messages: none

DATA



Languages: Fortran, Assembler, PL/1

Usage:

The DATA request provides the user with the capability of specifying only once the terminal input to the current program. This is accomplished through the automatic creation of a file (DATA**** EXEC) which is saved on the user's P-disk.

If the user issues the DATA request with no parameters, QUICK responds with:

```
ENTER INPUT DATA
NEW FILE
INPUT:
```

The data which would normally be entered from the terminal is then entered in the format expected by the program. Multiple lines are permitted. When the desired data has been entered, hit an extra carriage return and upon receiving the system message, EDIT:, type FILE. This saves the data entered and returns to the QUICK environment (an asterisk is printed). Each time the $ request is issued, the data entered via the DATA request is stacked in the terminal input buffer such that each programmed read to the terminal obtains the desired input directly from the input buffer. When the user's program has read all the stacked input lines, remaining terminal read requests unlock the keyboard and the appropriate input data can be keyed in normally.

To suspend data stacking once the data request has been insured, the DATA request is reissued with the OFF argument. Similarly, the ON argument resumes data stacking and can be used to activate a DATA request issued in a prior QUICK session.

Notes:

1. When the DATA request is issued with no parameters, and a DATA**** EXEC file currently exists, the current file is erased and a new DATA**** file is created.
2. The DATA option is initialized to the OFF state each time the QUICK procedure is invoked.

Error Messages: None

EDIT

Format:

EDIT
E

Languages: Fortran, Assembler, PL/1

Usage:

The EDIT request is identical to the CMS command, EDIT filename filetype. If the file specified to QUICK does not exist, it is created for input. Otherwise, the existing FORTRAN, SYSIN, or PL/1 file can be edited. The CMS edit requests, FILE and QUIT, return control to the QUICK environment (an asterisk is printed).

Error Messages: none

OK

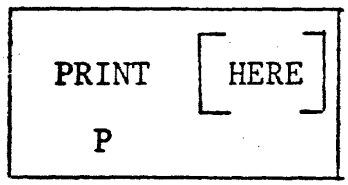
Languages: Fortran, Assembler, PL/1

Usage:

The current file is compiled (fortran and PL/1) or assembled (assembly language). If processing is successful, filename TEXT is created on the user's P-disk.

Error Messages: none

PRINT



Languages: Assembler

Description:

The PRINT request prints the LISTING file created when the current file is assembled. If the argument HERE is supplied, the LISTING file is printed at the terminal. Otherwise, the file is dumped to the high speed printer at the computer center.

Error Messages: none

QUIT

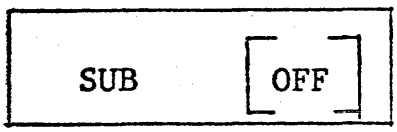
QUIT
Q

Languages: Fortran, Assembler, PL/1

Description: Control is returned to the CMS command environment.

Error Messages: none

SUB



Languages: Fortran, Assembler, PL/1

Description:

The SUB request allows the user to specify up to three TEXT routines which will be loaded with the current program each time the $ request is issued. When the SUB request is issued, QUICK responds with "ENTER SUBROUTINE NAMES." The routines then specified will be loaded with the main program until the SUB request is issued with the OFF argument or until the user leaves the QUICK environment.

Error Messages:

1. TOO MANY ROUTINES - CMS ENTERED
   Return to the QUICK environment with the QUICK CMS command.

$

Languages: Fortran, Assembler, PL/1

Usage:

The TEXT version of the current file is loaded and executed. The SUB request can be used to specify additional TEXT routines to be loaded each time the current program is executed. If a PL/1 program is to be executed, QUICK insures that PLILIB1, PLILIB2, and PLILIB3 are specified in the (LIBE) option at load time. If the argument, HERE, is supplied, SYSOUT for a PL/1 program is directed to the terminal.

Error Messages: none

3.5.22 DOSBATCH

PURPOSE:

The DOSBATCH command provide simple transfer of job files to the DOS batch machine, and allows the user to select the manner in which he receives output from these jobs.

FORMAT:

DOSBATCH	fn ft ⟨options⟩ BATCH DOS

defaults The default filename is BATCH and the default filetype is DOS.

OPTIONS:

Options determine where printed or punched output should go, either to the real device in the VCS computer center, or to the user's virtual reader. If no options are specified, the output will be routed to the appropriate real device.

LIST (or L) will return printer output from the batch job as a file in the user's reader.

DECK (or D) will return punched output from the batch job as a file in the user's reader.

The options are enclosed in parentheses. Either one or both options may be specified. To specify both, a blank must separate the options.

DOSBATCH short fortran (list deck)

If both options are specified, 2 files will be returned in the virtual reader.

USAGE:

The DOSBATCH command will transfer the one file specified to the batch machine to be run under DOS. Appropriate job control must be included in the file.

RESPONSES:

a. R; hh.mm.ss
b. FILE NOT FOUND
   E(00001)

3.5.22 DOSBATCH

EXAMPLES:

a. DOSBATCH

Uses the defaults. Sends the file BATCH DOS with no options.

b. DOSBATCH ROOT (DECK)

Uses the default filetype. ROOT DOS to be run under DOS. Any punched output will be returned as a file in the user's reader.

c. DOSBATCH SHORT FORTRAN (LIST DECK)

The file SHORT FORTRAN will be transferred to be run, and both printed and punched output will be returned as reader files.

NOTE:

The default filetype cannot be envoked unless the default filetype is also used. For example, to transfer the file BATCH COBOL, the command:

DOSBATCH COBOL (Options)

would be invalid since "COBOL" would be looked upon as filename rather than type.

This section contains those CMS commands which are utilitarian in function. They include SORT, which sorts a disk file, COMPARE, which compare two disk files record by record, TPCOPY, which copies a tape, MODMAP, which types out the load map of an existing MODULE file, and TPMATCH, which compares two magnetic tape files.

3.6.1 CNVT26

Purpose:

The CNVT26 command converts a BCDIC (026) card-image file to EBCDIC (029).

Format:

CNVT26	filename filetype

filename filetype is the file to be converted.

Usage:

The specified file is searched for on the P disk and then the T disk. 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 will be erased.

Examples:

a. 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 orginal name.

Error Messages:

E(00001) FILE NOT FOUND
The specified file does not exist on either the P disk or T disk.

E(00002) FILENAME(S) MISSING
Either the filename and/or filetype were not specified with the command. Reissue the command correctly.

3.6.2 COMPARE

Purpose:

The COMPARE command can be used to compare two disk files.

Format:

COMPARE	filename1	filetype 1	filename2	filetype2

Usage:

If the files are identical, a message will so indicate. If corresponding records in each file do not agree, the record from the first file will be printed followed by the record from the second file. If the end of one file is reached before the end of the other, a message will so indicate.

The maximum record size that can be COMPARED is 256 characters.

Responses:

EOF FILE n
The end of file has been reached on file n, where n is either 1 or 2.

NOT EOF FILE n
This message will follow EOF FILE n if the end of file is reached on one file prior to that of the other file.

FILES HAVE DIFFERENT CHARACTERISTICS
The files being compared do not have either the same record size or record format.

PARAMETER ERROR
Insufficient parameters were supplied.

RECORDS ARE TOO LONG
The records are longer than 256 characters, therefore they can not be compared.

Error Messages:

E(00001)	FILE NOT FOUND
E(xxxxx)	FATAL ERROR

3.6.3 CVTFV

Purpose:

The CVTFV command converts a file with fixed length records 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 deleted.

Usage:

If (T) is specified and the file consists of 80 character records (a normal card file), bytes 73 through 80 of each card image will be deleted regardless of contents. Therefore to convert card files without loss of information in columns 73 through 80, do not use the (T) option.

The P disk and the T disk are searched for the specified file. The old file is erased and the new file is given the original name, type, and mode.

CVTFV creates a work file with a filetype of CVTUT1 and the filename and mode of the specified file. If a file already exists with the same identifier as the work file, it will be erased.

Examples:

a. CVTFV USERGUID SCRIPT (T)
The file USERGUID SCRIPT is converted from fixed length records to variable length records. If the file contains 80 character records, columns 73-80 are deleted.

Error Messages:

E(00001) FILE NOT FOUND
The specified file does not exist on either the P disk or T disk.

E(00002) INCORRECT FORMAT
A T was not specified between the parentheses.

E(00003) INCORRECT FORMAT
Either a filename and/or filetype was not specified with the command. Reissue the command correctly.

E(00004) FILE IS ALREADY VARIABLE
The specified file contains variable length records, therefore it can not be converted.

3.6.4 MODMAP

Purpose:

The MODMAP command types the load map associated with the specified MODULE file on the console typewriter.

Format:

MODMAP	filename

filename 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.

Note:

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, therefore the load map does not exist.

3.6.5 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 can not write the output back into the input file (file1). If the output file already exists, either the old file can be erased, or the sort can be terminated, or 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 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:

$$NR = \frac{16\,000}{14 + NC}$$

where:
NR is the maximum number of input records
NC is the total number of characters in the defined sort field.

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. Either 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.

Examples:

a. SORT CLASS MEMO ALPHCLAS MEMO
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 through 10 and 25 through 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.
On 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 can not 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 can not be erased.

3.6.6 TPCOPY

Purpose:

The command TPCOPY can be used to copy tape files.

Format:

TPCOPY	TAPi	TAPo	n	YES
	TAP1	TAP2	1	no
	*	*	*	*

where:

TAPi	is the input symbolic tape unit
TAPo	is the output symbolic tape unit
n	is the number of files to be copied
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.

Notes:

1. 7 track tapes are read and written at 800 bpi, odd parity, converter on, translator off.
2. If TAPi and TAPo are not specified, TAP1 and TAP2 are assumed, which correspond to 180 and 181 respectively.
3. If n is not specified, the default is one file to be copied.
4. The maximum record size is 4096 characters.

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.

xxxxxxx RECORDS, LENGTH = xxxxx BYTES
The number of records on TAPi and the maximum length of each are specified.

xxxxxxx RECORDS COPIED
The xxxxxxx 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 cancelled.

Error Messages:

E(00001)	TAPE WRITE ERROR; TERMINATING.
E(00001)	ILLEGAL FILE COUNT FIELD.
E(00003)	SAME UNIT REQUESTED FOR INPUT AND OUTPUT

TPMATCH

Purpose: The TPMATCH command will compare two magnetic tape files.

Format: TPMATCH

Usage: One tape is mounted on device 180 and the other on device 181. Both are rewound to the load point, and the TPMATCH command is issued. The contents of the tape files are compared until an end-of-file condition is reached on either tape or two records which do not agree are encountered.

Responses:

END OF FILE ON TAPE 1
A tape mark has been read on the tape at device 180.

TAPE ERROR ON TAPE 1

An unexpected tape error has occurred on the tape at device 180.

END OF FILE ON TAPE 2
A tape mark has been encountered on the tape at device 181.

TAPE ERROR ON TAPE 2
The tape at device 181 has a bad record.

LOGICAL RECORD LENGTHS NOT EQUAL.
Program terminates if record sizes disagree.

TAPE FILES ARE NOT IDENTICAL
TPMATCH has encountered two records which do not agree.

RECORD # NNNNNN WAS THE LAST RECORD READ.
This message indicates the record number of the first records which did not match.

3.7.0 TEXT Libraries

TEXT libraries are used by the LOAD, USE, and REUSE commands to search for missing subroutines during loading. Unless the GLOBAL LOADER TXTLIB command is issued, LOAD, USE, and REUSE search only the TEXT library named SYSLIB.

For more information on library usage, refer to section 5.4.0.

3.7.1 SYSLIB TXTLIB

The library SYSLIB TXTLIB contains the FORTRAN Library as well as a number of CMS library subroutines. The TXTLIB directory indicating the entries in the library is shown in Figure 3.7.1-A. Refer to Form C28-6596, FORTRAN IV Library Subprograms (Program Numbers 360S-LM-501, and 360F-LM-619), for a description of the routines from the FORTRAN library. This section contains the descriptions of the subroutines written for use under CMS.

The routines described are:

ENTRY	INDEX	SIZE
IHCCGOTO	2	4
CGOTO#		
IHCCLABS	6	9
CDABS		
IHCCLAS	15	8
CDMPY#		
CDDVD#		
IHCCLEXP	23	9
CDEXP		
IHCCLLOG	32	12
CDLOG		
CDLG10		
IHCCLSCN	44	15
CDSIN		
CDCOS		
IHCCLSQT	59	10
CDSQRT		
IHCCSABS	69	8
CABS		
IHCCSAS	77	8
CMPY#		
CDVD#		
IHCCSEXP	85	9
CEXP		
IHCCSLOG	94	12
CLOG		
CLOG10		
IHCCSSCN	106	13
CSIN		
CCOS		
IHCCSSQT	119	9
CSQRT		
IHCDBUG	128	40
DEBUG#		
IHCFCDXI	168	11
FCDXI#		
IHCFCXPI	179	10
FCXPI#		
IHCFDUMP	189	14
DUMP		
PDUMP		
IHCFDVCH	203	7
DVCHK		
IHCFDXPD	210	9
FDXPD#		
IHCFDXPI	219	8
FDXPI#		
IHCFEXIT	227	6
EXIT		
IHCFIFIX	233	7
IFIX		
INT		
IDINT		
IHCFIXPI	240	7
FIXPI#		
IHCFMAXD	247	6
DMAX1		
DMIN1		
IHCFMAXI	253	9
MAX0		
MIN0		
AMAX0		
AMIN0		
IHCFMAXR	262	10
MAX1		
MIN1		
AMAX1		
AMIN1		
IHCFMODI	272	4
MOD		
IHCFMODR	276	6
AMOD		
DMOD		
IHCFOVER	282	7
OVERFL		
IHCFRXPI	289	8
FRXPI#		
IHCFRXPR	297	9
FRXPR#		
IHCFSLIT	306	10
SLITE		
SLITET		
IHCIBERH	316	8
IBERH#		
IHCIBERR	324	8
IBERR#		
IHCLASCN	332	12
DARSIN		
DARCOS		
IHCLATN2	344	16
DATAN		
DATAN2		
IHCLERF	360	20
DERF		
DERFC		
IHCLEXP	380	13
DEXP		
IHCLGAMA	393	18
DGAMMA		
DLGAMA		
IHCLLOG	411	14
DLOG		
DLOG10		
IHCLSCNH	425	11
DSINH		
DCOSH		
IHCLSQRT	436	9
DSQRT		
IHCLTANH	445	10
DTANH		
IHCLTNCT	455	12
DTAN		
DCOTAN		
QDTAN		
IHCSASCN	467	11
ARSIN		
ARCOS		
IHCSATN2	478	13
ATAN		
ATAN2		
IHCSERF	491	13
ERF		
ERFC		
IHCSEXP	504	10
EXP		
IHCSGAMA	514	15
GAMMA		
ALGAMA		
IHCSLOG	529	10
ALOG		
ALOG10		
IHCSSCN	539	10
COS		
SIN		
IHCSSCNH	549	10
SINH		
COSH		
IHCSSQRT	559	8
SQRT		
IHCSTANH	567	8
TANH		
IHCSTNCT	575	12
TAN		
COTAN		
QTAN		
IHCTRCH	587	17
IHCFCVTH	604	91
ADCON#		
INTGSW		
FCVEO		
FCVLO		
FCVIO		
FCVCO		
FCVAO		
FCVZO		
IHCFCONH	695	91
SAVAREA		
VFIOCS		
IBCOM#		
FDIOCS#		

FIGURE 3.7.1-A. SYSLIB TXTLIB Directory Map - October 1968

IHCFIOSH	786	63
FIOCS#		
IHCUATBL	849	8
IHCNAMEL	857	49
CPNMON		
CPNMOF		
FRDNL#		
FWRNL#		
RENAME	906	11
ERASE		
LOGDSK		
BLIP	917	13
TRAP		
IHCFAINT	930	6
AINT		
IHCLSCN	936	14
DCOS		
DSIN		
TAPSET	950	34
DEFTBLV	984	6
DEFINE	990	25
DSDSET	1015	17
REREAD	1032	47
REREADV		

184 ENTRYS IN LIBE
R; T=0.28/2.27 20.42.29

FIGURE 3.7.1-A. (continued)

3.7.1.1 TAPSET Routine

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 iteger from 1 to 4 indicating the virtual tape unit TAP1, TAP2, TAP3, or TAP4 corresponding to device address 180, 181, 182, or 183.

dsrn the FORTRAN data set reference number that will be used to reference the specified tape unit. This number must be from 1 to 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 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 to 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 to 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 through 4. For type 2 records, blksize must be an integral multiple of lrecl; refer to the Fortran IV (G) Programmer's Guide, C28-6639, 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 will be 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:

Virtual Device	Symbolic Device	Data Set Reference Number	Block Size	Modeset Code	Format Type	Logical Record Length
180	TAP1	11	80	1	1	80
181	TAP 2	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	TAPSET-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 to 15 are legal.
ABEND 5	TAPSET-RECFM CODE IS GREATER THAN 5; TERMINATING. Only the record format type numbers from 1 to 5 are legal.
ABEND 6	TAPSET-DSRN5, 6, AND 7 ARE ILLEGAL FOR TAPE UNITS.

3.7.1.2 TRAP Routine

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:

a) CALL TRAP (extrap)

where extrap is the name of an external routine which will be transferred to on an external interrupt.

b) CALL TRAP (0)

If the argument to TRAP is zero, the external interrupt return will be 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 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)                fortran testtrap
      CALL SLITE (0)                    T=2.23 15:26:58
      DO 10 K=1,10000
      CALL SLITET(2,I)                  $ testtrap
      IF (I.EQ.1) GOTO 20               EXECUTION BEGINS...
      IF (I.NE.2) GOTO 30                     50 TIMES
      IF (MOD(K,50) .EQ 0) WRITE (6,100)K    100 TIMES
10    CONTINUE                                150 TIMES
20    WRITE (6,200)                           200 TIMES
      STOP                                    250 TIMES
30    WRITE (6,300)                           300 TI"
      STOP                              e
100   FORMAT(1X,I15', TIMES'),          END OF RUN
200   FORMAT(' END OF RUN ')            IHC0021 STOP          0
300   FORMAT(' SENSE LITE ERROR')
      END
      SUBROUTINE EXTRAP
      CALL SLITE(2)
      RETURN
      END
```

3.7.1.3 BLIP Routine

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:

a) CALL BLIP ('character', count)

the count parameter must be an integer from 1 to 8 indicating the number of BLIP characters.

b) CALL BLIP ('c')

If the count is defaulted, a count of 1 is assumed.

c) CALL BLIP (0)

If the first parameter is a zero, the BLIP characters are reset to their non-printing default setting (a space followed by a backspace).

Usage:

The default setting of the BLIP characters is a sequence of non-printing 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, e.g., a single dot.

Error Returns:

None.

3.7.1.4 LOGDSK Routine

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 will be passed to CP.

3.7.1.5 REREAD Routine

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 is any data set reference number from 1-99 except unit 5, 6, or 7 and,

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 or BACKSPACE n statement must be executed before the READ statement; if a subsequent reread is issued without executing a REWIND or BACKSPACE statement, an END OF FILE condition will result. Any input/output statements can be issued between a write and a read on the reread unit.

Error Returns:

None.

3.7.1.6 DEFINE Routine

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 4 byte integer or integer variable.

name is the filename of the CMS file and it must be 8 bytes in length.

type is the filetype of the CMS file and it must be 8 bytes in length.

recno must be a 4 byte integer variable. Each file defined should use a different variable for recno. This parameter will indicate which record of the file is to be read or written.

recsiz a 4 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 will be reset by the DEFINE routine.

&n is a statement number specifying a non-standard return. This return is taken if the file does not exist.

All parameters except the non-standard return are required.

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 desired. 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 1 before any READ or WRITE statement is executed. All control operations are ignored; however, reseting the variable for recno to 1 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.

Notes:

a. The user should remember that each READ or WRITE statement increments the record number by at least 1. If the format statement indicates several records to be read or written, the record number will be 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 will undefine the old dataset and associate the dsrn with the new dataset.

e. A maximum of 20 files may be defined for random access.

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 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

Errors 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.

3.7.1.7 DSDSET Routine

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 will be used to reference the specified logical unit. This number must be from 1 to 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 to 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 through 4. For type 2 records, blksize must be an integral multiple of lrecl; refer to the Fortran IV (G) Programmer's Guide, C28-6639, for a discussion of lrecel and blksize for variable type records.

Usage:

A call to DSDSET is made to change the default settings for data set 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 defaults settings are as follow:

Data Set Reference Number	Block Size	Format Type	Logical Record Length
5	80	1	80
6	130	1	130
7	80	1	80

3.7.1.8 GETPAR Routine

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 (integer*4) in the initial parameter list of the parameter desired; it may be any non-negative value, with 0 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 will pass to statement 'last' if &last is specified.

'RITE' is an optional literal which causes the current parameter to be right-justified in its double-word 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 0, the command name or program entry will be returned. If the error return is not specified and the parameter corresponding to the item number was not specified, the routine will return without storing any parameter. Note that blanks are not returned for an unspecified parameter.

Example:

1. CALL GETPAR (PARAM (I), I, & 100)

   Parameter I will be stored in the array PARAM if it was specified; otherwise, control will pass to statement 100.

3.7.1.9 CPNMON/CPNMOF Routines

Purpose:

The CPNMON/CPNMOF 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 CPNMON
to set the namelist feature to the free format mode.

CALL CPNMOF
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 3.7.1.9-A.

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 3.7.1.9-B.

To use this namelist feature, the routine CPNMON 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 CPNMOF should be made. Thus, one can go from one to the other as desired.

Examples:

1. &name a=1., b=2., c=3.
   &end

   Data specified under the standard namelist mode.

2. 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

$ normal

EXECUTION BEGINS...
&list1
a=1.,
b=2., c=3.,4.,
5.,
&end
&LIST1
A= 1.0000000   ,B=      2,C=3.0000000   ,4.000000   ,   5.0000
&END
R;  T=0.32
```

FIGURE 3.7.1.9-A

printf freefrm fortran * * 72

```
      REAL*4A
      INTEGER B
      DIMENSION C(3)
      NAMELIST /LIST1/  A, B, C
      CALL CPNMON
      READ (5, LIST1)
      WRITE (6, LIST1)
      STOP
      END
R; T=0.03

$ freefrm
EXECUTION BEGINS...
1, 2, 3, 4, 5
      & LIST1
      & END
                A=1.0000000   ,B=      2,C=3.0000000   ,   4.0000000   ,   5.0000
R;T=0.28
```

FIGURE 3.7.1.9-B

3.7.1.10 CMS Routine

Purpose: The CMS subroutine enables the user to execute CMS commands from within a Fortran program.

Calling sequence:

CALL CMS (command, arg 1, arg 2, arg 3,...,arg N, <, &n>)

where: command is the CMS command name and it must be 8 bytes in length

arg 1,...,arg N are the arguments for the CMS command (as many as required) Each must be 8 bytes in length.

&n Where n is the (optional) error return statement number

Usage: "CMS" is used to allow Fortran programs to execute CMS commands. The number of arguments required in the CMS subroutine call is determined by the particular CMS command being executed.

Some suggestions can be made concerning the types of applications which are particularly useful. One is to alter a data file's name and type during Fortran execution to a file recognizable by Fortran, i.e., FILE FTxxFyyy. Another is to set a time limit for execution (TIMELIM) within the program to be timed. A user may wish to enter the DEBUG environment if unexpected conditions occur; he can do this via a call to CMS.

Error messages:

CMS COMMAND ERROR
An error return has been received in attempting to execute a CMS command.

COMMAND=xxxxxxxx
This message identifies the CMS command which returned an error code.

REG. 15= nnnnnnnn
The contents of register 15 on return from the CMS command are displayed. This value corresponds to the E(nnnnn) received in the CMS command environment.

Note: The arguments in the calling sequence must be either 8-byte literals enclosed in quotes ('), or REAL*8 variables.

Examples:

1.
```
CALL CMS ('ALTER   ', 'SORTDAT ', 'DATAFILE',
'P1      ', 'FILE    ', 'FT03F001', 'P1      ')
```

2.
```
REAL*8 TXEQ,MINS,SECS
DATA TXEQ/'TIMELIM '/,MINS/'25      '/,SECS/ '30      '/
CALL CMS (TXEQ,MINS,SECS)
CALL EXIT
END
```

4.0.0 CONTROL PROGRAM 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 (i.e., 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, 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, simulate the console interval timer switch, cause the Control Program to simulate an external interrupt, replace specified parts of core, ready specified I/O devices, and 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 Control Program environment is entered on the completion of the login procedure (see Section 2.2.2). 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 re-enter the Control Program from CMS, the user must hit the ATTN key once. This causes the keyboard to unlock and permits the Control Program to accept console functions. If the ATTN key is hit again while in the Control Program, the keyboard will unlock 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 will then be transferred to the environment from which the control program was entered. For instance, if the CMS command "ASSEMBLE LOOPX" had been issued and then the ATTN key 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 will be 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. Details for using BEGIN are discussed in Section 4.1.1.

4.1.C Console Function Descriptions

CP console functions are issued in the Control Program environment. Input can be entered in either upper or lower case. Entering input in lower case enable the user to distinguish the input from the output, since all output will be typed in upper case.

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 teletypes, 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.

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 will be that of the new environment. If a console function is rejected, the message "INVALID CP REQUEST" is typed out. If invalid operands are specified with a console function, the message "BAD ARGUMENT xx" is typed out where xx is the argument number. If only one operand is specified and it is invlaid, 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 will be executed properly. If a null line is entered to CP, i.e. 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	Section References
BEGIN	4.1.1
CLOSE	4.1.2
DETACH	4.1.3
DISCONN	4.1.19
DISPLAY	4.1.4
DUMP	4.1.5
EXTERNAL	4.1.6
IPL	4.1.7
LOGOUT	4.1.8
MSG	4.1.8
PURGE	4.1.16
QUERY	4.1.10

500

501

4.1.1 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 B	<hexloc>

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 will resume 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 the ATTN key 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 15CCC

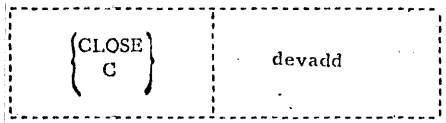
Control is transferred to hexadecimal core location 15000.

4.1.2 CLOSE

Purpose:

CLOSE releases the spooling areas from the specified multiplexor devices, and the actual I/O operation is performed for any output.

Format:



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 containing the user's identification, the date and the time of day in the following format:

CP67USERID USERID 07/25/69 09:55:16

The punched output is preceded by one card containing 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 OFFLINE READ (see Section 3.1.11).

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 will return either the first card of the next deck it has read for that user (if any), or reader-not-ready.

503

Notes:

a. "CLOSE 00E" must be given after the DUMP console function has been issued.

b. Logging out from CP will automatically issue a CLOSE for the virtual card reader, printer, and punch and release all spooling areas.

Responses:

BAD ARGUMENT.
An invalid device was specified.

Examples:

a. c E
The spooling area for the virtual printer is printed and released.

4.1.3 DETACH

Purpose:

DETACH removes the attached device whose address is specified from the virtual machine configuration.

Format:

{DETACH / DE}	devaddr

devaddr specifies the address of a device that had previously been attached to the virtual machine.

Usage:

DETACH is used in conjunction with the console function ATTACH, which can only be issued by the CP operator. Once a device has been attached to a virtual machine (see Section 4.2.0 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 address 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 will be rewound and unloaded.

Responses:

BAD ARGUMENT.
An invalid argument was specified.

DEV devadd DETACHED.
The specified device address is no longer usable. The device must be attached again for further use.

NONEXISTENT UNIT
The specified unit does not exist in the virtual machine configuration. It has either been detached or it was never attached.

Examples:

a. DETACH 181

```
console function:   detach 181
response:           DEV 181 DETACHED
```

Unit 181 is detached from the virtual machine configuration.

4.1.4 DISPLAY

Purpose:

DISPLAY types out in hexadecimal the contents of the virtual machine's core storage, general purpose registers, floating point registers, and/or program status word.

Format:

{DISPLAY / D}	[hexloc] [hexloc1-hexloc2] [Lhexloc] [Lhexloc1-hexloc2] [Greg] [Greg1-reg2] [Yreg] [Yreg1-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 the specified hexadecimal location hexloc.
Lhexloc1-hexloc2	displays in hexadecimal the contents of the specified hexadecimal locations hexloc1 through hexloc2.
Greg	displays in hexadecimal the contents of the general purpose register reg, where reg is an integer ranging from 0 to 15.
Greg1-reg2	displays in hexadecimal the contents of general purpose registers reg1 through reg2, where reg1 must be less than reg2 and each reg must be an integer ranging from 0 to 15.
Yreg	displays the contents of the floating point register reg, where reg is an integer ranging from 0 to 7. If reg is odd, it will be adjusted to the preceding even integer. The contents are displayed in hexadecimal in two forms: the internal format and in the E format.

Yreg1-reg2	displays the contents of the floating point registers reg1 through reg2, where reg1 must be less than reg2 and each reg must be an integer ranging from 0 to 7. If reg1 and/or reg2 is an odd integer, each will be adjusted to the preceding even integer. The contents are displayed in hexadecimal in two forms: the internal format and the E format.
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 full word boundary. The output is typed in multiples of a full word (8 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, up to one input line in length. If neither G, Y, 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 will be 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.

Responses:

The specified information is typed out.

BAD ARGUMENT.
An invalid argument was specified. If valid arguments were specified before the first invalid one, they were executed properly.

Examples:

a.
```
display 112403
L  12400 = 9208F17E
```

The core storage location 12403 is adjusted to the nearest full word boundary 12400 and one full word is displayed in hexadecimal

b.
```
display 12000-12010
L  12000 = 05C050E0   C7EE5830   C8BC5833   00385030   C8BE1821
```

The contents of hexadecimal location 12000 through 12010 is displayed in hexadecimal in multiples of full words.

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   00008340   00000082   00000400
```

The contents of general purpose registers 1 through 5 is displayed in hexadecimal.

e.
```
d y2
Y      2 = 0004560000000000      .1462729964O232350 E-78
```

The contents of floating point register 2 is displayed in hexadecimal in two forms: the internal format and the E format.

f.
```
d y1-6
Y      0 = 1230000000000000      .76468411734357709 E-56
Y      2 = 0004560000000000      .1462729964O232350 E-78
Y      4 = 0000004011100000      .88057543452313507 E-82
Y      6 = 0099999999990000      .51817011330510540 E-77
```

The contents of floating point registers 0, 2, 4, 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.

g.
```
d psw
PSW       = 00000000 80001374
```

The contents of the program status word (PSW) is displayed in hexadecimal.

h.
```
d g1-4 140-50 y4 psw
G      1 = 00011C98   00008990   00008340   00000082
L     40 = 00011CC0   0C000000   00011CD0   FFFFFFFF   7FFFDCCE
Y      4 = 0000000000000000      .0000000000000000 E 00
PSW       = 00000000 80001374
```

The contents of general purpose registers 1 through 4, hexadecimal locations 40 through 50, floating point register 4, and the program status word are displayed in hexadecimal with one console function.

4.1.5 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.

Format:

{ DUMP / DU }	[hexloc] [hexloc1-hexloc2] [Lhexloc] [Lhexloc1-hexloc2] [Greg] [Greg1-reg2] [Yreg] [Yreg1-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.

Lhexloc1-hexloc2 dumps in hexadecimal the contents of the hexadecimal locations hexloc1 through hexloc2.

Greg dumps in hexadecimal the contents of the general purpose register reg, where reg is an integer ranging from 2 to 15.

Greg1-reg2 dumps in hexadecimal the contents of the general purpose registers reg1 through reg2, where reg1 must be less than reg2 and each reg must be an integer ranging from 0 to 15.

Yreg dumps the contents of the floating point register reg, where reg is an integer ranging from 0 to 7. If reg is odd, it will be 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 through reg2, where reg1 must be less than reg2 and each reg must be an integer ranging from 0 to 7. If reg1 and/or reg2 is an odd integer, each will be adjusted to the preceding even integer. The contents are dumped in hexadecimal in two forms: the internal format and the E format.

PSW dumps as two hexadecimal words the contents of the program status word (PSW).

509

Usage:

Before the specified contents of core storage is dumped, alignment is made to the nearest full word boundary. The output is printed in multiples of a full word (8 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, 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 will be 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.

Responses:

After the completion of valid DUMP console functions, the keyboard is unlocked.

BAD ARGUMENT.
An invalid argument was specified. If valid arguments were specified before the invalid one, they were executed properly.

Example:

a.
```
dump g1-15
dump y0-6
dump 0-80
dump psw
c e
```

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, "C E" was issued. The dump output is shown in FIGURE 4.1.5-A.

```
 1 = 00011C98  00000990  00000842  00000400  00000035  00000000  00011CB0
 9 = 00000009  00000670  00001838  00005570  40001704  00001360
 0 = 00000000  00000000  00000000  00000000
 2 = 00000000  00000000  00000000  00000000  FFFFFFFF  FFFFFFFF
 4 = 00000000  00000000  00000000  00000000  FFFFFFFF  FF000009  50008A9A
 6 = 00000000  00000000  00000000  00000000  FFFFFFFF  00C43000  0000A10E
 8 = 00000000  00000000  00000000  00000000  FFFFFFFF  01040000  00000950
10 = 00000000  00000400  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
20 = FF000A4A  50008EA  FFFFFFFF  FFFFFFFF  FFFFFFFF  FF000009  50008A9A
40 = 00011C00  00000000  00011C80  7FFFDC8C  FFFFFFFF  00C43000  0000A10E
80 = 01040000  00005178  01040000  00007E10  01000000  00007F3C  00000950
PSW = 00000000 00001374
```

FIGURE 4.1.5-A. Output from DUMP console functions.

4.1.6 EXTERNAL

Purpose:

EXTERNAL simulates an external interrupt on the virtual machine and returns control to that machine.

Format:

```
+-----------+
|           |
|  EXTERNAL |
|     E     |
|           |
+-----------+
```

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 IPLed CMS, the Debug environment of CMS will be entered from the Control Program. If the user had not IPLed CMS, the external interrupt will cause the message

CP ENTERED, REQUEST, PLEASE...

to be typed on the terminal and the CP environment is entered.

Responses:

DEBUG ENTERED, EXTERNAL INT.
CMS was previously IPLed. When the external interrupt occurred, the Debug environment was entered.

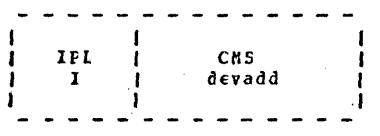
CP ENTERED, REQUEST, PLEASE.
No operating system had been previously IPLed; therefore, when the external interrupt occurred, control was returned to CP.

4.1.7 IPL

Purpose:

IPL causes the Control Program to simulate an Initial Program Load (IPL) sequence on a specified device.

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 IPLed.

Usage:

IPL simulates the IPL button on the computer console by resetting 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 will load in a copy of the nucleus which resides on disk 190. Periodically, a copy of this nucleus is saved by a CP utility. The "IPL CMS" console function will load 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" will be 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 IPLed. 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.

4.1.8 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:

LOGOUT LOG	<anything>

anything — If any non-blank 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 will be 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 'HOLD' 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 will indicate that another 'login' or 'dial' can procede.

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 non-blank character string was specified with LOGOUT; therefore the terminal remains connected to CP for another user.

516

4.1.9 MSG

Purpose:

MSG sends a specified message to either a specific user or to all users.

Format:

MSG M	userid CP	line

userid CP — specifies to whom the message should be sent. If 'CP' is specified, 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 or users.

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 and the message is not held until he logs on.

A message that is sent by MSG is in this format:

FROM userid: line

The message will be 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 will be held until a carriage return occurs.

Responses:

userid NOT RECEIVING
The user has suppressed his receiving of messages.

517

USER NOT ON SYSTEM
The specified user was not logged on the system so the message was not sent to him.

Example:

a. msg CP please attach a tape drive to 181

The message "please attach a tape drive to 181" is typed out at the systems operator's terminal.

4.1.10 QUERY

Purpose:

QUERY types out either the number of users logged on or dialing, the identification and terminal address of the users logged on, the maximum number of users allowed to log on the system, the log messages set by the CP operator, the maximum size of the console dispatching queue, and the number of spooled input and output files currently held by CP for the user.

Format:

QUERY Q	FILES LOGMSG MAX NAMES Q2 USER <names> USERID <userid>

FILES
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.

MAX
M — types out the maximum number of users that the operator allows to log on to CP. If it is zero, no maximum number has been set.

NAMES
N — types out the userid and terminal address of all users on CP.

Q2
Q — types the maximum size of the console dispatching queue.

USER — types out the number of 'virtual machine' users and the number of users attached to virtual machines using the CP-67 DIAL capacity.

USER NAMES

same function as QUERY names.

USER userid

same as QUERY userid.

userid — types out the userid and the terminal address where he is logged in, or gives 'USER NOT ON SYSTEM'.

Usage:

QUERY is used to determine the number of users logged on or dialing, who they are, and what their terminal address is, how many are allowed to log on, what the log message is, the size of the interactive dispatching queue, and the number of current spooled files for this user. 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.

nn USERS;
If the USER nn DIALED argument was specified, this message indicates that nn users are either logged in or "dialed in", respectively. If the MAX argument was specified, this message indicates the maximum number of users allowed to log on to CP, (only nn USERS printed).

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 PCH
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
HARMON - 024
OPERATOR - 009, APL -023, CJOHNSON -024
NCIMAN - 025, RIP -04B

A list of current users logged on the system is typed out on the terminal.

c. q max
00 USERS

A maximum number of users on the system has not been set by the operator.

d. q logmsg
TS DOWN AT 12 SHARE

When log message "TS DOWN AT 12 SHARE", set by the operator, is typed out.

e. q files
FILES: 01 RDR, 06 PRT, NO PCH

The number of separate spooled files for the user is shown. It is a total for all virtual card reader, printers or punches.

f. q temp21
TEMP21 ON 032

q temp4
USER NOT ON SYSTEM

4.1.11 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 simualtes 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.

Examples:

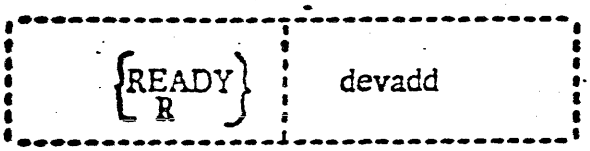
a. READY 191
A device end is simulated for the device whose address is 191.

4.1.11 READY

Purpose:

READY simulates a device end for the specified virtual address.

Format:



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 the 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.
An invalid device address was specified.

NONEXISTENT UNIT
The specified unit does not exist in the virtual machine configuration.

Examples:

a. READY 191
A device end is simulated for the device whose address is 191.

4.1.12 RESET

Purpose:

RESET simulates the system reset key on the system control panel.

Format:

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.

Response:

None.

4.1.12 RESET

Purpose:

RESET simulates the system reset key on the system control panel.

Format:

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.

Response:

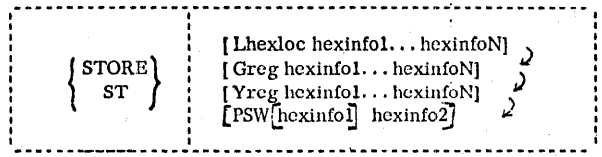
None.

4.1.13 STORE

Purpose:

STORE replaces the contents of specified locations in core storage, general purpose registers, floating point registers, and the program status word.

Format:



Lhexloc hexinfo1...hexinfoN stores the hexadecimal values hexinfo1...hexinfoN in successive full word locations starting at hexadecimal location hexloc.

Greg hexinfo1...hexinfoN 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 to 15 and successive values of reg cannot exceed 15.

Yreg hexinfo1...hexinfoN stores the hexadecimal values hexinfo1...hexinfoN in successive 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 will be adjusted to the preceding even integer.

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 will be made to the nearest full word boundary. If the hexadecimal value being stored is less than a full word (eight hexadecimal characters), it will be right adjusted in that word and filled in with high order zeroes.

The options can be combined in any other desired, separated by one or more blanks, up to one full line in length and issued in a single STORE console function. Either L, G, or Y must be specified or the option will be 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.

Response:

BAD ARGUMENT.
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.

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 full word boundary 12010. The hexadecimal number 579 is right justified, filled in with zeroes, 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 through 8 is displayed. The hexadecimal numbers 123, 456, and AA are right justified in separate words, filled in with zeroes and then stored in general purpose registers 5, 6, and 7. The contents of registers 5 through 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 is displayed. The hexadecimal number 55555 is right justified, filled with leading zeroes and then stored in the second word of the PSW and the interrupt code set to zero. The PSW is then displayed to reflect the change.

d.
```
d psw
PSW      = 00000000 80001374

st psw 111111 12000

d psw
PSW      = 00110000 00012000
```

The contents of the PSW is 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.

4.1.14 SET

Purpose:

SET allows the user to (1) save the card file in his virtual card reader, (2) control the messages and warnings typed at his terminal, (3) control his running status, and (4) convey information describing his physical terminal.

Format:

SET	CARDSAV	ON / OFF
	MSGOFF	
	MSGON	
	RUNOFF	
	RUNON	
	WNGOFF	
	WNGON	
	TIMEOUT	OFF / ON
	LINE	SHORT / LONG
	TABS	n_1 n_2 ... n_7

CARDSAV ON saves the card file in the virtual card reader so that it can be reread.

CARDSAV OFF erases the card file in the virtual card reader once the reader has been closed.

MSGOFF specifies that no messages are to be typed at the terminal.

RUNON 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.

RUNOFF places the user in the normal CP environment such that when the ATTN key is hit, the virtual machine stops.

4.1.14 SET

WNGON	specifies that all warnings are to be typed at the terminal.
WNGOFF	specifies that no warnings or messages are to be typed at the terminal.
TIMEOUT ON	specifies that the user's terminal is IBM 1050-like and lacks the timeout suppression feature.
TIMEOUT OFF	negates the effect of TIMEOUT ON.
LINE LONG	specifies that the user's terminal is teletype-like, but allows lines of 130 characters.
LINE SHORT	negates the effect of LINE LONG.
TABS	specifies physical tab settings on the user's terminal to reduce the time required to print lines.

Usage:

SET CARDSAV ON does not erase the cards in the virtual card-reader once the reader has been closed. Therefore, the SET CARDSAV ON allows the same virtaul card input to be read repeatedly from the beginning of the file. If a file is not completely read and the user wishes to re-read it, "CLOSE devadd" must be issued, where "devadd" is the address of the card-reader, to close the file and reposition pointers to the beginning of the file. The operation is effective for all the user's virtual card readers.

SET CARDSAV OFF is the normal mode of operation for the card-reader. Once a virtual input 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 MSGON and WNGON, respectively. Any messages directed to a user or broadcast by the operator will be typed at the terminal whenever the terminal is not ready for input. A warning sent by the CP operator will print immediately regardless of what is occurring at the terminal. If MSGOFF is specified, only warnings from the operator will type at the terminal, and any messages sent by another user or the operator will be lost.

Once MSGOFF or WNGOFF has been specified and the user desires to accept messages or warnings, SET MSGON or SET WNGON should be issued.

If WNGOFF is specified, no warnings or messages will ever type on the terminal. If WNGOFF and MSGON are both set, only messages will type on the terminal. SET WNGOFF should be used with discretion.

The normal mode of operation for running is RUNOFF. When the ATTN key is hit, the virtual machine stops running and the terminal waits for a CP console function. In a multi-access virtual machine, this is an acceptable method of operation; therefore, SET RUNON can be issued to CP to allow the virtual machine to continue running when the ATTN key is hit. When RUNOFF is issued after RUNON had been previously set, the virtual machine will continue to run until the ATTN key is hit; this will cause the machine to stop for input. The SET RUNOFF mode is automatically set if the user gets an "idle" virtual machine, i.e. with the message

CP ENTERED, REQUEST, PLEASE.

For specific terminals (e.g., the IBM 1050), when the system is awaiting input from the terminal, the keyboard will lock if no data has been entered for any 15 second interval during the "read" request. If SET TIMEOUT ON has been entered, the keyboard will unlock and allow the user to resume input.

SET TIMEOUT OFF is the normal mode of operation and indicates that the timeout situation need not be considered.

The standard teletype has an 80 character carriage; CP, therefore, will split a long line destined for a teletype or teletype-like device into two shorter lines. However, there are certain teletype-like devices that have full 130 character carriages. The user may indicate that he has such a terminal by entering SET LINE LONG, thus requesting full 130 character output lines.

SET LINE SHORT is the normal mode of operation for teletype users and indicates an 80 print position carriage.

SET TABS informs CP where the user has placed physical tab settings on his terminal. When a line is presented for terminal output, CP determines whether the user has requested that tabs be inserted in his output line. If so, any string of at least three (3) blanks extending to the user's tab settings will be removed, and replaced by a tab character. This reduces the number of characters in the line to be typed, and, thus, reduces the amount of time needed to type a full line of output.

To alter a previous setting, simply issue a new SET TABS command which will replace the old settings with the new. A SET TABS command with no specified tab setting will nullify any previous SET TABS command No tabs are assumed to be set until the first issuance of this command. A maximum of seven tab settings may be specified, and the tab settings specified must be in increasing order.

Responses:

BAD ARGUMENT xx
An invalid argument was specified.

Examples:

a. SET CARDSAV ON.

The virtual input file will be saved and not erased once the reader is closed

b. SET MSGOFF.

No messages will type at the terminal, only warnings from the operator.

c. SET RUNON

The virtual machine will immediately continue execution. If the user now activates the ATTN key, a CP 'read' will occur, but his virtual machine will continue to run.

d. SET TIMEOUT ON

The user has an IBM 1050 terminal without the timeout suppression feature. If his keyboard locks, it will subsequently unlock and become available again.

e. SET LINE LONG

The user has a teletype-like terminal with a 130 character carriage. The full width of this carriage may be used for terminal output.

f. SET TABS 10 16 31 40

The user's terminal has tabs physically set in columns 10, 16, 31, and 40 (relative to the left margin, which is column 1). All output to this terminal will be edited for possible insertion of tab characters.

g. SET TABS

This nullifies the effect of any previous SET TABS command.

4.1.15 XFER

Purpose:

XFER controls the passing of files between users.

Format:

XFER	DEVADD	TO USERID / OFF

DEVADD — Specifies the address of the device from which all succeeding output is to be either transferred or normally written out. The acceptable devices are the card-punch, normally 00D, or the printer, normally 00E.

TO USERID: — Turns on the transfer mode. "USERID" is the eight-character user identification of the user that is to receive the transferred file.

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" will be placed in the card-reader of "USERID". For "USERID" to receive the transferred files on his disk, "USERID" must read them into his files. i.e., by issuing commands such as "OFFLINE READ FILENAME FILETYPE" or, in the case of punched files, "DISK LOAD." To turn the transfer mode off, the option "OFF" must be specified. Any succeeding output to the "DEVADD" will be 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" will automatically turn off the XFER to the previous "USERID."

Responses:

BAD ARGUMENT.
An invalid argument was specified.

NONEXISTANT UNIT.
The specified unit does not exist and is invalid.

Examples:

A. XFER D TO USER1
The transfer mode is turned on. Any succeeding output to device 00D will be placed in the card-reader of USER1.

B. XFER D OFF
The transfer mode is turned off. Any succeeding output to virtual 00D will be written on the real device.

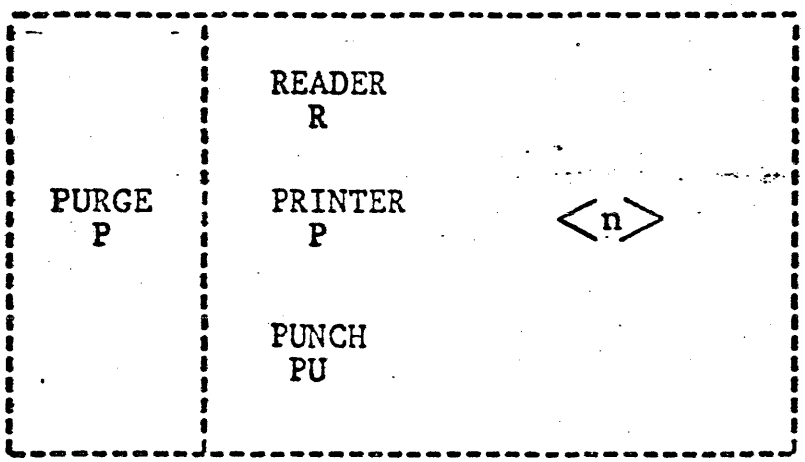
C. XFER E TO USER
The transfer mode is turned on. Any succeeding output to device 00E will be placed in the card-reader of USER1.

4.1.16 PURGE

Purpose:

PURGE allows the user to delete some or all spooled files still in the spooling area from his virtual printer or punch, or to delete some or all of the spooled input for his virtual card reader without reading the data.

FORMAT:



where n is the number of files to be purged, expressed as a decimal integer. If n is greater than or equal to the user's total number of files for that virtual device, all of the files for that virtual device are purged. The number n, if specified, must be greater than zero. If n is omitted, all files for that device will be purged.

Usage:

If the user determines that he does not require the first n (possibly all) files in his virtual printer, punch, or card reader, the PURGE command can be issued.

Responses:

BAD ARGUMENT XX

An invalid device type was specified, of the number of files to be purged was invalid.

4.1.16 PURGE

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 reader 6
04 FILES PURGED
The user wished to purge the first 6 reader spooled files. Only 4 such reader files existed, and these 4 were subsequently purged.

the userid is invalid.

If the user "XFERed" tc is currently logged on to CP-67, he will receive the following message:

** CARDS XFER'D BY userid **

where userid is the transferring user.

Notes:

A user may dc an XFER to himself tc allcw him to read a card file he created without requiring operator intervention and at a considerable savings in card handling.

Exercises:

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 tc clear the status.

Examples:

a. xfer d to user1

The transfer mode is turned on. Any succeeding output to device 00E will be placed in the card-reader of USER1.

r. xfer d cff

The transfer mode is turned cff. Any succeeding output to virtual 00E will be written on the real device.

c. xfer d to userA4

USERA4 NOT IN DIRECTORY

USERA4 does not exist, therefore cards will not be transferred to him. Any active transfer mode is turned cff.

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.

4.1.16 PURGE

532

Purpose:

PURGE allows the user to delete all spooled files (not currently processed for output) 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 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. pur r
02 FILES PURGED

The two spooled input files for the user have been deleted.

4.1.17 SLEEP

533

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 will place 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:

a. 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 will of course not receive messages, only warnings, if they are issued.

4.1.18 SPOOL

Purpose:

The SPCOI console function allows the user to (1) direct his spooled output to a specific unit record device at the computing facility and (2) to control the nature of reading spooled input files.

Format:

```
+-----------------------------+
|        |                    |
|        |  xxx <ON yyy>      |
| SPCOI  |                    |
|        |  ccc <CONT>        |
|        |                    |
+-----------------------------+
```

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 will be 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 can not be directed to the real printer.

To discontinue the directed spooling and return to normal spooling, issue SPOOL specifying only the virtual device address.

SPCOI 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 will receive 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 only the virtual card reader address.

Note:

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 usefull and necessary if, for instance, an installation has two printers; one with a "PN" train and another with a "TN" train. If th user has "script" output to produce, he may specify the desired output printed, perform his printing, and then return his printer to normal spooling.

Responses:

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 cn 30
begin
CMS
script print report (offline center)
R;T=05.21
                <--------ATTN key hit
CP              <--------confirmation of environment
sp e
```

The SPCOI 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. THE ATTN key is hit to go to CP and the directed output of printer 00E is terminated. Normal spooling will now occur on 00E.

b. spool c cont

```
begin
CMS
offline read * *
OFFLINE READ A FORTRAN
OFFLINE READ E FORTRAN
OFFLINE READ CALC FORTRAN
R;T=02.01
          <--------ATTN key hit
CP
query files
FILES: - NO RDR, 03 PRT, NO PCH
spool c
```

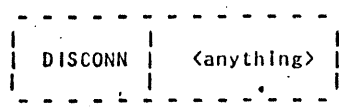
The SPOOL command is issued to read multiple spooled input files as if they were continuous. Control is transferred to CMS by BEGIN and the spooled input files are read. The ATTN key is hit to return to CP and the spooled files are quered. Input spooling is then returned to normal for non-continuous reading.

4.1.19 DISCONNECT

Purpose:

DISCONN causes the terminal to be disconnected from the virtual machine and the virtual machine continues to run.

Format:



anything is any non-blank character string that signifies the phone line is not to be hung up upon DISCONNecting.

Usage:

DISCONN causes the terminal to be disconnected from the virtual machine. The virtual machine will continue to run as though the user had issued BEGIN. Any "writes" to the terminal (virtual console) are ignored. The virtual machine will be automatically logged out if (1) a "read" is attempted from the terminal, or (2) the virtual machine goes into the disabled wait state. The DISCONNected virtual machine will run with low priority in Q2 of the dispatcher.

If the <anything> option is selected, the line will not be disabled. The terminal can then be used to "LOGIN" with another "userid" or to DIAL into a multi-access system.

At a later time, a user can "re-connect" with his DISCONNected machine simply by following the normal "login" procedure. This "re-connection" can be made from any terminal. The message:

RECONNECT AT xx/xx/xx ON xx/xx/xx

will be printed upon re-connection and the terminal will be placed in console function mode. To continue running the virtual machine a BEGIN is required.

4.1.20 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 LI	userid xxx yyy <W,R> <(NOPASS)>

userid	The name of the user "owning" the device xxx
xxx	The "owner's" virtual device address in hexidecimal
yyy	The hexidecimal 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 (i.e. a "public" device).

Usage:

LINK allows attachment of directory-defined virtual disks to the requestor's virtual machine. The device must be described in the CP-67 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. However, if write access is normally allowed, but 'W' is not specified, read-only access will be set. This allows the requestor to establish write protection on a device.

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 will be made in read-only mode.

Responses:

ENTER PASSWORD:

Type the required password. By convention, devices specified as "public" have the password 'ALL'. A null line will default to ALL.

BAD ARGUMENT

Missing or invalid arguments.

COMMAND ACCEPTED

The request has been honored as given.

READ-ONLY

A write request cannot be honored due to existing read link(s).

NONEXISTENT UNIT

Device xxx not found in the specified directory.

ALREADY ATTACHED

Requestor already has a device yyy.

BAD PASSWORD

The supplied password is not valid, or the device is not sharable.

USER LOGGING IN

'userid' is in process of logging in. Try again later.

DEVICE IN USE

A write link exists. LINK denied.

UNIT NOT READY

The required physical volume is not mounted. Notify system operator.

UNIT NOT DASD!

Device xxx must be a DASD! device for LINK.

USER NOT ON SYSTEM

'userid' is not in the directory.

4.2.0 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 when BEGIN is issued or the ATTN key is hit.

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. Issue "IPL CMS" or "IPL 190" again 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.

PURPOSE:

The CARD command can be used to control the order of spooled reader files.

FORMAT:

N1 Specifies the reader file which is to be moved. It can be any number 01 to NN depending on how many reader files the user has spooled. But it must not exceed the total number of files available.

FIRST Specifies that the reader file designated by N1 is to be moved to the beginning of the reader file chain, making it the first file in the reader.

LAST Specifies that the reader file designated by N1 is to be moved to the bottom of the reader file chain, making it the last file in the reader. Any new file read in is located last.

N2 Specifies the reader file that is to be swapped with N1. It can be any number 01 to NN. When a swap is made all other reader files remain in their same relative positions.

RESPONSES:

COMMAND ACCEPTED
The file designated by N1 was successfully moved.

FILE NOT FOUND
The N1 or N2 file is not present in the reader.

BAD ARGUMENT
An invalid argument was specified.

EXAMPLES:

When there are four files in the reader, the following commands are given:

A. CARDS 03 FIRST
The third file in the reader becomes the first;

B. CARDS 01 LAST
The first file in the reader becomes the last;

C. CARDS 02 04
The second file becomes the fourth and the fourth becomes the second. The first and third remain the first and third;

D. CARDS 04 02
Same results as C.

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 will be rewound and unloaded.

5.1.0 RECOVERY PROCEDURES

The recovery procedures discussed in this section 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, re-initializing CMS, file space full, and general recovery procedures.

Errors During LOGIN: If I/O errors occur during the CMS command "LOGIN UFD" or after CMS initialization and either a carriage return or the first command has been issued, CMS has recognized an error condition while trying to read in the user file directory and will repeat the LOGIN procedure. The user will be asked to LOGIN again. Re-issue the command "LOGIN UFD". If the I/O error still occurs, enter the Control Program by hitting ATTN and issue "IPL CMS" to reload the Cambridge Monitor System. Issue "LOGIN UFD" again. If the I/O error still persists, contact the responsible system programmer, as the user file directory is unreadable or has been destroyed.

If I/O errors occur during the CMS "LOGIN NO_UFD" command, issue "FORMAT P" to re-initialize 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 Appendix C. 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 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 will be 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 when the system goes down, it may not be completely lost. Issue a LISTF and see if the EDIT work files

"INPUT FILE" or "INPUT1 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.

ALTER the filename and/or filetype of the appropriate work file. Then issue EDIT for the ALTERed 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.

Re-initializing CMS. If DEBUG is entered, CMS can be re-initialized as follows: (1) the CMS "IPL" command can be issued from the DEBUG environment or the CMS command environment, (2) the ATTN key can be hit and "IPL CMS" or "IPL 190" issued to CP, or (3) "KX" issued in DEBUG and then "IPL CMS" issued to CP. Issuing "KX" to DEBUG is the only way among the previous three methods of re-initializing CMS to permanently update files that have been updated or created since the last T=xx.xx message. The Restart request in DEBUG should not be used as an all-purpose re-initializer, as it is primarily a special debugging feature for checkout of experimental systems and it is basically an in-core initializer.

If CMS does not work as it should, it could be that the copy of CMS in ths 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 the ATTN key and issue "IPL CMS" to reload CMS, or (2) issue "IPL" to the CMS command environment.

The commands "IPL", "LOGIN", and "FORMAT" can be issued at any time in the CMS command environment 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" or "DUMPREST" and then erase those files on disk. Whenever the files on tape are needed, the commands "TAPE LOAD" (if "TAPE DUMP" created the tape) or "DUMPREST" (if "DUMPREST" 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 carrige return and the response will confirm which environment has been entered.

To kill execution of a command or program in CMS, the ATTN key should be hit twice and "KX" entered. To kill a type out in CMS, the ATTN key should be hit twice and "KT" entered. To kill overrides in CMS, the ATTN key should be hit twice and "KO" entered. To truncate a line to 72 characters that is being typed out at the terminal, the ATTN key should be hit twice and "KE" 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 (see Section 3.3.2) or, if the program is written in Fortran, use the Fortran G Debug package (see IBM System/360: FORTRAN IV Language, Form C28-6515).

If transmission errors occur, issue the ECHO command, using each of the options U, S, and X (see Section 3.5.2) to test the transmission of data between the terminal and the computer.

5.2.C OFFLINE PROCEDURES

The 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 (i.e., 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 will be 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 will be typed at the console.

CARDS HAVE BEEN READ

If the deck was read by CP before the user logged in, the following message will be 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 will not be entered into the user's directory.

5.3.0 CHANGING OBJECT PROGRAMS

Files which contain relocatable object code and have a filetype of TEXT can be read into core storage and have their linkage's resolved by the LOAD, USE, and REUSE commands. Three types of cards can be added to a TEXT file. These are the Set Location Counter (SLC), the Include Control Section (ICS), and the Replace (REP) cards, which are used to set the core location where LOAD will begin placing the file in core and to make corrections and additions to the relocatable object code in core once the file is loaded. These cards can be added to the TEXT file(s) which have been OFFLINE PUNCHed and then they can be read back in, or they can be added using the SPLIT and COMBINE command or the EDIT command.

Set Location Counter Card. The Set Location counter card sets the location counter used with the loader. The file loaded in after the SLC card will be placed in core beginning at the address set by this SLC card. The SLC card has the format shown in FIGURE 5.3.0-A. It sets the location counter in one of three ways:

1. With the absolute address specified as a hexadecimal number in columns 7-12.

2. 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.

3. 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 will 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.

548

Set Location Counter Card

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	Address in hexadecimal (to be added to the value of the symbol, if any, in columns 17-22). The address must be right-justified in these columns, and unused leading columns filled in with zeros.
13-16	Blank.
17-22	Symbolic name, whose assigned location will be used by the loader. The symbol must be left-justified in these columns. If left 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 them blank or insert program identification for his own convenience.

FIGURE 5.3.0-A. Format of an SLC card.

Include Control Section 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 5.3.0-B. An ICS card must be placed at the front of the card deck or TEXT file.

Replace 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 5.3.0-C. 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 Include Control Section card, which defines the total length of the control section, must be placed at the front of the deck.

Include Control Section Card

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	Name of control section, left-justified in these columns.
23	Blank.
24	, (comma)
25-28	Length (in bytes) in hexadecimal notation of the control section. This must not be less than the actual length of a previously specified control section. The number must be right justified in these columns, and 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 them blank or insert program identification for his own convenience.

FIGURE 5.3.0-B. Format of an ICS card.

Replace Card

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	Starting address, in hexadecimal, of the area to be replaced, as assigned by the assembler. It must be right-justified in these columns, and unused leading columns filled in with zeros.
13-14	Blank.
15-16	External Symbol Identification (ESID), which is 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 half-word (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 them blank or insert program identification for his own convenience.

FIGURE 5.3.0-C. Format of a REP card.

5.4.0 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 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.

Both the user and the system may have macro libraries. The system macro library has the identifier SYSLIB MACLIB SY and resides on the system disk. It is composed of commonly used macro definitions. The user can generate his own libraries on his permanent disk by issuing the MACLIB command (see Section 3.1.10).

Macro libraries are searched during the ASSEMBLE command for missing macro definitions. Normally the only macro library searched is SYSLIB MACLIB SY. 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 (see Section 3.2.2). This command names from one to six MACLIB files that are to be searched for missing macro definitions. If the file SYSLIB MACLIB is to be searched in addition to the user's MACLIB files, SYSLIB must be specified as one of the six 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 either another GLOBAL ASSEMBLER command is issued, the CMS nucleus is re-initialized, or the user logs out of CP.

The second method is by generating a SYSLIB MACLIB file on either the permanent or temporary disk. The ASSEMBLE command normally searches for the file SYSLIB MACLIB for missing macro definitions. In looking for SYSLIB MACLIB, the ASSEMBLE command searches the permanent disk, the temporary disk, and then the system disk in that order. If a user file has the identifier SYSLIB MACLIB, it will be used in place of the system macro library.

To terminate the searching of all MACLIB files, including SYSLIB 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. The system text library has the identifier SYSLIB TXTLIB SY and resides on the system disk. It is composed of the FORTRAN library subroutines. The user can generate his own text libraries on the permanent disk by issuing the TXTLIB command (see Section 3.1.16).

Text libraries are searched by the LOAD, USE, and REUSE commands to find missing subroutines when resolving external references. Normally the only text library searched is SYSLIB TXTLIB SY. Once 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 (see Section 3.2.2). This command names from one to four 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 four 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 with no libnames specified. This resets SYSLIB TXTLIB as the only text library to be searched. The GLOBAL text libraries remain in effect until either another GLOBAL LOADER command is issued, the LOAD command is issued with the LIBE option, the CMS nucleus is re-initialized, 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. (See Section 3.2.3). 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 either the permanent or temporary disk. LOAD, USE, and REUSE normally search for the file SYSLIB TXTLIB for missing subroutines. In looking for SYSLIB TXTLIB, these commands search the permanent disk, the temporary disk, and then the system disk, in that order. If a user file has the identifier SYSLIB TXTLIB, it will be used in place of the system text library.

5.5.C 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 TECCEY 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 CE 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 will automatically type the message:

DEVICE xxx ATTACHED

Since there are only a limited number of tapes available for use by CMS user, 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 CE environment and type the CP console function:

DETACH devadd

When a user logs out of CP, all attached devices are automatically detached.

6.0.0 SAMPLE TERMINAL SESSION

A sample terminal session is given on the following pages. User input is in lower case; typeout from the system appears in upper-case. The following is a description of the terminal session:

After logging in to CP and utilizing CMS, a LISTF command is issued to obtain a list of all files stored on user CSC1's permanent disk. 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 re-edited.

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 LOAD command will also cause execution to begin after the file is loaded.

LISTF and ERASE operands 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 attention key is hit twice.

The OFFLINE command punches or prints the specified file on offline devices. STAT types out statistics regarding the amount of disk space used and remaining. The ALTER command changes the identifier of a file. KX, entered after hitting the attention key twice, stops execution of the current program and returns control to CP.

A new copy of CMS is obtained by issuing the IPL console function and an EXEC file (consisting of CMS commands) is created and filed. The file is then executed by issuing the EXEC command, which will cause each of the commands contained in the file to be executed individually. Operand substitution is illustrated by modifying and re-executing the file using & arguments.

Hitting the attention key 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.

Each of the commands illustrated in this terminal session is discussed in detail in the section dealing with that command. For a brief description of each of the CMS commands, refer to Appendix A. Command formats and references to the sections dealing with each command are given in Appendix B.


```
#
CP/67 Online    XDh65 Qsyosu

l csc1 <-------------------------Specify your own userid upon logging in.
ENTER PASSWORD:
      <-------------------------The protected password does not
CP WILL BE UP UNTIL 1500 TODAY.      print when entered.
READY AT 09.55.29 ON 04/12/68

ipl cms
CMS...VERSION 1.0 - 05/01/68

listf
FILENAME FILETYPE MODE NO.REC.
INDIAN   LISTING   P1   003
DUMPREST SYSIN     P1   009
SUPERSCR SYSIN     P1   070
MY       FORTRAN   P1   001
INDIAN   TEXT      P1   002
FORTCLG  EXEC      P1   001
LOAD     MAP       P5   003
DUMPREST SYSUTIL   P1   019
FIN      SCRIPT    P1   001
TUES     SCRIPT    P1   001
FRST     SCRIPT    P1   001
DUMPREST LISTING   P1   007
AGENDA   SCRIPT    P1   002
INDIAN   FORTRAN   P1   001
R; T=0.06

edit main fortran
FILE DOES NOT EXIST; WILL BE CREATED.
INPUT:
c main program  April 12, 1968
      write (6,10)
10#format (' a = ')
#read (5,20) a
20#format (8,3)
#write (6,25) a,x
#end
   <-------------------------Null line typed to transfer to Edit
EDIT:                          environment.
file
R; T=0.11

fortran main
   0004       20    FORMAT (8,3)
                            $
        01)   ERR 13 SYNTAX
COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
E(00032); T=0.32
```

```
edit main fortran
EDIT:
p 20

C MAIN PROGRAM   APRIL 12, 1968
      WRITE (6,10)
10    FORMAT (' A = ')
      READ (5,20) A
20    FORMAT (8.3)
      WRITE (6,25) A,X
      END
EOF REACHED BY:
P 20
1 /format/
10    FORMAT (' A = ')
1 /format/
20    FORMAT (8.3)
c /8/f8/
20    FORMAT (F8.3)
u 2
10    FORMAT (' A = ')
c / '/ ?'/
10    FORMAT (' A = ?')
f 20
20    FORMAT (F8.3)
i *x = a**2

EDIT:
p
      X = A**2

t
p 20

C MAIN PROGRAM   APRIL 12, 1968
      WRITE (6,10)
10    FORMAT (' A = ?')
      READ (5,20) A
20    FORMAT (F8.3)
      X = A**2
      WRITE (6,25) A,X
      END
EOF REACHED BY:
P 20
file
R; T=0.51

fortran main
                                ERR 22      UNDEFINED LABEL
      25
COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
E(00032); T=0.40
```



```
edit main fortran
EDIT:
1 /25/
      WRITE (6,25) A,X
i 25*format (' a = 'f8.3,' x = 20.3)
p
25    FORMAT (' A = 'F8.3,' X = 20.3)
c /20/' f20/
25    FORMAT (' A = 'F8.3,' X = ' F20.3)
file
R; T=0.14

$ ma¢                    <-------- The ¢ deletes the line.
fortran main
R; T=0.46

$ main
EXECUTION BEGINS...
 A = ?
2.5
 A =    2.500 X =                  6.250
R; T=0.33

load main
R; T=0.25

start
EXECUTION BEGINS...
 A = ?
3.1
 A =    3.100 X =                  9.610
R; T=0.05

load main (xeq)
EXECUTION BEGINS...
 A = ?
3.2
 A =    3.200 X =                 10.240
R; T=0.28

listf main *
FILENAME FILETYPE MODE NO.REC.
MAIN     LISTING   P1   003
MAIN     FORTRAN   P1   001
MAIN     TEXT      P1   002
R; T=0.03

listf * listing
FILENAME FILETYPE MODE NO.REC.
INDIAN   LISTING   P1   003
MAIN     LISTING   P1   003
DUMPREST LISTING   P1   007
R; T=0.01

erase * listing
R; T=0.03

listf * listing
FILE NOT FOUND
E(00002); T=0.03
```

```
printf main fortran

C MAIN PROGRAM   APRIL 12, 1968
      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)
      END

R; T=0.06

printf main fortran * 3 25

C MAIN PROGRAM   APRIL 12
      WRITE (6,10)
10    FORMAT (' A = ?')

R; T=0.04

printf main fortran

C MAIN PROGRAM   APRIL 12, 1968
      WRITE (6,10)
10 "

kt
R; T=0.06

offline punch main text@@@@fortran
R; T=0.07

offline print main fortran
R; T=0.06

offline printcc main listing
FILE NOT FOUND
E(00002); T=0.02

listf
FILENAME FILETYPE MODE NO.REC.
DUMPREST SYSIN     P1   009
SUPERSCR SYSIN     P1   070
MY       FORTRAN   P1   001
FORTCLG  EXEC      P1   001
LOAD     MAP       P5   003
DATA     FORTRAN   P1   001
DUMPREST SYSUT1    P1   019
DUMPREST SYSUT2    P1   019
FIN      SCRIPT    P1   001
TUES     SCRIPT    P1   001
FRST     SCRIPT    P1   001
DUMPREST SYSUT3    P1   001
AGENDA   SCRIPT    P1   001
MAIN     TEXT      P1   002
INDIAN   FORTRAN   P1   001
R; T=0.05

stat
P-DISK:  0142 RECORDS IN USE, 0258 LEFT (of 0400), 36% FULL (of 010 CYL.)
R; T=0.02
```

The ATTN key was hit twice to enter KT for killing the typeout.


```
alter main fortran * mainone * *
R; T=0.02

listf main fortran
FILE NOT FOUND
E(00002); T=0.03

listf * fortran
FILENAME FILETYPE MODE NO.REC.
MY       FORTRAN   P1   001
MAINONE  FORTRAN   P1   001
INDIAN   FORTRAN   P1   001
R; T=0.01

$ main
```

The ATTN key was hit twice to enter KX for killing execution of $.

```
kx
KILLING CMS EXECUTION...
P-DISK:  0142 RECORDS IN USE, 0258 LEFT (of 0400), 36% FULL (of 010 CYL.)
TOTAL CPU-TIME (IN SECONDS) = 11.18
CP ENTERED,  READY.

ipl cms
CMS...VERSION 4.0 - 04/01/68

listf mainonnge *
FILENAME FILETYPE MODE NO.REC.
MAINONE  FORTRAN   P1   001
R; T=0.03
```

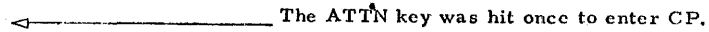
The @ deletes one character.

```
edit fortclgo exec
FILE DOES NOT EXIST; WILL BE CREATED.
INPUT:
fortran mainone
$@load mainone (xeq)

EDIT:
file
R; T=0.10

printf fortclgo exec

FORTRAN MAINONE
LOAD MAINONE (XEQ)

R; T=0.04

exec fortclgo
FORTRAN MAINONE
LOAD    MAINONE (XEQ)
EXECUTION BEGINS...
 A = ?
3.4
 A =    3.400 X =              11.560
R; T=0.86
```

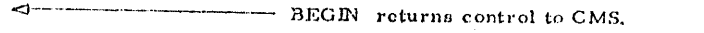
```
edit fortclgo exec
EDIT:
c /mainone/ &1/ * g
FORTRAN  &1
LOAD  &1 (XEQ)
EOF REACHED BY:
C /MAINONE/ &1/ * G
file
R; T=0.11

exec fortclgo mainone
FORTRAN MAINONE
LOAD    MAINONE (XEQ)
EXECUTION BEGINS...
 A = ?
5.1
 A =    5.100 X =                26.010
R; T=0.89

edit fortclgo exec
EDIT:
I &set err exit
p 9
&SET ERR EXIT
FORTRAN  &1
LOAD  &1 (XEQ)
EOF REACHED BY:
P 9
file
R; T=0.11

edit mainone fortran
EDIT:
p 4

C MAIN PROGRAM   APRIL 12, 1968
      WRITE (6,10)
10    FORMAT (' A = ?')
b aa
      FORMAT (' A = ?')
file badone
R; T=0.15

listf * fortran
FILENAME FILETYPE MODE NO.REC.
AY       FORTRAN  P1   001
MAINONE  FORTRAN  P1   001
INDIAN   FORTRAN  P1   001
BADONE   FORTRAN  P1   001
R; T=0.03

exec fortclgo badone
FORTRAN BADONE
    0002              FORMAT (' A = ?')
                             $
           01)  ERR 02  LABEL
COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).
 !!! E(00032) !!!
R; T=0.44
```



```
edit fortclgo exec
EDIT:
c /&1/&1 &2 &3 &4 &5/ * g
FORTRAN  &1 &2 &3 &4 &5
LOAD  &1 &2 &3 &4 &5 (XEQ)
EOF REACHED BY:
C /&1/&1 &2 &3 &4 &5/ * G
file
R; T=0.12

exec mefortclgo mainone
FORTRAN MAINONE
LOAD    MAINONE (XEQ)
EXECUTION BEGINS...
 A = ?
1.9
 A =    1.900 X =                3.610
R; T=0.93
```




```
q user
10 USERS

q user names
CURRENT USERS ARE..
 MADNICK  ON 029
 MARK     ON 04C
 MEYER    ON 047
 OPERATOR ON 009
 BAYLES   ON 049
 CJONES   ON 020
 ROSATO   ON 027
 WHO      ON 024
 SEYMOUR  ON 045
 CSC1     ON 028

q logmsg
CP WILL BE UP UNTIL 1500 CONTINUOUSLY.

begin
```




```
CMS
logoug
INVALID CMS COMMAND

logout
CMS LOGGING OUT...
TOTAL CPU-TIME (IN SECONDS) = 2.99
CP ENTERED,  READY.

)
LOGOFF AT 12.06.06 ON 04/12/68
#
CP/67 Online    XDh65 Qsyosu
```

7.0.0 CMS Batch Monitor

A Batch Monitor for running Express type jobs has been implemented and can be run from a terminal along with other conversational CMS users. The Batch Monitor permits a user to compile routines using the FORTRAN-G compiler and/or translate Assembly language programs using the ASSEMBLER-F translator. Programs can be selectively assembled or compiled and then executed. 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.

7.1.0 A CMS 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 a filetype. FORTRAN, ASSEMBLE, and TEXT control cards indicate that the filetype should be FORTRAN, SYSIN, and TEXT respectively. Unless specified, the filename will be 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 a filename the same as the filename of 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, 7, or 8. Files created by writing on one of the above FORTRAN logical units will be identified in the same way. These files may be PRINTed or PUNCHed using the appropriate control card and specifying the filename and filetype.

Execution of object modules (TEXT files) is initiated by the GO control card. This control card causes all TEXT files to be loaded into core. Execution is begun at the entry point specified in the GO control card or, if defaulted, at the first program encountered in the job stream. If during the execution of a program a program interruption occurs, the programmer will get a complete core dump and the job will be 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 will be ignored. An example of a CMS Batch job deck is shown in Figure 7.1.0-A.

7.2.0 CMS Batch Control Cards

A deck to be submitted for CMS Batch must have an OS/360 standard job card with a programmer's name specified at the beginning of the deck. 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 listed in Figure 7.2.0-A and 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 procedures names which are recognized and the CMS Batch control cards they are equivalent to are specified below:

OS/360 procedure names	CMS Batch control cards
FORTRAN	// FORTRAN
LARGFORT	// FORTRAN
LINKEDIT	// TEXT
LINKOBJ	// TEXT
LINKSRC	// TEXT
EXECUTE	// GO
DISKEXEC	// GO
ASSEMBLE	// ASSEMBLE
ASEMLINK	// ASSEMBLE
PUNCH	// PUNCH
TPPCHOBJ	// PUNCH

Note:

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. Also, only one job step per procedure call is allowed.

```
    Standard Job Card
    Standard Name Card
//  FORTRAN
      .
      .
    Fortran Program 1 Source Code
      .
      .
    END
      .
      .
    Fortran Program 2 Source Code
      .
      .
    END
//  ASSEMBLE ABLE
      .
      .
    Assembly Program 1 Source Code
      .
      .
    END
      .
      .
    Assembly Program 2 Source Code
      .
      .
    END
//  PUNCH ABLE
//  TEXT
      .
      .
    Object Module
      .
      .
//  GO
      .
      .
    Data Cards
      .
      .
//  PRINT FILE FT01F001
//  ASSEMBLE ABLE NOLIST
      .
      .
    Second Version of Assembly Program ABLE
      .
      .
    END
//  DATASET FILE FT02F001
      .
      .
    Data Cards
      .
      .
//  GO ABLE
```

Figure 7.1.0-A. Sample CMS Batch Stream

1.	// FORTRAN	[filename / *]	[MAP / NOMAP]	[LIST / NOLIST]	[BCD / EBCDIC]
2.	// ASSEMBLE	[filename / *]	[LIST / NOLIST]	[XREF / NOXREF]	[RENT / NORENT]
3.	// TEXT	[*]			
4.	// DATASET	filename	filetype		
5.	// GO	[entry point / default entry point]			
6.	// PUNCH	filename	[filetype / TEXT]		
7.	// PRINT	filename	filetype		

Figure 7.2.0-A. CMS Batch Control Cards

1. // FORTRAN [filename / * .] [MAP / NOMAP] [LIST / NOLIST] [ECD / EBCDIC]

2. // ASSEMBLE [filename / *] [LIST / NOLIST] [XREF / NOXREF] [RENT / NORENT]

3. // TEXT [*]

4. // DATASET filename filetype

5. // GO [entry point / default entry point]

6. // PUNCH filename [filetype / TEXT]

7. // PRINT filename filetype

8. // LINK userid devadd password

9. // RETURN filename filetype

Figure 7.2.0-A. CMS Batch Control Cards

7.2.1. / / FORTRAN

Format:

/ / FORTRAN [filename | *] [MAP | NOMAP] [LIST | NOLIST] [BCD | EBCDIC]

filename	specifies the name of the file which will contain the cards following the //FORTRAN control card up to the next control card. The filetype will be FORTRAN.
*	If no parameters are specified, a filename will be chosen by the system and the default options will be taken.
MAP	Includes tables of addresses of FORTRAN variables, in the LISTING file.
NOMAP	Suppresses the tables of variables.
LIST	Includes a listing of object codes 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 Decimal Interchange Code.

Usage:

All cards following the / /FORTRAN control card and up to the next control card will be assumed to be FORTRAN source code. These cards will be read and stored as a file and will be compiled using the FORTRAN-G compiler under the specified or defaulted options. The source cards may contain any number of routines each delimited by an END statement. If a filename is specified, the source cards will be stored as a file with this filename and filetype of FORTRAN. If filename is not specified, the system will choose a unique name. The options specified govern the compilation of the following source cards. Any combination of options may be specified in any order. If an option is not specified, the underlined default choice is assumed.

A compilation will result in a LISTING and a TEXT file being generated. The LISTING file will be automatically written out on sysout. The TEXT file, i.e., object modules, can be punched using a / /PUNCH control card. The filename of the TEXT file will be the same as the filename of the FORTRAN source file.

Object modules, i.e., TEXT files, will be loaded into core and executed when the / / GO control card is specified.

Output:

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 variables, 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 assembly language.

Diagnostic and error messages produced by the compiler are placed in the LISTING file and will be printed on sysout.

Notes:

a. Source cards can be punched in either Binary Coded Decimal (BCD) code, or Extended Binary Coded Decimal Interchange Code (EBCDIC). If the BCD code is used, the BCD option must be specified for the compilation.

b. The entry point for the first main program will be the same as the filename. Subsequent main programs will 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 1-8 may be used. The FORTRAN logical files are defined as follows:

Logical Files	Record Length
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

e. An ID consisting of four characters is punched in columns 73-76 of object decks (TEXT files) and is followed by a sequence number in column 77-80. For a subroutine, the ID is formed from the first four letters of the subroutine name. For a main program, the ID is formed from the first four letters of the filename if it is the first deck in the file, otherwise, an ID of 'MAIN' will be used.

References:

For information on the FORTRAN IV language, see IBM System/360: FORTRAN IV LANGUAGE, Form C28-6515, and FORTRAN IV Library Subprograms, Form C28-6596. For information on compiler operation and messages, see FORTRAN IV (G) Programmer's Guide, Form C28-6639. Information in the Programmer's Guide on Operating System job control language and data management is not applicable under CMS. The LOAD, NAME=, and LINECNT=options are not supported.

7.2.2 / / ASSEMBLE

Format:

/ / ASSEMBLE [filename / *] [LIST / NOLIST] [XREF / NOXREF] [RENT / NORENT]

filename — specifies the name of the file which will contain the cards following the / / ASSEMBLE control card up to the next control card. The filetype will be SYSIN.

* — If no parameters are specified, a filename will be chosen by the system and the default options will be taken.

[LIST / NOLIST] — NOLIST suppresses creation of the LISTING file.

[XREF / NOXREF] — NOXREF suppresses creation of the cross-reference symbol table within the LISTING file. This option is ignored if NOLIST is specified.

[RENT / NORENT] — RENT causes the assembler to generate messages in the LISTING file if non-reentrant coding is found in the assembled program.

Usage:

All cards following the / /ASSEMBLE control card and up to the next control card will be assumed to be Assembly Language source code. These cards will be read and stored as a file and will be translated using the F-level Assembler and the specified or defaulted options. The source cards may contain any number of routines, each delimited by an END statement. If a filename is specified, the source cards will be stored as a file with this filename and filetype of SYSIN. If filename is not specified, the system will choose a unique name. The options specified govern the translation of the following source cards. Any combination of options may be specified in any order. If an option is not specified, the underlined default choice is assumed.

An assembly will result in a LISTING and a TEXT file being generated. The LISTING file will be automatically written out on sysout. The TEXT file, i.e., object module, can be punched using a / / PUNCH control card. The filename of the TEXT file will be the same as the filename of the Assembly Language source file.

Object modules, i.e., TEXT files, will be loaded into core and executed when the / /GO control card is specified.

Output:

If LIST is specified, or defaulted, a listing of the assembled machine-language code together with the source statements and an external symbol directory is generated.

If XREF is specified, or defaulted, a cross-reference symbol table will be included with the generated listing.

If RENT is specified, messages will be included in the listing if non-reentrant coding is found in the assembled program.

Diagnostic and error messages will appear at the end of the listing.

Notes:

a. The assembler searches the system macro library (SYSLIB MACLIB) for macro definitions. Names of the macros included in this library are shown in Figure 7.2.1-A. Additional macros may be included with the source program.

References:

For information on the Assembly Language, see IBM OS/360 Assembler Language, Form C28-6514 and Assembler E Programmer's Guide, Form C28-6595. For information on the basic 360 machine, see IBM System/360 Principles of Operation, Form A22-6821.

CMSTYPE
CMSAVE
CMSREG
MADCALL
MADDPL
MADTYPE
CMSYSREF
FINIS
RDBUF
GEN
TYPE
FCB
STATE
CKEOF
RDTYP
WRBUF
SETUP
ERASE
TYPIN
WAITR
RDJFCB
CLOSE
DCB
DCBD
FREEMAIN
GET
FREEPOOL
GETMAIN
GETPOOL
LINK

OPEN
PUT
READ
SAVE
SPIE
TIME
WAIT
WRITE
WTO
WTOR
IHBRDWRD
IHBRDWRT
IHB01
IHBINNRA
IHBERMAC
IHBRDWRK
IHBRDWRS
IHBOPLST
IHBINNRB
NOTE
XCTL
CHECK
ABEND
FIND
POINT
PUTX
RETURN
FDIMEN
BSP

Figure 7.2.1-A. Assembler Languages Macros

8.2.3. / / TEXT

Format:

/ / TEXT

Usage:

The cards following the / / TEXT control card and up to the next control card will be read and stored as a file. A unique filename will be chosen by the system and the filetype will be TEXT.

Example:

/ / TEXT

7.2.4 / / DATASET

Format:

/ / DATASET filename filetype

filename filetype	specifies the identifier to be given to the file which will contain the cards following the / / DATASET control card up to the next control card.

Usage:

The cards following the / /DATASET control card and up to the next control card will be read and stored as a file. If the file identifies FILE FT0nF001 when n is 1, 2, 3, 4, 7, or 8, the file can be read as a FORTRAN logical unit.

Example:

/ / DATASET FILE FT01F001

7.2.5 / / GO

Format:

/ / GO [entry point / default entry point]

entry point — specifies the name of a control section or entry point to which control will be passed at execution time.

default entry point — the default entry point is the beginning of the first program encountered in the job stream.

Usage:

This control card causes all TEXT files to be loaded into core together with the required modules from the system library SYSLIB TXTLIB. The proper linkages are established between the program modules. Programs are loaded at X'12000' and many extend up to X'3D000'.

A load map will be created containing the location of control sections and entry points of the programs loaded and will be printed on sysout. After loading is completed, execution will be begun by transferring control to either the entry point specified or to the default entry point.

Data cards following the / / GO control card can be read from FORTRAN programs using logical unit 5. Data cards not read will be ignored. Output written onto FORTRAN logical unit 6 will be written onto sysout and will be printed.

Note:

An "entry point" must be either a control section name or an entry point name. It may not be a filename if the filename is not identical to either a control section name or an entry point name.

Examples:

a. / / GO

b. / / GO ENTRY1

Error Messages:

E(00001) DEFINEDMORE 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 that is built during loading.

E(00004) THE FOLLOWING NAMES ARE UNDEFINED - xxxxxxxx
The names xxxxxxxx are referred to in a file and have never been defined.

E(00005) NAME IS UNDEFINED - xxxxxxxx
The name xxxxxxxx specified as an entry point does not exist.

7.2.6 // PUNCH

Format:

```
/ / PUNCH  [ filename  [ filetype ]  ]
           [ *         [ TEXT     ]  ]
```

filename filetype — the identifier of the file to be punched.

* — indicates that all TEXT files are to be punched.

Usage:

Files with records up to 80 characters in length will be punched into an 80 column card.

Examples:

a. / / PUNCH
All TEXT files will be punched.

b. / / PUNCH PROG1
The file PROG1 TEXT will be punched.

c. / / PUNCH FILE FT01F001
The FORTRAN logical file 1 will be punched.

Error Message:

E(00002) FILE NOT FOUND.

7.2.7 / / 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 LISTING file is not typed and is used as a printer carriage control character.

The carriage control characters are interpreted as follows:

Character	EBCDIC	Action
" "	40	print line and space 1
"0"	F0	space 1, print line, and space 1
"1"	F1	skip to new page, print line, and space 1
	xx	any other character is used as a CCW command code.

Example:

/ / PRINT FILE FT01F001

Error Message:

E(00003) FILE NOT FOUND.

7.2.8 // LINK

Format: // LINK userid devadd password

userid is the owner of the device to be LINKed to

devadd is the virtual device address of the disk

password is the read-share password of the disk

Usage: This control card enables the Batch Monitor to access files on a disk which can be LINKed to with read-share privileges. The types of files which can be handles in this manner are FORTRAN, SYSIN, and TEXT files, and Fortran logical files (FILE FTxxFyyy). Instead of having to send large files to the Batch machine, the user can send a small deck of control cards. The file which contains the control cards must have filetype, RJE.

Notes:

a. The owner of the library disk to be LINKed to must not have the disk in write status. If the Batch machine is unable to LINK to the disk in question, the user's job is deleted and must be re-submitted.

b. If the // LINK control card is used, Batch will attempt to access all FORTRAN, SYSIN, TEXT, and Fortran logical files from the library disk.

c. If the library disk has no password, substitute the word, ALL, in place of password.

Examples:

1.
```
// JOB
// NAME TRYVTS99
// LINK TRYVTS99 291 ALL
// FORTRAN GENREP
// TEXT SUBRT1
// TEXT SUBRT2
// GO
/*
```

Batch LINKs to TRYVTS99's 291 with a password of "ALL", i.e., NOPASS. Batch attempts to access GENREP FORTRAN, SUBRT1 TEXT, and SUBRT2 TEXT. It compiles GENREP, loads the three TEXT files, and executes.

2.
```
// JOB
// NAME TRYVTS88
// LINK TRYVTS88 191 TY8
// FORTRAN ILDEC
// DATASET FILE FT02F001
// GO
// PRINT FILE FT04F001
/*
```

ILDEC FORTRAN and FILE FT02F001 are obtained from the P-disk (191) of TRYVTS88. ILDEC is compiled and executed, and its output file, FILE FT04F001 is printed on the high-speed printer.

Error Messages: None.

7.2.9 // RETURN

Format: // RETURN filename filetype

filename filetype — identifier of the file to be returned to the user

Usage: Output as the result of a Batch job can be returned to the user at the termination of his job. A // RETURN card must be included in the job stream for each file the user wishes to have sent to his machine. The returned decks are in DISK DUMP format.

Notes:

a. The returned files are XFERed to the USERID which appears on the // NAME card.

Examples:

1.
```
// JOB
// NAME TRYVTS99
// FORTRAN GRAPH
   DIMENSION  X(100), Y(100)
      .
      .
      .
      .
      .
      .
   END
// GO
// RETURN  GRAPH  LISTING
// RETURN  FILE  FT03F001
/*
```

The records between the FORTRAN and GO cards are compiled as file, GRAPH. The resulting text is executed; GRAPH LISTING and FILE FT03F001 are DISK DUMPed to TRYVTS99.

2.
```
// JOB
// NAME  VTSUSER
// LINK  VTSUSER  191  PSWD
// ASSEMBLE  SORTINE
// RETURN  SORTINE  TEXT
/*
```

VTSUSER's P-disk is used as a library disk. SORTINE is assembled, and the text is returned to VTSUSER.

7.3.0 Running CMS Batch

CMS Batch accepts an input stream from either the card reader or tape. The punched output can go to tape or directly to cards and the printer output can go to tape or the printer. The tape assignments are as follows:

symbolic address	virtual address	contents	track
TAP5	184	SYSIN	9
TAP6	185	SYSOUT	9
TAP7	186	PUNCH	9

If the SYSIN tape is not attached to the BATCH machine, the system will assume that the input stream will come directly from the card reader. If the PUNCH tape is not attached, the system will write the PUNCHed output directly to the online punch. If the SYSOUT tape is not attached, the printer will be used.

The tapes used are unlabeled ones. The SYSIN tape consists of unblocked card images. The PUNCH tape also consists of unblocked card images but each job consists of a separate file, where the first record of each file contains the job number and the programmer's name. The SYSOUT print tape consists of unblocked 133 character records.

Each job is limited to 4 minutes of execution time and 5000 lines of printer output.

To run the CMS BATCH stream,

a) LOGIN to CP as the BATCH user
b) IPL the Batch Monitor's Nucleus

and the batch stream is started automatically. To initiate another job stream, IPL the Batch Monitor's nucleus each time.

Messages will be printed on the BATCH user's console indicating:

a) the job number being run
b) a request to mount a new SYSOUT or PUNCH tape
c) the userid of the jobs which have PUNCH output

During the running of the Batch machine, the operator may cancel a job or request that the Batch monitor pause after the end of the current job so that the operator may modify a system parameter. To interrupt the running of the Batch monitor, the Attention Key should be pressed to enter CP. Storage location X'4C' can then be modified to indicate the action desired. An external interrupt to the Batch monitor will then cause location X'4C' to be examined and appropriate action taken. The byte at location X'4C' has the following meaning:

X'00' cancel current job without a dump (default value)
X'01' cancel current job with a dump
X'02' pause at end of current job.

For example, to set code 1 in location X'4C' and return to the Batch monitor, press the Attention key and type:

```
ST  L4C  1
E
B
```

When a job is cancelled by the operator, a message is written on SYSOUT to indicate to the programmer what action was taken. The following messages may be printed on the programmer's output.

TIME LIMIT EXCEEDED

The running time of the job exceeded 240 seconds (4 minutes).

OUTPUT LIMIT EXCEEDED

The number of lines of output is limited to 5000.

CANCELLED BY OPERATOR.

The operator can cancel a job either with or without a core dump.

GLOSSARY OF TERMS

Terms with specific meanings in CP and CMS are described below in alphabetical order. For the definition of any term not appearing in this Glossary, refer to the first page number given for that term in the index.

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 (up to a maximum of eight).

ARGUMENT: any alphanumeric 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 will effect a transfer of control between the Control Program and other environments. The terminal keyboard will be unlocked, regardless of current processing, and the input line will be processed by the environment which has control.

ATTENTION KEY: a key on the terminal keyboard 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, will delete the preceding n characters and itself from the input line. Currently defined as the @ character.

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 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 will begin if no starting location is specified; either that given in the first non blank 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", where xxxxx is the error code returned in general purpose register 15 and xx.xx is the 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 the permanent, temporary, and system disks. 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, will delete all preceding characters in the input line and itself. Currently defined as the ¢ character.

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 to be 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 double words 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 will always contain 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 will be retained until the user requests that they be deleted.

READY MESSAGE: the message "R; T=xx.xx" which is typed as a response indicating the successful completion of a CMS command and a return to the CMS command environment. xx.xx in the above message is the 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 non-error 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 (1) the CMS nucleus, of which each user receives his own copy, and (2) 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 will be 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 (1) TEXT files containing relocatable object code and (2) a dictionary indicating the location and size of each of these TEXT files within the library.

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 1 to 8 characters which uniquely identifies a user to the Control Program.

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.

APPENDIX A

Summary of Commands, Requests, and Console Functions

CMS Commands

The CMS commands are briefly described in alphabetical order below. Each of these commands constitutes the only valid input to the CMS command environment.

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.
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.
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.
DEFINE	creates an entry for a specified file in the user's copy of the system file directory, enabling that file to be referenced as a CMS command.
DISK	causes a CMS disk file to be punched out or read in from cards which are in CMS card format.

DUMPREST	dumps the contents of an entire disk area to magnetic tape or restores the contents of an entire disk area 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 card-image files on disk and to make changes to existing files from his terminal.
ERASE	deletes the entry for a specified file (or files) from the appropriate directory, rendering the file inaccessable 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.
FINIS	closes the specified file (or files) by writing the last record of that file on disk, updating the user's 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 non-relocatable core-image file on the user's permanent disk which is a copy of the contents of core between two given locations.
GLOBAL	specifies (1) macro definition libraries to be searched during the assembly process or (2) text libraries to be searched when loading files containing relocatable object code.
IPL	performs an initial program load sequence on the version of the CMS nucleus which has been saved by CP.

KE	truncates information currently being typed at the terminal to 72 characters per line. This truncation will remain in effect for the duration of the currently executing command or user program.
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 pripter.
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's permanent file directory; and logs out from CMS, transferring control to the CP environment.
LISTF	either types out at the terminal the identifier and size 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, loads them into core, and establishes the proper linkages.
LOADMOD	reads a MODULE file -- which is in non-relocatable 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. If LOGIN is not issued, the files will be saved.
LOGOUT	compacts the user's permanent file directory; executes any CMS command specified as an operand; and logs out of CMS, transferring control to the CP environment.
MACLIB	generates or adds to a specified macro library, or types out the contents of the dictionary of that library.
MAPPRT	creates, and optionally prints, a file containing a map of entry points in the CMS nucleus.
OFFLINE	creates a disk file from card input, prints a disk file on the offline printer, or punches a disk file on cards.
PLI	compiles program written in PL/I source language 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.

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.
SCRIPT	either (1) allows the user to create arbitrary alphanumeric text files on disk and to make changes to existing files of this type from the terminal or (2) types out the contents of the specified file, formatting it as indicated by control words contained in the text.
SETERR	sets error overrides which will 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 will 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 SPL1 interpreter language, and executes SPL1 programs.
SPLIT	copies the specified portion of a card-image file and 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; lists all user-defined commands; or compacts the user's permanent file directory, as specified.
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.
TAPRINT	prints tape files created by the ASSEMBLE or WRTTAP commands on the offline printer.
TXTLIB	either (1) generates or adds to a specified text library, (2) types out the contents of the dictionary for that library, or (3) creates a file containing a list of entry points and control section names contained in that library.

UNDEFINE	clears the specified entry (created by a previous DEFINE command) from the user's copy of the system file directory.
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.
WRTTAP	copies fixed-length files from any 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.

591

DEBUG Requests

BREAK	specifies the 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 either at the terminal or on the offline printer.
GO	begins execution at either 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; updates the user's permanent file directory; and logs out from CMS, transferring control to the CP environment.
ORIGIN	establishes a "base" address which will be 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	re-initializes 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, word, channel address word, or program status word by replacing it with specified information.
STORE	changes the contents of the specified core location by replacing it with specified information.
X	types at the terminal the contents of a specified or assumed number of bytes of core starting at the specified location.

EDIT Requests

The EDIT requests are described below in alphabetical order. These requests constitute the only valid input to the Edit environment, although some of the requests affect the format of input to the Input environment.

BACKSPACE	defines a logical backspace character for use in both the Edit and Input environments. The default character is %.
BACKUP	repositions the pointer the specified number of lines above the current line.
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 automatically 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 non-blank characters 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	see BACKUP.
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.

SCRIPT EDIT Requests

The SCRIPT EDIT requests are described below in alphabetical order. The requests constitute the only valid input to the Script Edit environment:

BOTTOM	prints the last line of the file being edited, positions the pointer below this line, and enters the Script Input environment.
CHANGE	replaces the specified string of information in the current line 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 Script Edit environment to the CMS Command environment.
INPUT	transfers the user from the Script Edit environment to the Script Input environment.
INSERT	causes a single line of input to be added to the file without transferring to the Script Input environment.
KEEP	creates a file on disk which consists of all material (1) from the beginning of the current SCRIPT file or (2) from the point at which the last KEEP was issued.
LOCATE	searches the SCRIPT file for the first occurrence of the specified string.
NEXT	moves the pointer forward in the file for the number of lines 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 Script Edit environment to the CMS Command environment without saving the edited file.
REPLACE	replaces the contents of the line at which the pointer is currently positioned with the line of specified information.

SEEK	scans each line of the file, starting at the one at which the pointer is currently positioned, for a column-dependent match with the specified information.
TOP	positions the pointer to a null line which precedes the first line of information at the beginning of the file.

SCRIPT PRINT Control Words

Page Format:

.PL n	specifies the number of lines to be typed on a page (The default value is 66).
.BM n	specifies the number of lines to be allowed for the bottom margin of each page (The default value is 3).
.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).

Spacing:

.DS n	double spaces the information being typed out.
.SS n	single spaces the information being typed out.
.SP n	inserts the specified number of carriage returns prior to typing the next line.

Paragraph Format:

.LL n	specifies the line length in characters (The default value is 60).
.IN n	allows the left side of the printout to be indented.
.OF n	allows printout to be offset by indentinf all but the first line of a section.

Page Control:

.PA n starts the next line on a new page.

.PN controls both external and internal page numbering of the file being printed.

.HE heading
causes the next line to be used as a header line, and to be typed at the top of each page.

Format Mode:

.BR causes a break, meaning that information appearing before and after the .BR request will be typed on separate lines.

.FI "fills" the lines being typed out by (1) lengthening short lines by moving words from the following line (2) shortening long lines by moving words to the following line and (3) adjusting lines to the right margin by inserting additional blanks between words in the line.

.NF types all lines exactly as they appear in the file, truncating lines longer than the currently-set line length (.FI is assumed if neither .NR nor .FI is specified).

.CE centers the next line between the left and right margins.

Special Features:

.RD n issues the specified number of reads at the terminal to allow user input to be inserted in the printer.

.CP n causes a page eject to occur if less than the specified number of lines remain on the current page.

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 file into the printout of another SCRIPT file.

Control Program Console Functions

The Control Program console functions are described below in alphabetical order. These console functions constitute the only valid input to the Control Program environment

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.

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. An EXTERNAL console function followed by a BEGIN will cause the Debug environment to be entered.

IPL Simulates the Initial Program Load sequence on the specified unit.

LOGOUT releases the user's virtual machine, including his temporary disk area, and closes any spooling areas which have not been released.

MSG types out the specified message at the terminal of the person whose userid is specified.

QUERY types out either the number of users logged onto the system, the names of these users, or the maximum number of users allowed to log on, or the log message set by the operator.

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 messages typed at the terminal.
STORE	replaces the contents of the specified register(s), core location(s), or program status word with the specified information.
TIMER	controls whether real time or CPU time is kept for the virtual machine.

CMS Functions

CARDPH	Punches a card from the specified area.
CARDRD	Reads one card into the specified area.
CONWAIT	Waits for console I/O to "finish" all stacked reads and writes.
DESBUF	Releases buffers used by console routines.
ERASE	Performs the function similar to theERASE Command.
FINIS	Closes one or more specified files.
POINT	Sets a pointer for subsequent file references.
PRINTR	Prints one line on the printer or issues a carriage restore.
RDBUF	Reads information from disk.
SCAN	Scans a line of input and places strings of non-blanks into a command buffer.
STATE	Provides information about disk files.
STDERR	Obtains information about the last error.
TAPEIO	Reads/Writes and performs various tape I/O functions.
TYPE	Types one line on the console with a carriage return.
TYPLIN	Types one line on the console and gives one carriage return.
WAIT	Wait state is entered to await an interruption from one of the devices specified as arguments.
WAITRD	Waits until one line is read from the console.
WRBUF	Writes information onto disk.

APPENDIX B

Format of Commands, Requests, and Console Functions

Contents

APPENDIX B

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 upper or lower case
lowercase	-	lower case information designates the contents of a field, and does not in itself constitute meaningful input. (All lowercase symbols are defined on page 3.)
()	-	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, 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

{ } - braces enclose stacked items, only one of which may be specified. With stacked items not enclosed in braces, the underlined item may not be specified.

... - an ellipsis indicated 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

) - this symbol indicates a continued line

<u> </u> - underlining indicates the value which will be assumed if none is specified. When no underlined item appears in bracketed [ ] information, the default value is none. Underlined items not enclosed within braces { } may not be specified by the user.

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
defent	-	default entry point
defset	-	default option settings
devadd	-	device address
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 full word or less of hexadecimal information
hexloc	-	hexadecimal core location
id	-	an alphanumeric 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
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 (see pages 14 and 15)
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

CMS Commands		Page Ref.
{ALTER / AL}	oldfn oldft {oldfm / *} {newfn / *} {newft / *} {newfm / *}	3.1.1-1
{ASSEMBLE / A}	fn1 ... fnN [(option1 ... optionN)]	3.4.1-1
{CLOSIO / CLO}	[READER] [PRINTER] [PUNCH]	3.1.2-1
{CLROVER / CL}		3.3.1-1
{COMBINE / C}	newfn newft newfm oldfn1 oldft1 oldfm1 ... oldfnN oldftN oldfmN	3.1.3-1
{DEBUG / DEB}		3.3.2-1
{DEFINE / DE}	fn	3.5.1-1
{DISK / DI}	{DUMP fn ft fm / LOAD}	3.1.4-1
{DUMPREST / D}		3.1.5-1
{ECHO / EC}	[{S / X / U}]	3.5.2-1
{EDIT / E}	[fn] ft	3.1.6-1
{ERASE / ER}	{fn / *} {ft / *} [{fm / *}]	3.1.7-1

		Page Ref.
{EXEC / EX}	fn [arg1 ... argN]	3.5.3-1
{FINIS / FI}	{fn / *} {ft / *} [{fm / *} / first]	3.1.8-1
{FORMAT / FORM}	{P / T} [{ALL / C} / norec]	3.5.4-1
{FORTRAN / F}	fn1 ... fnN [(option1 ... optionN)]	3.4.2-1
{GENMOD / G}	entry1 [entry2 / nextloc]	3.2.1-1
{GLOBAL / GL}	{ASSEMBLER MACLIB / LOADER TXTLIB} [libname1 ... libnameN]	3.2.2-1
{IPL / IP}		3.5.14-1
KE		3.5.5-1
KO		3.5.6-1
KT		3.5.7-1
KX		3.5.8-1
{LISTF / L}	[[{fn / *} {ft / *} [{fm / *}] [P and T]] [{(EXEC) / (E)} / typeout]]	3.1.9-1
{LOAD / LO}	fn1 ... fnN [(option1 ... optionN)] [libname1 ... libnameN / syslib]	3.2.3-1

		Page Ref.
LOADMOD / LOADM	fn	3.2.4-1
LOGIN / LOGI	{UFD / NO_UFD}	3.5.9-1
LOGOUT / LOG	[anycom]	3.5.10-1
MACLIB / M	{GEN libname fn1 ... fnN / ADD libname fn1 ... fnN / LIST libname}	3.1.10-1
MAPPRT / MAP	[{A / N / C} [{ON / OFF / NO}]]	3.5.13-1
OFFLINE / O	{PUNCH / PRINT / PRINTCC / PRINTPLI fn ft [fm] first; READ {fn ft [{fm / P1}]; *}}	3.1.11-1
PLI / PL	fn1 ... fnN [(option1 ... option N)]	3.4.3-1
PRINTF / P	fn ft [{n1 / *} [{n2 / *} [n3 length]]]	3.1.12-1
REUSE / R	fn1 ... fnN [(option 1 ... optionN) libname 1 ... libnameN]	3.2.5-1
SCRIPT / SC	{EDIT fn / PRINT fn} [(option 1 ... option N)]	3.1.13-1

		Page Ref.
SETERR / SETE		3.3.3-1
SETOVER / SE	[SAMELAST [option 1 ... optionN] / defset]	3.3.4-1
SNOBOL / SN	fn [(option 1 ... option N)]	3.4.4-1
SPLIT / SP	fn1 ft1 fn2 ft2 {id1 / n1} [{id2 / n2 / eof}]	3.1.14-1
START / STAR	[entry / defent] [arg 1 ... argN]	3.2.6-1
STAT / S	[{P / T / D / C} P, T, and D]	3.5.11-1
TAPE / T	{LOAD / REWIND / SKIP / WRITEOF; DUMP fn ft [{S / T / P}]}	3.1.15-1
TAPRINT / TAPR	[TAP1 / TAP2]	3.1.18-1
TXTLIB / TX	{{GENERATE / G} libname fn1 ... fnN; {ADD / A} libname fn1 ... fnN; {PRINT / P} libname; {LIST / L} libname}	3.1.16-1

Page Ref. 608

{UNDEFINE / UN} {fn / *}		3.5.12-1
{UPDATE / U} fn1 [[[{ft1 / SYSIN}] [{fn2 / fn1}] [{ft2 / UPDATE}]]] (P) .fn1		3.1.17-1
{USE / US}	fn1 ... fnN [(option1 ... optionN)] [libname1 ... libnameN]	3.2.7-1
{WRITTAP / W}	fn ft	3.1.19-1
$	fn [arg1 ... argN]	3.2.8-1

DEBUG Requests		Page Ref.
BREAK	n {symbol / hexloc}	3.3.2-4
CAW		3.3.2-10
CSW		3.3.2-11
DEF	symbol hexloc [{n / 4}]	3.3.2-12
DUMP	{ONLINE / id} [{symbol1 / hexloc1 / 0} {symbol2 / hexloc2 / * / 3}]	3.3.2-15
GO	[{symbol / hexloc / intloc}]	3.3.2-21
GPR	reg1 [reg2]	3.3.2-24
IPL		3.3.2-26
KX		3.3.2-27
ORIGIN	{symbol / hexloc}	3.3.2-28
PSW		3.3.2-30
RESTART		3.3.3-32
RETURN		3.3.2-33
SET	{CAW hexinfo / CSW hexinfo [hexinfo] / PSW hexinfo [hexinfo] / GPR reg hexinfo [hexinfo]}	3.3.2-34
STORE	{symbol / hexloc} hexinfo [hexinfo [hexinfo]]	3.3.2-38
X	{symbol [{n / length}]} {hexloc [{n / 4}]}	3.3.2-42

EDIT Requests

		Page Ref.
{BACKSPACE / BACK}	[{c / %}]	3.1.6-9
{BACKUP / UP / U}	[{n / 1}]	3.1.6-10
{BLANK / B}	line	3.1.6-11
{BOTTOM / BO}		3.1.6-12
{BRIEF / BR}		3.1.6-13
{CHANGE / C}	/string1/string2/ [{*n / 1} | [G / first]]	3.1.6-14
{DELETE / D}	[{n / /string/ / 1}]	3.1.6-16
FILE	[fn / editfn]	3.1.6-18
{FIND / F}	line	3.1.6-20
{INPUT / I}		3.1.6-22
{INSERT / I}	line	3.1.6-23
{LOCATE / L}	/string/	3.1.6-25
{NEXT / N}	[{n / 1}]	3.1.6-27

		Page Ref.
{OVERLAY / O}	line	3.1.6-28
{PRINT / P}	[{n / 1} [{LINENO / L / nolineno}]]	3.1.6-30
{QUIT / Q}		3.1.6-31
REPEAT	[{n / 1}]	3.1.6-32
{RETYPE / R}	line	3.1.6-33
SAVE	[fn / edit fn]	3.1.6-34
{SERIAL / SER}	{id / (NO)} [{n / 10}]	3.1.6-36
{TABDEF / TABD}	[{c / #}]	3.1.6-38
{TABSET / TABS}	[n1 ... nN / setmar]	3.1.6-39
{TOP / T}		3.1.6-41
{UP / U / BACKUP}	[{n / 1}]	3.1.6-42
{VERIFY / VER}	[{n / 72}]	3.1.6-43

612

7-19-68
APPENDIX B-12
SCRIPT EDIT REQUESTS

SCRIPT EDIT Requests

		Page Ref. 612
{BOTTOM / B}		3.1.13-8
{CHANGE / C}	/[string1] / [string2] /	3.1.13-9
{DELETE / D}	[{n / 1}]	3.1.13-11
{FILE / F}	[nfn / ofn]	3.1.13-12
{INPUT / I}		3.1.13-13
{INSERT / I}	line	3.1.13-14
{KEEP / K}	fn	3.1.13-15
{LOCATE / L}	/string/	3.1.13-16
{NEXT / N}	[{n / 1}]	3.1.13-17
{PRINT / P}	[{n / 1}]	3.1.13-18
{QUIT / Q}		3.1.13-19
{REPLACE / R}	[line]	3.1.13-20
{SEEK / S}	string	3.1.13-22
{TOP / T}		3.1.13-23

7-19-68
APPENDIX B-13
SCRIPT PRINT CONTROL WORDS 613

SCRIPT PRINT Control Words

		Page Ref.
.AP	fn	3.1.13-24
.BM	[n / 1]	3.1.13-25
.BR		3.1.13-26
.CE		3.1.13-27
.CP	n	3.1.13-28
.DS		3.1.13-29
.FI		3.1.13-30
.HE	line	3.1.13-31
.IM	fn	3.1.13-32
.IN	[n / setmar]	3.1.13-33
.LL	n	3.1.13-34
.NF		3.1.13-35
.OF	[n / setmar]	3.1.13-36
.PA	[n / seq]	3.1.13-37

.PL	n	3.1.13-38
.PN	{ON / OFF / OFFNO}	3.1.13-39
.RD	[n / 1]	3.1.13-40
.SP	[n / 1]	3.1.13-41
.SS		3.1.13-42
.TM	n	3.1.13-43

Control Program Console Functions		Page Ref.
{BEGIN / B}	[hexloc / intloc]	4.1.1-1
{CLOSE / C}	devadd	4.1.2-1
{DETACH / DE}	devadd	4.1.3-1
{DISPLAY / D}	[hexloc] [hexloc1-hexloc2] [Lhexloc] [Lhexloc1-hexloc2] [Greg] [Greg1-reg2] [Yreg] [Yreg1-reg2] [PSW]	4.1.4-1
{DUMP / DU}	[hexloc] [hexloc1-hexloc2] [Lhexloc] [Lhexloc1-hexloc2] [Greg] [Greg1-reg2] [Yreg] [Yreg1-reg2] [PSW]	4.1.5-1
{EXTERNAL / E}		4.1.6-1
{IPL / I}	{CMS / devadd}	4.1.7-1
{LOGOUT / L}		4.1.8-1

		Page Ref.
{MSG / M}	{userid / ALL} line	4.1.9-1
{QUERY / Q}	{USER [NAMES] / LOGMSG / MAX}	4.1.10-1
READY	devadd	4.1.11-1
{RESET / R}		4.1.12-1
SET	{MSGOFF / MSGON}	4.1.14-1
{STORE / ST}	[Lhexloc hexinfo1 ... hexinfoN] [Greg hexinfo1 ... hexinfoN] [Yreg hexinfo1 ... hexinfoN] [PSW [hexinfo 1] hexinfo 2]	4.1.13-1
{TIMER / TI}	[{ON / OFF / CPU}]	4.1.15-1

CMS Command Options

	Defaults	Meaning of Option	Option
ASSEMBLE			
3.4.1	DECK	do not create a TEXT file	NODECK
	DIAG	do not type diagnostics at the terminal	NODIAG
	LIST	do not create a LISTING file	NOLIST
	NORENT	check for re-entrant coding	RENT
	XREF	do not create a cross-reference table in LISTING file	NOXREF
	LDISK	write the LISTING file on the permanent disk	
		write the LISTING file on the tape whose symbolic address in TAPn	LTAPn
		write the LISTING file on the offline printer	LPRINT or PRINT
FORTRAN			
3.4.2	DECK	do not create a TEXT file	NODECK
	DIAG	do not type diagnostics at the terminal	NODIAG
	EBCDIC	Source program is in BCD code	BCD
	NOGO	force compiler processing to completion	GO
	NOLIST	include assembler mnemonics in LISTING file	LIST
	NOMAP	include storage map in LISTING file	MAP
	NOPRINT	print LISTING file on the offline printer and erase it from the permanent disk	PRINT
	SOURCE	do not include source code in LISTING file	NOSOURCE
LOAD			
3.2.3	MAP	do not create a LOAD MAP file	NOMAP
	NOCLEAR	clear load region to zeroes before loading	CLEAR
	NOTYPE	type load map contents at the terminal	TYPE
	NOXEQ	execute the loaded file(s)	XEQ
	PINV	do not include invalid cards in the load map	SINV
	PREP	do not include Replace cards in the load map	SREP
	SLC12000	start loading program at hexadecimal location xxxxx	SLCxxxxx
	SYSLIB	search only the specified TXTLIB files	LIBE

	Defaults	Meaning of Option	Option
PLI			
3.4.3	C	suppress compilation after compile-time processing has been completed	NC
	C60	48-character set available on offline printer	C48
	D.	suppress creation of TEXT file	ND
	E	suppress External Symbol Dictionary in LISTING file	NE
	EB	source program is punched in BCD	B
	FW	suppress warnings in LISTING file	FE
	FW	suppress warnings and errors in LISTING file-include only severe errors	FS
	NA	include Attribute Listing in LISTING file	A
	NL	include assembler mnemonics in LISTING file	L
	NLD	create a SYSLIN file	LD
	NM	compile-time processing is required	M
	NX	include a Cross Reference Listing in LISTING file	X
	S	suppress listing of source program	NS
	S2	suppress listing of input to compile-time processor	NS2
	ST	suppress statement numbers in execution-time diagnostics	NST
	suppress punching of TEXT file	punch TEXT file on offline punch	{CMSDECK / DK}
	{CMSPRINT / P}	suppress printing of LISTING file on offline printer	{CMSNOPRT / NP}
	suppress writing of LISTING file on tape	write LISTING file on tape 180	{CMSTAPE1 / T1}
		write LISTING file on tape 181	{CMSTAPE2 / T2}

	Defaults	Meaning of Option	Option
SCRIPT PRINT			
3.1.13	do not pause	pause at end of each page	STOP
	wait	start printing immediately	NOWAIT
	do not translate	translate lower case to upper case	TRANSLATE
	start at page 1	start printout at page xxx	PAGExxx
	type output at terminal	print output on offline printer	OFFLINE
	start at print position 1	center output on offline printer	CENTER
SNOBOL			
3.4.4	write LISTING on permanent disk	print LISTING file on offline printer	OFFLINE or PRINT
	write LISTING on permanent disk	type LISTING file at terminal	ONLINE
	create LISTING file	do not create a LISTING file	NOLIST
	compile SNOBOL file	execute SPL1 file	SPL1

	Defaults	Meaning of Option	Option
SETOVER			
3.3.4			
	record no GPR information	record general purpose register (GPR) contents before branch and after return	GPRS
		record GPR contents after return only	GPRSA
		record GPR contents before branch only	GPRSB
	record no FPR information	record floating point register (FPR) contents before branch and after return	FPRS
		record FPR contents after return only	FPRSA
		record FPR contents before branch only	FPRSB
	record 2 lines of parameter list	record no parameter list information	NOPARM
		record 1 line of parameter list	PARM1
	record basic line for WAIT	record no information for WAIT	NOWAIT
		record same information for WAIT as for other routines	WAITSAME
		record 1 line of parameter list for WAIT	WAIT1
		record 2 lines of parameter list for WAIT	WAIT2
	use current settings	reset options to default values	DEFAULT

APPENDIX C

CP/CMS MESSAGES

MESSAGE	SOURCE	ERR. CODE	PAGE REF.
A FILE HAS THE WRONG RECORD LENGTH.	TXTLIB		3.1.16-3
A FILE IS MISSING	TXTLIB		3.1.16-3
ALREADY DEFINED	DEFINE	03	3.5.1-2
AN INPUT FILE DOES NOT EXIST.	MACLIB	05	3.1.10-4
AT LEAST ONE OF THE FILES TO ASSEMBLE DOESN'T EXIST OR DOESN'T HAVE A CORRECT TYPE NAME.	ASSEMBLE	01	3.4.1-7
AT LEAST ONE OF THE FILES TO ASSEMBLE HAS INCORRECT RECORD LENGTH.	ASSEMBLE	01	3.4.1-7
AT LEAST ONE OF THE FILES TO BE COMPILED DOESN'T EXIST OR DOESN'T HAVE A 'FORTRAN' TYPE NAME.	FORTRAN	01	3.4.2-7
AT LEAST ONE OF THE FILES TO BE COMPILED HAS LOGICAL RECORD LENGTH DIFFERENT OF 80 BYTES	FORTRAN	01	3.4.2-7
ATTEMPT TO COMBINE FIXED AND VARIABLE LENGTH FILES.	COMBINE	13	3.1.3-2
ATTEMPT TO PRINT RECORD GREATER THAN 132 COLUMNS.	OFFLINE	06	3.1.11-6
ATTEMPT TO PUNCH RECORD GREATER THAN 80 COLUMNS	OFFLINE	03	3.1.11-6
ATTEMPT TO WRITE OUTPUT FILE ON SYSTEM DISK ILLEGAL.	COMBINE	12	3.1.3-2
BAD ARGUMENT	CP		4.1.0-1
	CP BEGIN		4.1.1-1
	CP CLOSE		4.1.2-2
	CP DETACH		4.1.3-1
	CP DISPLAY		4.1.4-2
	CP DUMP		4.1.5-2
	CP IPL		4.1.7-1
	CP MSG		4.1.9-1
	CP QUERY		4.1.10-1
	CP READY		4.1.11-1
	CP STORE		4.1.13-2
	CP TIMER		4.1.15-1
BAD MODE	SPLIT		3.1.14-2
BAD OUTPUT TYPE	SPLIT		3.1.14-2

BUFFER TOO SMALL	SPLIT		3.1.14-2
CALLEE=XXXXXXXX	CLROVER		3.3.1-3
CALLER=XXXXXXXX	CLROVER		3.3.1-3
CE AND DE NOT FOUND TOGETHER (VERY STRANGE)	FORMAT	04	3.5.4-2
CE AND DE NOT TOGETHER CHECKING NO. CYLINDERS	FORMAT	06	3.5.4-2
CMS LOGGING OUT... TOTAL CPUTIME (IN SECONDS) = XX.XX CP ENTERED, READY.	CMS LOGOUT		3.5.10-2
CMS... VERSION I.O - 05/01/68	CMS		2.2.2-3
COMMAND-TO-BE-UNDEFINED NOT FOUND IN SSTAT	UNDEFINE	02	3.5.12-2
COMPILATION CANCELLED DUE TO SOURCE PROGRAM ERROR(S).	FORTRAN	32	3.4.2-7
CONDITION-CODE 1 ON SIO IN FORMATING DISK(BAD)	FORMAT	02	3.5.4-2
CONDITION-CODE 2 ON SIO IN FORMATING DISK(BAD)	FORMAT	02	3.5.4-2
CONDITION-CODE 3 ON SIO IN FORMATING DISK(BAD)	FORMAT	02	3.5.4-2
CORRECT FORM: ERASE FILENAME FILETYPE FILEMODE, NAME AND/OR TYPE MAY BE *, AND MODE MAY BE * OR BLANK.	ERASE	01	3.1.7-2
CORRECT FORM IS DISK DUMP FILENAME FILETYPE FILEMODE OR DISK LOAD .	DISK	01	3.1.4-3
CORRECT FORM IS: COMBINE N1 T1 M1 N2 T2 M2... WHERE N1,T1,M1 ARE THE NAME, TYPE, AND MODE OF THE FILE TO BE CREATED. AND N2 T2 M2, ETC. ARE THE FILES TO BE COMBINED.	COMBINE	10	3.1.3-2
CORRECT FORM IS: LISTF FILENAME FILETYPE MODE WHERE FILENAME, FILETYPE, AND MODE ARE OPTIONAL. FILENAME AND/OR FILETYPE MAY BE*. MODE MAY BE P,T,S,*, OR OMITTED. SEARCH IS (P,T) IF OMITTED, OR (P,T,S)IF *.	LISTF	01	3.1.9-3
CORRECT FORM IS:'OFFLINE' COMMAND FILENAME FILETYPE OPTIONAL-FILEMODE WHERE COMMAND ISO 'READ', 'PRINT', 'PRINTCC', OR 'PUNCH', IF FILENAME=* UNDER READ MODE, CONTROL CARDS ARE EXPECTED IN INPUT STREAM.	OFFLINE	01	3.1.11-6
CORRECT FORM IS: 'PRINTF' FILENAME FILETYPE STARTLINE ENDLINE LINE-LIMIT, WHERE 'STARTLINE', 'ENDLINE', AND 'LINE-LIMIT' ARE OPTIONAL	PRINTF	01	3.1.12-2

CORRECT FORM IS: 'SCRIPT' COMMAND NAME WHERE COMMAND IS 'EDIT' OR 'PRINT'.	SCRIPT	69	3.1.13-6
CORRECT FORM IS: SCRIPT EDIT FILENAME.	SCRIPT	01	3.1.13-5
CORRECT FORM: MACLIB COMMAND MACLIB-FILE (ASP360 FILES) WHERE COMMAND IS GEN , LIST , OR ADD .	MACLIB	01	3.1.10-3
CORRECT FORMS ARE: TAPE DUMP FILENAME FILETYPE FILEMODE TAPE LOAD TAPE REWIND TAPE WRITEOF TAPE SKIP	TAPE	01	3.1.15-4
CURRENT USERS ARE...	CP QUERY		4.1.10-1
DEBUG ENTERED...	DEBUG		3.3.2-2
DEBUG ENTERED, EXTERNAL INT.	DEBUG CP BEGIN CP EXTERNAL		3.3.2-2 4.1.1-1 4.1.6-1
DEBUG ENTERED BREAKPOINT XX AT XXXXXX	DEBUG		3.3.2-2
DEBUG ENTERED PROGRAM INT. PSW=XXXXXXXXXXXXXXXX	DEBUG		3.3.2-2
DEFAULT TABS SET.	SCRIPT		3.1.13-5
	EDIT		3.1.6-7
DEFINED COMMAND(S) = 'COMMAND1...COMMANDN'	STAT		3.5.11-2
DEFINED MORE THAN ONCE - XXXXXXXX	LOAD REUSE USE $	01 01 01 03	3.2.3-6 3.2.5-2 3.2.7-2 3.2.8-2
DEV XXX DETACHED.	CP DETACH		4.1.3-1
DISK ERROR.	GENMOD LOADMOD PRINTF SCRIPT SPLIT $	02-13 01-10 02 02 03	3.2.1-2 3.2.4-1 3.1.12-2 3.1.13-5 3.1.14-2 3.2.8-3
DISK ERROR WHILE READING.	COMBINE SCRIPT TAPE	02 12 02	3.1.3-2 3.1.13-6 3.1.15-4
DISK ERROR WHILE WRITING.	COMBINE TAPE	03 03	3.1.3-2 3.1.15-4
DISK FULL	SPLIT		3.1.14-2
DUMP---XXXXXXXX	DEBUG DUMP		3.3.2-15

DUMP/RESTORE MOVED NNN CYLINDERS. THERE WERE NNN RECOVERABLE TAPE ERRORS.	DUMPREST		3.1.5-1
E(XXXXX) T=XX.XX	CMS		
E(XXXXX)	EXEC	XXXXX	3.5.3-4
EDIT:	EDIT SCRIPT		3.1.6-7 3.1.13-5
END CARD MISSING FROM DISK LOAD DECK.	DISK	06	3.1.4-3
END OF TAPE	TAPRINT		3.1.18-1
ENDING RECORD OF FILE MISSING.	TAPE	06	3.1.15-5
ENTER PASSWORD:	CP LOGIN		2.2.2-2
EOF REACHED.	SCRIPT DELETE SCRIPT LOCATE SCRIPT NEXT SCRIPT PRINT SCRIPT SEEK SPLIT		3.1.13-11 3.1.13-16 3.1.13-17 3.1.13-18 3.1.13-22 3.1.14-2
EOF READ ON TAPE	TAPRINT		3.1.18-1
EOF REACHED BY: XXX...XXX	EDIT BLANK EDIT CHANGE EDIT DELETE EDIT FIND EDIT LOCATE EDIT NEXT EDIT OVERLAY EDIT PRINT		3.1.6-11 3.1.6-15 3.1.6-16 3.1.6-20 3.1.6-25 3.1.6-27 3.1.6-28 3.1.6-30
ERRET=XXXXXXXX	CLROVER		3.3.1-4
ERROR DURING IPL SID.	CP IPL		4.1.7-1
ERROR IN NAME, TYPE, OR MODE OF OUTPUT FILE	COMBINE	05	3.1.3-2
ERROR ON DISK	SPLIT		3.1.14-2
*****ERROR-OVERRIDE	CLROVER		3.3.1-3
ERROR WHILE LOADING THE IEYFORT MODULE.	FORTRAN	16	3.4.2-7
ERROR WHILE PRINTING.	OFFLINE	07	3.1.11-6
ERROR WHILE PUNCHING.	OFFLINE	05	3.1.11-6
ERROR WHILE READING	MACLIB TXTLIB	03	3.1.10-3 3.1.16-3
ERROR WHILE READING DISK.	OFFLINE	04	3.1.11-6

ERROR WHILE WRITING	MACLIB TXTLIB	02	3.1.10-3 3.1.16-3
ERROR WHILE WRITING DISK.	OFFLINE	08	3.1.11-7
ERRORS ENCOUNTERED. SYSIN REMAINS UNCHANGED.	UPDATE		3.1.17-4
EXECUTION BEGINS...	LOAD START $		3.2.3-5 3.2.6-2 3.2.8-2
FATAL DISK ERROR	DISK TAPE	03 03	3.1.4-3 3.1.15-4
FATAL ERROR	WRTTAP	05	3.1.19-2
FATAL ERROR 1	UPDATE		3.1.17-3
FATAL ERROR 2	UPDATE		3.1.17-3
FATAL ERROR 3	UPDATE	02	3.1.17-5
FATAL PUNCH ERROR.	DISK	02	3.1.4-3
FATAL READER ERROR.	DISK	04	3.1.4-3
FATAL TAPE ERROR WHILE READING.	TAPE	04	3.1.15-4
FATAL TAPE ERROR WHILE WRITING.	TAPE	02	3.1.15-4
FILE DOES NOT EXIST.	EXEC GENMOD LOADMOD	01 XXXXX 01-04	3.5.3-3 3.2.1-2 3.2.4-1
FILE DOES NOT EXIST WILL BE CREATED.	EDIT		3.1.6-6
FILE EMPTY-EXIT TAKEN	EDIT FILE EDIT SAVE	02	3.1.6-18 3.1.6-34
FILE FILENAME FILETYPE FILEMODE NOT FOUND	COMBINE TAPE	01 01	3.1.3-2 3.1.15-4
FILE HAS WRONG RECORD SIZE.	EXEC	04	3.5.3-4
FILE MODIFIED	SPLIT		3.1.14-2
FILE NOT CHANGED	SPLIT		3.1.14-2
FILE NOT FOUND	LISTF OFFLINE PRINTF WRTTAP	02 02 03 04	3.1.9-3 3.1.11-6 3.1.12-2 3.1.19-2
FILE NOT FOUND - XXXXXXXX	LOAD REUSE USE	06 06 06	3.2.3-6 3.2.5-2 3.2.7-2

LIBRARY NOT FOUND.	TXTLIB		3.1.16-3
LOAD FAILED.	$	03	3.2.8-4
LOAD PAPER HIT RETURN	SCRIPT		3.1.13-5
LOGGED ON AT TERM. XXX LOGOFF AT XX.XX.XX ON XX/XX/XX	CP LOGIN		2.2.2-2
LOGIN OR FORMAT PLEASE	CMS		
LOGIN NO UFD FAILED TRY FORMAT P	CMS LOGIN	02	3.5.9-2
LOGIN UFD FAILED	CMS LOGIN	01	3.5.9-2
LOGOFF AT XX.XX.XX ON XX/XX/XX.	CP LOGOUT		4.1.8-1
MACLIB DICTIONARY OVERFLOW -- TOO MANY MACROS. SORRY.	MACLIB	08	3.1.10-4
MACLIB FILE SPECIFIED DOES NOT EXIST.	MACLIB	06	3.1.10-4
MACLIB FILE SPECIFIED NOT IN CORRECT FORM.	MACLIB	07	3.1.10-4
MAX. = OF USERS EXCEEDED	CP LOGIN		2.2.2-2
MODE SPECIFIED FOR OUTPUT FILE IS ILLEGAL.	COMBINE	11	3.1.3-2
MORE THAN EIGHT ACTIVE FILES-REDUCE NESTING.	SCRIPT	24	3.1.13-6
MORE THAN 256 LIBRARY ENTRIES. SORRY.	TXTLIB		3.1.16-3
NAME IS UNDEFINED - XXXXXXXX	REUSE START $	05 05 05	3.2.5-2 3.2.6-2 3.2.8-4
NEW FILE ALREADY EXISTS	ALTER SCRIPT KEEP	02	3.1.1-2 3.1.13-15
NEW MODE ILLEGAL	ALTER	06	3.1.1-2
NN USERS	CP QUERY		4.1.10-1
NO CHANGE	SCRIPT CHANGE		3.1.13-10
NO CHANGES MADE	ALTER	04	3.1.1-2
NO CURRENT LINE	SCRIPT PRINT		3.1.13-18
NO CURRENT LINE. INPUT:	SCRIPT BOTTOM		3.1.13-8
0 DEFINED COMMANDS	STAT		3.5.11-2
NO EXEC, MODULE, OR TEXT VERSION OF FILE XXXXXX FOUND. $ COMMAND CANNOT BE EXECUTED.	$	02	3.2.8-4

NO FILE TO BE COMPILED IS DEFINED.	FORTRAN	01	3.4.2-7
NO KNOWN T-DISK	STAT		3.5.11-2
NO MORE ROOM FOR USER-DEFINED COMMANDS	DEFINE	01	3.5.1-2
NO PRIMARY NAME SPECIFIED.	EDIT		3.1.6-6
NO PRIMARY NAME SPECIFIED-RETRY	EDIT FILE EDIT SAVE		3.1.6-18 3.1.6-34
NONEXISTENT UNIT	CP DETACH CP READY		4.1.3-1 4.1.11-1
NONSTANDARD FILE	SPLIT		3.1.14-2
NORMAL OVERRIDE	CLROVER		3.3.1-3
****NOTE--NORMAL- AND ERROR-OVERRIDES HAVE NOW BEEN CLEARED****	CLROVER		3.3.1-4
... NOW ASSEMBLING FILE FILENAME	ASSEMBLE		3.4.1-3
NRMRET= XXXXXXXX	CLROVER		3.3.1-3
OLD FILE NOT FOUND	ALTER SCRIPT KEEP	01	3.1.1-2 3.1.13-15
OLD MODE ILLEGAL	ALTER	03	3.1.1-2
OPEN FILE LIMIT	SPLIT		3.1.14-2
OPEN FOR READ	SPLIT		3.1.14-2
OPEN FOR WRITING	SPLIT		3.1.14-2
OUTPUT TAPE FULL. CHANGE IT AND HIT CARRIAGE RETURN.	ASSEMBLE		3.4.1-4
OVERLAY ERROR	LOAD REUSE USE $	02 02 02 03	3.2.3-6 3.2.5-2 3.2.7-2 3.2.8-2
P-DISK: NNNN RECORDS IN USE, NNN LEFT (OF NNNN), NN% FULL (OF NNN CYL.)	STAT		3.5.11-2
PARAMETER ERROR	UPDATE WRTTAP	02	3.1.17-3 3.1.19-2
PARM.-LIST=XXXXXXXX...XXXXXXXX	CLROVER		3.3.1-4
PASSWORD INCORRECT. LOGOFF AT XX.XX.XX ON XX/XX/XX	CP LOGIN		2.2.2-3

PERMANENT I/O ERROR ON DISK, ASSEMBLY CONTINUES WITHOUT WRITING LISTING FILE ON DISK.	ASSEMBLE		3.4.1-4
PERMANENT I/O ERROR ON TAPE	TAPRINT		3.1.18-1
PERMANENT I/O ERROR ON TAPE LISTING FILE WILL BE WRITTEN ON DISK WRITING ON TAPE IS CANCELLED.	ASSEMBLE		3.4.1-4
PERMANENT I/O ERROR ON THE PRINTER, LISTING FILE WILL BE WRITTEN ON DISK.	ASSEMBLE		3.4.1-4
PERMANENT I/O ERROR WHILE READING SYSIN FILE FILENAME UNABLE TO ASSEMBLE ALL THE FILE.	ASSEMBLE	20	3.4.1-7
PLEASE 'LOGIN UFD' OR 'LOGIN NO UFD'	CMS LOGIN	03	3.5.9-2
PLEASE READY THE PRINTER	ASSEMBLE TAPRINT		3.4.1-4 3.1.18-1
PLEASE SPECIFY DISK: PERMANENT (P) OR OR TEMPORARY (T).	FORMAT	01	3.5.4-2
R;T=XX.XX	CMS		
READ ERROR	UPDATE		3.1.17-3
READ ONLY	SPLIT		3.1.14-2
READY AT XX.XX.XX ON XX/XX/XX	CP LOGIN		2.2.2-3
READY THE PRINTER.	FORTRAN		3.4.2-4
READY THE TAPE UNIT AND HIT CARRIAGE RETURN	ASSEMBLE		3.4.1-4
REPEAT LOGIN	CP LOGIN		2.2.2-2
REFERENCE TABLE OVERFLOW	LOAD REUSE USE $	03 03 03 03	3.2.3-6 3.2.5-2 3.2.7-2 3.2.8-3
REPLACES OLD BREAKPOINT XX AT XXXXX	DEBUG BREAK		3.3.2-7
SEM --- VERSION 2/5/68	SCRIPT		3.1.13-4
SETTING ERROR-OVERRIDE TO PROVIDE A DYNAMIC TRACE OF CMS (AND CS) SVC-CALLS...	SETERR CLROVER SETOVER		3.3.3-3 3.3.1-3 3.3.4-4
SPECIFY 'DUMP' OR 'RESTORE':	DUMPREST		3.1.5-1
START TERMINAL TEST	ECHO		3.5.2-2
SVC-CLD-PSW=XXXXXXXXXXXXXXXX	CLROVER		3.3.1-3
SYMBOLIC TAPE ADDRESS INCORRECT.	TAPRINT	01	3.1.18-2

SYMBOLIC TAPE ADDRESS INCORRECT LISTING FILE WILL BE WRITTEN ON DISK. WRITING ON TAPE IS CANCELLED.	ASSEMBLE		3.4.1-3
16 SYMBOLS ALREADY DEFINED	DEBUG DEF		3.3.2-13
TAPE ERROR	WRTTAP	XXXXX	3.1.19-2
TAPE IS NOT IN TAPE LOAD FORMAT.	TAPE	05	3.1.15-5
THE FOLLOWING NAMES ARE UNDEFINED - XXXXXXXX	LOAD REUSE USE $	04 04 04	3.2.3-6 3.2.5-2 3.2.7-2 3.2.8-3
TOO MANY DUMMY ARGUMENTS.	EXEC	03	3.5.3-4
TOO MANY FILES	SPLIT		3.1.14-2
TOO MANY LEFT PARENTHESES.	FORTRAN	01	3.4.2-7
TRUNCATED	EDIT SCRIPT CHANGE		3.1.6-7 3.1.13-10
TYPE NOT FOUND	SPLIT		3.1.14-2
UNABLE TO COMPILE MORE THAN 32 FILES IN ONE RUN. PLEASE SPLIT YOUR REQUEST.	FORTRAN	01	3.4.2-7
UNABLE TO IPL SPECIFIED UNIT TYPE	CP IPL		4.1.7-1
USER NOT IN DIRECTORY. LOGOFF AT XX.XX.XX ON XX/XX/XX	CP LOGIN		2.2.2-2
UNEXPECTED UNIT-CHECK CHECKING NO. CYLINDERS	FORMAT	07	3.5.4-2
UNEXPECTED UNIT-CHECK IN FORMATTING DISK	FORMAT	03	3.5.4-2
USER NOT ON SYSTEM	CP MSG		4.1.9-1
VARIABLE LENGTH FILE	WRTTAP	03	3.1.19-2
WRITE ERROR	UPDATE		3.1.17-3
WRONG NUMBER OF PARAMETERS	SPLIT		3.1.14-2
X-DISK: NNN CYL.	FORMAT		3.5.4-1

CMS USER'S GUIDE

INDEX

CP-67/CMS USER'S GUIDE

Update Temporary R2

Previous Date	Section	Page Numbers	Action	New Date
7-19-68	Table of Contents	0.0.0-1 thru 5	R	5- 1-69
9-20-67	Terminal Usage	2.2.0-1	R	5- 1-69
9-21-67	Terminal Characteristics	2.2.1-8 thru 10	A	5- 1-69
9-27-67	Logging Procedures	2.2.2-1 thru 6	R	5- 1-69
	Dialing a Multi-Access System	2.2.3-1 thru 4	D/A	5- 1-69
9-27-67	General Typing Conventions	2.2.4-1 thru 2	A	5- 1-69
	Attention Interrupt	2.2.5-1	A	5- 1-69
9-27-67	Environment Conventions	2.3.0-1 thru 2, -4	E	5- 1-69
4- 1-68	CMS Commands	3.0.0-1 thru 2	R	5- 1-69
7-31-67	Alter	3.1.1-1 thru 2	R	5- 1-69
9-15-67	Closio	3.1.2-3	E	11- 1-68
4- 1-68	Edit	3.1.6-8A thru 8B, 44 thru 45	E	11- 1-68
9-18-67	Listf	3.1.9-1 thru 4	R	5- 1-69
9-12-67	Maclib	3.1.10-1 thru 6	R	5- 1-69
7-19-68	Offline	3.1.11-1 thru 7	R	5- 1-69
4- 1-68	Script	3.1.13-1 thru 69	R	5- 1-69
1-22-68	Split	3.1.14-1 thru 3	R	11- 1-68
7-19-68	Tape	3.1.15-1 thru 6	R	5- 1-69
9-19-67	Txtlib	3.1.16-1 thru 5	R	10-23-68
	State	3.1.20-1	A	5- 1-69
	Tapeio	3.1.21-1 thru 3	A	5- 1-69
	Dumpf	3.1.22	A	11- 1-68
	Dumpd	3.1.23	A	11- 1-68
8-16-67	Execution Control	3.2.0-1 thru 2	R	5- 1-69
4- 1-68	Genmod	3.2.1-1 thru 2	R	5- 1-69
9-13-67	Global	3.2.2-1 thru 4	R	5- 1-69
9-18-67	Load	3.2.3-1 thru 7	R	5- 1-69
4- 1-68	Loadmod	3.2.4-1 thru 3	R	5- 1-69
9-18-67	Reuse	3.2.5-1 thru 3	R	5- 1-69
9-18-67	Start	3.2.6-1 thru 2	R	5- 1-69
9-18-67	Use	3.2.7-1 thru 3	R	5- 1-69
9-21-67	Assemble	3.4.1-1 thru 9	R	5- 1-69
1-22-68	CMS Macros	3.4.1.2-1 thru 2	R	5- 1-69
	OS Macros	3.4.1.3-1 thru 2	A	5- 1-69
	CMS Nucleus Routine	3.4.1.4-1 thru 2	A	5- 1-69
	Attn	3.4.1.4.1-1	A	5- 1-69
	Cardph	3.4.1.4.2-1	A	5- 1-69
	Cardrd	3.4.1.4.3-1	A	5- 1-69
	Conwait	3.4.1.4.4-1	A	5- 1-69
	Cpfunctn	3.4.1.4.5-1	A	5- 1-69
	Desbuf	3.4.1.4.6-1	A	5- 1-69

Previous Date	Section	Page Numbers	Action	New Date
	Erase	3.4.1.4.7-1	A	5- 1-69
	Finis	3.4.1.4.8-1	A	5- 1-69
	Hndint	3.4.1.4.9-1 thru 2	A	5- 1-69
	Hndsvc	3.4.1.4.10-1 thru 2	A	5- 1-69
	Logdsk	3.4.1.4.11-1	A	5- 1-69
	Point	3.4.1.4.12-1	A	5- 1-69
	Printr	3.4.1.4.13-1	A	5- 1-69
	Rdbuf	3.4.1.4.14-1	A	5- 1-69
	State	3.4.1.4.15-1	A	5- 1-69
	Tapeio	3.4.1.4.16-1thru3	A	5- 1-69
	Trap	3.4.1.4.17-1	A	5- 1-69
	Type	3.4.1.4.18-1	A	5- 1-69
	Typlin	3.4.1.4.19-1	A	5- 1-69
	Wait	3.4.1.4.20-1	A	5- 1-69
	Waitrd	3.4.1.4.21-1thru2	A	5- 1-69
	Wrbuf	3.4.1.4.22-1	A	5- 1-69
	Fortran Programming	3.4.2.1-9 thru 11	E	11- 1-68
6- 1-68	PLI	3.4.3-1 thru 5	R	5- 1-69
	PL/I Programming	3.4.3.1-1 thru 12	A	5- 1-69
	Bruin	3.4.5-1 thru 2	A	5- 1-69
	Libend	3.5.1-1 thru 2	D/A	5- 1-69
7-19-68	Exec	3.5.3-1 thru 11	R	11- 1-68
1-22-68	Kx	3.5.8-1 thru 2	R	5- 1-69
4- 1-68	Logout	3.5.10-1 thru 2	R	5- 1-69
	Blip	3.5.12	D/A	5- 1-69
	Rt	3.5.15-1	A	11- 1-68
	Chardef	3.5.16-1 thru 2	A	5- 1-69
	Cpfunctn	3.5.17-1 thru 2	A	5- 1-69
	Utilities	3.6.0-1	A	5- 1-69
	CNVT26	3.6.1-1	A	5- 1-69
	Compare	3.6.2-1	A	5- 1-69
	CVTFV	3.6.3-1 thru 2	A	5- 1-69
	Modmap	3.6.4-1	A	5- 1-69
	Sort	3.6.5-1 thru 3	A	5- 1-69
	Tpcopy	3.6.6-1 thru 2	A	5- 1-69
	Text Libraries	3.7.0	A	11- 1-68
	Syslib Txtlib	3.7.1-1 thru 3	A	11- 1-68
	Tapset Routine	3.7.1.1-1 thru 3	A	11- 1-68
	Trap Routine	3.7.1.2	A	11- 1-68
	Blip Routine	3.7.1.3	A	11- 1-68
	Logdsk Routine	3.7.1.4	A	11- 1-68
	Reread Routine	3.7.1.5	A	11- 1-68
	Define Routine	3.7.1.6-1 thru 2	A	11- 1-68
	Dsdset Routine	3.7.1.7-1	A	11- 1-68
	Getpar Routine	3.7.1.8	A	11- 1-68
	Cpnmon/Cpnmof Routines	3.7.1.9-1 thru 2	A	11- 1-68
4- 3-68	Console Function Descriptions	4.1.0-1 thru 2	R	5- 1-69
7-19-68	Begin	4.1.1-1	R	5- 1-69

Previous Date	Section	Page Numbers	Action	New Date
4- 3-68	External	4.1.6-1	R	5- 1-69
7-19-68	Ipl	4.1.7-1 thru 2	R	5- 1-69
9-17-67	Logout	4.1.8-1 thru 2	R	5- 1-69
9-17-67	Msg	4.1.9-1 thru 2	R	5- 1-69
4- 3-68	Query	4.1.10-1 thru 3	R	5- 1-69
9-17-67	Ready	4.1.11-1	R	5- 1-69
4- 3-68	Set	4.1.14-1 thru 3	R	5- 1-69
	Xfer	4.1.15-1 thru 3	D/A	5- 1-69
	Purge	4.1.16-1	A	5- 1-69
	Sleep	4.1.17-1	A	5- 1-69
	Spool	4.1.18-1 thru 3	A	5- 1-69
	Disconn	4.1.19	A	5- 1-69
	Link	4.1.20	A	5- 1-69
7-19-68	Offline Procedures	5.2.0-1	R	5- 1-69
	Tape Procedures	5.5.0-1	A	5- 1-69
	CMS Batch Monitor	7.0.0-1 thru 3	A	11- 1-68
	// Fortran	7.2.1-2 thru 3	A	11- 1-68
	// Assemble	7.2.2-1 thru 3	A	11- 1-68
	// Text	7.2.3-1	A	11- 1-68
	// Dataset	7.2.4-1	A	11- 1-68
	// Go	7.2.5-1 thru 2	A	11- 1-68
	// Punch	7.2.6-1	A	11- 1-68
	// Print	7.2.7-1	A	11- 1-68
	Running CMS Batch	7.3.0-1 thru 2	A	11- 1-68
	Appendix	A-14	E	11- 1-68

672

IBM Cambridge Scientific Center
CP-67/CMS Support Group
545 Technology Square
Cambridge, Massachusetts 02139
May 26, 1969

To: CP-67/CMS Installations

Enclosed is one copy of the rough drafts of updates to the CP-67/CMS User's Guide. These updates reflect the contents of CMS Version I. 6 and CP-67 Version 2. 4. These updates will be revised and corrected and sent out formally to all holders of the User's Guide at a later date. Please note that this is only a temporary update for informing your users of the new contents of the system.

Also enclosed is a write-up on the installation of the CMS Batch Monitor.

The CP-67 Installation Guide does not explain how to define TTY 33 or 35's in the REALIO deck. To define Teletype lines, use the type "TT35T" in the DMXDV macro. For example,

```
TTY024 DMXDV RDEVADD = 024, TYPE = TT35T, SAD = 2,        x
               RDEVPNT = TTY025
```

The type TT35T defines a Teletype Model 33 or 35, whereas the type 2702T defines either a 1052 or 2741. The SAD address does not indicate the terminal type.

The following is a brief list of new changes and additions to CP-67 and CMS:

CP-67

1. DIAL improvements.
2. New user console functions - XFER, DIAL, PURGE, SLEEP, SPOOL, DISCONN, LINK
3. New operator console functions - DRAIN, START, SPACE.
4. Changes to console functions - MSG, QUERY, EXTERNAL, SET.

673

CMS

1. New commands:

CHARDEF	TPCOPY
LINEND	MODMAP
LINK	STATE
CPFUNCTN	TAPEIO
SORT	DUMPF
CNVT26	DUMPD
CVTFV	BLIP
COMPARE	RT

2. LISTF (EXEC) now creates the file CMS EXEC, not LISTF EXEC.
3. When a command is typed in, the order of search is for an EXEC filetype and then a MODULE filetype.
4. Multiple commands can be stacked on one input line - See LINEND.
5. The line delete and character delete symbols, as well as the logical tab and backspace characters, can be redefined via CHARDEF.
6. The # is the default for the line-end character as well as the logical tab character, with the line-end character taking precedence.
7. Dynamic loader facilities.
8. PL/I execution.
9. New SCRIPT PRINT options and control words.
10. CP console functions can be issued from CP - see CPFUNCTN.
11. Mini-disks can be dynamically added to a user - see LINK.
12. Changes to the existing commands:

LISTF	MACLIB
OFFLINE	TXTLIB
TAPE	GLOBAL
START	LOAD
KX	

Good Luck and Happy Timesharing!

Love Seawright

CMS Batch Monitor Installation

Release I. 6

Using a copy of the NUCLEUS text deck of version I. 6, remove the DEBUG text deck - located about three/fourths into the deck and preceeded by "OFFLINE READ DEBUG TEXT." Replace the Debug deck with the following text decks:

BATCH, BOMB, DECODECC, IPL, JCB, LIST, PRESTORE, SYSCTL

These text decks may be obtained by assembling the respective source decks from the CMS SOURCE file.

Next, place the following REP cards into the NUCONTS deck - the first text deck after the loader deck:

```
Cols:   1   2     7        14                              50

      12-2-9 REP 000354  0010185,0000,E3C1,D7F5  TAP5
      12-2-9 REP 000360  0010186,0000,E3C1,D7F6  TAP6
      12-2-9 REP 000390  0010187,0000,E3C1,D7F7  TAP7
```

And place the following REP into the BATCH text deck:

```
12-2-9 REP 001034  001D046     IPL Rep
```

On the BATCH virtual machine, IPL the nucleus card deck. The following questions will be asked:

Q: REWRITE NUCLEUS?

R: YES.
Prepare to write a core-image copy of the Batch Nucleus.

Q: DEVICE ADDRESS?

R: 0CUU
Where CUU is the address of a disk area that will be used solely for the Batch nucleus - say, two cylinders on device 194.

Q: STARTING CYLINDER?

R: 000N
Where N is at least "1" and not more than "the number of cylinders minus one." For a 2314, 2 cylinders is sufficient; 2311, 4 cylinders are needed.

Q: VERSION IDENTIFICATION:

R: BATCH I. 6 (or any 15 bytes of prose)

The nucleus will be written onto device CUU starting at cylinder N. Then the message "READY" will be typed.

Before running any Batch job streams, IPL CUU.