XDS

Xerox Data Systems

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

User's Guide

Price: $3.50

BATCH TIME-SHARING MONITOR (BTM) USER'S GUIDE

for

XDS SIGMA 5/7 COMPUTERS

FIRST EDITION

February 1970

90 16 79A

XDS

Xerox Data Systems/701 South Aviation Boulevard/El Segundo, California 90245

Printed in U.S.A.

RELATED PUBLLICATIONS

Title	Publication No.
XDS Sigma 5/7 Batch Time-Sharing Monitor (BTM) Reference Manual	90 15 77
XDS Sigma 5 Computer Reference Manual	90 09 59
XDS Sigma 7 Computer Reference Manual	90 09 50
Mathematical Routines Technical Manual for XDS Sigma Computers	90 09 06
XDS Sigma Symbol and Meta-Symbol Reference Manual	90 09 52
XDS Sigma 5/7 BASIC Reference Manual	90 15 46
XDS Sigma FORTRAN IV Reference Manual	90 09 56
XDS Sigma FORTRAN IV Operations Manual	90 11 43
XDS Sigma FORTRAN IV Library Technical Manual	90 15 24
XDS Sigma FORTRAN IV-H Reference Manual	90 09 66
XDS Sigma FORTRAN IV-H Operations Manual	90 11 44
XDS Sigma FORTRAN IV-H Library/Run-Time Technical Manual	90 11 38
XDS Sigma 5/7 Batch Processing Monitor Reference Manual	90 09 54
XDS Sigma 5/7 Batch Processing Monitor Operations Manual	90 11 98
XDS Glossary of Computer Terminology	90 09 57
XDS Sigma Multipurpose Keyboard/Display Reference Manual	90 09 82
XDS Sigma Message-Oriented Communications Equipment Reference Manual	90 15 68
XDS Sigma Character-Oriented Communications Equipment Reference Manual	90 09 81

CONTENTS

ILLUSTRATIONS

TABLES

1. INTRODUCTION

The User's Guide is intended to introduce the terminal user to the basic functions of the BTM system. Information is divided into chapters, by subsystem, and presented largely in graphic and tabular format. Simple examples illustrate command usage. For additional information, the user should refer to the BTM Reference Manual (90 15 77).

GENERAL

As indicated in Figure 1-1, on-line services are provided by the following subsystems:

- BASIC processor
- EDIT source file editor

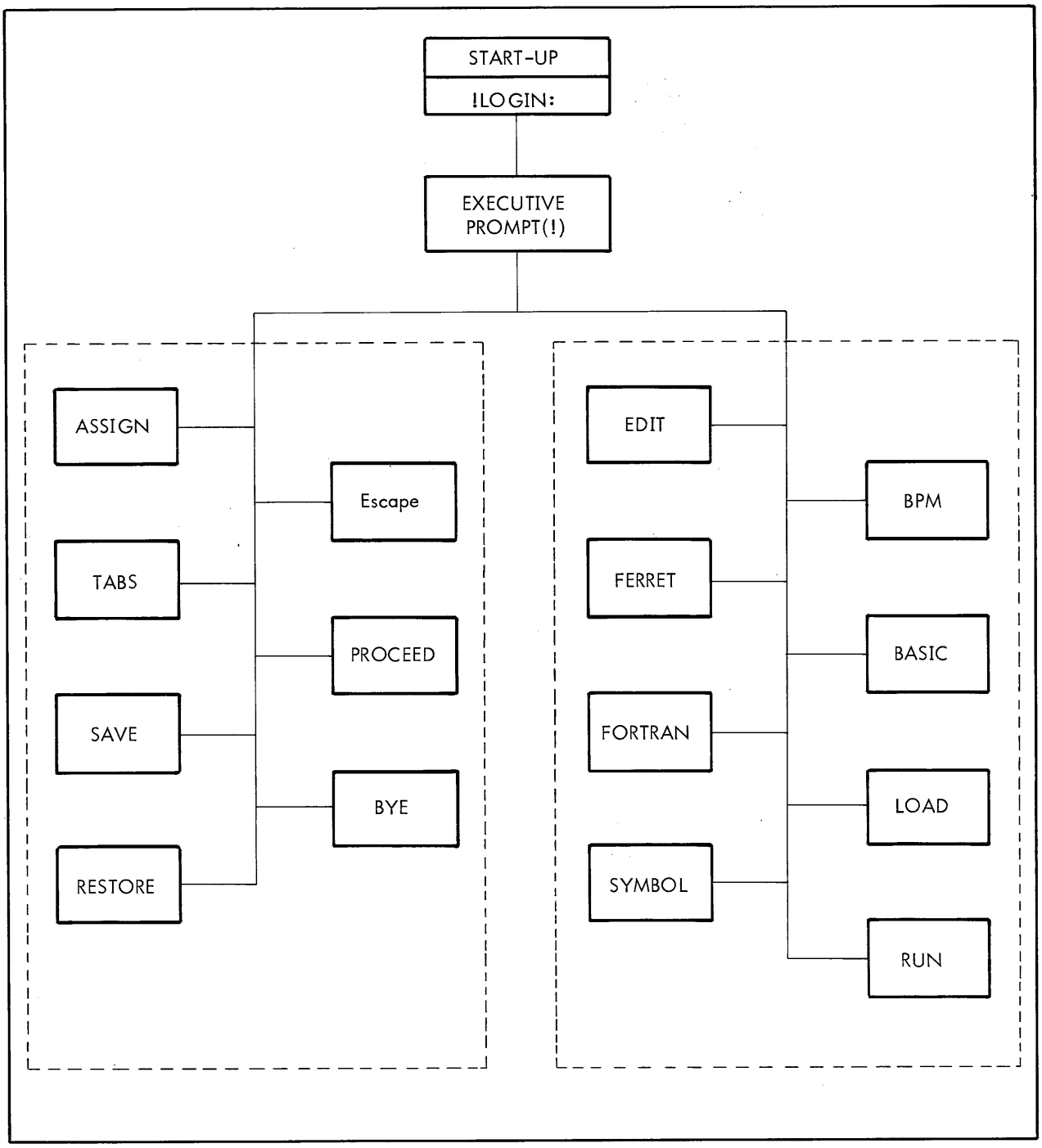


Figure 1-1. BTM System

- FERRET file utility subsystem
- FORTRAN IV-H compiler
- SYMBOL assembler
- LOADER object module loader
- BPM batch job controller
- RUN execution and debug subsystem

Batch processing services are accessed by:

- Local batch job decks submitted at the computer.
- Remote batch input through a high-speed card reader.
- Terminal batch input from an on-line Teletype terminal.

EXECUTIVE FUNCTIONS

The BTM Executive is a supervisory program responding to control commands from the on-line terminal. Such commands provide the user with a means of directing the flow of on-line work. In addition to the commands calling the BTM subsystems, the following commands may be used to perform various Executive functions:

- ASSIGN (controls file assignments)
- TABS (sets Teletype tabs)
- SAVE (saves current work)
- RESTORE (restores saved work)
- Escape (terminates currently activity)
- PROCEED (resumes terminated activity)
- BYE (terminates current work session)

SUBSYSTEM SERVICES

BTM includes subsystems that perform a broad range of text-editing, compiler, loader, assembler, debugging, file management, and terminal batch entry services.

TEXT-EDITING SERVICES

A powerful text-editing subsystem, called EDIT, enables terminal users to create and modify a source text file that can then be used for assembly or compilation either on-line under Symbol, FORTRAN, or BASIC, or in the batch mode under Meta-Symbol or other processors. EDIT operates on a line-number basis or by context and permits convenient intra-line character string editing. EDIT functions include the following:

- Create new files
- Copy files
- Delete files
- Add, delete, insert lines
- Insert comments at specified column position
- Search for specified character string and delete, insert, replace, or list line numbers
- Reorder lines within a file
- Renumber lines
- Perform character editing by line in step mode

- List file contents with or without line numbers
- Shift character strings within a line

BASIC SERVICE

The BASIC language developed for BTM is a highly extended version of the original Dartmouth BASIC. It is an easy-to-use language, developed specifically for on-line use. It includes most features of other BASIC implementations, in addition to new features not currently available elsewhere. BASIC programs can either be processed on-line at a terminal or placed into the batch job stream for execution. XDS Sigma 5/7 BASIC includes the following extended features:

- Alphanumeric variables
- Chaining of sequential BASIC programs
- ON statement (address switch)
- Direct statement execution (calculator mode)
- Matrix functions
- IMAGE statement
- Tab function (output format control)
- File input/output
- RUN/FAST compile option, which permits checked-out BASIC programs to execute faster without unnecessary array checking during run time

The execution of Sigma 5/7 BASIC is highly efficient, with an instantaneous compilation rate of 600 statements per second, and the size of a user program is constrained only by the amount of core space made available to the user.

FORTRAN IV-H COMPILER SERVICE

The FORTRAN IV-H compiler subsystem enables on-line terminal users to compile a FORTRAN source-program file, thus generating an object-program file. The source file can be created directly on-line with the FORTRAN subsystem, via the on-line EDIT subsystem, or entered through the batch system. When source code is being created on-line with the FORTRAN compiler, each statement is immediately syntax-checked and the user can correct any errors without waiting until the end of the entire program input. Users have three compile options: debug run-time functions, source listing output file with object code, and on-line source listing.

All error messages with the error statements are printed out at the user's terminal. User object programs can also assign I/O directly to the terminal teletypewriter.

After completing compilation, terminal users can call for either EDIT, to correct source-program errors and recompile, or the FORTRAN loader and run-time subsystem, to execute compiled programs. At the user's option, the compiled FORTRAN program can also be placed into the batch job stack for later execution in the batch mode.

FORTRAN EXECUTION SERVICE

To permit the compiled FORTRAN IV programs to be executed on-line, a FORTRAN loader and run-time subsystem is provided through the LOADER subsystem. The FORTRAN loader reports loading errors to the terminal user. If such errors are not serious, the user can call for excecution. Run-time diagnostics are reported to the terminal during program execution. Debugging output can also be printed at the terminal, it the Debugging option is specified at compile time. Debugging functions include the ability to:

- Trace source statements reached during execution
- Display values that have been stored into variables as a result of assignment statements
- Stop the program prior to execution of specified source lines

- Step the program one statement at a time
- Request that execution be continued after a stop

During program execution, the user's FORTRAN program can be used "conversationally" if I/O has been assigned to the terminal.

LOADER SUBSYSTEM

The LOADER subsystem loads XDS standard object-language programs from specified element files. It can also load library modules from specified account files or the public library file. Multiple object modules generated by Symbol, FORTRAN IV-H, and Meta-Symbol can be loaded for execution.

If the user's program has been compiled in the debug mode, the FORTRAN debug run-time package will be loaded with the object program. For assembly language programs, the user can specify the Debugging option which will cause the DELTA debug package to be brought in with his program. The LOADER subsystem provides the following facilities:

- Assignment of input element files
- Library search for unsatisified references option
- Load map option
- Specification of temporary storage stack size
- Specification of FORTRAN I/O assignment
- Control of loading error severity alternatives
- Control of execution start
- Debugging option

All activities of the LOADER subsystem are carried on conversationally, and all error messages are directed to the user's terminal.

ASSEMBLER SERVICES

The Symbol assembly subsystem, provided under BTM, permits terminal users to assemble source text files and to create user object program modules. Available assembly options include:

- Assembly with or without binary output
- Assembly with or without a listing output file
- Assembly with or without a symbol table for on-line debugging
- Terminal listing

Error messages are printed at the terminal and, after an assembly is completed, control is returned to the Terminal Executive program.

FERRET SUBSYSTEM

FERRET is a utility subsystem that enables the on-line terminal user to obtain information about his permanent files. It also provides for limited file manipulation. FERRET performs the following functions:

- List all file names under specified account number
- Report date of creation and size of files

- Indicate accessibility of specified file
- Delete file in user's account
- Copy file
- Examine specified file for number of records
- Print specified records in file (in EBCDIC or hexadecimal code)

TERMINAL BATCH ENTRY SUBSYSTEM

The BPM Terminal Batch Entry subsystem enables the terminal user to create a job control file on-line and submit it to the BPM batch job stream. The user is then disconnected from that job and is free to pursue other activities. At any time, the user can also employ the Terminal Batch Entry subsystem to interrogate the system concerning the status of a submitted job. This status may be either (1) waiting to be run, (2) running, or (3) completed.

The output of the job may be directed to standard peripheral output devices or to the user's permanent file storage. In the latter case, it is then convenient for the user to resume conversational activity at his terminal with the results of the batch execution.

The Terminal Batch Entry subsystem also provides a convenient editing feature to prepare control commands for job execution.

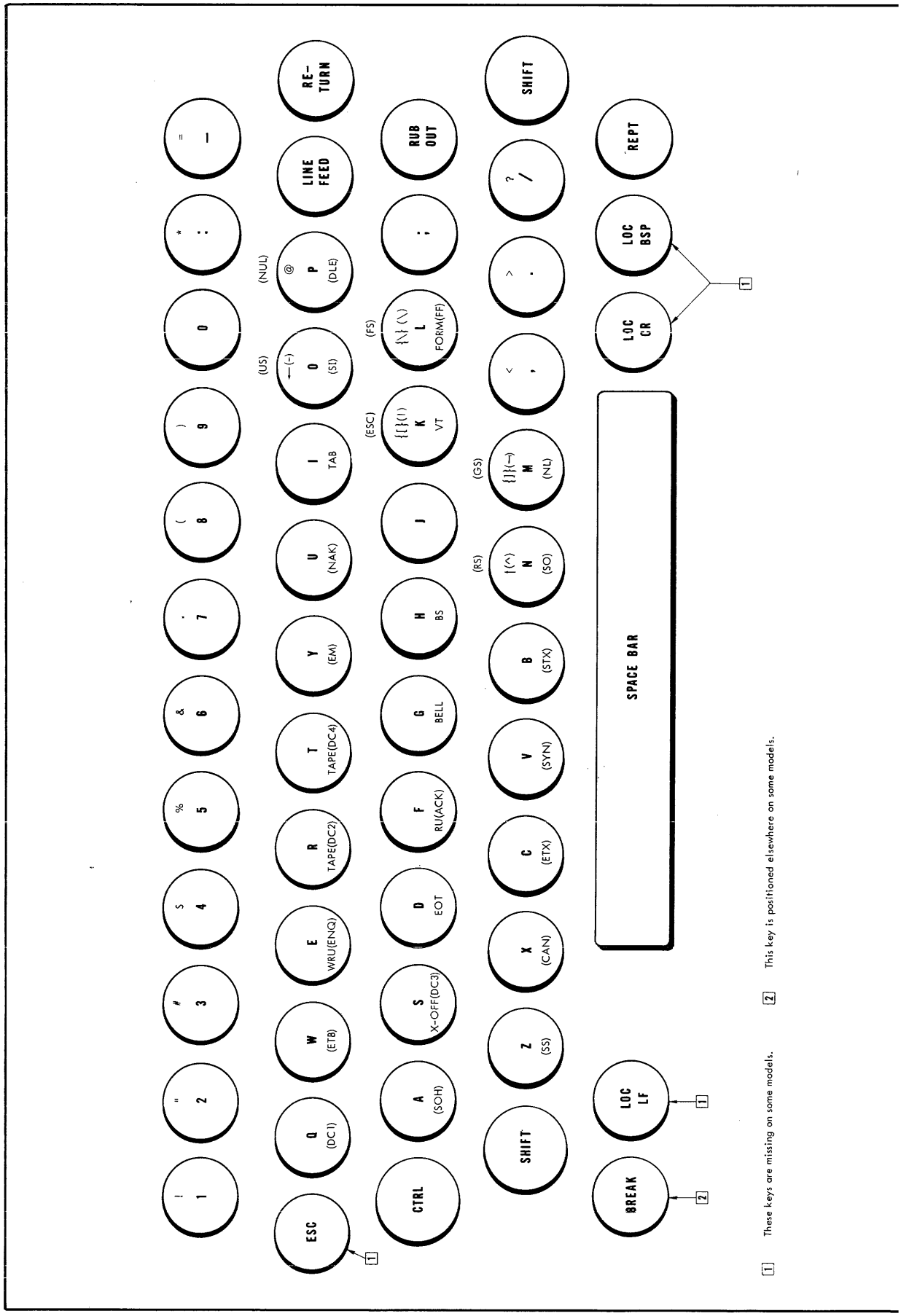


Figure 2-1. Keyboard Layout

ACTIVATION CHARACTER EFFECTIVITY

Following is a breakdown showing when the various groups of activation characters are in effect.

When Under Executive Control

1. Immediately after the Executive prompt character (!) the activation mode is "activate on every character". There may be a delay in the echoing of each character, and no backspacing or correction is possible. This mode is in effect when the first two characters of an Executive command are typed; e.g.,

   !AS

2. When using the ASSIGN command, the remainder of the specification is typed in the mode "activate on (RET), (LF), or punctuation". For example, in the command

   !ASSIGN M : DO,(HERE) (RET)

   activation occurs after the typing of:

   a. A

   b. S

   c. M : DO

   d. (

   e. HERE

   f.)

3. The other Executive commands that require specifications cause the mode to be "activate on (RET) or (LF) ".

   In the command

   !SAVE ALPHA,PERM (RET)

   activation occurred after the typing of:

   a. S

   b. A

   c. ALPHA,PERM

When Under Control Of Subsystems

1. All subsystem input that is read through Data Control Blocks assigned to the user's terminal (such as source language input to a compiler) is gathered one line at a time under the mode "activate on (RET) or (LF)". This input is always prompted by the typing of a colon (:) at the beginning of the line.

2. BASIC, EDIT, and FERRET subsystems read all input in the mode "activate on (RET) or (LF)". The BPM subsystem reads input in this mode, except when it expects a Y or N answer; then the mode is, "activate on every character".

3. The FORTRAN, LOADER, SYMBOL, and RUN subsystems acquire all option lists and input specifications in the "activate on (RET) or (LF) mode".

4. The debugging program Delta reads in the "activate on /, =, (TAB), ↑, (RET), or (LF) mode".

SPECIAL TELETYPE DIRECTIVES

There are a number of simple operations that may be performed during the typing of input to the computer. These operations are functions of the program that handles immediate Teletype communication, and are not to be confused with Executive or subsystem functions (see Table 2-1).

Table 2-1 Special Teletype Directives

Directive	Description	Indication
ACKNOWLEDGE (ESC Q)	Ascertains that the Batch Timesharing Monitor is operating at any given time. Used to question a long delay in Teletype response.	BTM operating will produce double exclamation mark (!!) immediately.
BACKSPACE (ESC RUB)	Causes the last textual character typed to be deleted. Additional BACKSPACE causes the next previous character to be deleted, and so on, until it reaches an activation character. Example: MISPELL ← ← ← ← SPELL The above, typed at the terminal, results in corrected spelling of MISSPELL. Backspacing over certain control characters will cause the next previous textual character to be deleted. Attempted backspacing over an activation character will be ignored (since it is already in processing).	Each time BACKSPACE is struck, (←) is echoed, and the carriage is advanced one position.
LOCAL NEW LINE ESC RET	Causes the Teletype to be positioned to a new line.	Does not terminate the current logical line.
ERASE (ESC X)	Effectively erases all input back to the last activation character. It is not possible to erase an activation character or anything typed preceding it.	Carriage skips to beginning of next line, to signal that the erasure has been performed.
RETYPE (ESC R)	Causes carriage to skip to the next line and retype all data following last activation character as it now exists in computer buffer. Checks results of backspacing, tabbing and erasing operations. Example: MISPELL ← ← ← ← SPELL ESC R Next line: MISSPELL	
TAB (ESC I or TAB)	Causes a skip to the next tabular position across the page, inserting the proper number of blank spaces into the computer input. (Use Executive command TABS to set up column tabular positions.) Note: If TAB is hit inadvertently, it is necessary to backspace the number of times equal to the number of blanks inserted in skipping forward.	Column positions for tabbing are counted starting after the last activation character, regardless of whether this begins a new line. The column position count is reduced automatically if backspaced, and reset to one if erased. Attempt to tab beyond last tab position set results in an echo of (?) and a one-position space input.

Table 2-1. Special Teletype Directives (cont.)

Directive	Description	Indication
ECHO CONTROL ESC E	Causes suppression of the echo return. Used in half-duplex operations to prevent double printing and in full-duplex mode to suppress printing of private information. Example: LOGIN: ESC E ... ESC E (The information between the two ESC E's is not printed.)	Printing of the echo return is suppressed by an ESC E and restored by the same command.
PUNCH ONLY ESC P	Causes output to go to the terminal's paper tape punch only. A second ESC P restores normal operation.	

TEXT SYMBOLS

Some symbolism is used in the text to describe formats for writing commands and parameter information. Table 2-2 lists this information.

Table 2-2. Format Symbols

Symbol	Description and Use
CAPITAL LETTERS	Capitals are used for all words that are to be explicitly typed, such as AS of the ASSIGN command, the word FILE in one of the parameters, etc.
lower case letters	Lower case is used to name values which are to be filled in, such as "dcb name" or "file name" in the ASSIGN command.
[ ]	Brackets are never to be typed. They enclose parts of the format which are optional, or which are not always to be filled in. For example, L[IST] indicates that the whole word LIST may be typed, or only the letter L.
( ) , -	Parentheses, commas, and dashes are separators and are always to be typed in as shown.
...	Ellipsis indicates that there may be a series of two or more elements of the same type as that which precedes it.
{ }	Braces indicate a required choice.
___	Underlining indicates characters printed by the system.

3. START-UP AND THE EXECUTIVE

GENERAL

When telephone contact has been made with the on-line system and the user has logged in, it is possible to issue commands to the Executive program. The Executive will connect the user with the subsystem employed on on-line work. In addition, the Executive permits some specification of set-up conditions, interruption and continuation of processing, and sign-off at the end of a work session.

COMMANDING THE EXECUTIVE

Whenever the Executive is ready to accept a command, it types an exclamation mark (!). The first two letters of the command mnemonic are typed, and the Executive replies by typing the remainder of the word plus one space. Executive functions are listed in Table 3-1.

Table 3-1. Executive Functions

Function	Description
ASSIGN	Specify or change input/output assignments.
TABS	Set column tabs for terminal input.
BASIC	Enter the BASIC language subsystem.
BPM	Enter the subsystem for scheduling batch jobs from the terminal.
EDIT	Enter the EDIT subsystem for input and modification to files.
FERRET	Enter the FERRET subsystem for accessibility checking and file operations.
FORTRAN	Enter the FORTRAN language subsystem.
LOAD	Enter the LOADER subsystem.
SYMBOL	Enter the SYMBOL language subsystem.
RUN	Enter the RUN subsystem.
SAVE	Save the core and register data of the present activity in order to continue later.
RESTORE	Restore a previously saved activity.
Escape	Interrupt processing and return to the Executive.
PROCEED	Resume interrupted processing.
BYE	Sign-off.

START-UP AND LOG-IN

Sequential procedures for activating the terminal and logging in appear in Table 3-2, below

Table 3-2. Start-Up and Log-In Procedures

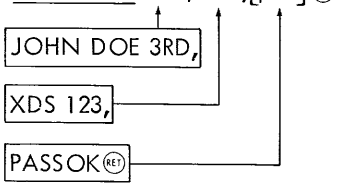
Step	Description	Indication
1	Depress "ORIG" or "BREAK" key to activate terminal.	Key lights up and remains depressed. Dial tone is audible.
2	Dial on-line system number. If there is a busy signal or no answer, depress "CLR" to deactivate and try again in a few minutes. To establish contact with the computer, strike (ESC) (ESC).	Answering "beep" indicates that connection with computer has been made.
3	When contact is made, Executive types message. Exclamation point (!) indicates communication Executive. Colon (:) indicates that information must be typed in (step 4).	BTM SYSTEM -x IS UP date and time !LOGIN:
4	name may be up to 12 characters long, may include letters, characters, numbers, and blanks. acct is up to 8 characters long and may include letters, numbers, and blanks. pass also up to 8 characters long and may include letters, numbers, and blanks, or nonprinting control characters. Note: Only one password may be validated for each name of user with a given acct, but several users may have same password.	!LOGIN: name,acct,[pass] (RET)  If the user fails to log in within one minute following the !LOGIN: request, he is disconnected from the computer.
5	Name, account and password are checked for validity. If not accepted, Executive returns another request for log-in information. If the name, account, and password are accepted, the Executive types an ID character, and on the next line, an exclamation mark (!) indicating that it is ready to accept a command. Note: The special ID character is assigned as a means of preventing duplicate names for certain temporary files. This is explained more fully later in this chapter.	? !LOGIN: ID = X !
6	Type first two letters of the command mnemonic, such as AS. Executive replies by typing the remainder of the word, plus one space.	!ASSIGN
7	If command requires parameter information, type the information and terminate the command with (RET). There are no blanks in the parameter portion of a command.	!ASSIGN M:SI,(FILE,X),(IN),(TEMP) (RET)

Table 3-2. Start-Up and Log-In Procedures (cont.)

Step	Description	Indication
8	Commands that do not require parameter information terminate themselves as soon as the Executive types the remainder of the word, and no carriage return is necessary.	!BASIC >
9	If the Executive does not recognize the letters of a command, it replies with a question mark (?). Retype.	!BZ ?
10	If it is desired to cancel a command during the typing of its parameters, depress the "BREAK" key. The Executive will ignore the command, skip to the next line, and type an exclamation mark, awaiting next command.	!ASSIGN M:SI,(F![BREAK] !

ASSIGN COMMAND

This command changes file assignments and file options. File options, their meaning and description appear in Table 3-3.

An important use of ASSIGN is prior to calling the FORTRAN, LOADER, BPM, and SYMBOL subsystems where it is desired to specify files of the user as source input, and where it is desired to save listings and compiled binary program modules in files of the user. If no special assignments are made, these subsystems make appropriate default assignments. Refer to Table 3-4.

Also in the above subsystems M:DO, dignostic output, is initially assigned to the user's terminal but may be reassigned. In the case of all the remaining subsystems, file assignments are made after the subsystems are called.

Table 3-3. File Options

Option	Meaning	Description
IN	Input file.	A previously existing file to be read only.
OUT	Output file.	A new file to be written only, not read.
INOUT	Update.	A previously existing file, whose records may be read, then written.
OUTIN	Scratch file.	A new file to be written, then read.
PERM	Permanent.	Will not be deleted following use. Unless otherwise indicated, PERM is applied to all input files.
TEMP	Temporary.	Will be deleted at sign-off time. Unless otherwise indicated, TEMP is applied by default to all output files.

Table 3-3. File Options (cont.)

Option	Meaning	Description
HERE	User's Teletype.	This input or output through user's own terminal.
REL	Release.	Releases the permanent storage space occupied by a file. Use REL when releasing a file that previously was being permanently saved.
SAVE	Save.	Obtains permanent storage space for a file. Specify SAVE with PERM when it is desired to save a file after sign-off (it need be specified only once per file).
READ	Specify access.	Limits read access by other accounts to user's file. Following READ, up to eight account IDs (separated by commas) may be specified to be permitted to read user's file When the READ option is not specified, it is assumed that all accounts may read the file.
READ, NONE	Specify access.	NONE may be specified to prevent any other account from reading the file.
READ, ALL	Specify access.	ALL may be specified to permit all accounts to read the file.
WRITE	Specify access.	Specifies other accounts that are permitted to both write and read the file. Following WRITE, up to eight account IDs (separated by commas) may be specified to read or write file. If the WRITE option is omitted, it is assumed that no other accounts are permitted write access to the file. If there is a conflict between READ and WRITE option specifications, WRITE takes precedence.
WRITE, NONE	Specify access.	NONE may be specified to prevent any other account from reading or writing.
WRITE, ALL	Specify access.	ALL may be specified to permit all accounts to read and write.

Table 3-4. DCB Default Assignment

Subsystem	DCB Name	Meaning	Default Assignment
FORTRAN	M:SI M:LO M:BO M:SO	Source Input Listed Output Binary Output Source Output	Terminal Terminal Temporary File BOTEMPx Temporary File SOTEMPx
LOADER	M:BI	Binary Input	Temporary File BOTEMPx
BPM	M:SI	Source Input	Terminal
SYMBOL	M:SI M:LO M:BO	Source Input Listed Output Binary Output	Terminal Terminal Temporary File BOTEMPx

SPECIAL ID CHARACTER

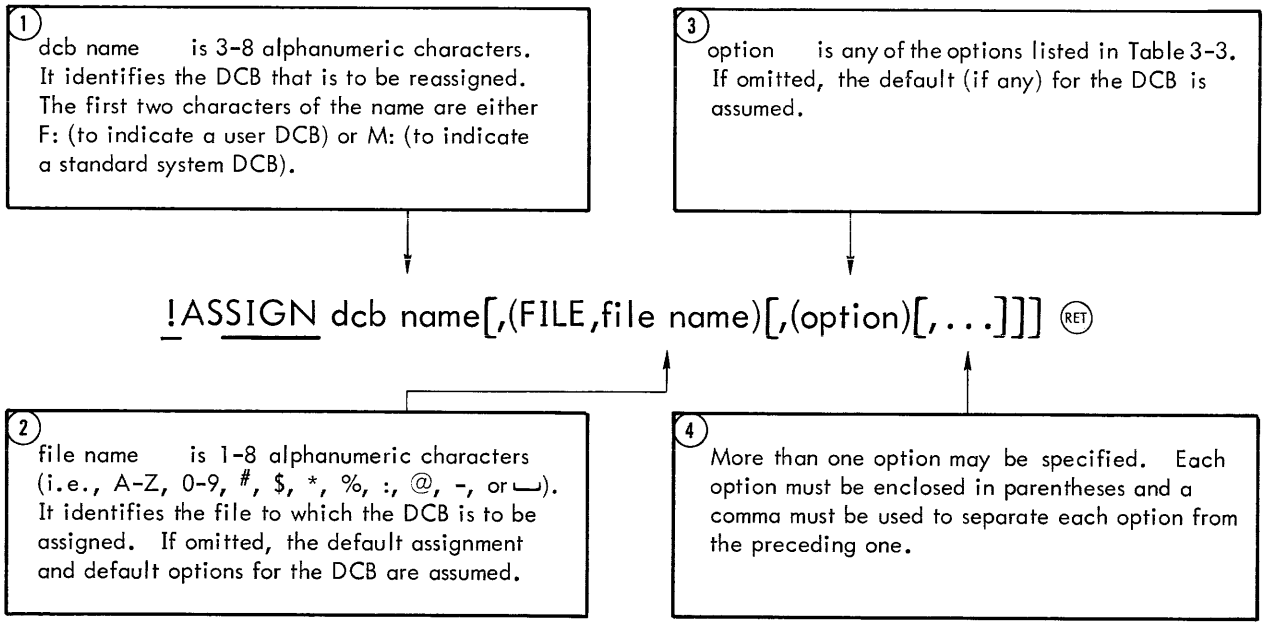
The special ID character is assigned as a means of preventing the occurrence of duplicate names for certain temporary files automatically created by subsystems, in the event that two or more users are operating simultaneously under the same account. The assigned ID will be a letter of the alphabet or one of the digits 1 to 6. This character will be made a part of the names of these temporary files when any of them are created during use of the system. The Executive indicates the ID character assigned to the user by typing ID = x after the user logs into the system. Table 3-5 lists the subsystems which create temporary output files, with x representing the unique ID character in the file names.

Table 3-5. Temporary Output Files

Subsystem	File Name	File Function
BASIC	RUNxFIL	Scratch file.
SYMBOL	BOTEMPx	Default binary output.
FORTRAN	BOTEMPx SOTEMPx	Default binary output. Default source language output.
LOAD	BOTEMPx SDxFIL	Default binary output. Scratch file for debugging symbol tables.

ASSIGN (Assign DCB)

ASSIGN causes specified changes to be made in the assignment of a given Data Control Block (a table of information affecting I/O operations).



Note that the action of ASSIGN commands is not cumulative; each successive ASSIGN for a given DCB cancels any previous ASSIGN completely.

Example:

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

This specifies that a random access device file named SIMBOLIK is to be used as source input. The file may be read only, and is not to be saved after sign-off.

```
!ASSIGN M:BO,(FILE,LOADMOD1),(OUT),(PERM),(SAVE),(READ,NONE)RET
!
```

The binary output is to be made a new random access file named LOADMOD1. It is to be saved as a permanent file, and no other account number may read or write on it.

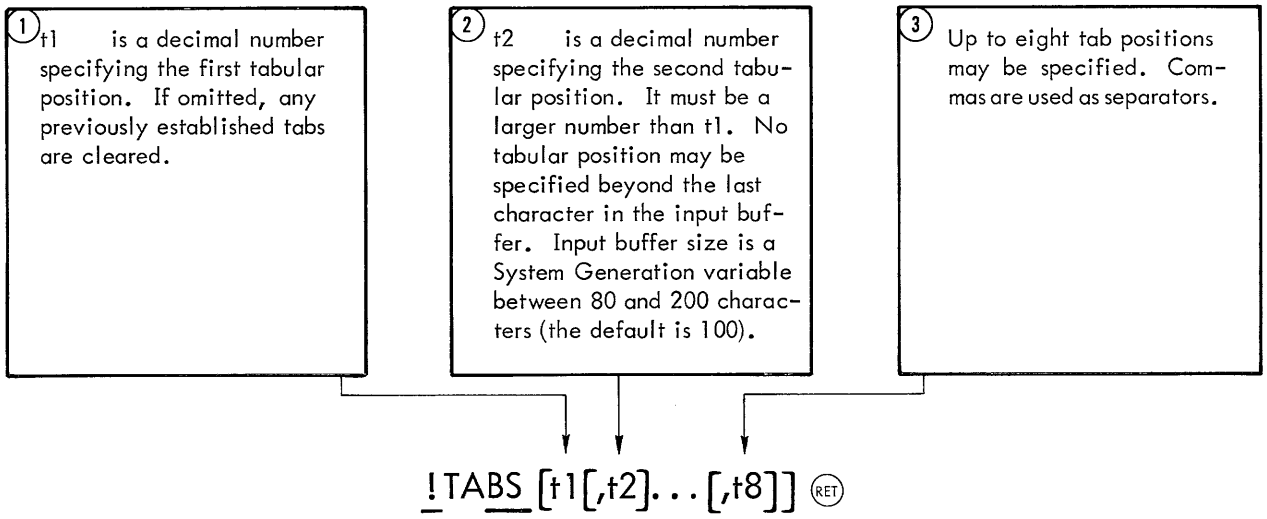
```
!ASSIGN M:DO,(HERE)RET
!
```

This causes diagnostic output to come out on the Teletype. This command might have been issued some time after the diagnostic output had been assigned to another device; however, since the normal default device is the Teletype, it would have been sufficient to command

```
!ASSIGN M:DO RET
!
```

TABS (Set Tabs)

TABS causes tab positions to be established for input from the user's Teletype.



When Teletype input is typed, the input image is spaced to the next tab position by striking the ESC key and then striking the letter I. If no tab has been established beyond the present column, a question mark is echoed and the input image is spaced by one column.

Example:

```
!TABS 5,10,15,20 RET
!
```

In this example, tab settings are established at columns 5, 10, 15, and 20. The command clears any previously established tab settings.

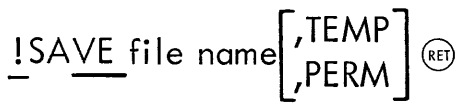
SAVE (Save Activity)

SAVE causes the machine environemnt (i.e., register contents, etc.) and the contents of all core areas used in the current session to be saved as a file in secondary storage, so that the present activity may be resumed at a later time.

(1) file name is 1-8 alphanumeric characters, (i.e., A-Z, 0-9, #, $, *, %, :, @, -, or ␣). It is a name selected by the user to identify the file in which the current activity is to be saved. If the name is that of an existing file, the contents of that file are replaced.

(2) TEMP specifies that the file is not to be saved after the user logs out of the system.

PERM specifies that the file is to be saved after the user logs out of the system.



Prior to saving the current activity, BTM automatically closes and saves all open files. However, the user is responsible for the positioning of any files before giving the SAVE command. Thus, it usually is not possible to successfully resume an assembly, compilation, or load.

A SAVE command (or any Executive command except Escape) may be given only when the BTM Executive is in control of the system, as indicated by the "!" prompt character.

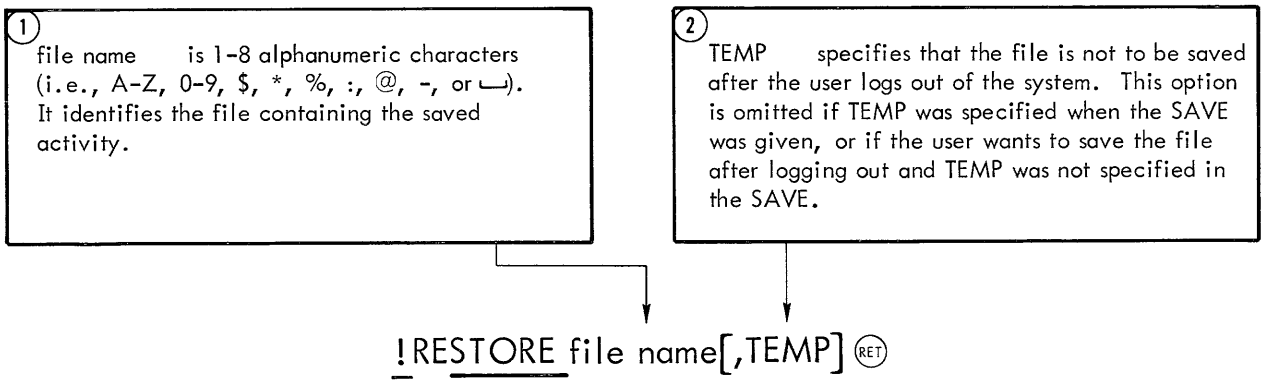
Example:

```
!SAVE ALL-THIS RET
!
```

In this example, the current activity is saved in a permanent file named ALL-THIS. The activity can be resumed later by specifying the name ALL-THIS in a RESTORE command (see description of RESTORE command).

RESTORE (Restore Activity)

RESTORE causes the machine environment and core information of a previously saved activity to be restored (see description of SAVE command), so that the previous activity may be resumed.



Example:

```
!RESTORE ERSTWILE RET
!
```

In this example, the activity previously saved in file ERSTWILE is restored, and processing can be resumed by means of a PROCEED command (see description of PROCEED command).

Escape (Return to Executive)

Escape causes control of the system to be returned to the BTM Executive. This is the only Executive command that may be given when the Executive is not in control.

1. Striking the (ESC) key twice in succession returns control to the Executive if a BTM subsystem has control.

2. Striking the (ESC) key four times in succession returns control to the Executive if a BTM subsystem is not in control (e.g., if a user's program is executing).

When the Executive gains control of the system, it prompts the user terminal with an exclamation character. Any Executive command may then be given or any BTM subsystem may be called.

Example:

```
!FERRET
≥D TROP (RET)
≥ (ESC) (ESC)
!
```

In this example, the user returns control to the Executive from the FERRET subsystem by striking the (ESC) key twice.

PROCEED (Resume Processing)

PROCEED causes resumption of an activity that has been interrupted by an Escape (see description of Escape function).

!PROCEED

If the user wants to resume a previously saved activity (see description of SAVE command), he must give a RESTORE command followed by a PROCEED command.

Example:

```
!SAVE THIS (RET)
!RESTORE THIS,TEMP (RET)
!PROCEED
≥
```

In this example, an interrupted subsystem activity is restored and processing resumed.

BYE (End Session)

BYE causes the user to be logged out of the system.

```
!BYE
```

Following the BYE command, the Executive prints the date and time followed by the number of 512-word RAD granules used, minutes of CPU time used, minutes of I/O time used, and minutes of CPU time devoted to Monitor service functions.

Example:

```
!BYE
10/15/70 8:20
RAD SPACE 07
CPU         03.467
I/O         05.156
SERVICES    04.315
```

4. BASIC SUBSYSTEM

GENERAL

This section outlines the method of writing, compiling, and executing statements and programs in BASIC from the user's terminal. On-line BASIC operates in two distinct modes: edit, and compilation and execution (see Figure 4-1).

EDIT MODE

Programs may be created at the terminal and saved on disc. Previously saved programs may be retrieved and revised and/or loaded for further execution. The system is initially in the edit mode with an empty text area.

COMPILATION AND EXECUTION MODE

In this mode, the program constructed during the editing phase is compiled and, if no errors exist, executed. If compilation errors are found, editing mode is restored automatically, and the text area will contain the program text just compiled.

Program execution may be done immediately, using the on-line system in desk calculator fashion. Direct use is not possible in the edit mode.

BASIC OPERATIONS

The BASIC subsystem provides for the following operations:

1. Enter a new program and run.
2. Load an old program and run.
3. Enter statements and execute directly.

The BASIC subsystem is entered by typing BA following the executive prompt character (!). The Executive types the rest of the word BASIC, gives a carriage return, and then (after a brief pause) the subsystem types its prompt character (>) to indicate that it is ready to receive a command.

If a BASIC statement is typed (beginning with a valid line number), or if one of the editing commands is typed, the editing mode is entered. Subsequent typing of RUN or FAST will effect the compilation and execution mode. If program execution is unsuccessful, corrections can be made in the editing mode and execution retried. The command CLEAR provides a fresh start, wiping out all previous program statements from the computer memory. It is necessary to request RUN or FAST before entering any direct program statements. This may be done upon initial entry to BASIC or immediately after a CLEAR command. All BASIC statements to be executed directly are typed without line numbers.

Statement lines of a program may be entered into the computer by typing each line following the prompt character (>). All statements except those executed directly must begin with a line number of up to five numeric digits, with the total line length not exceeding 85 characters.

Statements are compiled according to line number. If an existing line has the same number, it will be deleted; following this, the new statement will be syntax-checked and, if valid, inserted. If it is invalid, it will be deleted and a BASIC diagnostic message will be typed.

COMPILATION AND EXECUTION

Upon the command RUN (RET), compilation of the program begins. FAS[T](RET) causes compilation in the fast mode without diagnostics. If compilation is successful, the program is executed.

Should the program require input from the terminal, the character "?" will be typed during execution to prompt each line of input.

When execution has been completed, the subsystem again prompts with ">". At this point, either further editing operations may be performed, or a CLEAR command may be given followed by statements for direct "desk calculator" operations.

DIRECT EXECUTION OF STATEMENTS

Statements for direct execution must be typed without line numbers. Any valid BASIC statement may be used except the following.

DATA	PRINTUSING
DEF	FOR
DIM	NEXT
Image	Any statement containing ON

Direct execution is especially useful for on-line debugging and verification. By preceding a BASIC program with a STOP instruction and then giving a RUN command, the user may effect compilation and obtain any diagnostic messages prior to attempting program execution. The user may then test a selected portion of the compiled program, for example, by typing a GOTO statement while still in the direct mode. Execution will proceed to the next STOP, PAUSE, or END statement (unless execution is aborted due to an error condition).

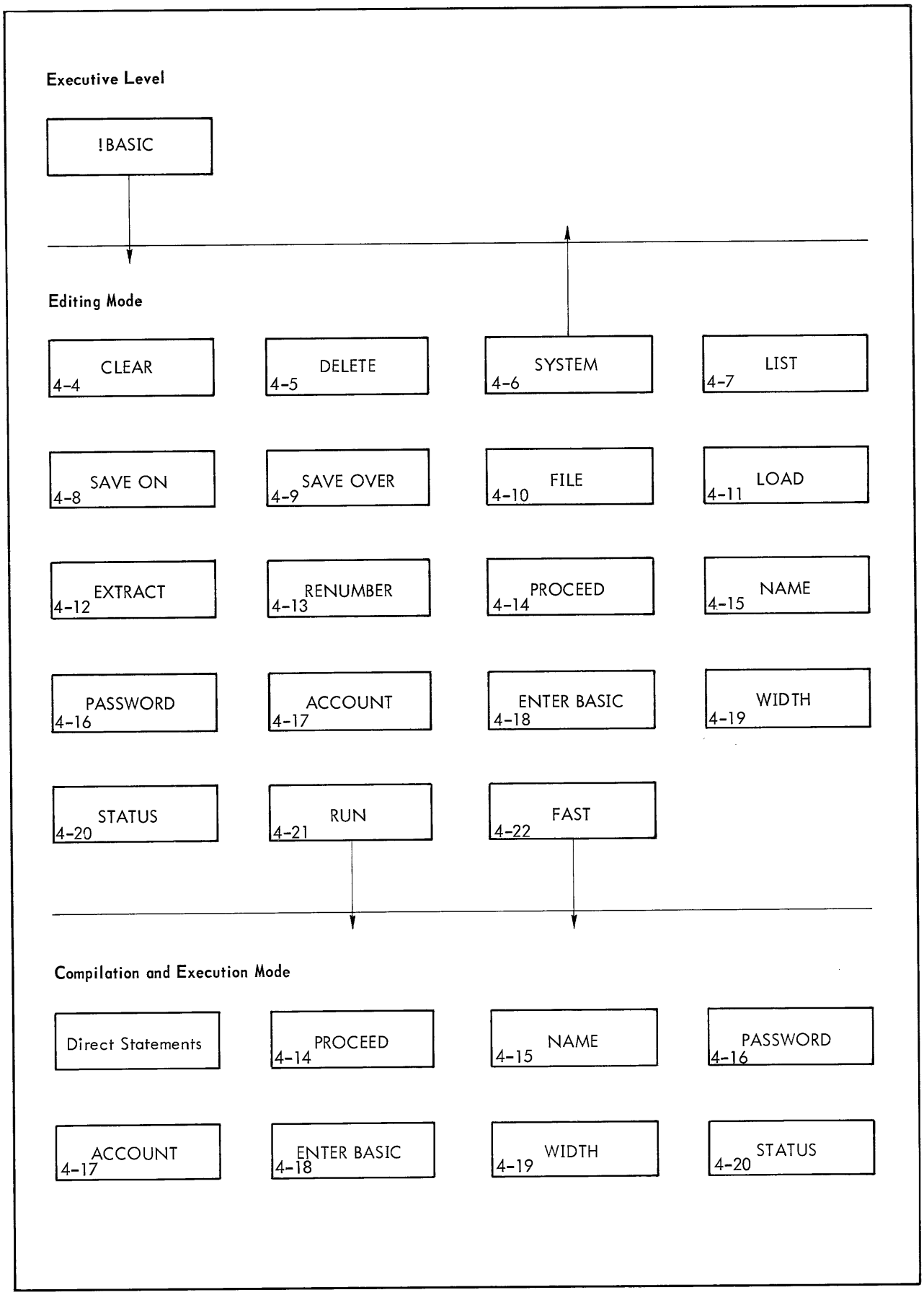


Figure 4-1. BASIC Editing Commands (and Page Numbers)

CLEAR (Erase Text Area)

CLEAR causes everything in the text editing area to be erased. It should be used if the current program is to be deleted in its entirety.

≥CLE[AR] ⓃⓁ

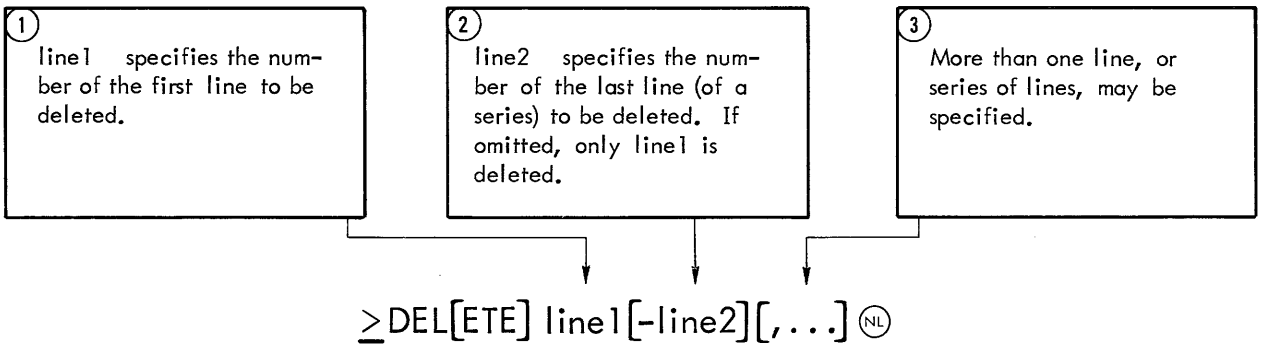
Example:

```
!BASIC
≥10 LET X=2 ⓃⓁ
≥20 LET Y=3 ⓃⓁ
≥30 LET Z=5 ⓃⓁ
≥CLE ⓃⓁ
≥
```

In this example, the command "CLE" causes lines 10, 20, and 30 to be deleted.

DELETE (Delete Lines)

DELETE causes specified lines to be deleted from the text editing area.



Example:

```
!BASIC
≥10 LET X=1 (NL)
≥20 LET X=2, Y=3 (NL)
≥30 LET Q=X+Y (NL)
≥40 PRINT Q (NL)
≥DEL 10 (NL)
≥
```

In this example, the command "DEL 10" causes line 10 to be deleted. The remainder of the program is retained in the text editing area.

SYSTEM (Return to Executive)

SYSTEM causes control to be returned to the BTM Executive.

≥SYS[TEM] ⓃⓁ

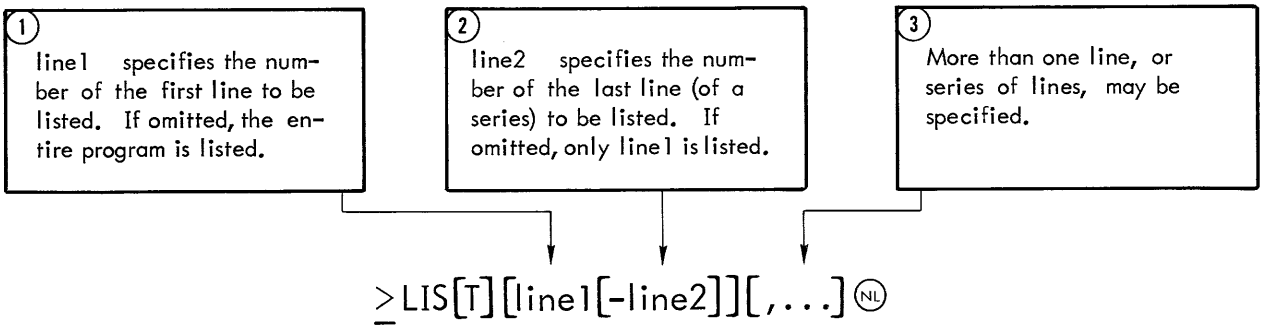
Example:

```
!BASIC
≥LOAD SUMPRO ⓃⓁ
≥RUN ⓃⓁ
          GR 1        GR 2
           450         1761
           520         1981
           610         2646
           621         2696
           788         2786
≥SYS ⓃⓁ
!
```

In this example, the "SYS" command returns control to the Executive following completion of BASIC operations.

LIST (List Lines)

LIST causes specified lines of a program to be listed on the user terminal.



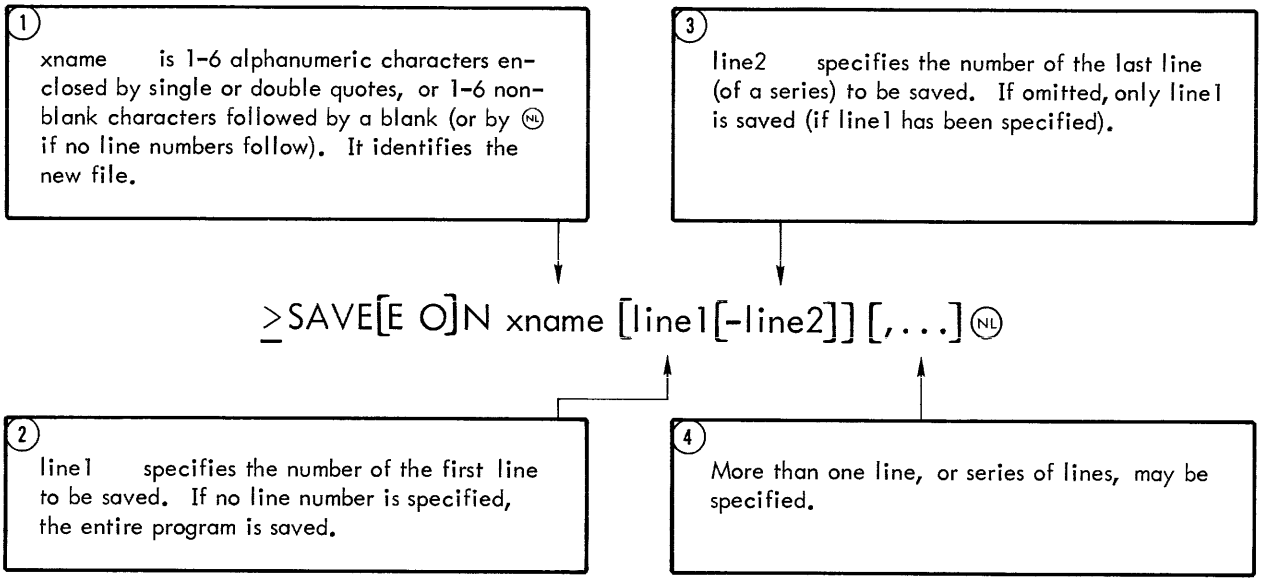
Example:

```
!BASIC
≥10 LET A=1NL
≥15 LET A=A+1NL
≥20 IF A<45 THEN 15NL
≥LIST 5-15NL
10 LET A=1
15 LET A=A+1
≥
```

In this example, the command "LIST 5-15" causes lines 10 and 15 to be listed. The contents of the text editing area are unaffected.

SAVE ON (Save on Temporary File)

SAVE ON causes specified lines of the current program to be saved in a newly created temporary file.



Although it is unlikely that the programmer would want to specify a password for a temporary file, a password will apply to the file created by a SAVE ON command if a PASSWORD command is currently in effect (see description of PASSWORD command). If an account number has been specified, it is reset prior to execution of a SAVE ON command and must be respecified if needed for subsequent file operations.

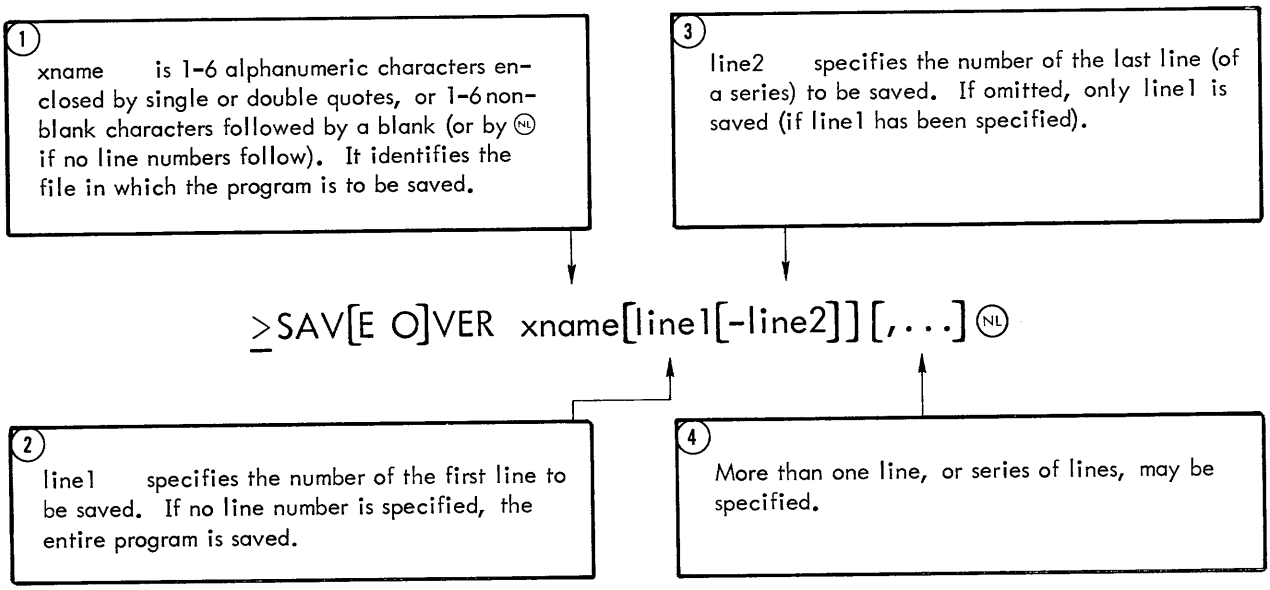
Example:

```
!BASIC
≥10 INPUT X NL
≥20 INPUT Y NL
≥30 LET Z=X-Y NL
≥40 PRINT Z NL
≥50 END NL
≥SAVN PROGRM NL
≥
```

In this example, lines 10 through 50 are saved in a temporary file named PROGRM. The contents of the text editing area are unaffected.

SAVE OVER (Save on Permanent File)

SAVE OVER causes specified lines of the current program to be saved in a permanent file. The file may have the same name as an existing file.



If a PASSWORD command is currently in effect (see description of PASSWORD command), the password will apply to the file specified in a SAVE OVER command. If an account number has been specified, it is reset prior to execution of a SAVE OVER command and must be respecified if needed for subsequent file operations.

Example:

```
!BASIC
≥10 LET A=2 NL
≥20 LET B=3 NL
≥30 LET C=4 NL
≥40 LET D=50 NL
≥SAVE OVER "XNAY" 10-20,30-40 NL
≥
```

In this example, lines 10 through 20 and 30 through 40 are saved in a permanent file named XNAY. The contents of the text editing area are unaffected.

FILE (Save on Runfile)

FILE causes the current program to be saved in the "runfile" maintained by the BASIC subsystem.

≥FIL[E] ⓃⓁ

A program saved via a FILE command replaces the previous contents of the runfile, if any. The saved program can be retrieved for later use (see description of "LOAD", "RUN", and "FAST" commands).

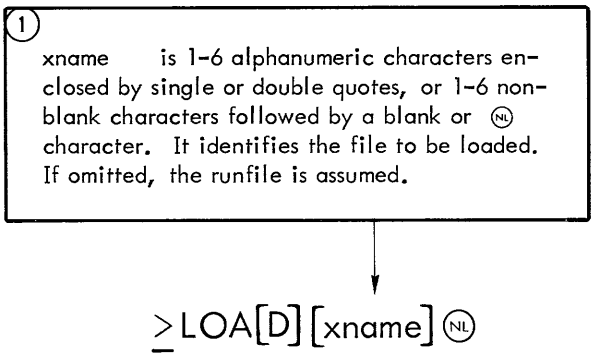
Example:

```
!BASIC
≥10 LET B=5 ⓃⓁ
≥20 LET C=10 ⓃⓁ
≥30 LET D=B+C ⓃⓁ
≥40 PRINT D ⓃⓁ
≥50 END ⓃⓁ
≥FILE ⓃⓁ
≥
```

In this example, steps 10 through 50 are saved in the runfile. The contents of the text editing area are unaffected.

LOAD (Load Program)

LOAD causes a specified file, containing a previously saved BASIC program, to be retrieved and placed in the text editing area.



The contents of the specified file are merged with the previous contents of the text editing area, replacing the contents of correspondingly numbered lines, if any. If a LOAD command is used in the compilation and execution mode (see description of "RUN" and "FAST" commands), the text editing area is cleared prior to loading from the file.

Note that no compilation is done in response to a LOAD command, and the file loaded must be in standard BASIC source format.

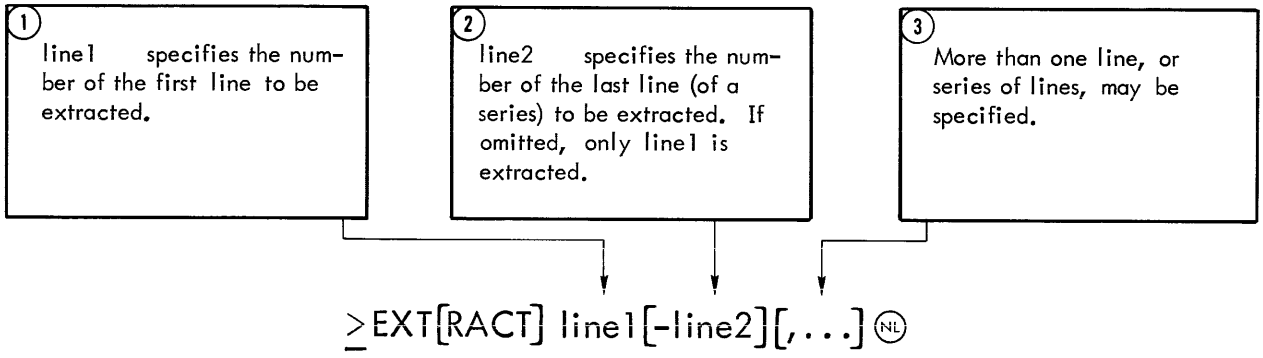
Example:

```
!BASIC
>LOA OLDPRO NL
>
```

In this example, the contents of file OLDPRO are loaded into the text editing area.

EXTRACT (Extract Lines)

EXTRACT causes everything except specified lines to be deleted from the text editing area.



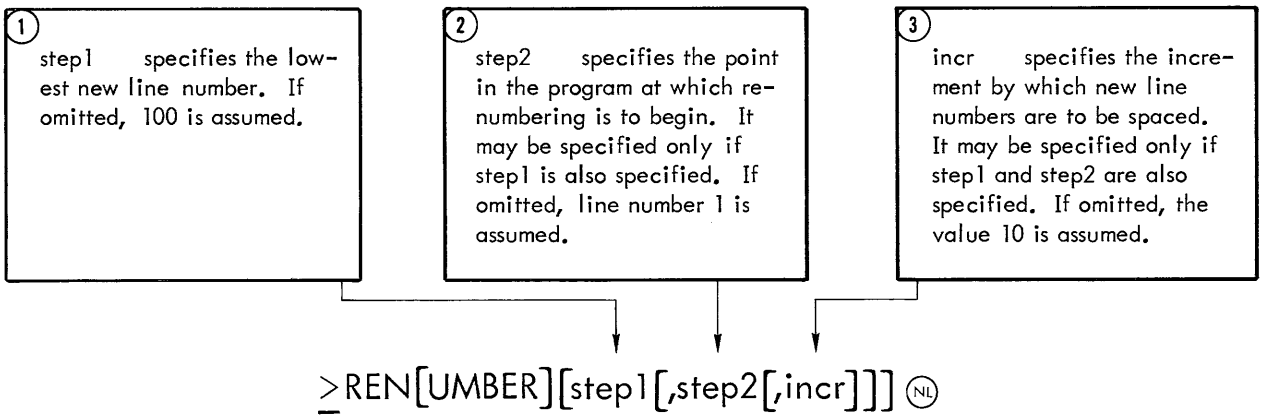
Example:

```
!BASIC
≥10 LET X=1 NL
≥20 LET X=2, Y=3 NL
≥30 LET Q=X+Y NL
≥40 PRINT Q NL
≥EXT 20,30-40 NL
≥
```

In this example, line 10 is deleted and lines 20, 30, and 40 are retained in the text editing area.

RENUMBER (Renumber Lines)

RENUMBER causes program steps to be renumbered, starting with a specified line number and renumbering in specified increments.



Example:

```
!BASIC
≥10 INPUT A NL
≥20 INPUT B NL
≥30 INPUT C NL
≥40 LET P=B+C NL
≥50 PRINT P NL
≥60 END NL
≥REN 10,20,30 NL
≥
```

In this example, steps 20 through 60 are renumbered as lines 10 through 130. The original step 10 is not renumbered but is replaced by the step originally numbered 20.

PROCEED (Continue After Escape)

PROCEED causes BASIC processing to continue after being stopped by use of the ESC key (see description of Executive commands).

≥PRO[CEED] NL

Since I/O file conditions are not necessarily preserved when processing is interrupted, the resumption of compilation or program execution may not always be successful.

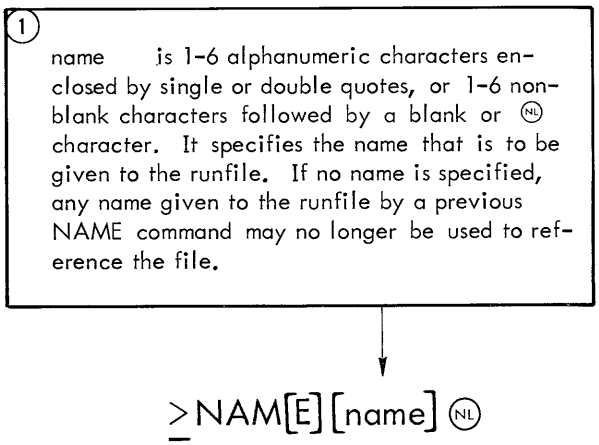
Example:

```
!BASIC
≥10 LET A=1 NL
≥20 PRINT A NL
≥30 LET A=A+1 NL
≥40 IF A>4 THEN 60 NL
≥50 GOTO 20 NL
≥60 END NL
≥RUN NL
 1
 2
 3 ESC ESC
≥PRO NL
 4
≥
```

In this example, program execution is interrupted by an Escape and then resumed by use of the PROCEED command.

NAME (Name Runfile)

NAME causes a specified name to be given to the runfile.



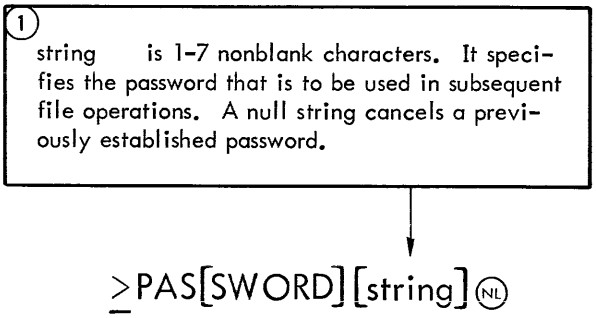
Example:

```
!BASIC
≥10 LET C=3 NL
≥20 PRINT C NL
≥30 END NL
≥FILE NL
≥CLEAR NL
≥NAME "RUNFIL" NL
≥LOAD RUNFIL NL
≥
```

In this example, the program created at the console is stored in the runfile, and the text editing area is cleared. The runfile is given the name RUNFIL and then the program is returned to the text editing area via a LOAD command.

PASSWORD (Set/Reset Password)

PASSWORD causes a specified password to be established for use in subsequent SAVE ON, SAVE OVER, LOAD, RENUMBER, RUN, or FAST file operations. If a null password is specified, any previous password is reset.



A password is not used in runfile operations, and such operations (e.g., FILE or LOAD) cancel any previously established password.

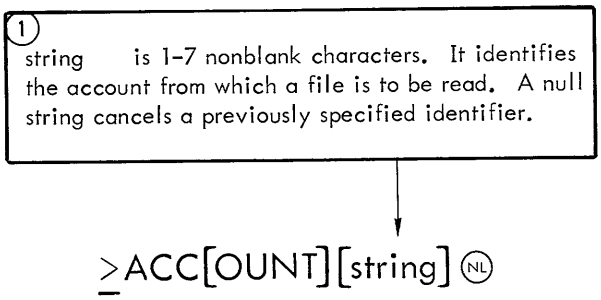
Example:

```
!BASIC
>10 LET C=99 NL
>20 PRINT C NL
>30 END NL
>PASSWORD SECRIT NL
>SAVVER AFILE NL
>SYS NL
!BASIC
>PAS SECRIT NL
>LOAD AFILE NL
>
```

In this example, the password SECRIT is given to file AFILE and a return to the BTM Executive is made. BASIC is then called and the password reestablished to allow AFILE to be loaded into the text editing area.

ACCOUNT (Set/Reset Account)

ACCOUNT causes a specified account identifier to be established for use in reading files from an account other than the current one. If a null identifier is specified, any previously established identifier is reset.



Since BASIC does not allow files to be written outside the log-in account, SAVE ON or SAVE OVER operations reset any account identifier.

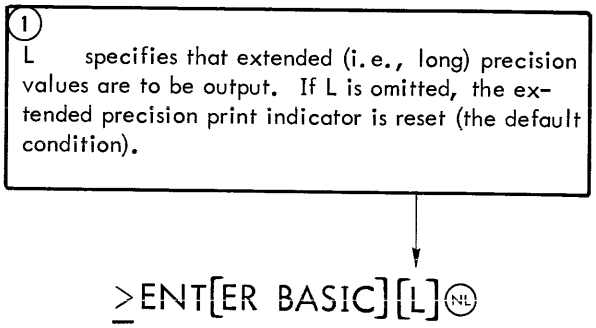
Example:

```
!BASIC
>ACC :SYS
>LOAD SUMFIL
>SAVN TFIL
>CLE
>ACC :SYS
>LOAD ANOTHR
>
```

In this example, account :SYS is specified as the account from which file information is to be read. Since the command "SAVN TFIL" resets the account identifier, it must be specified again before reading another file.

ENTER BASIC (Set/Reset Precision)

ENTER BASIC causes the extended precision print indicator to be set or reset.



Example:

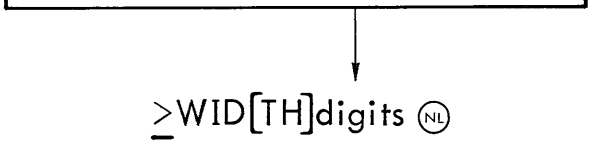
```
!BASIC
>LOAD PIE NL
>ENT L NL
>RUN NL
 3.1415926535897932
>
```

In this example, the output of the program PIE is printed with extended precision.

WIDTH (Set Print Width)

WIDTH causes the print width to be changed from the default value of 72.

(1) digits specifies the desired print width. The specified value must not be less than 32 or greater than 85.



>WID[TH]digits (NL)

Example:

```
!BASIC
>LOAD XYZ (NL)
>WID 35 (NL)
>RUN (NL)
'TWAS BRILLIG, AND THE SLITHY TOVES
 DID GYRE AND GIMBLE IN THE WABE;
>
```

In this example, print width is set at 35, causing a single text string to be printed in two lines.

STATUS (Give Status)

STATUS causes the status of the current program to be printed. One of three responses is possible: EDITING, COMPILING, or RUNNING. If program execution is in progress, an ESCAPE (i.e., (ESC) (ESC)) must be used before the STATUS command can be keyed in. The line number of the current statement is printed preceding the RUNNING message.

≥STA[TUS] (NL)

Example:

```
!BASIC
≥10 LET K=1 (NL)
≥20 PRINT K (NL)
≥30 LET K=K+1 (NL)
≥40 IF K>4 60 (NL)
≥50 GOTO 20 (NL)
≥60 END (NL)
≥RUN (NL)
 1
 2
 3 (ESC) (ESC)
≥STA (NL)
30 RUNNING
≥PRO (NL)
 4
≥
```

In this example, the message "30 RUNNING" indicates that program execution was interrupted at step 30.

RUN (Compile, Check, and Run)

RUN causes BASIC to compile a program from the runfile or from the text editing area if no runfile exists. If no errors are found, the program is then executed.

≥RUN (NL)

If no program exists in the runfile or the text editing area when the RUN command is used, BASIC compiles and executes unnumbered statements directly as they are input from the console. However, the following types of BASIC statements cannot be executed directly and must not be used in the direct execution mode:

DATA	PRINTUSING
DEF	FOR
DIM	NEXT
Image	Any statement containing "ON"

Compilation takes place in the "safe" mode in which the subscripts of all variables are checked against absolute dimensions. If input is required from the console during execution, BASIC types a question mark as a prompt character. When program execution is complete, the usual ">" prompt is typed.

Example:

```
!BASIC
≥10 LET A=5(NL)
≥20 PRINT A(NL)
≥30 END(NL)
≥RUN(NL)
 5
≥
```

In this example, the program in the text editing area is executed, causing the character "5" to be printed.

FAST (Compile and Run)

FAST causes BASIC to compile a program from the runfile or from the text editing area if no runfile exists. The program is then executed.

>FAS[T] NL

If no program exists in the runfile or the text editing area when the FAST command is used, BASIC compiles and executes unnumbered statements directly as they are input from the console.

However, the following types of BASIC statements cannot be executed directly and must not be used in the direct execution mode:

DATA	PRINTUSING
DEF	FOR
DIM	NEXT
Image	Any statement containing "ON"

When compilation takes place in the "fast" mode, no subscript checking is done (see description of RUN command). If input is required from the console during execution, BASIC types a question mark as a prompt character. When program execution is complete, the usual ">" prompt is typed.

Example:

```
!BASIC
>FAST NL
>LET B=4 NL
>PRINT B NL
 4
>
```

In this example, a program is executed in the direct mode as it is input from the console, causing the character "4" to be printed.

BASIC MESSAGES

Message	Meaning
xxxxxx LINE # ERROR	The indicated line number is too long. The line must be retyped with a number less than six digits in length.
LINE TOO LONG	More than 85 characters were typed on one line. The line must be retyped using fewer characters.
SPACE LIMIT NEAR	Not more than 500 characters may be added to the program in the text editing area.
PROGRAM TOO LARGE	No lines may be added to the program in the text editing area. The contents of this area may be saved in a file and the text editing area may then be cleared to allow more lines to be created. (See descriptions of FILE, SAVE ON, SAVE OVER, and CLEAR commands.)
NO PROGRAM	A specified line or series of lines does not exist in the text editing area. The user may type in the missing information.
ILLEGAL LOAD	An illegal line has been encountered in loading a program into the text editing area. The line, printed prior to this message, must be retyped correctly.
UNABLE TO OPEN	A specified file does not exist or cannot be accessed for some other reason (e.g., incorrect password).
ILLEGAL	A RENUMBER command has been encountered with illegal syntax. The command must be retyped with proper syntax.
RUN? ILLEGAL	A direct statement was encountered while in the editing mode. The RUN or FAST command may be used to enter the compilation and execution mode.

5. EDIT SUBSYSTEM

GENERAL

With the EDIT subsystem, the on-line user may enter data from the terminal and build files on the RAD. Records may be added to and deleted from files and changes may be made within records. EDIT may be used with files of source language for processors such as BASIC, FORTRAN, Symbol, etc., and with job control statements for terminal batch job entry. The EDIT subsystem enables the following:

1. Creation of sequence-numbered files on the RAD.
2. Deletion of one or more records from an existing file.
3. Insertion of single records or series of records into an existing file.
4. Replacement of single records or series of records with new records.
5. Reordering records and record groups within a file.
6. Changes within any record in a file. To facilitate this, there is a group of commands to substitute, insert, and shift strings of characters within a record.
7. Copying an existing file.

EDIT COMMANDS

The EDIT subsystem is entered through a command to the Executive which, in turn, completes the word, then passes control to the subsystem. On the following line, EDIT types an asterisk which is the prompt character for all EDIT commands.

There are three types of EDIT commands, arranged in a hierarchical relationship (see Figure 5-1).

1. File Commands: Commands that apply to an entire file.
2. Record Commands: Commands that act upon one record or a group of records within a file.
3. Intra-Record Commands: Commands that make changes within an individual record. These generally manipulate character strings within a specific record.

RECORD COMMANDS

Every record in a file contains a unique sequence number, and sequence numbers are referenced in the execution of many commands that operate on records and groups of records. Sequence numbers have an implied decimal point and, when listed by the Editor, are displayed with three decimal places (e.g., 20.000).

INTRA-RECORD COMMANDS

Note that before any commands that do operations within individual records may be typed, either SS, ST, or SE must be given to specify the starting record sequence number. Also, multiple intra-record commands may be typed on a line separated by semicolons (;).

FILE RECORD FORMAT

All file records handled by EDIT are from 1 to 140 characters long. All records are ordered in the file according to sequence number, which is the key by which records are specified in EDIT commands. If EDIT is used to build a file, successive sequence numbers will be automatically assigned. If it is desired to use EDIT with differently formatted

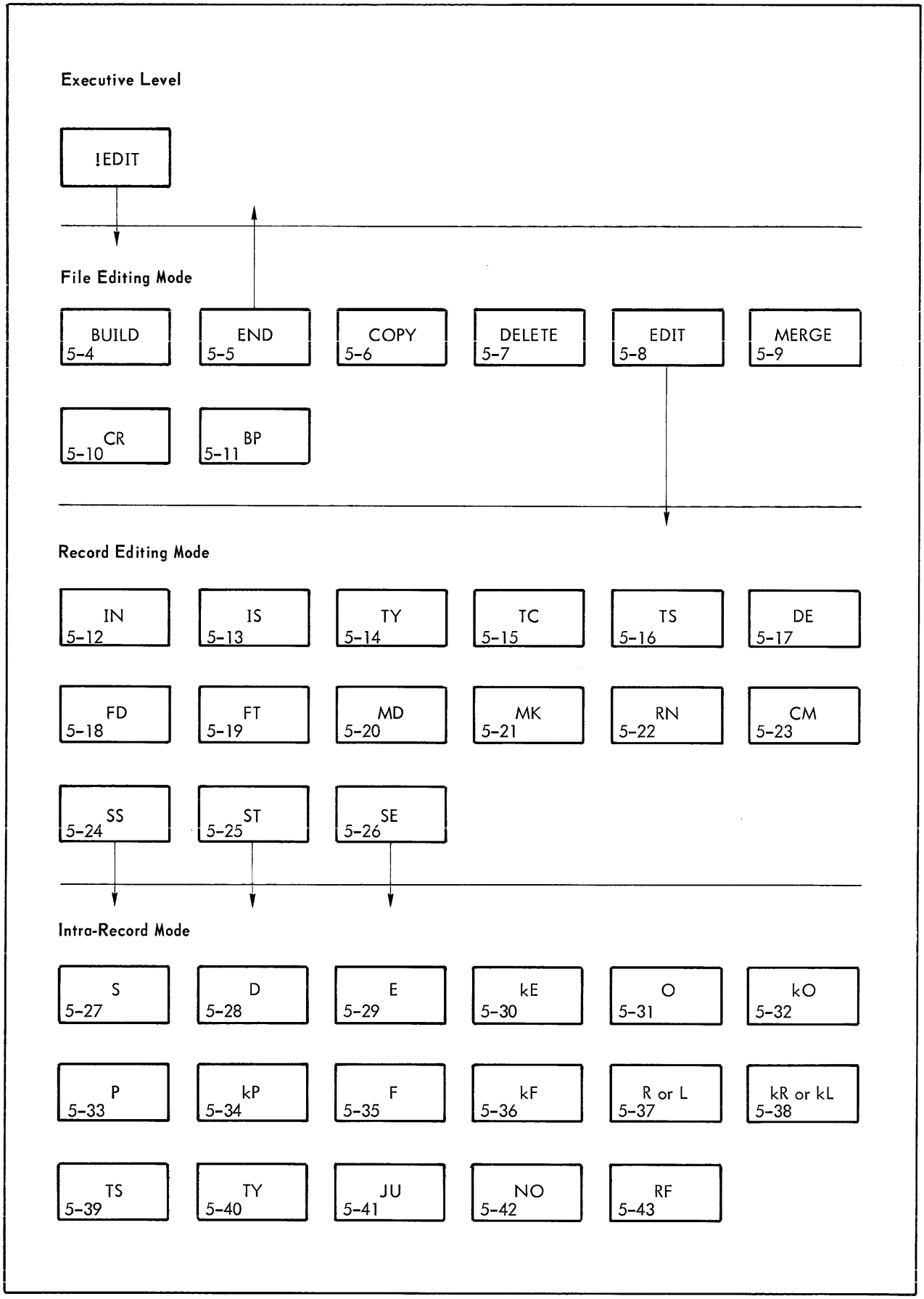


Figure 5-1. EDIT Commands (and Page Numbers)

files, it is necessary to use the COPY command to create a copy of the file with sequence numbers in correct format; in the event that this is necessary, a message will be typed. Since EDIT commands make immediate changes to the actual records in a file, it is advisable to make a backup copy of any file being updated in the event erroneous commands cause loss of data.

EDIT MESSAGES

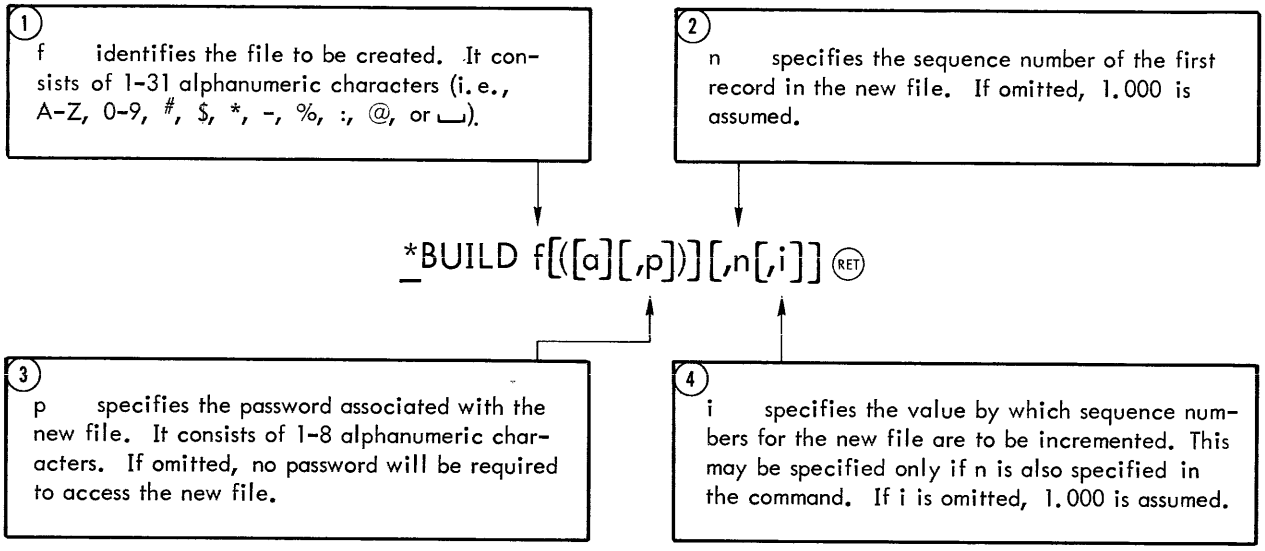
There are a number of messages that the EDIT subsystem may return to the operator in the course of executing commands. Table 5-1 lists the conventions that are observed with the various messages.

Table 5-1. EDIT Message Conventions

Convention	Description	Example
..(message)	Messages preceded by two periods give comments on the progress of operations. They do not indicate unusual or error conditions	.. COPY DONE A COPY operation has been completed.
--(message)	Messages preceded by two hyphens indicate an unexpected event of which the operator should be aware. The command did not abort, but completed operations to whatever extent was possible.	--EOF HIT An end-of-file was encountered.
-(message)	A single hyphen indicates an error condition which causes the present command to be aborted, and no action will be taken on further commands typed on the same line.	-C4P1: NO SUCH REC The first parameter of the fourth command referenced a nonexistent record.

BUILD (Create a New File)

BUILD causes the Editor to create a new file in disc storage.



Following the BUILD command, the system prompts by typing a sequence number. The user then types in the first line of the new file. Each line comprises a record of up to 140 characters and is terminated by a carriage return. A null record, consisting of only a carriage return, terminates the action of the BUILD command.

Example:

```
*BUILD ALPHA-RALPHA,10,2 RET
10.000 SYSTEM   SIG7 RET
12.000 DEF      B RET
14.000 REF      A RET
16.000 B        A RET
18.000 END RET
20.000  RET
*
```

In this example, the user creates file ALPHA-RALPHA comprising five records numbered 10.000 through 18.000 (null record 20.000 does not appear in the file).

END (Exit to Executive)

END causes the Editor to close all active files and return control to the Executive.

```
*END (RET)
```

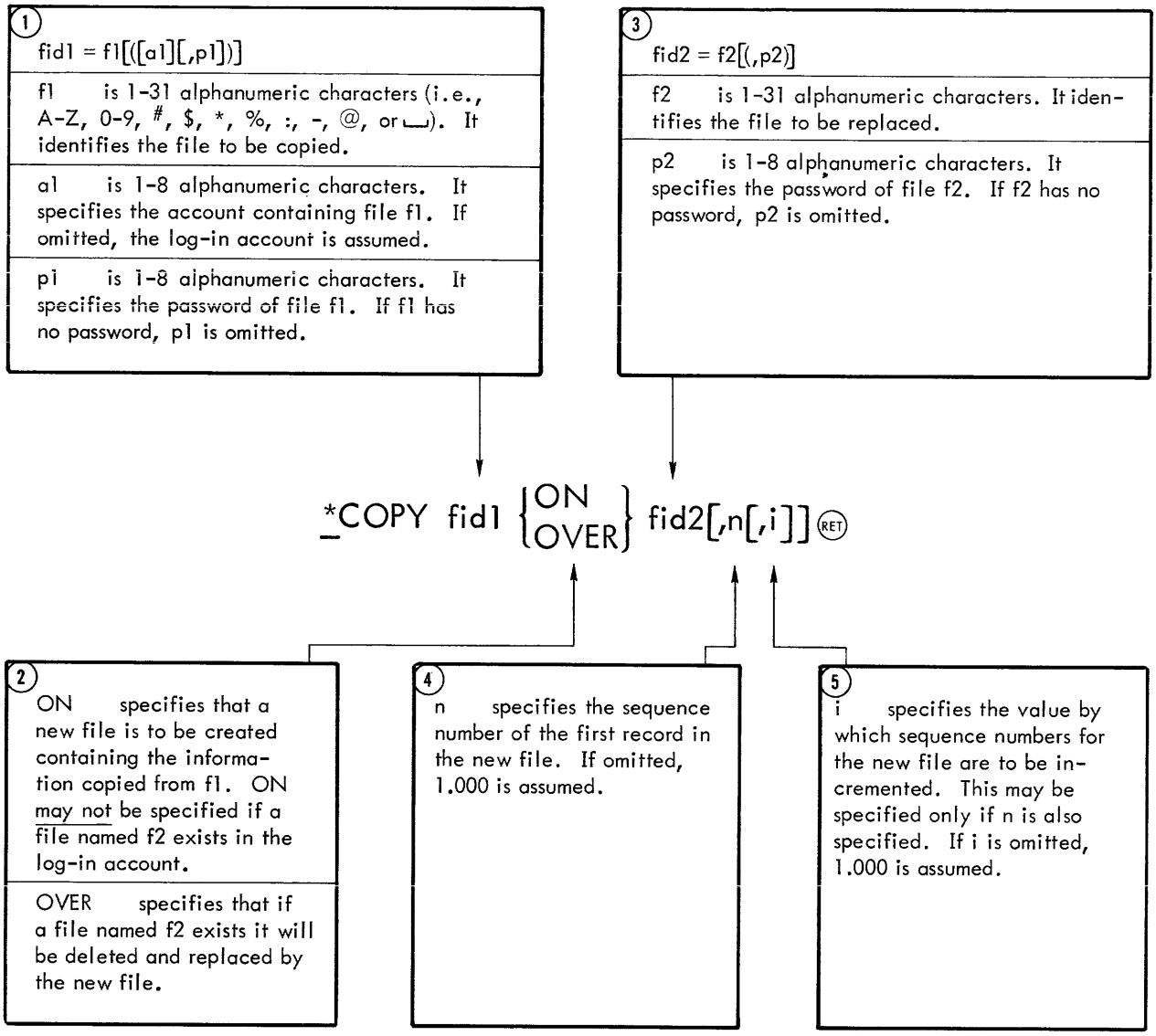
Example:

```
*END (RET)
!
```

The exclamation character on the line following the END command indicates that the Editor subsystem has returned control to the Executive. Any Executive command may then be given.

COPY (Copy a File)

COPY causes the Editor to copy a specified file.



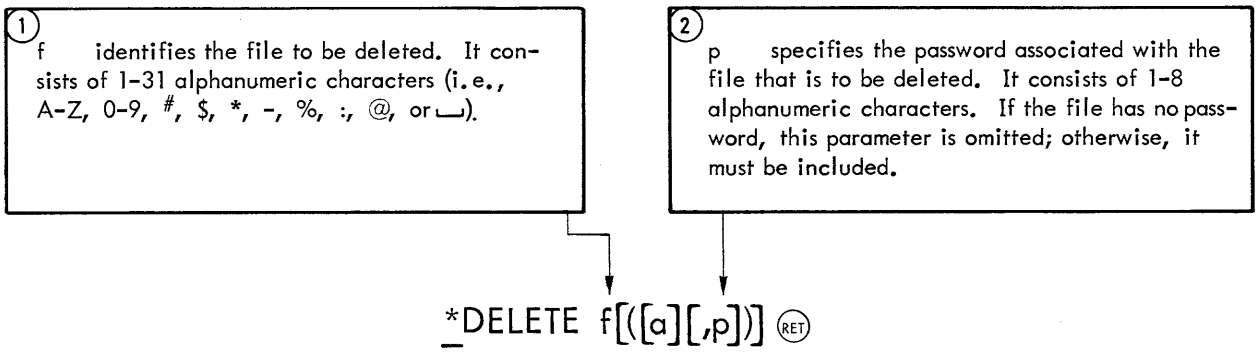
Example:

```
*COPY PELION ON OSSA(,HUSH)
..COPYING
..COPY DONE
*
```

In this example, the information from the file named PELION is copied to a new file named OSSA. The password HUSH is associated with the new file.

DELETE (Delete a File)

DELETE causes the Editor to delete a specified file from the log-in account.



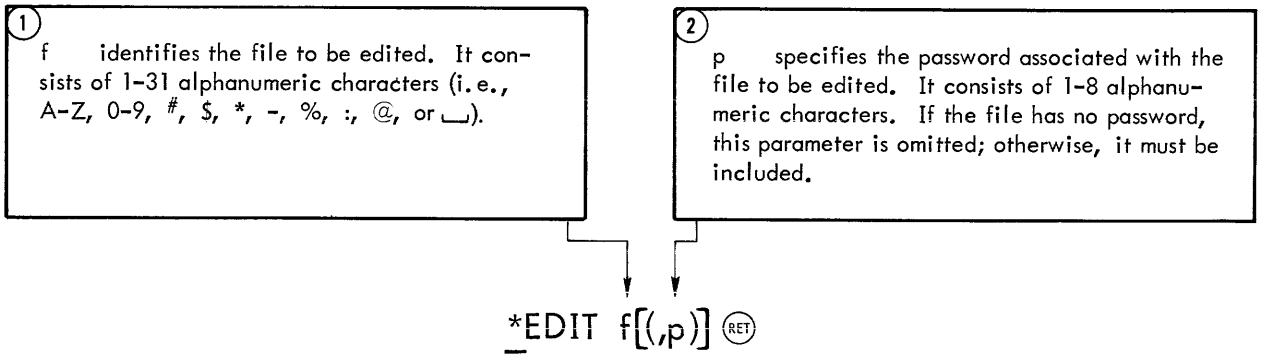
Example:

```
*DELETE SUPERNUMERARY(,COVERT) RET
.. DELETED
*
```

In this example, the file SUPERNUMERARY having the password COVERT is deleted from the log-in account.

EDIT (Edit a File)

EDIT causes the Editor to open a specified file for editing.



The EDIT command must be used to enter the record editing mode and to identify the file that is to be edited. The following commands may be used in the record editing mode: IN, TY, TC, TS, DE, FD, FT, MD, MK, RN, CM, SS, and ST. Use of any of the following commands terminates the record editing mode: BUILD, DELETE, and COPY. If an EDIT command is given while in the record editing mode, the previously open file is closed and the specified file is opened.

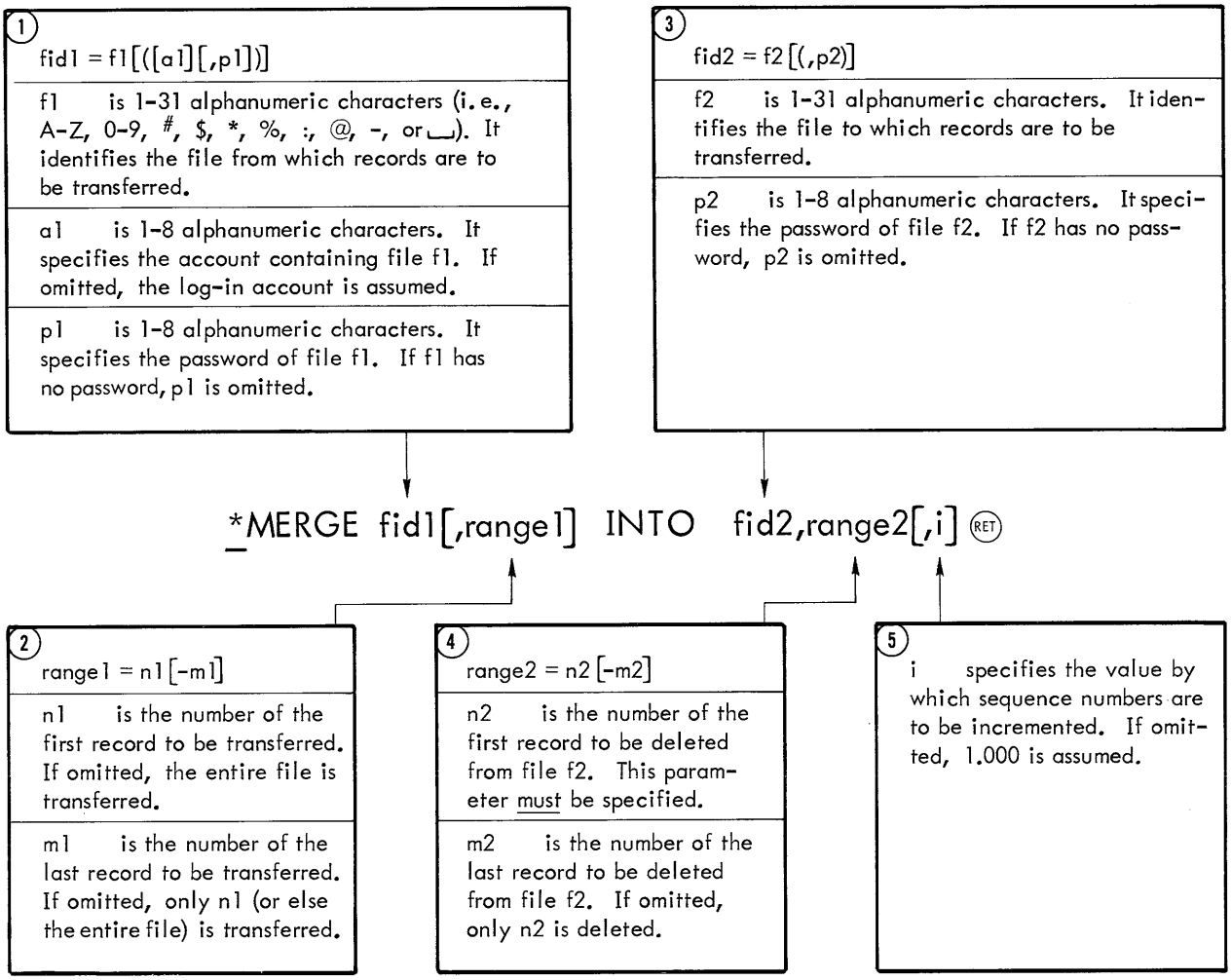
Example:

```
*EDIT PLOWBOY(,BUCOLIC)RET
*
```

In this example, the file named PLOWBOY having the password BUCOLIC is opened for editing. Any previously opened file is closed.

MERGE (Transfer Records)

MERGE causes the Editor to transfer records between specified files.



If file f1 does not exist, contains no records within the specified range, or is not in keyed format, the command is aborted. If file f2 does not exist in the log-in account, the Editor creates such a file containing the specified records.

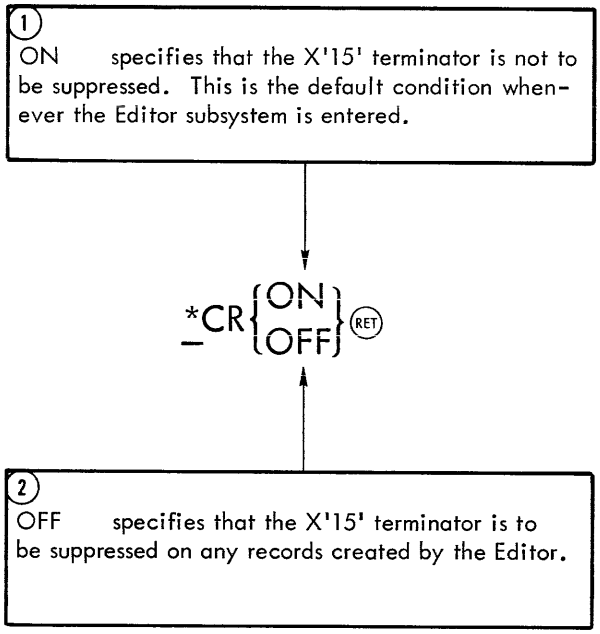
Example:

```
*MERGE NULL INTO VOID,100-125 RET
..MERGE STARTED
--DONE AT 120
*
```

In this example, file NULL comprising 21 records is transferred to file VOID beginning at sequence number 100.000 and continuing through sequence number 120.000. Any records in file VOID within the range 120.001 through 125.000 are deleted.

CR (Suppress Terminator)

CR controls the inclusion of the new-line character (X'15') that normally terminates records created by the Editor.



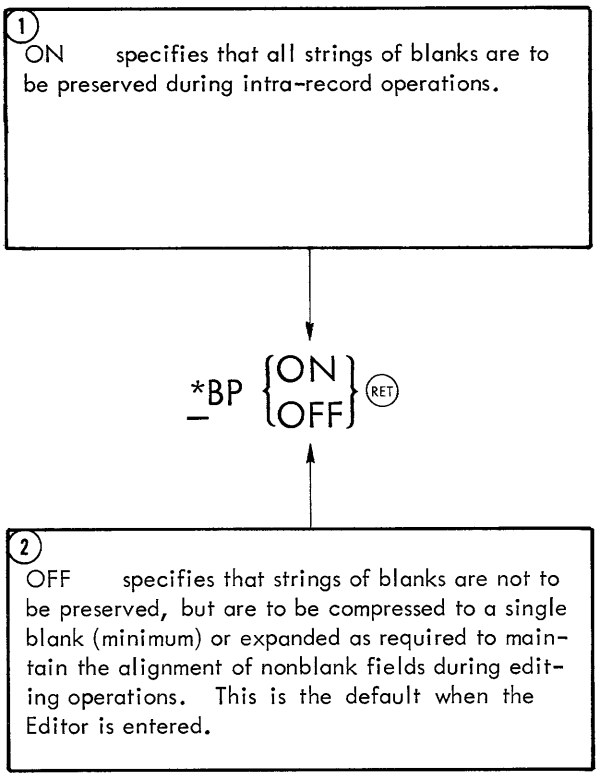
Example:

```
*CR OFF ↵
*BUILD NEWFILE ↵
1.000 HICKORY ↵
2.000 DICKORY ↵
3.000 DOCK ↵
4.000 ↵
*EDIT NEWFILE ↵
*TS 1-3 ↵
HICKORY
DICKORY
DOCK
*
```

In this example, the file NEWFILE is created without the inclusion of new-line characters at the end of each record. Note that this has no effect on the typing of records by the Teletype. However, records output to paper tape would be affected.

BP (Set Blank Preservation Mode)

BP sets the blank preservation mode on or off.



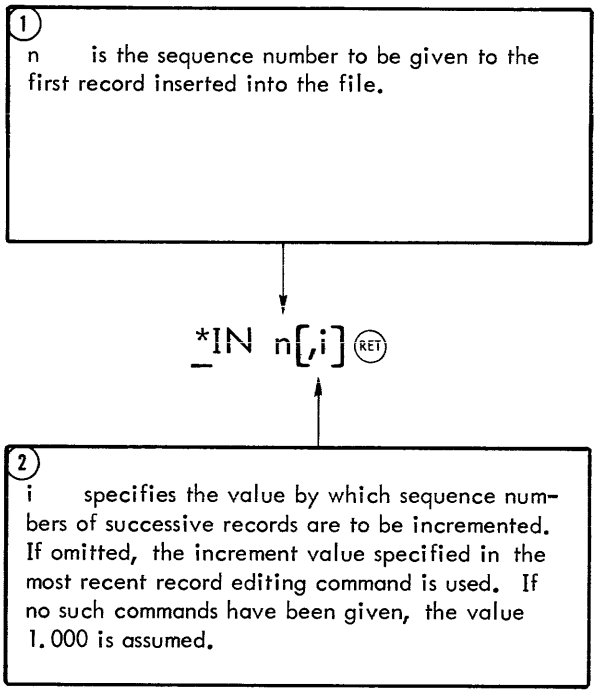
Example:

```
*EDIT OLDFILE RET
*SE 99;TS RET
  B  $+3
*/B/S/BNEZ/;TS RET
  BNEZ $+3
*BP ON RET
*SE 99 RET
*/BNEZ/S/B/;TS RET
  B $+3
*
```

In this example, record 99.000 of file OLDFILE originally has two blanks between nonblank fields. When the string "B" is replaced by "BNEZ", string "$+3" is shifted right 2 columns, compressing the intervening blank field to a single column. When "BNEZ" is replaced by "B" with blank preservation in effect, the single blank is preserved.

IN (Insert Records)

IN causes the Editor to insert records into a file that has been opened for editing (via an EDIT command).



New records are inserted beginning with record n. If a record with sequence number n already exists in the file, it is replaced by the newly inserted record. Note that existing record n is the only record that may be replaced in this way. If a subsequently inserted record would equal or exceed the sequence number of an existing record, the IN command is terminated and the console bell is rung.

The Editor prompts the user console with the sequence number of each record to be inserted. A null record (carriage return only) terminates the command.

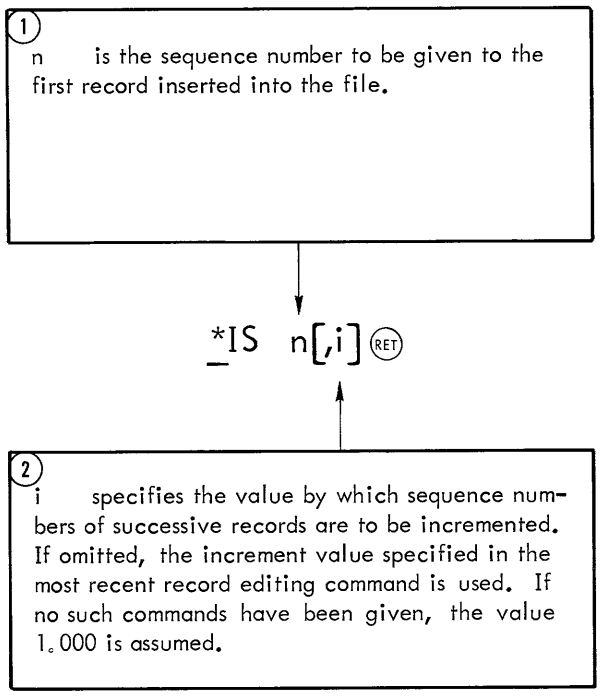
Example:

```
*EDIT ANYFILE RET
*IN 10,.5 RET
10.000 LI,R3 X'F1' RET
10.500 STW,R3 ONE RET
*
```

In this example, existing record 10.000 of file ANYFILE is replaced by a new record typed by the user. New record 10.500 is also inserted into the file. Because a record with sequence number 11.000 already exists, the command terminates and the bell is rung. ANYFILE remains open for further editing.

IS (Insert Records Without Prompt)

IS causes the Editor to insert records into a file that has been opened for editing (via an EDIT command).



New records are inserted beginning with record n. If a record with sequence number n already exists in the file, it is replaced by the newly inserted record. Note that existing record n is the only record that may be replaced in this way. If a subsequently inserted record would equal or exceed the sequence number of an existing record, the command is terminated and the console bell is rung.

The Editor does not prompt the user console with the sequence number of each record to be inserted. A null record (carriage return only) terminates the command.

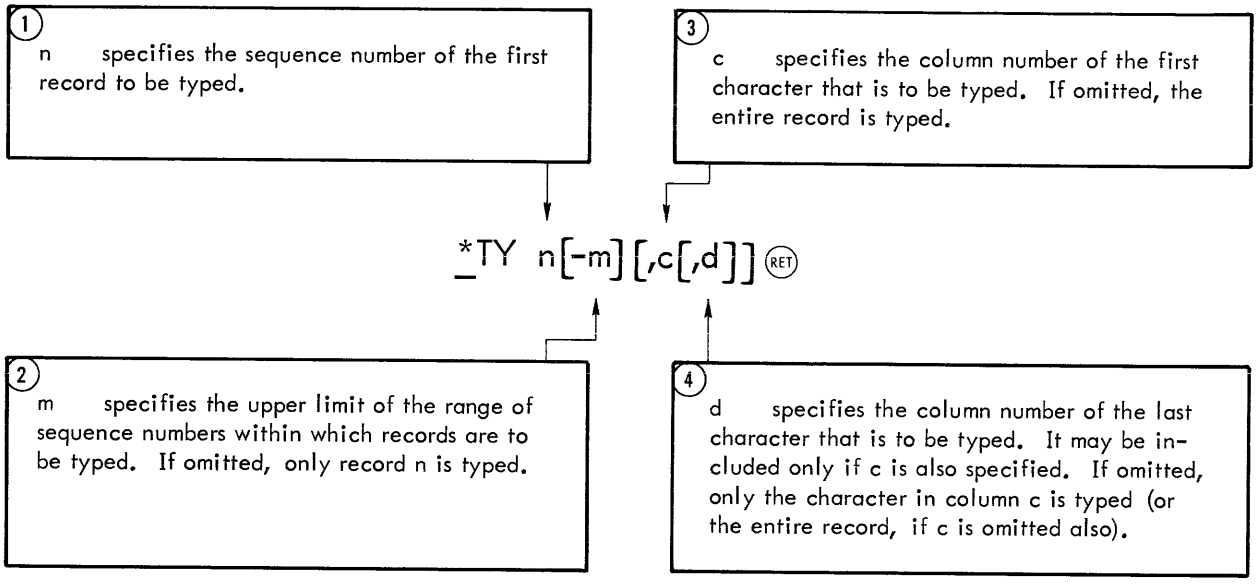
Example:

```
*EDIT ANYFILE RET
*IS 10,.5 RET
LI,R3 X'F1' RET
STW,R3 ONE RET
*
```

In this example, existing record 10.000 of file ANYFILE is replaced by a new record typed by the user. New record 10.500 is also inserted into the file. Because a record with sequence number 11.000 already exists, the command terminates and the bell is rung. ANYFILE remains open for further editing.

TY (Type Records)

TY causes the Editor to type specified columns of one or more records in the currently open file.



There is also an intra-record form of this command (see page 5-4).

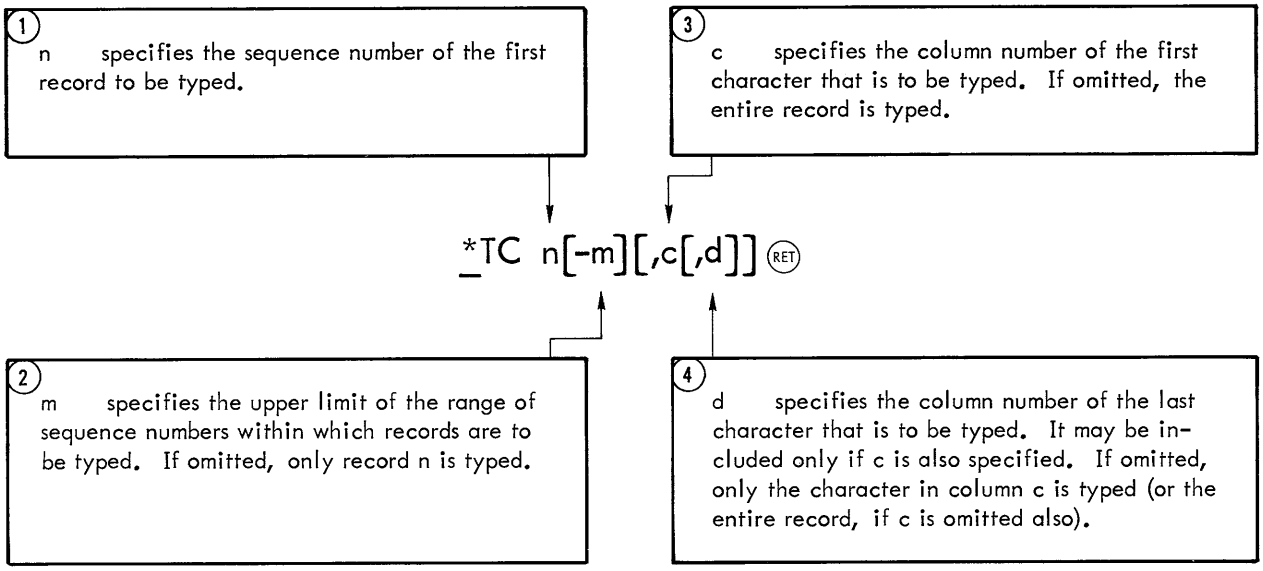
Example:

```
*EDIT SOURCEFILE RET
*TY 1-2,4,8 RET
1.000    EQU
1.200    SYST
1.400    REF
1.600    DEF
1.800    PAGE
2.000    ITIAL
*
```

In this example, the characters in columns 4 through 8 of all records in the sequence number range 1-2 of file SOURCEFILE are typed following the sequence number of each record.

TC (Type Compressed)

TC causes the Editor to type specified columns of one or more records in the currently open file. Any nonblank strings within the columns typed are shifted to the left to compress any blank strings to a single blank. This compression affects only the typed output; the records themselves are not affected.



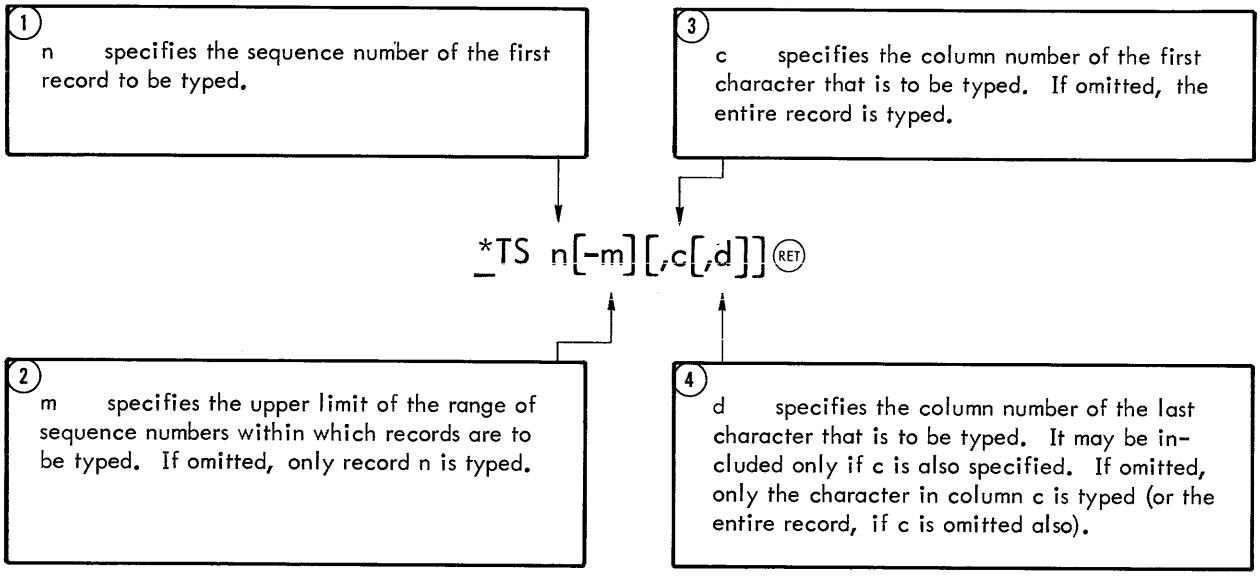
Example:

```
*EDIT SOURCEFILE RET
*TC 1-2,1,7 RET
1.000 A EQU
1.200  SYS
1.400 B REF
1.600 C DEF
1.800  PAG
2.000 *INITIA
*
```

In this example, character strings found in columns 1 through 7 of records 1 through 2 of file SOURCEFILE are typed, in compressed format, following the sequence number of each record.

TS (Type Without Sequence)

TS causes the Editor to type specified columns of one or more records in the currently open file. Its function is similar to TY, except that sequence numbers are not typed for each record.



There is also an intra-record form of this command (see page 5-39).

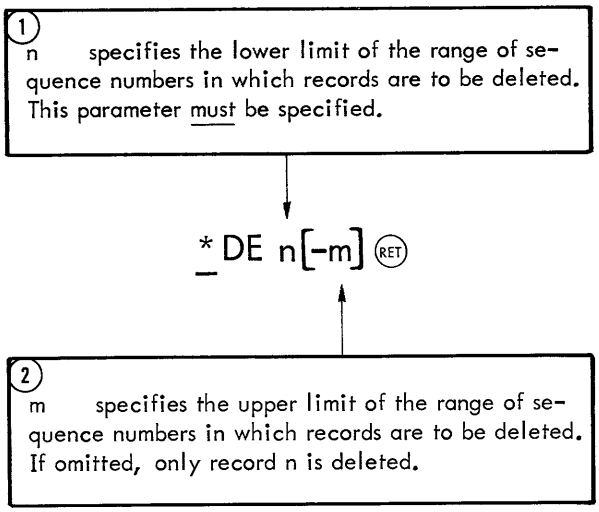
Example:

```
*EDIT SOURCEFILE RET
*TS 1-2,1,7 RET
A    EQU
     SYS
B    REF
C    DEF
     PAG
*INITIA
*
```

In this example, the characters in columns 1 through 7 of all records in the sequence number range 1-2 of file SOURCEFILE are typed without the sequence number of each record.

DE (Delete Records)

DE causes the Editor to delete, in the currently open file, all records whose sequence numbers lie in a specified range.



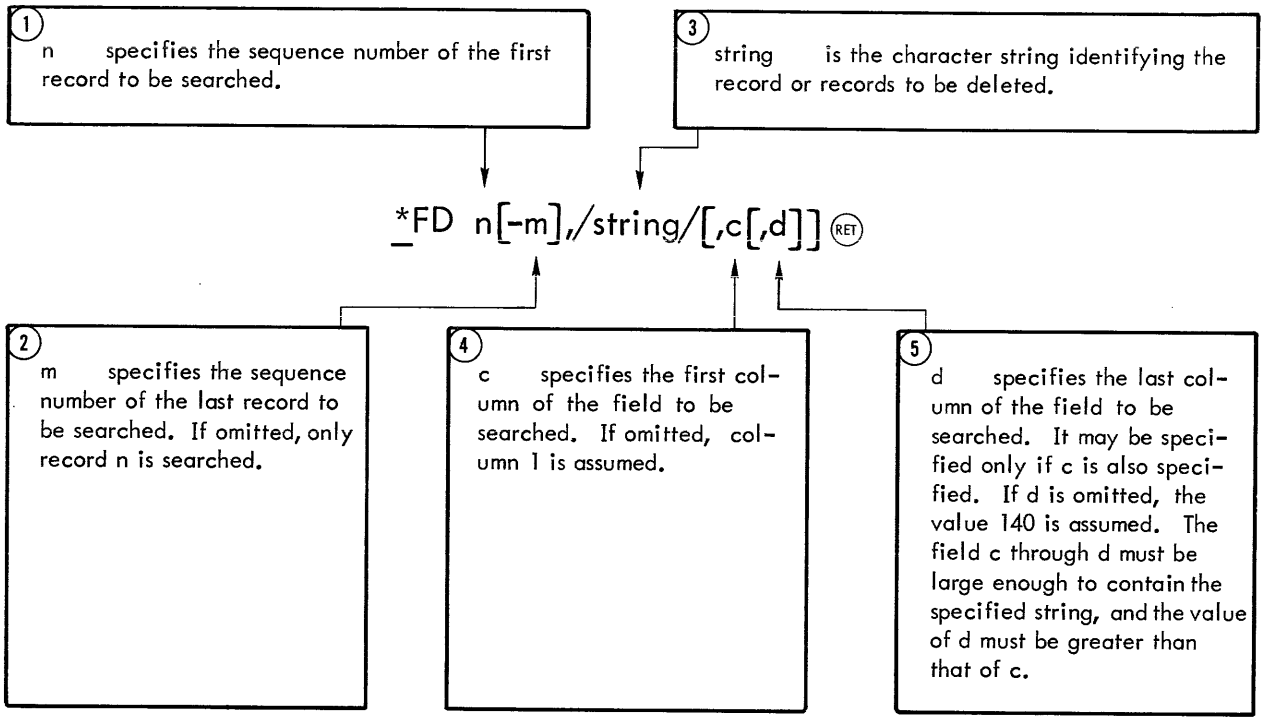
Example:

```
*EDIT SOMEFILE RET
*DE 25-30 RET
*
```

In this example, any records in the range 25.000 through 30.000 of the file SOMEFILE are deleted. The file remains open for further editing.

FD (Find and Delete)

FD causes the Editor to search a given range of records (in the currently open file) for a specified character string between designated columns. Any records within the affected range that contains the specified character string within the designated columns is deleted from the file.



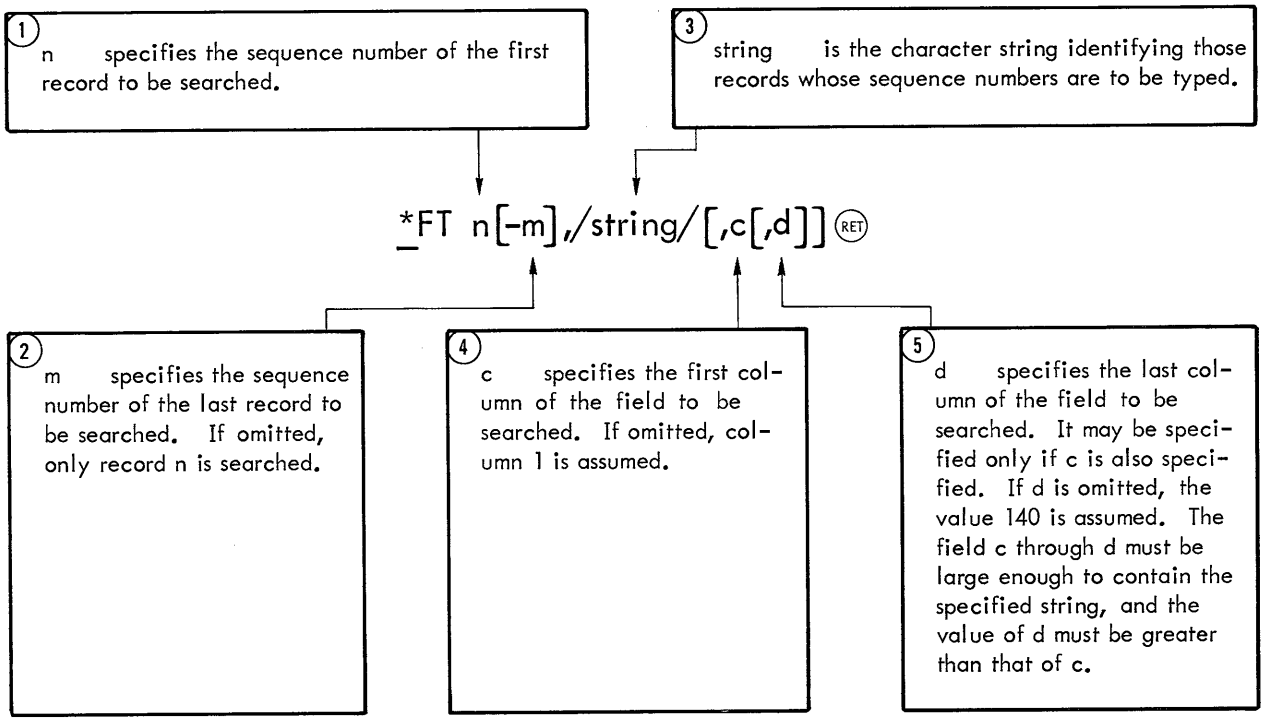
Example:

```
*EDIT SOMEFILE RET
*FD 10-20,/CW,R7/,10,14 RET
--002 RECS DLTED
*
```

In this example, the Editor deletes from file SOMEFILE two records located within the range of sequence numbers 10.000 through 20.000 and containing the string "CW,R7" in columns 10 through 14.

FT (Find and Type)

FT causes the Editor to search a given range of records (in the currently open file) for a specified character string between designated columns. The Editor will type the sequence number of each record satisfying the search criteria.



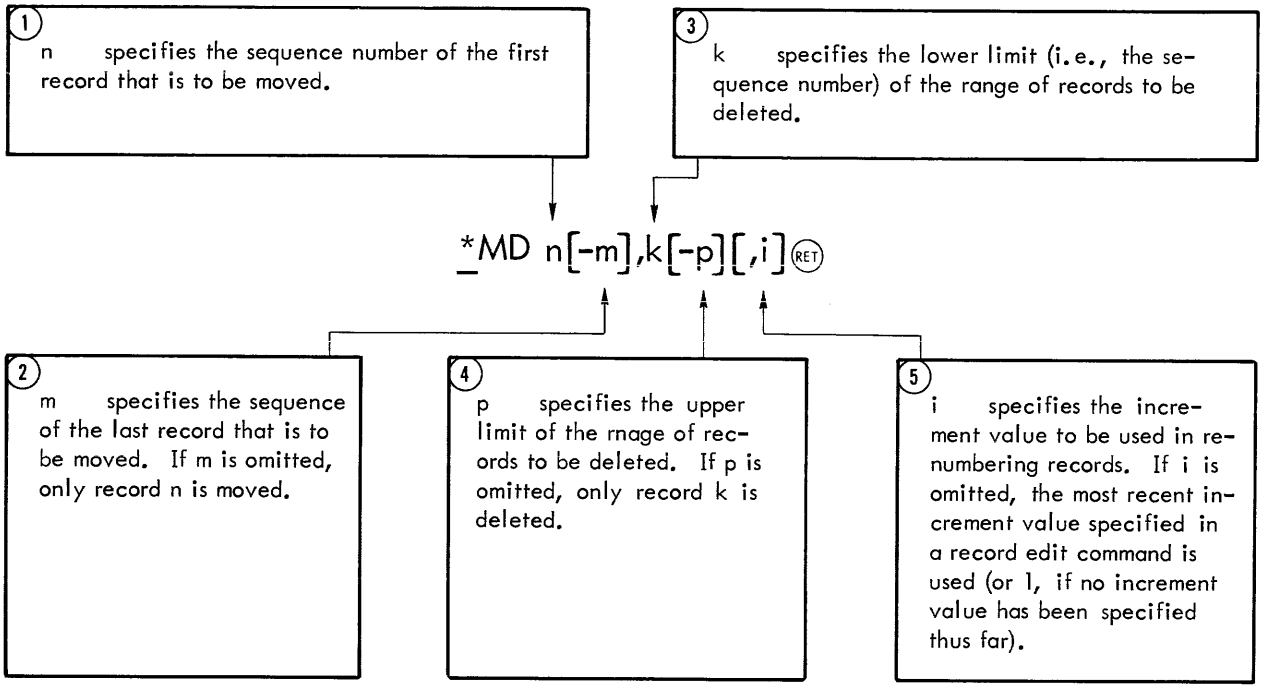
Example:

```
*EDIT SOMEFILE RET
*FT 10-20,/BE/,10,11 RET
15.000
*
```

In this example, the Editor types the sequence number 15.000, indicating that the string "BE" in columns 10 through 11 was found only in one record within the range 10.000 through 20.000 in file SOMEFILE.

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 two ranges must not overlap.



The first record (record n) is renumbered as record k. Successive records from the range n through m are renumbered consecutively higher, with increment i. As each record from the range n through m is moved, that record is deleted from the original area. At the end of this operation, a message is printed specifying the new sequence number of the last record moved from the range n through m.

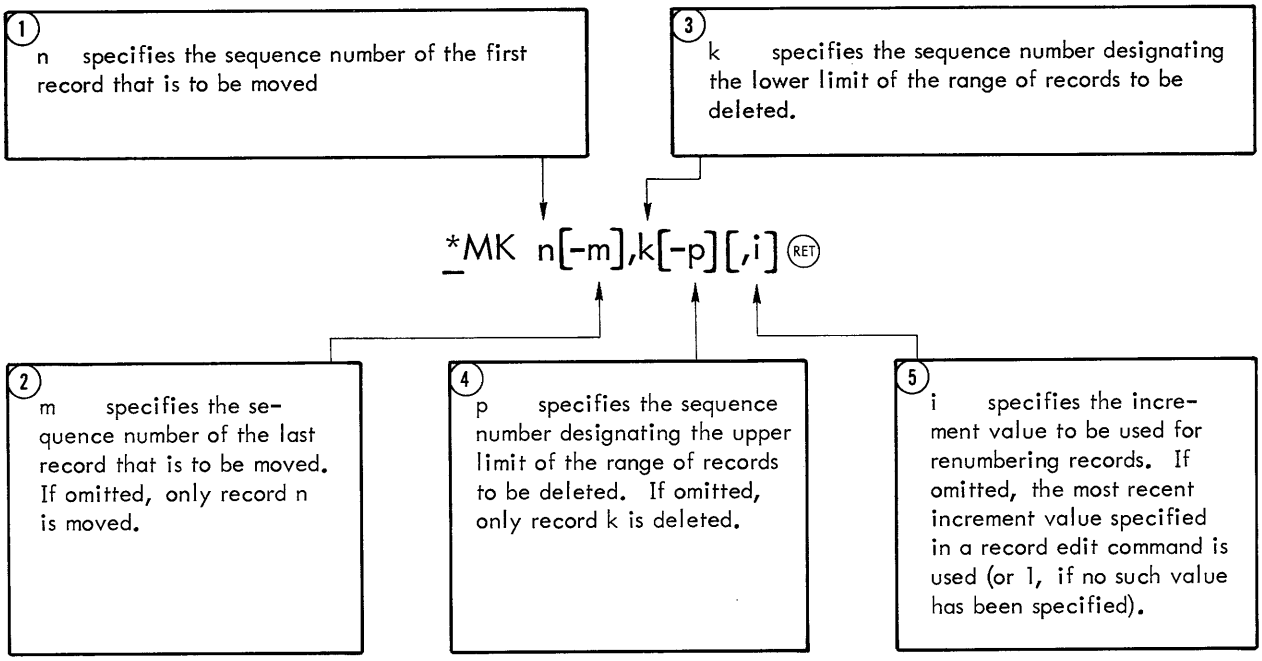
Example:

```
*MD 5-21,100-101,.02RET
--DONE AT 100.32
```

In this example, records 100 through 101 are deleted and records in the range 5 through 21 are moved to that area. The 17 records moved are renumbered 100.000 through 100.320, in increments of .020.

MK (Move and Keep)

MK causes the Editor to delete, in the currently open file, all records in a specified range and to then move records in another range into this area. Its action is similar to MD, except that records in the range n-m are not deleted as they are moved.



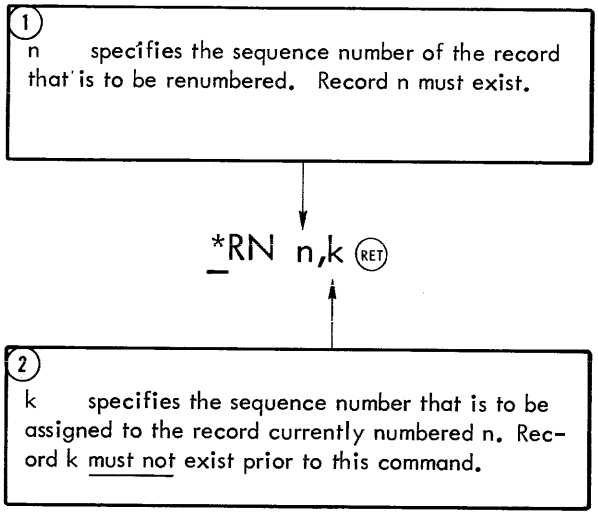
Example:

```
*EDIT AFILE RET
*MK 1,20 RET
--DONE AT 20.000
*
```

In this example, record 1.000 of file AFILE is moved to sequence number 20.000 but is not deleted from its original location.

RN (Renumber Record)

RN causes the Editor to renumber a specified record of the currently open file, deleting it from its old location.



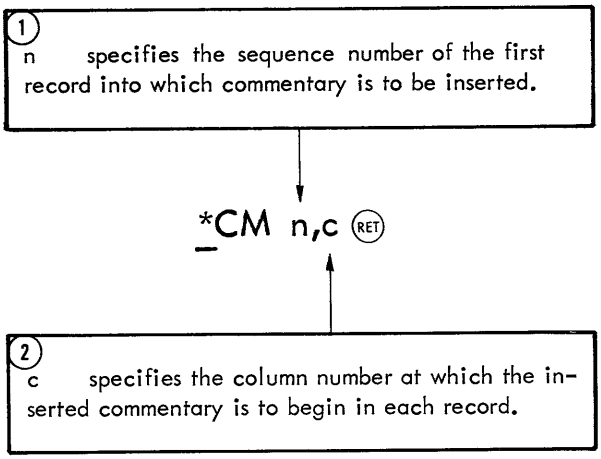
Example:

```
*EDIT THISFILE RET
*RN 10,20.1 RET
*
```

In this example, record 10.000 of file THISFILE is renumbered 20.100 and moved to the corresponding location in the file.

CM (Commentary)

CM causes the Editor to insert commentary into specified columns of each successive record of the currently open file, beginning at a specified sequence number.



The Editor prompts with the sequence numbers of successive records, beginning with record n. The user types the desired commentary following each sequence number. A null record (carriage return only) terminates the command.

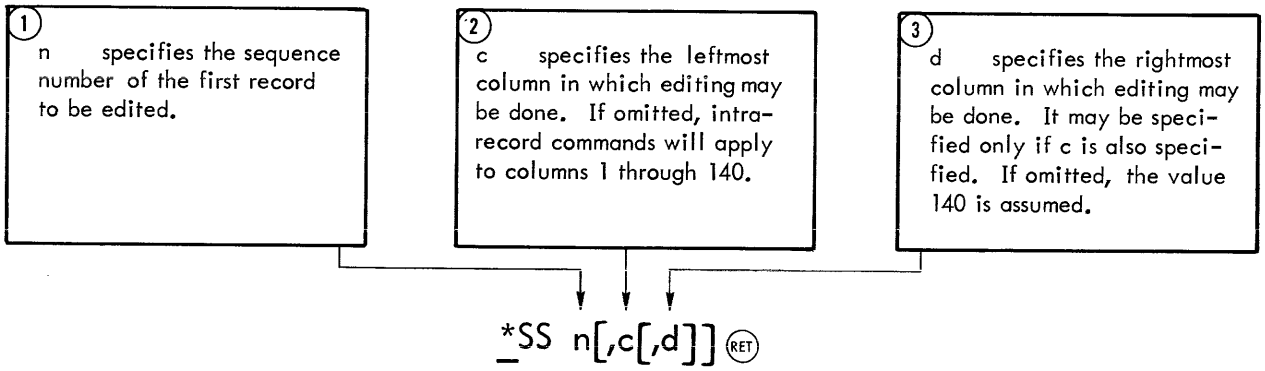
Example:

```
*EDIT AFILE RET
*CM 20,30 RET
20.000 RETURN ADDR RET
20.100 SUBR ENTRY RET
21.000 READ NXT RCD RET
22.000 RET
*
```

In this example, commentary is added to records 20.000, 20.100, and 21.000 of file AFILE. Commentary begins at column 30 in each record.

SS (Set and Step)

SS causes the Editor to start at a specified record in the currently open file and proceed to each record in succession, accepting one line of intra-record commands to update the current record.



Intra-record commands are applicable only to characters and strings of characters within the specified column limits of each record. Characters outside these limits may not be examined by intra-record commands.

The Editor prompts by typing the sequence number of each record in succession, followed by a double asterisk. The user may then type a line of intra-record commands to be applied to the current record. The SS command is terminated by typing a null record (carriage return only) following the double asterisk.

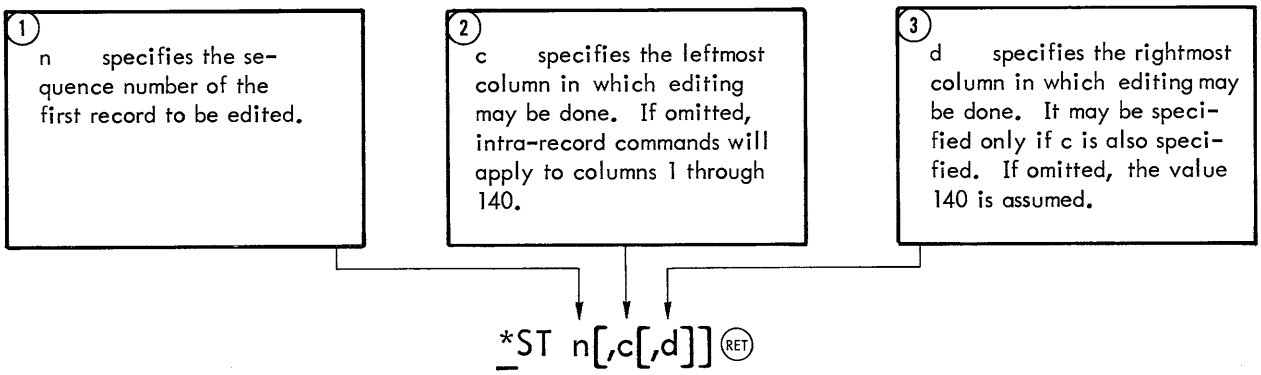
Example:

```
*EDIT THISFILE RET
*SS 1 RET
1.000** /BE/S/BEZ/ RET
2.000** /F:/S/M:/ RET
2.500** RET
*
```

In this example, records 1.000 and 2.000 of file THISFILE are edited by intra-record commands following use of the SS command. The SS command is then terminated by a null record following the typing of sequence number 2.500.

ST (Set, Step, and Type)

ST causes the Editor to start at a specified record in the currently open file and type each record in succession, accepting one line of intra-record commands to update the current record.



Intra-record commands are applicable only to characters and strings of characters within the specified column limits of each record. Characters outside these limits may not be examined by intra-record commands.

The Editor types each record in full, following the sequence number, and then prompts by typing a double asterisk on the following line. The user may then type a line of intra-record commands to be applied to the current record. The action of an ST command is terminated by typing a null record (carriage return only) following the double asterisk.

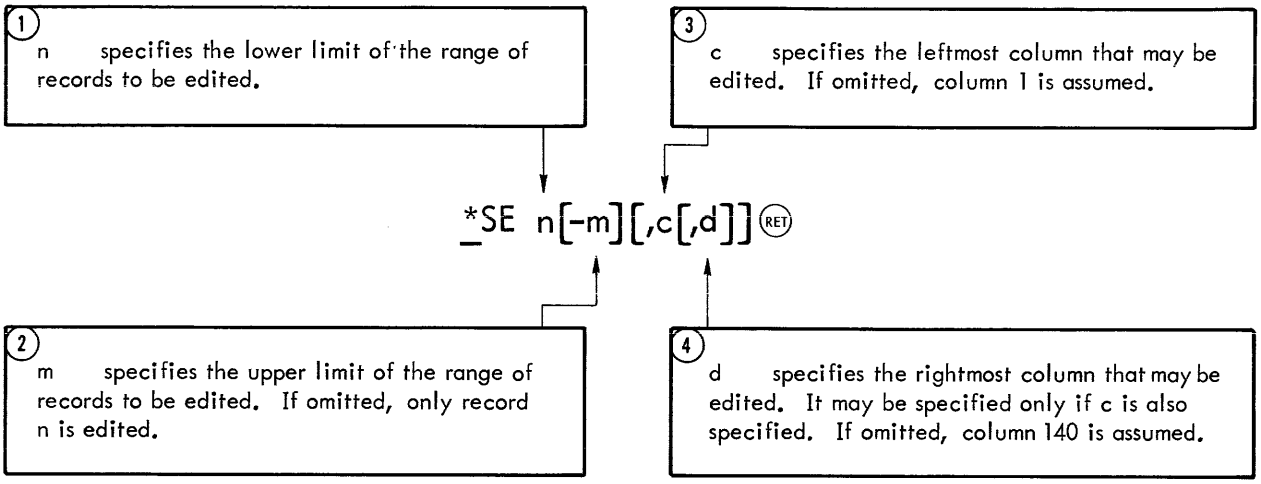
Example:

```
*EDIT SOMEFILE RET
*ST 6,8,16 RET
6.000 DEL22 BAL,R15 EXNEXT
** /R15/S/R10/ RET
7.000       B       DEL26
** NO RET
8.000 DEL14 BAL,R15 CBINT
** RET
*
```

In this example, record 6.000 of file SOMEFILE is edited by an intra-record command following use of the ST command. Records 7.000 and 8.000 are also displayed but not altered. The ST command is terminated by a null record following the display of record 8.000.

SE (Set Intra-Record Mode)

SE causes the Editor to accept successive lines of intra-record commands that are to be applied to a specified range of records in the currently open file.



Intra-record commands input following an SE command are applied, in order, to each record in the range n through m. If several commands are entered on one line, all of the commands on that line are executed and applied to the current record before the next record is edited. The first occurrence of a file-oriented or record-oriented command terminates the effect of the SE command.

Intra-record commands are applicable only to characters and strings of characters within the specified column limits of each record. Characters outside these limits may not be examined by intra-record commands.

If the SE command is used in the same line as other intra-record commands, it must be the first command in the line.

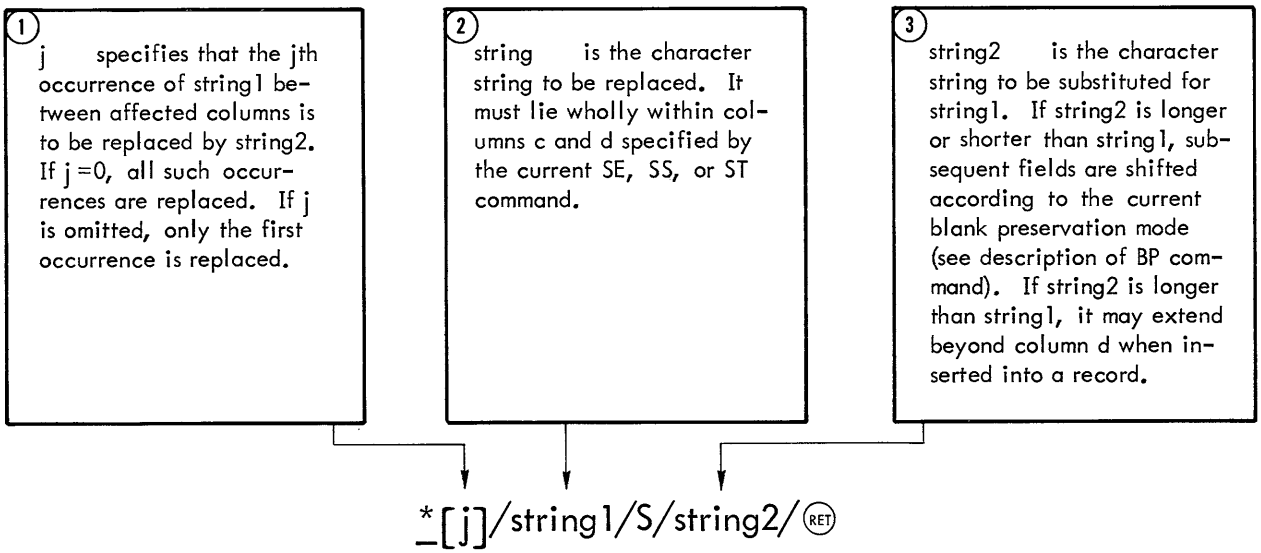
Example:

```
*EDIT MYFILE RET
*SE 1-100;/=X'00C4C5D3'/S/=X'D3'/RET
*EDIT ANOTHERFILE RET
*
```

In this example, an S (Substitute String) intra-record command is applied to all records of file MYFILE that lie within the range 1.000 through 100.000. The second EDIT command terminates the SE command. Note that intra-record commands appearing in the same line are separated by a semicolon.

S (Substitute String)

S causes the Editor to locate a given occurrence of a specified string and replace it with another specified string. The records and columns examined by the S command depend on the action of the current SE, SS, or ST command.



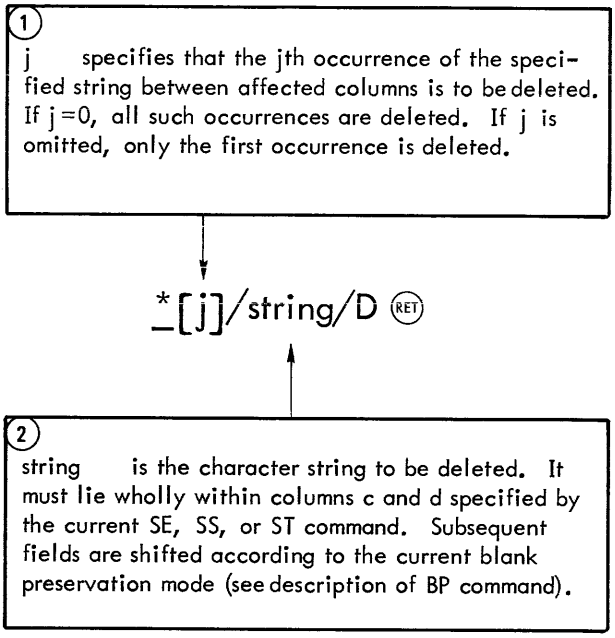
Example:

```
*EDIT AFILE RET
*SE 1-10;0/EQU/S/SET/RET
*
```

In this example, every occurrence of string "EQU" in records 1.000 through 10.000 of file AFILE is replaced by the string "SET".

D (Delete String)

D causes the Editor to locate a given occurrence of a specified string and delete it. The records and columns examined by the D command depend on the action of the surrent SE, SS, or ST command.



If a deletion would leave a gap in a nonblank field, the righthand portion of the field is shifted left to close the gap.

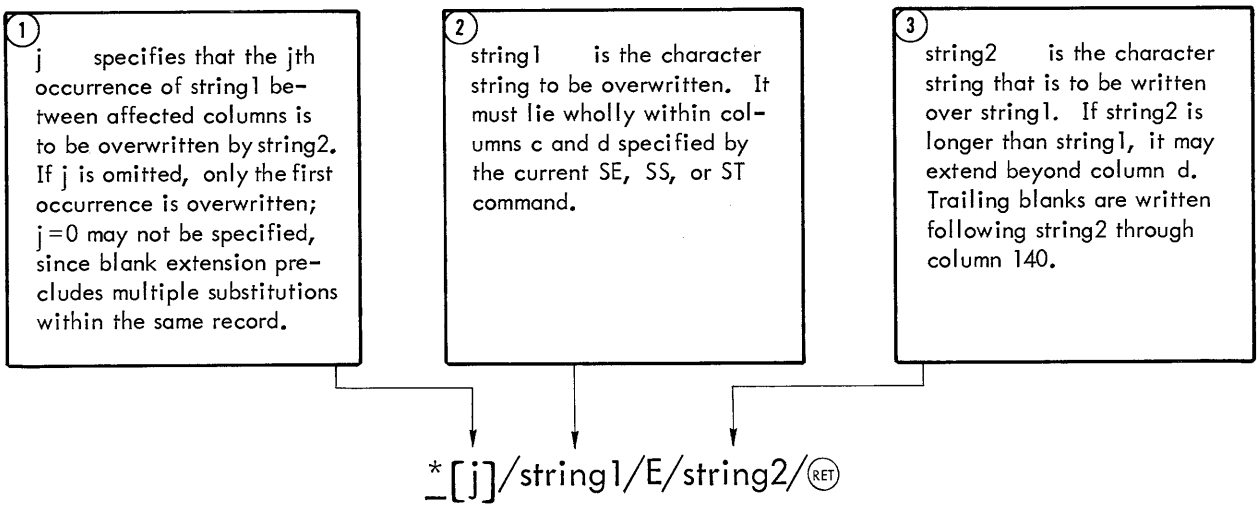
Example:

```
*EDIT ANYFILE RET
*SE 1-20, 10,15;/PAGE/D RET
*
```

In this example, the first occurrence of string "PAGE" within columns 10 through 15 in records 1.000 through 20.000 of file ANYFILE is deleted.

E (Overwrite String and Extend Blanks)

E causes the Editor to start at the column occupied by the first character of a given occurrence of a specified string and overwrite with another specified string followed by trailing blanks.



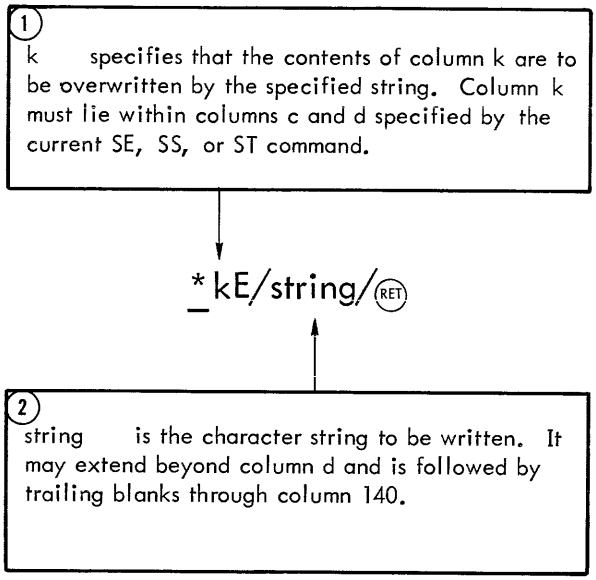
Example:

```
*EDIT THISFILE RET
*SE 1-99;1/ LOC1/E/ LOC2/RET
*
```

In this example, the first occurrence of the string " LOC1" in records 1.000 through 99.000 of file THISFILE is overwritten by string " LOC2" followed by trailing blanks.

kE (Overwrite Column and Extend Blanks)

kE causes the Editor to start at a given column and overwrite with a specified string followed by trailing blanks.



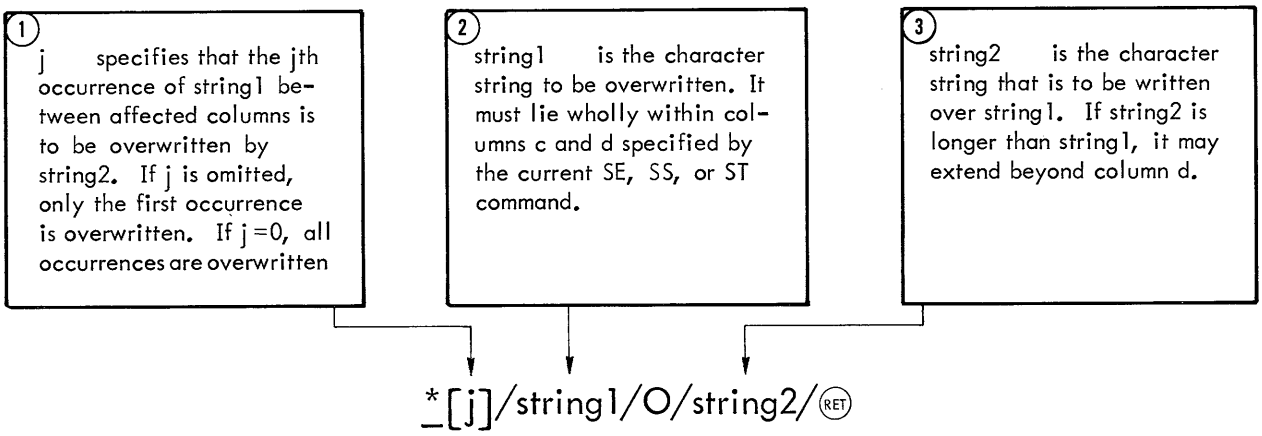
Example:

```
*EDIT SOMEFILE RET
*SE 18-20;15E/R6/ RET
*
```

In this example, the string "R6", followed by trailing blanks, is written over the contents of column 15 in records 18.000 through 20.000 of file SOMEFILE.

O (Overwrite String)

O causes the Editor to start at the column occupied by the first character of a given occurrence of a specified string and overwrite with another specified string.



In the case where j = 0, string2 is not scanned by the Editor after string1 is overwritten. The Editor begins scanning with the column following string2.

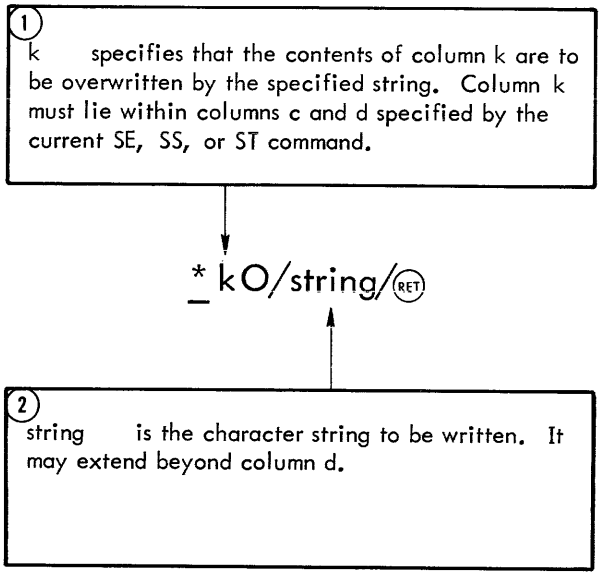
Example:

```
*EDIT SOMEFILE RET
*SE 1-10;O/ ADR3/O/ ADR4/ RET
*
```

In this example, every occurrence of the string " ADR3" in records 1.000 through 10.000 of file SOMEFILE is overwritten by string " ADR4".

kO (Overwrite Column)

kO causes the Editor to start at a given column and overwrite with a specified string.



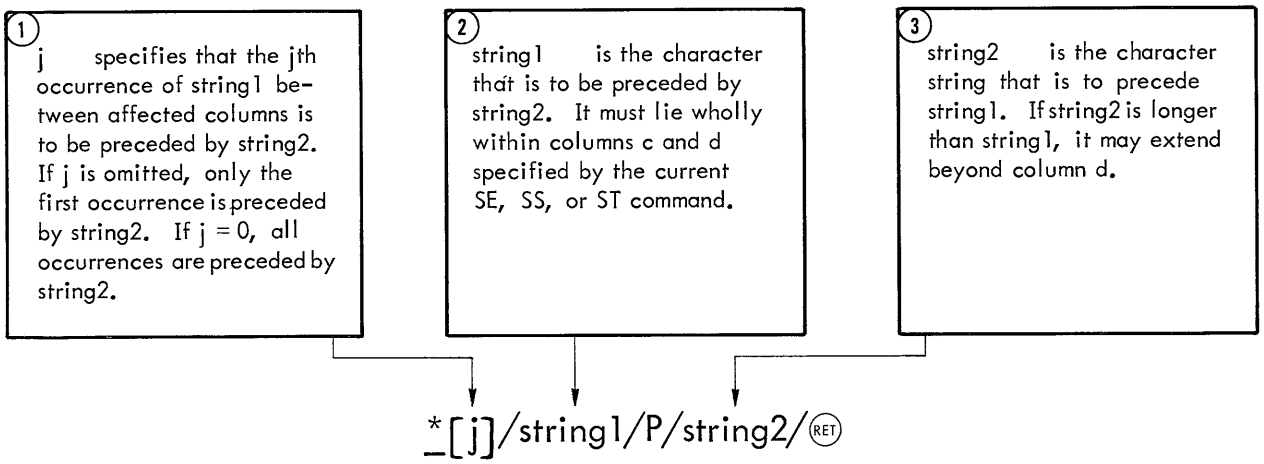
Example:

```
*EDIT THISFILE RET
*SE 15-20;110/R3/RET
*
```

In this example, the string "R3" is written over the contents of column 11 in records 15.000 through 20.000 of file THISFILE.

P (Precede String)

P causes the Editor to insert a specified string at the column occupied by the first character of a given occurrence of a specified string, shifting characters of the displaced string to the right as necessary.



In the case where j = 0, the Editor inserts string2 at all occurrences of string1 between columns c and d. Scanning for the next occurrence of string1 resumes following the last character of the previously shifted string1. If a given occurrence of string1 is shifted beyond column d, due to previous insertions, it will not be scanned.

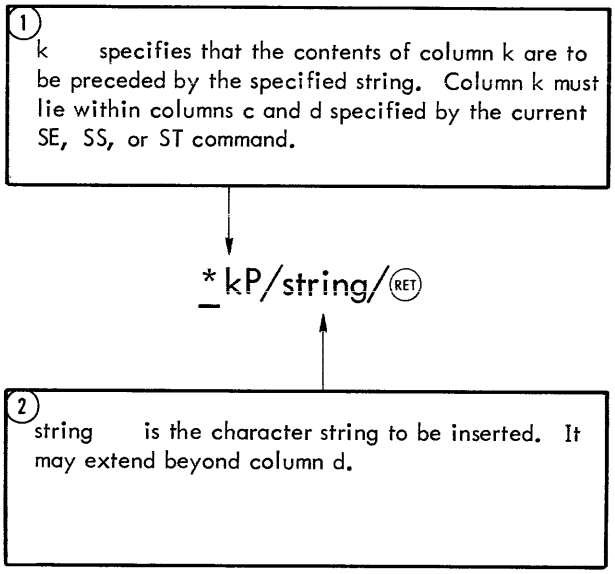
Example:

```
*EDIT THEFILE RET
*SE 25;TS;0/XY/P/,/;TS RET
XXYXYZWXYZ
X,XY,XYZW,XYZ
*
```

In this example, a comma is inserted prior to each occurrence of the string "XY" in record 25.000 of file THEFILE.

kP (Precede Column)

kP causes the Editor to insert a specified string at a given column, shifting displaced characters to the right as necessary.



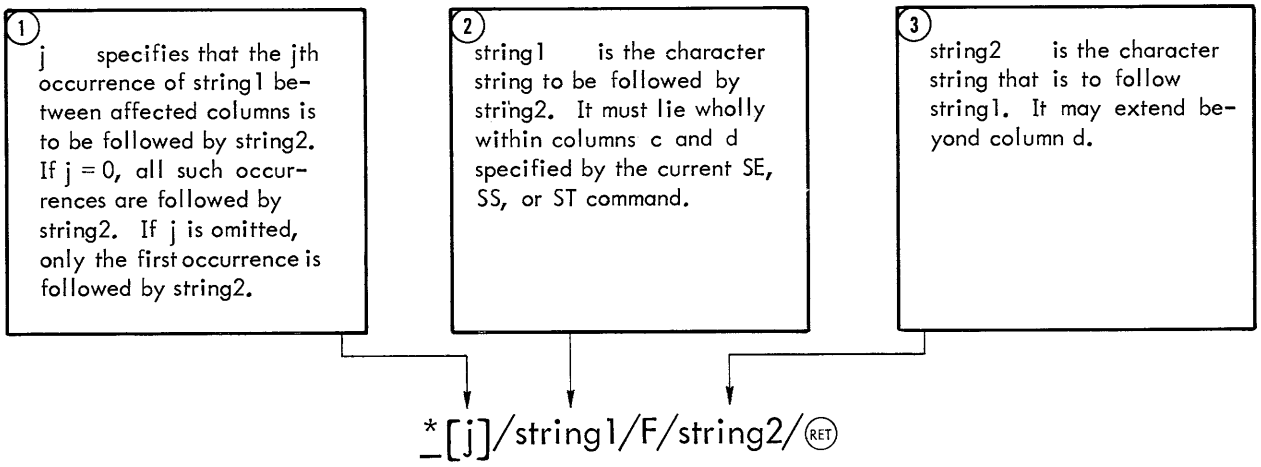
Example:

```
*EDIT SOMEFILE RET
*SE 6;TS;7P/G/;TS RET
ABCDEFHIJ
ABCDEFGHIJ
*
```

In this example, the string "G" is inserted prior to the contents of column 7 in record 6 of file SOMEFILE. Displaced characters are shifted one column to the right.

F (Follow String)

F causes the Editor to insert a specified string following the last character of a given occurrence of a specified string, shifting displaced characters to the right as necessary.



In the case where j = 0, the Editor inserts string2 at all occurrences of string1 between columns c and d. Scanning for the next occurrence of string1 resumes following the last character of string2. If a given occurrence of string1 is shifted beyond column d, due to previous insertions, it will not be scanned.

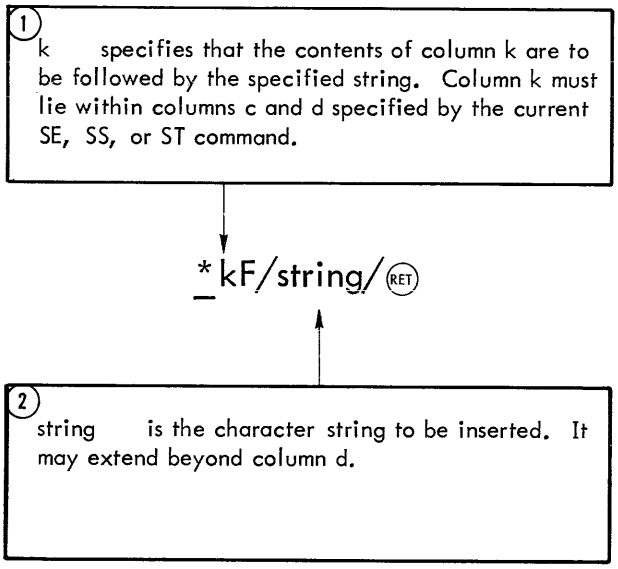
Example:

```
*EDIT THATFILE RET
*SE 1;TS;/LOC/F/+1/;TS RET
        LW,4 LOC LOAD MASK FROM LOC
        LW,4 LOC+1 LOAD MASK FROM LOC
*
```

In this example, the string "+1" is inserted following the first occurrence of string "LOC" in record 1.000 of file THATFILE.

kF (Follow Column)

kF causes the Editor to insert a specified string following a given column, shifting displaced characters to the right as necessary.



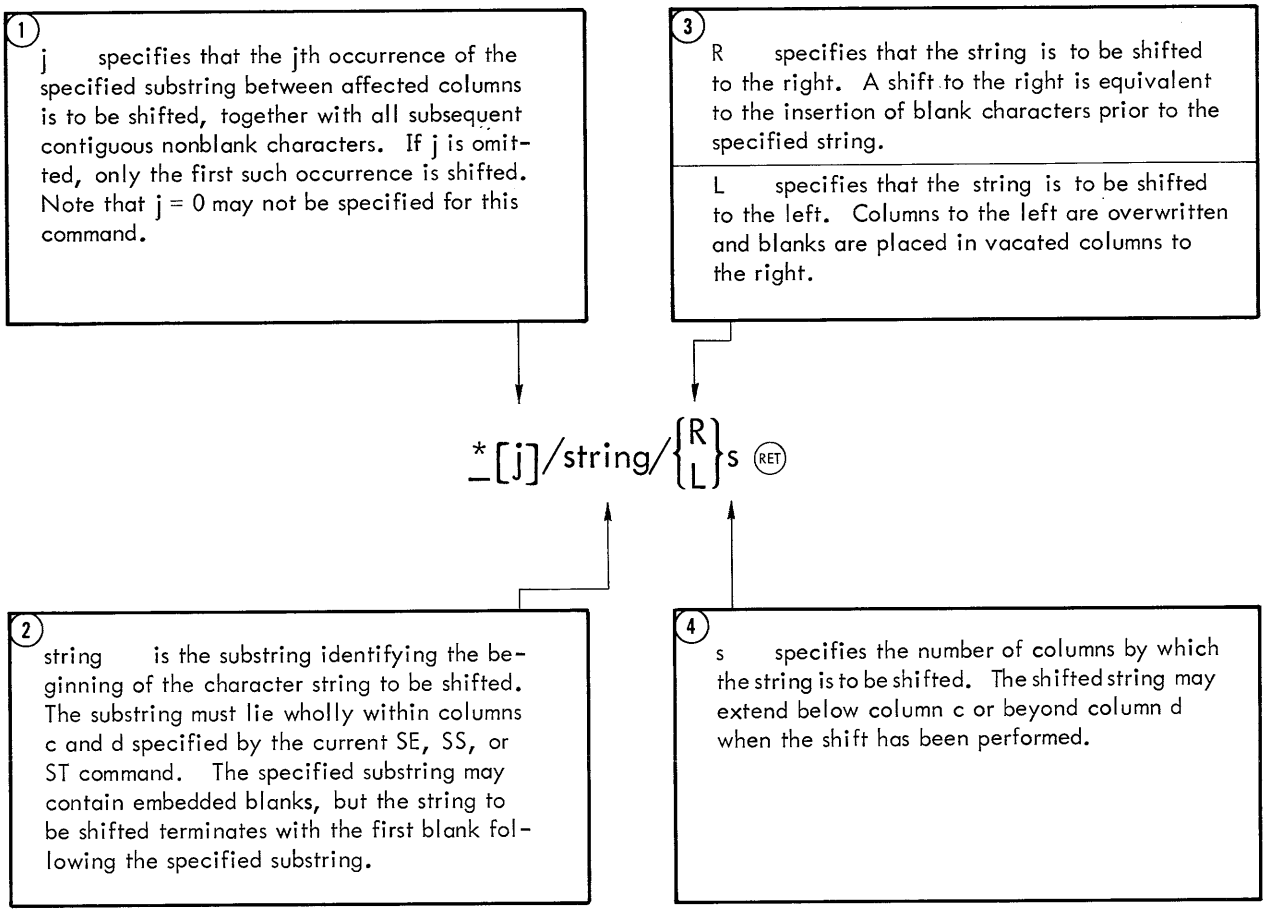
Example:

```
*EDIT OLDFILE RET
*SE 5;TS;10F/B/;TS RET
AAAAAAAAAAA
AAAAAAAAAABAA
*
```

In this example, the string "B" is inserted following column 10 in record 5 of file OLDFILE. Displaced characters are shifted one column to the right.

R or L (Shift at Substring)

R or L causes the Editor to shift, to the right or left, a contiguous string of nonblank characters beginning with a given occurrence of a specified substring.



Example

```
*EDIT THEFILE RET
*SE 100;TS;/IN/L2;TS RET
THE RAIN IN SPAIN
THE IN   IN SPAIN
*
```

In this example, the first occurrence of the string "IN" in record 100.000 of file THEFILE is shifted left by two columns.

kR or kL (Shift at Column)

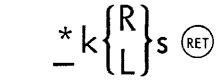
kR or kL causes the Editor to shift, to the right or left, a contiguous string of nonblank characters beginning with the character at a specified column.

(1) k specifies the beginning of the character string to be shifted. Column k must lie within columns c and d specified by the current SE, SS, or ST command. The string to be shifted terminates with the first blank following the specified column.

(2) R specifies that the string is to be shifted to the right. A shift to the right is equivalent to the insertion of blank characters prior to the specified string.

L specifies that the string is to be shifted to the left. Columns to the left are overwritten and blanks are placed in vacated columns to the right.

(3) s specifies the number of columns by which the string is to be shifted. The shifted string may extend below column c or beyond column d when the shift has been performed.



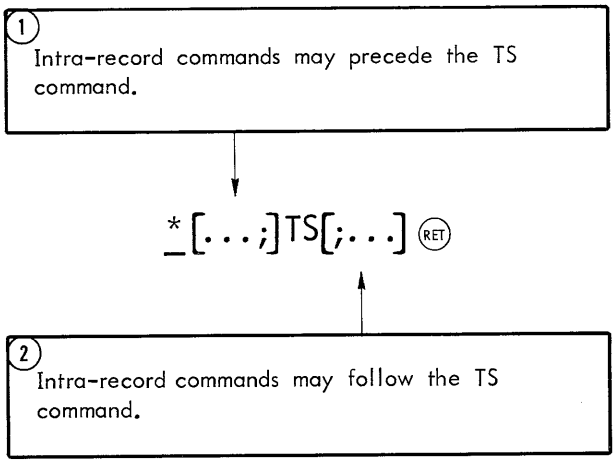
Example:

```
*EDIT THEFILE RET
*SE 101;TS;6R3;TS RET
LIES MAINLY IN    THE
LIES    MAINLY IN THE
*
```

In this example, the string "MAINLY" beginning at column 6 in record 101.000 of file THEFILE is shifted right by three columns. String "IN" is also displaced three columns to the right, to maintain a single blank between nonblank strings, but string "THE" is unaffected (blank preservation is assumed to be OFF; see description of BP command).

TS (Type Without Sequence)

TS causes the Editor to type the contents of the record currently open for editing under control of an SE, SS, or ST command. If more than one record is processed by the current SE, SS, or ST command, each such record is typed after all editing up to the TS command has been done. Sequence numbers are omitted.



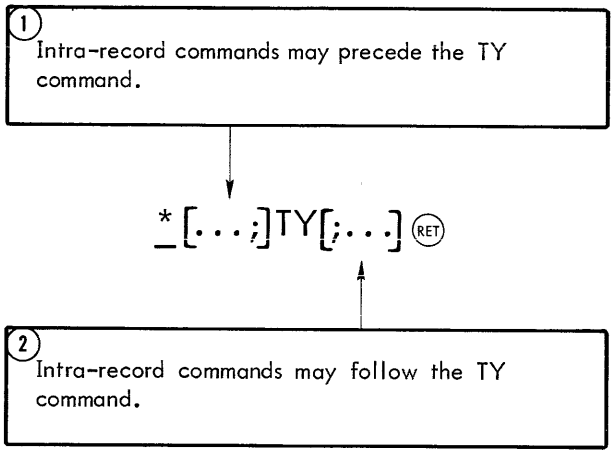
Example:

```
*EDIT AFILE RET
*SE11-13;TS RET
PRINXT BAL,R15 SPACE
       LI,R8   COMPRT
       BAL,R15 GEXT1
*
```

In this example, records 11.000 through 13.000 of file AFILE are typed without sequence numbers.

TY (Type Including Sequence)

TY causes the Editor to type the contents of the record currently open for editing under control of an SE, SS, or ST command. If more than one record is processed by the current SE, SS, or ST command, each such record is typed after all editing up to the TY command has been done. Each record typed is preceded by its sequence number.



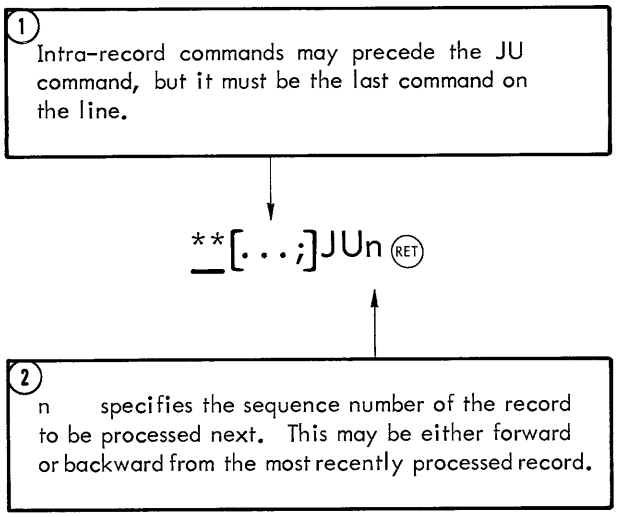
Example:

```
*EDIT FILE RET
*SE10;TY;/HERE/P/T/;TY RET
10.000 HERE  BAL,R1 EXIT
10.000 THERE BAL,R1 EXIT
*
```

In this example, record 10.000 of file FILE is typed before and after editing, preceded in each case by its sequence number.

JU (Jump)

JU causes the Editor to jump to a specified record while under the control of an SS or ST command, and to continue processing records from that point.



Example:

```
*EDIT SOMEFILE RET
*SS 1 RET
1.000**/BSD/S/BAD RET
2.000**37E/X/;JU9 RET
9.000**
```

In this example, the Editor processes records 1.000 and 2.000 of file SOMEFILE under control of an SS command. It then jumps to record 9.000 and continues under control of the SS command.

NO (No Change)

NO causes the Editor to perform no editing on the current record, while under control of an SS or ST command.

```
**NO (RET)
```

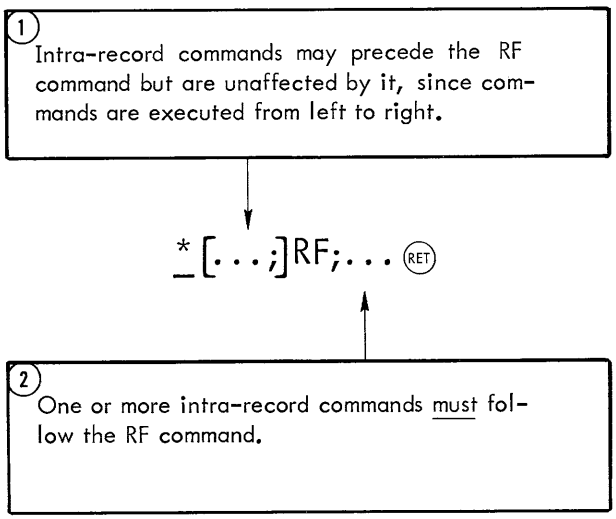
Example:

```
*EDIT THEFILE (RET)
*SS 1 (RET)
1.000**/$+3/S/LOC22/ (RET)
2.000**NO (RET)
3.000**
```

In this example, the Editor performs no editing on record 2.000 of file THEFILE under control of an SS command. Record 3.000 is then opened for editing.

RF (Reverse Blank Preservation)

RF causes the Editor to reverse the blank preservation state temporarily. Blank preservation is restored to its initial state (ON or OFF) when a new prompt character is typed. Thus, RF must be used as part of a compound command line. More than one RF command may be used in a single line.



Example:

```
*EDIT AFILE RET
*ST 10 RET
10.000 L5 LW,4  X
**RF;4R2;TY RET
10.000 L5   LW,4  X
11.000      STW,4 Y
**
```

In this example, record 10.000 of file AFILE is edited with blank preservation (initially OFF) reversed, so that two spaces are maintained before "X" when "LW,4" is shifted to the right.

EDIT MESSAGES

Messages	Meaning
..EDIT STOPPED	The record editing mode has been terminated.
--OVERFLOW	More than 140 characters have been typed on a line or characters have been shifted past column 1 or 140. Excess characters are lost.
-FILE EXISTS: CAN'T BUILD	An existing file has the same name as that specified in a BUILD command.
..COPYING	A COPY operation has begun.
..COPY DONE	A COPY operation has been completed.
-P2: FILE EXISTS	A COPY ON command specified the name of an existing file.
-P1: NO SUCH FILE	A COPY command has specified that a nonexistent file is to be copied.
-P1: FILE NOT KEYED & P3 NULL	A file to be copied has no sequence numbers and no sequencing has been specified. The COPY operation has been aborted.
-NO SUCH FILE	A specified file does not exist.
..DELETED	A specified file has been deleted.
-FILE NOT KEYED: MUST COPY	A specified file has no sequence numbers. The file must be copied with sequencing specified.
-NOT ON/OFF	A parameter other than ON or OFF has been specified in a BP or CR command.
--EOF HIT	One or both sequence numbers specified are higher than the highest one in the file.
--NOTHING TO DE	No records (to be deleted) were found in the specified range.
--xxx RECS DLTED	The indicated number of records have been deleted.
--NONE	No records have been deleted as the result of an FD operation.
--DONE AT x	A specified operation has been completed. The value x is the current sequence number of the last record affected.
--NOTHING TO MOVE	No records (to be moved) were found in the specified range.
--CUTOFF AT x(y)	A specified operation could not be completed because of a conflict between an existing sequence number and a new one. The value x is the current sequence number of the last record affected (formerly record y).
--RNG OVERLAP	Specified ranges of sequence numbers overlap. The command has been ignored.
-P1: NO SUCH REC	A specified record does not exist. The command has been aborted.
-P2: REC EXISTS	A specified record already exists. The command has been aborted.

Message	Meaning
-Cn: COMND ILGL HERE	The nth command of the current line is invalid and the intra-record mode has been terminated.
--Cn: OVERFLOW	The nth command of the current line has caused characters to be shifted past column 140. Processing continues.
--Cn: NO SUCH STRING	A specified string was not found and no substitution was made in processing the nth command of the current line. Processing continues.
-MISSING SE	No SE, SS, or ST command is currently in effect. The specified intra-record task has been aborted.
--Cn: 'ALL' IGNORED	The value 0 was specified for j. Since this value is not meaningful for the command, the value 1 has been assumed.

6. FORTRAN SUBSYSTEM

GENERAL

The FORTRAN subsystem enables the user to write, compile, execute, and save FORTRAN programs from an on-line terminal.

COMPILER INPUT/OUTPUT ASSIGNMENTS

During compilation there are various inputs to the compiler, and various outputs will be produced. The user may set up his own I/O assignments by using the Executive command ASSIGN prior to calling the FORTRAN subsystem (refer to Chapter 3 of this manual.) Default assignments will be made automatically for any input/output assignments not specified. Table 6-1 lists the compiler input/output DCBs.

Table 6-1. Compiler Input/Output DCB Assignments

DCB	Description
M:BO	Binary output of assembled object program. By default this goes into temporary file BOTEMPx, where x is the special ID (see Chapter 3) for the user's terminal; or, the operator may specify a file of his own. This is the file specified to the Loader subsystem when it is desired to run the program.
M:DO	Diagnostic output. Images of source lines in which errors are found will be printed, along with error messages. Default assignment is the user's terminal. Also, diagnostic output is always included if a program listing is produced; hence if M:LO is assigned to a different file than M:DO, the diagnostic will be output to both files.
M:LO	Listing output. Default assignment is the user's terminal.
M:SI	Source language input. Default assignment is the user's terminal. A previously created file may also be specified through use of the EDIT subsystem or through M:SO in a previous compilation.
M:SO	Source Output. This is a means of saving source language that has been typed and corrected. Default assignment is to SOTEMP, a temporary file. Note: Refer to Tables 3-4 and 3-5.

COMPILER OPTIONS

The subsystem is called by typing FO following the Executive prompt character. The Executive types the rest of the word and turns control over to the subsystem which requests a list of options.

```
!FORTRAN
OPTIONS:
```

Any of the options listed in Table 6-2 may be selected, separating them with commas. If a mistake is made, hit (ESC) X and start the list again. Terminate the list and each line with (RET). To specify no option list, simply hit (RET).

Table 6-2. Compiler Options

Option	Description
BO or NOBO	Binary Output file. Operator could specify NOBO when he does not expect to be able to execute but would like diagnostics.
LS or NOLS	List Source language. Operator might specify NOLS when recompiling a program which has previously been debugged, if another program listing is not desired.

Table 6-2. Compiler Options (cont.)

Option	Description
SO or NOSO	Source Output. This saves a copy of the source program in a file on the RAD. It is a means of saving a program typed in at the terminal. A request for this option is ignored if the input is from a disc file.
DB or NODB	Debugging Mode. If this option is specified, then it is possible to stop, start and trace execution, and obtain values of program variables during execution of the program.
S or NOS	Symbolic machine language instructions may be written within FORTRAN source programs.
Note: The operator may request to have or not have any of the above options. The underlined alternatives will be assumed when there is no specification.	

SOURCE LANGUAGE INPUT

During compilation, each FORTRAN statement is input and analyzed. Source language may be input from a file on the RAD or it may be typed line by line at the user's terminal. For terminal input, the subsystem will prompt the user with the line number of each successive line.

When entering FORTRAN statements to the subsystem, the user may type a colon (:) to indicate the end of each complete statement and cause the line to be processed immediately. When an error occurs in a line typed at the terminal, the operator is given the opportunity to retype the line. Table 6-3 is an example of a FORTRAN compilation.

Table 6-3. Source Language Input From Terminal

!FORTRAN	Select FORTRAN.
OPTIONS:(RET)	Default options.
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.

FORTRAN DEBUGGING MODE

The FORTRAN debugging capability constitutes a self-contained portion of the compiled FORTRAN object module.

When the program has been loaded and is ready for execution, the program identification is typed by the Monitor, then an asterisk on the next line. The operator may then type in one or several of the debugging commands appearing in Table 6-4 (see also Figure 6-1).

Table 6-4. FORTRAN Debugging Commands

Command	Function
S	Execute the program in step mode.
NS	Cancel step mode.
T	Trace source statement lines reached during program.
NT	Stop tracing execution.
V	Dump variables.
NV	Stop dumping.
Adddd	Stop at dddd.
NA	Cancel stop. (See above).
G	Go.
R	Do not stop at subroutine entries.
NR	Stop at all subroutine entries.
Hxxxxxx	Halt only upon subroutine entry names "xxxxxx".
NH	Halt at all subroutine entries.

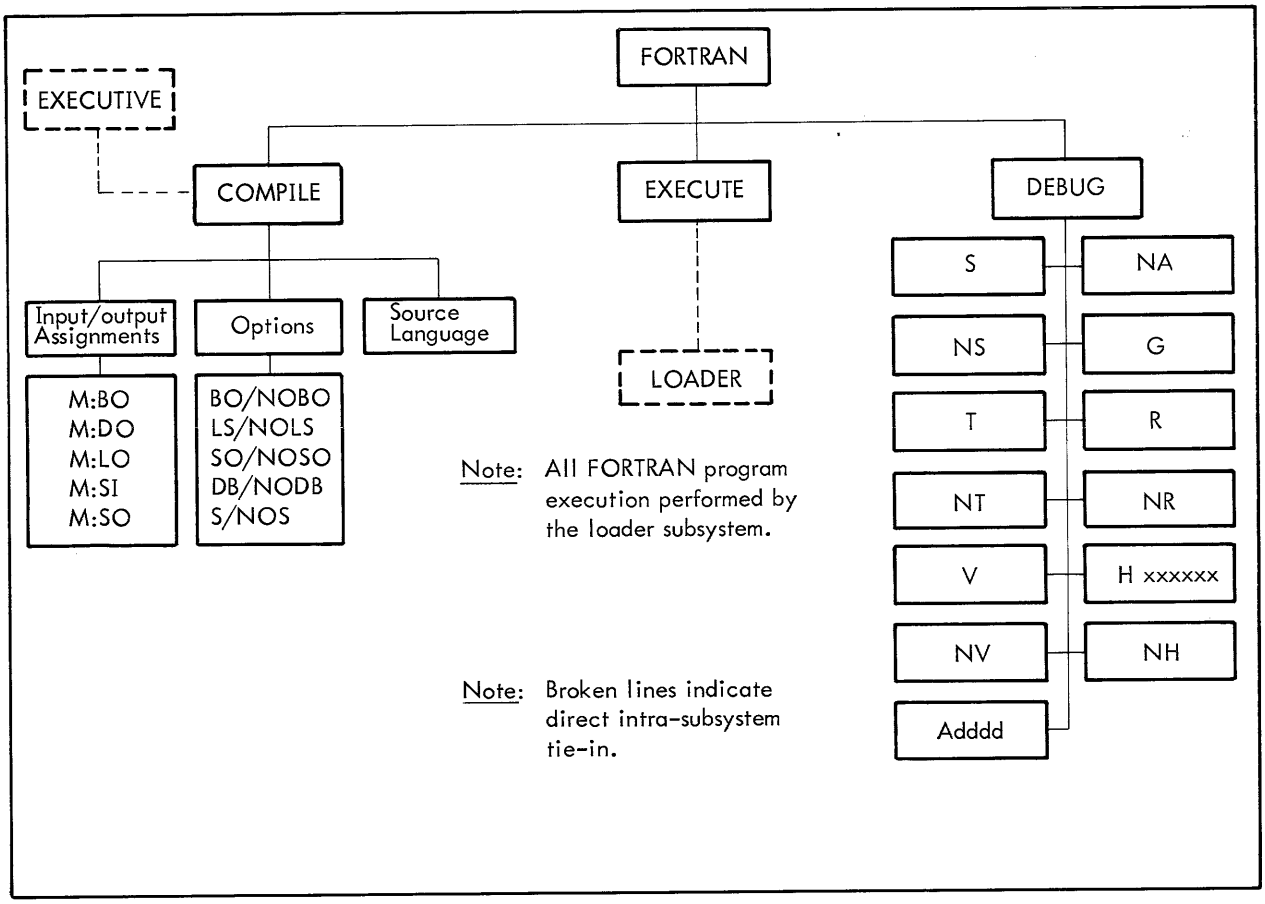


Figure 6-1. FORTRAN Subsystem

7. LOADER SUBSYSTEM

GENERAL

The Loader subsystem will cause a program to be loaded and executed at the user's on-line request. The program will consist of one or more relocatable modules of object language coding which have been assembled by BTM Symbol, BTM FORTRAN IV-H, Standard Symbol (off-line), Standard Meta-Symbol (off-line), XDS FORTRAN IV (off-line), or FORTRAN IV-H (off-line).

OPERATION

When the Loader subsystem is called, the operator is requested to specify various information. During operation, the Loader subsystem will return some information and diagnostic messages. Table 7-1 lists the various steps in the use of the Loader.

Table 7-1. Loader Subsystem Operating Steps

Step	Procedure
1	Specify the element file or files that contain the program to be loaded.
2	Specify the various options to be in effect during program loading and execution.
3	Subsystem will find and load all necessary object modules.
4	Subsystem will, if requested in step 2, type a load map containing all references by which various loaded modules are related to one another.
5	Assign or change any assignments of various DCBs for program input-output.
6	Subsystem types out severity level of errors encountered during loading. Decide whether or not to proceed with execution of the program.
7	If execution is to proceed, specify values for unsatisfied external and internal references (if the D option has been specified).
8	Program is executed.

ELEMENT FILES

Input of object modules is through M:BI. Specify what program is to be loaded by designating the element files to be input. If an element file is not specified, it will be assumed that the input is from the temporary file BOTEMPx which is also the default output file for on-line assemblies.

If it is desired to specify a single element file, ASSIGN M:BI before calling the Loader subsystem. After calling the subsystem, it types a request after which the user may type a list of element files.

Examples:

!LOAD

ELEMENT FILES: (RET)

The Loader subsystem was called. As soon as it took control, it typed a request for the list of element files. In the example, the user hit the carriage return, indicating no list; therefore, loading will be from the temporary file BOTEMPx.

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

Prior to calling the Loader subsystem, the user file MINE was assigned to input DCB M:BI.

```
!LOAD
ELEMENT FILES: MINE,OURS,HIS(ACCT4,PSST)↵
```

The modules to be loaded are in the files MINE and OURS in the log-in account, and HIS in the account ACCT4 with password PSST. These modules will all be loaded, and if any of them contain primary external references to additional modules, a library search will be made. All referenced modules will be loaded as they are found.

LOADER OPTIONS

The next step is the request by the Loader for specification of desired options, these options are listed in Table 7-2.

```
!LOAD
ELEMENT FILES:↵
OPTIONS: P,M,D↵
```

In the above example, three options have been specified: program modules will be loaded at hexadecimal addresses with two low-order zeros, a load map will be produced, and the debugging feature will be employed.

Table 7-2. Loader Options

Option	Description
N	Do not search the system library to satisfy external references. This library is the file :BLIB (binary object language library) in the account :BTM, and contains various standard program modules for general use.
M	Produce a complete load map containing all external definitions and references. If this option is not specified, only a list of unsatisfied references will be typed by the Loader.
U (acct 1) (acct 2)	This specifies those accounts other than the user's own that may be searched to satisfy external references. After the user's own binary object language library is searched, the file :BLIB will be searched in each of the specified accounts. If the reference is still unsatisfied, the system library will then be searched (unless the N option has been specified).
P	Specify this option if all program modules are to be loaded starting at hexadecimal hundreds' addresses (i.e., at addresses with two low-order zeros). This is useful during debugging, as it facilitates locating instructions from their addresses as they appear in an assembly listing.
D	With the D option (the debugging feature), the program will be executed under control of the Delta debug program (see BTM Reference Manual). This will enable performance of diagnostic and corrective operations on the program.
L	With this option, the load module will be placed in the user's program library.

ERROR MESSAGES

The next step is that of finding and loading all required program modules. Messages are typed by the Loader for any errors that occur. These messages consist of two lines of information; the first specifies the error, the second locates the error. Loader error messages with their corresponding meaning appear in Table 7-3.

Table 7-3. Loader Messages

Message	Meaning
NO LIB FILE	The specified library could not be found.
NO ELEMENT FILE	The specified element file could not be found.
ILLEGAL ORIGIN	An attempt has been made to load outside the available area in the computer memory.
ILLEGAL ROM DATA	The Relocatable Object Module being loaded contains illegal object language.
CHECKSUM ERROR	There was a checksum error in the record specified.
SEQUENCE ERROR	A discrepancy was found in the numbering of successive records within an object module, or the first record of an object module is missing.
STACK OVERFLOW	There is not enough room in memory to fit the Loader, the program, and associated areas.

Note: When each of the above messages is typed, the next line will contain one of the following (xx and yy are hexadecimal numbers):

1. LOADING ELEMENT FILE (name) SEQ. NO. (xx) OVERALL ROM NO. (yy)
2. PROCESSING LIBRARY (account) SEQ. NO. (xx) OVERALL ROM NO. (yy)
3. LOADING FROM BI SEQ. NO (xx) OVERALL ROM NO. (yy)

LOAD MAP

If the M option has been specified, a load map is produced at the completion of loading. The load map lists external definitions, giving the address at which each externally defined program location has been loaded into computer memory. The locations of the beginnings of all loaded modules are shown, as well as the lowest and highest program locations, and all DCBs. In addition, the load map will list unsatisfied external references and unused or double definitions. The abbreviations used in the load map are listed in Table 7-4.

Table 7-4. Load Map Abbreviations

Abbreviation	Meaning
SREF	A secondary reference. Such a reference cannot be satisfied in on-line operations. A secondary reference causes an object module to be found, but delays loading of the module until a later overlaying operation. Note: No overlaying can be done when operating on-line.
PREF	An unsatisfied primary reference (REF). The indicated name has not been found in any of the libraries searched.

Table 7-4. Load Map Abbreviations (cont.)

Abbreviation	Meaning
DEF	An external definition which was referenced and has been found, with the corresponding object coding loaded. The present computer memory address is expressed in hexadecimal as xxxx y (xxxx is nearest word, y is byte in word).
UDEF	Unused definition. This is an external definition included within that loaded object code to which no reference has been made.
DDEF	Double definition. More than one definition has been found for this name. The first one found has been used.
Note: When used with the D option, the Loader will provide an additional list of undefined internal symbols for programs assembled on-line by Symbol, or off-line by Meta-Symbol with the SD option. The undefined internal symbols are grouped according to the element files in which they occur.	

DCB SPECIFICATIONS

Following the load map and undefined internals, the subsystem requests any additional DCB specifications which may be needed for program input/output operations. The subsystem prompts with (F:), and the operator may type one specification per line. This may be used where individual specifications are needed to cause the Loader to build a DCB.

ERROR SEVERITY

In the next step, the Loader types the highest error severity level which occurred during loading, then asks whether or not it is to proceed.

REFERENCE SATISFACTION

If execution is requested, and unsatisfied external references exist, the operator will be requested to supply values for any references needed (if the D option was specified). The subsystem will type:

```
**SATISFY EXTERNALS**
<
```

When running under option D, a list of any undefined internal references will be typed, which the operator will be asked to satisfy.

```
**SATISFY INTERNALS**
```

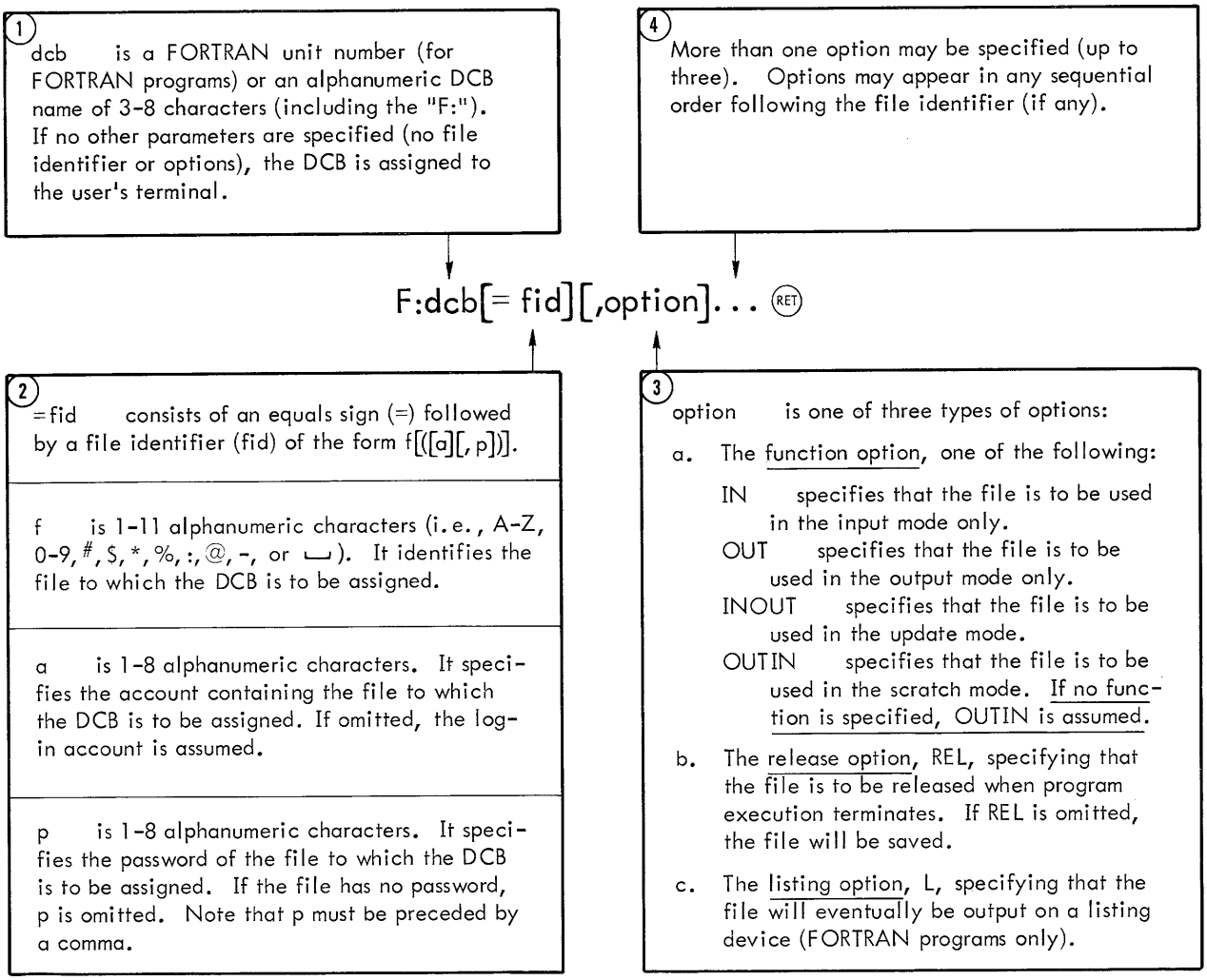
The name of the element file will be typed on the next line, and on the line following, the prompt character (<).

DEBUGGING

If option D has been specified, the program will be loaded in a special manner and executed under control of the debugging program Delta (refer to BTM Reference Manual). Execution may be broken at various points to search, inspect, and alter contents of computer memory.

F: (DCB Specification)

F: prompts the user to specify any required DCBs that are not designated as external REFs in the user's program or DCBs for which one or more options are to be changed. Any FORTRAN DCBs for unit numbers other than 101-106 or 108 fall into this category and must be specified in this way.



The options specified will be inserted into the DCB and, thus, will act as default parameters that may be overridden by ASSIGN commands or procedure calls. The Loader will continue to prompt DCB specifications by printing "F:" at the beginning of each new line until the user inputs a null specification (carriage return only).

Example:

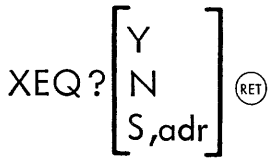
```
!LOAD
ELEMENT FILES: MINE RET
OPTIONS: L RET
F:5 RET
F:6=MYFILE RET
F:7=HISFILE(BTM6),IN RET
F: RET
SEV.LEV.=0
XEQ?N RET
!
```

In this example, the user specifies that F:5 is to be assigned to the user's terminal, F:6 to scratch file MYFILE, and F:7 to input file HISFILE in account BTM6. A null specification then ends the list of assignments.

XEQ? (Execution Request)

XEQ? prompts the user to specify whether or not the load module just formed is to be executed and, if so, allows a start address to be specified.

(1) Y specifies that the load module is to be executed. The absence of an option (carriage return only) will cause the load module to be executed as though a Y had been typed.
N specifies that the load module is not to be executed. The Loader responds by returning control to the BTM Executive.
S,adr specifies that the load module is to be executed, beginning at location adr (either an external definition [±hex. addend] or else a signed positive absolute hexadecimal address). Hexadecimal numbers are identified as such by means of a leading period (the hexadecimal equivalent of decimal 26 would be +.1A).



Example:

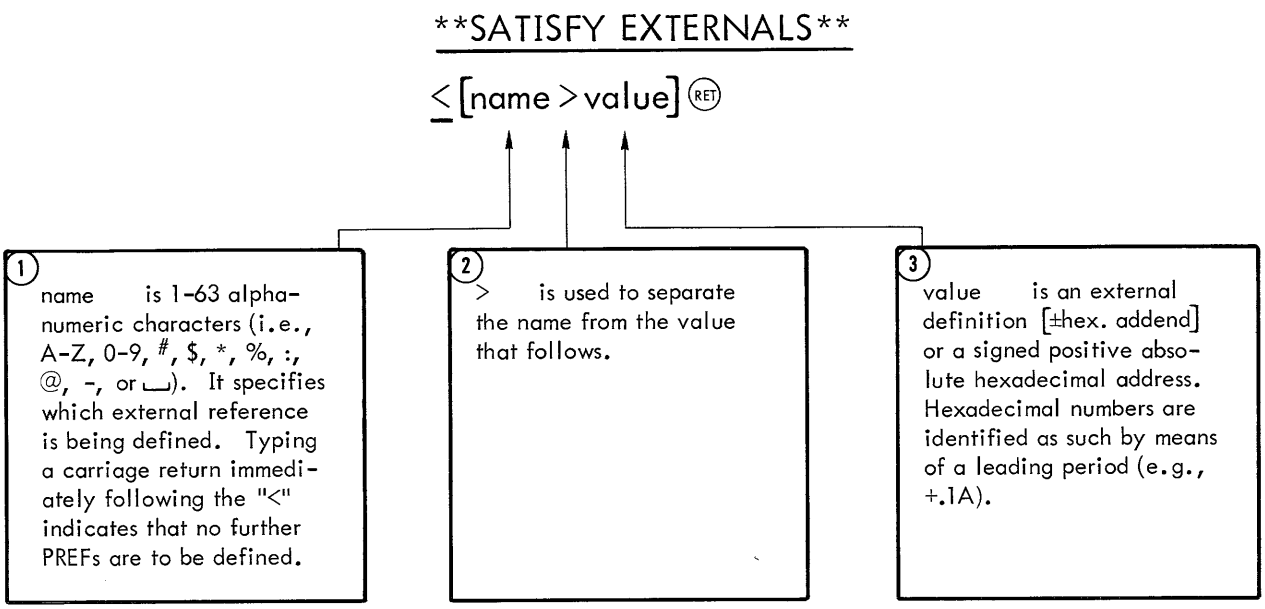
```
SEV.LEV=0
XEQ? S,START+.A3 RET
```

In this example, program execution is to begin at a location 163 words higher than external definition START.

SATISFY EXTERNALS
(Satisfy External References)

SATISFY EXTERNALS, followed by a "<" character at the beginning of the next line, prompts the user to satisfy undefined external references (any symbol listed as a PREF in the load map).

This feature is provided only if the 'D' option has been specified.



Example:

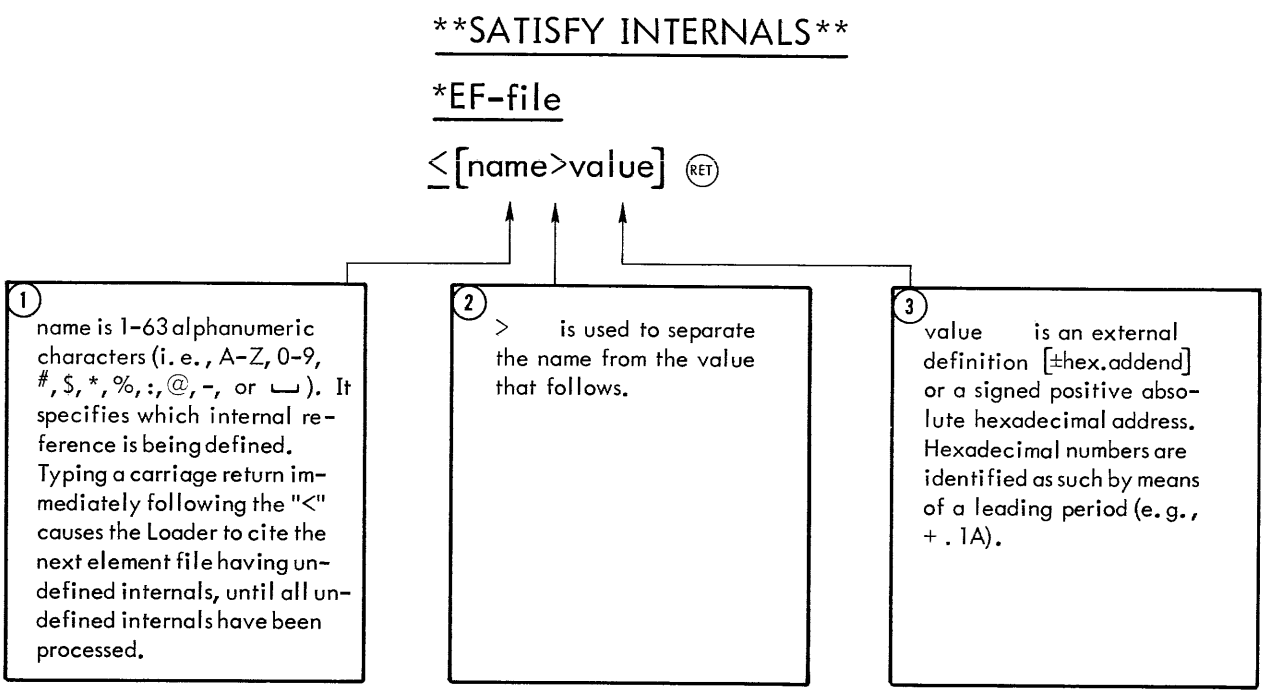
```
**SATISFY EXTERNALS**
<SOUBRIQUET>COGNOMEN+.B (RET)
< (RET)
```

In this example, the PREF named SOUBRIQUET is defined as the location 11 words higher than external definition COGNOMEN. The (RET) following the second prompt indicates that no more PREFs are to be defined.

SATISFY INTERNALS
(Satisfy Internal References)

SATISFY INTERNALS, followed by a line stating the name of an element file followed by a "<" character at the beginning of the next line, prompts the user to satisfy undefined internal references for the cited element file. The user may respond with the name and value of any one of the symbols listed as undefined internals (in that element file) in the load map.

This feature is provided only if the 'D' option has been specified.



Example:

```
**SATISFY>INTERNALS**
*EF-FIRSTFILE
<AGNOMEN>+.1A2B
<
*EF-NEXTFILE
<WHATEVER>THERE-.1
<
```

In this example, the symbol AGNOMEN in element file FIRSTFILE is defined as the absolute hexadecimal value 1A2B. The RET following the next prompt causes the Loader to cite element file NEXTFILE. The user then defines the symbol WHATEVER as one word location lower than external definition THERE (in the same element file). When all unsatisfied references have been defined, the console bell is rung.

8. FERRET SUBSYSTEM

GENERAL

FERRET is a utility subsystem that provides a general capability for obtaining information pertaining to entries in the file management system. It is a useful tool for checking files in storage, and provides functions for file manipulation (see Figure 8-1).

OPERATION

To call the subsystem, the operator types FE following the (!) prompt character from the Executive. The Executive will then reply by finishing the word (FERRET) and will display a prompt character (>), ready for the user's command.

```
! FERRET
>
```

Command keywords (e.g., LIST) may be spelled out completely, but only the initial letter is mandatory. If something is typed that the subsystem does not recognize, the message COMMAND NOT LEGAL is printed. When any operation is completed, the prompt character (>) is printed again. To exit from the subsystem and return to the Executive, the operator has only to type an X following the prompt character. It is also possible to exit to the Executive at any time, by typing ESC ESC.

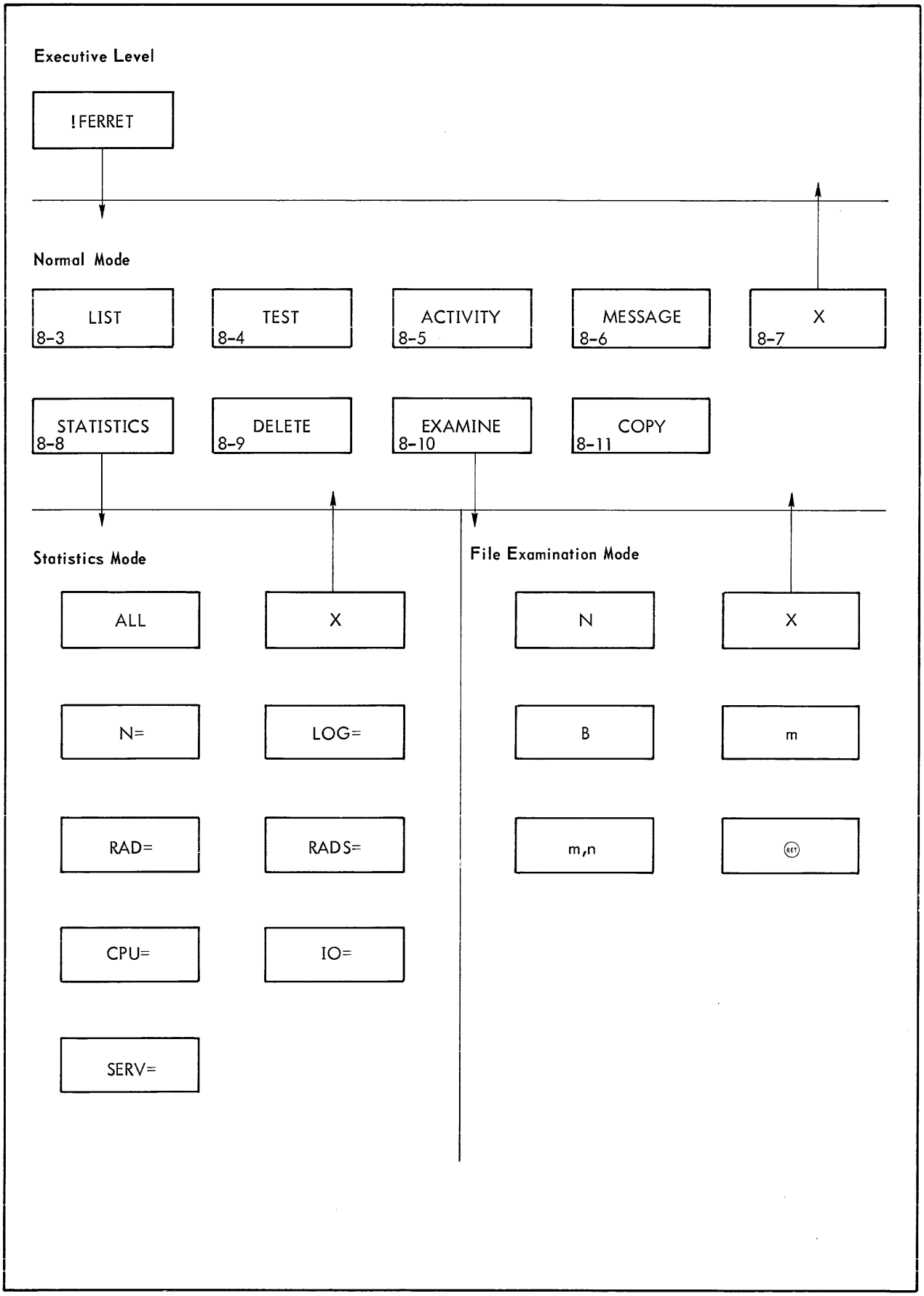
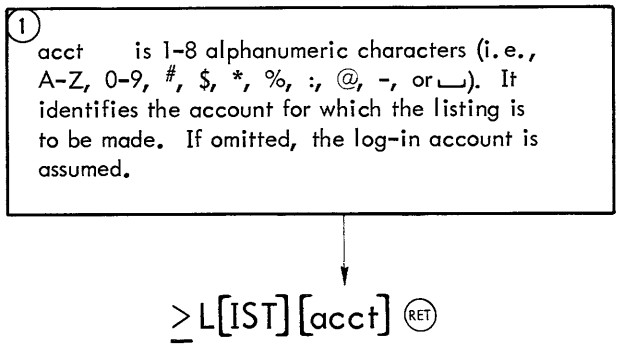


Figure 8-1. FERRET Commands (and Page Numbers)

LIST (List File Names)

LIST causes FERRET to print a list of all files in a specified account.



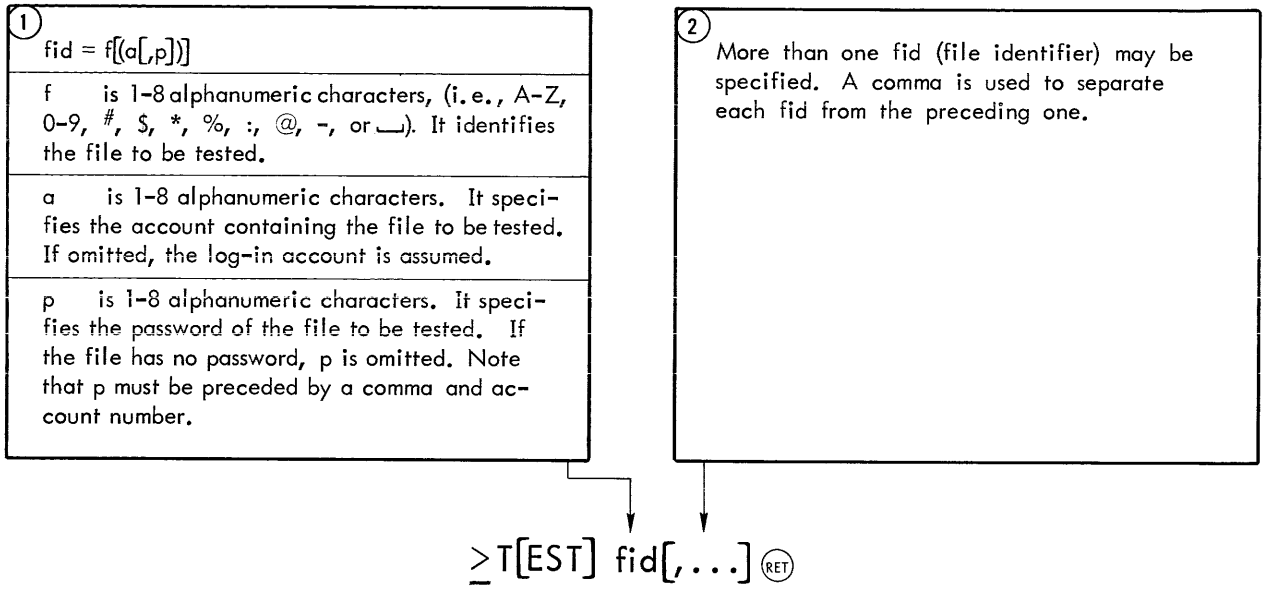
Example:

```
!FERRET
≥LIST HIS-ACCT RET
THISFILE
THATFILE
SOMEFILE
≥
```

In this example, FERRET lists "THISFILE", "THATFILE", and "SOMEFILE" as names of files in account HIS-ACCT.

TEST (Test File Accessibility)

TEST causes FERRET to determine whether the user may read one or more specified files.



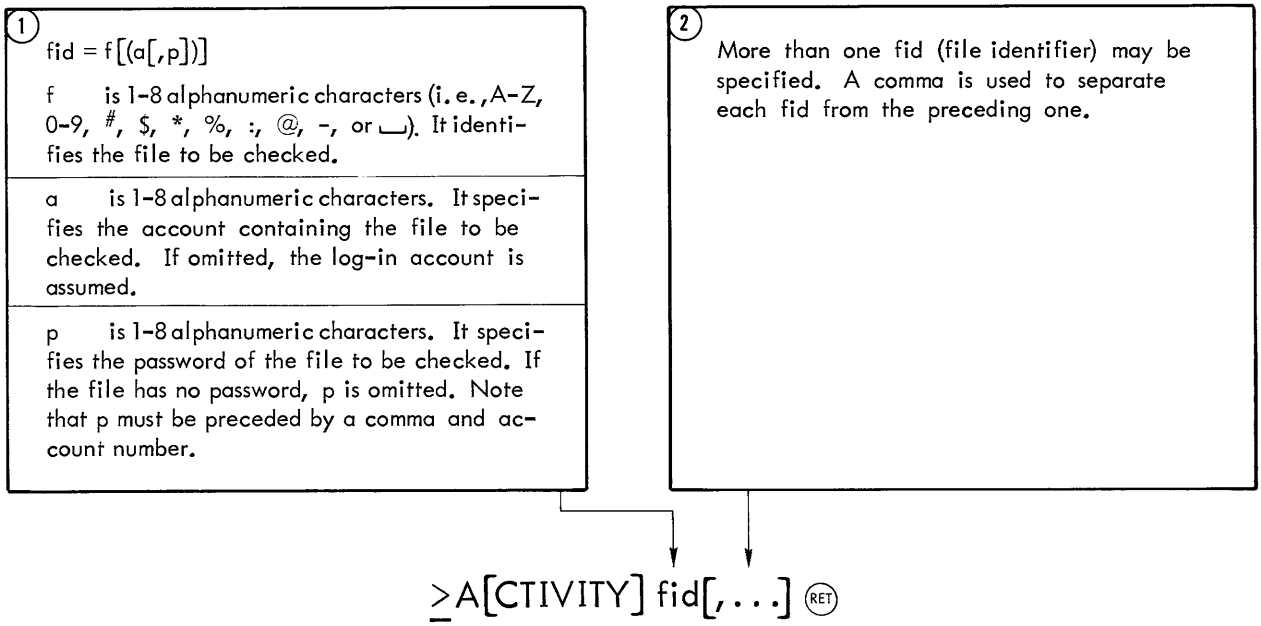
Example:

```
!FERRET
≥TEST THIS(MY,FILE),ALSO(ANOTHER) RET
THIS WAS CREATED 6,30,70 AND HAS
15 GRANULES IN IT.
CANNOT ACCESS FILE ALSO
≥
```

In this example, the user requests a test of file THIS having the password FILE and file ALSO in account ANOTHER. FERRET indicates that the user may access file THIS but not file ALSO.

ACTIVITY (Check File Activity)

ACTIVITY causes FERRET to check the current activity of one or more specified files.



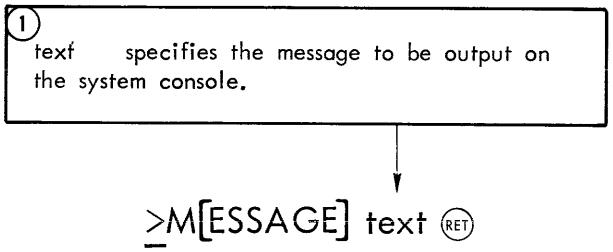
Example:

```
!FERRET
>A FILE(MY,#$*%) RET
FILE IS INACTIVE
>
```

In this example, the user requests a check of file FILE having the password #$*% in the log-in account. FERRET indicates that the file is inactive and, hence, may be opened.

MESSAGE (Message to Operator)

MESSAGE causes a specified message to be printed on the system operator's console, prefaced by a note identifying the originating console.



Example:

```
!FERRET
>M IS BATCH JOB Z9M9Z,SMITH DONE? RET
>
NOT YET
>
```

In this example, the user specifies a message to the operator and waits for the reply.

X (Return to Executive)

X causes control to be returned to the BTM Executive.

>X (RET)

Example:

```
!FERRET
>DELETE ANYTHING (RET)
>X (RET)
!
```

In this example, the X command is used to return control to the Executive after completion of FERRET operations. The exclamation character indicates that the Executive is in control.

STATISTICS (Give Statistics)

STATISTICS causes FERRET to enter the statistics mode.

>S[TATISTICS] (RET)

In this mode, FERRET prompts with a "#" character indented 4 spaces to the right. Following this, the user may request any or all of various statistics:

#ALL (RET)	Return all statistics listed below, then exit from the statistics mode to the normal mode.
#X (RET)	Exit from the statistics mode to the normal mode.
#N =	Return number of users currently logged in.
#LOG =	Return amount of time current user has been logged in.
#RAD =	Return number of RAD granules used in current session.
#RADS =	Return number of RAD granules available at start of current session.
#CPU =	Return current amount of CPU execution time used in current session.
#IO =	Return current amount of IO wait time used in current session.
#SERV =	Return current amount of Monitor service time used in current session.

Example:

```
!FERRET
>COPY THISFILE,THATFILE (RET)
>S (RET)
    #RAD=50
    #X (RET)
>
```

In this example, the user requests a listing of RAD space used for the current session. FERRET indicates that 50 granules have been used.

DELETE (Delete File)

DELETE causes one or more specified files to be deleted from the log-in account.

(1)

fid = f[(a,p)]

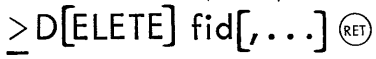
f is 1-8 alphanumeric characters (i.e., A-Z, 0-9, #, $, *, %, :, @, -, or ␣). It identifies the file to be deleted.

a is 1-8 alphanumeric characters. It identifies the log-in account.

p is 1-8 alphanumeric characters. It specifies the password of the file to be deleted. If the file has no password, p is omitted. Note that p must be preceded by a comma and account number.

(2)

More than one fid (file identifier) may be specified. A comma is used to separate each fid from the preceding one.



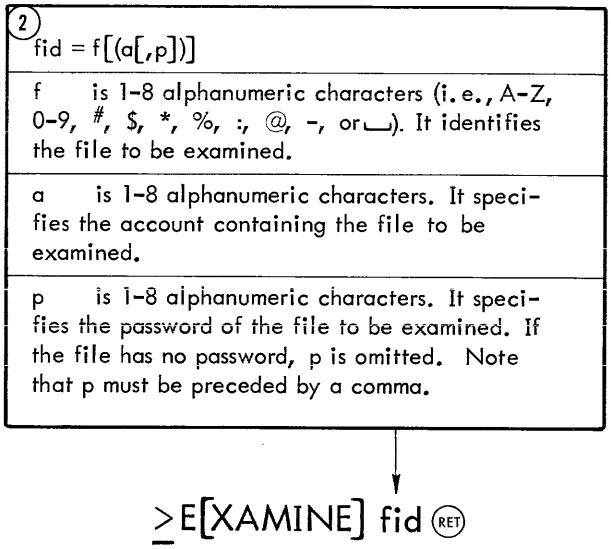
Example:

```
!FERRET
>DELETE THISFILE,THATFILE,ANOTHER RET
>
```

In this example, the user requests FERRET to delete files THISFILE, THATFILE, and ANOTHER from the log-in account.

EXAMINE (Examine File)

EXAMINE causes FERRET to enter the file examination mode.



In the file examination mode, FERRET prompts with a "#" character. Following this, the user may request any of various actions:

#X (RET)	Exit from file examination mode to the normal mode.
#N (RET)	Display, in decimal, the number of records in the file.
#[H] (RET)	Print, in EBCDIC (or hex., if H is specified), the contents of all records in the file.
#B[m[,n]] (RET)	Print, in decimal, the number of bytes in record m, records m through n, or the entire file.
#[H]m (RET)	Print, in EBCDIC (or hex., if H is specified), the contents of record m.
#[H]m,n (RET)	Print, in EBCDIC (or hex., if H is specified), the contents of records m through n.

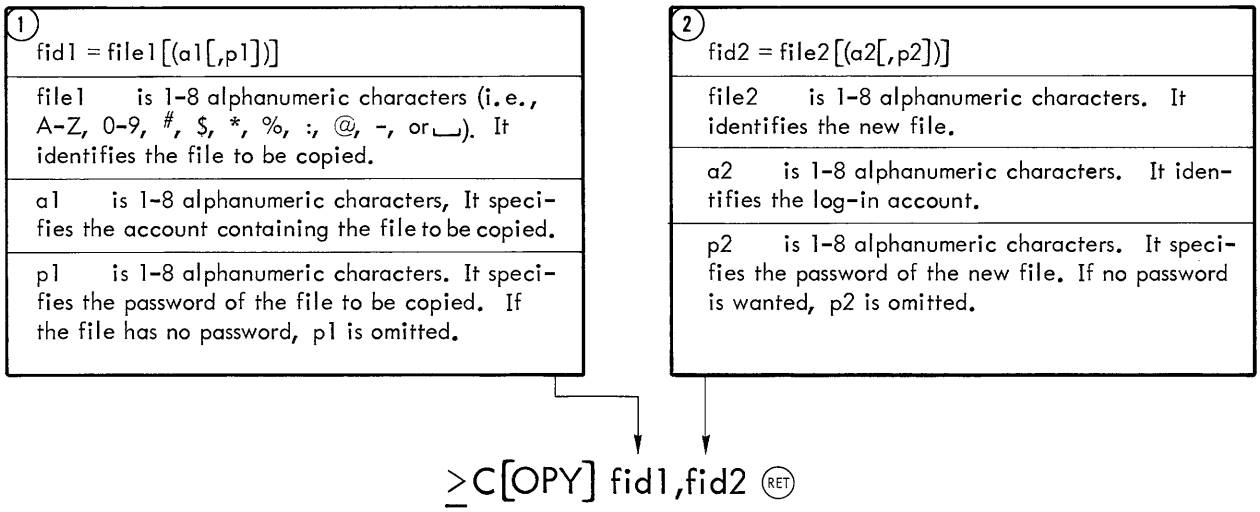
Example:

```
!FERRET
≥E PLURIBUS(UNUM) (RET)
#25 (RET)
LOC8 BE $+3
#X (RET)
≥
```

In this example, the user requests FERRET to print the contents of record 25 of file PLURIBUS in account UNUM. FERRET responds by printing "LOC8 BE $+3" and the user then ends the file examination mode by typing an X followed by a carriage return.

COPY (Copy File)

COPY causes FERRET to create a new copy of an existing file.



Example:

```
!FERRET
≥C SOMEFILE(:SYS),MYCOPY(MY,XANADU) RET
≥
```

In this example, the contents of file SOMEFILE are copied from account :SYS into the log-in account with the name MYCOPY and having the password XANADU.

FERRET MESSAGES

Message	Meaning
COMMAND NOT LEGAL	FERRET has encountered an illegal command. The command must be retyped correctly.
CANNOT ACCESS FILE xxxxxxxx	The specified file cannot be accessed by the user.
xxxxxxxx IS INACTIVE	The specified file can be opened.
xxxxxxxx IS ACTIVE	The specified file cannot be opened.
CANNOT DELETE FILE xxxxxxxx	The specified file cannot be deleted by the user.
CANNOT CREATE FILE xxxxxxxx	The specified file cannot be opened in the output mode.
CANNOT COPY - - RECORDS TOO LARGE	A COPY command has been aborted because a record longer than 512 words was encountered.
RECORD EXCEEDS BUFFER SIZE, 512 WORDS GIVEN	A record longer than 512 words was encountered in the file examination mode.
FIRST RECORD NON-EXISTENT	The first record requested in a file examination command was not found in the file.
UNEXPECTED EOF AFTER RECORD j	An end-of-file was encountered prior to reading the last record specified in a file examination command.

9. SYMBOL SUBSYSTEM

GENERAL

The Symbol subsystem assembles programs in XDS Symbol source language. The language is described in the XDS Symbol/Meta-Symbol Reference Manual (90 09 52). Input is source coding, either typed directly at the user's terminal or from a file. Output is a program listing and/or an assembled object program which may be loaded and executed by the Loader subsystem (refer to Chapter 7).

INPUT/OUTPUT ASSIGNMENTS

Prior to calling the Symbol subsystem, it is possible to make input/output assignments by use of the Executive ASSIGN command (refer to Chapter 3). Input/Output assignments are listed in Table 9-1.

Table 9-1. Input/Output Assignments

Symbol	Description
M:SI	Source language input. The default assignment is to the user's terminal. An alternative is for the user to specify a file previously created by use of the EDIT subsystem.
M:LO	Listing output. Default assignment is to the user's terminal.
M:BO	Binary output of assembled object program. By default this goes into temporary file BOTEMPx, where "x" is the special ID for the user's terminal. The user may also specify a file of his own. This is the file to be specified to the Loader when it is desired to run the program.

ASSEMBLER OPTIONS

The subsystem is called following the Executive prompt character by typing SY. The Executive will then type the rest of the word and turn control over to the Symbol Subsystem, which then requests a list of options. The operator may specify options listed in Table 9-2, separating them with commas. If no options are specified (carriage return only), all the options listed in Table 9-2 and shown in Figure 9-1 are assumed.

Table 9-2. Symbol Options

Symbol	Option
BO	Binary output of an assembled object program.
LO	Output a program listing.
CN	Include a cross-reference list in the program listing.
SD	Include special symbol tables for use by the Loader subsystem's debugging feature at run-time.

The following is an example of a Symbol assembly with source input from a file on the disc, and listing output to a file on the disc. All options are selected with the exception of debugging feature symbol tables (SD).

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

LISTING FORMAT

If the program listing is typed on the user's terminal, it will automatically be reformatted to fit the carriage width. Each listing line will be typed as two lines:

1. The first line will contain the source image.

2. The second line will contain the line number and object code portion of the normal listing. In addition, if the source file was on disc in EDIT format, the EDIT file sequence number will be typed in decimal format.

If the assembly listing is not being displayed at the terminal, any errors found in the assembly are displayed both at the terminal and in the listing file. Three lines are typed at the terminal:

1. The offending source line.

2. The normal Symbol error indicator (****) and a letter positioned under the image.

3. The line number, object code produced, and sequence number of the record.

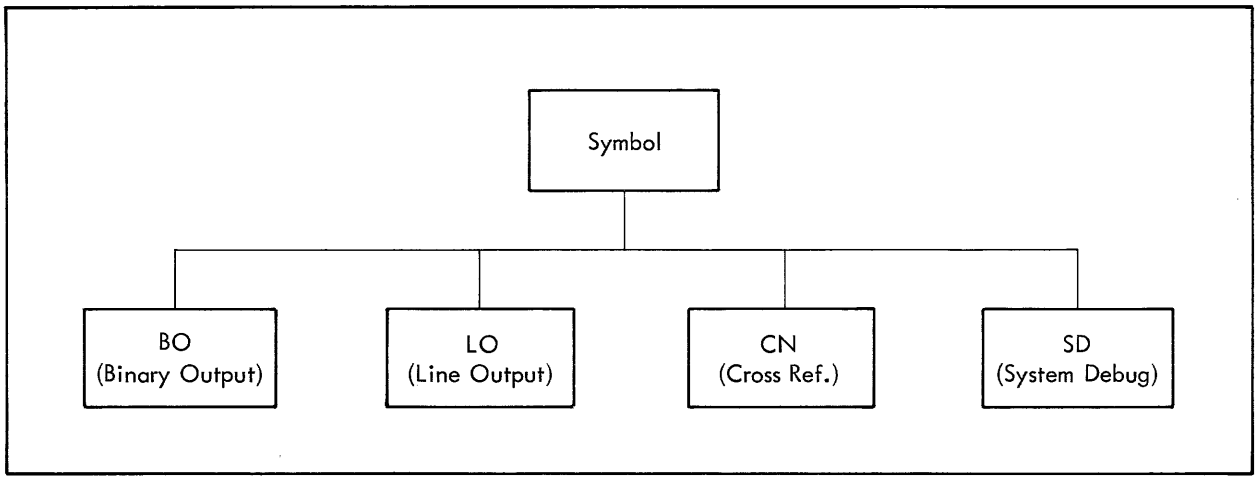


Figure 9-1. Symbol Subsystem

10. BATCH PROCESSING MONITOR (BPM) SUBSYSTEM

GENERAL

Batch jobs, complete with control commands, may be entered into the queue of jobs to be processed (see Figure 10-1). This batch processing is performed simultaneously with on-line operations of the computer system, and independently of them. For this reason, the user knows only that when a job is scheduled, it will be run in its proper turn sometime later. Inquiries are permitted from the on-line terminal pertaining to the status and progress of the job.

OPERATION

The BPM Subsystem may be called directly and a job inserted, or the operator may first contruct a file using the EDIT subsystem and have BPM read the file. The BPM subsystem expects that input is to be typed at the user's terminal. If it is desired that the input be read from a previously created file, the assignment is changed (via an ASSIGN M:SI) prior to calling BPM. After log-in, the operator types the letters BP for the subsystem, which replies with the question, INSERT JOB?. The operator types Y if a job is to be entered, or N for status check of previously scheduled job, then a carriage return (see Figures 10-2 and 10-3).

!BPM

INSERT JOB? {Y / N} (RET)

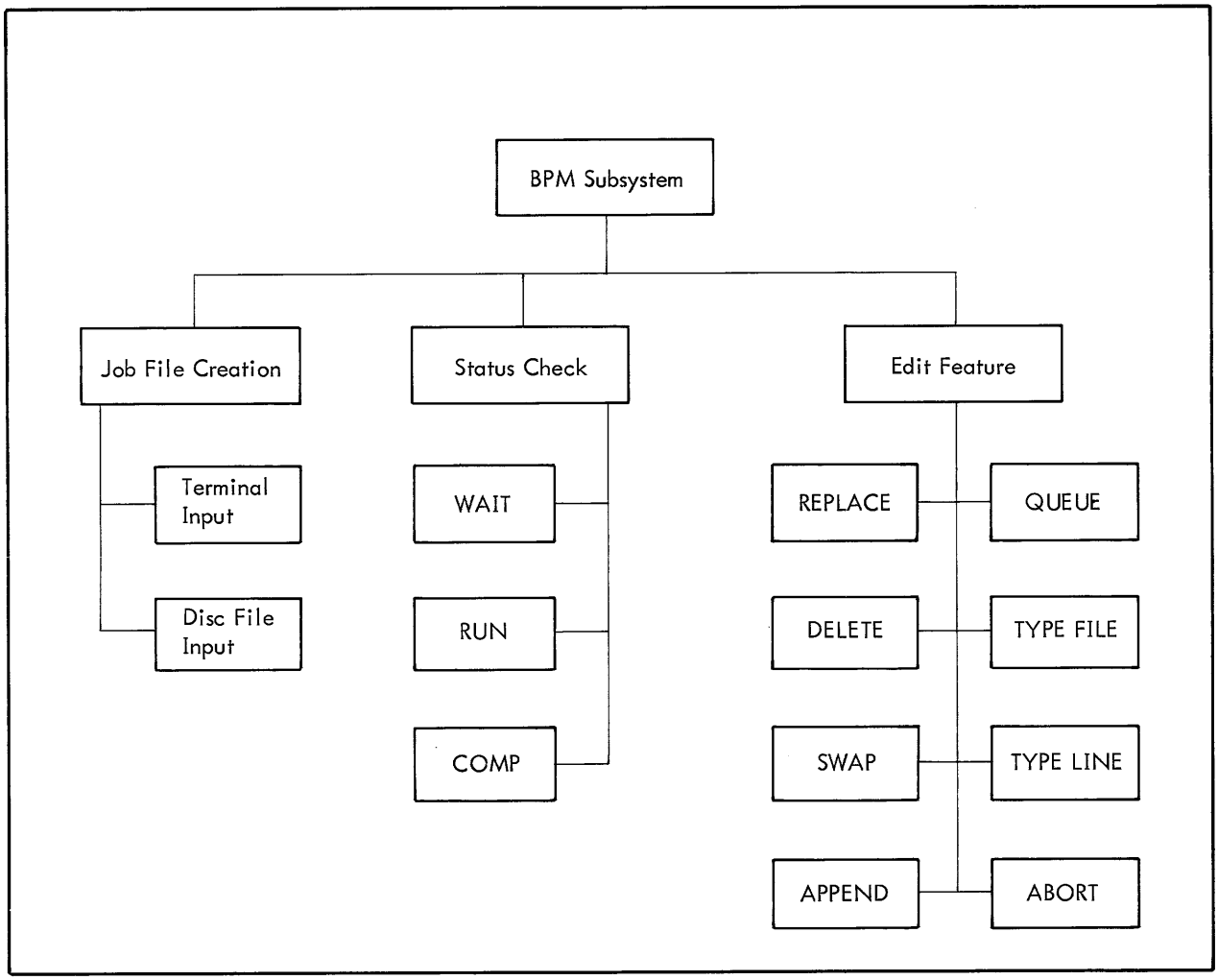


Figure 10-1. BPM Subsystem

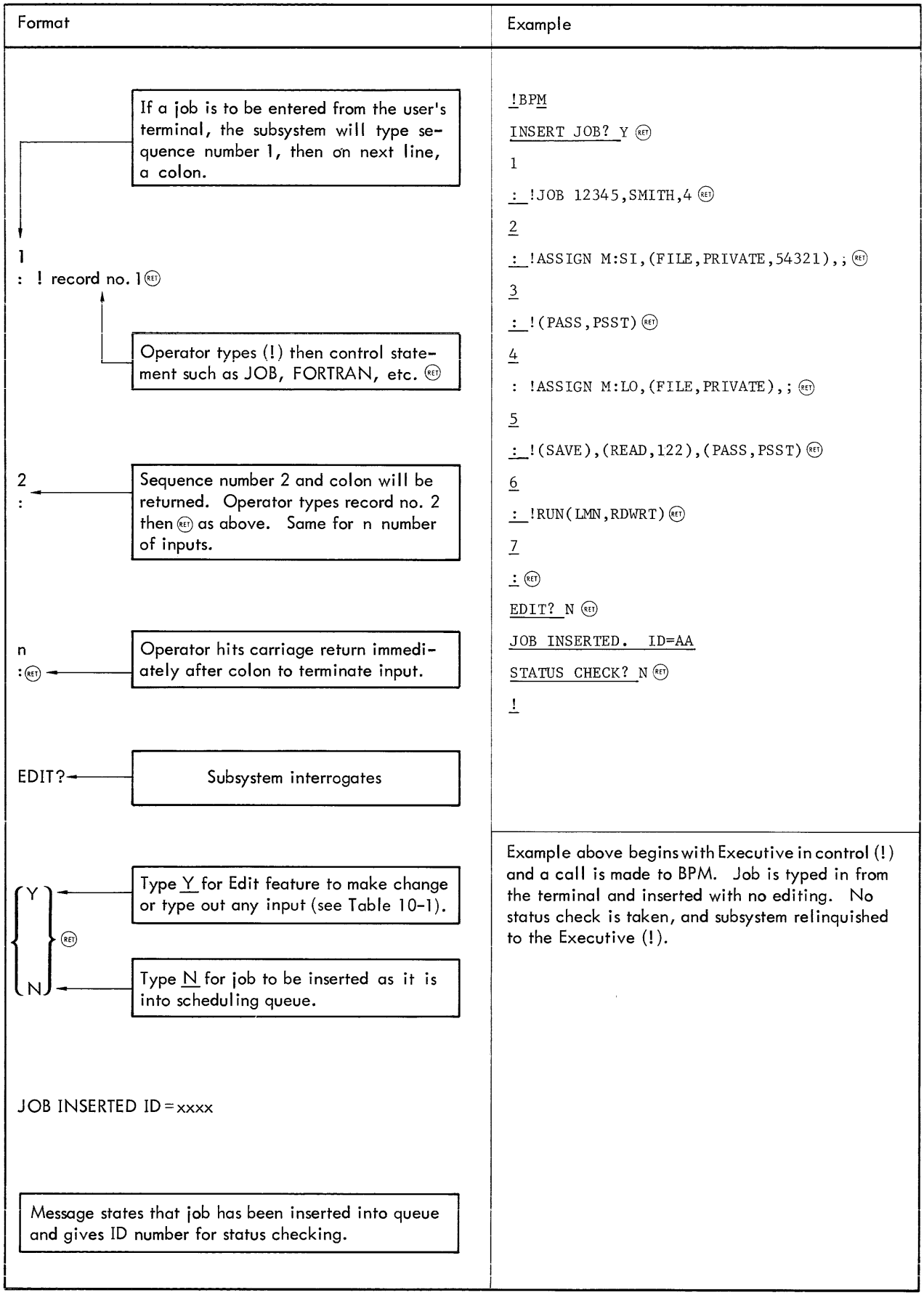


Figure 10-2. Terminal Input

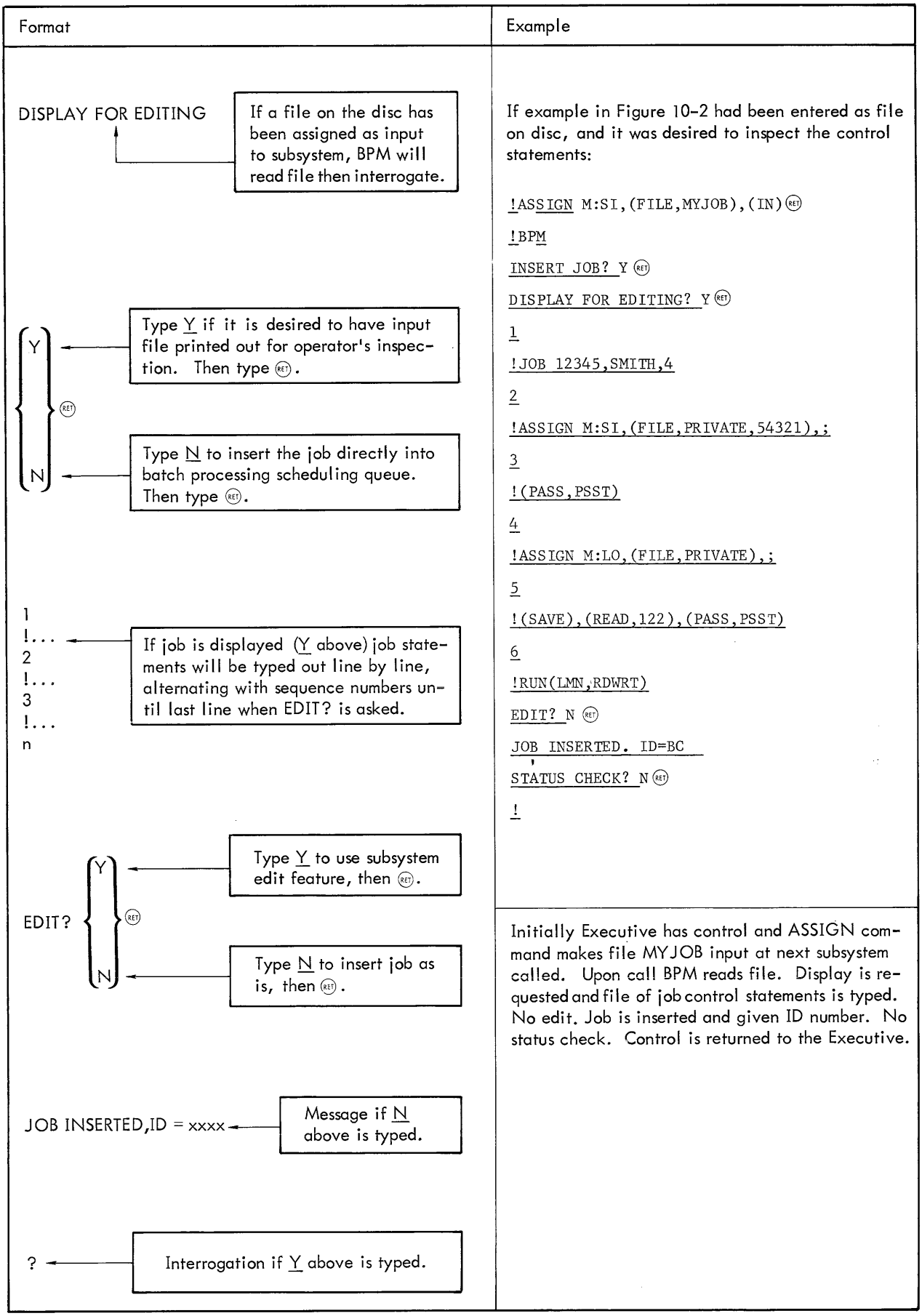


Figure 10-3. Disc File Input

Table 10-1. BPM Edit Feature

Command	Function to be Performed
?T ⏎	Type file
?Txx[,yy] ⏎	Type lines xx [thru yy]
?X ⏎	Abort current task
?G ⏎	Go (run current task)
?Sxx,yy ⏎	Swap lines xx and yy
?Dxx[,yy] ⏎	Delete xx [thru yy]
?Axx ⏎	Add the following lines after line xx
?Rxx[,yy] ⏎	Replace xx [thru yy] with the following lines

BPM STATUS CHECK

It is possible to check the status (waiting, running, or completed) of any job previously inserted into the batch processing queue. When the subsystem inserts a job, it inquires whether the user wishes to check status, or the user may call BPM and enter the status-checking routine by answering N to the question INSERT JOB?. The subsystem then prints out ID =, and the operator types in the hexadecimal ID number.

ID =A4 ⏎

The subsystem then types out one of the following:

1. RUNNING
2. COMPLETED
3. NON-EXISTENT
4. WAITING ON OUTPUT
5. WAITING: n AHEAD
   CURRENT ID: x

BPM MESSAGES

Message	Meaning
MISSING !JOB COMMAND, OR RECORDS OUT OF ORDER EDIT?	JOB command was not recognized in first line, so job cannot be scheduled. Either the edit feature to correct the condition, or X command to abort must be used before it is possible to continue.
IMPROPER JOB PRIORITY EDIT?	An improper job priority value has been given in the job statement. Hexadecimal numbers 1 to F only are valid. This message might result if an attempt was made to continue the job statement to the second line. This will not prevent insertion of the job but priority level is automatically set to 1. If the job does not specify a priority level at all, level 1 is assumed with no message.
!FIN CARD IGNORED	A FIN command has been discovered in the job and has been ignored. Insertion of the job is not prevented.
BAD I-O. ABNORMAL CODE - xx	An input/output error or malfunction has occurred during processing by the subsystem. The I/O system code for the fault is given. BPM aborts, returning control to the Executive. If job has not been inserted, BPM must be called.
FILE TOO LARGE. TASK ABORTED	The input job is too large to be handled. Begin with smaller job file.
NO INPUT DATA. BYE	Nothing in job file. No action, and control returns to the Executive.
NON EXISTENT LINE	A line referenced by an Edit command does not exist in the job file. It is possible that the line was inadvertently deleted.
ERRONEOUS COMMAND IGNORED	An Edit command has been rejected because the command or its operands have been improperly written. No action has been taken. Retype the command.
ILLEGAL CAL	The present system monitor is not capable of accepting a batch job. The job cannot be inserted.
INVALID ID	An erroneous job ID has been given in a status-check request. Retype ID.
IMPROPER JOB ACCOUNT	The account number in the job statement does not match the account number given at log-in. Use Edit feature to correct job statement.
AUTHORIZATION REQUIRED	User is not authorized to enter a job from a terminal.

11. RUN SUBSYSTEM

GENERAL

The RUN subsystem allows the on-line user to execute previously formed load modules. It simulates several BPM services that otherwise would not be available to the on-line user, allowing overlayed modules to be executed. Thus, most load modules capable of batch execution will also execute on-line. However, the execution bias must be at least one page above the lower limit of the user area. If the module is relocatable, it will be relocated automatically.

All program changes made via the RUN subsystem apply to the current memory image of the program. If the affected location lies within the bounds of the segment selected by the most recent s;S command (but not in the root or backward path), the change is also added to a table of modifications to be made each time the segment is brought into core. Changes affecting only the current image are destroyed if that image is overlayed as the result of an M:SEGLD.

Two or more breakpoints may be placed at the same core location, but in different overlay segments. When a breakpoint is cleared, it will not be activated unless explicitly restored by an e;B command; if the affected segment is currently in core, the replaced instruction is restored in the core image as well as in the load module.

A load module executed under control of the RUN subsystem must be smaller than the on-line user area minus the size of the RUN subsystem itself (which occupies about 2.5K of core). During execution, the user program may acquire dynamic data pages in an amount up to the difference between user area size and load module size.

If the load module is overlayed, it may have not more than three files open at any one time. Nonoverlayed programs may have up to four files open at the same time. All I/O done through DCBs will be output to the user's terminal unless an Executive-Level disc-file assignment of the DCB exists or the DCB contains a "built-in" disc-file assignment.

In addition to the BPM CALs available to all on-line programs, the RUN subsystem allows the use of the following:

M:SEGLD	M:GL
M:GP	M:GCP
M:FP	M:FCP
M:SMPRT	M:TIME

The RUN subsystem is entered by the Executive command

!RUN ⏎

On entry, RUN requests load module identification, as follows:

LOAD MODULE FID:

The user then supplies the file identifier of the load module in the form

name [([acct] [,pass])] ⏎

For example:

```
!RUN ⏎
LOAD MODULE FID: SL1(:SYS,PASS) ⏎
```

PRE-EXECUTION DEBUGGING

When the load module has been identified, RUN will wait for one or more commands pertaining to execution of the program. The following commands, comprising a subset of the Delta debug language, may be used.

[s];S Selects a symbol table by overlay segment name (or root if s is omitted). All REFs and DEFs of the specified segment and its backward path are made available to RUN.

[e]/ Displays the contents of cell e (or register 0 if e is omitted). The cell may then be modified (i.e., it is "open").

e1,e2/ Displays the contents of cells e1 through e2 and opens cell e2.

(RET) Closes the currently open cell.

e (RET) The expression e is assembled and stored in the currently open cell.

(LF) Opens the next higher cell.

e (LF) Modifies the currently open cell and opens the next higher cell.

↑ Opens the next lower cell.

e↑ Modifies the currently open cell and opens the next lower cell.

(TAB) Displays and opens the cell addressed by the last quantity typed (represented by the symbol ;Q).

e (TAB) Modifies the currently open cell and then displays and opens the cell addressed by the last quantity typed.

;G Begins execution at the point determined by the contents of the current execution location counter (represented by the symbol ;I).

e;G Begins execution at the location specified by the expression e.

;P Causes execution to proceed from a breakpoint.

;B Displays all breakpoints as they are encountered during execution.

e;B Inserts a breakpoint at the location specified by the expression e.

n;B Clears (i.e., removes) breakpoint n ($0<n\leq 8$).

e;I Stores the value of the expression e into the current execution location counter.

e;C Sets the value of the condition code to that of the expression e.

;A Causes location values to be displayed in hexadecimal.

;R Causes location values to be displayed as a symbol plus a hexadecimal offset.

Expressions in the above commands may comprise any combination of sums and/or differences of symbols and constants. Constants may be either decimal or hexadecimal (indicated by a leading period). In addition to any of the symbols in the symbol table most recently specified by an s;S command, any of the following symbols may be used in an expression.

Symbol	Meaning
.	Address of most recently opened cell
;I	Value of current execution location counter
;C	Condition code value
;Q	Most recently displayed value