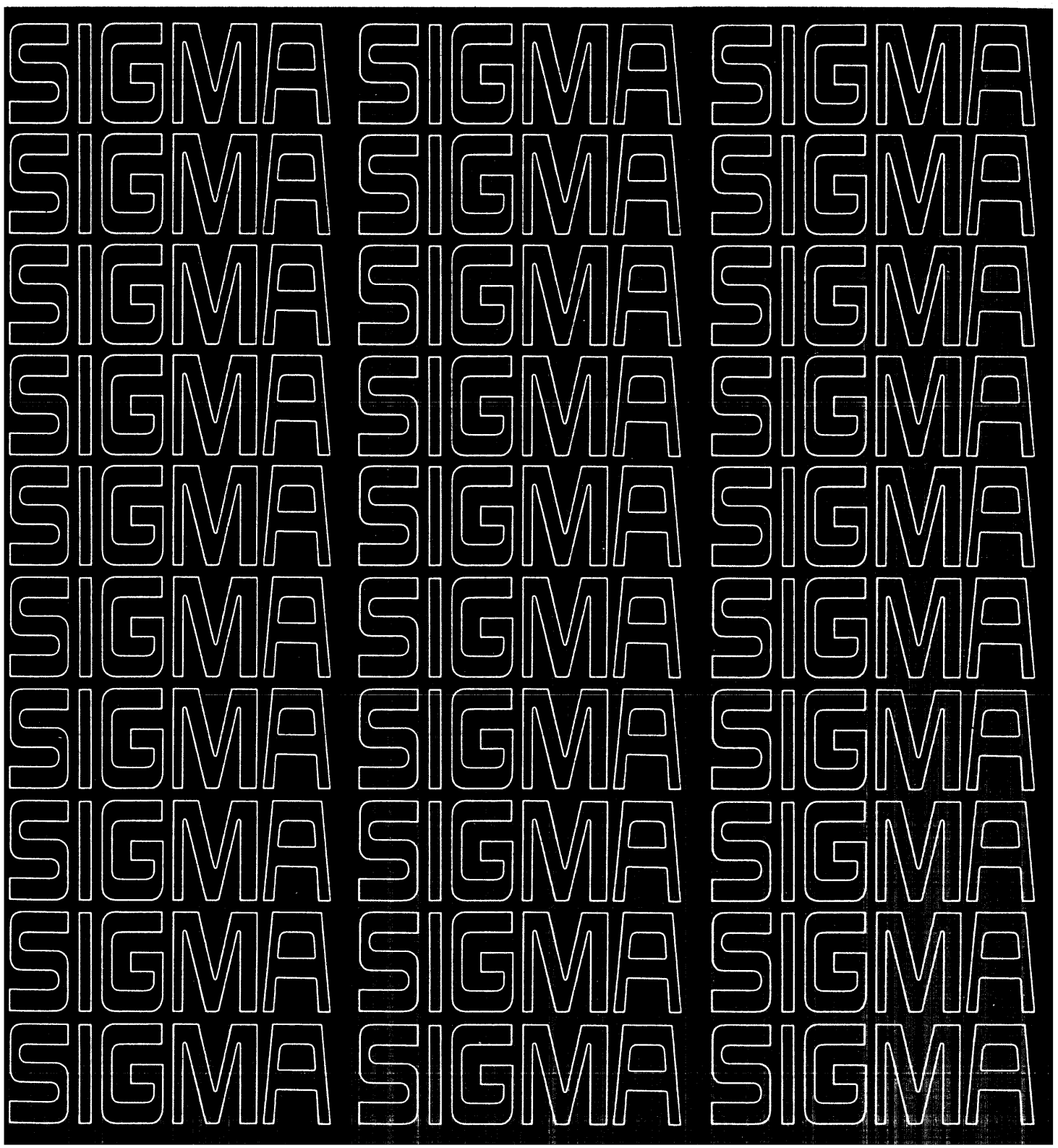
SDS SIGMA 5/7 BATCH TIME-SHARING MONITOR (BTM)

Reference Manual

SCIENTIFIC DATA SYSTEMS



SYSTEM CALs

BPM CALs

Call	FPT Code	Function	Comments
CAL1,1	X'01'	M:REW	Allowed
	X'02'	M:WEOF	Ignored
	X'03'	M:CVOL	Ignored
	X'04'	M:DEVICE (PAGE)	Ignored
	X'05'	M:DEVICE (VFC)†	Allowed
	X'06'	M:SETDCB	Allowed
	X'0B'	M:DEVICE (DRC)	Ignored
	X'0C'	M:RELREC	Allowed
	X'0D'	M:DELREC	Allowed
	X'0F'	M:TFILE	Allowed
	X'10'	M:READ†	Allowed
	X'11'	M:WRITE†	Allowed
	X'12'	M:TRUNC	Allowed
	X'14'	M:OPEN	Allowed
	X'15'	M:CLOSE	Allowed
	X'1C'	M:PFIL	Allowed
	X'1D'	M:PRECORD	Allowed
	X'20'	M:DEVICE (LINES)	Ignored
	X'21'	M:DEVICE (FORM)	Ignored
	X'22'	M:DEVICE (SIZE)	Allowed
	X'23'	M:DEVICE (DATA)	Ignored
	X'24'	M:DEVICE (COUNT)	Ignored
	X'25'	M:DEVICE (SPACE)	Ignored
	X'26'	M:DEVICE (HEADER)	Ignored
	X'27	M:DEVICE (SEQ)	Ignored
	X'28'	M:DEVICE (TAB)	Ignored

†Available on console I/O.

Call	FPT Code	Function	Comments
CAL1,1	X'29'	M:CHECK	Allowed
	X'2A'	M:DEVICE (LINES)	Ignored
CAL1,2			Illegal
CAL1,3			Illegal
CAL1,4			Illegal
CAL1,5			Illegal

FREQUENTLY-USED COMMANDS[†]

[†]Delta commands are indexed on page 52 of this manual.

Price: $3.50

BATCH TIME-SHARING MONITOR REFERENCE MANUAL

for

SDS SIGMA 5/7 COMPUTERS

PRELIMINARY EDITION

90 15 77A

February 1969

SCIENTIFIC DATA SYSTEMS/701 South Aviation Boulevard/El Segundo, California 90245

Printed in U.S.A.

RELATED PUBLICATIONS

Title	Publication No.
SDS Sigma 5 Computer Reference Manual	90 09 59
SDS Sigma 7 Computer Reference Manual	90 09 50
Mathematical Routines Technical Manual for SDS Sigma Computers	90 09 06
SDS Sigma Symbol and Meta-Symbol Reference Manual	90 09 52
SDS Sigma 5/7 BASIC Reference Manual	90 15 46
SDS Sigma FORTRAN IV Reference Manual	90 09 56
SDS Sigma FORTRAN IV Operations Manual	90 11 43
SDS Sigma FORTRAN IV Library Technical Manual	90 15 24
SDS Sigma FORTRAN IV-H Reference Manual	90 09 66
SDS Sigma FORTRAN IV-H Operations Manual	90 06 44
SDS Sigma FORTRAN IV-H Library/Run-Time Technical Manual	90 11 38
SDS Sigma 5/7 Batch Processing Monitor Reference Manual	90 09 54
SDS Sigma 5/7 Batch Processing Monitor Operations Manual	90 11 98
SDS Glossary of Computer Terminology	90 09 57
SDS Sigma Multipurpose Keyboard/Display Reference Manual	90 09 82
SDS Sigma Message-Oriented Communications Equipment Reference Manual	90 15 68
SDS Sigma Character-Oriented Communications Equipment Reference Manual	90 09 81

CONTENTS

APPENDIXES

ILLUSTRATIONS

TABLES

1. INTRODUCTION

The SDS Sigma 5/7 Batch Time-Sharing Monitor (BTM) is an extension of the Sigma 5/7 Batch Processing Monitor (BPM)[†]. The BTM time-sharing system offers efficient, comprehensive, and compatible batch processing service concurrently with conversational time-sharing.

On-line services are provided by the following subsystems:

- BASIC processor
- EDIT source file editor
- FERRET file utility subsystem
- FORTRAN IV-H compiler
- SYMBOL assembler
- LOADER object module loader
- BPM batch job controller
- DELTA debug package

Batch processing services are accessed by:

- Local batch submitted at the central computer
- Remote batch through high-speed terminals
- Terminal batch for on-line users

BTM users may choose between any of the available system services with ease and flexibility, in either the on-line or batch mode, because:

- Batch mode equivalents are provided for the on-line FORTRAN IV-H, BASIC, and SYMBOL processors
- A common file-management system provides compatible file structures
- Comprehensive and compatible accounting information is provided for both modes of operation

Installation management may control job processing by:

- Specifying which remote terminals may have access to the batch processing background
- Permitting system access only to those users whose name and account number are recognized
- Broadcasting messages to all terminal users from the central computer
- Shutting down user terminals, from the central computer, to recover the on-line core memory space for batch processing
- Specifying the highest batch job priority allowed for each individual terminal user
- Altering the priorities of jobs in the batch stream

Typical hardware configuration requirements for use of BTM are as follows:

- CPU (Sigma 5 or 7) with memory protection, floating-point arithmetic, and two each of the following: register blocks, external interrupt levels, and real-time clocks
- 48K core memory
- Card reader
- Typewriter
- 6MB RAD storage (system and file storage)
- High-speed RAD and Selector IOP (swap storage for 24-user system)
- Two magnetic tape units
- Line printer
- Character-oriented-communication controller (line interface units, send/receive modules, and format timing group as required)
- Teletype unit for each on-line user

Recommended in addition to the above are:

- Card punch
- Decimal arithmetic option
- 16K core memory

USER TERMINAL FUNCTIONS

The user terminal consists of an ASR Teletype (with paper tape capability) or KSR Teletype (keyboard only). The Teletype provides the communication link between the user and the computer. Input is typed in by the user and output is printed by the BTM system. Figure 1 shows the Teletype keyboard and Figure 2 the control panel.

[†]See the SDS Sigma 5/7 Batch Processing Monitor Reference Manual, publication number 90 09 54.

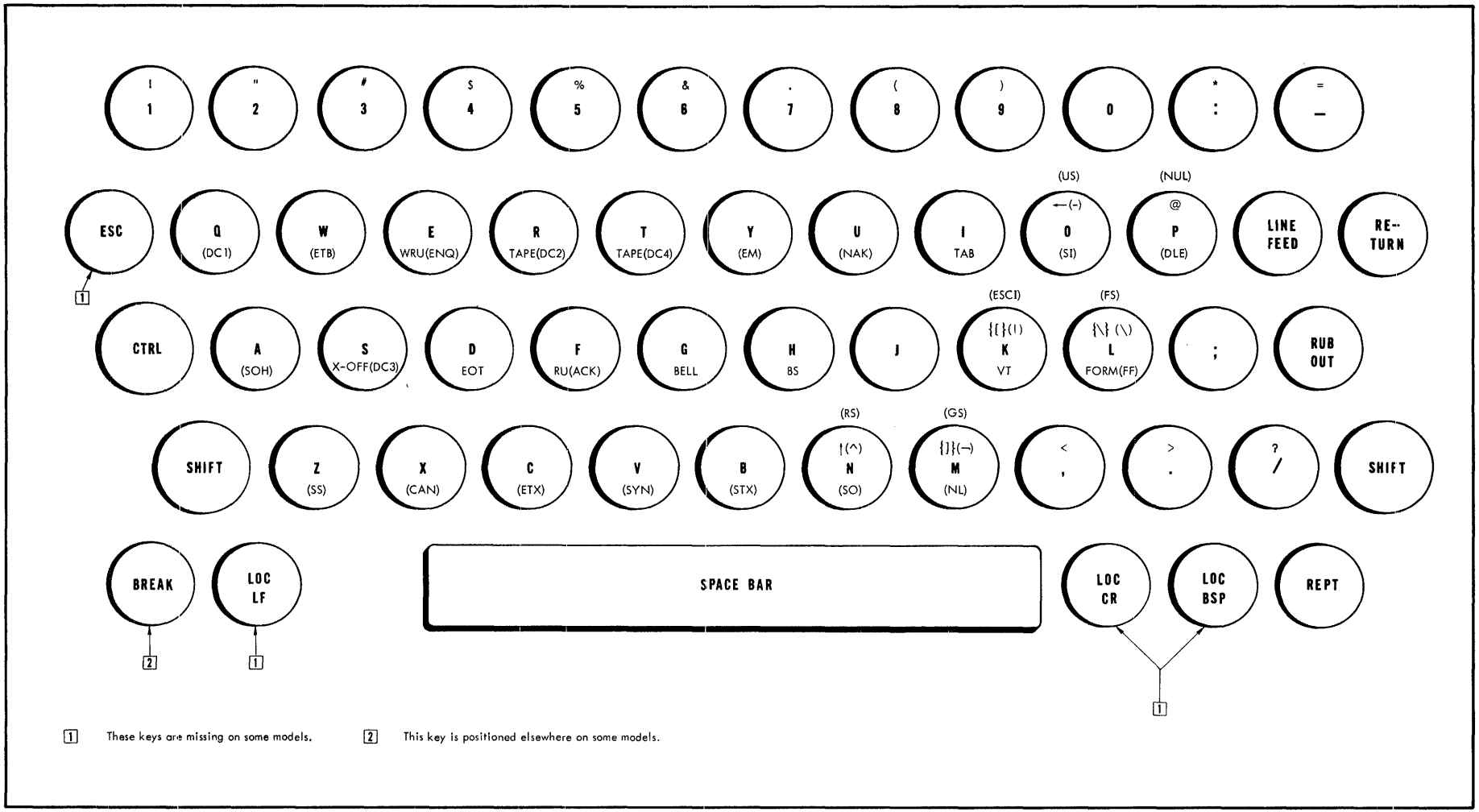


Figure 1. Teletype Keyboard Configuration

In this diagram, characters obtained by depressing the SHIFT key are shown at the top of the key and characters obtained by depressing the CTRL key are shown at the bottom of the key. Characters obtained by depressing both SHIFT and CTRL keys are shown above the character key. On the actual keyboard, all unparenthesized forms appear at the top of the key.

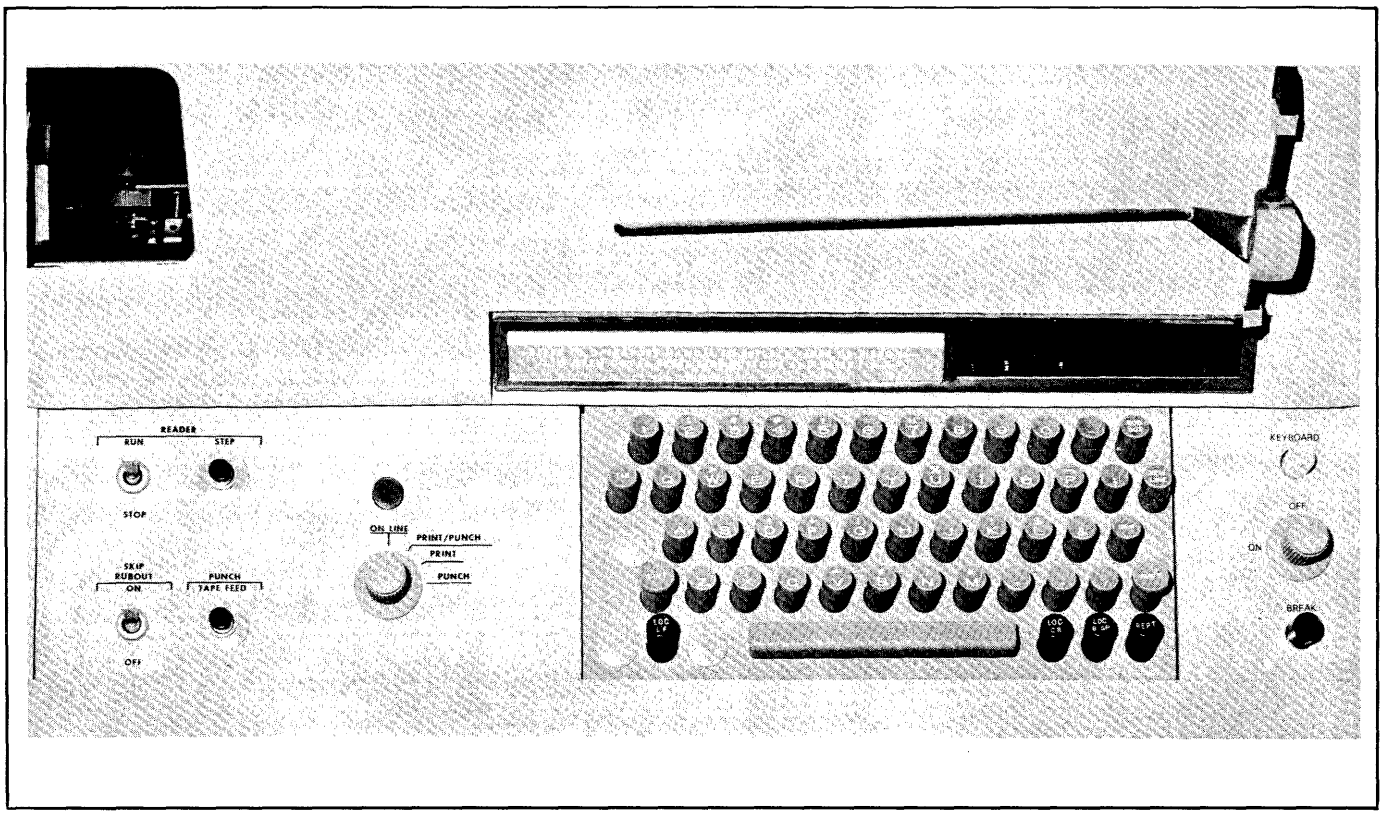


Figure 2. User Terminal Control Panel

Appendix A describes the functions of the operating controls and special keys.

ON-LINE OPERATIONS

Time-sharing, or more precisely resource sharing, in the BTM system is achieved by alternately processing part of a batch job and part of an on-line job. Batch and on-line processing modes are each allocated a specified time slice or "quantum". When the current batch quantum expires, the next active on-line user's job is processed. The core/disc swapping of on-line jobs is typically performed during the batch time quantum. The area of core memory dedicated to batch processing is not swapped to disc storage.

I/O FILES

I/O files for on-line users are maintained on the RAD. The terminal user may request that these files be copied to card punches, line printers, and magnetic tapes at the central computer or to his terminal. Files input from punched cards or magnetic or paper tape may be saved on the RAD for later access by the terminal user. Complete file security is provided by the comprehensive file management system.

OFF-LINE[†] OPERATIONS

Batch jobs may be entered from a user terminal through use of the BPM subsystem (see Chapter 6), thus affording the terminal user full capabilities of batch processing.

All of the BPM processors (e.g., COBOL, SORT, MERGE, SDS FORTRAN IV, 1401 Simulator, Meta-Symbol, etc.) are available to the on-line user in the batch mode.

Batch jobs submitted in this manner are then merged into the background job stream on the RAD and are queued for execution (during the batch quantum) according to their relative priorities.

The on-line user may, at any time, query the system to determine the current status (i.e., waiting, running, or completed) of any specified batch job.

[†] "Off-line" usually refers to operations performed by peripheral devices not controlled by a CPU. For convenience in this manual, however, "off-line" is used extensively to identify all operations not performed by the on-line BTM system.

2. START-UP PROCEDURE

On-line service is obtained from BTM by activating the user terminal and logging into the system as outlined below.

ACTIVATION PROCEDURE

The user terminal is activated by depressing the "ORIG" key (see Figure 2, Teletype Control Panel). The key latches in the depressed position and is illuminated; it remains in this condition until the user deactivates the terminal (see "BYE Command", Chapter 3).

LOG-IN PROCEDURE

After the terminal is operational, the user alerts the BTM system by momentarily depressing the BREAK key or (ESC) (ESC) (see Figure 1, Teletype Keyboard). When the system is operative, the following messages will be printed:[t]

BTM SYSTEM IS UP
date and time
!LOGIN:

The exclamation character in the last line of the above message informs the user that he is communicating with the BTM Executive; the colon signifies a request for data. In this instance, the data required from the user is as follows:

!LOGIN: name,acct[,pass] (RET)

where

name is any 1 - 12-byte name.

acct is any 1 - 8-byte account ID.

pass is any 1 - 8-byte password.

(RET) is the Carriage Return key.

Embedded blanks are possible in any of the specifications. Only one password is legal for each unique name-account pair. The same password can be associated with several users and several names may be used with the same account. The password may be made up of nonprinting control characters[tt] if the user so desires.

After the user responds to the log-in request, his name, account, and password combination is checked for validity. If valid, that is, if the name-account-password combination is in the accounting file, he is logged in and Executive services are at his disposal. If invalid, a new log-in request is issued. If no accounting file is in existence, anyone may log-in under an arbitrary name-account pair.

After LOGIN, the system prints on the user's console:

ID = α

where α is A, B, . . . , Z, 1, . . . , 6 depending on the COC line number of the user's terminal which may range from 0-31. Higher line numbers recycle through the characters A through 6. For instance, a terminal connected to line number 3 receives the message:

ID = D

In any of the subsystems that provide built-in output or scratch files, this ID letter is made part of the name in order to provide unique file names for simultaneous users under the same account.

These subsystems and file names are:

Subsystem	File Name	File Function
BASIC	RUNαFIL	scratch file
SYMBOL	BOTEMPα	default binary out
FORTRAN	BOTEMPα	default binary out
	SOTEMPα	default source out
LOAD	BOTEMPα	default binary input
	SDαFIL	scratch file for debugging symbol tables

When the user logs out (BYE), his running statistics are updated and his statistics for the current session are displayed, along with the amount of disc space still available to him. The accumulated totals are available only to the installation supervisor (see Chapter 12, "Supervisory Subsystem").

Following the log-in procedure, BTM outputs an exclamation character, indicating that the Executive is ready to accept commands.

If BTM prints an exclamation character as soon as the BREAK key is depressed, this indicates that a previous user of the terminal has failed to log out. In such a case, the user should key in a BYE command before logging in.

The following example shows the proper log-in procedure.

BTM SYSTEM IS UP
10/15/69 09.42
!LOGIN: SMITH, SDS 123, PASSOK (RET)
!

---

[t] Messages output by BTM are underlined throughout this manual, although in an actual session such Teletype output is not underlined.

[tt] A control character is produced by holding down the CTRL key and then depressing a key for any character on the user console.

3. BTM EXECUTIVE

The BTM Executive responds to control commands that direct the flow of the user's on-line work. There are 14 such commands:

1. ASSIGN
2. TABS
3. BASIC
4. BPM (Terminal Batch Entry)
5. EDIT
6. FERRET
7. FORTRAN
8. LOAD
9. SYMBOL
10. SUPER
11. SAVE
12. RESTORE
13. PROCEED
14. BYE

ASSIGN allows the user to specify RAD I/O file assignments. TABS enables tab stops to be set for the Teletype at the user terminal. BASIC, BPM, EDIT, FERRET, FORTRAN, LOAD, SUPER and SYMBOL are subsystem calls. SAVE and RESTORE allow the user to interrupt his on-line work and resume the session at a later time. PROCEED permits an interrupted subsystem to continue operation. BYE is used to terminate an on-line session.

Executive commands may be given only when the Executive is in control, as indicated by the printing of an exclamation character on the Teletype. The user types in the first two letters of the mnemonic and the Executive prints the remaining letters. If the Executive does not recognize a command, it informs the user by printing a question mark. It then begins a new command line with an exclamation character, so that the user may repeat the command.

After the Executive has completed the command mnemonic, the user supplies any necessary parameters and terminates the command by giving a carriage return. Commands that never include parameters following the mnemonic (e.g., all subsystem calls) are terminated automatically by the Executive: the user should not supply the ⏎. At any time prior to the termination of a command, the user may cancel the command by depressing the BREAK key momentarily. The Executive ignores a canceled command and begins a new command line.

Discussions of the use of Executive commands and the BREAK key are given below.

RAD FILE ASSIGNMENT

ASSIGN COMMAND

The ASSIGN command controls all RAD file assignments for BTM users, with the exception of assignments made automatically by subsystems. The format of this command is similar to that of the ASSIGN command of the BPM. The general form of the command is shown below. Items shown in capital letters are required; those in lower case represent parameters. Optional items are denoted by brackets, although no brackets are actually used in ASSIGN commands.

!ASSIGN dcb name[,(FILE,file name)] [,(option)...]

where

dcb name is a DCB name of up to eight alphanumeric characters. The first two characters must be either F: or M: (F: denotes user DCBs and M: denotes system DCBs).

file name is a user-selected identifier of up to eight letters and/or digits.

Files can be saved for future use through the PERM option (see below). The various ASSIGN options recognized by the BTM Executive are listed below. Each option may be enclosed in parentheses, and options must be separated by commas. No blanks are allowed within the specification field. If only the DCB name is specified, default values are assumed for all option parameters and the default assignment (if any) for the DCB is assumed also.

Option	Meaning	Comment
IN	Input file	Existing file to be read only.
OUT	Output file	New file to be written only.
INOUT	Update file	Existing file to be read and written.
OUTIN	Scratch file	New file to be written and read.
PERM	Permanent	Default for input files.
TEMP	Temporary	Default for output files.
HERE	Teletype	Assign the DCB to the user Teletype.
REL	Release	Same meaning as in BPM ASSIGN command.
SAVE	Don't release	
PASS	Password	
READ	Read accts.	
WRITE	Write accts.	

Sample ASSIGN commands are shown below:

!ASSIGN M:SI,(FILE,SIMBOLIK),(IN),(TEMP)

The above command specifies that the RAD file named "SIMBOLIK" is to be used as a source input file. It may only be read, and is not to be saved when the user logs out.

!ASSIGN M:DO,(HERE)

This command specifies that the user terminal is to be used for diagnostic output.

!ASSIGN M:BO

This command specifies that the M:BO DCB is to be returned to its default assignment.

TAB SETTING

TABS COMMAND

The TABS command sets up a tab sequence for the input from the user's Teletype. The form of the command is shown below.

!TABS [t_1[,t_2]···[,t_8]]

where

t_i specify the tab positions desired. A blank specification will cause all existing tabs to be deleted.

A TABS command has the effect of clearing any previous tab settings. Thus, a maximum of eight tabs may be used in any one line of input. While keying in a line, a previously set tab is employed by depressing the ESCAPE key followed by the letter I. At this point, the input image is spaced to the next tab position, if any. If no tab exists beyond the present column, a "?" is echoed and the input image is spaced by one column.

SUBSYSTEM CALLS

FILE ASSIGNMENT

With the exception of calls to the FORTRAN, LOADER, BPM, and SYMBOL subsystems, all file assignments are made after the subsystem is called. The user terminal is the default assignment for DO, but it may be reassigned. In the case of FORTRAN and SYMBOL, the user may assign SI, LO, and BO to suit his own requirements. In the case of BPM, SI may be assigned to meet user requirements. The user terminal is the default assignment for SI and LO, and BO is assigned to the file "BOTEMPα" unless reassigned by the user. BOTEMPα is also the default assignment for BI, the default input to the LOADER subsystem.

CALLS

Subsystem calls may be made whenever the BTM Executive is in control, assuming that proper file assignments have been made. The forms of the seven subsystem calls are shown below.

!BASIC
!BPM
!EDIT
!FERRET
!FORTRAN
!LOAD
!SYMBOL
!SUPER

Once a subsystem call has been made, control is transferred to the called subsystem and the Executive remains inactive until control is returned to it by the subsystem or by the user (see "ESCAPE and PROCEED", below).

SAVE AND RESTORE

The SAVE and RESTORE functions are Executive services that permit the user to interrupt his on-line task and resume it at a later time.

SAVE COMMAND

The SAVE function causes the user's context block (i.e., machine environment), subsystem area, and user area to be written to a disc file under a user-specified file name. The SAVEd file can then be retrieved from the RAD at a later time during the same session or during a future on-line session.

The user may want to retrieve the SAVEd file from the RAD prior to logging out, but may not wish to SAVE it permanently, in which case he may choose to declare the file temporary (i.e., to be released at log-out time). This may be done by means of the TEMP option (see below).

Prior to saving the environment, BTM automatically closes all active files. The user is responsible for any required file positioning, etc. Hence, it usually is not possible to successfully resume a compilation, assembly, or load.

As with any Executive command, a SAVE command may be given only when the BTM Executive is in control of the system. The ESC key (see below) can be used to return control to the Executive.

The SAVE command has the form shown below.

!SAVE file name[,TEMP / ,PERM]

where

file name specifies a user-selected name of up to eight letters and/or digits.

TEMP specifies that the SAVEd file is to be released when the user logs out. If TEMP is specified, the file cannot be RESTOREd (see below) after logging out, but may be RESTOREd at any time prior to logging out.

PERM specifies that an existing TEMPorary file is to be changed to PERManent.

RESTORE COMMAND

The RESTORE function reloads and restarts the activity represented in a file previously created in response to a SAVE command. If the specified file does not exist, was not SAVEd, or cannot be accessed for any other reason, a question mark will be printed. If the TEMP option (see below) is specified in the RESTORE command, the file will be released at the next log-out time.

The form of the RESTORE command is shown below.

!RESTORE file name[,TEMP]

where

file name specifies the name previously assigned to a SAVEd file.

TEMP specifies that the file is to be released at log-out time.

The user is returned to the Executive level. The PROCEED command may be used to restart a process interrupted by ESC ESC (see below).

ESCAPE AND PROCEED

The ESCAPE and PROCEED functions are Executive services that permit the user to interrupt a subsystem or user's program and return to it from the Executive level.

ESC KEY

The ESC key returns control to the BTM Executive. If a subsystem is in control, a single ESCAPE[†] is sufficient; if the user's program is being executed, two successive ESCAPEs are required.

Once the Executive has gained control, any Executive command can be given. However, if the user wishes to return to the interrupted activity without restarting it, he must not give any subsystem calls before returning control to the interrupted subsystem or program by means of a PROCEED command (see below).

[†]Each ESCAPE requires the ESC key to be depressed twice, which reduces the likelihood of accidental ESCAPEs.

PROCEED COMMAND

The PROCEED command continues a subsystem or program that has been interrupted by a previous ESCAPE (see above).

The PROCEED command has the form shown below.

```
!PROCEED
```

LOG-OUT PROCEDURE

An on-line work session is terminated by giving the Executive command BYE.

BYE COMMAND

The BYE command informs the Executive that the user is logging out of the system. The Executive responds by printing the time and date on the line following the command. It then prints a summary of statistics for the current session (see below).

The BYE command has the form shown below.

```
!BYE
time-date
RAD SPACE      xx      (Granules of disc space used)
CPU         xx.xxx     (Minutes of CPU time used)
I/O         xx.xxx     (Minutes of I/O time used)
OVERHEAD    xx.xxx     (Minutes of CPU overhead time
                       used)
```

The user terminal is disconnected from the computer automatically, following a BYE command.

4. TELETYPE OPERATIONS

All Teletypes used in conjunction with the BTM system are required to operate in the full-duplex mode. In this mode, the input keyboard and the output printing mechanism are completely decoupled, so that depressing a key on the keyboard does not, in itself, cause the corresponding graphic to be printed. Printing can be achieved only by direct command of the attached computer. In the normal mode, the interrupt routine which processes the input character will immediately transmit back to the Teletype printing mechanism each such input character, thus causing it to be "echoed". Should there be any delay in retransmitting the character, the echo may fall behind the input – a disconcerting situation, at first. Moreover, since the input and output operations are entirely distinct, one may even type input while a subsystem is printing output.

Under BTM, a subsystem is "activated" whenever a sufficient amount of Teletype input has been accumulated for it to process, and it is "dismissed" if it requires further input when none is available. Because there is a substantial amount of overhead involved in activating one subsystem (and thereby dismissing another subsystem), it is desirable to allow as much Teletype input as possible to accumulate before activating the subsystem to process it. As the requirements for activation differ among subsystems, BTM provides each subsystem with the capability to specify what conditions constitute sufficient Teletype input. There are four such sufficient conditions (comprising the "activation classes"), as follows:

1. Input of a line feed, (LF) or carriage return, (RET) only.
2. Input of one of the characters /, =, (ESC)I, ↑, (RET), or (LF).
3. Input of a punctuation character (see Appendix A), or of (LF) or (RET).
4. Input of any character whatever.

Clearly, condition 4 is the most severe, in that it requires the subsystem to be activated after each character is typed; whereas condition 1 is the most desirable, in that it requires activation only after each complete line of input.

The primary effect of this on the user results from the fact that BTM will stop echoing input immediately after an activation character is typed in and will not resume echoing until the subsystem has been activated and has read from BTM all the accumulated input, through the first character of the current message string. (This procedure is necessary to keep input and output correctly sequenced, since the subsystem may type out after each activation character.) Thus, echoing always stops until the subsystem has completely read the last activation image from BTM. Because some subsystems may change activation classes as they run, it is sometimes difficult to know at a given point what the current activation class is. Basically, when the Teletype stops echoing input, an activation character has been typed and the subsystem has not yet been called.

SPECIAL EDITING FEATURES

BTM provides the Teletype user with several in-line editing capabilities, including the ability to backspace over characters and to erase lines. However, editing may only be performed on those characters typed in since the last activation character was typed. That is to say, one cannot edit input across the boundaries defined by activation characters. When using the editing features described herein, the user must be aware of the activation class of the subsystem being executed to make correct use of these features, although trying to edit across an activation character boundary will never cause any unintended change to the input, but will rather cause part or all of the effect of the edit to be lost.

Because the Teletype is a low-speed device, all Teletype I/O must be buffered, raising the possibility of I/O buffer overflow. Whenever the input buffer gets within ten characters of being full, each further input character will cause the Teletype bell to be rung immediately, to signal the impending buffer overflow. When the bell rings in this manner, the user should wait a few seconds before typing further. When the bell rings to signal impending buffer overflow, a control S (XOFF) is generated to turn off the paper tape reader (if in use). Later, when BTM requires input, a control Q (XON) is generated to turn the paper tape reader back on. When the active subsystem has processed some of the data from the input buffer, the bell will no longer ring when input is entered. If, despite this warning, the input buffer becomes completely filled, each overflow character will be entered into the buffer as a (RET)(which always causes activation) and will be echoed as a question mark. The actual input character is lost, and the (RET) replaces the last character in the buffer.

Should the output buffer approach overflow while a subsystem is doing Teletype output, that subsystem will be dismissed for a while to allow the buffer to empty. Because the input echo is placed in the output buffer the same as any other output, it is possible for the user, by typing ahead of the echo, to overflow the output buffer with echo characters. In such a case, some echo characters will be lost. The actual input is not lost and will be correctly passed to the subsystem. Due to the way BTM sequences teletype I/O, such a loss of echoing can occur whenever the sum of the number of characters output by a subsystem and the number of characters the user has typed ahead of the echo exceeds the output buffer size (normally, one hundred characters).

ACKNOWLEDGE ((ESC) Q)

This code does not perform an editing function but is rather a device for letting the user know if BTM is still operating. This is especially useful when the Teletype becomes inactive for a long period and it is not clear whether the system has gone down or whether the subsystem is slow in responding. Typing this character will thus result in a double

exclamation point (!!) being printed immediately (not in echo sequence) if BTM is still operating.

BACKSPACE (ESC RUB)

This code causes the last deletable[†] character typed in to be "erased". Generally, this will be the character immediately preceding the backspace code, but in cases in which the preceding character is one which cannot be backspaced over, the backspace will erase the last character which can be backspaced over. Multiple backspaces are cumulative in that two successive backspace codes will erase the last two deletable characters, etc. Backspacing can be performed only back to (but not including) the last activation character. Any attempt to backspace over or beyond the activation character is ignored. A ← is echoed to signal that the backspace has been performed.

ERASE (ESC X)

Typing this character causes all input from the ESC X code back to (but not including) the last activation character to be erased. It is not possible to erase the activation character or beyond, and any attempt to do so will be ignored. A LF and RET echoed to signal that the erase has been performed.

LOCAL NEW LINE (ESC RET)

Typing this code causes a LF and RET to be echoed, but no character is passed to the user subsystem. This permits the Teletype to be positioned physically to a new line without logically terminating the current input line.

RETYPE (ESC R)

Typing this code causes all input from the last activation character through the ESC R to be retyped with all editing correctly performed (namely, backspacing, tabbing, and erasing.) Thus, the retyped image will be exactly identical to the image that the subsystem sees. This retyped image starts on a new line.

TAB (ESC I)

This code is used in conjunction with the TABS command. With the TABS command, the user may set or reset up to 8 tab stops on his Teletype. When tab stops are set, the user may type in the tab code at any point, to skip over to the next tab stop. When entered, the tab code will echo the appropriate number of spaces to the next tab stop to produce the same effect as on a normal typewriter. If an attempt is made to tab beyond the last tab stop set, a question mark (?) will be echoed to signal this fact to the user, but a single space will be passed to the subsystem (and retyped) for this tab.

[†]See Appendix A for a complete listing of characters that may not be backspaced over.

The input line is numbered from 1 to n for the purpose of tabbing. Position 1 is always the position of the first character after the last activation character. Thus, all tab stops are measured relative to the last activation character, not the beginning of the physical line; tabbing is therefore primarily of value when the activation class is LF or RET. In addition, the line is numbered according to the edited input image, not the image as echoed on the Teletype. Giving a tab, even after assorted backspaces, erases, or other such editing functions, will always result in the correct number of spaces being echoed and transferred to the subsystem.

For the purposes of backspacing, typing an ESC I code is the same as typing the n spaces that the tab generates. Thus, each backspace after a tab will erase one of the n spaces of the tab, not the entire tab. To erase all n spaces, n backspace characters must be typed.

STANDARD INPUT MODES

To allow the user to relate the information on Teletype operation to actual subsystem usage, the activation classes for most standard console operations are summarized below. Note that the majority of I/O is done under the activation class: "activate on LF or RET". This is desirable because the special editing features are easiest to use in this mode.

At the Executive level:

1. Immediately after typing the Executive "prompt" character (!), the activation class is: "activate on every character". In this mode, no editing whatever is possible and any editing character typed will be passed to the Teletype command processor unchanged. Thus, in the sequence

   ! C ESC RUB B,

   not only is the effect of the backspace lost, but BTM will receive the four characters, 'C ESC RUB B', rather than the two it expects.

2. After entering the ASSIGN processor, the remainder of the command is read under the class: "activate on punctuation, LF or RET". That is,

!ASSIGN	M:LO, etc.
activate on every character	activate on punctuation, etc.

3. All the remaining Executive level command processors read the rest of the command under class: "activate on LF or RET". For example,

!SAVE	ALPHA,PERM RET
activate on every character	activate on LF or RET

At the subsystem/user level:

1. All subsystem input that is read through DCBs assigned to the console is gathered under the activation class: "activate on ⓛⓕ or ⓡⓔⓣ", and is prompted by a colon (:).

2. The BPM, BASIC, EDIT, FERRET and SUPER subsystems read all input under the class: "activate on ⓛⓕ or ⓡⓔⓣ", with the following exception:

   a. When the BPM subsystem expects a Y or N answer, it reads this character in the "activate on every character" mode.

3. The FORTRAN, LOAD, and SYMBOL subsystems acquire all option lists and input specifications in the "activate on ⓛⓕ or ⓡⓔⓣ " mode.

4. The machine language debugger, Delta, reads in the "activate on /, =, ⓔⓢⓒ I, ↑, ⓡⓔⓣ, or ⓛⓕ" mode.

5. BASIC SUBSYSTEM

The BASIC sub-system is called by the Executive Command

!BASIC

When ready to accept input, BASIC prompts the console with the ">" character.

BASIC runs in two distinct modes:

1. Editing - In this mode programs may be created at the console. A program, once created, may be saved on the disc file, and any previously saved program may be loaded for further editing or execution. The system is initially in the editing mode with an empty text area.

2. Compilation and execution - In this mode, the program developed during the editing phase is compiled and, if no errors are detected, executed. If compilation errors are found, editing mode is restored automatically, and the text area will contain the program text just compiled.

EDITING MODE

LINE INSERTION

If, following the prompt character (>), the user types a valid BASIC line number, followed by any non-numeric character[†], and if the line contains less than 86 characters, the line will be entered into the text edit area. Any line with the same line number previously entered into the text area will be deleted. Following this, a syntax check is made on the line; and, if it is not legal, the line is deleted.

If more than five leading numeric digits (the first of which is not 0) are typed, the message

LINE # ERROR

will be typed preceded by the first six characters encountered in acanning the line.

If more than 85 characters are typed on the line, the message

LINE TOO LONG

will be typed. The line will be completely ignored.

If the syntax check uncovers any errors, the appropriate diagnostic message (as described in the reference manual) will be typed at the console. The prompt character is then issued and the system waits for input. Lines cannot be inserted when only 150 bytes of core storage remain.

[†]Blanks are never significant to BASIC unless they are within a pair of quotes to form a valid alphanumeric constant or text string, or in Image statements.

If one tries, he gets the error message

PROGRAM TOO LARGE

When 500 bytes or less remain, the message

SPACE LIMIT NEAR

results.

LINE DELETION

If following the prompt, the user types a valid BASIC line number followed by a carriage return, the line (in the text editing area with that line number will be output with the error message:

NO PROGRAM

If the user wishes to start with a fresh text editing area, he should type CLE[AR] followed by a carriage return. This will put the user level in the editing mode with an empty text area. Items enclosed in brackets [ ] are optional.

The user may also type

$$\text{DEL[ETE]}\left\{\begin{matrix}\text{line \#}\\ \text{line \#}_1 - \text{line \#}_2\end{matrix}\right\}\left[,\ \begin{matrix}\text{line \#}\\ \text{line \#}_1 - \text{line \#}_2\end{matrix}\right] \ldots \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

after the prompt, and the indicated lines will be deleted. The ellipsis indicates an optional multiple occurrence of the preceding item. If the upper form is used, and the designated line number(s) does not exist, the line number(s) will be retyped preceding the error message:

NO PROGRAM

If the lower form is used, all statements whose line numbers, n, are such that

$$\text{line \#}_1 \leq n \leq \text{line \#}_2$$

will be deleted from the text area if at least one such line exists. If no statement numbers exist in the bounded area, the subsystem responds with

line #$_1$ - line #$_2$ NO PROGRAM

The printing of the NO PROGRAM error message does not inhibit continued processing of the line. The absence of such a message indicates that the specified line(s) has been deleted.

If an illegal line number is encountered, i.e., one whose format is incorrect, the error message

LINE # ERROR

is printed preceded by the characters, up to six, that make up the line number. In this event, further processing of the line is inhibited.

If the user wishes to delete the entire program, he should use the CLEAR method rather than DEL followed by line numbers or groups that give the same result, e.g., DEL 1-99999. The CLEAR provides a complete recovery of the text space in minimum CPU time, while the latter method does not clean up the text edit area and requires much more CPU time.

TEXT LISTING

The user may type

$$\text{LIS}[\text{T}]\left[\begin{matrix}\text{line\#}\\ \text{line\#}_1 - \text{line\#}_2\end{matrix}\right]\left[\begin{matrix},\text{line\#}\\ ,\text{line\#}_1 - \text{line\#}_2 \ldots\end{matrix}\right] \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

This will cause a listing of the appropriate lines from the text editing area. Error messages identical to those described under "Line Deletion" are output, as appropriate. If the user types LIST RET, this implies that he wants all lines in the text area to be listed. If none exist, the error message

NO PROGRAM

will be output.

TEXT SAVING

The user may type

$$\text{SAV}[\text{E O}]\left\{\begin{matrix}\text{N}\\ \text{VER}\end{matrix}\right\} \text{ aconst } \left[\begin{matrix}\text{line\#}\\ \text{line\#}_1 - \text{line\#}_2\end{matrix}\right]$$

$$\text{(followed by)} \quad \left[\begin{matrix},\text{line\#}\\ ,\text{line\#}_1 - \text{line\#}_2\end{matrix}\right] \ldots \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

where "aconst" is a 1-6 character string enclosed by single quotes or double quotes, or 1-6 non-blank characters followed by a blank or RET if no line numbers are given. If the SAVE ON form is used, a check will be made to see that no other file exists with the same name before opening the file for output. A file created by the SAVE ON method is declared to be a TFILE and is released when the user logs off.

The SAVE OVER option unconditionally writes the file, whether a previous file exists or not. The SAVE OVER option creates a permanent file provided there was no previously executed SAVE ON command using the same file.

Both of these commands reset the account information, if any exists.

PROGRAM LOADING

The user may type

$$\text{LOA}[\text{D}][\text{aconst}] \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

This causes the file indicated by "aconst" or the runfile (see below) if no "aconst" is specified to be opened and read completely. The processing of the information is performed exactly as previously specified for typed input under line insertion. Also, if the user level is in the execution mode just prior to typing the above line, actions equivalent to those following typing of CLEAR are performed after opening the file and prior to reading any records. If a record which constitutes an illegal line is loaded, the error message

ILLEGAL LOAD

will be output.

If no such file exists, the error message

UNABLE TO OPEN

will be output.

LOGICAL INVERSE OF LINE DELETION

The user may type

$$\text{EXT}[\text{RACT}]\left[\begin{matrix}\text{line\#}\\ \text{line\#}_1 - \text{line\#}_2\end{matrix}\right]\left[\begin{matrix},\text{line\#}\\ ,\text{line\#}_1 - \text{line\#}_2\end{matrix}\right] \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

This causes the deletion of all lines in the text editing area except for those specified by the line numbers or line number groups. If no line numbers are specified, no operation takes place and the prompt is issued.

COMPILATION AND EXECUTION MODE

While in editing mode, the user may type

$$\left\{\begin{matrix}\text{RUN}\\ \text{FAS}[\text{T}]\end{matrix}\right\} \left\{\begin{matrix}\text{RET}\\ \text{LF}\end{matrix}\right\}$$

This causes the entire text editing area to be copied onto the runfile (see below). The subsystem then enters the compilation mode, and the compiler is directed to compile from the runfile (in the safe mode, if RUN was specified; and in the fast mode otherwise).

If compilation is successful, the user level is set to execution mode; and the compiled program will be executed.

If the compilation contained errors, the subsystem re-enters the editing mode, and the prompt is issued after the text area is restored to what was put in the runfile.

If a FAST or RUN command is given in editing mode prior to the creation of any program, the subsystem will enter execution mode, and allow direct execution of statements from the console, as described below.

Should the user's BASIC program require input from the user, it will prompt the console with a question mark.

When execution is complete, the subsystem prompts with the character >. At this time, statements may be directly executed (see below) or the user may revert to editing

mode by typing an editing command.

Both of these commands reset the account information, if any exists; and, if the standard runfile is used the password information will be reset, if any has been set.

MISCELLANEOUS

ESCAPE AND PROCEED

An (ESC)(ESC) unconditionally causes the input and output buffers to be emptied and a BREAK indicator to be set. Following this, the prompt is issued. If any character is typed at this time, the BREAK indicator is reset; however, if the (ESC)(ESC) is given again before any character is typed, the subsystem will return to the BTM Executive. The ESCAPE is inhibited for the first activation temporarily during updates of the line table maintained by the Editor and while a file is being written because of SAVE requests. A second (ESC)(ESC) activation will never be inhibited.

Following the prompt, the user may type

$$\geq \text{PRO [CEED]} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

In this event, the user level will be restarted with the core (not file) conditions in effect immediately preceding the most recent ESCAPE. This service cannot be effective in all instances.

DIRECT STATEMENTS

If the user level is in the execution mode, any valid BASIC statement except for CLOSE, DATA, DEF, PUT, GET, MAT PUT, MAT GET, END, PAUSE, STOP, OPEN, DIM, Image, PRINT USING, FOR, NEXT, and any statement containing ON may be typed without a leading line number. This will indicate that the statement should be executed immediately.

When the subsystem is in the editing mode, attempted use of the direct statement capability will result in the error message

RUN? ILLEGAL

The occurrence of I/O errors, or random machine or program failure, will cause the subsystem to print

IRRECOVERABLE ERROR

and revert to editing mode with a clean text area.

FILE OPERATIONS

For each user, there exists a preset default file name consisting of the characters RUNαFIL (where α represents one of the 32 characters A, B, ..., Z, 1, 2, 3, 4, 5, 6); α will be different for each terminal, provided no more than 32 consoles are attached to the communications controller. Initially the name of the runfile referred to above is set to this seven character string. However, the user may change the name of his file by typing

$$\text{NAM [E] [aconst]} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

If an aconst appears, the name of the runfile is changed to the aconst. If no aconst appears, the default name is restored.

In all file operations (SAVE, LOAD, RUN, and FAST), file name, account number, and password logic of the Batch Processing Monitor file management system is employed. Originally, the password and account number are empty.

To establish the password or account information, the programmer types

$$\left\{\begin{matrix}\text{PAS [SWORD]}\\\text{ACC [OUNT]}\end{matrix}\right\} \text{[string]} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

where "string" extends from the first nonblank character up to seven characters preceding the New Line character. Once set, an account or password will be used in every operation that requires the opening of a file, except as specified above under "Text Saving" and "Compilation and Execution Mode". Also note that execution of an OPEN or CHAIN statement by the object program will result in modification of the account and password information.

PRECISION OF OUTPUT

If the programmer types

$$\text{ENT [ER BASIC] [L]} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

the extended precision print indicator will be set or reset, depending on whether the optional L is or is not typed, respectively.

PRINTER WIDTH

To change the width of the printer from its default value of 72, the programmer types

$$\text{WID [TH] digits} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

where the value of the digit string, interpreted as a decimal integer, must be within the allowed maximum and minimum values of 85 and 32, respectively.

STATUS

To determine the status of his program at any time, the programmer types

$$\text{STA [TUS]} \left\{\begin{matrix}(\text{RET})\\(\text{LF})\end{matrix}\right\}$$

The system will respond with one of three messages: EDITING, COMPILING, or RUNNING, as appropriate. To facilitate loop detection, the RUNNING message will be preceded by an appropriate line number if execution was

interrupted while one of the instructions in the loop was being executed.

SIGN OFF

To exit from BASIC, the user should depress the ESC key four times and wait for the Executive level prompt (!). At this point, he may invoke another processor (e.g., FERRET, to allow copying of TFILE's to permanent status), or he may vacate the terminal by typing

! BYE

MODE SWITCHING

To enter execution mode, the user must execute a RUN command or FAST command which produces no diagnosis at compilation time. Execution time diagnostics or break activations will be followed by a > prompt while still in the execution mode.

The user may switch to editing mode by typing (after a > prompt has been issued) a non-direct BASIC line or any command that results in line deletion, text listing, text saving, program loading, or line extraction.

To enter the desk calculator mode, the user may type CLEAR followed by RUN or FAST.

BATCH CONTROL AND OPERATION

To operate a BASIC batch job, the following setup instructions should be observed:

1. The first record in the M:C file must be a Monitor !JOB card.

2. Optionally, other batch control cards (e.g., !LIMIT, !STDLB, !MESSAGE, !TITLE, !ASSIGN, etc.) may follow in the M:C file. In particular, since any programs that are to be invoked by execution of a CHAIN statement must exist as named sequential files on the disc, it will be natural to accomplish this setup by use of the FMGE system at this point.

3. Next, the processor control card !BASIC must appear in the M:C file.

4. Optionally, the next (first) card in the M:SI file may be an option declaration card. This card must have an * character in column 1, and column 2 must be blank. Column 3 may contain the character S, indicating the safe mode for array references, or any other alphabetic character, which will cause the fast mode for array references. Column 4 may contain the character D, indicating default printer line width which is an assembly parameter (currently 100), or column 4 may contain a digit. If neither a digit nor the character D is in column 4, further processing of the card is inhibited. If a digit appears in column 4, successive columns of the card are scanned until a non-digit character is encountered. The digits are interpreted as a decimal integer specifying the printer line width subject to maximum and minimum width assembly parameters (currently 131 and 32 respectively). The character following the D or the last digit of the digit string is ignored and may be any character. If the next character is an L, the contents of the M:SI file will be listed on the M:LO file. This listing will contain all records that follow the option declaration card until an end of file or a !EOD record is encountered, or a card with the characters *RUN in columns 1-4 is encountered. The !EOD will not be listed, but the *RUN will.

5. The next cards in the M:SI file must constitute a BASIC program, i.e., a set of BASIC lines in line number order.

6. If the BASIC program is followed by an !EOD record or a card with the characters *RUN in columns 1 through 4, and if the compiler detects no errors, the resultant object program will be executed starting with the statement with the lowest line number.

7. Optionally, the next records in the M:SI file may contain information to be processed in response to execution of INPUT statements. The end of this information is indicated by an end of file, !EOD record, or a record with the character * in byte 1. Attempting to input when no more information exists will result in termination of execution with the OUT OF DATA diagnostic message.

8. The next cards in the M:C file may be other Batch control cards (e.g., !ASSIGN, !FMGE, etc.). In particular, since all files created by execution of OPEN statements with the O option are declared to be temporary files (M:TFILE), the user must convert such files to permanent files if he wishes to retain the information contained.

6. TERMINAL BATCH ENTRY (BPM) SUBSYSTEM

The Terminal Batch Entry (BPM) subsystem controls insertion of jobs into the batch job queue, and is available only when BTM is operating in a symbiont BPM environment.

The Executive command for this subsystem is

!BPM

The subsystem will transfer a file of control and data records from M:SI to the batch job stream. Normally, this file will be created at the console so that M:SI need not be assigned; default assignment is the user console. However, if a job is to be repeatedly executed, the control records themselves may be made up as a permanent disc file by using the EDIT subsystem.

The control records are formatted exactly as the control cards for a normal batch job. In general, any sequence of control and/or data records that will execute properly when submitted by normal means through a single device may be submitted through the console. A FIN control command, however, will be rejected. BIN control commands will be ignored, and should not be used.

A job inserted from a terminal must be set up to run under the terminal's log-in account. That is, the account number on the !JOB card must be the same as the account the user has logged in with.

When the job is inserted, the job ID, priority, and COC line number of the originating console are printed on the operator's log.

Output files from background jobs may not be assigned directly to Teletypes, and input will normally come from disc or tape.

MESSAGE Monitor control commands may be used to convey any unusual operating procedures to the operator.

Immediately after entry, the BPM subsystem inquires

INSERT JOB? {Y / N}

An affirmative reply (Y) causes the subsystem to prepare to enter a job file, as described in the following sections on job file creation. A negative reply (N) causes the subsystem to go immediately to the status checking portion of the program.

JOB FILE CREATION

Input to the BPM subsystem can be from the console or from an existing disc file. In either case, the subsystem will allow some editing of the job file. These editing functions are intended only to provide recovery from errors of omission committed while composing a small job file at the console. It is strongly recommended that the user prepare large job files through use of the EDIT subsystem (see Chapter 7) prior to insertion by the BPM subsystem.

CONSOLE INPUT

The BPM subsystem prompts each record with a record number. Therefore, the user creates a dialogue much like the following when giving commands from the console:

```
!BPM
1
:record no. 1 (for example, !JOB 85025, BPM, 1) (RET)
2
:record no. 2 (RET)
.
.
.
n
:record no. n (RET)
n+1
: (RET) (A null record terminates input.)
```

The subsystem then inquires

EDIT? {Y / N}

If the reply is negative, the file is transferred to the batch job queue and the subsystem types the message

JOB INSERTED. ID = xxxx

An affirmative reply allows the user to modify his job file prior to batch insertion.

DISC FILE INPUT

After BPM is called, it will respond with the following request:

DISPLAY FOR EDITING? {Y / N}

The user responds with Y or N. If the reply is negative (N), no further communication between user and subsystem takes place. The file is transferred to batch and the subsystem types the message

JOB INSERTED. ID = xxxx

The file input process may be interrupted with the (ESC) key, which will return the user to the Executive level. The user may then restart the process, or enter another subsystem.

Should the user reply Y, the contents of the file will be read and displayed in the following format:

```
!BPM
DISPLAY FOR EDITING? Y
1
record no. 1
2
```

record no. 2
:
n
record no. n

When the file is completely displayed, the subsystem prompts:

EDIT? {Y / N}

As before, a negative reply causes job insertion. An affirmative reply allows the changes described in the following section.

JOB FILE EDITING

Prior to insertion of the file, the following functions may be performed:

1. Deleting any record or sequence of records.
2. Interchanging any two records.
3. Inserting a sequence of records.
4. Replacing a sequence of records.
5. Retyping any record or sequence of records.

When the user responds affirmatively to a request to edit the filw, the subsystem prompts with a question mark:

EDIT? Y
?

At this point, the user may respond as follows, where xx and yy are record numbers as displayed on the console.

Function	Command
Replace xx through yy	? Rxx, yy (RET)
Swap xx and yy	? Dxx, yy (RET)
Append after xx	? Sxx, yy (RET)
Terminate update, insert job file	? G (RET)
Type file	? T (RET)
Type xx through yy	? Txx, yy (RET)
Abort the task	? X (RET)

When a request is made to append or replace records, the program prompts with the next record number it expects.

For Example:

10
: (RET)
EDIT? Y
?A4
5
:

Records are inserted until a null record is supplied.

After insertion or deletion of records, subsequent requests must take into account the number of records added or taken away. Edit requests are syntax checked prior to any action, and rejected if erroneous; however, the change to the job file is made immediately thereafter.

STATUS CHECKING

After any job has been inserted, the subsystem inquires:

STATUS CHECK? {Y / N}

If the reply is negative, the subsystem returns to the Executive level. Otherwise, the program inquires:

ID = xxxx (RET)

The user responds with the four hexadecimal digits of a previously inserted job. If fewer than four digits are entered, leading zeros are supplied. If only a carriage return is entered, the subsystem returns to the Executive level.

After receiving a valid job ID, the program will print one of the following three status indicators:

WAITING,
RUNNING,
COMPLETED.

The program then issues another ID request. It should be noted that if a hexadecimal number is supplied that does not correspond to a job ID for the current symbiont run, the status will be as though a valid job were complete. There will be no error indication; therefore, the user must be careful to supply the proper ID.

ERROR CONDITIONS

Error conditions are indicated by the following messages.

```
MISSING !JOB COMMAND, OR RECORDS
OUT OF ORDER.
EDIT?
```

A JOB command must be the first record transferred. After this message, the user should abort the task or supply the necessary JOB command.

```
IMPROPER JOB PRIORITY; LEVEL 1 ASSIGNED
```

The JOB command may not have statement continuation. If it contains a priority field, the priority must be expressed as a single character ranging from 0-F. If the JOB command does not contain a priority field, level 1 priority will be assigned, with no message to the user.

```
!FIN CARD IGNORED
```

The job file contains a FIN command.

BAD I-O. ABNORMAL CODE - xx

An I/O error or malfunction has been detected. The system is returned to the Executive level.

FILE TOO LARGE. TASK ABORTED

The number of records that may be transmitted is limited only by user memory size, at 20 words per record. In a 16K user area, this would amount to approximately 700 records.

NO INPUT DATA. BYE

The file to be transferred contains no input data. The system is returned to the Executive level.

NONEXISTENT LINE

An edit command references a nonexistent line.

ERRONEOUS COMMAND IGNORED

An edit command contains improper parameters.

ILLEGAL CAL

If the BPM subsystem is used with a non-symbiont monitor, the insertion of the job will abort with an erroneous CAL3 notification.

INVALID ID

An erroneous job ID has been supplied. A job ID must be a hexadecimal number followed by a blank or carriage return.

IMPROPER JOB ACCOUNT

The job account does not match the log-in account.

AUTHORIZATION REQUIRED

The user is not authorized to enter a job from the terminal.

7. EDIT SUBSYSTEM

INTRODUCTION

The EDIT subsystem allows the user to create or modify disc resident source files. The user has the ability to:

1. Create a sequenced source file.
2. Delete a record or sequence of records from an existing file.
3. Insert a record or sequence of records into an existing file.
4. Replace a record or sequence of records in an existing file with a new set of records.
5. Reorder groups of records within a file.
6. Perform intra-record character string substitution and manipulation.
7. Copy a specified file.

A summary of commands by function is included at the end of this chapter.

The Executive command for the subsystem is

! EDIT

The subsystem then prompts command input with the "*" character.

RECORD FORMATS

The editing process is based on a sequence number associated with each record. Such sequence numbers may be automatically generated by the Editor if not initially present on the file.

Each source record is assumed to contain an EBCDIC sequence number in columns 74-80. For purposes of editing, this sequence number has an implied decimal point between columns 77-78. Thus, if columns 74-80 contain 1234567, the sequence number is expressed as 1234.567.

In later examples, any reference to a sequence number without a decimal point implies three trailing zeros. Any sequence number with a decimal point is assumed to have sufficient trailing zeros to make up three fractional digits. For example:

Sequence Number	Implies
50	50.000
50.01	50.010
50.5	50.500
50.008	50.008

Edit files are kept on disc with keyed organization, the keys being derived from the above sequence number. Since the Editor uses a keyed file structure, files introduced by BPM FMGE commands will have to be copied by the Editor before any editing may be performed. The user will be notified if this is necessary.

In using the Editor, it mast be stressed that the edit takes place as the commands are given; the file is edited in place. Therefore, a backup file should always be kept.

COMMAND STRUCTURE

EDIT commands fall into the following three catagories:

1. File oriented commands that may be given at any time.
2. Record editing commands that may only be given after a file has been selected for editing.
3. Intra-record (usually character string) editing commands that may only be given after a specific set of records has been selected by a command of type 2, above.

In those commands that select files for editing, copying, etc., the input and output files are identified by one of the following constructs, called a "fid" or file identifier.

file-name	Default log-in account
file-name(account)	Specific account
file-name(account, password)	Account and password
file-name(,password)	Password only

However, in any edit operation that writes into the specified file, the file must be under the log-in account. The account specification is primarily of value for copying a file from a differenct account. The user is only able to delete files in his log-in account.

For example:

* DELETE BPMS (,MK)

The command shown above specifies that the file "BPMS", having the password "MK", is to be deleted from the user's log-in account.

Commands that reference character strings within a record require that the user identify the string by delimiting it with the "/" character. For example, the character string A+2 is made into a string identifier by typing /A+2). A single / may be included in the string by typing two slashes in succession. That is, /A//2/ identifies the string A/2.

MESSAGES

During the course of executing any command, the Editor may communicate with the user through a variety of

messages. Possible messages are summarized with the description of each edit command.

The following conventions are used in regard to message formats:

1. A message preceded by two periods is a comment on some system-oriented operation. For example:

   ..COPY DONE

2. A message preceded by two minus signs indicates the occurrence of some event (during the execution of a command) of which the user should be aware; the command is not aborted, however. For example:

   --EOF HIT

3. A message preceded by a single minus sign is an error message describing a condition which aborts the current command and causes any others on the same line to be skipped.

   Such a message may be particularized as to cause, by the following prefixes:

Prefix	Cause of Error
$-C_k$:	The kth command of the previous line caused the error.
$-P_k$:	The kth parameter of the first command on the previous line caused the error.
$-C_kP_j$:	The jth parameter of the kth command of the previous line caused the error.

   For example:

   -P1: NO SUCH REC

In addition to the particular errors discussed with commands, a large number of self-explanatory syntax messages are given when errors occur.

FILE COMMANDS

BUILD (Build new file)

BUILD causes the Editor to create a new file on disc. The BUILD command has the form shown below.

*_ BUILD fid [,n[,i]]

where

fid is the identifier (see "Command Structure", above) of the file that is to be created.

n is the sequence number at which the new file is to start. The default value is 1.

i is the value by which sequence numbers for the new file are to be incremented. The default value is 1.

The system prompts by typing a sequence number; the user then types in the corresponding line. A null line terminates the build operation and closes the file. Files created with BUILD are automatically permanent.

Example:

```
*_ BUILD SOFILE
1.000          SYSTEM     SIG5 (RET)
2.000          DEF        B (RET)
3.000          REF        A (RET)
4.000B         B          A (RET)
5.000          END (RET)
6.000 (RET)    The null record, consisting of only
*_ (RET)       a carriage return, terminates the
               command and does not appear in
               the output file.
```

Comments:

..EDIT STOPPED — An EDIT was active when the BUILD command was given and has been terminated.

Errors:

--OVERFLOW — More than 72 nonblank characters were entered.

-FILE EXISTS: CAN'T BUILD — A file with the same fid already exists.

END (Exit)

END causes the Editor to close all active files and return control to the Executive. The END command has the form shown below.

*_ END

COPY (Copy file)

Copy causes the Editor to copy a specified file. The COPY command has the format shown below.

*_ COPY fid_1 {ON / OVER} fid_2 [,n[,i]]

where

fid_1 identifies the file that is to be copied.

fid_2 identifies the file to which the file identified by fid_1 is to be copied.

n see "BUILD" above.

i see "BUILD" above.

If ON is specified, a new file is created (and must not already exist). If OVER is specified, fid_2 may exist and if it

does it will be deleted and replaced by the copy of fid_1. If n is omitted, the old sequence numbers on fid_1 are retained in the copy. If n is present, fid_2 is resequenced starting at n and incrementing by i.

Comments:

..EDIT STOPPED	An edit was active when COPY was given and has been terminated.
..COPYING	The COPY has been started.
..COPY DONE	The COPY is done.

Errors:

-P2: FILE EXISTS	A COPY ONE has been given but fid_2 exists.
-P1: NO SUCH FILE	The file identified by fid_1 does not exist.
-P1: FILE NOT SEQD & P3 NULL	There are no sequence numbers on the file identified by fid_2 and resequencing has not been specified; thus, if copied, the resultant file could not be edited.

DELETE (Delete file)

DELETE causes the Editor to delete a specified file. The DELETE command has the format shown below.

*DELETE fid

Comments:

..DELETED	The specified file has been deleted.
..EDIT STOPPED	An edit file was active when the DELETE command was given.

Error:

-NO SUCH FILE	The specified file does not exist.

EDIT (Edit file)

EDIT specifies that a specified file is to be edited. The EDIT command has the format shown below.

*EDIT fid

This command must be given before any record oriented commands can be executed. The effect of the EDIT command is terminated whenever a BUILD, DELETE, or COPY is given. Giving an EDIT command while a previous EDIT is active will cause the previous file to be closed and editing to be begun on the new file.

Errors:

-NO SUCH FILE	The specified file does not exist.
-FILE NOT KEYED; MUST COPY	The file is not in the correct keyed format needed by the Editor and must first be copied before it can be edited.

BP (Set blank preservation mode)

BP sets the blank preservation mode on or off. The BP command has the format shown below.

*BP [{ON / OFF}]

When "on", all strings of blanks are preserved during intra-record operations. When "off", blank strings are compressed to a single blank or expanded as required to retain column alignment of nonblank fields. As discussed in "Intra-record Operations", below, a group of commands is provided that allows substitution and insertion of character strings within a record. This can be of great value, for example, when correcting a complicated FORTRAN expression or changing the name of a variable in a program.

However, when a string is inserted or replaced in a manner that changes the number of characters in a record, a problem arises in how to adjust the record format.

When the blank preservation mode is off, any string operation that expands or contracts a nonblank string causes the blank count between the target string and the next nonblank string to the right of the image to be increased or decreased as necessary to preserve the columnar position of the righthand string. Nonblank strings are never connected, however; one blank will always be left between strings.

When the blank preservation mode is on, the blank counts are preserved, as one would wish in a FORMAT or TEXT statement.

For example, the following string substitution command (these commands are discussed later in this chapter)

*/8/S/LINK/

substitutes the string "LINK" for the string "8" in the instruction

```
$10 BAL,8    SUB
```

adjusting blanks as indicated below:

old	$10 BAL,8 SUB
BP-OFF, new	$10 BAL,LINK SUB
BP-ON, new	$10 BAL,LINK SUB

Errors:

-NOT ON/OFF	A parameter other than ON or OFF was specified (the default is OFF).

RECORD EDITING COMMANDS

IN (Insert new records)

IN causes the Editor to insert new records into a file. The IN command has the format shown below.

```
*IN n[,i]
```

New records are inserted starting at the record with sequence number n, with each successive record being sequenced from n with increment i. If a record with sequence number n exists in the file, it is replaced by the newly inserted record n.

The Editor will prompt the user console with the first sequence to be inserted, and repeats the requests for each subsequent insertion, increasing the sequence number by the increment i. (If i is omitted, the increment size specified in the most recent record editing command is used. If no such commands have been given, the value 1 is assumed by default.)

The insertion can be terminated in either of two ways. If a null record is supplied, the insertion terminates. An equivalent action takes place if an incremented sequence equals or exceeds a sequence existing in file. In the latter case, the console bell is rung.

For example:

```
*IN 100.,1
100.000   10   A=2.5      Replaces the existing
                          record.
100.100        B=0.
*                         Record insertion terminates
                          because sequence number
                          100.200 existed previously;
                          the console bell is rung.
```

Errors:

--OVERFLOW	More than 72 nonblank characters were typed.

The user wishing to insert records may use a variant of this command by typing IS. The action is equivalent to IN except the Editor does not prompt with sequence numbers.

For example:

```
*IS 100,.1
10  A=2.5
    B=0.
*
```

TY (Type records)

TY causes the Editor to type each record lying within a specified range of sequence numbers, together with its sequence number. The TY command has the form shown below.

```
*TY n [-m]
```

If m is omitted, only n will be typed. If the entire record will not fit on the same line with the sequence number, the sequence number is typed first on a separate line. Once the editor shifts to two-line format it will continue in that mode for the remainder of the range of the TY command.

Error:

--EOF HIT	The range m-n passes beyond the end-of-file.

TS (Type records, suppressing sequence number)

TS causes the Editor to type each record in a specified range, but without an accompanying sequence number. The TS command has the form shown below.

```
*TS n [-m]
```

Error:

--EOF HIT	The range n-m passes beyond the end-of-file.

DE (Delete records)

DE causes the Editor to delete all records whose sequence numbers lie in a specified range. The DE command has the form shown below.

```
*DE n [-m]
```

If m is omitted, only record n is deleted.

For example:

* DE 50	Deletes record 50.000 only.
* DE 50-60.5	Deletes all records in the range 50.000 through 60.500, inclusive.

Comments:

--NOTHING TO DE	No records were found in the specified range.

Error:

--EOF HIT	The range n-m passes beyond the end-of-file.

FD (Find and delete)

FD causes the Editor to search for a specified string between specified columns. If the string is found, the record is deleted from the file. The FD command has the form shown below.

*FD n[-m],/string/[,c[,d]]

where

n specifies the sequence number of the first record to be searched.

m specifies the sequence number of the last record to be searched. If omitted, only record n is searched.

/string/ specifies the character string identifying the record to be deleted.

c specifies the lower limit (i.e., column number) of the field to be searched. The default value is 1.

d specifies the upper limit of the field to be searched. The default value is 72.

The specified string must be entirely contained within columns c through d to cause deletion. At the end of this operation, a message is printed telling how many records were deleted.

Example:

*FD 5-20.4,/DATA/,10,18

--006 RECS DLTED

Comments:

--NONE — There were no records (in the specified range) containing the indicated string.

Errors:

--EOF HIT — The specified range passes beyond the end-of-file.

FT (Find and type)

FT causes the Editor to search for a specified string between specified columns. If the string is found, the Editor types out the sequence number of the record. The FT command has the form shown below.

*FT n[-m],/string/[,c[,d]]

The specified string must be entirely contained within columns c through d (see "FD" above).

Example:

*FT 1-100,/LW/,10

5.000

9.000

21.480

73.000

*

Comments:

--NONE — There were no records in the specified range containing the indicated string.

Errors:

--EOF HIT — The specified range passes beyond the end-of-file.

MD (Move and delete records)

MD causes the Editor to delete all records in a specified range and to then move records in another range into this area. The MD command has the form shown below.

*MD n[-m],k[-p][,i]

where

n specifies the sequence number of the first record that is to be moved.

m specifies the sequence number of the last record that is to be moved. If omitted, n only is moved.

k specifies the lower limit (i.e., sequence number) of the range of records to be deleted.

p specifies the upper limit of the range of records to be deleted. If omitted, k only is deleted.

i specifies the increment value to be used for renumbering records. If omitted, the most recent increment value specified in a record edit command is used (or 1, if no increment value has been specified).

The first record (n) is renumbered as k. Successive records from the range n-m are renumbered consecutively higher, with increment i.

As each record from the range n-m is moved, it is deleted from the original range (n-m). At the end of this operation, a message is printed out specifying the new sequence number of the last record moved from the range n-m.

Example:

*MD 5-21, 100-101, .02

--DONE AT 100.28

If the increment is too large to permit all records in the range n-m to be moved into the space between k and the next record after p, a message is printed out specifying the sequence numbers, from both ranges, of the last record moved.

In this case the original contents of range k-p will be lost, but only those records in the range n-m which have actually been moved will have been deleted. Thus the user can perform another move (with a smaller increment) to move the remaining records in the range n-m.

Example:

```
*MD  10-30, 100-110, 1

--CUTOFF AT 110.  (20.)
```

The ranges n-m and k-p may not overlap.

Errors:

--NOTHING TO MOVE	No records exist in the range n-m.
--RNG OVERLAP	Ranges n-m and k-p overlap.
--EOF HIT	Range n-m passes beyond the end-of-file.

MK (Move and keep records)

MK acts similarly to MD except that the records in the range n-m are not deleted as they are moved, rather a second copy of the range n-m is made.

The MK command has the form shown below.

```
*MK  n [-m], k [-p] [,i]
```

RN (Renumber record)

RN causes the Editor to renumber specified records. The RN command has the form shown below.

```
*RN n,k
```

This has the same effect as deleting record n and then entering a new record with sequence number k with the same contents as n. Sequence number k must not already exist.

Errors:

-P1: NO SUCH REC	Record n does not exist.
-P2: REC EXISTS	Record k already exists.

CM (Commentary)

CM causes the Editor to insert commentary into specified columns of each successive record beginning at a specified sequence number. The CM command has the format shown below.

```
*CM n,c
```

where

n is the record number.

c is the column number.

The sequence number of each record is typed and then the user types in the data he wants inserted starting at column c. The data he types in is blank filled to the right through column 72, as required. A null record terminates the command. It is not necessary to delimit commentary with slashes.

Example:

```
*CM 37.6,  40 (RET)

37.600  COMMENT 1 (RET)

37.800  COMMENT 2 (RET)

40.500 (RET)
```

Errors:

-P2: COL>72	Column c > 72.
-P1: NO SUCH REC	Record n does not exist.
--EOF HIT	The end-of-file has been encountered.
--OVERFLOW	Commentary typed in has overflowed past column 72 with non-blank characters.

SS (Set and step)

SS causes the Editor to start at a specified record and proceed to each record in succession, accepting one line of intra-record commands to update the current record. The SS command has the format shown below.

```
*SS  n [,c [,d]]
```

Intra-record commands will only be effective on strings that lie wholly within columns c through d.

The Editor prompts commands for each successive record with the sequence number, followed by a double asterisk.

The SS command is terminated by typing a null record in place of an intra-record command.

Example (to interpret this example in detail, see the commands discussed in "Intra-Record Operations", below):

```
*SS  300.5,  37 (RET)

300.500**/BSD/S/BAD/ (RET)

301.000**37E/X/ (RET)

303.450** (RET)
```

Errors:

-P1: NO SUCH REC	Record n does not exist.
--EOF HIT	The end-of-file was encountered.
-Cn: COMND ILGL HERE	The nth command of the input line is not an intra-record command; the "set and step" mode is terminated.

ST (Set, step, and type record)

This command is similar to SS except that the contents of each record are typed, along with its sequence number, prior to accepting a command. The ST command has the format shown below.

*ST n [,c[,d]]

Example:

```
*ST 50 (RET)
50.000 XQ1          LW,5          AGRD,6
** NO (RET)
51.000              STW,5         *KX9
**/X/F/1/;TS (RET)
        STW,5       *KX19
52.000              MW,8          BQ
** (RET)
```

INTRA-RECORD OPERATIONS

Commands in the intra-record group may be concatenated on a single line through use of the ";" character. The following command sequence would select a line, type the original, edit, and type the new version:

*SE 100 ; TY ; /TEMP/S/B/;/JK/F/+BETA/;TY (RET)

Prior to performing any of the operations described in this section, the user must either give an SS or ST command (which provide for step mode modification of successive images) or the following SE command which allows repeated modification of the same range of images.

SE (Set intra-record mode)

SE causes the Editor to accept successive lines of intra-record commands. The SE command has the format shown below.

*SE n [-m] [,c[,d]]

Commands input will be applied, in order, to each record in the range n through m. If several commands are entered on one line, all commands on the line are executed on one record before the next record is processed. The first occurrence of a file or record oriented command (e.g., IN) terminates the effect of the SE command.

All commands executed in the intra-record mode apply only to the strings lying entirely within columns c through d. Each new input line of commands will be applied to the whole range (n-m).

SE may be used on the same input line with other intra-record commands, but when so used it must be the first command on the line.

Errors:

--EOF HIT	The range n-m passes beyond the end-of-file.
-P1: NO SUCH REC	Record n does not exit.

The following conventions are used with intra-record commands:

1. j/string/x

   means that command x is to operate on the jth occurrence of the indicated string found between columns c through d as specified by an SE, SS, or ST command. If j = 0 this means that the command x is to operate on all occurrences of the string between columns c and d. If j is missing, the default is 1.

2. k x

   means that command x is to operate on the character contained at column k, where k must lie between columns c and d of the SE, SS, or ST command.

The following general errors are possible:

-MISSING SE	No SE command was given.
--Cn: NO SUCH STRG	The string referred to by the nth command of the input line does not exist between columns c and d. When the SE command operates on a range of lines, this message will be given once, should the condition occur at any time during scanning of the range.
--Cn: COL>LIMIT	The value specified for k is greater than d for the nth command.
--Cn: COL<LIMIT	The value specified for k is less than c for the nth command.

In any intra-record command that seeks a matching string in the image, only those strings that lie totally within the specified column bounds will be found. Partial matches to a column boundary will be ignored. In subsequent examples, references to columns c and d pertain to the column boundaries given in the SE, SS, or ST command.

S (String substitution)

S causes the Editor to locate a given occurrence of a specified string between columns specified by an SE, SS, or ST command and replace it with another string. The S command has the format shown below.

*[j]/string$_1$/S/string$_2$/

The image to the right of string$_1$ is adjusted right or left as required, if the lengths of string$_1$ and string$_2$ differ. String$_2$ may extend past column d if d < 72.

If j = 0 is used, all occurrences of string$_1$ between columns c and d are replaced by string$_2$; otherwise, only the jth occurrence is replaced.

Error:

--Cn: OVERFLOW	Nonblank characters have been pushed off the right end of the card image by the action of the nth command.

For example:

Command	Result			
*/LW/S/CW		LW,R5	ALPHA+2	old
		CW,R5	ALPHA+2	new
*/10/S/5/		LW,R10	B	old
		LW,R5	B	new
*/$10/S/ENTRY	$10	LW,R5	ALPHA	old
	ENTRY	LW,R5	ALPHA	new
*/ALPHA/S/B/		LW,R5	ALPHA+2,R6	old
		LW,R5	B+2,R6	new
*2/5/S/55/	15	C=DSQRT(TEMP+ 2.5*BASE)		old
	15	C=DSQRT(TEMP+ 2.55*BASE)		new

D (Delete string)

D causes the Editor to locate a given occurrence of an indicated string, between columns specified by an SE, SS, or ST command, and delete it. The D command has the format shown below.

*[j]/string/D

If j = 0, all occurrences between c and d are deleted.

E (Overwrite and extend blanks)

E causes the Editor to start at the column occupied by the 1st character of a given occurrence of a specified string or column and overwrite with another string. The E command has the format shown below.

*[j]/string$_1$/E/string$_2$/

or

*kE/string$_2$/

Blanks are extended from the end of string$_2$ through column d. String$_2$ may overwrite beyond column d if d < 72, but blank extension only occurs through column d.

Error:

--Cn: 'ALL' IGNORED	The specification j = 0 was used, but since it was not meaningful for E, j = 1 was substituted.
--Cn: OVERFLOW	String$_2$ overflowed past column 72 with nonblank characters.

O (Overwrite)

O causes the Editor to start at the column occupied by the first character of a given occurrence of a specified string (or column) and overwrite with another string. No blank preservation or other adjustment is done and all columns not overwritten remain unchanged. The O command has the form shown below.

*[j]/string$_1$/O/string$_2$/

or

*kO/string$_2$/

String$_2$ may overwrite beyond column d if d < 72. If j = 0 all occurrences of string$_1$ between columns c and d are overwritten, but string$_2$ is not scanned while searching for occurrences of the first string.

Error:

--Cn: OVERFLOW	Nonblank characters have overflowed beyond column 72.

P (Precede by)

P causes the Editor to start before the first character of a given occurrence of a specified string (or column) and insert another string, pushing characters of the first string to the right as required to make room. The P command has the form shown below.

*[j]/string$_1$/P/string$_2$/

or

*kP/string$_2$/

String$_2$ may legally extend beyond column d if d < 72. The first character of string$_2$ will occupy the column vacated by the first character of string$_1$, etc.

If j = 0, the Editor will insert string$_2$ before all occurrences of string$_1$ between columns c and d. However, after string$_1$ has been found once and string$_2$ inserted before it, scanning for the next occurrence resumes at the next character after string$_1$, as adjusted by the insertion.

Example:

```
_* SE 17.69 (RET)
_* TS;0/AA/P/./;TS (RET)
AAAAAAA
.AA.AA.AAA
```

Errors:

--Cn: OVERFLOW	A nonblank character has been pushed off the right of the card image.

F (Follow by)

F causes the Editor to start after the last character of a given occurrence of a specified string (or column) and insert another string, pushing everything from this column right as required to make room. The F command has the format shown below:

_*[j]/string$_1$/F/string$_2$

or

_*kF/string$_2$/

String$_2$ may legally extend past column d if d < 72. String$_2$ is not scanned when searching for all occurrences of the first string. Furthermore, insertion is made from left to right, so that an occurrence of string$_1$ which was between columns c and d before F was executed may no longer be in this range by the time it is scanned, if previous insertions have taken place.

For example:

Command	Result		
_*/AB/F/+2/	LW,R6	AB,R2	old
	LW,R6	AB+2,R2	new

Errors:

--Cn: OVERFLOW	Nonblank characters were pushed off the right-hand end of the record image.

R and L (Image shifting)

R and L commands cause portions of the image to be shifted right (R) or left (L). The R and L commands have the form shown below.

_*[j]/string/ {R / L} s

or

_*k {R / L} s

SHIFT LEFT

The nonblank field to the right of the first character of the jth occurrence of the indicated string (or column k) is shifted left s positions. Blanks are supplied on the right, and columns to the left are overwritten. The shift may legally overwrite below column c.

SHIFT RIGHT

The nonblank field to the right of the first character of the indicated string is shifted right s positions. Blanks are inserted into vacated positions and absorbed on the right if blank preservation is OFF. The shift may legally push characters beyond column d, if d is less than 72.

For example:

Command	Result			
_*/L/R2	$10	LW,R6	B	old
	$10	LW,R6	B	new
_*/L/R10	$10	LW,R6	B	old
	$10		LW,R6B	new
_*/L/L2	$10	LW,R6	B	old
	$10	LW,R6	B	new

Errors:

--Cn: 'ALL' IGNORED	The value j=0 was specified, but j=1 was substituted.
--Cn: UNDERFLOW	Characters were lost to the left of the record.
--Cn: OVERFLOW	Characters were lost to the right of the record.

TS (Type, suppressing sequence number)

TS causes the Editor to type the contents of the current active line of the SE, SS, or ST command.

The TS command has the form shown below.

_*[...;]TS[;...]

Example:

```
* SE 5 ; TS ↵
L1    LW,5     K
* 19O/KLB/ ; TS; 37O/GET KLB/ ; TS ↵
L1    LW,5     KLB
L1    LW,5     KLB          GET KLB
```

Because all commands on a single input line are executed for the first record before the second record is processed, etc., TS will type each line in turn after all editing up to the TS command has been done.

Example:

```
* SE  10-10.2 ↵
* 2/A/F/,4/;TS ↵
      DATA,4    X'FF'          (10.0)
      DATA,4    0,5            (10.1)
      DATA,4    GQX,X'0B'      (10.2)
```

TY (Type, including sequence number)

TY is the same as TS, except that each line is printed with its sequence number.

JU (Jump)

JU may only be used while in the "step" mode (i.e., while under the control of an SS or ST command). JU causes the SS or ST command to jump to a specified record and then continue stepping from that point. The JU command has the form shown below.

```
** [...,]    JUn
```

Record n may be forward or backward from the current sequence number at the time JU is given. JU may be used on compound lines (i.e., a line with more than one command on it), but in such a case JU must be the last command on the line.

Error:

```
--Cn: NO SUCH REC     Record n does not exist.
```

NO (No change)

NO may be used only while in the "step" mode and specifies that no editing is desired on the current active line under the set. The NO command has the form shown below.

```
** NO
```

Example:

```
* ST 27.5 ↵
27.500          LW,6          BLK
** NO ↵
30.000          STW,6         ALT
** /ALT/F/+/19/ ; TY ; JU34 ↵
30.000          STW,6         ALT+19
34.000          AI,F          X'91'
**
```

RF (Reverse blank preservation flag)

RF causes the current setting of the blank preservation flag ("on" or "off") to be reversed temporarily. The RF command has the form shown below.

```
*[...;] RF ;...
```

or

```
* ...; RF [;...]
```

The flag is reversed only for the duration of the input line in which RF appears, and blank preservation is restored to its initial setting when a new input line is entered (i.e., at the time a new prompt character is given). Thus, RF must always be used as part of a compound input line to have any effect.

Example:

```
* SE  10; TY
10.000 L5    LW,4 X GET CURRENMT     ADDR
* RF;/NM/S/N/;TY
10.000 L5    LW,4 X GET CURRENT      ADDR
```

Without using RF in this case (assuming that BP OFF is the initial setting), one would get two blanks after CURRENT. In all cases, the BP flag is restored to the value it had before any RF commands were given.

EDIT COMMAND SUMMARY

A summary of EDIT commands is given below.

Command	Page	Function
BP	20	Select character insertion mode
BUILD	19	Create new file
COPY	19	Copy file 1 to file 2
DELETE	20	Delete file
EDIT	20	Select file for editing

Command	Page	Function
END	19	Exit to Executive
CM	23	Insert commentary
DE	21	Delete records
FD	22	Delete records containing specified character string
FT	22	List sequence numbers of specified character string
IN (also IS)	21	Insert records
MD (also MK)	22, 23	Reorder records within file
RN	23	Renumber record
ST (also SS)	24	Perform character operations on records in step mode
TY (also TS)	21	Type individual records
SE	24	Perform character operations on group of records
R or L	26	Shift string
S	25	Substitute string
D	25	Delete string
O (also E)	25	Alter string
F, P	25, 26	Insert string

8. FERRET SUBSYSTEM

FERRET is a utility subsystem providing a general capability for obtaining information about entries in the file management system. The subsystem also provides for limited file manipulation.

FERRET is called with the Executive command

!FERRET

and when ready to accept input, prompts with the ">" character.

Any FERRET commands may then be given. If the subsystem does not recognize a command, it responds with

COMMAND NOT LEGAL.

The user may exit to the Executive by typing an X, i.e.,

>X
!

At any time, (ESC)(ESC) may be used to exit to the Executive level. The PROCEED command, at that point, will continue the previous operation.

FERRET COMMANDS

LIST (List account contents)

LIST causes the program to list all file names in the specified account. If no account number is specified, the log-on account is used. The LIST command has the form shown below.

>L[IST] [acct.] (RET)

TEST (Test file accessibility)

TEST checks whether the user may read the specified file(s) under his log-in account.

The TEST command has the format shown below.

>T[EST]file[,file,...,file] (RET)

where

file may have any of the three following formats:

1.	name	User's acct. assumed.
2.	name (acct.)	Specified acct.
3.	name (acct., pass.)	Specified acct. and pass.

If the file is accessible, the following message is printed:

file name WAS CREATED month,day,year,AND HAS xxxx GRANULES IN IT.

If the file does not exist or cannot be accessed under the user's account or the password specified, the following message is given:

CANNOT ACCESS FILE file

ACTIVITY (Check file activity)

The file specification formats are as in "TEST", above. These files are checked for current activity by attempting to open them in the INOUT mode. A discussion of such modes is contained in the BPM Reference Manual, 90 09 54. If the file can be opened, it is inactive and the program prints

file IS INACTIVE

The ACTIVITY command has the format shown below.

>A [CTIVITY] file[,file,...,file] (RET)

If the file cannot be opened, it is either currently open or the user does not have write access to the file, and in either case the file is considered active. The program prints

file IS ACTIVE

Should any of the indicated files not be accessible to the user as described under file accessibility, the CANNOT ACCESS FILE message will be given.

DELETE (File deletion)

If the user may access the file (see "TEST", above) and it is in his log-in account, the file will be deleted.

>D [ELETE]file[,file,...,file] (RET)

If the user may not access the file, or it is not in his log-in account, the file will not be deleted and the following message is given.

CANNOT DELETE FILE file

COPY (Copy file)

COPY copies $file_1$ to $file_2$. It has the format shown below.

>C [OPY]$file_1$,$file_2$ (RET)

If $file_1$ is not accessible to the user (see "TEST", above), the CANNOT ACCESS FILE message is given. If $file_2$ cannot be opened in the output mode, the following message is given.

CANNOT CREATE FILE file

FERRET can copy records up to 512 words in length. A larger record causes the copy to be aborted with the message

CANNOT COPY - - RECORDS TOO LARGE

EXAMINE (Examine file)

Should the file not be accessible (see "TEST", above), the appropriate message will be given. EXAMINE has the format shown below.

≥ E[XAMINE] file ⏎

Since a number of actions are possible in connection with EXAMINE, the subsystem will prompt with the # character. For example,

≥ E ALPHA ⏎ (examine file ALPHA)
#

The following characters cause the indicated action.

# X ⏎	Exit from file examination mode.
# N ⏎	Display, in decimal, the number of records in the file.
# ⏎	Print all records in the file in EBCDIC.
# m ⏎	Print record m in EBCDIC.
# m,n ⏎	Print records m through n in EBCDIC.

In any of the print commands, above, typing H as a first character causes the print to be in hexadecimal, with four words of the record per line. Lines of zeros will be suppressed. For example,

#H20 ⏎

causes a hexadecimal dump of record 20.

Records larger than 512 words will be truncated to 512 with the following message.

RECORD EXCEEDS BUFFER SIZE, 512 WORDS GIVEN

If the first record requested is not in the file, the following message is given.

FIRST RECORD NON-EXISTENT

If an end-of-file occurs before the last record requested, the dump is terminated with the message

UNEXPECTED EOF AFTER RECORD j

where j is the highest record number read.

9. FORTRAN IV-H SUBSYSTEM

The FORTRAN compiler processes FORTRAN source lines (see SDS Sigma 5/7 FORTRAN IV-H Reference Manual, 90 09 66) in the order in which they are read. It is important that the source program (input through M:SI) be in an order acceptable to the compiler.

FORTRAN OPTIONS

When the FORTRAN subsystem is called, it responds with a request for options. The user types an option list in which the options are separated by commas, and the list is terminated by a carriage return. Any time prior to typing the carriage return, a user may enter ESC X to erase the entire option list, and then enter a new one.

The following table lists all the forms of each option. The underlined options are the default options, used when only a RET is given in response to the OPTIONS request. Should the user type an option request, he receives only those he requests.

Option Form		
Wanted	Not Wanted	Description
BO	NOBO	Binary Output file. Ouput through M:BO.
LS	NOLS	List Source. Each source statement is output through M:LO.
SO	NOSO	Source Output. Source card images will be output through M:SO. The SO option allows the creation of a permanent source file while compiling from the console. It is not honored when input is from the RAD.
DB	NODB	Debug, Basic. Causes the binary output (BO) to be created with links to the basic run-time debug package (see "FORTRAN Execution with Debug Option"). If DB is requested, BO is assumed and need not be requested.
S	NOS	Symbolic machine language instructions may be written within FORTRAN source programs. (For details on format and usage, see Appendix B, SDS Sigma 5/7 FORTRAN IV-H Operations Manual).

Note that if any errors are encountered during compilation, they will unconditionally be printed along with the offending source lines, through M:DO. If M:LO is assigned to a different file than M:DO, the error lines and messages will also be output through M:LO (even if no listing was requested).

The default assignments of all the DCBs used during compilation are as follows:

DCB	Default Assignment
M:BO	File named BOTEMPα.
M:DO	The user terminal.
M:LO	The user terminal.
M:SI	The user terminal. This means that a user should be careful to type his source in the proper columns. The TABS Executive command greatly simplifies this.
M:SO	File named SOTEMPα.

The sequence of events during compilation is:

1. The source lines comprising a statement are input through M:SI and scanned.

2. If an error is found in the statement, the statement, followed by an error message, is unconditionally printed through M:DO and also output on M:LO (if different from M:DO).

3. If no error is found and the listing option has been given, the source statements will be printed through M:LO.

4. For each source statement free of error, the compiler will generate the code in SDS standard object language and output through M:BO. If DB has been requested, each statement will also have a link generated to the run-time debug package.

5. When an END statement is encountered, the compiler prints a program summary (if LS was requested) and completes generation of the relocatable object module through M:BO.

6. When compilation is complete, the compiler returns to the BTM Executive. A compilation may be interrupted via ESC ESC if the user desires to change ASSIGNments or options. To continue the compilation, give the Executive command PROCEED. Upon re-entering the compiler, the user will receive another request for options. If he does not wish to change from his previous selection of options, he merely types a RET (the assumed

options become whatever he chose at the beginning of the compilation). However, any changes desired in options may be made at this time. Users should not try to select either BO or DB in the middle of compilation when NOBO and/or NODB were requested initially, because the resultant object module will not be executable.

7. The normal manner in which to create and debug FORTRAN programs is to use the Editor to first create the source file; then compile, load, and execute.

   However, it is possible to type a program directly to the compiler, a line at a time, while using SO to create the source file. The following error recovery procedure exists in the compiler for use by the experienced FORTRAN programmer wishing to create a small program directly from the console.

   Since the FORTRAN language denotes continuation by putting the continuation flag in the card that is the continuation, a compiler must always read ahead one card to see if the source line it just read was continued. For the on-line user compiling from the console, this causes him to have to input a line following any complete statement to denote the end of that previous statement thus getting any diagnostics a line later. When using the compiler in this direct manner, a user should (but need not necessarily) type a colon as the last non-blank character before his (RET). The colon will signify that no continuation lines follow, so analysis will take place immediately and diagnostics will be output as follows, without the user having to type in another line.

   a. After the error is printed, and if M:SI is assigned to the console, a retry request is given to the user. If he does not wish to re-enter his erroneous statement, then compilation will continue.

   b. If the user wishes to retry, he may input a new set of lines comprising the replacement for the statement. Upon termination of this set of lines, the statement(s) are compiled.

   c. If the retry statement(s) has no errors, then the compiler will check to see if a source line was input after the last line of the original statement. If so, the line will be output on the Teletype and the user will be requested to accept or reject it. If he accepts it, compilation continues; if he rejects it, a new record is read from the console and compiled. If there was no source line input after the original statement due to the ":" convention, regular compilation will continue.

   d. If the retry statement(s) contains errors, the series of events is exactly the same as (a) - (c) above.

   When obtaining source output from the compiler, the edit sequence number is implicitly k.000; where k is the statement number of a line typed to the compiler.

Example 1.

!FORTRAN		Select FORTRAN.
OPTIONS: (RET)		Default options - BO, LS, SO, NODB, NOS.
1:	A = B: (RET)	Source line 1 by user (with a "no continuation" mark).
2:	C = D (RET)	Source line 2 by user (it may be continued).
3:	E = F: (RET)	Source line 3 by user (with a "no continuation" mark).
4:	G = H (RET)	Source line 4 by user (it may be continued).
5:	END (RET)	Source line 5 by user (because it is an END statement, it is assumed to have no continuation).
SUBPROGRAMS : : PROGRAM END		Program summary.

Example 2.

!FORTRAN		Select FORTRAN.
OPTIONS: (RET)		Default options
1:	A = B, P: (RET)	Source line 1, with error.

1: A = B,P $	Diagnostic
****SYNTAX	
RETRY? 1: A = B*P ⏎	User retries
MORE? ⏎	End of user retry
2: C = D ⏎	Continue normal input
3: END ⏎	

Example 3.

!FORTRAN	
OPTIONS: ⏎	Standard options - LS, BO, SO, NODB, NOS.
1: A = B,P ⏎	Source line 1 by user.
2: C = D ⏎	Source line 2 by user.
1: A = B,P $	Diagnostic on source line 1.
****SYNTAX	
RETRY? 1: A = B*P ⏎	Retry request for source line 1, user retries.
MORE? 2: ⏎	Request for additional source lines on retry, but user has none.
ACCEPT? 2: C = D ⏎	User asked to accept previously input line. Y ⏎ or ⏎ means yes; N ⏎ is no.
3: END ⏎	End of program.
SUBPROGRAMS : PROGRAM END	Program summary.

Example 4

!FORTRAN	
OPTIONS: ⏎	Standard options.
1: A = B,P ⏎	Source line 1 by user.
2: A = B*P ⏎	Source line 2 by user.
1: A = B,P $	Diagnostic on source line 1.
****SYNTAX	

RETRY? 1: B = 1. ⏎	Retry request for source line 1, user retries.
MORE? 2: P = 3.7 ⏎	User types additional line in retry mode.
MORE? 3: ⏎	No more retry lines.
ACCEPT? 3: A = B*P ⏎	User accepts previous input (resequenced as source line 3).
4: C = A*P ⏎	Source line 4 by user.
5: END ⏎	Source line 5 by user.
SUBPROGRAMS ⋮ PROGRAM END	Program summary.

EXECUTION OF FORTRAN PROGRAMS

All object modules (BO) created by the FORTRAN compiler must be loaded by the Loader subsystem. The Loader will build DCBs for the SDS standard unit numbers (101, 102, 103, 104, 105, 106, and 108) and for any other DCBs that have been referenced by an ASSIGN command prior to entering the Loader subsystem.

SDS Standard Unit Number	Corresponding DCB	Default Assignment
101	F:101	Users terminal
102	F:102	Users terminal
103	F:103	Users terminal
104	F:104	Users terminal
105	F:105	Users terminal
106	F:106	Users terminal
108	F:108	Users terminal

All FORTRAN I/O statements refer to unit numbers, and there must be a DCB (F:u, where u = unit number) for each unit number used. If I/O is performed only via the FORTRAN II type READ/PUNCH/PRINT statements, then all DCBs will be provided (READ references F:105, PUNCH references F:106, and PRINT references F:108). However, if a user writes on any unit besides SDS standard units, he will have to make a request during loading of his program to define the corresponding DCB.

Example:

(FORTRAN statements)

```
          WRITE     (6,10) A,B,C
10        FORMAT    (3F10.4)
```

When the Loader has completed loading the program it will issue a request for the user to give the name of any nonstandard DCBs to be used. The user might respond

F:6 = MINE ⏎

F: ⏎

This causes the DCB, F:6, to be built and included with the user's program. F:6 is assigned to a file name MINE. The second line beginning with F: terminates the requesting of DCBs (see Loader subsystem for a discussion of DCB requests).

Example:

(FORTRAN statements)

```
          READ      (2,10)    I
          READ      (3,12)    J
          READ      (4,10)    K
          READ      (5,11)    L,M
          WRITE     (6,13)    I
          WRITE     (7,14)    J
          WRITE     (8.15)    K,L,M
10        FORMAT    (I10)
11        FORMAT    (6I4)
12        FORMAT    (3I7)
13        FORMAT    (2HI=,I10)
14        FORMAT    (1H1,5I5)
15        FORMAT    (6I20)
```

(Corresponding requests for DCBs)

F:2 ⏎

F:3 = DATA1,IN ⏎

F:4 = DATA2 (ACCT3), IN ⏎

F:5 = DATA1, IN ⏎

F:6 = OUT1, L ⏎

F:7 = OUT2 (ACCT3), OUT, L ⏎

F:8 ⏎

F: ⏎

The result of these assignments is as follows:

F:2 is assigned to the user's console.

F:3 is assigned to an input file named DATA1.

F:4 is assigned to an input file, DATA2, in the account ACCT3.

F:5 is assigned to the input file DATA1 (same as F:3).

F:6 is assigned to the file OUT1, which is a listing (L) file.

F:7 is assigned to the file OUT2, in the account ACCT3, which is an output (OUT) listing (L) file.

F:8 is assigned to the user's console.

F: ⏎ terminates the list of DCB definitions.

FORTRAN EXECUTION WITH DEBUG OPTION

The FORTRAN compiler offers two kinds of source-language debugging capabilities. One capability provides a trace of the numbers of source lines reached during execution of a program. The other capability traces the values stored into variables as the result of assignment statements. A special code will be generated into the user's program to provide these debugging features if the option is selected.

The DB (debug) option, specified in the FORTRAN option list, causes FORTRAN to generate the necessary linkage to allow the debugging capabilities. At run time, a program compiled in debug mode enables the user to trace program execution, request a snapshot dump, halt execution at a particular statement, and to execute this program in the step mode (i.e., the program will halt after the execution of each statement). The following discussion assumes that the object program was compiled in the debug mode.

After the program has been loaded into core storage for execution, the program identification is typed by BTM:

#MAIN

BTM then types an asterisk on a new line. The debug codes given in Table 1 will be accepted as input. In this table default values for the debug codes are underlined.

Each acceptable input code is acknowledged by a ⏎. If an input code cannot be recognized, a question mark (?) is typed instead of the ⏎.

The size of FORTRAN programs that may be loaded and executed is limited only by the size of the time-sharing environment. Since the Loader must be able to fit in core with the user program during load, the total loadable FORTRAN program size in a 16K on-line area is approximately 8000_{10} words, not including the library. Total program size of FORTRAN programs is printed on each compilation summary.

The diagnostic comments at compilation time are described in Chapter 2 of the SDS Sigma 5/7 FORTRAN IV-H operations Manual (90 11 44).

FORTRAN-LIBRARY/RUN-TIME DESCRIPTION

All basic external functions and subprograms available in the FORTRAN IV-H library are described in the SDS Sigma 5/7 FORTRAN IV-H Operations Manual (90 11 44).

Whenever an error is detected during execution, a diagnostic message is written through M:DO (and M:LO, if a different file). For a list of all possible diagnostics and their meaning, see the SDS Sigma 5/7 FORTRAN IV-H Operations Manual (90 11 44).

Table 1. Debug Codes

Code	Meaning
S ⏎	Step. The program is to be executed in the step mode. Before each statement is executed, its statement number is typed. This is followed by a carriage return and asterisk, and an input code is expected.
NS ⏎	No step. This input code terminates the step mode.
T ⏎	Trace. Trace the source lines reached during program execution. Before being executed, the statement numbers are printed, through M:DO, across the line (up to column 80) and are separated by blanks.
NT ⏎	No trace. This input code terminates the trace.

Table 1. Debug Codes (cont.)

Code	Meaning
V ⏎	Snapshot. A snapshot displays the values tored into variables as the result of assignment statements. A snapshot is of the form: variable name = constant (for scalars) variable name (element number) = constant (for arrays) The value of the constant is printed through M:DO in a format that corresponds to the type of the variable.
NV ⏎	No snapshot. This input code terminates the snapshot.
Adddd ⏎	Stop. The program is to stop each time it reaches line #dddd. When line #dddd is reached, the program types: STOP AT dddd and halts. On a new line, an asterisk is printed and any of the Debug input codes may be entered. When a G code (see below) is entered, program execution proceeds, beginning with statement dddd.
NA ⏎	No stop. The program will no longer stop at the statements previously designated by the A (stop) code.
G ⏎	Go. The program begins execution, and the indicated debug function are performed at the appropriate times. This code is the only one that does not result in a new input request.
R ⏎	Do not stop at lower levels of subroutine entry. Suppresses printing of subroutine name.
NR ⏎	Stop at subroutine entry.
H&bbbbbb	Halt only upon entry to the subroutine or function named "&bbbbbb" (as opposed to all subprograms).
NH ⏎	Halt at all subprograms.

10. LOADER SUBSYSTEM

The Loader subsystem loads SDS Sigma 5/7 Standard Object Language programs comprising Relocatable Object Modules (ROMs) from specified element files and/or through M:BI. It may also load library load modules from the file :BLIB in any specified account and also the :BTM account. The Loader will load one or more object modules that have been assembled by BTM Symbol, BTM FORTRAN IV-H, standard Symbol (off-line), standard Meta-Symbol (off-line), SDS FORTRAN IV (off-line), or FORTRAN IV-H (off-line). It will not build overlay structures, and always loads modules as protection type 00 (regardless of the type specified).

To enter the Loader subsystem, the following command is used:

!LOAD

LOADER OPTIONS

When the Loader subsystem is entered, it responds first with a request for the names of all element files from which the user wishes to load. If no element files are named, the Loader will assume input is through M:BI. (The default assignment of M:BI is the file BOTEMPα.)

ELEMENT FILES:

Upon typing the request for element files (above), the Loader expects a user to list his element file names with a comma separating the names and a carriage return terminating the list.

file name[(account,pass)],file name[(account,pass)]...,(RET)

The total number of characters in "file name", "(account, pass.)", and "," may not exceed 100.

Each file name consists of 1-11 alphanumeric characters and may optionally be followed by the account number and password in parentheses. If the user has assigned M:BI to his file of object modules or if he is using the default assignment of M:BI, then his element file list is just a carriage return. Any time prior to typing the carriage return, a user may enter (ESC) X to erase the entire element file list.

Example:

!ASSIGN M:BI,(FILE,MINE) (RET)
!LOAD
ELEMENT FILES: (RET)

In the above example, M:BI was assigned to the user file MINE and was the only element file needed during loading.

Example:

!LOAD
ELEMENT FILES: MINE, OURS, HIS (ACCT4) (RET)

This example shows the user of several different element files during load. The files MINE and OURS are both in the user's account, but the file HIS is in account ACCT4.

Upon accepting the element file list, the Loader issues a request for options:

OPTIONS:

This request expects an option list consisting of the following options separated by commas and terminated by a carriage return.

option[,option]...[,option] (RET)

Any time prior to typing the carriage return, a user may enter (ESC) X to erase the entire option list.

Options	Description
N	No system library search will be made for unsatisfied primary references. Unless this option is specified the :BLIB file in the BTM system account (:BTM) will be searched for unsatisfied primary references.
M	A load map† of all DEFs and REFs will be output through M:LO when loading is completed. If this option is not specified, only SREFs (secondary references) and PREFs (primary references) will be output through M:DO (and M:LO if it is assigned to a different file).
U(a_1,a_2,...,a_n)	Unsatisfied references should cause a search of the :BLIB file in each of the accounts specified before the optional search of :BLIB in the BTM system (:BTM) account. If this option is not specified, no search of nonsystem accounts will be made to satisfy primary references.
P	When option P is specified, all programs defined by separate modules will be started on the next highest X'100' word boundary. The starting bias for a program is X'200' in the on-line memory area. Specifying the P option simplifies the task of relating assembly listings to memory locations while debugging. It should be noted however that separate control sections within each module will be contiguous and will not be started on any particular boundary, other than doubleword.

†See sample load map at the end of this chapter.

Options	Description
D	Option D provides for execution of programs under control of a debug program, and is primarily of use in checking out assembly language programs; however, it may be put in control, with no ill effect, during the execution of any program that can be run using the LOAD subsystem.

When the option list has been accepted, the loader loads the specified ROMs. If any errors are encountered during loading, an error message is output through M:DO (and M:LO if it is assigned to a different file). The following section, "Loader Error Messages", describes each message in detail. For any unsatisfied REFs to DCBs (F:alpha, where alpha is a name consisting of 1 to 8 alphanumeric characters), the DCBs will be built by the Loader and the default assignment will be to the user terminal.

The Loader will also request that the user specify any DCBs he will need for which there are no REFs or for which to change the assignment. (Any FORTRAN unit numbers aside from 101, 102, 103, 104, 105, 106, 108 fall into this category. See "FORTRAN IV-H Subsystem" for details.) The request is of the form:

F:[n][=name[(account,pass)]][,option]...[,option]

where

F: prompts the DCB description

n is the unit number (for FORTRAN users) or an alphanumeric name (up to eight characters).

=name indicates the name of the file to which the DCB is assigned. If a carriage return has been given right after "n", the Loader would have assumed the DCB was assigned to the users terminal.

(account,pass) specifies the account and password in which the file "name" (above) exists if it is not in this user's account.

options: there are three possible types of options that may be requested aside from the file name (the file name must be the first item mentioned after the "=", but these options may be in any order following the file name):

1. The function option may be any of the following:

   IN

   OUT

   INOUT

   OUTIN

   where

   IN specifies the file is only to be used for input.

   OUT specifies the file is only to be used for output.

   INOUT specifies the file is to be used in an update mode.

   OUTIN specifies the file is to be used as a scratch file. This is the default option.

2. The release option is

   REL

   which specifies the file is to be released at the end of this execution (the default option is that it will be saved).

3. The listing file option. This specifies that the file will eventually be listed on a listing device. The option is

   L

Note: When assigning an input disc file, it is important that the IN option be given. The default assignment in all Sigma loaders is OUTIN, meaning a scratch file. Attempting to read before writing will cause the Monitor to create a new file of the same name.

Example:

F (RET)

The user does not wish to have any DCBs assigned to files. This is also the way a user ends his DCB requests (see next example).

Example:

F:5 (RET)
F:6 = MINE (RET)
F:7 = YOURS (BTM1),IN (RET)
F: (RET)

The user wanted F:5 assigned to his terminal and F:6 assigned to the scratch file MINE, which is to be saved at the end of the job. F:7 is assigned to an input file, YOURS, in account BTM1. The F: (RET) ends the list of assignments.

When the load has been completed, the Loader will issue a message denoting the highest error severity level encountered.

SEV. LEV. = n

where

n is a hexadecimal digit.

Then the Loader requests the user to specify whether or not the load module just formed should be executed:

XEQ? { Y / N / S,adr } (RET)

where

Y or (RET) means yes, execute the load module.

N means no, do not execute the load module and exit to the Executive.

S means yes, execute the load module and use the following address (adr) as a start address for execution.

adr is either an external definition (optionally followed by a hexadecimal addend value) or a signed absolute hexadecimal address.

The first digit in the absolute hexadecimal constant must be a "." (e.g., S,.1AC).

Example:

SEV. LEV. = 0

XEQ? Y (RET)

In this example, there were no load errors and the user requested execution of his program.

Example:

SEV. LEV. = 3

XEQ? S,MYPROG +.A3 (RET)

This load module has a severity level of 3, but the user wishes to execute it anyway. A start address of MYPROG $+A3_{16}$ is given. MYPROG is an external definition (i.e., appears in a DEF statement in a Symbol or Meta-Symbol program or is a subroutine or function name in a FORTRAN program).

Example:

SEV. LEV. = C

XEQ? N

!

There were serious errors encountered (in the above example) and the user decided not to try to execute his load module.

EXECUTION

When a user program is in execution, the Load subsystem monitors the process and will respond to any errors, such as a nonexistent instruction, with an appropriate message.

Should the terminal user key in (ESC)(ESC), the Load subsystem is put back in control and it will inquire

PROCEED?

If the user responds with

Y

execution will continue. Any other response causes the Executive to be put in control.

Keying in the (ESC)(ESC) command twice while a user program is in execution will cause the Executive to be put directly in control. Should the Executive PROCEED command be given, the Load subsystem will still inquire as above, whether user-program-level execution is to continue.

LOADER ERROR MESSAGES

If a user specifies a nonexistent option in his option list (see "Loader Options" above) the Loader issues the message:

ILLEGAL OPTION

and regenerates a request for options.

During the load process, any errors are output through M:DO (and M:LO if it is assigned to a different file). The form of the error messages is as follows:

error statement 1 (any message in Table 2)

error statement 2 (any set of comments in Table 3)

Table 2. Loader Error Messages

Message	Description
NO LIB FILE	The specified library could not be found.
NO ELEMENT FILE	The requested element file could not be found.
ILLEGAL ORIGIN	An attempt was made to load outside the available area.
ILLEGAL ROM DATA	The ROM contained illegal object language.
CHECKSUM ERROR	There was a checksum error in the specified record.
SEQUENCE ERROR	The sequence of the record following the specified one was not equal to the current sequence plus one, and the current record is not the last record in a ROM, or the first record in a ROM is missing.
STACK OVERFLOW	There is not sufficient room in memory for the Loader, the program, and the loader stacks.

Table 3. Loader Error Comments

Comment	Description
PROCESSING LIBRARY account SEQ. NO. xx OVERALL ROM NO. yy	The error document in error statement 1 (see Table 2) occurred after the Loader had processed all the ROMs contained in the specified element files. This error was encountered in the "account" named in line 2 of this comment. It occurred in record "xx" of the "yy"th ROM encountered in this entire load operation ("xx" and "yy" are hexadecimal numbers).
LOADING FROM BI SEQ. NO. xx OVERALL ROM NO. yy	The loader was processing ROMs through M:BI when the error occurred in the "xx"th record of the "yy"th ROM ("xx" and "yy" are hexadecimal numbers).
LOADING ELEMENT FILE name SEQ. NO. xx OVERALL ROM NO. yy	The loader was processing ROMs from the file, "name", and the error occurred in the record sequenced "xx" in the "yy"th ROM encountered in entire load operation ("xx" and "yy" are hexadecimal numbers).

Example:

ILLEGAL ORIGIN
LOADING FROM BI
SEQ. NO. 1A
OVERALL ROM NO. 1A

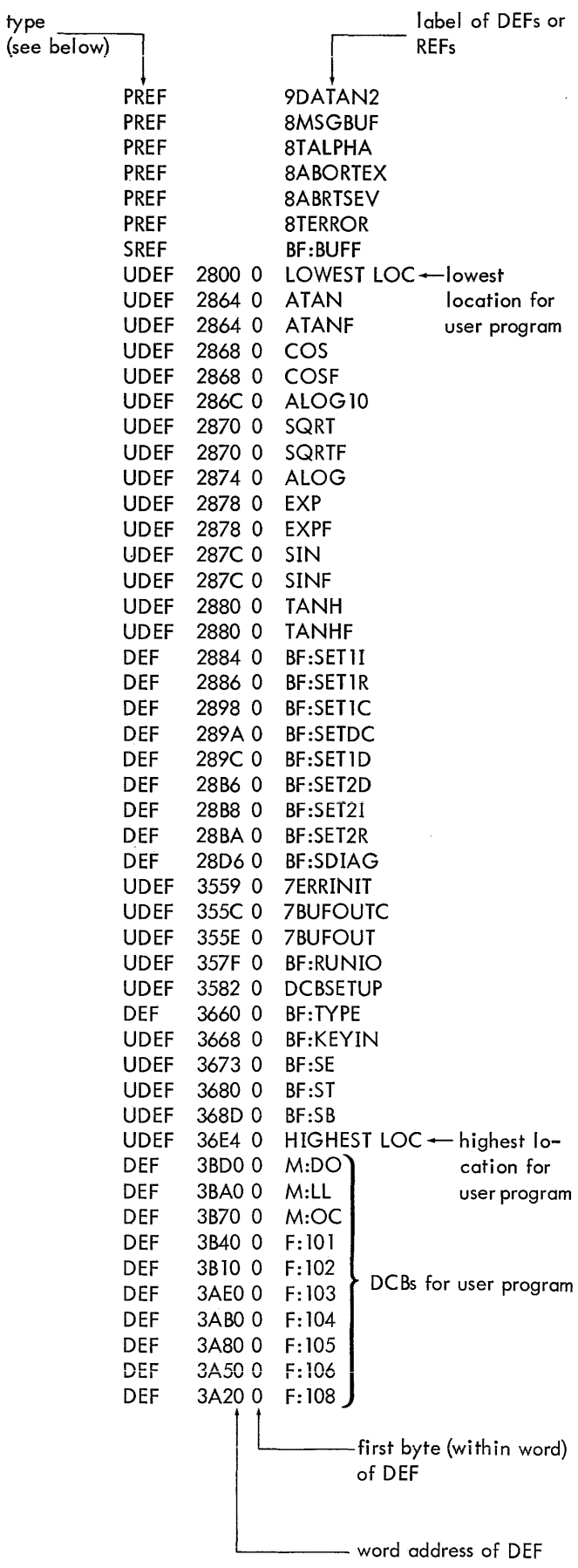
Example:

SEQUENCE ERROR
PROCESSING LIBRARY
MY ACCT SEQ. NO. 4E
OVERALL ROM NO. 31

Example:

CHECKSUM ERROR
LOADING ELEMENT FILE
MINE SEQ. NO. 1C
OVERALL ROM NO. 2B

LOAD MAP EXAMPLE



where "type" is:

SREF (Secondary REFerence.) The label specified on this line was referenced only as a secondary reference (see Symbol and Meta-Symbol manual 90 09 52).

PREF (Primary REFerence.) The label specified was a primary reference, but no corresponding definition was made for that label.

DEF (External DEFinition.) The label has been both defined and referenced.

UDEF (Unused DEFinition) The label has been defined but not referenced.

DDEF (Double DEFinition) The label has been encountered more than once during loading. The first definition is used.

UNDEFINED SYMBOLS IN LOAD MAP

When the D option has been specified, the load map will also include undefined internal symbols for each element file included in the loading process. For example:

```
!LOAD
ELEMENT FILES:  BO1, BO2
OPTIONS: D, P
    PREF      ALPHA
    PREF      BETA
**UNDEFINED INTERNALS**
* EF - BO1
    USYM      GAMMA
    USYM      DELTA
* EF - BO2
    USYM      ZAP
F:
```

only with 'D' option

The undefined-symbol map will always directly follow the normal map produced by the Loader. Undefined internal symbols can be detected only for those object modules assembled on-line by Symbol, or in the background by Meta-Symbol, with the SD option.

The loader also builds tables of internal symbols for those modules containing symbolic debugging information. "Internal symbols" are those symbols in an assembly that are not made external through use of the DEF/REF directives, nor defined under control of a LOCAL directive. That is, they are the "global" symbols of the particular assembly.

An internal symbol table is built for each element file included in the load process, for later use by the debug program and to provide values for undefined symbols displayed in the load map. Only seven characters of any symbol are retained, along with its value and type/resolution codes. Should symbols in the element file have duplicate initial characters and length, the first such symbol encountered will be retained. The same name length restriction applies to element file names; therefore, they should be restricted to seven characters when creating the binary input.

SATISFYING UNDEFINED SYMBOLS

After any DCB assignments have been made and the severity level printed, the user will be requested to satisfy undefined symbols.

If any PREFs were detected, the Loader will prompt with

```
**SATISFY EXTERNALS**

<
```

The user may respond by typing the name of a PREF and its satisfying value, after which he will receive another "<" prompt. The PREF name being satisfied is terminated with the character ">". For example:

```
<PREF>value
```

where

"value" is an expression of the form

DEF name

DEF name ± hex. constant

hex. constant (identified by a leading ".")

The following error messages are possible:

-NAME ERROR	(Improper external name.)
-CONSTANT ERROR	(Error in hex.constant.)
-VALUE ERROR	(Error occurred in value field.)

At any time the user may enter a carriage return only, signifying that no further PREFs are to be satisfied. When this is done, the Loader will place Delta in control — unless there were undefined internal symbols. In the latter case, the subsystem prompts with

```
**SATISY INTERNALS**

* EF - BO1

<
```

The user may respond with a value for any of the symbols listed as undefined for this element file. A carriage return alone causes the Loader to go to the next element file until all those that contain undefined symbols have been processed.

When satisfying internal symbol values, the user may also mention an existing internal symbol in the concerned element file. The only time this will be prohibited is when the combined size of the Loader, the loaded program, the internal symbol tables, and the DEF/REF stack exceeds the size of the on-line memory area; when this is the case, it will be noted by a console message.

EXECUTION AND SYSTEM INTERFACE UNDER THE "D" OPTION

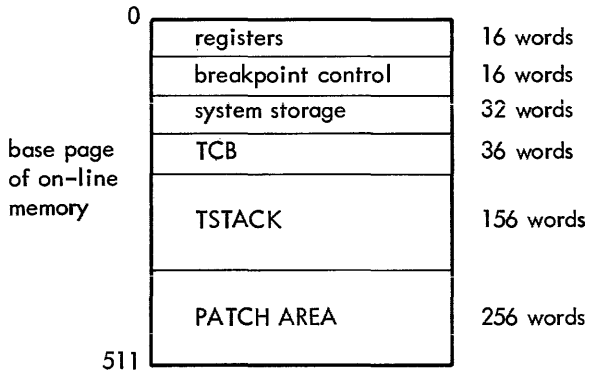
When the loading process is complete, the Delta debug program is put in control and it will prompt input with the console bell. At this point, the user's program is completely initialized and ready to start execution. The ;G command with no expression will start the program at its normal entry; however, any of the debugging commands may be used to alter the program, insert break points or start execution at any location.

The user will find this an excellent point at which to break to Executive level and execute the SAVE command. This will provide the ability to restart the program from scratch without having to go through a rather lengthy loading process.

When the Loader has completed its task, the user's program is set up for execution in the following manner:

The first page of memory is used for system functions. It contains

1. The user's TCB (Task Control Block).
2. System storage for registers, breakpoint control, and error messages.
3. The user's temp stack (containing 156 words).
4. A pre-allocated patch area of 256 words.



A special symbol, %P, is built into Delta to give the base of the patch area. It may be displayed by the debug command

```
%P = .C100
```

The user's program is loaded starting at X'200' of the on-line memory area and continues to an even page boundary. If the P option was specified, individual ROMs will be loaded on X'100' word boundaries. Beyond the user program, the loader reserves two pages of memory into which it builds the DCB name table and any M: and F: DCBs the user may require. The last word of this area is associated with the symbol %H, in the debug program.

The entire range of the program is considered to be program data, and accordingly will be swapped in and out during execution.

A self-contained program, i.e., one that does not require any memory cells external to itself, will have little necessity for interface with the BTM Executive.

On entry, register zero contains the TCB address and the user may push and pull indirectly through this address to make use of the pre-allocated temp stack (156 words).

It is possible to do disc file I/O using the standard BPM CALIs for the OPEN/CLOSE, READ/WRITE, and positioning functions. The only other functions normally required would be the Teletype I/O and exit CALs[†] for use under BTM.

With these, the program can

1. Read characters (CAL3,0) from the console.
2. Write characters (CAL3,1) to the console.
3. Set activation class (CAL3,2).
4. Exit to the Executive (CAL3,6).

The following example is a simple program to write a message to the user console and exit.

```
!BTM SYSTEM IS UP
2/1/69  12:02
!LOGIN:NAME,85066
!TABS 8,16

!EDIT
*BUILD TEXT
     1.000           DEF AA,ENTRY,TEXT
     2.000 AA        EQU     5
     3.000 TEXT      TEXTC   'HELLO!'
     4.000 ENTRY     LI,1    1
     5.000           LB,2    TEXT
     6.000           LB,0    TEXT,1
     7.000           CAL3,1  0
     8.000           AI,1    1
     9.000           BDR,2   $-3
    10.000           CAL3,6  0
    11.000           END     ENTRY
*END

!ASSIGN M:SI,(FILE,TEXT)

!SYMBOL
OPTIONS:  BO
**  END OF ASSEMBLY  **

!LOAD
ELEMENT FILES:
OPTIONS:  P, M, D
UDEF      5     AA
UDEF   C200 0   TEXT
UDEF   C200 0   LOWEST LOC
UDEF   C202 0   ENTRY
UDEF   C20A 0   HIGHEST LOC
DEF    C7D0 0   M:DO
F:

SEV. LEV. = 0
**  NO UNDEFINED INTERNALS**

;G
HELLO!
XIT AT ENTRY+.6
```

[†]These CALs are described in "Teletype Operations", Chapter 4.

Programs that must build tables external to themselves have a slightly more difficult time, because the burden of notifying the Monitor about a change of program size rests with the user.

Initially, the swap size of the program is set to include the base page, the program pages, and the Loader-built DCB pages. Additional space will always be available in the on-line memory area, however, an attempt to use it without expanding the swap size would almost certainly be doomed to fail.

It is possible for the program to obtain the maximum number of pages available (CAL3,14)†, and then set the swap size to this value (CAL3,14)†. However, this is poor practice since it may produce an unnecessarily large amount of swap I/O and should be done only when complex data structures require it.

In the ideal case, the program requiring additional data storage will start building its tables extending upward in memory from the DCB pages, expanding its swap size by a page at a time as necessary. The format of the CAL3,13 also allows a separate data area at the high end of core memory. When building expanding tables in this manner, the user must ensure that the data handling subroutines detect the necessity for expansion of swap size prior to insertion of data. It is possible (although not likely) that the program could be swapped out between the insertion of data into an inactive page and a subsequently given CAL3,13.

The following computations will be of interest for programs of the above type:

1. On entry, R0 contains the TCB address in the base page. The start of the on-line memory area can be obtained as: START = TCB address and X'0001FE00'.

2. If the CAL3,13 is given with R0 negative, the Monitor returns, in R0 and R1, the byte pattern currently describing the program structure (see Appendix D). The following instruction sequence thus obtains the page count of the program (since it starts as all program data).

```
LI,0      -1
CAL3,13   0
STW,0     PD
```

3. The first available data address of the first inactive page is then equal to

   START + (PD**9) = D1

4. CAL3,14 returns in register zero the maximum number of pages (MP). The highest available address is then

   START + (MP**9) -1 = TOP

From these basic values, a program can perform all necessary computations for use with the memory management CALs.

†These CALs are discussed under "BTM System CALs", below.

DEBUGGING

Delta is specifically designed for the debugging of programs at the assembly-language and machine-language level. It operates on object programs and the tables of internal and global symbols accompanying them, but does not demand that the tables be at hand. With or without symbol tables, it recognizes machine instruction mnemonics and can assemble, on an instruction-by-instruction basis, machine language programs. Its main purpose, however, is to facilitate the activities of debugging. These are

1. The examination, insertion and modification of elements of programs: instructions, numeric values, encoded information, etc.
2. Control of execution, including the insertion of breakpoints into a program.
3. Tracing execution by displaying information at designated breakpoints.
4. Searching programs and data for specific values.

To assist in the first activity, the assemblers will include in a program's table of symbols information about what type of data each symbol represents: symbolic instructions, decimal integers, floating-point values, single and double precision values, EBCDIC encoded information, and others.

The following summary lists the Delta commands and facilities in eleven broad groupings. Delta commands are discussed briefly in "Delta Commands", below, and in greater detail under "Command Descriptions".

1. Evaluation of expressions consisting of symbols, constants, special symbols, and the operators plus and minus (+-).
2. Commands for printing the contents of memory cells and opening them in preparation for change.
3. Format codes that enable the user to control the output format used in the evaluation and display commands of groups 1 and 2 above.
4. Commands for storing new contents in open memory cells.
5. Format codes that control the conversion of input constants typed by the user.
6. Special symbols used to examine machine flags and to control operating bounds for Delta.
7. Commands to insert into, delete from, change completely, and otherwise control the symbol tables used by Delta.
8. Commands to initiate and continue execution.
9. Commands to insert, delete, and control breakpoints.
10. Commands for searching memory.
11. Miscellaneous commands.

In outlining the commands, the following conventions are used in depicting the format of the orders typed by the user:

1. Special characters numbers, and upper case letters stand for themselves. Thus, in the command "e;G" the user actually types the semicolon and the G.
2. Lower case letters are placed where the user has a choice of things to type. The letter e alone or postscripted is used to stand for any expression consisting of symbols, special symbols, constants, and the operators plus and minus (+-). At times, other lower case letters are used to stand for expressions when some additional mnemonic content seems desirable.
3. Examples are n, loc, val, m.
4. The letter f stands for one of the format characters.
5. Abbreviations for user key strokes are:

Letters used in text	User Keystroke
(RET)	(RET)
(LF)	(LF)
↑	SHIFT and N
\	SHIFT and L
(TAB)	CTRL and I

Most of the Delta commands are terminated (and thus delivered to Delta from the resident COC handler) by the carriage return (RET) character; however, certain other characters also delimit commands to allow dialog within a single typed line. The command terminating characters of Delta are (RET), (LF), ↑, (TAB), \, and =.

DELTA COMMANDS

EXPRESSION EVALUATION

e = Evaluates and types the value of the expression e in the most appropriate format.

e(f= Evaluates and types the value of e in format f (see format codes listed below).

DISPLAYING AND OPENING MEMORY CELLS

e/ Displays the contents of a cell e in the most appropriate format. The cell is opened; that is, it may now be changed.

e(f/ Displays the contents of cell e in format f.

e1,e2/
e1,e2(f/ Displays the contents of cells e1 through e2 in the most appropriate format or in the specified format. Cell e2 is opened.

e Opens, but does not display, cell e.

/ A slash alone, following a display, displays but does not open the cell addressed by the display. (Displays the cell addressed by the last quantity typed (;Q)).

(TAB) Tab alone, following a display, displays and opens the cell addressed by the display.

STORING IN OPEN MEMORY CELLS

e (RET) Stores the word specified by e in the currently open cell and closes the cell.

e (LF) Stores e in the currently open cell, closes it, and opens and displays the next higher addressed cell.

e Stores e in the currently open cell, closes it, and opens and displays the next lower addressed cell.

e (TAB) Displays and opens the cell addressed by the last quantity typed (;Q). If an expression precedes the (TAB) it is stored in the previously open cell.

FORMAT CODES FOR / AND = COMMANDS

F	Symbol table format type
X	Hexadecimal word
I	Signed decimal integer
C	EBCDIC characters
R	Symbolic instructions with symbolic addresses
A	Symbolic instructions with hexadecimal addresses
D	Double word decimal integer
S	Short floating-point number
L	Long floating-point number
(f;/	Sets the default format for / commands to f
(f;=	Sets the default format for = commands to f

INPUT CONVERSIONS AND EXPRESSIONS

Expressions for evaluation, display, and storage are formed from the program symbols, explicit constants, and special symbols using the operators plus and minus (+-).

The conversions that may be specified for explicit constants are: 1) hexadecimal when introduced by a period (e.g., .BAD), 2) EBCDIC characters when surrounded by single quotes (e.g., 'BAD'), and 3) decimal when the constant c consists only of numerics (e.g., 1234).

SPECIAL SYMBOLS

Special symbols are recognized by Delta and may by used in expressions. Used as commands, they set the value of the corresponding symbol table entry.

$ or .	Last opened cell address	
;I	Instruction counter	As set by the last entry to Delta or changed by the user
;C	Condition code	
;F	Floating Controls	
;M	Search mask.	
;1	Lower search bound.	
;2	Upper search bound.	
;Q	Last quantity typed.	

SYMBOL TABLE CONTROL

s;S	Select internal table s.
e(f<s>	The symbol s is assigned value e and format f.
s(f!	The symbol s is assigned the value of the currently open cell ($) and format code f.
s;K	Symbol s is removed from the symbol table.
;K	Remove all global symbols except instruction mnemonics.

EXECUTION CONTROL

e;G	Begin execution at e.
e;X	Execute the instruction e.
n;P	Proceed with execution n times.

BREAKPOINTS

e,n;B	Set the n^{th} instruction breakpoint at location e.
e;B	Set the next available breakpoint at location e.
e,n,loc(F;B e,n,loc;B e,,loc;B e,,loc(F;B	Same as above but display contents of loc when the break occurs. Display is in default format or that specified.
e,n,loc;BT e,n,loc(F;BT	Same as above but proceed from the break after printing (trace mode). Display is in format given by (F, or default format.
n;B	Remove the n^{th} instruction breakpoint.
0;B	Remove all instruction breakpoints.
;B	Display all active instruction breakpoints.
;P	Proceed from the break.
n;P	Proceed and do not break until the breakpoint has been passed n times.
;T	Proceed automatically from the break after printing (set trace mode).
(ESC) (ESC)	Break at the current execution point (analogous to the machine's RUN-IDLE switch).

Output produced when a breakpoint is reached is

n;B>loc

where

n is the breakpoint number.

loc is its location.

If a display is specified, the output produced is

n;B>loc addr/contents

MEMORY SEARCHING

Memory between the bounds specified in ;1 and ,2 (initially set to the lower and upper limits of the user's program) is searched under the mask in ;M (initially all ones).

e;W	Search for and display words that match e under the mask ;M.
e;N	Search for and display words that do <u>not</u> match e.
e;1	Set the memory search lower bound to e.
e;2	Set the memory search upper bound to e.
e1,e2;L	Set ;1 to e1 and ;2 to e2.
e;M	Set the search mask to e.

MISCELLANEOUS COMMANDS

;R	Display locations of displayed cells as symbol plus relative hexadecimal offset.
;A	Display locations as hexadecimal numbers.
e1,e2;Z	Zeros memory from e1 through e2.

SYNTAX DESCRIPTION

The language of Delta follows the formula of simplified expressions and single (or a few) letter commands, which minimizes the number of keystrokes required. Because every keystroke counts, and users can find most errors easily by eye, only a few error conditions are explicitly commented. The most common commands have been assigned to lower case keys in order to simplify typing.

COMMAND DELIMITERS

In order to interface efficiently with the time-sharing system, Delta has been made "message" oriented. That is, only certain characters are recognized as command line delimiters (end-of-message characters) and cause the COC handler to deliver the command line to Delta for interpretation. The characters used as command line delimiters are:

/	The open and display command.
=	The expression evaluation command.
(RET)	The store command and delimiter of other commands.
(LF)	The store and open next command.
↑	The store and open previous command
(TAB)	The store and open indirect command

With the exception of / and =, the commands above cause a carriage return and line feed. The slash (/) and equal (=) commands interact within a single typed line.

More than one command can be input on a command line. The command delimiter within a command line is the space character. For example:

```
TAG1;B   TAG5,,.15   ;BT
```

SYMBOLS

The symbols used by Delta for reference to memory locations, computing values, and formatted displays are those supplied from the assembly of the program plus any added from the terminal by the user. They are carried in Delta's symbol table as seven characters plus count. Symbols longer than seven characters are truncated to include only the first seven, although the count of characters is retained. Thus, symbols that were originally longer than seven characters are indistinguishable from each other; only the last received definition is retained.

The symbols used by Delta follow the same rules as those for Symbol and Meta-Symbol – they are made up of the alphabetic characters A-Z, the numerics 0-9, and the specials $, @, #, :, _ ; the first of which must be nonnumeric, and the number of characters must be less than 8.

Symbols have an associated type code which allows Delta to use a conversion for display that matches the symbols original use. Symbols have either a constant value or are associated with a memory location. If the latter is the case, then the type code describes the contents of the location. Symbol types are:

1. Instruction.
2. Integer.
3. EBCDIC Text.
4. Short floating-point number.
5. Long floating-point number.
6. Hexadecimal.

SPECIAL SYMBOLS

The initial contents of the symbol table include the mnemonic names of machine instructions and a list of special symbols associated with program debugging. The special symbols may be used in expressions for values. The special symbols and values associated are given as follows:

Symbol	Value
$ or .	Memory location of the last opened cell.
;I	Instruction counter contents at program interrupt.
;C	Condition code contents at program interrupt.
;F	Floating control contents at program interrupt.
;M	The mask used in memory searches.
;1	The lower bound used in memory searches.
;2	The upper bound used in memory searches.
;Q	The last quantity typed by Delta, or the value stored by the user with the commands (RET), (LF), and (TAB).

The following values are initialized in the debug program on entry from the loader:

Symbol	Value
;1	Lowest user program location – excluding base page used for TCB, TSTACK and system storage.
;2	Highest program location. This corresponds to the last cell of the page containing Loader-built DCBs.
;I	Start address of the user program.
%P	First location of the patch area.
%H	Address of first word of the next page above the Loader-built DCBs.

Except for "$", ".", and ";Q", the value of these symbols table entries can be set by using a special command form in which a defining expression is given followed by the special symbol to be set and a carriage return:

```
.C46B;I     Set ;I to hex.C46B
.FFF;M      Set ;M to hex.FFF
```

The value of all special symbols may be displayed using the = command.

```
;C=4
;I=.C3BD
;F=2
```

The symbols $ and . always have the location of the last opened cell as their common value. The shorthand is convenient in the same way as in symbolic assembly code.

```
A/  LW,4 K45   $(X=.C105   $/  LW,4 K45
```

The ;Q shorthand for the last thing typed is similarly convenient in special situations:

```
ALPHA/AI,5  7   ;Q+2

./AI,5  9
```

INPUT OF EXPLICIT CONSTANTS

When the user wishes to type in numbers, he must specify the conversion that he wishes made on his input. Three conversion types are provided by Delta; hexadecimal, obtained by introducing the constant with a period (.); EBCDIC, obtained by enclosing the characters in single quotes ('); and decimal, the conversion used on strings of numerals. EBCDIC character strings follow the same rules as symbols used by Delta except that the maximum length is four characters.

Some examples of input constants in various formats are;

```
.ACE    100    .100    14    .A
'EBCD'    'A'
```

Note that the single quote (') is required to terminate the EBCDIC text string, and that the string must consist of no more than four characters. If fewer than four, they are right-justified and zero filled.

EXPRESSIONS

Expressions are typed by the user to represent a location value, parameter value, or a value to be assembled into a machine instruction. Expressions are composed of symbols, explicit constants, and the operators plus (+), minus (-), and space ( ). Multiplication, division, and other operations are not allowed and, in fact, the characters usually used to indicate them are used for other things – the asterisk to indicate addressing in instructions and the slash as the command for display.

The user should have little trouble constructing legitimate and correct expressions for the values he wishes, as can be seen from the examples below:

```
A
A+3
A+3-B
AI,1    2
STW,7    *LOC
LW,7    TAB,5
CAL1,3    LIST
```

The space character, in addition to its use to introduce the address field in expressions to be assembled into instructions, is also used to mean plus (+). This convention is convenient for typing, as a space does not require the case shift that a plus does. Thus, some equivalent expressions and commands are:

```
A   3   and   A+3

LW,5   ALPHA+3   and   LW,5   ALPHA   3
```

COMMAND DESCRIPTION

MEMORY LOCATION DISPLAY: THE / COMMAND

The / character is a command to Delta to open a memory cell and display its contents. The cell is indicated by an expression preceding the / character. The expression is evaluated and the word address portion is used as a memory address. If no format is given and the default is F (the normal case), then the symbol table is searched to find a symbol at the next smaller location than the indicated address, and the data type associated with the symbol found is used to control output formatting. The following are examples of printout that might result from various slash commands:

```
100/        .34
A1/         BAL,6 ALPHA
A+1/        STW,5 BETA
BETA/       ABCD
```

The user may either temporarily or permanently override this output format control by the symbol table code. Temporary change is accomplished by indicating the desired format in the command. The expression for the location is followed by a left parenthesis character, then by one of the format codes (see "Output Format Control" below for a complete list), and finally by the slash (/) command.

```
X(X/        .C1        hexadecimal conversion
X(C/        A          EBCDIC character conversion
X(I/        193        decimal integer conversion
```

Permanent change in output format is achieved by the command "(f;/" where f is the desired format code.

```
X/          .C1
(C;/        X/      A
```

If a slash is typed without preceding typing by the user, the cell addressed by the last item printed by the computer is examined but not opened. This allows the user to look at the indirect contents of a cell. In the example below, ALPHA remains the open cell even though the contents of cell DCT8 are displayed.

```
ALPHA/   LW,5   DCT8   /   .32
```

A cell may be opened without displaying its contents, by the use of the \ command. (\ is produced by pressing SHIFT and L keys together). This mode is convenient when the user wishes to insert new contents in memory and is not interested in the current contents. Delta remembers the mode of opening for cells and on (LF) and ↑ commands opens them in the remembered mode.

```
ALPHA\  BAL,4    SUB (LF)
ALPHA+.1\  STW,5    DCT2 (LF)
ALPHA+.2\  AI,6    .100 (RET)
```

More than one cell may be displayed using a single / command. Two expressions separated by a comma define the limits of display. They are the word address of the lower limit followed by that of the upper limit. Following display of the upper limit cell, it is open for change.

```
ALPHA,ALPHA+2/   BAL,4  SUB
ALPHA+.1/   STW,5  DCT2
ALPHA+.2/   AI,6   .100
```

Format codes may be specified with "(" as in the basic "/" command.

```
100,101 (X/    .58000100
101/    .68000200
```

EXPRESSION EVALUATION: THE = COMMAND

Expressions consisting of program symbols, explicit constants, special symbols, and the operators plus and minus (+-) may be evaluated by use of the = command. The expression may be that just typed by the user or the last one typed by Delta.

```
2 + 2 =.4
5 + 5 =.A
ALPHA/  BAL,5   SUB = .6A5006B3
5 + 5 =.A
5 + 5 (I = 10
```

The user can set the default format type by using the command "(f;=," where f is the desired format type. The initial default format is X for hexadecimal.

MEMORY MODIFICATION: THE ⏎, ⏎, ↑, AND ⇥ COMMANDS

Four commands allow the user to store a value into a memory location – the one opened by a /, \, or one of the modification commands ⏎, ↑, or ⇥. If no expression preceded the command character, the action taken is as described except that nothing is stored in the open cell.

e ⏎ The expression e is assembled and stored in the open memory cell. Carriage return and new line codes are sent to the user terminal and display modes are reset to default values.

```
A/      BAL,4 JWS     BAL,4          GEB ⏎
A/      BAL,4 GEB ⏎
JED/    EXU  LS (X/   .68000643  /   .78C ⏎
./      EXU  LS
```

Note in the preceding that a temporary display format was established by the "(X/" which carried over until the command reset it.

e ⏎ When the user terminates an expression with the ⏎ command, the value of the expression is stored in the currently open cell, that cell is closed, a new line is produced at the terminal, and the cell with the next higher location value is opened. The mode of the initial cell opening is preserved and carried forward on succeeding openings as is the display format.

```
A (I/       435          435 ⏎
A+.1/       763 ⏎
A+.2/       7689 ⏎
EM \        STM,4        ERS ⏎
EM+.1 \     BAL,6        LP ⏎
EM+.2 \     BGE          BB ⏎
```

e ↑ The action is exactly the same as for e ⏎ except that the cell within the next lower location value is opened.

```
EM+4/       .0      B       JH↑
EM+.3/      .0      AI,3    1 ⏎
```

e ⇥ The ⇥ command causes the typed expression to be stored in a currently open cell. Following output of a carriage return, the cell addressed by the most recently closed cell is opened and displayed. The effect is like that of a ⏎ command followed by a ;Q/. The ⇥ command is useful for patches:

```
A/          BAL,5   SUB ⏎
A+.1/       STW,6   BETA    B   PATCH ⇥
PATCH/      0       AI,6    1 ⏎
PATCH+.1/           0       STW,6  BETA ⏎
PATCH+.2/           0       B   A+2 ⏎
```

OUTPUT FORMAT CONTROL

Displays of the contents of memory locations via the / command and expression evaluation via the = command have their output format controlled by codes given with the / or = command or by the default format as set using the (f;/ and (f;= commands. The original default setting of the output conversion format is hexadecimal (X) for = commands and under control of the nearest symbol table type (F) for / commands. Temporary conversion type settings set by using e(f/ or e(f= are retained until the next ⏎ command is given. In particular, the temporary conversion type is retained over successive ⏎, ↑, /, =, and ⇥ commands.

```
(I;/
A (X/       .C ⏎
A+.1/       .D ⏎
A+.2/       .E ⏎
A+3/        15
```

The codes that provide for directing output formatting and conversion are given below. In all conversions, leading zeros in the printout are suppressed.

X The word (contents of memory or expression) is typed out as a hexadecimal number. Hexadecimal numbers are always typed with a leading period (.). X (hexadecimal) is the original default code for = commands.

F Conversion is according to the format code given in the symbol table for the location displayed or that for the next lower valued location symbol if no symbol occurs at the location in question. For = commands, F conversion is equivalent to X conversion. F conversion is the default code for / commands.

I The word is converted as a signed decimal integer.

C The word is converted to EBCDIC characters; that is, it is sent to the terminal directly. Non-printing characters may be output in this way, including the EOT (04) character, which will turn off some types of terminals.

R The word is converted to a symbolic instruction. Output has the form OP, R, ADDR, X (similar to assembler symbolic machine instruction format). OP is the symbol table value of the operation code part of the word (bits 0-7) – %XX is printed if the operation code value (XX) of the word is not an instruction. R is the value of the register field (bits 8-11) and (if nonzero) is printed as a decimal integer. If zero, it is suppressed along with the preceding comma. ADDR, the address field, is printed with a leading "*" if bit 0 is a 1 and is followed by the symbol obtained from lookup of value in bits 15-31. If no symbol corresponds to the value, then the next lower location symbol plus a relative hexadecimal offset is printed. Values less than the base address of the on-line memory area are always printed in hexadecimal.

If the index field (bits 12-14) is nonzero, it is printed as a decimal integer (1-7), following the address and a comma. If OP, R, and X are zero, only the address is printed.

A The word is converted in exactly the same way as for R format, except that the address field is always given as a hexadecimal number.

S Short floating-point number. The word is converted from internal floating-point format to the form XXXXX E±YY.

L Long floating-point number. Same as s except the current word plus the next highest addressed word are converted (same as S for = command).

D Double word decimal integer. The current word plus the next word are converted as a 64-bit decimal integer with sign (same as I for = command).

EXECUTION CONTROL: THE ;G, ;P AND ;X COMMANDS

The three commands described in this section allow the user to begin and continue execution of his program. Each of the commands is terminated by a carriage return (or a space, if it is in a multi-command line). Execution is started by typing "e;G" where e is an expression for the starting location. (The value of the expression is masked to form a word address.)

```
BEGIN;G
```

Execution can be stopped in any of three ways:

1. Encountering a breakpoint.
2. A user interruption via the ESC key (depressed twice).
3. An error causing a machine trap (illegal instruction, memory protect violation, etc).

In each case, the cause of the stop is reported by an appropriate message, the values of ;I, ;C, and ;F are set, and terminal control returns to the user.

```
BRK AT   .C5C3

PRIVIL INSTR AT   .C77B

;I=   .C77B
```

The user may proceed from a stop condition caused by error or manual intervention, by typing the ;P or the ;G command without a preceding address expression. The effect is to continue execution from the location specified by the current value of ;I (i.e., where execution left off, or a location specified by the user). The use of ;P for instruction breakpoints is covered in the next section. For user interruptions via the ESC key, execution continues as if the interruption had not occurred.

```
BRKAT   .C68C

;P
```

Proceeding from a machine trap will, in general, cause re-execution of the violating instruction and another trap.

```
MEM PROTECT FAULT AT   .C74B

;G

MEM PROTECT FAULT AT   .C74B
```

(In either of the above cases any expression typed before the ;P is ignored.) The ;X command assembles and executes the expression just preceding the ;X.

```
LH,3      TABLE+4;X

STB,6     *LOC;X
```

If the expression does not result in a legitimate instruction, the illegal instruction message results just as if the error had been in an executing program. If the expression is a branch, instruction control goes to the user's program (or causes a memory violation). Thus, the commands "B GO;X" and "GO;G" are equivalent. If the expression is a subroutine jump, the subroutine is entered and if it returns normally (to the calling location plus 1, 2, or 3), CPU control returns to Delta and terminal control is returned to the user. If the return is to other than the calling location plus 1, 2, or 3, the results are unpredictable.

BREAKPOINTS: THE ;B COMMAND

Delta provides the user with eight separate breakpoints on instruction execution. As each breakpoint is reached, a small amount of information is printed out, giving the break location and an associated value. A special mode allows execution to continue automatically after the breakpoint report to provide a limited kind of trace of the flow of execution control.

e,n;B The n^{th} breakpoint (there are eight, numbered 1-8) is set to stop execution and return control of the terminal to the user when the instruction at location e is reached. If n is not specified, Delta will assign the next available break number. If none are available, Delta produces the message "NONE". The user may then release one of the breakpoints he has set and try again. The breakpoint stop occurs before execution of the instruction at e. When the breakpoint is reached, Delta prints the number and type of breakpoint and its location.

```
A+3,1;B    A;G

1;B>A+.3
```

A third field of the breakpoint command may be used to specify a location to be displayed when the breakpoint is reached. Registers as well as core locations can be displayed in this way:

```
A+3,1,R5;B    A;G

1;B>A+.3   5/   .54
```

A particular display format other than the default can be specified by giving the break command in the format e, n, loc (f;B.

If the execution has stopped at a breakpoint, the user may examine and modify his program as appropriate and then continue from the point of interruption by giving the command ;P. A count may be given with the ;P command. If the count is n, the breakpoint will be passed n times before the break occurs. Execution can also be continued by the ;G command; however, the proceed count cannot be input with the ;G.

```
PH+8,2,R2(I;B    PH;G

1;B>PH+8   R2/    4     ;P or ;G

1;B>PH+8   R2/    5     ;P or ;G

1;B>PH+8   R2/    6     5;P

1;B>PH+8   R2/   11
```

The breakpoint is an interruption logically prior to execution of the instruction. A break at a particular location followed by immediate modification of the break location will cause the modified contents of the break location to be executed when the ;P is given. This is true even if the active break is released before or after the modification of the break location.

The n^{th} breakpoint may be removed by the command n;B. All breakpoints can be removed by the command 0;B. If the user wishes to trace a particular instruction, he may give either of the forms above (display or no display) and specify the T mode: e,n,loc(F;BT. In this mode, when the instruction at e is reached, the breakpoint reporting information is printed and execution continues.

```
A+3,4,5(I;BT   A;G

4;B>A+3   5/   54

4;B>A+3   5/   -1

4;B>A+3   5/   -175
```

The trace mode can be set after a break occurs by specifying ;T.

The currently valid instruction breakpoints may be listed for inspection with the command ;B. The list has the form

n{T}loc display

for each established breakpoint, where: "n" is the breakpoint number, a "T" is printed if the trace mode is set for that breakpoint, "loc" is the break location, and "display" is the address to be displayed when the break occurs.

MEMORY SEARCHING: THE ;W AND ;N COMMANDS

The two active search commands, "e;W" and "e;N", search memory for a match or no match with expression e. Display of all matching cells (bit for bit identically) occurs in the case of ;W and of all nonmatching cells in the case of ;N. The search is carried out in the closed interval defined by the symbol table values of ;1 and ;2. The initial value of ;1 is the lowest, and ;2 the highest current user data area address. Before the test for a match is made, the word from memory is masked with a word that is the symbol table value of ;M. The initial value of ;M is all ones.

The values of ;1, ;2, and ;M are set by the commands e;1, e;2, and e;M (each followed by ⏎). In addition, the limits may be set with the single command e1, e2;L which sets ;1 to e1 and ;2 to e2.

```
A;1  }
BB;2 }   A,BB;L is equivalent

2;M  }
2;W  }   Mask bit 30 of the words between A and BB
         which have a 1 in bit 30.

A+.2/    2
A+.3/    3
A+.6/    6
A+.7/    7
A+.A/    .A
BB/      .B
.1FFFF;M   L,L+.100;L   ERR;W
L+.3/    BAL,4    ERR  }
L+.A/    BAL,4    ERR  }   All words between L
L+.D/    BAL,4    ERR  }   and L+.100 with ad-
L+.6A/   AWM,1    ERR  }   dress equal to ERR.
```

SYMBOL TABLE CONTROL: THE ;K ;S ! AND <> COMMANDS

The symbol table available to Delta after a load is completed consists of the global symbols (those defined by DEF directives) and a set of internal symbol tables (one for each element file loaded) which are stored under the element file name.

During debugging, the user always has the global symbols of the load and he may select one of the internal symbol tables by using the s;S command. It replaces, for reference purposes, any previously selected internal symbol set.

```
.B73/   LW,4   IOP+.A7 (LF)

IOP+.CB/   BAL,6   IO+.17F   IOPF;S †

IOPT2+.6/   LW,4   K34
```

Symbols may be defined by the user at any time during his debugging session. Symbols so defined are added to the set of global symbols associated with the program load.

s;S — Select the internal symbol table corresponding to EF name "s".

s(f! — Adds the symbol s to the global symbol table with the location value of the currently open cell ($ or .) and format type f. If f is omitted, symbolic instruction (R) type is assumed.

e(f<s> — Adds the symbol s to the global symbol table, with the value defined by the expression e and format code f. In addition to the normal codes, the letter K may be used to indicate constant value. If f is omitted, R is assumed.

s;K — Removes the symbol s from the symbol table. The removal is permanent if s is in the global table and temporary if s is in an internal symbol table. (It will return if the user switches to another internal symbol table and back again.)

;K — Is used to remove all symbols from the symbol table. Symbols defining instruction codes are not erased. Individual internal symbol tables are recoverable using an s;S command.

MISCELLANEOUS COMMANDS: THE ;A ;R AND ;Z COMMANDS

The commands discussed in this section cause Delta to change its normal or default modes for display and to clear specified areas of memory. All commands in this section are terminated by a carriage return.

The commands ;R and ;A are complementary to one another; they control how Delta displays location values when typing the contents of cells. The mode of display is either relative (;R) or absolute (;A). When in the relative mode, Delta looks up the location value in the symbol table and displays the symbol if one corresponds to the value; if not, it displays the symbol with next smaller value and a word offset in hexadecimal. The form value;R may be used to specify a maximum offset "value" that may be displayed in the relative mode. Offsets in excess of this will cause the display to revert to the absolute mode. If the mode is absolute (;A), then all location values are displayed as hexadecimal numbers. Note that these commands control the display of location values and not the display of the address parts of instructions contained in those locations.

;R Display Example:

```
A,A+5/   LI,1    .10

A+.1/    CW,1    K45

A+.2/    BGE     ZZZ

A+.3/    AI,1    1

A+.4/    B       A17

ZZZ/     STW,2   BR13
```

;A Display Example:

```
A,A+5/   LI,1    .10

.5CD/    CW,1    K45

.5CE/    BGE     ZZZ

.5CF/    AI,1    1

.5D0/    B       A17

.5D1/    STW,2   BR13
```

The command for zeroing memory takes the form

a,b;Z

where

a is the lower limit

b is the upper limit of memory to be zeroed. Expressions may be used for a and b. An error results if the value of b is less than that of a, or if the range does not lie within the on-line memory area.

Example:

```
A,A+5;Z

100,1;Z

? 7
```

The last line of this example is a syntax error message referring to the previous command. The upper limit was less than the lower limit and the error was discovered when Delta processed the seventh character in the command line (see "syntax errors" in the following section).

ERRORS AND ERROR MESSAGES

Errors resulting in machine traps are reported explicitly to the user and console control is returned to him to await further commands. Each message is accompanied by the location (symbolic if possible) of the offending instruction. The messages are:

NONEXIST INSTR AT ___

NONEXIST MEM REF AT ___

PRIVIL INSTR AT ___

MEM PROTECT FAULT AT ___

I/O ERR AT ___

UNIMP INSTR AT ___

FIXED ARITH OVFLW AT ___

FLOAT FAULT AT ___

DECIMAL FAULT AT ___

BAD CAL AT ___

Syntax errors are reported by the message "? n". Where n is the number of characters in the command line that Delta was processing when the error occurred. This message is sent to the user whenever Delta cannot understand the user's command syntax. It is usually simpler for the user to identify the error than for Delta to be specific about it. Some errors and the reasons for them are shown below:

X, Y, Z, 2, 7/ ? 8		(Too many commas)
'ABCDE' = ? 6		(Constant value larger than one word)
ABC;K ? 5	}	(Symbol not in symbol table)
FF;M 100,XY;L .6B;W ? 13	}	(Symbol value not found. Remainder of command string ignored.)
A,5;E ? 5	}	(Command unknown)
LW*5 ALPHA= ? 3		(Asterisk misplaced)
.3ACR/ ? 5		(Illegal character in hexadecimal number)
(B;/ ? 2		(Illegal format character)
;T ? 2	}	(No break in effect on which to set trace mode)

INDEX TO DELTA COMMANDS

11. SYMBOL SUBSYSTEM

The BTM Symbol assembler is an extended version of SDS Sigma 5/7 Symbol, described in the Symbol/Meta-Symbol Reference Manual (90 09 52). The assembler accepts source images through M:SI and creates binary and listing files through M:BO and M:LO.

Default assignments for these files are:

M:SI	User's console
M:LO	User's console
M:BO	File BOTEMPα

The Executive command for entering the SYMBOL Subsystem is:

```
!SYMBOL
```

The assembler then requests options.

```
OPTIONS:
```

The user should then enter a list of options selected from the following set.

Options	Purpose
BO	Write binary output through M:BO.
LO	Write listing output through M:LO.
CN	Include cross-reference listing in LO.
SD	Include on-line debug symbol tables in BO.

A carriage return alone requests all of the above options. If any individual options are stated, only those are used.

The following example shows a Symbol assembly with listing output to a disc file, and binary output to the default file, BOTEMPα

```
!ASSIGN M:LO, (FILE, CMPLO)
!ASSIGN M:SI, (FILE, CMPS)
!SYMBOL
OPTIONS: BO, LO, CN
**END OF ASSEMBLY**
```

The default assignment for the assembly listing is the user terminal. The listing is reformatted in this instance due to the width of the console carriage. Each source line produces two listing lines:

1. The source image.
2. The line number and the object code portion of the normal listing.

If the source input file is sequenced according to EDIT conventions, the sequence number will also be displayed in decimal format on the second line.

If the assembly listing is not being displayed on the console, any errors found in the assembly are displayed on both the console and the listing file. The console display is in the form of

1. The offending source line.
2. The normal Symbol error indicator, positioned under the image.
3. The line number, object code produced, and sequence number of the record (if it had one).

12. SUPERVISORY SUBSYSTEM (SUPER)

SUPER is available as a batch processor and as an on-line subsystem. It provides the capability to add or delete legal users, list all legal users, and to obtain the passwords for all files in a specified account. SUPER can only be accessed while running with account :BTM.

The commands to SUPER are described below. Records containing commands must begin in column 1. Some of the command records can be followed by records specifying concerned accounts or users. Records specifying users begin in column 2 and each group of records is terminated by an EOD. These records have the general format:

↑name,acct,pass,disc

Name,acct, and pass have the same meaning as in LOGIN; disc is a hexadecimal number specifying the maximum disc space that may be used (in granules). Functions that do not require pass or disc parameters may omit them. SUPER reads the control device (Teletype if on-line) and outputs to M:LO. The batch version lists all of the input records.

SUPER COMMANDS

U[SERS] (Authorize on-line users)

The specification records that follow cause the indicated "name,acct,pass" combinations to be added to the accounting log. Should such a user already be in the log, the fact is noted and the specification record is ignored. If a password is not specified at this time, it will not be required for log-in.

K[ILLUSERS] (Cancel on-line access)

All users mentioned on subsequent specification records are removed from the accounting log and will thus be denied access to the system. If a summary of the user's statistics is desired, it should be requested prior to the command removing his entry from the accounting log.

S[TATS] (Summarize accounting totals)

The pass and disc options are not meaningful on this command. Specification records should be of one of the following formats:

↑name,acct

↑,acct

The first form summarizes the statistics for the individual user, while the second form summarizes the statistics for all users in the account. These summaries are output to the M:LO device.

Should the STATS command not be followed by any specification records, the entire accounting log will be summarized by account.

D[ELSTATS] (Initialize statistics)

For all account or name-account specifications following, the statistics are cleared to zero, with the exception of maximum disc space and disc space used.

L[IST] [acct] (List users)

This command does not require subsequent specification records. If the LIST command specifies an account, all users outhorized under that account are summarized. If the LIST command does not specify an account, the program summarizes all users authorized for access to the system.

P[ASSWORDS] acct (File summary)

This command requires no specification records. On encountering this command, the processor will list all file names, with their passwords, that are in the specified account. This capability allows the installation supervisor to remove entries in the file management system. If a file has no password, ***NONE*** is listed. If the password cannot be represented by printable graphics, it is listed in hexadecimal format. Synonymous files are ignored.

!EOD (End job)

Processing terminates when an !EOD record is encountered while the program is expecting a control record.

APPENDIX A. BPM SYSTEM CALs

Programs executing at subsystem and user levels may do RAD or console I/O through use of standard CAL1 operations as described in the BPM Reference Manual (90 09 54). The following table summarizes the CALs available to BTM users.

Call	FPT Code	Function	Comments
CAL1,1	X'01'	M:REW	Allowed
	X'02'	M:WEOF	Ignored
	X'03'	M:CVOL	Ignored
	X'04'	M:DEVICE (PAGE)	Ignored
	X'05'	M:DEVICE (VFC)†	Allowed
	X'06'	M:SETDCB	Allowed
	X'0B'	M:DEVICE (DRC)	Ignored
	X'0C'	M:RELREC	Allowed
	X'0D'	M:DELREC	Allowed
	X'0F'	M:TFILE	Allowed
	X'10'	M:READ†	Allowed (wait is implied)
	X'11'	M:WRITE†	Allowed (wait is implied)
	X'12'	M:TRUNC	Allowed
	X'14'	M:OPEN	Allowed. The ASSIGN image, if given, overrides any DCB options. Parameters from the FPT, in turn, override these. Only four files may be open at one time.
	X'15'	M:CLOSE	Allowed
	X'1C'	M:PFIL	Allowed
	X'1D'	M:PRECORD	Allowed
	X'20'	M:DEVICE (LINES)	Ignored
	X'21'	M:DEVICE (FORM)	Ignored
	X'22'	M:DEVICE (SIZE)	Allowed
	X'23'	M:DEVICE (DATA)	Ignored
	X'24'	M:DEVICE (COUNT)	Ignored
	X'25'	M:DEVICE (SPACE)	Ignored
	X'26'	M:DEVICE (HEADER)	Ignored
	X'27'	M:DEVICE (SEQ)	Ignored
	X'28'	M:DEVICE (TAB)	Ignored
	X'29'	M:CHECK	Allowed
	X'2A'	M:DEVICE (LINES)	Ignored
	X'2B'	M:DEVICE (CORRES)	Allowed
CAL1,2			Illegal
CAL1,3			Illegal
CAL1,4			Illegal
CAL1,5			Illegal
CAL1,8	X'14'	M:TRAP	Allowed
CAL1,9 5		M:TRTN	Allowed
CAL1,9 1-3		Exit to Executive	Allowed

†Available on console I/O.

ERROR CODES FROM FILE OPERATIONS

All error codes applicable to BPM have their normal meanings. The following additional errors are possible:

Code	Error
101	Invalid DCB address.
102	Invalid FPT address.
103	Invalid FPT operation code.
104	Invalid address in FPT (e.g., illegal key buffer address).
105	Invalid error or abnormal address.
106	Invalid buffer address.
107	No terminator on file parameter list.
108	Invalid entry in variable parameter list.
109	Invalid DCB table address.
10A	Too many files open.
10B	Invalid TCB address.
10C	Malformed DCB table.
10D	DCB address not in DCB table.
10E	Variable file parameter list too large.
10F	Improper "file" option in an Open or Close.
110	Improper "function" option in DCB.
111	User attempted to use more than his available disc space.
112	Key too long.

APPENDIX B. SUBSYSTEM CONVENTIONS FOR TELETYPE INPUT

There are three CALs associated with Teletype input:

CAL3, 0 — Returns in register 0 (in EBCDIC format) the next character in the Teletype input buffer. If there is no activation character in the buffer, the calling subsystem is dismissed until an activation character is typed, at which time the CAL then completes and the subsystem resumes normally†.

CA CAL3, 2 — Changes the activation type to the value contained in register 0.

CAL3, 3 — Sets the Condition Codes to xx10 if an activation character has been read into the Teletype input buffer; otherwise, it sets the condition codes to xx00.

The action of BTM on Teletype input is dependent upon the activation type in force and the type of character input. Tables B-1 and B-2 contain a complete summary of these relationships. In all cases, however, should the input buffer become almost full, the bell will ring for each character input until the buffer is sufficiently emptied. Should the buffer become completely full, each character input will still ring the bell, but the input character will be lost and instead a (RET) will be placed over the last character in the buffer. This character will echo as a "?" and will always cause activation. The bell always rings immediately rather than in echo sequence.

Depressing the BREAK key causes any Teletype output in progress to be terminated immediately and any untyped output is lost. Any unprocessed input waiting in the Teletype input buffer is also lost.

The subsystem activation type specifies which class of characters will cause activation, as follows:

0 Activate on all characters. Do not echo.

1 Activate on all characters. Echo.

2 Activate only on punctuation, control code, (LF), or (RET). Echo.

3 Activate only on control code, (LF), (ESC) I, ↑, /, =, or (RET). Echo.

4 Activate only on (LF) or (RET). Echo.

†"Normally" means in accordance with Tables B-1 and B-2.

The echo and type of a character (see Table B-2) specifies to which activation class a character belongs and whether it is echoable. The five basic types are:

0 Nonexistent character (i.e., a character that cannot be typed in).

1 Nonactivating character.

2 Punctuation.

3 (ESC) I, ↑, /, or =.

4 (LF) or (RET).†

A basic type with the symbol # added means that the character is echoable; without a # added, the character is not echoable.

An echo type of the form @ + j, where j is an integer, means that the associated character is a special control code known to BTM. There are nine such special codes.

Table B-1. Conventions for Activation Types 0 and 1

Input	Echo (if any)	Character User Gets	Notes
Letter or digit	Same as input	Same as input	
# $: @ ↑	Same as input	Same as input	
Punctuation†	Same as input	Same as input	
Control code	Note	Same as input	
(LF) or (RET)	Same as input	Same as input	
Any other character††	None	None	Character is ignored

†"Punctuation" means: ! " % & ' ( ) * + , - . / ; < = > ? [\ ] ↑ or ←.

††"Any other character" normally means only those characters that cannot be typed in on a standard Teletype (see Appendix I).

†Echo type 4 is used only internally by BTM.

Table B-2. Conventions for Activation Types 2, 3, and 4

Input	Echo Type	Function	Erasable ?	Echo	User Gets	Notes
Letter or digit	# + 1	-	Yes	Same as input	Same as input	
# $: @ ↑	# + 1	-	Yes	Same as input	Same as input	
Punctuation[t]	# + 2	-	Yes[ttt]	Same as input	Same as input	
/ = ↑	# + 3	-	Yes[ttt]	Same as input	Same as input	
ESC R	@ + 1	Retype	No	LF RET	None	Retypes from last activation character
ESC Q	@ + 0	Acknowledge	No	! !	None	Each is returned immediately rather than in echo sequence
ESC RUB	@ + 2	Backspace[ttt]	No	←	None	
ESC I	@ + 3	Tab	Yes[tttt]	n spaces[tttt]	n spaces[tttt]	
ESC RET	@ + 4	Local new line	No	LF RET	None	
ESC X	@ + 6	Erase[ttt]	No	LF RET	None	Erases to last activation character
Any other control character	3	-	Yes[ttt]	? or none[ttttt]	Same as input	
LF	@ + 7	Line feed	No	LF RET	LF RET	User gets LF for activation types 2 and 3, RET for activation type 4
RET	@ + 7	Carriage return	No	LF RET	RET	
Any other	0	-	No	None	None	Character is ignored

[t] "Punctuation" means: ! % & ' ( ) * + , - . ; < > ? [\ ] or ←.

[tt] "Any other character" normally means only those characters that cannot be typed in on a standard Teletype (see Appendix I).

[ttt] An activation character cannot be backspaced over (or erased); thus an ESC RET immediately after an activation character is ignored.

[tttt] If no tab stop is set, a ? is echoed. The user will get a single space if he reads such an illegal tab. Each backspace erases one space of the tab; thus to erase the tab completely, n backspaces must be given.

[ttttt] If the activation type is 4, a ? is echoed; otherwise, there is no echo. The user always gets the actual input character regardless of the echo.

APPENDIX C. SUBSYSTEM CONVENTIONS FOR TELETYPE OUTPUT

There is one CAL associated with Teletype output:

CAL3, 1 This CAL takes a character (in EBCDIC format) from the lower 8 bits of register 0 and places it into the Teletype output buffer.

Associated with every possible EBCDIC character is a 7-bit USASCII conversion code and a 1-bit control flag, as follows:

Flag	Meaning
0	The corresponding USASCII character is unprintable on a standard Teletype.
1	The corresponding USASCII character is printable.

The code is the USASCII equivalent of the EBCDIC character, or 0.

Any character whose flag is 0 or whose conversion code is 0 is ignored by CAL3, 1. A conversion code of 0 implies that the EBCDIC character has no corresponding USASCII equivalent, See Appendix J for a complete list of EBCDIC-USASCII correspondences - any character that has a blank entry in the "prints as" column of this table has a 0 conversion code and thus is not printed.

APPENDIX D. BTM SYSTEM CALs

While in operation, a subsystem has a variety of ways in which it can communicate with the terminal user, the BPM file management system, and the resident BTM Executive.

BTM provides for two levels of program execution. Both use the same memory area, so they may not run concurrently; however, for each terminal in the system, an area of RAD swap storage is allocated for each type of program.

The first area, called subsystem storage, is used to maintain the current state of the particular subsystem the user has called, for example, EDIT, as it is time-sliced on the way to completion of the task.

The second area, called user storage, is used to maintain an image of the terminal user's object program after it has been loaded by the Load subsystem, and during its execution. The user level is also where the BASIC subsystem compiles and executes BASIC programs. In both these cases, while the user's programs are in execution the subsystem level monitors all abnormal conditions, thus maintaining effective control.

The resident Executive is aware at all times of the level applicable to the job being executed by any terminal. A program may only be started at user level by a subsystem; a program running at user level will always be interrupted and control given to the initiating subsystem on any occurrence of an invalid operation, trap, etc.

All of the console I/O capability set forth in the section on Teletype operation is available to both subsystems and user level programs.

The following CAL3 calls are available at the levels indicated to perform system control and memory management functions.

CAL3,4 (Executive and Subsystems)

This CAL fetches a subsystem in absolute format from a dedicated RAD area and transfers control to it.

For Executive calls, R1 contains the subsystem name table index. For subsystem calls, R1 contains the test of the subsystem name.

The subsystem name table and start address table are built at boot time when the subsystem storage area of the RAD is initialized.

CAL3,5 (Executive and Subsystems)

This CAL starts a new (user or subsystem) process on the next lower level. Register 0 must contain the first half of the PSD to be used when the new process is initialized; the format of register 0 contents must be:

Bits	Contents
0-3	Condition Code (CC)
4-7	Floating Controls (FC)
15-16	Instruction Address (IA)

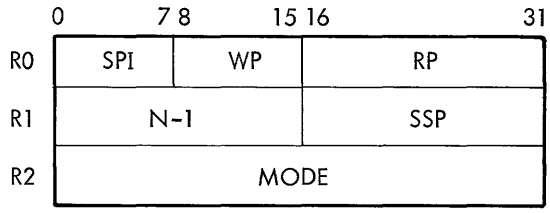
Return from the level invoked by the CAL3,5 is to the instruction following the CAL3,5 (or the instruction following the first EXU of any chain of execute instructions resulting in the execution of a CAL3,5).

CAL3,6 (Subsystem and User Level)

This CAL performs a normal return to the next higher level process (same results as ESC ESC .

CAL3,7 (Subsystems Only)

CAL3,7 is used to swap pages between the subsystem area of memory and user-level swap storage. This allows a subsystem to initialize a user-level process, or examine particular pages of the user level program in the event that control has been returned due to error. Swapping is controlled by registers 0, 1, and 2 which must be in the following format.



SSP Initial page number of subsystem area which is involved in the swap

SPI Swap consecutive pages after SSP (SPI=0) or swap SSP repeatedly (SPI=1).

N Number of pages to be transmitted.

WP Initial page number of user swap storage into which page SSP of subsystem memory will be written.

RP Initial page number of user level swap storage which is to be read into memory starting at page SSP.

With the above parameters, the CAL allows one-way transmission between memory and swap area and vice versa, or two way transmissions between groups of pages in memory and on RAD. This is controlled by the MODE parameter:

MODE Value	Transmission Mode
<0	User level swap area to subsystem memory. User pages RP through RP+N-1 overwrite subsystem pages SSP through SSP+N-1.
= 0	Two way transmission. Pages SSP through SSP+N-1 of memory are written to pages WP through WP+N-1 of user level swap storage; then pages RP through RP+N-1 of swap storage overwrite pages SSP through SSP+N-1 of subsystem memory.
> 0	Subsystem memory to user level swap storage. Subsystem pages SSP through SSP+N-1 overwrite pages WP through WP+N-1 on the RAD.

Example 1:

(0) = 00030006, (1) = 0002000A, (2) = 0

Write pages A, B, C of subsystem to pages 3, 4, 5 of user level swap storage, then read pages 6, 7, 8 of user level swap storage into pages A, B, C of the subsystem memory.

Example 2:

(0) = 01000000 (1) = 01F000F (2) = 1

Write page F of the subsystem to pages 0 through 1F of user level storage (clear user memory if page F is cleared).

CAL3,8 (Subsystems only)

This CAL specifies (in register 0) to BTM the address of the Level 2 TCB address.

CAL3,9 (Executive and Subsystems)

This CAL calls the past or current program status doubleword (PSD) for the desired level. Whenever a process is restarted via CAL3,5 the old "current" value is saved as PAST and the new value in register 0 is entered as current. Registers 2 and 3 must contain the following:

Reg.	Contents
R2	Level number
R3	Even (bit 31 = 0) if past PSD is desired
R3	Odd if current PSD is desired

CAL3,9 returns results in registers 0 and 1 as follows:

Reg.	Contents
R0	CC, FC, IA (same as in CAL3,5)
R1	Error code (see table below)

Error Code	Meaning
0	Normal return caused by ESC ESC or CAL3,6
1	Nonexistent instruction
2	Nonexistent memory address
3	Privileged instruction
4	Memory protection violation
5	Unimplemented instruction
6	Push-down stack limit reached
7	Fixed-point arithmetic overflow
8	Floating-point fault
9	Decimal arithmetic fault
10	Improper arguments to a call
11	Illegal call
12	Read error on RAD during transfer

CAL3, 10 (Subsystems)

This CAL transfers the error message associated with the most recent CAL1 error into words 20_{16} through 37_{16} of subsystem memory.

CAL3, 14 (Subsystem and User Level)

If R0 is positive or zero, this CAL returns the maximum number of pages that can be activated in user memory (and subsystem memory), in register 0. This enables users and subsystems to allocate memory appropriately.

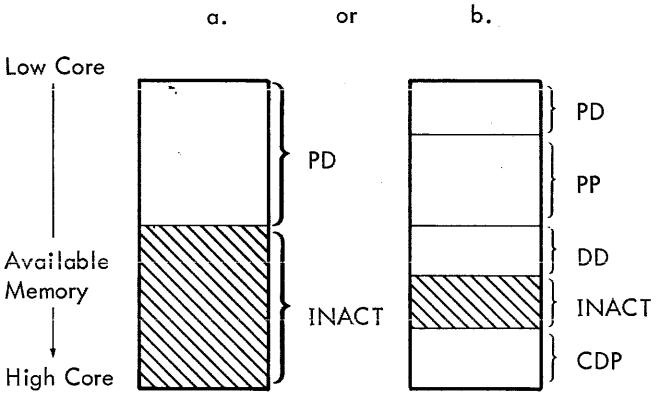
CAL3, 11 (Subsystems) and

CAL3, 13 (User Level Programs)

These two CALs have the same functions for the two levels of operation. They allow a subsystem or subsystem-initiated user level program to describe memory in such a way as to make swapping most efficient.

CAL3, 14 will supply a program with the total amount of memory it may use. Initially, the Monitor will only swap in and out the actual amount of memory needed to contain the program. At any time the program may change its swap size within the maximum allowed, through use of CAL3, 11 or CAL3. 13.

A program describes itself in one of the two following ways:

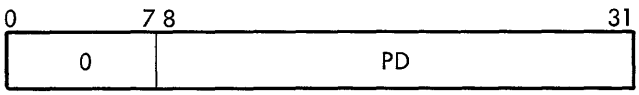


where

Area	Contents
PD ≥ 1	Program data pages swapped in and out.
PP ≥ 0	Pure procedure pages (unmodified during execution), swapped in only.
DD ≥ 0	Dynamic data pages swapped in and out.
INACT ≥ 0	Inactive pages, not swapped.
CDP ≥ 0	Common data pages, building down from top of available memory; swapped in and out.

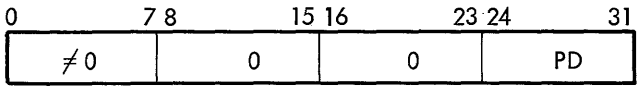
a. This is the simplest case and exists when a subsystem is started. Size may be changed by appropriate CAL with PD communicated in R0. INACT is, by default, everything other than PD.

(R0)

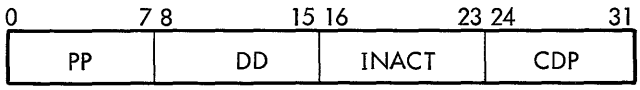


b. This case allows the most efficient description of the program. Page counts of the various types are computed, and communicated to the Monitor in registers 0 and 1 as follows:

(R0)



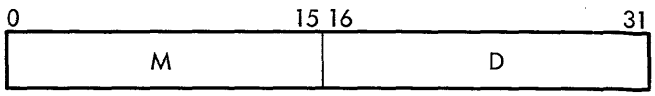
(R1)



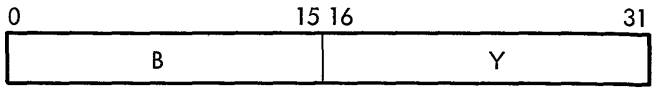
CAL3, 15 (Subsystem and User Level)

This CAL returns the date and time in registers 0-2 in the following format:

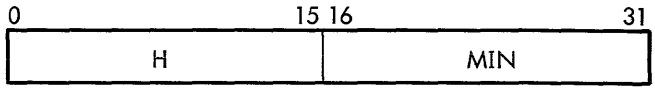
R0



R1



R2



where

M = month 1-12
D = day 1-21
B = blanks
Y = last two digits of year
H = hour 0-23
MIN = minute 0-59

} Two EDCBIC digits each

APPENDIX E. SUBSYSTEM INTERFACE

There are several simple rules that BTM expects each subsystem to follow. These rules apply to the way a subsystem is coded and the way a subsystem is loaded into the BTM system.

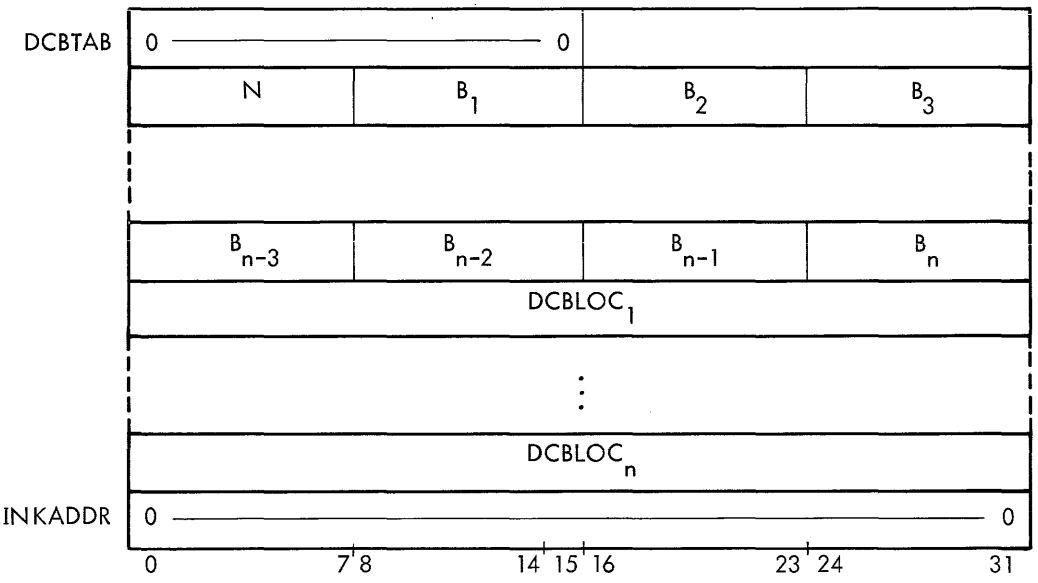
CODING REQUIREMENTS

The first 40_{16} locations in any subsystem should contain the following data.

Location	Data Description
0	Contains the word address of the first word of the subsystem's TCB (Task Control Block).
1	=0
⋮	⋮
8	=0
9	Contains the word address of the normal entry point for the subsystem. The entry point for a "PROCEED" is assumed to be this address plus one.
A_{16}	=0
⋮	⋮
F_{16}	=0
10_{16}	Reserved for use by Monitor.
⋮	⋮
$3F_{16}$	Reserved for use by Monitor.

When a subsystem is entered, R1 contains the COC line number (in binary), R4 and R5 contain the log-in account designation (in EBCDIC, left-justified and blank filled), and R2 contains the terminal job entry flag (0 indicates that the console is excluded from the system, and a value of 1 - F indicates the maximum priority).

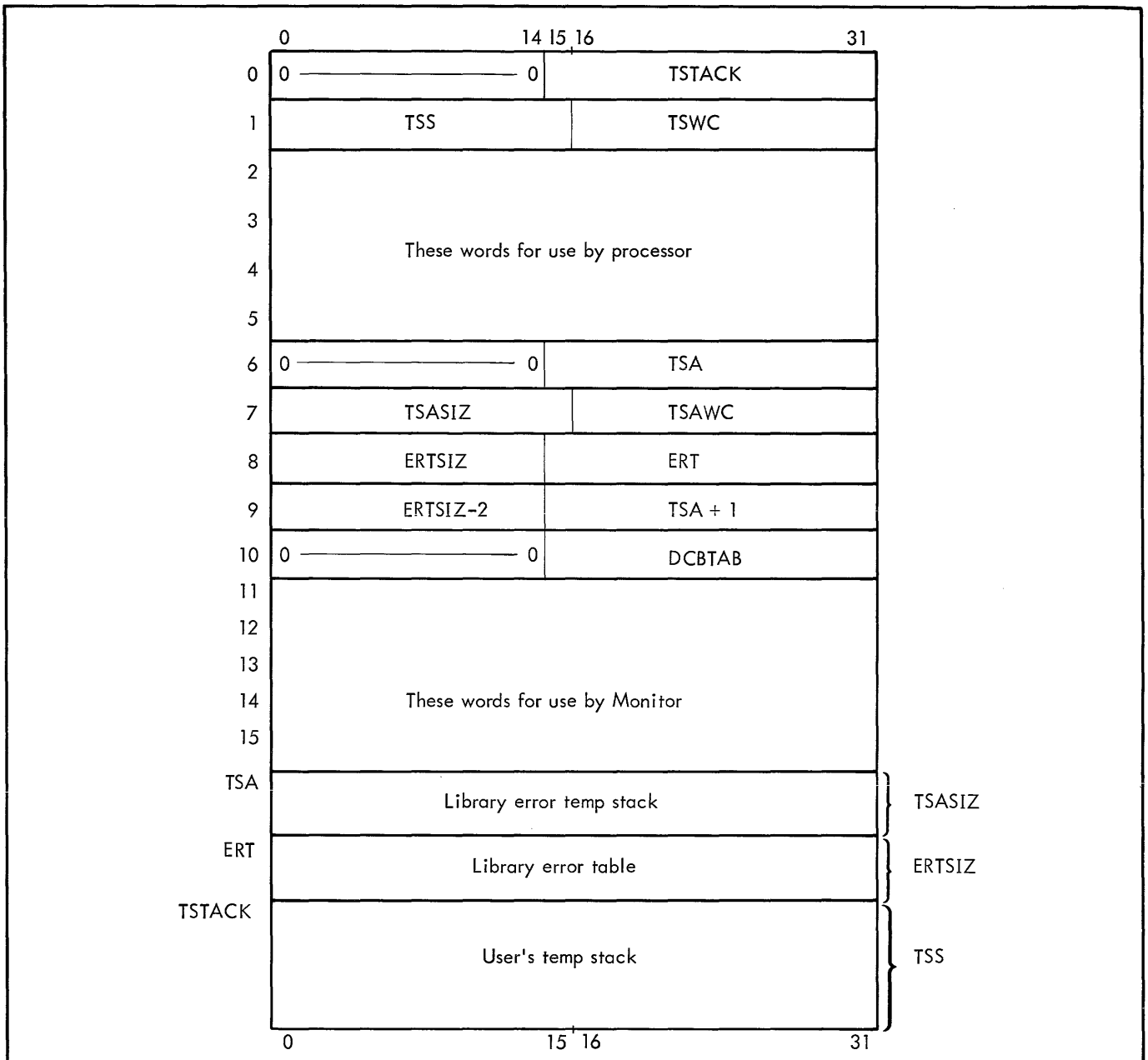
Included with the ROMs that constitute a subsystem there must appear a DCB name table (see Figure E-1) pointed to by word 10 of the TCB (see Figure E-2), and all the necessary DCBs assembled with protection type 00.



where

LINKADDR	is the address of the location provided for storing a return address.
N	indicates the number of characters in the DCB name.
B_1-B_n	indicates the EBCDIC name of the DCB.
DCBLOC	is the address of the first word location of the DCB.

Figure E-1. DCB Name Table



where

TSTACK	is the address of the current top of the user's temp stack.
TSS	indicates the size, in words, of the user's temp stack.
TSWC	is the temp stack word count giving the current number of words in the user's temp stack.
TSA	is the address of the temp stack used by the library error package.
TSASIZ	indicates the size, in words, of the temp stack used by the library error package.
ERTSIZ	indicates the isze, in words, of the error table used by the library error package.
ERT	is the address of the error table used by the library error package.
DCBTAB	is the address of a table of names and addresses of all of the user's DCBs. This table has the form shown in Figure E-1.

Figure E-2. TCB Format

There are several CALs available (for use by a subsystem only) to provide service for the subsystems. These CALs mainly deal with changing the size of swap areas for subsystems and users, finding out the amount of core currently being used by subsystem and user, and performing swaps between user and subsystem memory. They are described in Appendix D, "BTM System CALs".

The format for DCBs is essentially the same as described in the BPM Reference Manual, 90 09 54. The one difference is the ability to assign DCBs to a user's console. This is done by setting the ASN (bits 28-31) in word 0 of a DCB to 5 (ASN is the file assignment type indicator – 0 means null, 1 means FILE, 2 means LABEL, 3 means DEVICE, 4 means CORE, 5 means user's console).

LOADING REQUIREMENTS

The ROMs that comprise the subsystem should be loaded into the system by running the following job under BPM.

```
!JOB      :SYS,user,1
!LOAD     (ABS),(BIAS,loc),(NOTCB),(PERM),;
!         (LMN,name:)
          .
          .
          .
          subsystem ROMs
          .
          .
          .
!FIN
```

where

name is the subsystem name to which console users will refer.

loc is the hexadecimal value of the base of the online memory area.

When the system is booted from the RAD, the load module will be initialized in absolute swap storage as a subsystem callable by the first two characters of the name.

APPENDIX F. BTM SCHEDULING

The BTM resident Executive is an extension of the Batch Processing Monitor. It is not organized as a real-time task, but is essentially a subroutine called by the BPM at intervals governed by one of the hardware clocks. The following is a short discussion of the manner in which the machine is shared.

Interrupts generated by the COC are always handled by the BTM Executive, and it is the receipt of the initial break character that causes the BTM system to start accepting input from a terminal. Input characters for all terminals are buffered in the resident Executive, until the on-line user's job is able to fetch them.

When no consoles are actively processing on-line jobs, the clock guarantees that the BTM Executive will be allowed to scan the active lines once every batch quantum to determine whether any user requires the services of a nonresident BTM function. If so, the swap-in of that processor is started and batch processing continues.

After this point, and while several on-line users are sharing the machine, the scheduling of on-line and batch jobs is performed in an asynchronous manner, with the clock only enforcing certain maximum and minimum time intervals. Figure F-1 illustrates this, and assumes that the swap-in of an on-line job has just been completed, and that several on-line users exist and are being serviced.

The on-line quantum (OLQ) is a SYSGEN parameter (default 100 ms) which specifies the maximum length that a subsystem may run before a batch job is serviced. The on-line task may not be dismissed in the middle of a RAD I/O action through the file management system, and therefore under certain circumstances may run slightly longer. Should the subsystem request Teletype input when none is available, it is dismissed immediately. An actual on-line time slice may vary from a minimum of 2 or 3 ms (if dismissed at once) to a few ms more than the specified quantum (to allow I/O to be completed). Receipt of the necessary Teletype input marks the subsystem for activation.

At dismissal time, all I/O actions necessary to complete the swapout are queued up as direct access, and the batch job is started. (If, at the end of a user's on-line quantum, no other user in the system is ready for execution, the swap-out I/O is not started. The user's task in memory will receive subsequent on-line quanta, as it is able to use them, until another on-line user requires service. In this single-user case, successive batch quanta would be terminated by the clock.)

The completion of of the SIO1 operation triggers SIO2. The swap-in I/O requests are queued up and the batch job continues. (From the diagram, it is seen that if the swapping RAD and the batch file management RAD have the same

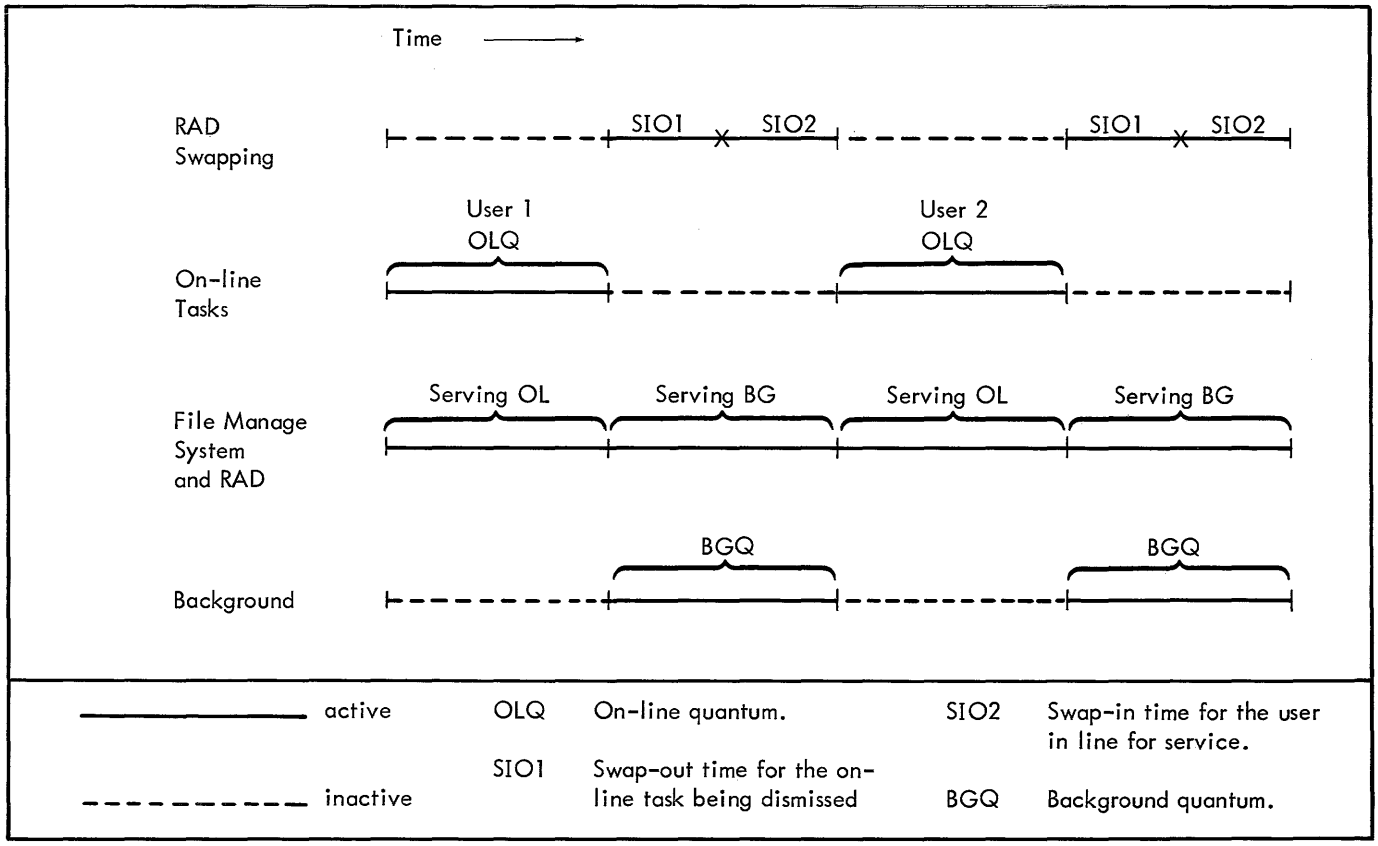


Figure F-1. On-Line Time-Sharing

controller, there is a severe conflict when the batch job requires RAD I/O or Monitor overlays.)

The background quantum (BGQ) is a SYSGEN parameter (default 200 ms) that specifies the length of time a batch job may run before servicing the next on-line task. However, it is almost always the case in practice that the background quantum is set by the length of SIO1 and SIO2. Batch processing will never be interrupted before the swap is done, and with the 7204 RADs this is on the order of .8-1 second.

The intervals SIO1 and SIO2 vary greatly for each subsystem, depending on size and on how the subsystem has described itself with the memory management CALs. This variance is greatest with the Model 7204 and 7232 RADs, because of their lower transfer rates.

When the high-speed RAD is in the system for swapping, the background quantum is used as the minimum time during which batch jobs must run, to prevent batch jobs from being effectively locked out by on-line jobs with short swap and long execution times.

With the high-speed RAD, it is seen that the most that can be done for optimizing response time is to set the SYSGEN value of the background quantum very low, so that the on-line programs run on a demand basis. In this case, batch jobs would utilize no more than the "busy time" of the swapping RAD.

APPENDIX G. BTM MACHINE OPERATION

Machine operation with BTM is virtually identical to that of the standard BPM system. Perhaps the most noticeable difference occurs when the system is booted from disc, for example, to start operation after loading the disc from tape.

After the normal BPM header information is typed, the BTM Executive must find all of the on-line processors in the system account and initialize them in absolute swap storage. This will typically take 4-5 seconds, after which the resident Executive types a second message:

BTM SWAP AREA IS FROM xxxx TO END OF DISC ndd.

where

xxxx is the first disc address of absolute swap area.

ndd is the channel and device designator of the swapping RAD.

It is not necessary to wait for the second message before initiating batch operations. Although batch jobs may be started immediately, consoles will not respond to the ESC ESC input until the second message has been typed.

As on-line users enter the system, this will be noted on the operator's console by a message similar to the following:

* CONSOLE xx LOG ON.

where

xx is the hexadecimal representation of the user's COC line number.

If the BTM accounting package is implemented in the system, the log-in message will be accompanied by the user's name and account number identification.

When a user leaves the system, the fact will be noted on the operator's console by the message

* CONSOLE xx LOG OFF.

KEY-INS

The machine operator exercises ultimate control over the system with the following key-ins, which are given in the standard BPM fashion.

!BTMM [,xx] text

This key-in causes the text message to be printed on all consoles (or console xx) without affecting their current operation. For example:

!BTMM SYSTEM DOWN IN FIVE MINUTES.

!BTMX[xx]

This key-in terminates the activity of the indicated console. If no specification is given, all active consoles are deactivated. The message

BTM HALTED BY OPERATOR
time date

is output to all consoles currently active. Background operations are started if not already active; this ensures that a quiescent state will be reached (see "System Save for Restart"). This key-in, when used to deactivate all consoles, also disables the time-sharing portion of the system. Attempts to activate consoles are met by the message

BTMQ

!BTMS

When the system is first booted from disc, the time-sharing portion of the system is active. If the !BTMX key-in has been used to terminate operations, they may be restarted with this key-in. Following the key-in, the consoles resume service, and the message

BTM IS OPERATIONAL

is broadcast to all consoles still connected to the system.

TERMINAL JOB INSERTION

When BTM is being supported by a symbiont BPM system, on-line users are able to insert jobs into the batch queue. The machine operator will be alerted, when this occurs, by the message

* JOB yyy (p) FROM CONSOLE xx.

where

yyy is the JOB ID, and p the priority.

This will aid him in dispatching output.

SYSTEM SAVE FOR RESTART

In order to save user's files and provide a restart position, it is good policy to dump the file area of secondary storage onto tape periodically. This is also necessary whenever the system is to be shut down and the RAD will not be kept intact (e.g., preventive maintenance or use of another operating system). If the system is to be shut down but RAD storage will be preserved, it is not necessary to perform a dump; the system can be initiated from secondary storage with the standard boot procedure.

Either of the above operator actions, booting from RAD or dumping to tape, can be done only when the entire system is quiescent.

Conditions required for the system to become quiescent are:

1. BPM must be in the "wait state".
2. No symbiont may be active.
3. No symbiont files can exist (input or output).
4. All consoles must be logged off. This can be assured with the BTMX key-in.

When these conditions are met, the Monitor prints the message

SYSTEM IS QUIESCENT.

This must be the last entry on the operator's log if there is to be a restartable system. Dumping to tape or booting from disc under any other circumstances cripples the system.

The portions of the system that must be saved so the system may be restarted are PSA (Permanent System Area) and PFA (Permanent File Area). Neither PER (symbiont storage) nor swap storage need be saved, since they are scratch areas.

SYSTEM ERROR RECOVERY

From the above discussion, it is obvious that in the event of failure (for any reason) the system may not be initiated from the RAD via a standard boot process. If the machine is hung up due to an irrecoverable error in the middle of a job, it is impossible for the Monitor to attain quiescence.

It is possible to recover permanent files and symbiont output for all but the jobs that were executing at the time of failure, using the recovery procedure provided with the Power-on/Power-off routing. This procedure does the following:

1. Saves crucial parts of HGP (disc allocation map) and ACNCFU (pointer to account directory) on disc.
2. Stores the symbiont file directory in high memory.
3. Simulates a disc boot.
4. Moves the symbiont file directory from high memory back to its proper place.

If symbionts were active at the time of the failure, the following precautions should be taken before attempting recovery:

1. Do a DISPLAY key-in.
2. If any output files still exist for the batch job that was executing at the time of failure, delete them. If any input files were partially created by input symbionts. delete them by means of a DELETE key-in.
3. Do an S key-in (see BPM Reference Manual) to process outstanding input files. Do not initiate input symbionts until these jobs are processed, as job IDs assigned by the system start with 0 and conflicts may occur.

Some disc space is lost in the recovery procedure. Permanent file area for all currently open output files will be lost. Symbiont disc storage for symbiont files currently being output, and for symbiont files currently being read by the system will be lost.

To initiate the recovery procedure, the operator simply presses SYSTEM RESET and moves the RUN-IDLE switch to RUN (the reset will have put the machine in the idle state).

APPENDIX H. TIMING

The time required to process a single Teletype input character is as follows:

1. If the character can be echoed immediately, the processing time is

   177.0 + Q + R + 17.5/n μs)

   (min: 257.4 μs — max: 300.5 μs)

2. If the echo must be deferred, the processing time is

   219.6 + Q + R + 27.9/m μs

   (min: 310.4 μs — max: 353.5 μs)

where

$$Q = \left\{\begin{matrix}31.8 \text{ if the buffer is empty} \\ 43.6 \text{ if the buffer is not empty}\end{matrix}\right\}$$ At the time of the input interrupt

$$R = \left\{\begin{matrix}31.1 \text{ if the buffer is empty} \\ 62.4 \text{ if the buffer is not empty}\end{matrix}\right\}$$ At the time of the input interrupt

and

n = The number of characters input simultaneously by all Teletypes.

m = The number of characters input since the last activation character but not echoed.

APPENDIX I. USASCII TO EBCDIC CONVERSION

USASCII Code		Teletype Character		EBCDIC Character		Echo[4] and Type
Hex.	Octal	Char[1]	Key[2]	Hex.	Prints as[3]	
00	00	(NUL)	P[CS]	00		0
01	01	(SOH)	A[C]	01		1
02	02	(STX)	B[C]	02		1
03	03	(ETX)	C[C]	03		1
04	04	EOT	D[C]	04		1
05	05	WRU (ENQ)	E[C]	09		1
06	06	RU (ACK)	F[C]	06		1
07	07	BELL (BEL)	G[C]	07	bell	#+1
08	10	(BS)	H[C]	08		1
09	11	TAB (HT)	I[C]	05		3
0A	12	LF	LINE FEED	25	line feed	@+7
0B	13	VT	K[C]	0B		1
0C	14	FORM (FF)	L[C]	0C		@+1
0D	15	CR	RETURN	15	carriage return	@+7
0E	16	(SO)	N[C]	0E		1
0F	17	(SI)	O[C]	0F		1
10	20	(DLE)	P[C]	10		1
11	21	(DC1)	Q[C]	11		1
12	22	TAPE (DC2)	R[C]	12		1
13	23	X-OFF (DC3)	S[C]	13		1
14	24	(DC4)	T[C]	14		1
15	25	(NAK)	U[C]	0A		1
16	26	(SYN)	V[C]	16		1
17	27	(ETB)	W[C]	17		1
18	30	(CAN)	X[C]	18		1
19	31	(EM)	Y[C]	19		1
1A	32	(SS)	Z[C]	1A		1
1B	33	(ESC)	K[CS]	1B		1
1C	34	(FS)	L[CS]	1C		1
1D	35	(GS)	M[CS]	1D		1
1E	36	(RS)	N[CS]	1E		1
1F	37	(US)	O[CS]	1F		1
20	40	blank	SPACE BAR	40	blank	#+1
21	41	!	1[S]	5A	(!)	#+2
22	42	"	2[S]	7F	(")	#+2
23	43	#	3[S]	7B	#	#+1
24	44	$	4[S]	5B	$	#+1
25	45	%	5[S]	6C	%	#+2
26	46	&	6[S]	50	&	#+2
27	47	'	7[S]	7D	'	#+2
28	50	(	8[S]	4D	(	#+2
29	51	)	9[S]	5D	)	#+2
2A	52	*	:[S]	5C	*	#+2
2B	53	+	;[S]	4E	+	#+2
2C	54	,	,	6B	,	#+2
2D	55	-	-	60	-	#+2
2E	56	.	.	4B	.	#+2
2F	57	/	/	61	/	#+3
30	60	0	0	F0	0	#+1
31	61	1	1	F1	1	#+1
32	62	2	2	F2	2	#+1
33	63	3	3	F3	3	#+1
34	64	4	4	F4	4	#+1
35	65	5	5	F5	5	#+1
36	66	6	6	F6	6	#+1
37	67	7	7	F7	7	#+1
38	70	8	8	F8	8	#+1
39	71	9	9	F9	9	#+1
3A	72	:	:	7A	:	#+1
3B	73	;	;	5E	;	#+2
3C	74	<	,[S]	4C	<	#+2
3D	75	=	-[S]	7E	=	#+2
3E	76	>	.[S]	6E	>	#+2
3F	77	?	/[S]	6F	(?)	#+2
40	100	@	P[S]	7C	@	#+1

USASCII Code		Teletype Character		EBCDIC Character		Echo[4] and Type
Hex.	Octal	Char[1]	Key[2]	Hex.	Prints as[3]	
41	101	A	A	C1	A	#+1
42	102	B	B	C2	B	#+1
43	103	C	C	C3	C	#+1
44	104	D	D	C4	D	#+1
45	105	E	E	C5	E	#+1
46	106	F	F	C6	F	#+1
47	107	G	G	C7	G	#+1
48	110	H	H	C8	H	#+1
49	111	I	I	C9	I	#+1
4A	112	J	J	D1	J	#+1
4B	113	K	K	D2	K	#+1
4C	114	L	L	D3	L	#+1
4D	115	M	M	D4	M	#+1
4E	116	N	N	D5	N	#+1
4F	117	O	O	D6	O	#+1
50	120	P	P	D7	P	#+1
51	121	Q	Q	D8	Q	#+1
52	122	R	R	D9	R	#+1
53	123	S	S	E2	S	#+1
54	124	T	T	E3	T	#+1
55	125	U	U	E4	U	#+1
56	126	V	V	E5	V	#+1
57	127	W	W	E6	W	#+1
58	130	X	X	E7	X	#+1
59	131	Y	Y	E8	Y	#+1
5A	132	Z	Z	E9	Z	#+1
5B	133	{[} ([)	K[s]	4F	[	#+2
5C	134	{\} (\)	L[s]	4A	(¢)	#+2
5D	135	{]} (¬)	M[s]	5F	(¬)	#+2
5E	136	↑ (^)	N[s]	6A		#+3
5F	137	← (-)	O[s]	6D	(-)	#+2
⋮	⋮	⋮	⋮	⋮	⋮	0
⋮	⋮	⋮	⋮	⋮	⋮	
7E	176	ESC	ESCAPE	1B		0
7F	177	DEL	RUBOUT	FF		0

Notes:

1. The forms in parentheses appear only on SDS Teletype #7015, whereas the unparenthesized forms appear on the specified keys of all standard model Teletypes. (Some models lack the ESCAPE key). The forms in braces print when the specified key is depressed, but they do not appear on the keys.
2. Superscript c indicates use of the CTRL key; superscript s indicates use of the SHIFT key.
3. The forms in parentheses are contained in the SDS 63- and 89-graphic character sets but not in the standard 57-graphic character set. On printers equipped with only the standard character set, these forms will print as blanks.
4. The echo and type specifies the echoability and activation type of a character. See Appendix B for a more complete description.

The Teletype character mnemonics have the following meanings:

ACK	Acknowledge	ENQ	Enquire	NAK	Negative acknowledge
BEL	Bell	EM	End of medium	NUL	Null
BS	Backspace	EOT	End of transmission	RS	Record separator
CAN	Cancel	ESC	Escape	SI	Shift in
CR	Carriage return	ETB	End of transmission block	SO	Shift out
DC1	Device control 1	ETX	End of text	SOH	Start of header
DC2	Device control 2	FF	Form feed	SS	Start of special sequence
DC3	Device control 3	FS	File separator	STX	Start of text
DC4	Device control 4	GS	Group separator	SYN	Synchronize
DEL	Delete	HT	Horizontal tab	US	Unit separator
DLE	Data link escape	LF	Line feed	VT	Vertical tab

APPENDIX J. EBCDIC TO USASCII CONVERSION

EBCDIC Code		Teletype Character		USASCII Code	
Hex.	Prints as[3]	Char[1]	Key[2]	Hex.	Octal
00		(NUL)	P^{cs}	00	00
01		(SOH)	A^{c}	01	01
02		(STX)	B^{c}	02	02
03		(ETX)	C^{c}	03	03
04		EOT	D^{c}	04	04
05		TAB (HT)	I^{c}	09	11
06		RU (ACK)	F^{c}	06	06
07	bell	BELL (BEL)	G^{c}	07	07
08		(BS)	H^{c}	08	10
09		WRU (ENQ)	E^{c}	05	05
0A		(NAK)	U^{c}	15	25
0B		VT	K^{c}	0B	13
0C		FORM (FF)	L^{c}	0C	14
0D	carriage return	CR	RETURN	0D	15
0E		(SO)	N^{c}	0E	16
0F		(SI)	O^{c}	0F	17
10		(DLE)	P^{c}	10	20
11		(DC1)	Q^{c}	11	21
12		TAPE (DC2)	R^{c}	12	22
13		X-OFF (DC3)	S^{c}	13	23
14		(DC4)	T^{c}	14	24
15	carriage return	CR	RETURN	0D	15
16		(SYN)	V^{c}	16	26
17		(ETB)	W^{c}	17	27
18		(CAN)	X^{c}	18	30
19		(EM)	Y^{c}	19	31
1A		(SS)	Z^{c}	1A	32
1B		(ESC)	K^{cs}	1B	33
1C		(FS)	L^{cs}	1C	34
1D		(GS)	M^{cs}	1D	35
1E		(RS)	N^{cs}	1E	36
1F		(US)	O^{cs}	1F	37
:	:	:	:	:	:
25	line feed	LF	LINE FEED	0A	12
:	:	:	:	:	:
40	blank	blank	SPACE BAR	20	40
:	:	:	:	:	:
4A	(¢)	{\} (\)	L^{s}	5C	134
4B	.	.	.	2E	56
4C	<	<	,^{s}	3C	74
4D	(	(	8^{s}	28	50
4E	+	+	;^{s}	2B	53
4F	|	{[} (|)	K^{s}	5B	133
50	&	&	6^{s}	26	46
:	:	:	:	:	:
5A	(!)	!	1^{s}	21	41
5B	$	$	4^{s}	24	44
5C	*	*	:^{s}	2A	52
5D	)	)	9^{s}	29	51
5E	;	;	;	3B	73
5F	(¬)	{]} (¬)	M^{s}	5D	135
60	-	-	-	2D	55
61	/	/	/	2F	57
:	:	:	:	:	:
6A		↑ (^)	N^{s}	5E	136
6B	,	,	,	2C	54
6C	%	%	5^{s}	25	45

EBCDIC Code		Teletype Character		USASCII Code		EBCDIC Code		Teletype Character		USASCII Code	
Hex.	Prints as[3]	Char[1]	Key[2]	Hex.	Octal	Hex.	Prints as[3]	Char[1]	Key[2]	Hex.	Octal
6D	(-)	← (-)	0^s	5F	137	D6	O	O	O	4F	117
6E	>	>	$.^s$	3E	76	D7	P	P	P	50	120
6F	(?)	?	$/^s$	3F	77	D8	Q	Q	Q	51	121
⋮	⋮	⋮	⋮	⋮	⋮	D9	R	R	R	52	122
7A	:	:	:	3A	72	⋮	⋮	⋮	⋮	⋮	⋮
7B	#	#	3^s	23	43	E2	S	S	S	53	123
7C	@	@	P^s	40	100	E3	T	T	T	54	124
7D	'	'	7^s	27	47	E4	U	U	U	55	125
7E	=	=	$-^s$	3D	75	E5	V	V	V	56	126
7F	(")	"	2^s	22	42	E6	W	W	W	57	127
⋮	⋮	⋮	⋮	⋮	⋮	E7	X	X	X	58	130
C1	A	A	A	41	101	E8	Y	Y	Y	59	131
C2	B	B	B	42	102	E9	Z	Z	Z	5A	132
C3	C	C	C	43	103	⋮	⋮	⋮	⋮	⋮	⋮
C4	D	D	D	44	104	F0	0	0	0	30	60
C5	E	E	E	45	105	F1	1	1	1	31	61
C6	F	F	F	46	106	F2	2	2	2	32	62
C7	G	G	G	47	107	F3	3	3	3	33	63
C8	H	H	H	48	110	F4	4	4	4	34	64
C9	I	I	I	49	111	F5	5	5	5	35	65
⋮	⋮	⋮	⋮	⋮	⋮	F6	6	6	6	36	66
D1	J	J	J	4A	112	F7	7	7	7	37	67
D2	K	K	K	4B	113	F8	8	8	8	38	70
D3	L	L	L	4C	114	F9	9	9	9	39	71
D4	M	M	M	4D	115	⋮	⋮	⋮	⋮	⋮	⋮
D5	N	N	N	4E	116	FF		CD	RUBOUT	7F	177

Codes X'81' through X'A9', which represent the lower case letters, are exactly congruent to X'C1' through X'E9' and may be used interchangeably. All will print as capital letters on standard Teletypes.

Notes:

1. The forms in parentheses appear only on SDS Teletype #7015, whereas the unparenthesized forms appear on the specified keys of all standard model Teletypes. (Some models lack the ESCAPE key.) The forms shown in braces print when the specified key is depressed, but they do not appear on the keys.

2. Superscript c indicates use of the CTRL key; superscript s indicates use of the SHIFT key.

3. The forms in parentheses are contained in the SDS 63- and 89-graphic-character sets but not in the standard 57-graphic-character set. On printers equipped with only the standard character set, these forms will print as blanks.

APPENDIX K. BTM SYSTEM GENERATION

GENERAL INFORMATION

This discussion is intended only as a supplement to the SYSGEN documentation in the BPM Reference Manual, Chapters 10 and 11, to aid a person already familiar with the SYSGEN process.

The principal change in system generation has been the addition of the :BTM card, and the necessity for including on a standard BPM BI tape the ROMs required to provide the BTM functions.

The :BTM card allows the specification of system parameters that are installation dependent. It has the form:

:BTM (option), (option), . . . , (option), ;

(option), etc.

Option	Meaning	Default	Limits	Units
NUMUSERS, n	Total number of time-sharing consoles that may be in use at one time.	8	1-64	Dec.
USERSIZE, n	Size, in words, of the time-sharing memory area. The size must be a multiple of 512 (one page).	16384	12288-65536	Dec.
NUMSYSTS, n	This provides an upper limit on the number of subsystems that can be in the system. It is prudent to provide for more than the standard set, so they can be added after the system is generated. This should also be taken into account when allocating swap area.	12	10-30	Dec.
BPMQTM, n	The minimum amount of time BPM runs before time-sharing users are scheduled.	200	10-500	ms
BTMQTM, n	The maximim amount of time an on-line user may run before BPM receives another quantum.	200	50-500	ms
IBUFSIZE, n	COC input buffer size. The number of bytes that may be typed ahead before data is lost. This is also the maximum number of characters that may be typed before an activiation character is typed.	100	80-200	Dec.
OBUFSIZE, n	The number of characters which can be held in each user's COC output buffer without further program intervention.	100	80-400	Dec.
IINT, n	The location of the COC input interrupt.	60	60-13F	Hex.
OINT, n	The location of the COC output interrupt.	61	60-13F	Hex.

The BI tape must have all the normal BPM modules plus the following files:

File	Contents (ROMs)
BTMBO	BTM power on/off, initialization, and Executive routines.
BTM:BLIB	On-line FORTRAN run-time and math library.
BTMFLINT	On-line FORTRAN interface program.
BTMFORT	On-line FORTRAN compiler (cat. no. 704176).
BTMLOAD	LOAD subsystem (cat. no. 705260).
BTMEDIT	EDIT subsystem.
BTMSYMB	On-line SYMBOL (cat. no. 704158).
BTMFER	FERRET subsystem.
BTMBPM	BPM subsystem.
BTMBASIC	on-line BASIC.
BTM	SYSGEN :BTM card interpreter.

The PASS2 processor has had the segment "BTM" added to it to process the :BTM system Generation control card. The tree of the PASS2 processor is now

```
!TREE CCLOAD - PASS2CCI-MODIFY-DECBS-(DEVICE,
    SDEVICE,;

!MONITOR,DLIMIT,RJITGEN-(RESERVE,INTSR),;

!CLOCK,ABS,BTM)
```

SYSGEN OPERATIONAL INFORMATION

PASS 1

The :SELECT cards should select the additional modules needed for the particular system being generated:

BTMBO is required for any BTM system

BTM:BLIB is required for FORTRAN

BTM is required only if a PASS2 processor is to be part of the object system.

Note: The BASIC subsystem requires both floating-point and convert instructions, and FORTRAN requires floating-point, so the appropriate simulators must be included in the absence of the hardware options.

PASS 2

:STDLB and :DEVICE cards should be set up as in BPM except as discussed below.

The last :DEVICE card specifying a DC device will be used as the swap device. The last part of this device will be used for swap storage and is not available for other allocation. In addition, this RAD must have large enough capacity to provide all swap storage required. The amount of swap storage may be computed as the sum of:

Subsystem	Storage
FORTRAN	15 granules
LOAD	7 granules
EDIT	7 granules
SYMBOL	9 granules
FERRET	3 granules
BPM	3 granules
BASIC	17 granules

Temporary swap area, in granules, may be computed as:

(2*(USERSIZE/512)+4) * (NUMUSERS)

For example, 680 granules are needed for 10 users with a 16K on-line area. The definition of a granule is the BPM definition, i.e., that number of sections needed to make up one page (512 words).

The user's allocation of swap area can be checked after the system is booted. The system types out

BTM SWAP AREA IS FROM aaaa TO END OF DISC ndd.

An installation desiring to add subsystems subsequent to SYSGEN should allow ample USERSIZE when generating the system.

A card defining the communications controller address should be included as follows:

:DEVICE COndd, (HAND,COC)

:SDEVICE, :DLIMIT, :RESERVE, and :ABS cards are identical to those used with normal BPM.

:MONITOR card is identical. However, the following recommendations are made:

TSTACK,300 for symbiont system (250 for nonsymbiont).

SPOOL, CPOOL, MPOOL, SFIL, no change.

CORE The correct size must be specified; the system cannot be patched to run on a different size machine.

QUEUE This parameter depends upon the type of RAD used as the swap device.

Cat. No. 7204 - QUEUE, 80

Cat. No. 7212 or 7232 - QUEUE, 20

CFU If none is specified, 10 are supplied. With BTM, the number should be NUMUSER * 2+10.

Note: If CFU is specified, M:CPU must precede IO or IOSYM on the TREE card for M:MON.

:RESERVE should not include a RESDF specification.

:INTS, :INTR, and :TIME should not be used.

A :BTM card must be included to generate a BTM system. It generates tables that are dependent upon user area size, number of users, etc. It may appear anywhere after the :DEVICE and :SDEVICE cards.

The interpretation of the :BTM card produces a load module named M:BTM, which contains all of the variable storage needed by the resident Executive. The resident Executive (BTMBO) and M:BTM are included in the generation of the Monitor.

LOAD AND OVERLAY CARDS

M:MON should have BTMBO and M:BTM added to the list of element files. These should be added to the tree specification immediately before IORT and COOP (if included). The remainder of the tree structure is unchanged, except that the element file, PFSR should be deleted.

BPM processor control cards are unchanged.

BTM Subsystems. The LOAD cards for the BTM subsystems take the following form:

```
!LOAD (MAP), (NOTCB), (PERM), (ABS),;
!(BIAS,bias),;bias = CORE - USERSIZE in hex.
!(LMN,name), (EF, (ef))
```

The module name and element file entries have the following values for each of the subsystems:

Subsystem	LMN	EF(s)
FORTRAN	FORTRAN:	BTMFINT,BTMFORT
LOAD	LOAD:	BTMLOAD
EDIT	EDIT:	BTMEDIT
SYMBOL	SYMBOL:	BTMSYMB
SUPER	SUPER:	BTMSUPER
FERRET	FERRET:	BTMFER
BPM	BPM:	BTMBPM
BASIC	BASIC:	BTMBASIC

DEF CARD

The format of the DEF card is exactly as in BPM. If on-line FORTRAN is to be used, BTM:BLIB should be specified in the (INCL,...) list.

SYSTEM BOOT AND INITIALIZATION

System boot and initialization is exactly as in BPM. The two initialization links are preempted for special BTM initialization.

USRINIT1 initializes the COC hardware, and the swap storage and BTM tables.

USRINIT2 initializes the links to the power on/off routines, and the recovery routine.

JOB COMMAND

After the system has been generated, the first of the following two jobs must be run if on-line FORTRAN is to be used. The second JOB command is optional but is recommended, as it maximizes available disc space.

```
!JOB :BTM, ON LINELIB, F          } Creates :BLIB
!ASSIGN M:BI,(FILE,BTM:BLIB,:SYS) } file under
!LOPE (PERM,LIB)                    account :BTM
!JOB :SYS, DELETE                 } Deletes BTM:
!ASSIGN M:EI, (FILE,BTM:BLIB)     } BLIB in :SYS
:FMGE (DELETE)                    }
```

The :BLIB file in the :BTM account is used as the standard library file by the LOAD subsystem.

CREATING SUBSYSTEMS

All subsystems and libraries peculiar to BTM are introduced through use of the BPM file manage and load functions, and are independent of the actual SYSGEN process in the sense that they can be added at any time.

The user must create a :BLIB file under the :BTM account.

The LOAD subsystem uses the :BLIB file in the :BTM account as the library, although an option exists for specifying others. Currently, the library file contains only the on-line version of the FORTRAN IV-H run-time, along with the standard mathematical routines.

The on-line version of the FORTRAN IV-H run-time is obtained by replacing the following routines in the standard deck set-up (which assemble either for BTM, BCM, RBM, or BPM) with the BTM versions.

Cat. No.	Module
704216	BF:SV
704293	BF:SC
704294	BF:SW
704295	BF:GCOMS

Cat. No.	Module
704305	BF:DIAG
704306	EXIT
704307	BF:SP
704308	BF:SO
704309	BF:MXXX
704310	BF:TSNUM
704334	BF:NLOC
704387	BF:RUNIO
705293	BF:EXIT

A user wishing to create a library that will be used by the LOAD subsystem can do so by using LOPE to create a :BLIB file in his account.

For example:

```
!JOB ACCT,A,1
!ASSIGN M:EO,(FILE,NEWLIB)
!FMGE (ENTER)
:
ROMs
:
!ASSIGN M:BI,(FILE,NEWLIB)
!LOPE (PERM,LIB)
!FIN
```

The LOAD subsystem can then be directed to use this library by including the following library specification in the option list.

```
OPTIONS:    △(ACCT)
```

A subsystem is defined by loading it under the BPM system account (:SYS) and forming a load module with the name ending in a colon (:). For example:

```
(LMN,SYMBOL:)
```

The load module must conform to the interface rules laid down under "Subsystem Interface".

When the BTM Monitor is booted from disc, it searches the system account for all load modules with this naming convention and incorporates them as on-line subsystems.

The first two characters of the name become the Executive command by which the subsystem may be called, and the remainder of the name (exclusive of the colon) is echoed by the Executive. For example:

```
!SY MBOL
```

When the system is booted, the subsystem load modules are copied to a dedicated area of the RAD and referenced from there in absolute format.

The Executive service routines called by the ASSIGN, RESTORE, SAVE, and TABS commands are part of the resident Monitor and therefore do not have corresponding load modules.

COMPONENT SIZES IN A BTM SYSTEM

FIXED OVERHEAD

Fixed overhead is as follows:

BPM (B00)	7K (Nonsymbiont) - 10K (Symbiont)
BTM Exec.	4.5K

VARIABLE OVERHEAD

Tables are generated at SYSGEN time which vary with the number of users and subsystems. The formula for the number of words required is:

$$(NU(IB+OB)+3)/4+(9NS+8)/2+23(NU+1)+(5(5NU+12))/4+(3(NS+2))/4$$

where

IB = input buffer size in characters

OB = output buffer size in characters

NU = number of users

NS = number of subsystems

Figures for three typical systems are:

IB = 100, OB = 100, NS = 12

NU = 8	NU = 16	NU = 24
1K(735 words)	1.5K(1369 words)	2K(2003 words)

For safety, the numbers are rounded up to the nearest page, then combined with the fixed overhead figures. Therefore, resident requirements for an 8-user symbiont system are 17K.

BACKGROUND AND ON-LINE AREAS

These memory areas must be allocated from the remainder.

on-line = USERSIZE

background = (core size less on-line area)

2K of context area must be allocated from the on-line area.

Monitor blocking buffers, normally 1.5K-2K are allocated from the background area.

SYSGEN DECK SETUP

The following listings show system generation deck setups for two representative systems. The first deck generates a 65K, 24-user symbiont system. The second deck generates a 32K, 4-user system that permits either batch or on-line processing, but not both running concurrently.

```
*
:GENDCB (M:BI,:SYSGEN,(INSN,BTMB)
 END
 JOB :SYS,MAKER00M
 ASSIGN M:EI,(FILE,M:M0N)
 FMGE (DELETE)
 JOB :SYSGEN,SYSGEN
 ASSIGN M:BI,(INSN,BTMB)
 PASS1
:SELECT (FILE,R00T,7TAP,FBCD,I0RT,HANDLERS,T0PRT,I0,;
:PRGMLDR,TYPR,I0D,DEBUG,DUMP,EXIT,M:15,     M:16,M:17,KEYIN,M:14,;
:M:18,M:1A,M:1B,M:1C,M:1D,RDF,;
:0PNL,0BSE,M:1E,0PN,CLS,M0DIFY,CLS1,SEGL0AD,LDPRG,MEMAL0C,CALPR0C,;
:WRTF,WRTD,DUMMYCCL,LBLT,M:19,P0S,ALTCP)
:SELECT (FILE,CRD0UT,PTAP,DFBCD)
:SELECT (FILE,I0SYM,C00P,CCL0SE)
:SELECT (FILE,M:CDCB,M:BIDCB,M:CIDCB,M:EIDCB,M:SIDCB,M:B0DCB,M:C0DCB,;
:M:D0DCB,M:E0DCB,M:L0DCB,M:S0DCB,M:P0DCB,M:ALDCB,M:LLDCB,M:SLDCB,;
:M:0CDCB,M:LIDCB,M:G0DCB)
:SELECT (FILE,M:CKDCB)
:SELECT  (FILE,SSSR0M,BD0R0M)
:SELECT  (FILE,SIGMET,SIG7FDP,BPM)
:SELECT (FILE,L0PER0M,:BLIB)
:SELECT  (FILE,BTM:BLIB,BTMFINT,BTMF0RT,BTML0AD,BTMEDIT,
:         BTMSYMB,BTMDEBG,BTMFER,BTMBPM,BTMBASIC,BTMB0)
:SELECT (FILE,BTMNRES)
:SELECT (FILE,BPMSUPER,BTMSUPER)
:SELECT  (FILE,BTMDELTA)
:SELECT (FILE,CCIR00T,J0B,LIMIT,ASSIGN,L0AD,TREE,;
:TELSCPE,RUN,CCIDBUG,READBI,ENDJ0B,AB0RT)
:SELECT (FILE,LDR,IN1,PS1,IN2,PS2,ALL,EVL,WRT)
:SELECT (FILE,FILEMNGE,TPECHST,FMGEDCBS)
:SELECT (FILE,F0RTRAN)
:SELECT (FILE,FREN)
:SELECT (FILE,TAPEFCN)
:SELECT (FILE,CHKPTR0M)
:SELECT (FILE,SYMBL)
:SELECT (FILE,$M:PASS0,$M:ASSEM,$M:R00T)
:SELECT (FILE,BPMBASIC)
 E0D
 PASS2
:STDLB (C,CRA03), (0C,TYA01),(L0,LPA02),(LL,LPA02),(D0,LPA02)
:STDLB (P0,CPA04),(B0,CPA04),(LI,CRA03)
:STDLB (SI,CRA03),(BI,CRA03),(SL,LPA02)
:STDLB (S0,CPA04),(CI,CRA03),(C0,CPA04)
:STDLB (AL,CPA04),(EI,CRA03),(E0,CPA04)
:DEVICE  TYA01,(HAND,KBTI0)
:DEVICE  CRA03,(HAND,CRDIN)
:DEVICE CPA04,(HAND,CRD0UT)
:DEVICE PPA05,(HAND,PTAP)
:DEVICE PRA05,(HAND,PTAP)
:DEVICE LPA02,(HAND,PRT0UT),(PAPER,26,132)
:DEVICE 9TA80,(HAND,MTAP)
:DEVICE 9TA81,(HAND,MTAP)
:DEVICE 9TA82,(HAND,MTAP)
:DEVICE 9TA83,(HAND,MTAP)
:DEVICE 7TAE0,(HAND,MTAP)
:DEVICE 7TAE1,(HAND,MTAP)
:DEVICE DCAF0,(HAND,DISCI0),(SS,5A),(PSA,50),(PFA,1B0),(PER,00)
:DEVICE DCAF1,(HAND,DISCI0),(SS,5A),(PSA,00),(PFA,200),(PER,00)
:DEVICE DCAF2,(HAND,DISCI0),(SS,5A),(PSA,00),(PFA,100),(PER,100)
:DEVICE DCBF0,(HAND,DISCI0),(SS,100),(PSA,0),(PFA,0),(PER,0)
:DEVICE C0A10,(HAND,C0C)
:SDEVICE (LMN,ISSEG,CRA03       ),(LMN,0SSEG,LPA02,CPA04)
:M0NIT0R (TSTACK,300),(QUEUE,20),(C0RE,64),;
:         (SP00L,8),(CP00L,6),;
:         (SFIL,30),;
:         (MP00L,10),(CFU,30)
:DLIMIT  (TIME,15),(L0,100),(D0,100),(U0,60),(P0,500),;
```

```
;        (TST0RE,512),(PST0RE,200),(FP00L,2),(IP00L,2)
:RESERVE (MPATCH,044)
:ABS,1024 (L0ADER),(CCI),(METASYM),(SYMB0L),(FMGE),(F0RTRANH),;
;         (PFIL),(WE0F),(REW),(L0PE),(F0RTRAN),(C0B0L),(BASIC)
:BTM (NUMUSERS,24)
 0VERLAY (LMN,M:M0N),;
 (EF,(R00T),;
 (M:RESDF),;
 (CRD0UT),;
 (FBCD),;
 (BTMB0),;
 (BTMNRES),;
 (M:BTM),;
 (M:ABS),;
 (7TAP),;
 (PTAP),;
 (CCL0SE),;
 (C00P),;
 (M:SDEV),;
 (I0SYM),;
 (M:CPU),(M:JIT),(I0TABLE)                              ,(I0RT),;
 (HANDLERS),(T0PRT),   (PRGMLDR),(TYPR),(I0D),(DEBUG),(DUMP),;
 (M:1C),(M:1D),(RDF),(0PNL),(0BSE),(M:1E),(0PN),(CLS),(M0DIFY),;
 (EXIT),(KEYIN),(M:14),(M:15),(M:16),(M:17),(M:18),(M:1A),(M:1B),;
 (CLS1),(SEGL0AD),(LDPRG),(MEMAL0C),(CALPR0C),(WRTF),(WRTD),;
 (LBLT),(M:19),(P0S),(ALTCP)),;
 (BIAS,0),(N0TCB),(MAP),(ABS),(SL,F),(N0SYSLIB),(PERM)
 TREE R00T-;
 M:RESDF-;
 M:ABS-;
 M:SDEV-;
 M:CPU-M:JIT-I0TABLE-;
 HANDLERS-;
 I0SYM-;
 CRD0UT-;
 FBCD-;
 PTAP-;
 7TAP-;
 BTMB0-;
 M:BTM-;
 C00P-;
 I0RT-T0PRT-;
 (PRGMLDR,TYPR,I0D-BTMNRES,;
                  DEBUG-DUMP,EXIT-DUMP,M:15,M:16,M:17,KEYIN,M:14,;
 M:18,M:1A,M:1B,M:1C,M:1D,MEMAL0C,RDF-(CALPR0C-;
  (0PNL-0BSE,M:1E-0BSE,0PN-0BSE,CLS-M0DIFY-;
 CLS1,SEGL0AD-0BSE,;
         WRTF,WRTD-CCL0SE  ,LBLT,M:19,P0S,;
 ALTCP),LDPRG))
 0VERLAY (LMN,CCI),;
 (EF,(CCIR00T),(J0B),(LIMIT),(ASSIGN),(L0AD),(TREE),(TELSCPE),;
 (RUN),(CCIDBUG),(READBI),(ENDJ0B),(AB0RT),(M:DLIMIT),(M:JIT))
 (BIAS,4400),;
           (N0TCB),              (MAP),(PERM),(SL,F),(N0SYSLIB)
 ASSIGN M:EI,(FILE,M:RESDF)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:DLIMIT)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:FC0M)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:JIT)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:CPU)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,I0TABLE)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:SDEV)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:BTM)
 FMGE (DELETE)
 ASSIGN M:EI,(FILE,M:ABS)
 FMGE (DELETE)
```

```
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   FØRTRAN:),(EF,          (BTMFINT),(BTMFØRT))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   LØAD:)   ,(EF,          (BTMLØAD))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   EDIT:)   ,(EF,                    (BTMEDIT))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   SYMBØL:) ,(EF,          (BTMSYMB))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   FERRET:) ,(EF,          (BTMFER))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   BPM:)    ,(EF,          (BTMBPM))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   BASIC:)  ,(EF,          (BTMBASIC))
LØAD (MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,SUPER:),(EF,(BTMSUPER))
LØAD          (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,COOO),;
(LMN,   DELTA:)  ,(EF,          (BTMDELTA))
JØB :SYS,DELETE,F
ASSIGN M:EI,(FILE,PASS2)
FMGE (DELETE)
ASSIGN M:EI,(FILE,PASS1)
FMGE (DELETE)
JØB :SYSGEN,SYSGEN
ØVERLAY (LMN,FMGE),;
(EF,(FILEMNGE),(TPECHST),(FMGEDCBS)),;
(BIAS,4400),;
                    (MAP),(SL,F),(NØSYSLIB),(PERM)     ,(TSS,100)
ØVERLAY (LMN,LØADER),;
(EF,(ALL),(EVL),(IN1),(IN2),(LDR)          ,(PS1),(PS2),(WRT)),;
(BIAS,4400),;
          (NØTCB),(MAP),       (PERM),(SL,F),(NØSYSLIB)
LØAD (LMN,LØPE),(MAP),(NØTCB),(EF,(LØPERØM)),;
(BIAS,4400),;
(PERM)
LØAD (LMN,SUPER),(NØTCB),(MAP),(ABS),(EF,(BPMSUPER)),;
(BIAS,4400),;
(PERM)
LØAD (LMN,SYMBØL),(PERM),(MAP),;
(BIAS,4400),;
(EF,(SYMBL),;
(M:ØCDCB),(M:DØDCB),(M:GØDCB),(M:SIDCB),(M:BØDCB),(M:LØDCB),(M:CDCB) ),;
(SL,F)
ØVERLAY (EF,($M:RØØT),($M:PASSO),($M:ASSEM),;
(M:DØDCB),(M:CDCB),(M:LØDCB),(M:BØDCB),(M:GØDCB),          (M:SIDCB),;
(M:SØDCB),;
(M:CIDCB),       (M:CØDCB),(M:LLDCB),(M:ØCDCB));
                                          ,(ABS),(PERM),(SL,F),;
(BIAS,4400),;
(LMN,METASYM),            (TSS,80),(MAP),(UNSAT,(:SYSGEN))
TREE $M:RØØT=M:BØDCB=M:CDCB =M:CIDCB-M:CØDCB=M:DØDCB=M:GØDCB=;
M:SØDCB=;
M:LLDCB=M:LØDCB=M:ØCDCB=M:SIDCB=($M:PASSO,$M:ASSEM)
ASSIGN M:EI,(FILE, $M:ASSEM)
FMGE (DELETE)
ASSIGN M:EI,(FILE, $M:PASSO)
FMGE (DELETE)
ASSIGN M:EI,(FILE, $M:RØØT)
FMGE (DELETE)
LØAD (LMN,FØRTRANH),(PERM),(EF,(FØRTRAN),(M:CDCB),(M:SIDCB),;
(M:DØDCB),;
(M:BØDCB),(M:LØDCB)),;
(SL,F),;
```

```
(BIAS,4400),;
(MAP)
ASSIGN M:EI,(FILE, F0RTRAN)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMF0RT)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMBASIC)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMDEBG)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMSYMB)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTML0AD)
FMGE (DELETE)
L0AD (LMN,FRAN),(EF,(FREN)),(N0TCB),(MAP),(PERM),;
(BIAS,4400),;
(ABS)
L0AD (LMN,BASIC),(MAP),(EF,(BPMBASIC),(M:EIDCB),(M:E0DCB),(M:D0DCB) ,;
(M:SIDCB),(M:L0DCB)),;
(BIAS,4400),;
(PERM)
L0AD (LMN,BDB ),(EF,(BDBR0M )),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:AL),(EF,(M:ALDCB)),(MAP),(PERM,LIB)           ,(N0SYSLIB)
L0AD (LMN,M:BI),(EF,(M:BIDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:B0),(EF,(M:B0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:CI),(EF,(M:CIDCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:C ),(EF,(M:CDCB)),(MAP),(PERM,LIB)          ,(N0SYSLIB)
L0AD (LMN,M:CK),(EF,(M:CKDCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:C0),(EF,(M:C0DCB)),(MAP),(PERM,LIB)                ,(N0SYSLIB)
L0AD (LMN,M:D0),(EF,(M:D0DCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:EI),(EF,(M:EIDCB)),(MAP),(PERM,LIB)                  ,(N0SYSLIB)
L0AD (LMN,M:G0),(EF,(M:G0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:E0),(EF,(M:E0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:LI),(EF,(M:LIDCB)),(MAP),(PERM,LIB)          ,(N0SYSLIB)
L0AD (LMN,M:LL),(EF,(M:LLDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:L0),(EF,(M:L0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:0C),(EF,(M:0CDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:P0),(EF,(M:P0DCB)),(MAP),(PERM,LIB)            ,(N0SYSLIB)
L0AD (LMN,M:SI),(EF,(M:SIDCB)),(MAP),(PERM,LIB)            ,(N0SYSLIB)
L0AD (LMN,M:SL),(EF,(M:SLDCB)),(MAP),(PERM,LIB)             ,(N0SYSLIB)
L0AD (LMN,M:S0),(EF,(M:S0DCB)),(MAP),(PERM,LIB)           ,(N0SYSLIB)
L0AD (LMN,SSS ),(EF,(SSSR0M )),(MAP),(PERM,LIB)        ,(N0SYSLIB)
0VERLAY (LMN,REW),;
(EF,(TAPEFCN),(TPECHST),(M:0CDCB),(M:CDCB),(M:BIDCB),(M:B0DCB),;
(M:SIDCB),(M:EIDCB),(M:LLDCB),(M:L0DCB),(M:P0DCB),(M:E0DCB),;
(M:C0DCB),(M:D0DCB),(M:CIDCB),(M:SLDCB),(M:ALDCB),(M:S0DCB),;
(M:LIDCB),(M:G0DCB)),;
(BIAS,4400),;
                    (MAP),(SL,F),(N0SYSLIB),(PERM)  ,(TSS,100)
TREE    TAPEFCN=TPECHST=M:0CDCB=M:CDCB=M:BIDCB=M:B0DCB=M:SIDCB;
=M:EIDCB=M:LLDCB=M:L0DCB=M:P0DCB=M:E0DCB=M:C0DCB=M:D0DCB=M:CIDCB;
=M:SLDCB=M:ALDCB=M:S0DCB=M:LIDCB=M:G0DCB
ASSIGN M:EI,(FILE,REW)
ASSIGN M:E0,(FILE,WE0F)
FMGE (ENTER,PERM)
ASSIGN M:E0,(FILE,PFIL)
FMGE (ENTER,PERM)
0VERLAY (LMN,9LAY  ),;
(EF,(ALL),(EVL),(IN1),(IN2),(LDR)          ,(PS1),(PS2),(WRT)),;
(BIAS,4400),;
            (N0TCB),(MAP),        (PERM),(SL,F),(N0SYSLIB)
TREE    LDR=(IN1,PS1,IN2,PS2=(ALL,EVL,WRT        ))
J0B :SYSGEN,DEFIT
ASSIGN M:P0,(DEVICE,9T),(0UTSN,P0SS)
DEF (INCL,SIGMET,SIG7FDP,BPM,:BLIB,BTM:BLIB)
FIN
```

```
*
:GENDCB (M:BI,:SYSGEN,(INSN,BTMB))
 END
 JOB :SYS,MAKER00M
 ASSIGN M:EI,(FILE,M:M0N)
 FMGE (DELETE)
 JOB :SYSGEN,SYSGEN
 ASSIGN M:BI,(INSN,BTMB)
 PASS1
:SELECT  (FILE,BTM:BLIB,BTMFINT,BTMFORT,BTMLOAD,BTMEDIT,
:         BTMSYMB,BTMDEBG,BTMFER,BTMBPM,BTMBASIC,BTMBO)
:SELECT (FILE,BTMNRES)
:SELECT (FILE,BPMSUPER,BTMSUPER)
:SELECT  (FILE,BTMDELTA)
:SELECT (FILE,ROOT,7TAP,FBCD,IORT,HANDLERS,TOPRT,IO,;
:PRGMLDR,TYPR,IOD,DEBUG,DUMP,EXIT,M:15,     M:16,M:17,KEYIN,M:14,;
:M:18,M:1A,M:1B,M:1C,M:1D,RDF,;
:OPNL,OBSE,M:1E,OPN,CLS,MODIFY,CLS1,SEGLOAD,LDPRG,MEMALOC,CALPROC,;
:WRTF,WRTD,DUMMYCCL,LBLT,M:19,POS,ALTCP)
:SELECT (FILE,CRDOUT,PTAP,DFBCD)
:SELECT (FILE,IOSYM,COOP,CCLOSE)
:SELECT (FILE,CCIROOT,JOB,LIMIT,ASSIGN,LOAD,TREE,;
:TELSCPE,RUN,CCIDBUG,READBI,ENDJOB,ABORT)
:SELECT (FILE,LDR,IN1,PS1,IN2,PS2,ALL,EVL,WRT)
:SELECT (FILE,FILEMNGE,TPECHST,FMGEDCBS)
:SELECT (FILE,FORTRAN)
:SELECT  (FILE,SIGMET,SIG7FDP,BPM)
:SELECT (FILE,FREN)
:SELECT (FILE,LOPEROM,:BLIB)
:SELECT (FILE,TAPEFCN)
:SELECT (FILE,M:CDCB,M:BIDCB,M:CIDCB,M:EIDCB,M:SIDCB,M:BODCB,M:CODCB,;
:M:DODCB,M:EODCB,M:LODCB,M:SODCB,M:PODCB,M:ALDCB,M:LLDCB,M:SLDCB,;
:M:OCDCB,M:LIDCB,M:GODCB)
:SELECT (FILE,M:CKDCB)
:SELECT  (FILE,SSSROM,BDBROM)
:SELECT (FILE,CHKPTROM)
:SELECT (FILE,SYMBL)
:SELECT (FILE,$M:PASSO,$M:ASSEM,$M:ROOT)
:SELECT (FILE,BPMBASIC)
:SELECT (FILE,CN704363,CN704364,CN704365,CN704366)
 EOD
 PASS2
:STDLB (C,CRA03), (OC,TYA01),(LO,LPA02),(LL,LPA02),(DO,LPA02)
:STDLB (PO,CPA04),(EO,CPA04),(LI,CRA03)
:STDLB (SI,CRA03),(BI,CRA03),(SL,LPA02)
:STDLB (SO,CPA04),(CI,CRA03),(CO,CPA04)
:STDLB (AL,CPA04),(EI,CRA03),(EO,CPA04)
:DEVICE  TYA01,(HAND,KBTIO)
:DEVICE  CRA03,(HAND,CRDIN)
:DEVICE CPA04,(HAND,CRDOUT)
:DEVICE LPA02,(HAND,PRTOUT),(PAPER,26,132)
:DEVICE 9TA80,(HAND,MTAP)
:DEVICE 9TA81,(HAND,MTAP)
:DEVICE DCAF0,(HAND,DISCIO),(SS,5A),(PSA,50),(PFA,1B0),(PER,00)
:DEVICE DCAF1,(HAND,DISCIO),(SS,5A),(PSA,00),(PFA,140),(PER,C0)
:DEVICE DCAD0,(HAND,DISCIO),(SS,05A),(PSA,0),(PFA,0),(PER,0)
:DEVICE C0A10,(HAND,C0C)
:SDEVICE (LMN,ISSEG,CRA03),(LMN,OSSEG,LPA02)
:MONITOR (TSTACK,250),(QUEUE,10),(CORE,32),          (SFIL,30),;
:         (SPOOL,6),(CPOOL,4),;
:         (MPOOL,05)
:DLIMIT  (TIME,15),(LO,100),(DO,100),(UO,60),(PO,500),;
:         (TSTORE,512),(PSTORE,200),(FPOOL,2),(IPOOL,2)
:RESERVE (MPATCH,020)
:ABS,1024 (LOADER),(CCI),(METASYM),(SYMBOL),(FMGE),(FORTRANH),;
:          (PFIL),(WEOF),(REW),(LOPE),                  (BASIC)
:BTM (NUMUSERS,4),(USERSIZE,15872)
 OVERLAY (LMN,M:M0N),;
 (EF,(ROOT),,;
```

```
(M:RESDF),;
(DFBCD),;
(CN704364),(CN704365),(CN704366),;
(BTMB0),;
(M:BTM),;
(BTMNRES),;
(M:ABS),;
(CCL0SE),;
(C00P),;
(M:SDEV),;
(I0SYM),;
(M:CPU),(M:JIT),(I0TABLE)                              ,(I0RT),;
(CRD0UT),;
(HANDLERS),(T0PRT),   (PRGMLDR),(TYPR),(I0D),(DEBUG),(DUMP),;
(M:1C),(M:1D),(RDF),(0PNL),(0BSE),(M:1E),(0PN),(CLS),(M0DIFY),;
(EXIT),(KEYIN),(M:14),(M:15),(M:16),(M:17),(M:18),(M:1A),(M:1B),;
(CLS1),(SEGL0AD),(LDPRG),(MEMAL0C),(CALPR0C),(WRTF),(WRTD),;
(LBLT),(M:19),(P0S),(ALTCP)),;
(BIAS,0),(N0TCB),(MAP),(ABS),(SL,F),(N0SYSLIB),(PERM)
TREE R00T.;
M:RESDF.;
I0SYM.;
M:ABS.;
M:SDEV.;
M:CPU.M:JIT.I0TABLE.;
HANDLERS.;
CRD0UT.;
DFBCD.;
CN704364.CN704365.CN704366.;
BTMB0.;
M:BTM.;
C00P.;
I0RT.T0PRT.;
(PRGMLDR,TYPR,I0D.BTMNRES,;
                    DEBUG.DUMP,EXIT.DUMP,M:15,M:16,M:17,KEYIN,M:14,;
M:18,M:1A,M:1B,M:1C,M:1D,MEMAL0C,RDF.(CALPR0C.;
 (0PNL.0BSE,M:1E.0BSE,0PN.0BSE,CLS.M0DIFY.;
CLS1,SEGL0AD.0BSE,;
        WRTF,WRTD.CCL0SE   ,LBLT,M:19,P0S,;
ALTCP)),LDPRG)
0VERLAY (LMN,CCI),;
(EF,(CCIR00T),(J00),(LIMIT),(ASSIGN),(L0AD),(TREE),(TELSCPE),;
(RUN),(CCIDBUG),(READBI),(ENDJ0B),(AB0RT),(M:DLIMIT),(M:JIT))
(BIAS,3800),;
            (N0TCB),                (MAP),(PERM),(SL,F),(N0SYSLIB)
ASSIGN M:EI,(FILE,M:RESDF)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:DLIMIT)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:FC0M)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:JIT)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:CPU)
FMGE (DELETE)
ASSIGN M:EI,(FILE,I0TABLE)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:SDEV)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:BTM)
FMGE (DELETE)
ASSIGN M:EI,(FILE,M:ABS)
FMGE (DELETE)
L0AD            (SL,F),(MAP),(N0TCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   F0RTRAN:),(EF,          (BTMFINT),(BTMF0RT))
L0AD            (SL,F),(MAP),(N0TCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   L0AD:)   ,(EF,          (BTML0AD))
L0AD            (SL,F),(MAP),(N0TCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   EDIT:)   ,(EF,                      (BTMEDIT))
```

```
LØAD           (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   SYMBØL:) ,(EF,          (BTMSYMB))
LØAD           (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   FERRET:) ,(EF,          (BTMFER))
LØAD           (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   BPM:)    ,(EF,          (BTMBPM))
LØAD           (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   BASIC:)  ,(EF,          (BTMBASIC))
LØAD (MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,SUPER:),(EF,(BTMSUPER))
LØAD           (SL,F),(MAP),(NØTCB),(PERM),(ABS),;
(BIAS,4200),;
(LMN,   DELTA:)  ,(EF,          (BTMDELTA))
ØVERLAY (LMN,FMGE),;
(EF,(FILEMNGE),(TPECHST),(FMGEDCBS)),;
(BIAS,3800),;
                    (MAP),(SL,F),(NØSYSLIB),(PERM)     ,(TSS,100)
ØVERLAY (LMN,LØADER),;
(EF,(ALL),(EVL),(IN1),(IN2),(LDR)          ,(PS1),(PS2),(WRT)),;
(BIAS,3800),;
            (NØTCB),(MAP),       (PERM),(SL,F),(NØSYSLIB)
JØB :SYS,DELETE,F
ASSIGN M:EI,(FILE,PASS2)
FMGE (DELETE)
ASSIGN M:EI,(FILE,PASS1)
FMGE (DELETE)
JØB :SYSGEN,SYSGEN
LØAD (LMN,SUPER),(NØTCB),(MAP),(ABS),(EF,(BPMSUPER)),;
(BIAS,3800),;
(PERM)
LØAD (LMN,SYMBØL),(PERM),(MAP),;
(BIAS,3800),;
(EF,(SYMBL),;
(M:ØCDCB),(M:DØDCB),(M:GØDCB),(M:SIDCB),(M:BØDCB),(M:LØDCB),(M:CDCB) ),;
(SL,F)
ØVERLAY (EF,($M:RØØT),($M:PASSØ),($M:ASSEM),;
(M:DØDCB),(M:CDCB),(M:LØDCB),(M:BØDCB),(M:GØDCB),          (M:SIDCB),;
(M:SØDCB),;
(M:CIDCB),      (M:CØDCB),(M:LLDCB),(M:ØCDCB));
                                                ,(ABS),(PERM),(SL,F),;
(BIAS,3800),;
(LMN,METASYM),            (TSS,80),(MAP),(UNSAT,(:SYSGEN))
TREE $M:RØØT=M:BØDCB=M:CDCB =M:CIDCB=M:CØDCB=M:DØDCB=M:GØDCB=;
M:SØDCB=;
M:LLDCB=M:LØDCB=M:ØCDCB=M:SIDCB=($M:PASSØ,$M:ASSEM)
ASSIGN M:EI,(FILE, $M:ASSEM)
FMGE (DELETE)
ASSIGN M:EI,(FILE, $M:PASSØ)
FMGE (DELETE)
ASSIGN M:EI,(FILE, $M:RØØT)
FMGE (DELETE)
LØAD (LMN,FØRTRANH),(PERM),(EF,(FØRTRAN),(M:CDCB),(M:SIDCB),;
(M:DØDCB),;
(M:BØDCB),(M:LØDCB)),;
(SL,F),;
(BIAS,3800),;
(MAP)
LØAD (LMN,LØPE),(MAP),(NØTCB),(EF,(LØPERØM)),;
(BIAS,3800),;
(PERM)
ASSIGN M:EI,(FILE, FØRTRAN)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMFØRT)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMBASIC)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMDEBG)
```

```
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTMSYMB)
FMGE (DELETE)
ASSIGN M:EI,(FILE,BTML0AD)
FMGE (DELETE)
L0AD (LMN,FRAN),(EF,(FREN)),(N0TCB),(MAP),(PERM),;
(BIAS,3800),;
(ABS)
L0AD (LMN,BASIC),(MAP),(EF,(BPMBASIC),(M:EIDCB),(M:E0DCB),(M:D0DCB) ,;
(M:SIDCB),(M:L0DCB)),;
(BIAS,3800),;
(PERM)
L0AD (LMN,BDB ),(EF,(BDBR0M )),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:AL),(EF,(M:ALDCB)),(MAP),(PERM,LIB)           ,(N0SYSLIB)
L0AD (LMN,M:BI),(EF,(M:BIDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:B0),(EF,(M:B0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:CI),(EF,(M:CIDCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:C ),(EF,(M:CDCB)),(MAP),(PERM,LIB)          ,(N0SYSLIB)
L0AD (LMN,M:CK),(EF,(M:CKDCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:C0),(EF,(M:C0DCB)),(MAP),(PERM,LIB)                ,(N0SYSLIB)
L0AD (LMN,M:D0),(EF,(M:D0DCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:EI),(EF,(M:EIDCB)),(MAP),(PERM,LIB)                  ,(N0SYSLIB)
L0AD (LMN,M:G0),(EF,(M:G0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:E0),(EF,(M:E0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:LI),(EF,(M:LIDCB)),(MAP),(PERM,LIB)          ,(N0SYSLIB)
L0AD (LMN,M:LL),(EF,(M:LLDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:L0),(EF,(M:L0DCB)),(MAP),(PERM,LIB)        ,(N0SYSLIB)
L0AD (LMN,M:0C),(EF,(M:0CDCB)),(MAP),(PERM,LIB)         ,(N0SYSLIB)
L0AD (LMN,M:P0),(EF,(M:P0DCB)),(MAP),(PERM,LIB)            ,(N0SYSLIB)
L0AD (LMN,M:SI),(EF,(M:SIDCB)),(MAP),(PERM,LIB)            ,(N0SYSLIB)
L0AD (LMN,M:SL),(EF,(M:SLDCB)),(MAP),(PERM,LIB)             ,(N0SYSLIB)
L0AD (LMN,M:S0),(EF,(M:S0DCB)),(MAP),(PERM,LIB)           ,(N0SYSLIB)
L0AD (LMN,SSS ),(EF,(SSSR0M )),(MAP),(PERM,LIB)        ,(N0SYSLIB)
0VERLAY (LMN,REW),;
(EF,(TAPEFCN),(TPECHST),(M:0CDCB),(M:CDCB),(M:BIDCB),(M:B0DCB),;
(M:SIDCB),(M:EIDCB),(M:LLDCB),(M:L0DCB),(M:P0DCB),(M:E0DCB),;
(M:C0DCB),(M:D0DCB),(M:CIDCB),(M:SLDCB),(M:ALDCB),(M:S0DCB),;
(M:LIDCB),(M:G0DCB)),;
(BIAS,3800),;
                    (MAP),(SL,F),(N0SYSLIB),(PERM)  ,(TSS,100)
TREE    TAPEFCN-TPECHST-M:0CDCB-M:CDCB-M:BIDCB-M:B0DCB-M:SIDCB;
-M:EIDCB-M:LLDCB-M:L0DCB-M:P0DCB-M:E0DCB-M:C0DCB-M:D0DCB-M:CIDCB;
-M:SLDCB-M:ALDCB-M:S0DCB-M:LIDCB-M:G0DCB
ASSIGN M:EI,(FILE,REW)
ASSIGN M:E0,(FILE,WE0F)
FMGE (ENTER,PERM)
ASSIGN M:E0,(FILE,PFIL)
FMGE (ENTER,PERM)
0VERLAY (LMN,0LAY  ),;
(EF,(ALL),(EVL),(IN1),(IN2),(LDR)          ,(PS1),(PS2),(WRT)),;
(BIAS,3800),;
           (N0TCB),(MAP),        (PERM),(SL,F),(N0SYSLIB)
TREE    LDR-(IN1,PS1,IN2,PS2-(ALL,EVL,WRT        ))
J0B :SYSGEN,DEFIT
ASSIGN M:P0,(DEVICE,9T),(0UTSN,P0SS)
DEF (INCL,SIGMET,SIG7FDP,BPM,:BLIB,BTM:BLIB)
```

INDEX

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

SDS 701 South Aviation Blvd./El Segundo, California 90245