OSBORNE EXECUTIVE®

Reference Guide

Part No. 3F00186-00

Osborne Computer Corporation reserves the right to make changes or improvements to the equipment, software, and documentation herein described at any time and without notice.

Notice: Please be advised that any tampering with the internal hardware of the Osborne Executive Computer may void the warranty.

Table of Contents

Introduction

This **Reference Guide** includes:

- a complete list of commands for each of the programs supplied with the **Osborne Executive**
- a list of error messages for these programs
- technical information and system specifications about the **Osborne Executive** computer.

The **Reference Guide** is a summary of information. It supplements the material in the other **Osborne Executive Guides** and presupposes basic familiarity with using the computer.

Glossary

Terms used in the **Reference Guide** have the meanings shown below.

Block—section of text
Byte—one character of information
Cell—a single SuperCalc spreadsheet location
Constant—a fixed-value number, integer, or string
Default—settings automatically in effect unless changed
Delim—a punctuation mark which separates characters or entries
Dot command—command character preceded by a period (.)
(Exp)ression—a single constant, variable, or sequence of characters
Hex—hexadecimal notation
Integer expression—an expression with integer value
Justification—alignment of text margins
Logged drive—disk drive in use
Numeric exp—an expression with real or integer numeric value
Parm—parameter
Prompt—a screen symbol meaning the system is ready for an instruction or command
Real exp—an expression with a real numeric value
Scroll—screen movement
Statement number—a valid CBASIC or MBASIC line number
String exp—an expression with a string value
System—CP/M Plus Operating System
Toggle—switch which turns a feature or function ON and OFF
Variable—expression or character which can be replaced by specific values or names
Worksheet—the SuperCalc spreadsheet

Symbols

<**ESC**>—the Escape key
R/O—Read only
x:—variable drive identifier
<**cr**>—Carriage Return
^—CTRL key
#—number
K—kilobyte
{ }—enclosed parameters may be repeated
[ ]—enclosed parameters are optional

WordStar

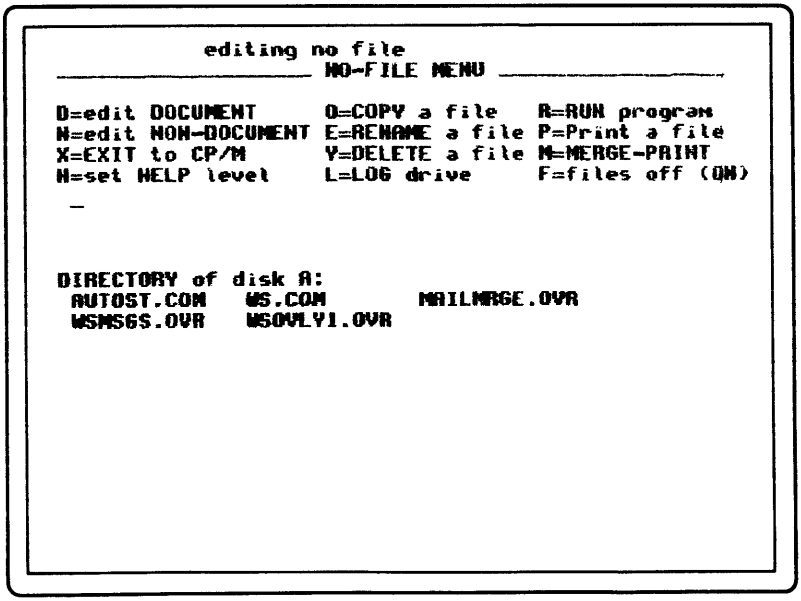
Getting WordStar Started

In order to start WordStar, place the WordStar diskette in drive A and a formatted diskette in drive B, then press **RETURN**. The diskette in drive B will be used to store text files you generate.

Once WordStar is running the message

editing no file

appears at the top of the screen with a menu of WordStar operations below it. Each operation is identified by a single letter which you press to select the desired operation. Pressing **CTRL** is not necessary when you enter commands from the No-File menu—in fact this is the only menu from which WordStar uses non-control letters as commands. Here is the No-File menu you will see:



```
                editing no file
 ________________ NO-FILE MENU ________________

D=edit DOCUMENT      O=COPY a file    R=RUN program
N=edit NON-DOCUMENT  E=RENAME a file  P=Print a file
X=EXIT to CP/M       Y=DELETE a file  M=MERGE-PRINT
H=set HELP level     L=LOG drive      F=files off (ON)
 _


DIRECTORY of disk A:
 AUTOST.COM   WS.COM        MAILMRGE.OVR
 WSMSGS.OVR   WSOVLY1.OVR
```

Initiate the following operations by pressing the letter indicated:

CREATE OR EDIT A DOCUMENT—*creates or retrieves a document file*

D is used for general word processing. A file name is requested. Supplying a new file name causes a new file to be created under the specified name. Entering an existing file name causes the specified file to be fetched from the current drive and displayed on the screen.

EDIT A NON-DOCUMENT—*creates or retrieves a nontext file*

N is used to create or edit a data file for merge-printing. The non-document mode is generally used by programmers to create source-program files.

MERGE-PRINT—*initiates merge-printing of a file*

M initiates a merge-print operation. Merge-print merges files during printing, thereby generating form letters, boilerplate text, mailing lists, and large documents.

DISPLAY DIRECTORY—*toggles appearance of file directory*

F turns the file directory OFF or ON again. The menu displays the current status. When the file directory is ON, the names of all text files on the logged drive are displayed.

CHANGE LOGGED DISK DRIVE—*activates alternate disk drive*

L selects the "logged" or "active" drive; WordStar assumes that all text files are on the diskette in this drive. When you press **L**, a message asks you to select the drive to activate. Type the drive letter followed by a colon, and press **RETURN**.

RUN A PROGRAM—*runs a program from the No-File menu*

R runs a program without exiting from WordStar. You can execute CP/M Plus programs, such as DIR and SHOW, by supplying their file names to the COMMAND? prompt.

HELP SET—*establishes the level of assistance displayed on the screen*

H selects the level of information the menus display. As you become more experienced with WordStar, you may want to decrease the level of assistance displayed in the menus at the top of the screen.

EXIT TO SYSTEM—*relinquishes control to CP/M*

X exits WordStar and returns control to the CP/M operating system.

PRINT A FILE—*toggles printing of a named file*

P initiates and halts printing of a text file. The name of the file to print is requested, followed by prompt questions regarding the print operation. The current print status is displayed on the screen.

DELETE A FILE—*deletes the specified file*

Y deletes a file from the diskette in the currently active drive. A file name is requested, and the named file is subsequently deleted.

FILE COPY—*copies a file*

O

copies a file from source to destination. The name of the file to be copied, and the name of the file where the copy is to be transferred are requested. If the destination file name exists, it will be erased unless the command is abandoned.

FILE RENAME—*renames a file*

E

assigns a new name to a specified file. The existing file name is requested; when the name is supplied, the new file name is requested. Supplying the new file name and pressing **RETURN** completes the renaming process.

Block Operations

An entire section of text may be moved, copied, deleted, or written to another file. All these operations are performed by block commands. To manipulate a section of text, you must first "block" it. To block a section of text, place the cursor at the beginning of the text to be blocked and type **^KB**; a < B > will appear on the screen to indicate the position of the beginning marker. Next, move the cursor to the end of the text you wish to enclose within the block and type **^KK**. The entire marked block will be displayed in half intensity so it is easy to distinguish.

You can set a beginning and end block marker in the middle of a paragraph to manipulate a sentence, or in the middle of a sentence to manipulate a word. A feature for manipulating columns of text is also provided. **^KN** turns this column block feature ON or OFF.

You can have only one marked block in your text file at a time. A new **^KB** or **^KK** command will replace any previous block beginning or end marker, if one exists. The marked block is always the implied source for any block operation. Those block commands that require a destination assume the cursor position as the location. Following are the BLOCK commands:

BLOCK MARK BEGINNING—*defines the beginning of a block*

^KB marks the beginning of a block at the cursor position. The beginning block marker is displayed as a < B >. Alternatively, **^KB** hides the displayed marker.

BLOCK MARK END—*defines the end of a block*

^KK marks or hides the ending of a block. It is used with the beginning block marker to enclose a section of text so you can perform block operations on it.

BLOCK COLUMN—*specifies a column block*

^KN switches between column block and normal block modes. When in column block mode, a block extends from the column in the line of the beginning marker to the column in the line of the end marker. The usual block commands are used to manipulate the marked block.

BLOCK COPY—*copies a block from source to destination*

^KC

copies the currently marked block of text to where the cursor is positioned. The original text remains unaltered. The block markers move with the text.

To copy a text block, place the cursor at the desired destination and type **^KC**. The current block is then copied to the cursor position. The cursor stays at the beginning of the copy.

The block markers are transferred with the copy of the block and remain displayed. The command **^KH** hides the block markers following a block operation.

You may make as many copies of the block as you desire by typing **^KC** as many times as necessary. Copies may be made in different locations by moving the cursor to the desired position between copy commands.

Using **^QV** after a block copy returns you to the source of the block.

After copying a block, you may use the REFORM command, **^B**, to reformat the text.

NOTE

Unlike Block markers, Place markers do not move with the marked block.

BLOCK DELETE—*removes a block from a file*

^KY

deletes the currently marked block. The block must be visible (not hidden) for **^KY** to delete it. Since large amounts of text can be deleted by accident, it is recommended that a block be hidden when you're not operating on it.

When a block is deleted, both block markers are hidden and left at the position of the deleted text. You can use the **^QV** command to move the cursor to the delete location.

BLOCK MOVE—*moves a block to the desired position*

^KV

moves a block to the cursor position. Place the cursor at the desired destination and type **^KV**. The cursor is left at the beginning of the moved text.

The block markers move with the block and remain displayed. The command **^KH** hides the block markers after the block operation.

Following a BLOCK MOVE, the **^QV** command returns to the source of the block. Also, the REFORM command, **^B**, can be used to reformat text after the block is moved.

NOTE

There is a limit to the size of block you can move or copy. If a BLOCK-TOO-LONG error occurs, divide the block into smaller sections and perform the BLOCK operation on each section.

BLOCK WRITE TO FILE—*sends content of block to a named disk file*

^KW transfers the contents of a block to another file on the logged drive, or drive specified. The name of the file where the block is to be transferred is requested, the contents of the block are then written to the named file. The entire contents of any existing file with the same name will be erased. To prevent unintended erasures, if the file name that is specified already exists, WordStar responds:

```
FILE d:name.typ EXISTS....OVERWRITE? (Y/N):
```

Pressing **Y** causes the BLOCK WRITE function to write over the previous contents of the specified file; pressing **N** causes the file name to be requested again.

BLOCK WRITE lets you extract text from a document and save it as a separate document. BLOCK WRITE may also be used to move a section of text large distances within a file: write the block to a temporary file, then move the cursor where you want the block moved and read the file containing the block with **^KR**.

BLOCK HIDE/REDISPLAY—*toggles display of a block*

^KH hides a text block so that no BLOCK operations can be performed on it. Though hidden, the text remains blocked until another section of text is blocked. Alternatively, the **^KH** command redisplays the hidden marked block making it once again subject to BLOCK operations.

Changing the Logged Disk Drive

The standard procedure for starting WordStar involves activating the disk drive where files will be stored. After you place the WordStar program's diskette in drive A and press **RETURN**, the A drive is activated (logged). What this means is that, unless you append the drive identifier B:, any files you create or edit will be stored on the program's diskette in drive A.

"Logging onto" the B drive provides the most convenient method of managing your files. You should log onto drive B before issuing any other command from the No-File menu. Subsequently, all your files will be stored and retrieved from the logged drive B. Here is a summary explanation of the command used to log drives:

CHANGE LOGGED DISK DRIVE—*activates alternate disk drive*

^KL (**L** from the No-File menu) changes the logged drive where files will be stored. The currently active drive is identified, and you are asked which drive to log. The file directory reflects the contents of the currently active drive.

After you type **^KL** (or **L**), the name of the drive where files are currently being stored is displayed. You are asked to identify the disk drive to activate. Enter the drive letter followed by a colon (**A:** or **B:**). Files will subsequently be written to, and read from this drive.

Remember, WordStar will always read from and write to files on the currently logged disk drive unless you explicitly specify the other drive by appending a disk identifier (A: or B:) to the front of a file name.

Cursor Motion

Cursor-motion commands move the cursor within a document. Basic cursor movement is accomplished using six control characters positioned according to the direction they move the cursor:

```
      ^E
^A  ^S  ^D  ^F
      ^X
```

The arrow keys, as well as a variety of other commands, also move the cursor to specific locations in the file. Following are the cursor-motion commands:

CURSOR LEFT A CHARACTER—*moves cursor left one character*

^S, ^H,

or ← each move the cursor one character position to the left. The cursor will move to the end of the preceding line if it is located at the beginning of the current line. These commands are used to backspace over characters and to make corrections.

CURSOR LEFT A WORD—*moves cursor left a word*

^A moves to the beginning of the word to the left of the cursor. A word is a string of characters separated by a punctuation mark (. , : ; ! ?, a space, or a carriage return).

CURSOR LEFT EDGE OF SCREEN—*moves cursor to left edge of screen*

^QS moves the cursor to the far left of the screen.

CURSOR RIGHT A CHARACTER—*moves cursor right one character*

^D or $\boxed{\rightarrow}$ moves the cursor one character position to the right. If the cursor is positioned at the end of the current line, it will go to the beginning of the next line.

CURSOR RIGHT A WORD—*moves cursor one word to the right*

^F moves the cursor to the beginning of the next word.

CURSOR RIGHT END OF LINE—*moves cursor to the end of a line*

^QD moves the cursor to the end of the current line of text. To get the cursor to move beyond the actual characters at the right end of a line, you must space or tab over to the desired column.

CURSOR UP A LINE—*moves the cursor up one line*

^E

or ↑

moves the cursor to the next line above. The cursor remains in or near the same column.

CURSOR DOWN A LINE—*moves the cursor down a line*

^X

or ↓

moves the cursor down to the beginning of the next line in the document. The cursor remains in the same column, but will move to the left to avoid landing beyond an end of a line; the cursor will also jog around print-control characters when necessary.

CURSOR TO BEGINNING OF FILE—*positions cursor at file's beginning*

^QR moves the cursor to the beginning of the file being created or edited. If the document involved is a large one, and you are near the end, use the FILE SAVE command, **^KS**, since it is faster and uses up less temporary diskette file space.

CURSOR TO END OF FILE—*positions cursor at file's end*

^QC moves the cursor to the end of file. The cursor ends up in the position following the last character of the document.

CURSOR TO SCREEN BOTTOM—*moves the cursor to the screen bottom*

^QX moves the cursor to the bottom of the text displayed on the screen. The cursor remains in its current column position.

CURSOR TO SCREEN TOP—*moves the cursor to the screen top*

^QE moves the cursor to the top of the currently displayed text.

CURSOR TO BLOCK BEGINNING—*moves cursor to block's beginning*

^QB moves the cursor to the beginning of the currently marked block. If the text block is hidden, the beginning marker is redisplayed.

CURSOR TO BLOCK END—*moves cursor to the end of a block*

^QK moves the cursor to the end of the currently marked text block. If the block is hidden, the end-block marker will be redisplayed.

CURSOR TAB—*moves the cursor to the next tab stop*

^I or **TAB** advances the cursor to the next tab stop when INSERT is OFF, or inserts spaces up to the next tab stop when INSERT is ON.

CURSOR TO PLACE MARKER—*moves cursor to indicated place marker*

^Q0 to **^Q9** moves the cursor to any of the ten place markers. Each PLACE MARKER is identified by a number. To send the cursor to a previously set place marker, type **^Q** and the number of the marker (0–9).

CURSOR FIND—*moves cursor to a specified string of characters*

^QF

moves the cursor to a specified word or phrase. The word or phrase is requested through a prompt. The cursor moves to the first occurrence of the specified word or phrase. **^QF** can be used with the FIND/REPLACE AGAIN command, **^L**, to find all occurrences of a given word. When you type **^QF**, WordStar asks for the word to be located using the prompt

```
FIND?
```

This question will appear below the menu, moving the top of the file display area down one line. Respond by typing any sequence of characters that you wish to locate, then press **RETURN**. (This function is further discussed in the FIND command.)

Send the cursor to the position it occupied before the last FIND or REPLACE by using the CURSOR-TO-LAST-FIND command, **^QV**.

CURSOR TO PREVIOUS POSITION—*moves cursor to previous position*

^QP

moves the cursor to where it was when the last command was issued. The **^QP** command is frequently used to continue editing after saving a file with **^KS**. This command is also used following a paragraph REFORM, **^B**, to return to the position where you were making editing changes.

CURSOR TO SOURCE—*moves cursor to source of last BLOCK or FIND operation*

^QV moves the cursor to the position it occupied before the last FIND or REPLACE operation, or to the source of the last block operation.

Deletions

DELETE A CHARACTER RIGHT—*eliminates characters at the cursor*

^G deletes one character at the cursor position. Text, spaces, and carriage returns are deleted. Characters to the right of the deletion are drawn to the cursor position and replace the deleted characters.

DELETE A CHARACTER LEFT—*eliminates characters to the left*

^_ deletes one character to the left of the cursor. When the left end of the line is encountered, the cursor eliminates the carriage return and starts deleting on the next line up.

DELETE A WORD RIGHT—*eliminates a word from the right*

^T

deletes a word or portion of word to the right of the cursor. If the cursor is in the middle of a word, the part of the word to the right of the cursor is deleted.

You can delete spaces between words by placing the cursor between the characters you wish to join and pressing **^T**. If the cursor is set at the end of a line, the carriage return and any following spaces will be deleted.

DELETE TO BEGINNING OF LINE—*deletes to beginning of line*

^Q ^-

deletes all characters from the cursor position leftward to the beginning of the line. This command, however, does not delete the carriage return at the end of the line.

DELETE TO END OF LINE—*deletes to end of line*

^QY

deletes all characters to the end of the line from where the cursor is positioned. Carriage returns and any overprint lines will not be deleted at the end of this line.

DELETE A LINE—*eliminates the current cursor line*

^Y deletes the entire line containing the cursor. The line below moves up and takes the place of the deleted line. Screen continuation lines and associated overprint lines are also deleted.

DELETE A BLOCK—*eliminates the currently marked block*

^KY deletes the currently marked block of text. (See BLOCK DELETE.)

DELETE A FILE—*deletes a specified file*

^KJ (or Y from the No-File menu) asks for the name of the file to be deleted and then deletes the specified file.

Display Commands

DISPLAY PAGE BREAK—*toggles appearance of page break line*

^OP hides or displays the line used to illustrate where one page ends and the next begins. It also changes the status line to display the number of the cursor character (FC=cursor character number from beginning of file) and the line number (FL=line number from beginning of document). The page break line is ON until the **^OP** command is issued. The current status of the page break line is displayed on the **^O** prefix menu.

DISPLAY PRINT-CONTROL CHARACTERS—*toggles appearance of print-control characters*

^OD hides or displays print-control characters. Print-control characters are initially displayed while you're entering them into text; however, the **^OD** command can conceal them from the screen display and thus show how your text should look when it is printed. You can examine the current status of the print-control feature on the **^O** prefix menu.

DISPLAY RULER LINE—*toggles appearance of ruler line*

^OT displays or hides the ruler line. The ruler line is normally displayed until hidden.

DISPLAY DIRECTORY—*toggles appearance of file directory*

^KF (or F from the No-File menu) hides or displays a directory listing all files contained on the diskette in the currently logged drive. The prefix menu indicates whether the directory display is ON or OFF.

File Manipulations

You can enter a file or exit WordStar from the No-File menu. The No-File menu also provides options that allow you to copy, delete, read, or rename files using the appropriate command. You can also undertake these file manipulations from within an open file. To manipulate files between drives, append a drive identifier followed by a colon to the file name. You can toggle the directory of the currently active drive ON or OFF by using the FILE DIRECTORY command **^KF**, or temporarily display it during a file manipulation by pressing **^F** after initially issuing a file manipulation command. Following are the commands used to manipulate files:

EDIT A DOCUMENT—*creates or retrieves a file*

D creates a new file or opens an existing file for editing. A file name is requested. If the specified file does not exist, then a new file is created. If a file with the specified name exists, it is retrieved so you can edit it. The file may be on another drive, in which case the drive letter and a colon must precede the file name.

EDIT A NONDOCUMENT—*creates or retrieves a nontext file*

N creates or retrieves a nontext file for editing. Dynamic pagination is disabled, and a different set of defaults is in effect. This command is typically used to prepare input for other text formatters, to enter data for application programs, or to edit program source files. Do not use the **N** command for general word processing.

NOTE

Programmers should not reform (^B) the contents of nondocument program files.

EDIT ABANDON—*closes file without saving current version*

^KQ abandons the file being created or edited without saving a copy. A backup copy will remain only if the file is being edited after previously being created. If the file is just being created, no copy of it will exist following this command.

EXIT TO SYSTEM—*relinquishes control to CP/M*

^KX leaves WordStar and returns control to the CP/M Plus operating system. When you issue **^KX**, the current file is saved before leaving WordStar. (Similar to the No-File command **X**.)

FILE COPY—*copies a named file*

^KO (or O from the No-File menu) copies a file from its source to a specified destination. Prompts ask for the name of the file to copy and the destination where the copy is to be sent. You can copy from one drive to another by preceding the file name with the letter and colon of the drives involved. (Performs the same function as the CP/M program PIP.COM.)

When you type **^KO** or **O**, the following prompts occur:

```
NAME OF FILE TO COPY FROM?
NAME OF FILE TO COPY TO?
```

Enter the name of the file that you wish to copy and press **RETURN**. Next, enter the name of the file where the copy is to be transferred. To make copies from one drive to another, add the appropriate drive identifier and a colon to the front of the file name, (e.g., **A:copyfrom, B:copyto**).

FILE DELETE—*erases a named file from the directory*

^KJ (or Y from the No-File menu) erases a named file from the current file directory. A drive identifier specifies a file on the inactive drive. This command performs the same function as the CP/M command ERA (also see DELETE A FILE).

FILE READ—*reads a named file into the currently open file*

^KR transfers and inserts the contents of a specified file to the cursor position of the file being created or edited. A file name is requested by WordStar as follows:

```
NAME OF FILE TO READ?
```

The contents of the indicated file is inserted into the current file at the cursor position.

FILE RENAME—*renames a disk file*

^KE (or E from the No-File menu) asks for the name of the file to rename and then assigns the new name to the specified file. These prompts occur:

```
NAME OF FILE TO RENAME?
NEW NAME?
```

To change the name of a file, supply its name, and press **RETURN**. Then enter the name with which you want the file identified in the future. This command performs the same function as the CP/M command REN.

Find Functions

The FIND command, **^QF**, moves the cursor to a given word or phrase within the file. The FIND, REPLACE command, **^QA**, locates a specific word or phrase and replaces it with another. After the desired word or phrase has been located and/or replaced, you may proceed to the next occurrence of the word or phrase by issuing the FIND/REPLACE AGAIN command, **^L**.

FIND—*locates a given word or phrase*

^QF finds the first occurrence of a specified word or phrase. A prompt asks for the word or phrase to search for:

```
FIND?
```

Reply by typing the word or phrase you wish to locate, then press **RETURN**. The cursor moves to the indicated word or phrase. Certain options are defined in the FIND REPLACE command below.

FIND, REPLACE—*locates and replaces a given string with another*

^QA finds the first occurrence of a specified word or phrase and replaces it with another. Prompts ask for the word or phrase you're searching for, as described above. After you have entered the string that you wish to locate and have pressed **RETURN**, WordStar asks:

```
REPLACE WITH?
```

Respond by entering the replacement characters and pressing **RETURN**. WordStar then asks:

```
OPTIONS (? FOR INFO)
```

The "OPTIONS" question allows you to specify certain options, such as matching whole words only, ignoring the distinction between uppercase and lowercase letters, or searching backwards instead of forwards. A question mark (?) will display a list of FIND options; use it to help refresh your memory. You can ignore the "OPTIONS" question by pressing **RETURN**, or you can answer it with one or more of the following option codes:

> #, when specified with the FIND command, locates the #th occurrence of the specified word or phrase. Used in the FIND, REPLACE command, it locates and replaces the specified word or phrase that number of times.
>
> G, when used with the FIND, REPLACE command, replaces every occurrence of the specified word or phrase from the cursor position to the end of the file. A (Y/N) prompt allows selection of each replacement as it appears. When used with the FIND command, it will search for the last occurrence.

N replaces words or phrases without asking the (Y/N) question.

U causes the find or replacement operation to ignore the distinction between uppercase and lowercase letters.

W matches only whole words during a FIND or REPLACE.

^P (control character) Control characters can be entered in response to the OPTIONS question. Some specialized ones follow:

B Instead of the search progressing towards the end of the file, as it usually does, this option causes the file to be searched backwards to the beginning of the file.

A matches any single character.
S matches any character other than a letter or digit.
O precedes a character so that a match will occur with any character but the one following **^O**.
N causes a match to be made with a carriage return or line feed.

NOTE

You can eliminate the "OPTIONS" question by pressing **ESC** *following either the "FIND" or "REPLACE" questions.*

After you've issued the FIND, REPLACE command (**^QA**), WordStar will search for the word or phrase to replace. On finding the word or phrase, WordStar will display the following prompt at the upper right side of the screen:

```
REPLACE (Y/N)?
```

The cursor flashes on and off to indicate that a decision is required. If you want to replace the word or phrase that is located, press **Y** for YES. If you do not wish to replace that particular occurrence of the word or phrase, press any other key. To repeat the most recent REPLACE command from the current cursor position, use the abbreviated FIND/REPLACE AGAIN command, **^L**.

NOTE

You may use the INTERRUPT command, **^U**, *to stop a FIND or REPLACE operation while it is in progress.*

FIND/REPLACE AGAIN— *continues a previous FIND or REPLACE function*

^L repeats the most recent FIND or REPLACE command and supplies identical responses for the options.

Flag Characters

All the columns on the screen, except the rightmost column, are available for text. This column is reserved for FLAG characters that indicate the status of text on the current file line as follows:

(blank) in the last column indicates that the line ends with a soft carriage return. This condition may be changed following a WORD WRAP or REFORM operation.

< indicates that the line ends with a hard carriage return. WORD WRAP or REFORM operations do not change this line break.

- means that the following line will be printed over the current line. This PRINT function creates special effects.

. indicates that the current screen line is below the existing text. This character will also appear at the end of the last text line if there is not a carriage return at the end of this line.

: appears if the current text line is above or before the beginning of the document.

+ indicates that the next screen line is a continuation of the initial line.

P appears only when the PAGE BREAK display is on: it indicates that a new page begins with the next line.

? is displayed when a line contains an unrecognized or possibly erroneous DOT command. It also appears while a DOT command line is being typed. You may ignore this character until entry is complete.

J indicates that the line ends in a line feed without a carriage return. This format is nonstandard and is never created during normal WordStar usage.

M indicates a line contains a MERGE-PRINT DOT command.

HELP Commands

HELP menus are shown at the top of the screen. You can eliminate them by degree as you become more adept at using the system. Also, WordStar provides further information to assist you in learning; certain commands display detailed information about the more involved WordStar functions. The ^J prefix menu explains the following list of subjects:

^JD explains print directives such as DOT commands and PRINT controls.

^JI explains command index for entering text.

^JM explains margins, line spacing, justification, and tabs.

^JB explains paragraph reform.

^JP explains place markers.

^JV explains moving text.

^JR explains the ruler line.

^JS explains the status line.

^JF explains flag characters.

HELP SET— *establishes the level of assistance displayed on the screen*

^JH, or H from the No-File menu, displays the current help level and requests a new setting. You can set the help level between 0, the least amount, or 3, the greatest amount of help. The amount of information displayed on the help menus corresponds to the help-level setting.

After you type the HELP SET command, a description of help levels and the current help-level setting are displayed. The display requests a new help-level setting.

As you gain experience using WordStar, you may reduce the level of assistance to coincide with your experience and ability. Help level 0 provides the least assistance and gives you the most screen area for file display.

Hyphens

WordStar has two kinds of hyphens: the soft hyphen, which indicates a syllable break, and the hard hyphen, which separates words or phrases.

A soft hyphen at the end of a text line separates a word that is too long to fit on the current line. The hyphen divides the word into syllables and continues it on the next line. The soft hyphen prints only if the divided word appears at the end of a line; if the word ends up in another position following some operation such as a REFORM, the hyphen will not be printed.

Generally, you enter soft hyphens by using the HYPHEN HELP feature, but you may enter them explicitly by turning ON the SOFT-HYPHEN ENTRY. Soft hyphens that you type when the SOFT-HYPHEN ENTRY is ON will divide a word only if it appears at the end of a line.

Hard hyphens, on the other hand, are used whenever a fixed divider is required between characters, strings, or phrases. A hard hyphen will always be printed, no matter where it appears in the text. A hard hyphen is entered automatically if the SOFT-HYPHEN ENTRY is OFF and HYPHEN HELP is not engaged. **^P-** unconditionally enters a hard hyphen.

To distinguish soft from hard hyphens, type **^OD**, which turns the print-control display ON and OFF. When this display is OFF, soft hyphens will not show up in the file document.

HYPHEN HELP— *toggles HYPHEN HELP feature ON or OFF*

^OH

turns HYPHEN HELP ON or OFF. When ON, the PARAGRAPH REFORM process pauses and positions the cursor at each instance where a word can be hyphenated. The **^O** prefix menu displays whether HYPHEN HELP is ON.

HYPHEN HELP checks that the word contains two syllables, and selects the proper position for the hyphen. You can then decide if the word should or should not be hyphenated at the selected position.

When using HYPHEN HELP, make sure that margins, line spacing, and justification are properly selected. Place the cursor at the beginning of a paragraph and press **^B**. When the hyphen position is located, the following message is displayed:

```
TO HYPHENATE, PRESS -. Before
pressing the -, you may move the cursor:
^ S = cursor left. ^ D = cursor right.
```

You can enter a hyphen at the suggested location, or you can move the cursor to the position where you want the hyphen to appear. If you do not want to hyphenate the word at all, press **^B** and the REFORM process will continue down the text.

SOFT-HYPHEN ENTRY— *toggles interpretation of hyphens*

^OE turns SOFT-HYPHEN ENTRY ON or OFF. When ON, hyphens are temporary and will not be printed unless they fall at the end of a line. Soft hyphens are highlighted, but the **^OD** command causes only hyphens that will be printed to be displayed in half intensity.

Interrupt Execution

INTERRUPT— *stops command execution*

^U stops any commands currently in progress. You can also enter this command in response to prompt questions, such as "FIND?", to abort the command making the request. When you press **^U**, the following message is displayed:

```
* * * INTERRUPTED * * *
```

All commands issued before the INTERRUPT are aborted and must be reentered if needed.

Insertions

INSERTION— *toggles character insertion ON and OFF*

^V toggles the INSERTION function to either insert text to the left of the cursor position or replace text at the current cursor position.

When INSERTION is ON, it inserts typed characters to the left of the cursor while the cursor moves text to the right. When INSERTION is OFF, typed characters replace those at the cursor position. The status line displays the current state of insertion.

You can determine if INSERTION is ON or OFF by looking at the STATUS LINE at the top of the display for the words INSERT ON.

INSERT CARRIAGE RETURN— *establishes a fixed carriage return*

^N
or ^M

insert fixed carriage returns. Any text to the right of the cursor moves to the beginning of the next line. Neither WORD WRAP nor any other operation can alter a hard carriage return. A hard carriage return will always appear in the printed version of a document.

The difference between these two commands is that **^M** moves the cursor with the text, whereas **^N** leaves the cursor in its current position.

Layout

You can format a document in many ways using the WordStar layout commands. These commands allow you to control the way text is displayed. All the following commands can affect already existing text when they're used with the PARAGRAPH REFORM command:

REFORM PARAGRAPH— *reorganizes a paragraph with new specifications*

^B reforms the paragraph below the cursor so that words are spaced evenly after editing. You can also use the REFORM command to change margins or line spacing, justify or unjustify text, or assist in hyphenation.

JUSTIFICATION— *toggles interpretation of text alignment*

^OJ when ON, aligns text with the right margin. When JUSTIFICATION is OFF, lines of text end in various columns. The **^O** prefix menu displays the current state of justification. You determine if this function is ON or OFF by pressing **^O** and looking at the menu.

WORD WRAP— *toggles carriage return requirements between lines*

^OW turns the WORD WRAP feature ON or OFF. This feature, which is normally ON, allows entry of text without the need for carriage returns except between paragraphs. Turn WORD WRAP OFF if you want to terminate every line with a carriage return as you would on an average typewriter.

CENTER CURSOR LINE— *centers current line between margins*

^OC centers the line containing the cursor within the margins. This command is generally used to center headings. To use this command, place the cursor anywhere on the line you wish to center and type **^OC**. This command deletes any spaces and tabs set at the beginning of the line, then enters the appropriate number of hard spaces needed to center the line.

LINE SPACING— *sets the line spacing*

^OS sets the number of blank lines that separate text lines. When you type **^OS**, WordStar will request a number between 1 and 9 that represents the number of carriage returns placed between text lines.

The LINE SPACING command also determines the number of line advances following every RETURN. You can change line spacing at any time by entering **^OS**, and reformatting the document (using the REFORM command).

RULER LINE— *toggles display of the ruler line*

^OT toggles the display of the RULER LINE below the menu. This dotted line shows the margin and tab formatting that is in effect. Left and right margins are illustrated by an "L" and "R". Exclamation marks indicate variable tab stops. Decimal tab stops are shown as # signs.

When a margin is set at a tab stop, the tab symbol is displayed. If the margin is temporarily moved in with the PARAGRAPH TAB command, **^OG**, the ruler display will show the extent of the temporary margin.

Tabs set outside the margins are not displayed until the margins are released or WORD WRAP is OFF. When the margin is set larger than the screen display, the RULER LINE doubles to show the current margin setting. You can hide the RULER LINE by typing the DISPLAY/HIDE RULER LINE command **^OT**. When the RULER LINE is hidden, a line of S's is displayed to separate the file directory from the file document.

You can specify margins in a text file by using the RULER LINE to identify where to set the margins. To enter a RULER LINE that will set the margins, type a line into the document with an exclamation point (!) at each column where a tab should be set, a number sign (#) at each column where a decimal tab should be set, and a hard hyphen (-) in every other column between the desired left and right margins. You can then enter this RULER LINE into the document by placing the cursor anywhere in the line and typing the MARGINS FROM FILE LINE command, **^OF**.

When you have entered this line, the desired tabs and margins will be set while all other default settings are cleared. The RULER LINE may be kept from appearing in the printed document by preceding it with two periods (see PRINT DOT COMMENT command).

NOTE

The **^O** *menu shows whether the following features are currently ON or OFF:*

HYPHEN HELP
VARIABLE TABBING
PAGE BREAK DISPLAY
WORD WRAP
JUSTIFICATION
PRINT-CONTROL DISPLAY
SOFT-HYPHEN ENTRY
RULER DISPLAY

Margin Arrangement

MARGIN LEFT— *establishes the left margin*

^OL sets the left margin between column 1 and column 240. You can specify the left margin setting by entering the column number of the new margin, or pressing ESC to set it at the cursor column. The current column number is displayed on the STATUS LINE. To set the left margin at the cursor position, press ESC following the **^OL** command.

MARGIN RIGHT— *establishes right margin*

^OR sets the right margin. To answer the request for a right margin, you may enter a column number, or press ESC to specify the current cursor position.

NOTE

You may change margins at any time by reforming the existing text with the new margin settings.

MARGINS FROM FILE LINE—*mimics existing margins*

^OF sets the margins to match those of the existing document. To set the margins with this command, place the cursor anywhere in the existing line and type **^OF**. The margins automatically set to the width of the current file line.

MARGIN RELEASE— *disengages existing margins*

^OX temporarily disengages the current margin settings. Text entered following the **^OX** command can extend beyond the established margins. The margins remain released until the cursor returns within the bounds of the original margin setting. MAR REL appears on the STATUS LINE while the margins are released. You can reset the margins with the same **^OX**.

Place Markers

PLACE MARKERS—*mark a position for later reference*

^K0–^K9

mark a position within the text where the cursor may be sent. There are ten PLACE MARKERS (0–9). You can set any of these markers within the file and subsequently reference them. The numbered marker will show up at the specified position, but it is not actually part of the document. Each numbered marker may be returned to by using the CURSOR TO MARKER command, **^Q**, followed by the number of the marker (0–9).

You can hide a PLACE MARKER by moving the cursor to the desired location and issuing the same command used to set the marker. In other words, this command acts as a toggle that alternately sets or hides the marker. Though hidden, a set marker is still in effect and will be redisplayed after it is accessed.

Print-Control Characters

You can insert control characters into a text file—by typing **^P**, followed by the print-control character—to control printing.

Displayed print-control characters tend to distort text. However, editing commands ignore print-control characters, and they are not printed.

Print-Control Toggle Commands

Toggle-control characters are those that must be placed on both sides of the affected text. The first toggle character initiates a control effect and the second toggle character terminates the effect. Listed here are the toggle commands:

STRIKEOUT TOGGLE

^PX prints dashes over the specified characters. Use this command to illustrate deleted text in the revised version of a document.

SUBSCRIPT TOGGLE

^PV prints the enclosed characters as subscripts. The subscripted characters will be positioned below the surrounding text. Determine the degree of subscript using the DOT command **.SR**. On printers without fractional lines, the next line must be blank.

SUPERSCRIPT TOGGLE

^PT prints enclosed characters as superscripts so they will appear slightly higher than the surrounding text.

BOLDFACE TOGGLE

^PB offsets slightly and overstrikes on daisywheel printers or any other printer capable of incremental motion. **^PB** multistrikes each character on teletype printers.

DOUBLE-STRIKE TOGGLE

^PD strikes each character twice with no offset. This command produces a lighter version of "boldface." This control character, used with a carbon ribbon, produces an extremely sharp impression of the entire document.

UNDERSCORE TOGGLE

^PS is placed on both sides of the section of text you want to underline. Only nonblank characters are underlined.

Other Print-Control Commands

Following are the control-character commands that control the printer:

LEFT/RIGHT, HEADING/FOOTING CONTROL

^PK

is used with heading and footing DOT commands to produce headings, page numbers, etc., that print on the left-hand side of even-numbered pages and on the right-hand side of odd-numbered pages. **^PK** formats headings and page numbers so they will always appear on the side of the page farthest from the binding in loose-leaf binders. Use this print-control character with the DOT commands **.HE** and **.FO**.

ALTERNATE CHARACTER PITCH

^PA

is used with daisywheel printers to change the character width from 10 (pica) to 12 (elite) characters per inch.

STANDARD CHARACTER PITCH

^PN

selects a standard 10-characters-per-inch width (pica) on daisywheel printers.

BACKSPACE

^PH makes the next character overprint the preceding character on the line. Use it to place accent marks over letters or to create special symbols by overprinting multiple characters. This control character is placed where the backspace is desired; it may be affected when text is reformed or justified.

NONBREAK SPACE

^PO prints a space, but the space is not treated as such for line breaks or justification during line formatting.

PHANTOM RUBOUT

^PG prints the character on a daisywheel printer that is associated with code 7F hex. This code prints a "not sign," "double underline," or graphic that is associated with the hex code.

PHANTOM SPACE

^PF prints the character, normally a space code, associated with hex code 20 on daisywheel printers.

STOP PRINT

^PC halts printing. This function gives you a chance to change ribbons or type fonts. You can use this control character within a line as often as needed. When printing stops, the prompt **PRINT PAUSED** appears on the status line. You can restart the printer by typing the PRINT command **^KP** (or **P**).

TAB

^PI displays and prints spaces to advance to the next multiple of eight columns; normally you don't enter it into the text except in VARIABLE TAB mode.

USER PRINT FUNCTION (1–4)

^PQ, ^PW, ^PE, ^PR are USER PRINT FUNCTIONS for special printer operations that WordStar does not otherwise perform. You have to establish each of these functions when you install WordStar. Each function can send a sequence of one to four characters to the printer.

FORM FEED

^PL causes a form feed to be entered into the text.

CARRIAGE RETURN

^PM is the same as a carriage return. Causes the current line to print over the preceding line. The flag character (-) appears in the rightmost column to indicate an overprint.

LINE FEED

^PJ is the same as line feed.

Print DOT Commands

The DOT commands are special characters you embed in text to control the final format of the printed text. DOT commands alter default formats. DOT commands are displayed but they are not printed.

A DOT command consists of a period in the first column of a line, a two-letter code, and optionally a number or some other argument. When you enter a period in the first column of a line, WordStar expects a DOT command and the left margin temporarily disengages; this is indicated by a ? prompt on the current screen line.

Most DOT commands may be placed anywhere in a text file, but the dynamic page break display requires that certain DOT commands appear at the beginning of a file.

The following five sections describe the various DOT commands:

Vertical Page Layout

The following vertical-page-layout DOT commands have to appear at the beginning of the file for page breaks to correctly interpret them.

LINE HEIGHT

.LH*n* sets the line height in 1/48ths of an inch on daisywheel printers and provides an alternative or supplement to the single, double, or triple spacing the PRINT CONTROL command **^OS** gives. Don't use this DOT command on printers that can't print incrementally. The default is 6 lines per inch.

PAPER LENGTH

.PL*n* determines the number of lines per page, including the top and bottom margins. The paper length must match the forms specification. The default is 66 lines.

TOP MARGIN

.MT*n* specifies the number of lines from the top of the paper to the beginning of the text. The default is 3 lines from the top.

BOTTOM MARGIN

.MB*n* specifies the number of lines not to be used for text at the bottom of the page. The page number or footing, if present, is printed within the bottom margin. The default is 8 lines.

HEADING MARGIN

.HM*n* determines the number of blank lines that will appear between a page heading and the body of the text.

FOOTING MARGIN

.FM*n* sets the number of lines between the last text line of the page and the page number or footing. The default for the HEADING and FOOTING margins is 2 lines.

Horizontal Page Layout

Most horizontal formatting is an integral part of text editing and does not involve DOTcommands. However, the following DOT commands cover special print operations.

PAGE NUMBER COLUMN

.PC*n* determines the column at which the page number is printed when neither the **.FO** or the **.OP** command is in effect. You can place the page number to the left or right of a page, but you must take the current character pitch into account.

PAGE OFFSET

.PO*n* sets the number of columns that the entire document will be indented from the printer's left margin, to offset text from the tractor-feed holes at the left of the paper. This feature allows you to load narrow paper near the center of wide printer carriages. The default is 8 columns.

Pagination

PAGE

.PA starts a new page unconditionally.

CONDITIONAL PAGE

.CP*n* starts a new page if there are less than *n* lines left on the current page. This keeps blocks of text together and suppresses pagination after a title, in the middle of a table, etc.

Page Heading, Footing, and Page Number

TEXT HEADING

.HE begins a line that will serve as a heading for each page until another heading is specified. Headings may be changed as often as necessary. To print a heading on the first page, put an **.HE** command in front of all text in the file.

TEXT FOOTING

.FO The rest of a line beginning with this command serves as a page footing for the current and following pages of a document. A document may contain numerous footing commands. Text specified in the most recently encountered footing command is printed.

If no footing is specified, or if an **.FO** command has no text following it, page numbers will be printed in the footing line at the column specified by the PAGE NUMBER COLUMN command, **.PC**. Page numbers are not automatically printed when a TEXT FOOTING DOT command is in effect. You can place a # symbol at the location where you wish a page number to appear.

The following 3 characters have special meaning within TEXT HEADING and FOOTING commands:

prints the current page number. Use it to position pag:e numbers wherever you want, at the top or bottom of the page.

\ prints the next character without special interpretation so that control characters may be printed as text.

^K is used with the **.HE** or **.FO** DOT commands. This control character directs printing to format a heading or page number depending on whether a page number is odd or even. All spaces following the **^K** character are ignored if the page number is even so that the heading or page number will be printed on the right-hand side of odd-numbered pages and on the left-hand side of even-numbered pages. Use this feature if your document will be printed on both sides.

OMIT PAGE NUMBERS

.OP suppresses printing of page numbers when no footing has been given. This DOT command has no effect if footing has been specified.

NUMBER PAGES

.PN turns page numbering back ON following a previous **.OP** command that turned it OFF. Page numbering will begin with the number 1, unless otherwise specified. (See **.PN *n*** below.)

PAGE NUMBER

.PN*n* turns page numbering back ON following a previously issued **.OP** DOT command. Page numbering will begin with the number you specify after **.PN**. The page numbers are printed at the bottom of the page unless a # character specifies otherwise.

Miscellaneous DOT Commands

CHARACTER WIDTH

.CW*n* sets the character width in increments of 1/120ths of an inch. The default standard pitch is 10 characters to the inch, and the default alternate pitch is 12 to the inch. This command only works with printers that have programmable character widths. CW 12 is the default.

SUB/SUPERSCRIPT ROLL

.SR*n* specifies the amount in 1/48ths of an inch that the carriage rolls before printing a superscripted or subscripted word. The default is 3/48ths of an inch or .SR3.

ON (1) OR OFF (0) MICROJUSTIFICATION

.UJ turns off microjustification which is normally ON. Microjustification spreads words evenly by adding soft spaces. When microjustification is OFF, the text will be printed as it appears on the display with soft spaces and returns. Turning microjustification OFF may be useful to make a columnar table print with the columns aligned as they appear on the screen, even with soft spaces inadvertently supplied by WORD WRAP or REFORM.

ON (1) OR OFF (0) BIDIRECTIONAL PRINT

.BP either enables or prevents the printer from printing back and forth across the page. Use it when you have a problem with the printer.

IGNORE TEXT

.IG allows display, but not printing, of commentary text in a file line.

After using the appropriate print-control characters and DOT commands to direct the printer, use the print command described below to print your file.

Printing a File

PRINT A FILE—*toggles printing of a file ON and OFF*

^KP (or P from the No-File menu) outputs the contents of a file to the printer. The same **^KP** or **P** commands halt the print operation. The current status of this command is displayed in the **^K** prefix menu.

These PRINTA FILE commands toggle printing so that, when first issued, the name of the file to be printed is requested. When this file name has been supplied, several print option questions are asked. You may simply press **RETURN** in response to each question if you wish to use the default settings, or press **ESC** to prevent the options from being offered. These option questions are:

DISK FILE OUTPUT (Y/N) sends the file to a diskette.

START AT PAGE NUMBER (RETURN for beginning) lets you specify a page number where printing should begin.

STOP AFTER PAGE NUMBER (RETURN for end) asks for a page number where printing should stop.

USE FORM FEED (Y/N) outputs form-feed controls to the printer.

SUPPRESS PAGE FORMATTING (Y/N) sends an exact replica of the screen display, including DOT commands, to the printer if you answer YES. No formatting is performed on the text before it is printed when page formatting is suppressed.

PAUSE FOR PAPER CHANGE BETWEEN PAGES (Y/N) stops the printer at the end of each page if you answer with YES. This option allows you to change paper between pages.

Ready printer, press RETURN indicates that text is ready to be printed. The specified file will output to the printer when you press **RETURN**.

To halt printing, reissue the **^KP** (or **P**) command, and the following messages appear:

```
TYPE Y TO ABANDON PRINT,
N TO CONTINUE, ^ U TO HALT
```

If you respond with **Y**, the print operation will stop. Pressing **N** causes the print operation to resume immediately. Pressing **^U** temporarily suspends printing; subsequently you can use the **^KP** (or **P**) command to resume printing. The STATUS LINE will display the prompt message PRINT PAUSED while printing is suspended.

Prompt-Question Control Characters

Special control characters entered in response to prompt questions perform specific functions. These control characters follow:

^X or ^Y erases the entire answer in order to enter another answer (no file menu).

^S, ^H, or [←] erase one character to the left (no file menu).

^D moves the cursor to the right and displays any previously erased character. This function will also display the character that was entered the last time that the question was asked (no file menu).

^R restores the erased answer, or the answer you entered the last time you received the current question (no file menu).

^F displays a file directory of the currently logged disk drive for the duration of the command (main menu, with **^K**).

^Z or ^W scrolls the file directory up (**^Z**) or down (**^W**) to bring additional files into view (main menu, with **^K**).

- enters a SOFT HYPHEN if the SOFT-HYPHEN ENTRY has been turned ON. Also permits using SOFT HYPHENS in response to a "FIND?" question prompt. (Use **^P** to search for a hard hyphen.)

^N, ^S, ^A, or ^O have special meaning only with the "FIND" question prompts. (See the FIND FUNCTION.)

^U interrupts and terminates the command in progress. (See the INTERRUPT command.)

Repeat a Command

REPEAT NEXT COMMAND—*repeatedly executes a command*

^QQ

(command) causes the command following it to be executed repeatedly until you press the SPACE bar. You can specify a rate of execution between 0 (slowest) and 9 (fastest). First type the command **^QQ**, then type the command that is to be repeated.

Run a Program

RUN A PROGRAM—*runs a program from the No-File menu*

R

runs a specified program. Enter the name of the program in response to a prompt, and the indicated program executes. You can examine the amount of disk space, run a spelling program, etc. Pressing **R** displays the following prompt:

```
COMMAND?
```

Answer this prompt question by entering the name of the program you want to run (i.e., SHOW) and then press **RETURN**. Precede the program file name with a drive identifier if it exists on a drive other than is currently logged.

Save Procedures

You can transfer a file to a disk and save it by using one of the following SAVE commands.

SAVE DONE—*saves file and returns to the No-File menu*

^KD saves the file currently being created or edited, then transfers to the No-File menu.

SAVE REEDIT—*saves file without closing it*

^KS saves the current file, then returns to the beginning of the saved file. This command should be used periodically to protect your file from accidental loss by saving it in increments. After the file has been saved, editing of the same file can continue.

The cursor will go to the beginning of the saved file, but you may return it to the position it occupied when you saved the file with the CURSOR PREVIOUS POSITION command, **^QP**. Also, the SAVE REEDITcommand is the fastest method of moving long distances to the beginning of a file.

SAVE EXIT—*saves file and exits to CP/M*

^KX saves the current file and exits from WordStar to CP/M. This command performs the same function as the EXIT TO SYSTEM command, **X**, from the No-File menu, except that **^KX** saves the file before leaving WordStar.

Scroll Commands

The SCROLL commands, which follow, move the screen display down or up a line, or move the entire display down or up one screen's distance.

SCROLL DOWN LINE—*scrolls display down a line*

^W moves the entire screen display down one line.

SCROLL DOWN SCREEN—*scrolls downward a screen's distance*

^R moves displayed text down one screen's distance so the text above is displayed. This command causes the portion of the file that is being displayed to move upward and disappear while text below takes its place.

SCROLL UP LINE—*scrolls up a line*

^Z moves the screen display up one line.

SCROLL UP SCREEN—*scrolls upward a screen's distance*

^C causes text currently occupying the screen to be replaced by an equal amount of text below it.

CONTINUOUS SCROLL—*scrolls upward or downward till stopped*

QW or **QZ** scrolls the display continuously up or down.

NOTE

REPEAT NEXT COMMAND, ^QQ, may be placed before any of the scroll commands to move the display continuously in either direction (see REPEAT NEXT COMMAND).

Status Line

The STATUS LINE at the top of the screen displays various information pertinent to the file being edited. The information displayed from left to right on the STATUS LINE is: the currently active drive, a colon, the name of the file being edited, the number of the page being edited, the line number, and the column number where the cursor is positioned.

When you issue a command, it will be displayed at the upper far left of the STATUS LINE until the command has been executed. If the INSERTION function is ON, it will be indicated at the upper far right of the STATUS LINE. (See INSERTION.)

Certain messages are displayed on the status line when appropriate. These messages are:

> **WAIT** is displayed when a file is being read or written. No data should be entered while this message is displayed.
>
> **MAR REL** indicates that margins have been released.
>
> **LINE SPACING *n*** shows what line spacing is in effect, unless it is set to the default of one line per inch.
>
> **PRINT PAUSED** is displayed when printing is suspended.
>
> **REPLACE** is displayed when the FIND REPLACE function is in effect.

Tab Arrangement

TAB SET—*establishes tab stops*

^O TAB or

^OI

sets a tab stop anywhere on a line to format a document or arrange columnar data. To set the tab, type the SET TAB STOP commands, **^O and TAB** or **^OI**, and a prompt question will ask for the column number where the tab should be set.

Typing a column number will set the tab at the specified column. Pressing **ESC** will set the tab at the cursor column, indicated on the STATUS LINE. All tab stops that are in effect are displayed on the RULER LINE as an exclamation point (!).

NOTE

Decimal tabs will align columns of numbers on the decimal point, or right align text on a tab stop. After you tab to a decimal tab stop, characters entered move to the left, pushing the entire field to the left of the decimal tab setting. The cursor will remain at the tab position. This right alignment function may be terminated at any time by typing a period.

Decimal tabs may be set by using either of the methods for setting tab stops. When using the **^OI** command, you type a # sign before entering the column number or pressing **ESC**. The decimal tab may also be set in the RULER LINE by placing a # sign instead of a **!** sign at the column where you want the decimal tab stop.

Decimal tabbing is only active when variable tabbing is ON. You may determine if the VARIABLE TABBING is ON or OFF by pressing **^O** and examining the menu.

TABBING VARIABLE—*toggles from fixed to variable tabs*

^OV turns variable tabbing ON or OFF. When ON, variable tab stops are in effect and spaces are entered into the files for tabs. When OFF, fixed tabs are in effect. Fixed tabs are not usually used during standard word-processing operations; this feature is used when writing computer programs. When variable tabbing is OFF, tab characters **^I** (09 hex) are used in the file and are displayed with fixed stops every 8 columns; multiple spaces are entered into the file when using variable tabbing. Variable tabbing should be turned OFF when programs are being developed under the CP/M text editor or Micropro Wordmaster.

When the variable-tabbing mode is turned OFF, each tab is a single character that edits differently than those used when the variable-tabbing mode is ON. The cursor cannot be placed within the white space on the screen, representing the TAB; the cursor advances over the tab. Text that is inserted before a tab will appear in front of the tab until enough text has been entered to force the text to move to the next tab position.

TAB ADVANCEMENT—*moves to the next tab stop*

^I or TAB

advances the cursor to the next tab stop. If no tab stops are encountered on a line, the cursor advances to the first tab on the following line. Only tab stops set within the current margins are used unless WORD WRAP is OFF or the margins have been released.

When INSERTION is ON, **^I** or **TAB** inserts spaces to the tab stop and positions the text to the right of the tab stop.

When INSERTION is OFF, the TAB command will advance the cursor over the existing text. A document line will be extended with spaces or a hard carriage return as an advance to the next tab stop occurs.

TAB PARAGRAPH—*temporarily relocates the margin to the next tab*

^OG

temporarily sets the left margin in one tab stop from its present setting. This command indents a paragraph or other section of text. This temporary margin setting will remain in effect until you press **RETURN**, issue another margin command, or move the cursor above (before) the indented text. **^OG** commands issued in succession further indent text.

TAB CLEAR—*disengages specified tab stops*

^ON clears the tab at a specified, prompted location. You can release all tabs by typing **A**, then pressing **RETURN** in response to the prompt.

MailMerge DOT Commands

MERGE-PRINT—*initiates merge-printing of files*

M initiates printing in the same manner as the print command, but additionally interprets MailMerge DOT commands. You are asked for the name of the file to be merge-printed, and are provided with the following options:

```
NAME OF FILE TO MERGE-PRINT?

DISK FILE OUTPUT (Y/N)?

START AT PAGE NUMBER (RETURN for
beginning)?

STOP AFTER PAGE NUMBER (RETURN for
end)?

NUMBER OF COPIES (RETURN for 1)?

USE FORM FEEDS (Y/N)?

SUPPRESS PAGE FORMATTING (Y/N)?

PAUSE FOR PAPER CHANGES BETWEEN
PAGES (Y/N)?

Ready printer, press RETURN:
```

The various DOT commands used to control merge-print operations are described below:

DATA FILE—*specifies the data file*

.DF identifies the data file that contains information to be printed in place of keywords when the calling file is merge-printed.

Format:

.DF d:filename [CHANGE]

The named data file provides the text for keywords in the calling file. The data file will be expected on the currently active drive unless another drive is specified.

Each record in a data file contains fields of information to be supplied for one printed document, such as a form letter. A comma separates each field of information to be printed for a keyword from the next. A data record provides replacement text for all keywords in the calling file. An **.RV** command in the calling file lists keywords in the sequence that text from the data file will be assigned.

The calling file is reprocessed once for every record in the data file. It ends when data has been taken from the last data-file record. You can access only one data file at a time.

To display a prompt message asking for a diskette change, place the word **CHANGE** at the end of the **.DF** command. CHANGE, the name of the expected data file, and the drive where it is expected are all displayed when the CHANGE option appears in a **.DF** command.

Examples:

.DF DATA.FYL

specifies that text variables listed in **.RV** will come from a data file named DATA.FYL.

.DF B:DATA.FYL2 CHANGE

asks that the diskette in drive B be changed.

READ VARIABLES—*reads data for variable keywords (with .DF)*

.RV lists keywords in the order that text fields from the data file are selected and assigned to them.

Format:

.RV keyword, keyword

The sequence with which keywords are listed in the **.RV** command must correspond to the order that text fields are listed within records in the data file.

The first text field in a record line is assigned to the first keyword listed in **.RV**, and so on. Usually one **.RV** is sufficient to list all keywords in a file, but you can use multiple **.RV**s when necessary.

If a text field is missing from a record in the data file, **.RV** will use the next available text field. Extra text fields are ignored. Processing of a document always starts at the beginning of the next record line.

Example:

.RV NAME, ADDRESS, STREET, COMPANY

ASKS VARIABLES—*requests data to be assigned to keywords*

.AV

asks for data to be entered from the keyboard that will be assigned to a corresponding keyword. Each **.AV** DOT command corresponds to a keyword and an optional message. Data entered at the keyboard is printed in place of the keyword wherever it appears in the file.

Format:

.AV [''message''], variable identifier, [max-length]

The keyword and a question mark ask for data to be input from the keyboard. Optionally, a prompt message may be displayed: type the desired message within quotation marks in front of the keyword.

You can specify the maximum number of text characters an **.AV** prompt accepts. To do so, place a comma and this maximum number at the end of the **.AV** command. No more than the maximum specified will be displayed or printed.

The following CONTROL characters— **^S** (erase character), **^Y** (erase answer), and **^R** (restore previous answer for this question)—can edit keyboard entries. Press **RETURN** after you supply the appropriate data for the displayed prompt.

To halt printing while **.AV** is asking for data, enter the requested input, then press **RETURN** and **P** in quick succession. This maneuver causes the "STOP" print command, **P**, to be received before the next **.AV** is processed.

Examples:

.AV "Enter First, Last Name", NAME

.AV "City, State", ADDRESS

.AV ZIPCODE, 5

SETS VARIABLE—*establishes text for keywords*

.SV assigns a fixed piece of information to a keyword.

Format:

.SV keyword, data

Data represents text that is permanently assigned to the keyword. Data, up to 200 characters long, may include keywords, providing they have had text previously assigned. You can enter a carriage return into the data with a **^N**.

Examples:

.SV DATE, Dec 20, 1981

.SV PARTY1, John Doe

.SVADDRESS, 22334 55th St. ^NHayward, CA, 94545

.SV PARTIES, &Partyl& and &Party2&

FILE INSERT—*references a file to be inserted*

.FI inserts and processes a named file in its entirety at the point where the **.FI** command is encountered. All commands in the inserted file are processed, including those that reference further insertions.

Format:

.FI d:filename [CHANGE]

Files you insert using **.FI** should end with a carriage return to separate text in the inserted file from text in the calling file. After the inserted file has been processed, processing of the calling files continues where it left off.

Inserted files may include **.FI** commands. This process of referencing files is called nesting. Files can be nested to a maximum of eight levels. An unlimited number of files can be nested when **.FI** is the last command in each file. A keyword in the **.FI** command allows you to enter a file name from the keyboard; you must set up the calling file with the appropriate **.AV** command.

Examples:

.FI DOC.FYL

.FI B:LETTER.FYL

.FI CHAPTER &KEYWORD&

REPEAT PROCESSING—*reprocesses a file*

.RP causes a file to be reprocessed.

Format:

.RP [n]:

A file containing the **.RP** command is processed until all the text in the associated data file has been used; or until it has been processed the specified number of times. If the number specified is larger than the number of records in a data file, processing continues reusing the records of the data file.

Example:

.RP 20 <causes file to be processed 20 times>

DISPLAY MESSAGE—*displays a message on the screen*

.DM displays a message on the screen. If you don't specify a message, a blank line appears. Each message is displayed on the next free line. If the screen is full, existing messages scroll up a line to make room. Keywords can form part of the message, providing you've previously assigned text to the keywords.

Format:

.DM (message)

Examples:

.DM (produces blank line on screen)

.DM This file prints form letters

.DM Printing letter to &NAME&

.DM Load special paper and press P

.DM Insert data diskette in drive B:

CLEAR SCREEN—*clears the screen*

.CS clears all accumulated messages from the screen. An optional message can be displayed following clearance of the screen by placing the desired message after **.CS**.

Format:

.CS [message]

Messages that follow a **.CS** command are displayed on the first blank line at the top of the screen. When the screen fills with messages, the entire display scrolls up one line to make room for the next message. You can place references to keywords in the message, providing you've previously defined data for the keywords.

Examples:

.CS clears message area of screen

.CS Press **RETURN**, P to stop, or enter data for next letter

PRINT FORMATTER—*controls print-time line formatting*

.PF

turns the print-time line formatter ON, OFF, or to default.

Formats:

.PF OFF:

This form suppresses print-time line formatting. Inserted text is printed without being formatted to accommodate the length of keywords.

.PF ON:

You must turn print-time line formatting ON before you can use other DOT commands to format printed text. Print-time line formatting remains ON until you issue a **.PF OFF:** or a **.PF DIS:** command.

Print-time line formatting with other DOT commands reformats text before it is printed.

.PF DIS: (default)

This form leaves print-time line formatting to the discretion of merge-print. Print-time line formatting is automatically turned ON when a variable identifier is detected, and is turned OFF when the next carriage return, line feed, form feed, or end of file is encountered.

RIGHT MARGIN—*sets right margin*

.RM affects the right margin setting during merge-printing.

Formats:

.RM ***n:***

A number between 1 and 240 specifies the column at which the right margin is set for the printed text.

NOTE

*Print-time line formatting must be ON before **.RM** will have any effect. (See the **.PF** command.)*

.RM DIS: (default)

This form uses the right margin you specified when you created the file.

LEFT MARGIN—*sets left margin*

.LM specifies the left margin setting.

Formats:

.LM ***n:***

When **.LM** and a number between 1 and 240 occur during merge-printing, the left margin is affected as specified. If the document contains hanging indentations or text that extends to the left of a desired margin setting, don't use the **.LM** command.

NOTE

Print-time line formatting must be turned ON before ***.LM*** *commands will work. Print-time line formatting is turned ON when a variable identifier is detected in the current paragraph, or when the DOT command* ***.PF ON:*** *is encountered.*

.LM DIS: (default)

.LM DIS: uses the left margin setting you specified when you created the file.

LINE SPACING—*sets line spacing*

.LS establishes line spacing during merge-printing.

Formats:

.LS ***n***

You can specify line spacing between 1 and 9. **.LS** has no effect unless print-time line formatting is ON, as described for **.LM**.

.LS DIS: (default)

.LS DIS: causes document to be printed with the line spacing used when you created the document file.

INPUT JUSTIFICATION—*determines input justification*

.IJ determines whether the input scanner interprets input as justified.

Formats:

.IJ ON:

.IJ ON: interprets input text margins as justified.

.IJ OFF:

.IJ OFF: assumes that input text will not be justified.

.IJ DIS: (default)

.IJ DIS: leaves input text right margins as you specified them when you created the file.

Small variations in the right margin indicate ragged right. A constant right margin and soft spaces between words indicate justification.

NOTE

When you intend to justify output from ragged-right input, or vice versa, use the ***.IJ*** *DOT command.*

OUTPUT JUSTIFICATION—*determines output justification*

.OJ

right-justifies printed text.

Formats:

.OJ ON:

.OJ ON: prints text with the right margin aligned.

.OJ OFF:

.OJ OFF: prints text with a ragged right margin.

.OJ DIS: (default)

.OJ DIS: prints text using the right margin you specified when you created the file.

NOTE

.OJ *has no effect unless print-time line formatting is ON, as described for* ***.LM****.*

Conditional Expressions

IF CONDITION—*initiates printing if a condition is met*

.IF

The .IF command is used to specify that providing a certain condition is met, skip to the END command and begin printing.

Format:

```
.IF &keyword& comparison character "expression" GOTO [ / B] [ ;]
```

The expression dictates the conditions under which printing will be initiated. The expression contains:

- a keyword enclosed in ampersands

- a comparison character

=	"equal"
<>	"not equal"
<	"less than"
>	"greater than"
<=	"less than or equal to"
=<	"equal to or less than"
>=	"greater than or equal to"
=>	"equal to or greater than"

- a variable data item enclosed in quotation marks

- option for searching backward, a label or comment.

The comparison characters compare strings of any type characters. The strings are read in a sequence defined by the (ASCII Collating Sequence) a sample of which is shown below:

```
lowest                        →                         highest
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

You can take advantage of this hierarchal sequencing in an expression to select only a portion of data file for printing.

A /B placed after the GOTO in a conditional command, causes MailMerge to search backwards through the file for the end-of-file command .EF.

A label can be used to branch to a specific .EF command identified by the same label. The label option can be any string consisting of 20 characters as long as there are no spaces and the first character is not a number.

A comment is entered after a space and a semicolon at the end of a conditional command. These comments do not get printed but are displayed on the screen.

EXCEPT CONDITION—*resumes printing except when a condition is met*

.EX

The .EX command is used to specify that except when a certain condition is met, skip to the END command and begin printing.

Format:

```
.EX &keyword& comparison character "expression" GOTO [ /B] [ ;]
```

The expression dictates the conditions under which printing will be resumed. The expression contains:

- a keyword enclosed in ampersands

- a comparison character

=	"equal"
<>	"not equal"
<	"less than"
>	"greater than"
<=	"less than or equal to"
=<	"equal to or less than"
>=	"greater than or equal to"
=>	"equal to or greater than"

- a variable data item enclosed in quotation marks

- option for searching backward, a label or remark.

A /B placed after the GOTO in a conditional command, causes MailMerge to search backwards through the file for the end-of-file command .EF.

A label can be used to branch to a specific .EF command identified by the same label. The label option can be any string consisting of 20 characters as long as there are no spaces and the first character is not a number.

A comment is entered after a space and a semicolon at the end of a conditional command. These comments do not get printed but are displayed on the screen.

END OF FILE—*specifies the end of a file*

.EF

Format:

.EF [Label]

The .EF concludes processing of MailMerge commands and initiates printing. The "End Of File" command only works in a document file that contains a conditional command. There is no limit to the number of .EF commands that can be used in the file as long as there is a corresponding conditional command. Never place an .EF command directly under a conditional command; there should always be a carriage return separating them. An optional label which corresponds to that used in a conditional expression may be used.

WordStar Default Values

	D Option	**N Option**
Left margin	column 1	column 2
Right margin	column 50	column 50
Variable tab stops	6, 11, 16, etc.	9, 17, 25, etc.
Word wrap	ON	OFF
Justification	ON	OFF
Ruler display	ON	OFF
Page break display	ON	OFF
Print-control display	ON	OFF
Soft-hyphen entry	OFF	ON
Hyphen-Help	ON	OFF
Insert mode	ON	OFF

SuperCalc

Cursor Movement

There are several ways to move the worksheet cursor to a new active cell. You can use the four arrow keys. Similarly, you can use the alternate diamond keys. Hold down the **CTRL** key while you press the **S**, **D**, **E**, or **X** key to move left, right, up or down in that order.

Alternately, you can use the = (address) command—also called the **GOTO** command—to move directly to the designated cell. The SuperCalc program will ask for the cell coordinates. When you type them, the display on your screen will change. If the designated cell is already on the display, it will show as the active cell. If not, the window will move to show the new active cell at the upper-left corner. There is a special case: if you type only = and press **RETURN**, the window will adjust to show the current active cell at the upper left.

Special Function Keys

? for help

When you use the SuperCalc program and need information about your current entry options, press **?**. The display screen will change to show you a list of entries that you can make relative to your present position within SuperCalc. This help function is available at any time and in any mode. Press any key to return to the previous display.

The ! key

The **!** command forces recalculation. In the manual-calculation mode, this command is the only way to have the program recalculate values. In the automatic mode, it provides an additional recalculation.

The ; key

; moves the worksheet cursor from one portion of a split window display to the other portion. See the **WINDOW** command.

The ESC key

The current-cell key is the **ESC** (escape) key. When you press it, the SuperCalc program puts the location of the active cell onto the entry line for you to use in a command or expression. After you press **ESC**, the arrow and alternate diamond keys control the worksheet cursor. If you move the worksheet cursor, the active cell address on the entry line changes dynamically to reflect the new location. When you press **ESC** again, the address stops changing, and the arrow and diamond keys are again available for editing.

Pressing **,** after the active-cell address is a special case. SuperCalc places another active-cell address after the colon. The address before the **,** is fixed; the address after the **,**
is still changeable.

The new active-cell location is temporary. When you press **RETURN** to enter the command or expression, the worksheet cursor returns to the prior active-cell location. If you are entering data into a cell, it will go into that prior location.

Data Entry

SuperCalc accepts numbers, formulas, and text. Ordinary numbers can have 16 significant digits plus a decimal point. Scientific, or exponential, numbers can have 16 significant digits and a decimal point, all raised to a power of ten. The limit is the 63rd power of 10. Text can have up to 110 characters. Formulas can can have up to 110 characters and can include arithmetic expressions, relational expressions, functions, and references to cells.

Once you begin to type a command or data on the entry line, the four arrows—and the alternate diamond keys—no longer move the worksheet cursor around the worksheet. Instead, you can use them to edit information on the entry line. You can always correct your commands or data while they are on the entry line. The **EDIT** command allows you to use the edit process, and enter the changed contents into the active cell after you have committed an entry to the worksheet.

Left and right arrows (**^S** or **^D**) move the data-entry cursor without erasing an entry so you can position the cursor where you want to make the change.

Because a cell can contain 110 characters—longer than entry line can show—SuperCalc will scroll your entry during the edit process, allowing you to examine any portion of it. Wherever the cursor is, you can enter a new character to replace the old one. The cursor then moves right one location.

Each time you press the down arrow (or **^X**) it deletes a character. The cursor stays in position.

The up arrow (or **^E**) inserts a new space at the cursor location each time you press it. The cursor stays in place, and spaces fill out to the right of it. The space(s) can then be filled with additional characters.

Remember, what you see on the entry line is what is entered into the active cell. When you finish making your changes and enter the data or execute the command, SuperCalc takes everything on the entry line, not just the material to the left of the cursor.

Data-Entry Limit

Numbers: 16 significant digits, plus optional decimal point and optional sign for ordinary numbers. 16 significant digits, plus decimal point and optional sign for exponential numbers (scientific notation). These 16 digits can be raised to the 63rd power of 10.

Largest ordinary number.	9999999999999999
Smallest ordinary number.	−9999999999999999
Largest exponential number.	9.99999999999999e62
Smallest exponential number.	−9.99999999999999e62

Text: 115 characters

Example: **"Expenses, January**

Formulas: 116 characters

Example: **7+A5,9+5*E7,SUM(B1:B9), MIN(A4,D4,G4)**

Distinguishing Numbers, Text, and Formulas When Entering Data

Numbers start with digits (0–9), +, −, or a period. An entry beginning with a period is assumed to be a decimal entry beginning with zero and a decimal point.

Text starts with a quotation mark (").

Formulas can start with the same characters as numbers—0–9, +, −, or period. They can also start with an open parenthesis —(. You can put arithmetic expressions, relational expressions, functions, and references to cells within formulas.

About Numbers

Numbers are ordinarily right-justified; optionally, they're left-justified. The way numbers appear when they're displayed depends on the display format you selected, not on the way they looked when you entered them into the cell. The display format doesn't affect the content of the cell.

Display options allow you to display numbers in the following ways:

- general (ordinary numbers if they fit column display width; otherwise, exponential);
- exponential (scientific notation), rounded if necessary; integer (integers only; if there is a decimal number, round up or down to make it integer);
- dollar amounts, rounded to the nearest cent; .00 is appended to whole numbers;
- graphic display, using asterisks to show relative values in bar-graph form.

For any display format, if the numeric display cannot fit into the column, then >>>> fills the column.

You can widen a column to display a number or text in full by setting column width from 1 to 126.

NOTE

Text is ordinarily left-justified; optionally it is right-justified. If text is too large for the column, the text display continues into the adjoining blank cell(s) to the right. If it cannot continue into adjoining columns, it is cut off at the right.

NOTE

*Ordinarily the resulting values rather than formulas are displayed; optionally, you can see the formula with the **F** option in the **GLOBAL** command. The formula, however, is shown on the status line. When the formula is displayed in a cell, it can continue into adjoining blank cells as text does.*

Status, Prompt, and Entry Information

SuperCalc uses the prompt and status lines near the bottom of your screen to send you messages. You use the entry line to respond. Here is a more detailed look at some of the things you may find on those lines:

Status Line

The status line is the first of the three lines. This line displays information about the active cell. The information displayed includes: the "Current Direction" that you have been moving, the active-cell location, the active cell's specific format and protection, and the textual contents of the active cell.

Here is an example of a status line:

>A5 L$TR P Text="February

Here is what it means:

> is the current direction of the worksheet cursor, set by the last arrow key pressed. It may be >, <, v, or ^.

A5 is the active-cell location. Data entered will go into that cell. Commands that use the current column or row will use the column or row containing that cell; in the example, column A and row 5.

L$TR shows the active-cell format settings; numbers are left-justified, $ is the format, and text is right-justified. (The detailed reference for /Format gives full information on these settings).

P shows data protection of the active cell. This area is blank if the cell is unprotected.

Text="February indicates the contents of the Active Cell —in this case, text. **"Rtxt="** indicates repeating text. Numbers or formulas are shown as **"Form"**; for example, "FORM =12*B9."

SuperCalc also uses the status line to display error messages and certain informational messages. These special messages disappear and the status information reappears when you press any key.

Prompt Line

The middle of the three information lines serves a dual purpose: while you are entering a command, this line "prompts" you by outlining the choice of possible entries you may make. For example, after you have issued the **DELETE** command, the prompt line reads:

```
R(ow) or C(olumn)
```

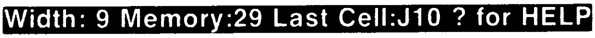
This tells you that you must next tell the SuperCalc program whether you wish to delete a Row or a Column. If you then ask for Row, the prompt line changes to:

```
Enter Row Number
```

to ask you which row number to delete.

Whatever the prompt is, if you press **?** a short but detailed explanation of your options will be displayed on the screen.

When you finish typing your command, the middle line reverts to its other function: global status. It tells you about your worksheet's current status. An example is:



```
Width: 9 Memory:29 Last Cell:J10 ? for HELP
```

This information is:

Width: 9—column width. This is the display width of the column that contains the active cell. The standard, or default, setting is 9, but you can specify a different width. You can set all columns to the same width or set different widths for different columns. If you change the default setting, the status line lists the display width that you select.

Memory: 29—available memory in kilobytes (a kilobyte is the memory sufficient to hold 1024 characters or digits). This number changes as you add data to the worksheet.

Cell: J10—this tells you the lower right-hand corner of an imaginary block that just contains all your worksheet. In other words, J is the right-most column that you have used, and 10 is the lowest row (biggest row number).

? for HELP—this reminds you that pressing **?** will always give you an explanation of the options you have at that moment. If you press **?**, you will receive an explanation of your choices.

/ precedes most commands. If you press it, the prompt line will change to list possible entries and the **?** symbol.

As you proceed within commands, or make other possible entries, the prompt line will change to show you your current choices.

Entry Line:

This is the line where you tell SuperCalc what to do by typing your commands or data. Your experience with *Volume 3: Working with Text and Numbers* and the information in this Reference Guide will give you all the information you need to type in the desired data.

SuperCalc Command Entry

Type all SuperCalc commands with **/** and the first letter of the command. The remaining letters in the command are automatically supplied on the entry line. For example, **/ B** causes the command word BLANK to be displayed on the entry line. The prompt line lists the choices available to you for that command. When you enter **/**, the prompt line shows the possible one-letter entries. After you choose a command, the prompt line changes to show the choices available for that particular command. Whenever you wish further information about your options, you can press **?**.

Some commands exhibit a sequence of prompts and entries before the command is executed. An example is the command to copy from one location to another. If you enter one of these multilevel commands, you can back out of your current entry by using the back (left) arrow. In fact, you can back entirely out of the command, level by level, till you return to the desired level.

You can edit commands just as you do data by using the in-line editor. Remember that when you press **RETURN**, everything visible on the entry line will be executed—not just the part of the command to the left of the cursor.

A few commands effect only the current cell, column, or row. Most allow you to specify which cell, column, or row to be affected in the command line. You can type column addresses as either capital or lowercase letters; SuperCalc converts lowercase column entries to capitals. If you want to specify the current cell, column, or row (as appropriate) in such commands, simply press the comma to enter the current location into the command line.

The current-cell key (**ESC**) can also enter the current cell, column, or row into the command line (if only the column or row is needed, the other part of the current cell location is ignored). Once you press **ESC**, you can move the active cell temporarily to a new location. Its address changes on your entry line, and you can use it in your command. By pressing **:** you can develop two cell addresses, such as B5:E5.

RETURN follows up a command, causing it to be executed. In some cases, a comma can also end a command because pressing the comma enters the last item of information needed in the command line. The command is complete, so SuperCalc executes it.

All commands consist of **/** and a single letter. SuperCalc's interpretive prompting fills out the rest of the word, and the prompt line lists the options available. These commands are summarized here and described in detail in the tutorial chapter.

Commands Involving Formulas

Some commands move formulas to new locations. It is usually desirable to adjust formulas for their new locations. For example, suppose cell D4 has the formula +B4*C4. If the contents of cells B4, C4, and D4 move to T7, T8, and T9, the formula in T9 should read +T7*T8. SuperCalc ordinarily makes such adjustments automatically.

Some commands optionally allow you to move formulas without adjustment, or query whether each cell reference for each formula should be adjusted. Some commands also have an option to move values only; formulas do not transfer, only their values move.

Commands Involving Formula Adjustments

DELETE, INSERT, and **MOVE** all cause automatic formula adjustment. They have no options. Deleting a column or row that contains a cell on which a formula, outside the range of the deletion depends, will cause an error.

COPY and **REPLICATE** allow formula adjustment. Adjustment is automatic, unless you specify otherwise by selecting one of the options. The options allow you to disable formula adjustments, or to choose whether SuperCalc should adjust individually for each outside reference.

LOAD adjusts formulas if you're loading the material into a worksheet location different from the one where it originated. In this case, you have the same options as in **COPY** and **REPLICATE**.

SuperCalc Commands

Data Commands

/BLANK—*blanks contents of a cell or range of cells*

Blanks the contents and clears the cell format of a cell, partial column, partial row, or block. Also clears the formatting of the cell if it has been formatted individually (that is, at the E(ntry) level; see **FORMAT**).

Prompt:

Enter Range

Formatting for a column or row is not affected, even if every cell in it is blanked. Only the **FORMAT**command can change the format for a column or row. Protected cells will be bypassed.

Examples:

/**Blank, c7** <**cr**>

/**Blank, c7:c12** <**cr**>

/**Blank, c7:h7** <**cr**>

/**Blank, c7:h7** <**cr**>

/EDIT—*transfers cell contents to entry line for editing*

The edit command lets you edit the contents of a specified cell, then place them back in the active cell. If the active cell is protected, you cannot edit.

Prompt:

From? Enter cell

Specify a cell in response to the prompt; , indicates the current or active cell. The cell contents come to the entry line, replacing the command on the line.

Edit with the in-line edit function. Use the arrow or diamond keys to move the cursor nondestructively left and right to characters you want to change. The character that will be altered is the one above the cursor. You can replace characters one-for-one by simply typing new characters over them. You can delete characters, including blanks, by pressing the down arrow (or **^X**). You can insert blanks by pressing the up arrow (or **^E**). Then if you wish, you can replace the blanks by typing other characters over them.

Example:

The active cell contains Janaurry.

/**E** and , bring this example to the entry line. Use the left arrow to move the cursor to the second "a" in Janaurry and type **ua**. Move the cursor right to one of the "r"s in Januarry, then press the down arrow to delete it, and press **RETURN**. (Remember, pressing **RETURN** puts the entire entry into the cell no matter where the cursor is positioned.)

The active cell now contains January.

/FORMAT—*specifies format for a given portion of the worksheet*

The Format command affects the worksheet **G**(lobally); by specific **C**(olumn), **R**(ow), **E**(ntry) cell; or range (col/row:col/row), in one or more of the following ways:

I(nteger) notation, **G**(eneral format), **E**(xponential) notation to the tenth power, (**$**) dollar format, **R**(ight) or **L**(eft) justification, (*****) asterisk fill relative to value. **D**(efault) is: the general display, numeric right-justified, text left-justified, and column width 9.

Prompts:

```
Enter Level: G(lobal), C(olumn), R(ow), or E(ntry)
```

Specify the portion of the worksheet to be affected; , will specify the current column or row. If you press **E**, you can specify a single cell or a range of cells; that is, a partial column or partial row. Using **E** to specify formatting at the cell level provides the highest priority of formatting.

The next prompt message you receive depends on the level of formatting you specified.

A level of **G** or **C** has this prompt:

```
Define Formats: (I,G,E,$,R,L,TR,TL, * ,D, column width)
```

A level of **R** or **E** has the same prompt, except that "column width" is not included because it is not a valid choice.

You may select as many of the formats as you wish. Here is a list of the possible format choices:

I displays numbers as integers. This rounds decimal fractions up or down to convert them to whole numbers.

E (exponential) displays the number in scientific notation, as a power of 10. For example: 1776 is 1.77e3, 1,000,000 is 1.0e6; round if necessary.

G (general) displays the number as an ordinary number if it fits in the column width; otherwise, it displays the number as an exponential number.

$ (dollar amount) rounds to the nearest cent and appends .00 to whole numbers. No dollar sign is displayed.

* (graphic display for numbers) uses asterisks to show the relative sizes of numbers. Allows bar graph display.

R,L (right-justify, left-justify) is for numbers.

TR,TL (text right, text left-justify) is for text.

0–126 is the column width for the specified column or for the worksheet.

D (default) resets to the next level of formatting. See note 2 below.

When your entries are contradictory, the SuperCalc program will act on the one entered last. For example, if you enter R,L,I,G, then L and G will take effect, and SuperCalc will ignore R and I.

NOTE

1. *Format does not apply to data entry. The contents of a cell remain as entered; format specifies how the contents are displayed.*

2. *Where formats differ, the order of precedence is first the cell (**E**), then row (**R**), column (**C**), and finally worksheet or global (**G**). Cell formatting overrides any format for the column or row where the cell is. Where row and column intersect, row formatting overrides. Any of these override the global settings.*

   *When the program starts up, these global format settings are in effect: general numeric display (**G**), numeric right justify (**R**), text left justify (**TL**), and a column width of 9.*

Examples:

/Format, **C**, **E**, **12** **<cr>**

/Format, **R**, **TR**, **<cr>**

/Format, **G**, **$**, **11**, **<cr>**

/Format, **E**, **E**, **<cr>**

Worksheet Adjustment Commands

/DELETE—*erases data from a specified column or row*

When you type the command **/D** you are asked whether to delete a column or row.

Prompts:

```
R(ow), C(olumn) or F(ile)?
```

If you reply **R** , the prompt becomes:

```
Enter Row Number
```

You may then type a number from 1 to 254, or type **,** for the current row.

If you reply **C** , the prompt will be:

```
Enter Column Letter
```

You may enter a letter designation from A to BK, or type (**,**) for the current column.

This command deletes the contents and formatting of the specified row or column. The command will not execute if a protected cell is in that row or column.

The rest of the worksheet makes the following adjustments:

- Rows below the deleted row move up, and all row numbering adjusts. If row 4 is deleted, row 5 moves up and becomes the new row 4, and so on.

- Columns to the right of the deleted column move left. If column D is deleted, column E moves and becomes column D, and so on.

Examples:

/Delete, **R,5, <cr>**

/Delete, **C,E, <cr>**

All formulas on the worksheet are automatically adjusted as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is deleted.
A prior reference was SUM(B2:B5)
That reference becomes SUM(B2:B4)
The contents that were at B5 are now at B4.

A reference to B3 would cause an error if column B or row 3 were deleted, because the contents vanish, and there can be no new reference to them. SuperCalc cannot assume that this is a special case, one where you want the old formula to refer to the new contents of cell B3. For example:

Cell A6 has the formula SUM(B3,F3,G3).
Column B is deleted.

Cell A6 will now display ERROR because the contents of B3 have vanished. To correct the error, you must correct the reference to B3 in cell A6.

/INSERT—*inserts an empty column or row where indicated*

The **INSERT** command inserts a column or row where needed. The inserted column or row replaces the specified column or row while the rest of the worksheet adjusts by reassignment of parameters.

Prompts:

```
R(ow) or C(olumn)?
```

If you reply **R** , the new prompt is:

```
Enter Row Number
```

You may select a number from 1 to 254, or type **,** for the current row.

If you reply **C** , the prompt is:

```
Enter Column Letter
```

You may enter a letter or letters from A to BK, or type a comma (**,**) for the current column.

This command inserts a new row or column of empty cells between existing rows or columns. A new row appears above the specified row; a new column appears to the left of the specified column.

The rest of the worksheet adjusts. Columns move right, rows move down. The contents of each column or row are preserved but have a new designation. The contents, if any, of the last row (254) or column (BK) are discarded. The command will not execute if that last row or column contains a protected cell.

Examples:

/Insert **R,5 <cr>**

/Insert **C,D <cr>**

All formulas on the worksheet are automatically adjusted as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is inserted.
A prior reference was SUM(B2:B5).
That reference becomes SUM(B2:B6).
The contents that were at B5 are now at B6.

A prior reference to B3 itself will become a reference to B4 when a new 3 is inserted.

/MOVE—*relocates a column or row of data*

The **MOVE** command transfers the contents from one column or row to another.

Prompts:

```
R(ow) or C(olumn)?
```

If your reply is **R** , the prompt is:

```
From? Enter row number
```

You may enter a number from 1 to 254, or type a comma (,) for the current row.

If you reply **C** , the prompt is:

```
From? Enter column letter
```

You may enter a column designation from A to BK.

After you have specified a row or column, SuperCalc will ask the destination of the move. The prompt is:

```
To? Enter column letter
```

Reply with a row or column designation, whichever is appropriate. Pressing **,** or the current-cell key (**ESC**) will designate the current row or column.

The **MOVE** command adjusts the worksheet without destroying any data or performing any formatting. It moves a specified column left or right and inserts it in a new location, or moves a specified row up or down and inserts it in a new location. The columns, or rows between, move to fill the old location. They move in the opposite direction of the basic move.

Examples:

/Move **R,5,12 <cr>**

/Move **C,E,A <cr>**

All formulas on the worksheet adjust automatically as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is moved to row 5.
The former rows 4 and 5 move up to become new rows 3 and 4.
The former row 3 becomes row 5.

A prior reference was SUM(B2:B5) —That reference becomes SUM(B2:B4). The contents of B5 are now at B4.

Copying and Replicating Rows or Columns

/COPY—*duplicates data from source to destination*

The **COPY** command allows a one-to-one copy of a cell, partial column, partial row, or block to a new location. Options give a choice of formula adjustment or copying values only.

Prompts:

```
From? (Enter Range)
```

Specify a cell, partial column, partial row, or block.

The next prompt is:

```
TO? (Enter Cell), then Return; or "," for Options
```

Copy makes a one-to-one copy of the source into a destination of the same shape and size. Enter a single cell address to give the new location:

For a partial column, give the upper cell.
For a partial row, give the left cell.
For a block, give the upper left cell.

Press **RETURN**, or if you wish a choice of options for copying formulas, press the comma key.

If you press **RETURN**, then all the formulas are copied and automatically adjusted; that is, all references to other cells are adjusted for their new location, if possible.

If you press , to select options, SuperCalc will enter the cursor's location as a destination. Delete if not wanted. It will prompt you with:

```
N(o) Adjust, A(sk for Adjust). V(alues)
```

N—Copies formulas exactly as they are.

A—Allows you to choose for each reference to another cell address within a formula whether to copy it as is, or to have the SuperCalc program adjust it.

V—Copies the values only, without formulas.

When you choose the (A)sk option, each formula that qualifies for possible adjustment is displayed on the entry line. Its source and destination address are shown on the prompt line. SuperCalc positions the cursor at each cell reference on the entry line, and asks you to reply **Y** or **N**. Y means yes, automatically adjust. N means no adjustment, transfer as is.

Examples:

/Copy, **b9, c12 <cr>**

copy cell to cell.

/Copy, **b9:b15, e9 <cr>**

copy partial column to partial column.

/Copy, **b9:g9, h12 <cr>**

copy partial row to partial row.

/Copy, **b9:g15, k20 <cr>**

copy block to block.

/Copy, **b9, c12,N <cr>**

copy without adjustment.

/Copy, **b9, b15, e9, A <cr>**

copy, ask for individual choice of adjustment.

/REPLICATE—*transfers source until specified range is filled*

The **REPLICATE** command makes a one-to-many copy of a cell to a group of cells, a partial column to a group of partial columns, or a partial row to a group of partial rows. Options give a choice of formula adjustment or replicating values only.

Prompts:

```
From (Enter Range)
```

Specify a cell, partial column, or partial row, followed by a comma.

The next prompt is:

```
To? (Enter Range), then Return;
or "," for Options.
```

Replicate makes a one-to-many copy of its source into a new destination that is larger than the source:

A cell into a partial column or partial row.

A partial column into a group of partial columns. The destination address is given as the left and right cell addresses on the top row of the destination group. The partial column will be copied once for each cell in that portion of the row.

A partial row into a group of partial rows. The destination address is given as the upper and lower cell addresses for the left column of the destination group. The partial row will be copied once for each cell in that portion of the row.

Specify the destination and press **RETURN**; then if you wish a choice of options for copying formulas, press ,.

The options are the same as those for **COPY**. If you press **RETURN**, formulas are adjusted automatically. The options are: no adjustment (**N**), whether to adjust for values only (**A**), or leave formulas behind (**V**). (See /**COPY** above for details.)

Examples:

/Replicate, **b12,e3:e8 <cr>**

replicates a cell into a partial column.

/Replicate, **b12,e3:j3 <cr>**

replicates a cell into a partial row.

/Replicate, **b3:b7,d3:j3 <cr>**

replicates a partial column into a group of partial columns. In this example, the partial column is five cells deep. The result will be a block of cells repeating that partial column seven times. The top of that block is on row 3.

/Replicate, **b3:e3, g5:g7** <cr>

replicates a partial row into a group of partial rows. The partial row here is four cells across. The result will be a block of cells repeating the partial row three times. The left side of that block is column G.

/Replicate, **b12, e3:e8,N** <cr>

replicates without adjustment.

/Replicate, **b12, e3:j3, A** <cr>

replicates and asks for individual choice of adjustment.

NOTE

*As a special case, /**REPLICATE** can make a one-for-one copy just as /**COPY** does. /**COPY** cannot make multiple copies. /**COPY** can, however, do something that /**REPLICATE** cannot do; it can copy a block.*

Data Protection Commands

/PROTECT—*provides protection against alteration of data*

The /**PROTECT** command shields the contents and formatting of specified cells from alteration. You can't enter or edit data in protected cells.

Prompt:

Enter Range

/BLANK, **/FORMAT**, **/COPY**, **/REPLICATE**, and **/LOAD** all bypass protected cells—that is, the commands operate on surrounding cells but leave the protected cells unchanged. **/DELETE** will not work if a protected cell is in the specified row or column.

There is one exception: the **/ZAP** command overrides protection.

Examples:

/Protect,**c3** <**cr**>

/Protect,**c3:c9** <**cr**>

/Protect,**c3:g3** <**cr**>

/Protect,**c3:g9** <**cr**>

/UNPROTECT—*allows exposure of previously protected cells*

This command removes protection from a cell, partial row or block.

Prompt:

Enter Range

Allows you to change cell contents or format. There is no error if you try to remove protection from something that is not protected.

Examples:

/Unprotect,c3 <cr>

/Unprotect,c3: <cr>

/Unprotect,c3:g3 <cr>

/Unprotect,c3:g9 <cr>

LOAD, SAVE and EXECUTE Commands

/LOAD—*loads and displays part or all of a disk file*

The **/LOAD** command reads a SuperCalc data file from a diskette and loads it into memory; the worksheet displays the contents of the file. You may load all or part of a worksheet at a location you specify. Options give a choice of formula adjustment or loading values only.

Prompts:

```
Enter File Name (or RETURN for directory)
```

Enter the name of the desired file with the drive designation, unless you want the file loaded from the SuperCalc diskette. The file name must have the :CAL-type "CAL". This extension is assumed, and you do not have to enter it. Do not leave blank spaces in the file name. For example:

SALESFEB <cr>

would load the file from the SuperCalc program diskette into drive A.

B:SALESFEB <cr>

would load the file from the B disk drive.

You receive a choice of loading the entire file or a specific portion of the file. The following prompt displays:

```
A(ll) or P(art)?
```

If you reply **A**, the entire worksheet is loaded into the original location.

If you reply **P**, then further questions appear on the prompt line:

```
From? (Enter Range)
```

Specify the position of the saved worksheet that you wish to load.

To? (Enter Range) then RETURN or "," for options.

Enter the cell address at the upper left of your destination, which may be a new location for that portion of your worksheet. Press **RETURN** if you wish automatic adjustment of formulas for the new location; otherwise, press **,** for options. The options are: **N**(o Adjustment), **A**(sk for Adjust), or **V**(alues) only. (See /**COPY** for an explanation of these options.)

NOTE

If there are protected cells in the destination area, they will remain unchanged.

Examples:

/Load, **QUARTER3** <cr>

/Load, **B;QUARTER3** <cr>

/SAVE—*stores data from the current worksheet to disk*

The /**SAVE** command stores the worksheet contents and all settings on a disk file. Options give a choice of saving all contents or values only.

Prompts:

```
Enter File Name (or < RETURN > for directory)
```

Enter the name you have chosen for saving your worksheet. Also enter the drive designation if you do not want to write it to the disk in the default drive (A). The SuperCalc program will automatically give the file the .CAL file-type extension. You do not need to enter it as part of the file name. The next prompt is:

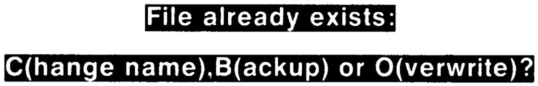
```
A(ll), V(alues) or P(art)
```

A specifies that all cell contents will be saved; **V** specifies that values will be saved without formulas. For either case, all of these are saved:

format settings, global options, title locking, window splitting, and active-cell location. **P** allows you to save only part of of your current worksheet. When saving part of the worksheet, you can decide whether to save the entire portion **A**; or just the values **V**.

NOTE

If you specify the name of an existing file, the program will display the following prompt:



```
File already exists:
C(hange name),B(ackup) or O(verwrite)?
```

Examples:

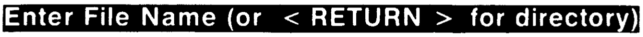
/Save, **WORK5** **<cr>**

/Save, **B:WORK5** **<cr>**

/XECUTE—*executes a group of commands from a disk file*

The /**Xecute** command causes the commands in a named file to be executed one after another. The /Xecute command allows you to execute a WordStar text file or SuperCalc file of command strings. When you enter /X the prompt line changes to:

Prompt:



```
Enter File Name (or  < RETURN >  for directory)
```

If you press **RETURN**, you will be given the option to display the directory (explained in the /Delete command). If you enter a file name, the SuperCalc program reads each of the commands in the specified file a character at a time. If the file is not in the proper format or a command is in error, an error message is displayed on the status line and the Xecute command is abandoned. You can terminate the command at any time with **^Z.**

NOTE

The default extension for command files is .XQT. If your file has no extension, you must still place a period after the file name.

Example:

```
/X TEST1 <cr>
```

The WordStar or SuperCalc file named TEST1 would look like this:

```
/ZY
/FCA,20
/LB:BALANCE,A
/GF/GM/FGD,$
```

Worksheet Display Commands

/TITLE—*provides method for fixing titles*

Title allows you to lock columns, rows, or both into their place on the display window. Locked information will not scroll; however, other information on the screen can scroll. Title lock uses the current row and column as the coordinates to be affected.

Prompt:

```
H(oriz), V(ert), B(oth), or C(lear)?
```

H locks the current row and all rows above it.

V locks the current column and all columns to the left of it.

B locks both the current row and column and all rows above and columns to the left.

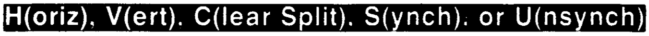
C removes the title lock.

A replaces a prior title lock with a new one.

/WINDOW—*splits the screen into two worksheets*

The **/WINDOW** command splits the display window into two parts. Each portion can have separate format settings and options. The screen is split at the current row or column.

Prompt:



```
H(oriz). V(ert). C(lear Split). S(ynch). or U(nsynch)
```

H The screen splits horizontally; the current row moves down and a second border replaces it. The active cell moves up one cell in its column.

V The screen splits vertically; the current column moves right and a second border replaces it. The active cell moves left one cell in its row.

NOTE

In both these cases, there is an alternate active cell in the original location. You can switch between the two active cells by pressing ; as they move independently.

C Clears the split screen. The portion that was above or to the left is the primary screen; it is now displayed in full.

S Synchronizes scrolling in the two portions.

U Unsynchronizes scrolling; the two portions will scroll independently.

Within the two portions of the screen, you can set formatting and global options independently. It is possible to show the same data with different formatting and options—for example, to show the same column as values and as formulas.

When the split is cleared, the options and formats for the primary screen remain. The primary screen is the portion above or to the left.

Data Display and Printing Commands

/OUTPUT—*sends worksheet contents to the printer or disk file*

This command writes part or all of the worksheet to the printer, the terminal, or a disk text file. You can write out a partial column, partial row, or block. If you write the report to a disk file, you can use WordStar to add further information or modify formats before printing, or to include the SuperCalc report within other text.

Prompts:

```
D(isplay) or C(ontents) report?
```

The worksheet information can be written out in the way it is displayed, or as the actual cell-by-cell contents. If you choose **D**, for display, the entire worksheet is output. If you choose **C**, for contents, the following prompt appears:

```
Enter range
```

After you specify the portion of the worksheet to output, a prompt asks whether you want the data output to the printer, console, or disk. You may also change the default printer settings:

P(rinter), S(etup), C(onsole), D(isk),

Type **P** to send the data to the printer, **C** to display it on the screen, or **D** to send it to a disk file. Type **^Z** to stop the output. If you type **S**, for setup, the following options are provided:

Select printer control:

```
L  = Change page length      (now # lines)
     (Length = 0 for continuous form)
W  = Change page width       (now # chars)
S  = Manual setup codes
P  = Print report
```

CNTRL-Z to cancel /O command

You may change one or more of the above parameters, then type **P** to output the report, or **^Z** to cancel the entire process.

GLOBAL Options, QUIT, ZAP

/GLOBAL— *manipulates screen formatting and calculations*

The /**GLOBAL** command lets you view formulas on which values are based, change the appearance of the screen display, and specify the order and sequence of calculations.

Prompt:

```
F(orm),N(ext),B(order),T(ab),R(ow),
        C(ol),M(an),A(uto)
```

If you respond to the prompt by pressing **F**, the display window will show the formulas contained in the cells instead of the values that result from the formula calculations. If formulas are currently being displayed, pressing **F** will display the values.

If you respond to the prompt by pressing **N**, the cursor will "auto-advance in the "current direction" after the data is entered into a cell. If auto-advance of the cursor is already in effect, then pressing **N** causes no auto-advance of the cursor after the data is entered into a cell.

Pressing **B** will suppress the display of the worksheet border. If you already suppressed the border display, then pressing **B** will restore the border display. ("Border" refers to the column and row designations across the top and down the left side of your display window.)

Pressing **T** activates the Tab mode, or deactivates it if SuperCalc is already in the Tab mode. In the Tab mode, advancing between cells skips all empty or protected cells. Therefore, you can never select a protected or an empty cell as the active cell in this mode.

Options **R**, **C**, **M**, and **A** concern recalculation.

> **R** means recalculate by rows, from the top down. (Rows are recalculated left to right.)
>
> **C** means recalculate by columns, from the left across. (Columns are recalculated from top down.)
>
> **A** means recalculation is automatic (default).
>
> **M** means recalculation occurs at your request, whenever you press the ! key.

/QUIT—*exits from SuperCalc and returns to CP/M*

The /**QUIT** command leaves SuperCalc and relinquishes control to the CP/M Plus operating system. You get a chance to save your work on diskette before the transition occurs.

Prompt:

```
EXIT SuperCalc? Y(es) or N(o)
```

If you reply **Y**, you return to CP/M as indicated by the A> prompt. If you reply **N**, you return to SuperCalc. Any other reply is ignored.

If you have work you could lose when you quit, SuperCalc gives you a chance to save the work before exiting.

Example:

/Quit <**cr**>

/ZAP—*clears the entire worksheet of data*

The /**ZAP** command clears the contents and formatting from the entire worksheet.

Prompt:

```
Y(es) to clear everything, else N(o)
```

All cells become empty. All format settings and modes of operation revert to their standard settings. Everything starts fresh, as if you had just started up the SuperCalc program.

ZAP is the only command that can override protection of cells.

NOTE

Remember, when you ZAP the worksheet, nothing remains.

Examples:

/ZAP, **Y**

/ZAP, **N**

SuperCalc Built-In Functions

ABS (value): Provides the absolute value.

AVERAGE (list): Provides the arithmetic mean of the nonblank values in the list.

COUNT (list): Returns the number of nonblank entries in the list.

ERROR, NA: Displays ERROR or NA (not available) for the cell having this function and for any cell with a formula referring to this cell.

EXP (value): Raises "e" exponentially. The value is the exponent.

OR (expression 1, expression 2): Results in "true" (value of 1) if either expression 1 or expression 2 is "true" (nonzero); otherwise, results in "false" (value of 0).

AND (expression 1, expression 2): Results in "true" (value of 1) if both expression 1 and expression 2 are "true" (nonzero); otherwise, results in "false" (value of 0).

NOT (expression): Results in "true" (value of 1) if expression is "false" (zero); otherwise, results in "false" (value of 0).

IF (expl,exp2, exp3): If expression 1 is true, then use expression 2; otherwise, use expression 3. Expression may be combined with AND or OR NOT to form expression 1.

INT (value): Returns integer portion of value. The value is not rounded. Do not confuse this function with **/FORMAT,I** which will round off numerical entries.

LOOKUP (value, column/row range): Searches the range for the last value less than or equal to the search value given. Returns the adjacent value from the column to the right of the search column or the row below the search row. Assumes the search range is in ascending order of values.

L*n* (value), LOG 10 (value): Provides the natural log, log base 10, of the value.

MAX (list), MIN (list): Provides the maximum or minimum value in the list.

NPV (discount, column/row range): Nets the present value of a group of cash returns at the given rate of discount. The cash amounts are assumed to be projected for equal time periods, such as every year; and the discount rate is for that interval. The first cash entry is discounted once, the second twice, and so forth, and added to form the total value.

PI: Returns Pi to 16 significant digits.

SIN (value), ASIN (value), COS (value), ACOS (value), TAN (value), ATAN (value): Trigonometric calculation of the value. ASIN is arcsine, etc. Trigonometric results are give in radians.

SQRT (value): Returns the square root of the value.

SUM (list): Returns the sum of the values in the list. Here is a quick explanation of what "value," "range," and "list" mean in this context: Value is a constant, the value of a cell, or a combination of these values made by using the arithmetic operators.

Formulas and Functions

Formulas specify calculations and comparisons. Formulas use values in other cells (which may be themselves the result of formulas), constants, and built-in functions. These values are combined using arithmetic and relational operators:

+	addition
-	subtraction
*	multiplication
/	division
^	raising to a power
=	is equal to
<>	is not equal to
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to

Examples:

Constants: **12,5.9,3.4e3**

Cell values: **A12, B19, BK54**

Combinations: **12+5.9,B19-3,7,A12*B14,(9+E5)/4**

The combinations are also called "expressions." They are evaluated from left to right; * and / are evaluated before + and -. Use parentheses to group terms in your expressions so that SuperCalc will evaluate them as you wish. Some examples follow:

5+4*3+1=18 (that is, 5+12+1)

(5+4)*3+1=28 (that is,9*3+1)

5+4*(3+1)=21 (that is, 5+4*4)

(5+4)*(3+1)=36 (that is, 9*4)

Here are some examples of functions with values:

BS(A12),SQRT(9.5*E7),LN(3.5e4),TAN(C5+E5)

Range is simply a partial column or partial row, such as B4:B12 or B4:H4. Here are some examples of functions that use both a value and a range:

LOOKUP(7,C5:J5)

LOOKUP(A4,D3:d12),

NPV(.18,D12:H12)

NPV(B4,G3:G8)

A list can have values, expressions, and ranges. Here are some examples:

SUM(A12,B9,D5)

SUM(C12:E12,H3:H7)

SUM(MAX(C12:E12)

COUNT(E3:E12,F8:J8)

AVERAGE(B7,B8:h8,C12:C20)

Sample worksheets provided on your SuperCalc disk give examples of these formulas in actual use. This material will help you understand how you can put the formulas to work; it is especially useful for **IF**, **LOOKUP**, and **NPV**.

Practical Suggestions

1. Keep your work in the upper left of the worksheet grid.
2. Keep your work in a rectangular shape. Try to avoid having long columns or rows projecting outside the basic shape.
3. Do not blank cells, protect cells, or format cells in the area below or to the right of the area that you actually need. Especially, do not put data below or to the right of the area you actually need.
4. When you have extra or interim work on the screen that you can get rid of, use the following procedure to free that space completely:

   a. /**DELETE** or /**BLANK** the material you do not need.
   b. Move the rest of the work to the upper left of the grid, and adjust it as you wish it to display.
   c. /**SAVE** your work.
   d. /**ZAP** the screen.
   e. Reload. You are now using the minimum space required for your worksheet.

Worksheet Display

A command, text, or formula too long for the entry-line information on the entry line will scroll left when it reaches the end of the line. You can enter a command, text, or formula that is too long to display in its entirety. You can then use the in-line editor to examine any part of the entry by moving the cursor to the left or right. The information will scroll to show the hidden part of the line. When you want to enter the line, press **RETURN**. SuperCalc will take the entire entry, not just the portion to the left of the cursor.

Column Width Greater Than Screen Width

You may sometimes want to make the width of a column greater than the width of the screen. In such cases, you can scroll to see all of the display. If you have a printer with a wide carriage, you can use the output command to print the full width of the information. This feature can be useful for long text notes, explanations, or graphic display of numeric values.

To See the Same Information in Different Formats

The window command lets you look at the same information simultaneously in different formats. Split the single display window into two smaller windows. After you have split the screen, you can move one window so that it shows the same information as the other. Each part of the screen can have its own format settings for entries, rows, columns, or the entire worksheet. Each can have its own GLOBAL options settings. By using this technique, you could display both values and formulas for the same cell contents.

When you set formats or GLOBAL options for a split screen, remember that the portion above or to the left of your screen is "dominant." That is, when you cancel the split, the settings that were in effect for the upper or left window will remain in effect for the entire single display window.

Building Worksheets

Combining worksheet portions to build entirely new worksheets is possible. The **/SAVE** command saves the entire worksheet, but the **/LOAD** command can load all or part of a worksheet. It can place the part loaded at any worksheet location. This means that you can construct the nucleus of a new worksheet from parts of one or more existing worksheets.

When you have a fully developed worksheet with data, you can save it both with and without data. For example, you have developed a monthly report, which you save. Then you blank all the variable contents of the report, which you save. Then you blank all the variable contents of the report and save only the information that will not change, such as: titles, formatting, the general layout of the sheet formulas, and any constant values. Next month you can load this file, fill in the new information, and save it as your current monthly report.

Using PROTECT to Build New Worksheets from Old

The **/BLANK**, **/COPY**, **/LOAD**, and **/REPLICATE** commands all bypass protected cells, leaving their contents unchanged while changing surrounding cells. You can use this capability to combine information in detail, protecting key information and then surrounding it with new information by using LOAD, COPY, or REPLICATE.

Summing a Partial Column or Partial Row

When developing a worksheet, you may often insert new columns or rows within a range covered by a SUM formula. This can be awkward. Inserting or deleting at the top or bottom of an existing column or at the left or right of an existing row can mean redoing your formula. For example, you wish to insert a new row 12 and have to change the formula SUM(C2:C12) to SUM(C2:C13).

Here is a way to avoid this difficulty. Include a header or title at the top or left and an extra cell at the bottom or right within your sum. For a column, the extra cell could have "--------- as a total line. For example:

```
                    C
 1: January Receipts
 2:                                  35
 3:                                 405
 .:
 .:
 .:
 9:                                  38
10: ________________________________
11: SUM(C1:C10)
```

Text C1 and C10 have a zero value. Including them in the sum makes no difference. You can insert or delete rows from 2 through 9 and have the SUM formula automatically adjust to the new situation.

Security

Security includes protecting your work from accidental loss or change and protecting confidential information in your worksheet.

Protecting Your Worksheets

The CP/M Plus operating system allows you to specify files or entire disks as "read only." Designating your worksheet files this way means others can examine them or print reports from them, but cannot change or erase them.

The SuperCalc option to save values only offers another protection. Your full worksheet may have important proprietary information within its formulas or lookup tables.

After you have saved a full copy for yourself, you can save a Values-Only worksheet for others to use. In that worksheet, you may wish to remove lookup tables.

Similarly, you can use the output command to put a Values-Only copy of selected portions of your worksheet on a disk file for others to use. They can print that file or use the system text editor to include it in their own text file.

Save Your Work Often

It is important to save your work frequently while you are entering data or building worksheets. This practice insures you against losing the time and effort you have invested. It protects you against problems that are completely out of your control —such as power failures or hardware problems with your disk drive.

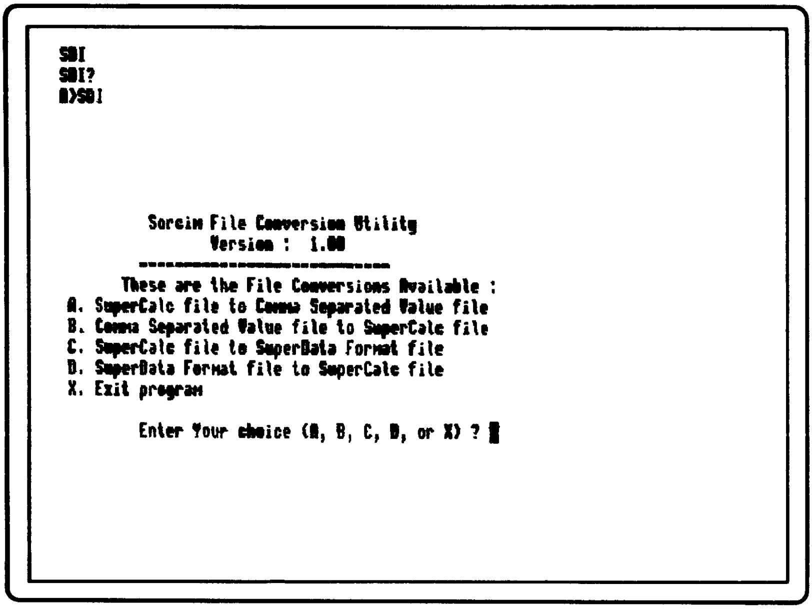
The Update option of the **/SAVE** command gives you a convenient way to do this. Every time you save your work, use the same name—for example, TRIALBAL. The first time you save your work, it is stored on the disk as TRIALBAL.CAL. The second time you save it, SuperCalc will tell you that there is a file of that name and ask you what you want to do. If you choose the Update option, your new worksheet will be saved as TRIALBAL.CAL and the earlier one will become TRIALBAL.BAK, your backup file. Whenever you use the Update option after that, SuperCalc will give you the two most recent files as filename.CAL and filename.BAK; it will erase any earlier files.

Having the backup file can be convenient; you may want to go back to that file in case a change does not work out in actual operation. You can use CP/M operations to change your file names so that filename.BAK becomes filename.CAL. Or you can directly load the file by giving its full name including the .BAK extension.

SDI—SuperData Interchange Program

SuperData Interchange (SDI) is a program that converts data files into a format which can be used by different programs. SDI will convert a SuperCalc file to either a "Comma Separated Value" (CSV) or a "SuperData Format" (SDF) file. SDI will also convert files in either of these two formats into a SuperCalc-format file.

SDI displays a menu (see following page) that allows you to select which kind of conversion you want.



```
SDI
SDI?
A>SDI



         Sorcim File Conversion Utility
               Version :  1.00
         ----------------------------
        These are the File Conversions Available :
A. SuperCalc file to Comma Separated Value file
B. Comma Separated Value file to SuperCalc file
C. SuperCalc file to SuperData Format file
D. SuperData Format file to SuperCalc file
X. Exit program

        Enter Your choice (A, B, C, D, or X) ?
```

You are then asked for the name of the file to be converted and the name of the file where the converted data will be stored. When you enter a file without a drive specification, the active drive is assumed. A file type of .CAL is automatically supplied for SuperCalc files, .CSV for Comma-Separated-Value files, and .SDF for SuperData Format files. The original file contents are not altered. If you select a conversion and the destination file name already exists, SDI will inform you of this and ask for your instructions:

```
FILE ALREADY EXISTS!
OKAY TO OVERWRITE THE FILE (Y/N)?
```

Any response other than Yor N will be ignored. An N response will cause the system to abandon the conversion process and redisplay the menu. A response of Y will cause the system to delete the existing file and create the new file.

It is important to note that the formulas SuperCalc can contain are not sent to the converted Comma-Separated-Value (.CSV) file. Only the values that show on the screen at the time of the conversion are sent to the file. This means that if you convert a SuperCalc-format file containing formulas to the .CSV format, and then immediately convert the .CSV file back to a SuperCalc-format file, some important information will have been lost.

Also note that if you immediately convert a SuperData-Format file that contained formulas to a SuperCalc file and then to a Comma-Separated-Value file as follows:

SuperCalc file (CAL) → SuperData Format (SDF) →

SuperCalc file (CAL) → Comma Separated Value (CSV),

the Comma-Separated-Value file will not contain any data that was in a cell with a formula. This is because the SuperData format carries the formula and not the value associated with it. To prevent this, simply load the SuperCalc file into the SuperCalc program and resave this file to disk before doing the conversion from SuperCalc format to CSV format.

Normally, you don't want the formulas transmitted to a .CSV file. Programs that create .CSV-type files cannot use formulas as data and do not create files with formulas either.

Creating an SDF or CSV File

The SDF file itself may be created in many ways. One such way is to have a program that reads and writes the SDF (or DIF) or the CSV-format file. The information may also be generated with a BASIC program.

Another way to generate the information is with WordStar. The file created must not contain any special control characters or "high-bit" set characters. This means that the "non-document" (N) mode must be used in WordStar.

It is also possible to generate the information on a mainframe computer and download the information to your system. This data can either be written on the mainframe in the format needed, or manipulated into the proper format by an editor or BASIC program after downloading.

Comma-Separated-Value Format File Structure

The Comma-Separated-Value (CSV) format is the simplest way to present data to the SDI conversion program. This format will carry only data and no formulas or formatting characteristics. This makes the structure of the file much simpler than that of an SDF file.

CSV FILE LAYOUT

The layout of a CSV file is basically what its name states: values separated by commas. The file consists of rows of data with each row terminated by a carriage return and a line-feed character. The data items on each row are separated by commas, with the string data enclosed in quotation marks.

The file contains no other control characters except the end-of-file character, control Z (which is represented as 1A in hex or 26 in decimal).

The file should have the same number of values on each line; however, this is not a requirement. If the number of values is not the same on each line, the layout of the data when converted to a SuperCalc worksheet will be less predictable and not look as pleasing. With the same number of values on each row, the layout of the SuperCalc worksheet will be a rectangular (or square) block of data.

The numeric values must be in SuperCalc-readable form. SuperCalc will accept integers, real numbers, and exponential numbers, as shown below:

123 123.345 −123 −123.345
12E4 123E-12 −12E5

The string values are a string of characters enclosed in double quotation marks (" "). The string may contain blanks, commas, and special characters like /, *, etc., as shown below:

"This is a string." "This too!"
"123,234.45 is a string also"

The following is an example of a CSV file containing numbers and strings:

```
123,"John Smith","ground beef",12.45<CR>
124,"Betty Jones","top sirloin",34.54<CR>
125,"Jane Johnson","chicken",4.67<CR>
```

The following is an example of a CSV file containing only numbers:

```
123.45,456.77,4322.56,837.233,9198.0,344.94<CR>
323.45,8989.84,3939.93,39.8,3494.343,343.99<CR>
```

SuperData-Format File Structure

The SuperData-Format (SDF) file structure is simple in concept, but more complex in implementation. The file may contain information about the general appearance of the spreadsheet, as well as the data in the spreadsheet.

The SDF file structure is a superset of the DIF structure used with Visi-series software products. SDF incorporates the major components of a DIF file, but has added other DATA and HEADER items to enable a file to carry more information. The original DIF specification contained only numeric and string data. In addition, the SDF structure also contains information on the formulas and formatting characteristics of the SuperCalc worksheet.

In the SDF file layout, the information on appearance is contained in the HEADER section, which describes the number of rows and columns, GLOBAL formatting, column width, etc. Once the general appearance of the spreadsheet is established, the contents of each cell is described, including special formatting instructions. This information is contained in the DATA section.

THE HEADER SECTION

The HEADER section is comprised of required and optional fields. The required fields are "TABLE" and "DATA." The DATA field must be the last field of the HEADER section. The following optional fields are also allowed between the TABLE and DATA fields: "COL-FORMAT," "ROW-FORMAT," and "GDISP-FORMAT."

Header Items

TABLE is always the first field of an SDF file. The only variable information is the contents of the TITLE.

Format	**Sample Entry**	
TABLE	TABLE	
0,version #	0,1	
"TITLE"	"BUDGET"	The title "BUDGET" is the name of the file itself.

ROW-FORMAT — The row number must be in the range 1–254 rows. As many ROW-FORMAT fields as necessary may be included in the header.

Format	**Sample Entry**	
ROW-FORMAT	ROW-FORMAT	
Row #,0	14,0	(row 14)
FORMAT STRING	TL$	(text left, $ formatting)

GDISP-FORMAT alters the GLOBAL format settings of the SuperCalc worksheet.

Format	**Sample Entry**	
GDISP-FORMAT	GDISP-FORMAT	
WIDTH,0	9,0	
STRING-FORMAT	$TL	(GLOBAL column width of 9, text left, $ format)

COL-FORMAT — The column number must be in the range 1–63 columns. This specifies the formatting of a particular column (for example, exceptions to the GLOBAL settings). As many COL-FORMAT fields as necessary may be included in the header.

Format	**Sample Entry**	
COL-FORMAT	COL-FORMAT	
Column #, Width	3,12	
STRING-FORMAT	1	(column 13 is 12 characters wide and integer format)

DATA must be the last field in the header and signifies the end of the HEADER section and the beginning of the DATA section.

Format	Sample Entry
DATA	DATA
0,0	0,0
STRING	" "

Now, putting this all together, we can show some valid SDF headers. The example below contains the minimum amount of data allowed in a header:

```
TABLE
0,1
"SAMPLE SDF FILE"
DATA
0,0
" "
```

The following example contains some optional fields:

```
TABLE
0,1
" "            (spreadsheet title not needed)
COL-FORMAT
1,40
" "            (no special formatting other than width)
COL-FORMAT
2,15
$              (dollar format)
GDISP-FORMAT
9,0
GTL            (general format with text left)
DATA
0,0
" "
```

THE DATA SECTION

The format of DATA items differs from that of HEADER items. SDF organizes data by rows. Within the rows, values are arranged according to the order of the columns.

Each data entry consists of three fields on two lines. For example:

Line 1	**Field 1, Field 2**
Line 2	**Field 3**

The first line contains two numeric values:

Field 1	**A type indicator**
Field 2	**A numeric value**

The second line contains a string variable:

Field 3	**A string value**

This could also be shown like this:

type indicator, numeric value
string value

The type indicator **must** be an integer from 0 to 1 or −1 to −5. The meaning of each indicator is as follows:

Type **Description**

0 **The value of the cell is a data value.** A 0 means the data is numeric and is stored in the field immediately following the type indicator. The string value will be a "V," indicating that this is a value. The string value may also have other values with different meanings such as "ERROR," perhaps due to an

invalid calculation. In this case, the numeric data field is set to 0. See the subsections on "String Values" and "Type Indicator Examples" for further explanation on the other meanings of the string value.

```
0,123.45
V
```

1 **The data item is a string.** For string data, the numeric field is ignored and the string value is stored on the second line of the data item.

```
1,0
STRING
```

−1 **A special data value.** The numeric value field is 0, and the string value field may have one of two special values: BOT or EOD. BOT means "Beginning Of Data" and EOD means "End Of Data."

```
-1,0
BOT
```

```
-1,0
EOD
```

−2 **The data item is an origin specifier.** This item can optionally give the cell address of the data following it.

```
-2,0
1:4
```

−3 **Entry-level display formatting.** This is used to set the formatting of a cell at the entry level (equivalent to /Format, Entry). This will set the format for the cell that contains the **preceding** data item.

```
-3,0
$
```

−4 **The data is a formula.** The formula is specified by the formula string. The numeric value of the formula is calculated when the formula is loaded into the SuperCalc program.

```
−4,0
A1+B1*4
```

−5 **The data is a repeat count.** This repeat count is the number of times that the previous data item is repeated after the first occurrence. The string value is usually "R" for repeat count.

```
−5,4
R
```

Numeric Values

The numeric value may be signed (+ or −) and may contain a decimal point. One or more blanks may precede or follow the numeric value. If the data value contains an exponent of a power of 10, the value is followed by the letter "E" and the signed or unsigned exponent.

The numeric value is the only place that the SDF-format file allows a non-integer value. When the data value is numeric, the string value field contains one of the values described below (usually "V").

String Values

When the type indicator is 1, the data is a string. The string value may also be used for other type-indicator values such as a formula for a type indicator of −4, an error condition for a type indicator of 0, etc. This string contains no control characters and need not contain any quotation marks.

If the string contains blanks or commas, it **must** be enclosed in quotation marks. If the string is null, the string value field contains quotation marks (" ").

When data is numeric (a type indicator of 0), the string value has one of the following uppercase notations not containing quotation marks:

V The data is a numeric value.

NA The data is not available. Indicates that the value requested in not available. The numeric value is 0.

ERROR The data is the result of an invalid calculation (such as dividing by 0). The numeric value is 0.

Special Data Values

If the type indicator field is −1, the data is one of two special data types, BOT or EOD. BOT flags the beginning of a row (equivalent of the carriage return in a CSV file). EOD flags the end of the last row (the end of the file). **No** data is interpreted past the EOD marker.

Type Indicator Examples

NUMERIC DATA

0 specifies numeric-type data. Values can include signs (+ or −), numbers, and a decimal point. The type of data is defined by the value indicator.

When the value indicator is:

V The numeric data has a decimal value.

NA The value for the cell is not available.

NULL The value of the cell is null or unoccupied.

ERROR The value is in error, perhaps due to an invalid calculation such as dividing by 0.

Format:

0,numeric value
value indicator

Example:

0,123.45	**0,0**
V	**ERROR**

STRING DATA

1 indicates that the data item is a string. The string-type value specifies the type of string present. If the string-type value is:

0 The string is ordinary text.

1 The string is repeating text (unique to SuperCalc).

The string value may be **optionally** enclosed in double quotation marks (" ").

Format:

1,string value indicator
string value

Example:

1,0	**1,1**
"This is text"	=

SPECIAL DATA

−1 indicates a special data value. The number value is 0. The two specifier strings recognized are BOTand EOD. BOT marks the beginning of a row, and EOD marks the end of the data section and should be the last item in the file.

Format:

```
−1,0
specifier string
```

Example:

```
−1,0        −1,0
BOT         EOD
```

ORIGIN SPECIFIER

−2 indicates that the file's flow of data is to be altered. The next data item should start being entered at the SuperCalc cell specified in this data item. The numeric value is 0. The string value contains the cell specification. The specification is two numbers separated by a colon (:).

The first number is the column location (1–63). Although in SuperCalc the columns are specified by an alphabetic notation, they must be converted to their numeric equivalents here. The second number is the row number (1–254) as used in SuperCalc.

Format:

−2,0
origin specifier

Examples:

```
−2,0          −2,0
3:20          28:10
```

Column	Value
A	1
Z	26
AB	28

DISPLAY FORMAT

−3 indicates that the string value is to be a formatting specification for the **previous** cell. The numeric value is 0. The formatting information is the same as for the GLOBAL Display item in the HEADER section (I, $, TL, etc.). If there is no previous data item, an error occurs.

Format:

format string

Example:

```
−3,0          −3,0
$TL           ITR
```

FORMULA

−4 indicates that the data item is a formula. The formula must be a standard SuperCalc formula. The numeric value is 0 and is ignored. The new numeric value of the cell is calculated when the file is loaded into the SuperCalc program.

Format:

```
        −4,0
       formula
```

Example:

```
−4,0          −4,0
A1+N4         IF(A1 = 0, B1, C2/100)
```

REPEAT COUNT

−5 indicates a repeat count. The **previous** data item is to be repeated into the next sequential cells for the number of times specified by the numeric value. If there is no previous data item, an error occurs.

Format:

```
     +5,repcount
         R
```

Example:

```
−5,4          −5,20
R             R
```

This is useful especially for padding a section of a worksheet with either null data or zeros. An example of filling a line of the worksheet with 10 zeros is as follows:

```
−1,0
BOT
0,0
V
−5,10
R
```

Standard or Default Settings

SuperCalc uses standard settings for display and formatting and standard modes of reference. These are also called default settings or modes. You can change these settings by choosing among the available options described earlier. For convenience, here is a list of the standard settings and standard modes.

You can change the following default settings by using the **/FORMAT** command:

Column Width: 9

Numeric Display:
Right-justified.

Standard numeric format. (Cells that contain formulas will have their values displayed; if the number is too large to fit into the column, the number will be displayed in scientific notation.)

Text Display:
Left-justified.

You can change the following default settings by using the **/GLOBAL** command:

Border Display: Row numbers (1–254) and column designations (A–BK) are always displayed. (When the screen is split, the row numbers and column designations are displayed for both windows.)

Calculation: Automatic calculation takes place upon reception of new or altered data followed by **RETURN**.

Order of Calculation: Calculation is performed by rows, from left to right and top to bottom.

Numeric Display: Standard numeric display. (Cells that contain formulas will have their values displayed.)

Tab Mode: The tab mode is inactive: The cursor advances to the next cell in the current cursor direction.

Automatic Cursor Advancing: Auto-advance mode is active. The cursor will advance to the next cell in the current cursor direction after data entry followed by **RETURN**.

Additional Standard Operations: When you execute a /COPY or /REPLICATE command, formulas with references to other cells automatically adjust to their new locations unless you choose an option provided for these commands.

CP/M Plus

This section describes the CP/M Plus utility programs supplied with your Osborne Executive Computer. The commands are in alphabetical order. A summary of each command is given, followed by a short explanation of its operation, format, and examples of its use. Sophisticated commands used mainly for programming, such as SID and MAC, are described more fully in separate publications.

If you are familiar with older versions of CP/M, you will notice that some utilities have been replaced. For instance, MAC replaces ASM; SHOW and DIR now include the previous STAT functions, and SID replaces DDT.

The beginning of this section describes the parts of file specifications used in CP/M Plus command lines. There are several parts in a file specification; to avoid confusion, each part is named. The four parts of a file specification are defined as follows:

Drive Specifier The disk drive (A:, B:, C:, etc.) that contains the file or group of files to which you are referring.

File Name The one- to eight-character name of a file or group of files.

File Type The optional one- to three-character category of a file or group of files. If the file type is present, a period must separate it from the file name.

Password The optional one- to eight-character password which allows you to protect your files. It follows the file type (or file name if no file type is assigned), and is separated from it by a semicolon.

If you do not include a drive specifier, CP/M Plus assumes you mean the default drive. The current default drive appears in the CP/M Plus command prompt A>.

A file specification defines a particular file or group of files in the directory of the on-line disk named by the drive specifier. For example:

```
B:MYFILE.DAT;PASSWORD
```

is a file specification that indicates drive B:, file name MYFILE, file type .DAT, and password ;PASSWORD. The term "file specification" is often abbreviated "filespec," especially in command format statements.

Some commands accept wildcards in the file name and file type parts of the command tail. For example:

```
B:MY*.A??
```

is a file specification with drive specifier B:, file name MY*, and file type A??. No password is present. This file specification matches any file in the directory with a file name beginning with "MY" and file type beginning in "A."

Altogether, the file specification is represented like this:

```
d:filename.typ;password
```

In the above form, "d:" represents the optional drive specifier, "filename" represents the one- to eight-character file name, ".typ" represents the optional one- to three-character file type, and ";password" represents the optional one- to eight-character password. The command format descriptions in this section use the term "filespec" to indicate any valid file specification. The following list illustrates all valid combinations of the elements of a CP/M Plus file specification.

filename
filename.typ
filename;password
filename.typ;password
d:filename
d:filename.typ
d:filename;password
d:filename.typ:password

Some characters have a special meaning and may not be used in file specifications except as indicated below:

Character	Meaning
< = , ! | > [ ] TAB, space, <cr> (carriage return)	File specification delimiters
:	Drive delimiter in file specification
.	File type delimiter in file specification
;	Password delimiter in file specification
*** ?**	Wildcard characters in a wildcard file specification
<> & ! | \ + –	Option list delimiters reserved for future use
[ ]	Option list delimiters for global and local options
( )	Delimiters for multiple modifiers inside square brackets for options that have modifiers
/ $	Option delimiters in a command line
;	Comment or password delimiter

Previous CP/M usage has already established several standard file types. Following is a list of the most common file types with a short description of each:

File Type	**Meaning**
.ASM	Assembler source file
.BAS	BASIC source program file
.COM	8080, 8085, or equivalent machine-language program file
.HLP	Help message file
.SUB	File containing a list of commands to be executed by SUBMIT
.$$$	Temporary file

In previous sections, we covered command keywords, control characters, default drive, and wildcards. Now, you also understand the terms filespec, drive specifier, file name, file type and password. These concepts give you the background necessary to compose complete CP/M Plus command lines.

How Formats Are Described

In this section, CP/M Plus commands appear in alphabetical order. The specific forms of each command are described. This section also describes the optional parts of the command line and other format notation. Each description is composed of the following parts:

Keyword The description begins with the command keyword in uppercase.

Explanation The Explanation section defines the general use of the command keyword, and points out exceptions and special cases. The explanation sometimes includes tables or lists of options that you can use in the command line.

Format The Format section gives you one or more general forms to follow when you compose the command line.

Example Examples on the various uses of the command are shown.

The following table defines the special symbols and abbreviations used in Format lines.

Command Format Conventions

Symbol	**Meaning**
DIR	Indicates Directory Attribute.
RO	Indicates Read-Only Attribute.
RW	Indicates Read-Write Attribute.
SYS	Indicates System Attribute.
[ ]	Items in square brackets are options or an option list. If the right bracket is the last character on the command line, it can be omitted.
CTRL	Represents the CTRL key on your keyboard. To generate a control character such as CTRL-C, depress and hold the CTRL key, then press C.
<cr>	Indicates a RETURN.

* Wildcard character: replaces all or part of a file name and/or file type.

? Wildcard character: replaces any single character in the same position of a file name or file type.

Let's look at some examples of format notation. The CP/M Plus DIR (DIRectory) command displays the names of files cataloged in the disk directory and optionally displays other information about the files.

DIR alone is a valid command and the command tail following the keyword DIR is optional. You can include a file specification, a drive specification, or just the options in the command line. Therefore:

```
DIR<cr>
DIR filespec<cr>
DIR d:<cr>
DIR [ RO] <cr>
```

are all valid commands. Furthermore, the file or drive specification can be followed by another optional value selected from one of the following sample list of DIR options.

```
RO
RW
DIR
SYS
```

Therefore:

```
DIR filespec [ RO] <cr>
DIR filespec [ RO SYS] <cr>
DIR filespec [ RW,SYS] <cr>
```

are valid commands.

The DIR command also accepts wildcards in the file specification. Using this format information, you can construct several valid command lines:

```
DIR
DIR X.PAS
DIR X.PAS [ RO]
DIR X.PAS [ SYS]
DIR *.PAS
DIR *.* [ RW]
DIR X.* [ DIR]
```

Another example of command format notation: The CP/M Plus command PIP (Peripheral Interchange Program) is the file copy program. PIP can copy information from the disk to the screen or printer. PIP can also combine two or more files into one longer file, or rename files after copying them. Let's look at one of the formats of the PIP command line for another example of how to use command line notation.

For this example, the destination filespec is defined as the file specification or peripheral device (printer, for example) that receives data. With PIP, the destination filespec comes first, then the source filespec. PIP also accepts wildcards in both the file name and file type. (See the PIP Command discussion further on in this section for details regarding other capabilities of PIP.) Many valid command lines can be derived from this format. Some of them are shown below:

```
PIP NEWFILE.DAT=OLDFILE.DAT
PIP B:=A:THISFILE.DAT
PIP B:X.BAS=Y.BAS,Z.BAS
PIP X.BAS=A.BAS,B.BAS,C.BAS
PIP B:=A:*.BAK
PIP B:=A:*.*
```

CP/M Plus Commands

Following are the CP/M Plus commands listed in alphabetical order.

COPYSYS—*copies the CP/M Plus Operating System*

The COPYSYS command copies the CP/M Plus operating system and its companion system file to a specified diskette. An area of each diskette is reserved especially for the CP/M Plus operating system. COPYSYS transfers a copy of the operating system to a diskette without affecting any other information that exists on the diskette, it just writes it onto the reserved system tracks. A special system file is also transferred.

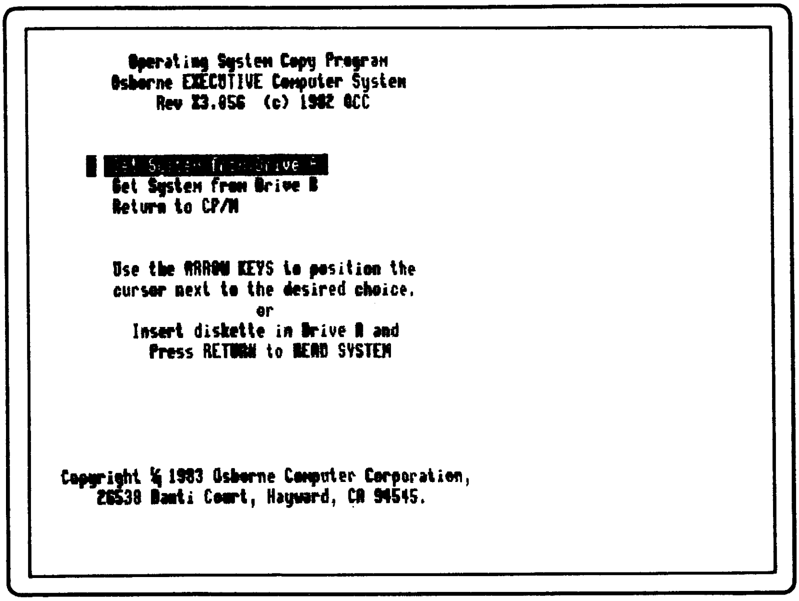
Format:

```
COPYSYS
```

Here is the procedure required in order to transfer a copy of the CP/M Plus operating system to a diskette.

1. Type COPYSYS and press the RETURN KEY.

   Once the COPYSYS program is initiated, the Executive will lead you through the steps needed to save the CP/M Plus operating system onto a diskette. Here are the messages displayed and the steps required of you:

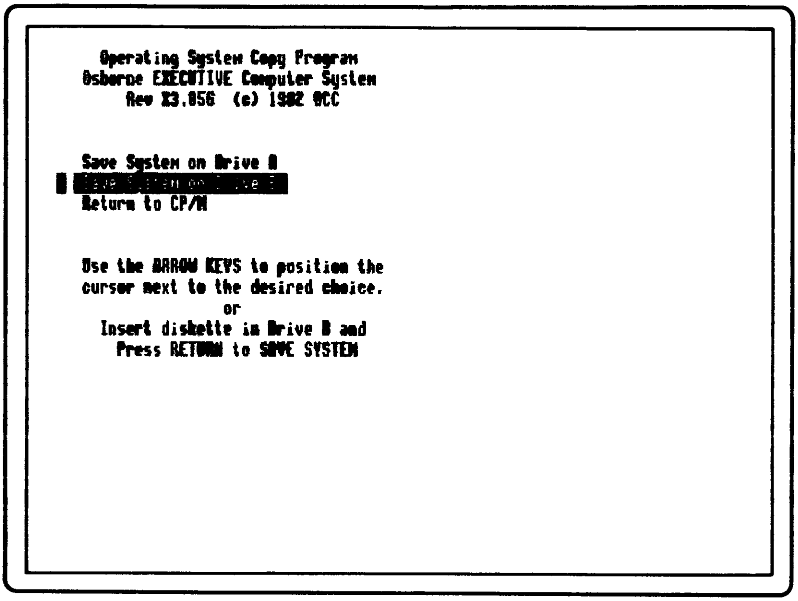


```
        Operating System Copy Program
      Osborne EXECUTIVE Computer System
           Rev X3.056  (c) 1982 OCC


  ▌ Get System from Drive A
    Get System from Drive B
    Return to CP/M


    Use the ARROW KEYS to position the
    cursor next to the desired choice.
                    or
      Insert diskette in Drive A and
        Press RETURN to READ SYSTEM




Copyright © 1983 Osborne Computer Corporation,
   26538 Danti Court, Hayward, CA 94545.
```

2. Select the source of the CP/M Plus operating system. Use the up and down arrow keys to position the cursor next to the desired location, then press RETURN. A message indicates that the CP/M Plus operating system is ready to be copied to the destination diskette.

```
System read successfully.

Press RETURN to continue.
```

3. Press RETURN to continue. The next prompt is:



```
     Operating System Copy Program
   Osborne EXECUTIVE Computer System
       Rev X3.056  (c) 1982 OCC


   Save System on Drive A
 █ Save System on Drive B
   Return to CP/M


   Use the ARROW KEYS to position the
   cursor next to the desired choice.
                  or
     Insert diskette in Drive B and
       Press RETURN to SAVE SYSTEM
```

4. Use the up and down arrows to position the cursor and indicate where you want the operating system to be saved. Make sure that the diskette where the copy will be transferred is in the specified drive, then press RETURN.

If the operating system and its companion system file are already on the diskette that you are saving to, this message is displayed:

```
CPM.SYS is already on drive B.

Do you want to overwrite it (y/n)?
```

You must decide whether to replace the operating system or not. Type Y for yes and N for no. When the operating system has been successfully copied to the specified drive, this message appears:

```
System copied successfully.

Press RETURN to continue.
```

5. Press RETURN to continue. If you want to copy the operating system to another diskette, press RETURN and again repeat the procedure. To leave the COPYSYS program, position the cursor beside the last option (Return to CP/M), and press RETURN.

Function complete

COPYSYS copies the system file CPM3.SYS from the default drive A to the new diskette. Remember, the CP/M Plus operating system requires the file CPM3.SYS on the system disk.

DATE—*sets or displays date and time*

The DATE command is a transient utility that lets you set or display the date and time of day. When you start up the CP/M Plus Utility diskette, the date and time are set to the date when your CP/M Plus Utility diskette was created. Use DATE to change this initial value to the current date and time.

Format:

```
DATE
```

Example:

```
DATE<cr>
```

This example displays the current day, date and time, for example:

```
Fri 08/13/82 09:15:37
```

Format:

```
DATE SET
```

This form of the DATE command prompts you through the steps necessary to set the time and date. The date and time format is entered as follows:

```
MM/DD/YY HH:MM:SS
```

where:

MM is a month value in the range 1 to 12.
DD is a day value in the range 1 to 31.
YY is the two-digit year value relative to 1900.
HH is the hour value in the range of 0 to 23.
MM is the minute value in the range of 0 to 59.
SS is the second value in the range of 0 to 59.

The system checks the validity of the date and time entry and determines the day of the week for the date entered. To keep the current system date or time, simply press carriage return.

Example:

DATE SET<cr>

In this example, the system prompts with:

```
Enter today's date (MM/DD/YY):
```

Enter the date or press the carriage return to skip. Then the system prompts with:

```
Enter the time (HH:MM:SS):
```

Enter the time or press the carriage return to skip. The system prompts with:

```
Press any key to set time
```

You then press any key to set the time to the exact value you specified. CP/M Plus then maintains the correct date and time automatically, as long as your computer is turned on.

Format:

DATE MM/DD/YY HH:MM:SS

This is the shorthand format for entering the date and time without going through the prompts.

Example:

DATE 08/14/82 10:30:00<cr>

The system responds with:

```
Press any key to set time
```

As soon as you press any key, DATE initializes the system time to the value specified in the command, and displays the day, date and time:

```
Sat 08/14/82 10:30:00
```

Format:

DATE CONTINUOUS

The above form of the DATE command displays the current date and time, updated each second. The CONTINUOUS option allows continuous display of the date and time, and can be abbreviated to C. You can stop the continuous display by pressing any key.

Example:

DATE C<cr>

This command displays the date and time continuously until you press any key.

DEVICE—*defines physical-to-logical device assignments*

The DEVICE command is a CP/M Plus transient utility that displays current assignments of system logical devices to physical devices. DEVICE allows you to assign CP/M Plus logical devices to peripheral devices attached to the computer. The DEVICE command can also be used to set the communications protocol and baud rate of a peripheral device, and display or set the current console screen size.

The following five logical devices are supported.

CONIN:	**Console (keyboard) input**
CONOUT:	**Console (display) output**
AUXIN:	**Auxiliary Input (such as tape reader)**
AUXOUT:	**Auxiliary Output (such as tape punch)**
LST:	**Printer Output**

These logical devices are also known by the following names:

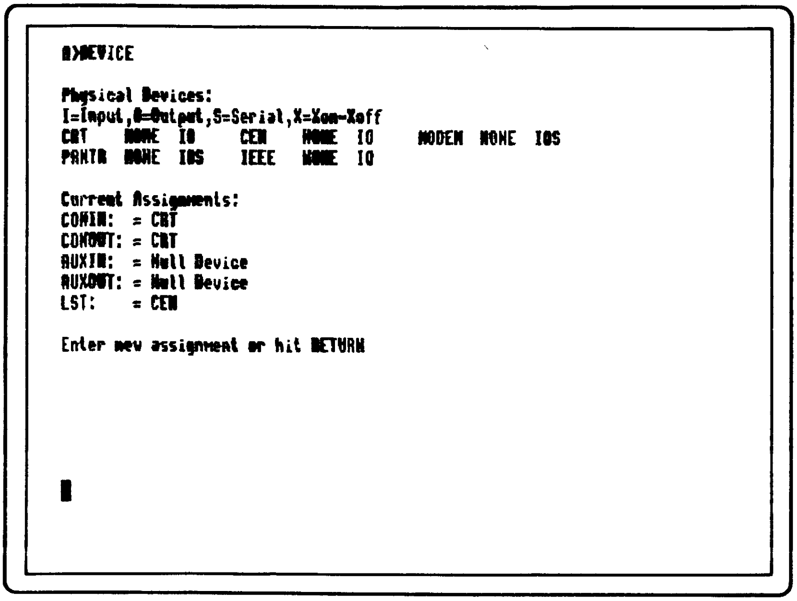
CON:	**(for CONIN: and CONOUT:)**
CONSOLE:	**(for CONIN: and CONOUT:)**
KEYBOARD:	**(for CONIN:)**
AUX:	**(for AUXIN: and AUXOUT:)**
AUXILIARY:	**(for AUXIN: and AUXOUT:)**
PRINTER:	**(for LST:)**

You can use the DEVICE command to display the names and attributes of the physical devices for the Osborne Executive Computer.

Examples:

```
DEVICE<cr>
```

The above command displays the physical devices and current assignments of the logical devices in the system. The following is a sample response:



```
A>DEVICE

Physical Devices:
I=Input,O=Output,S=Serial,X=Xon-Xoff
CRT    NONE  IO    CEN   NONE  IO    MODEM  NONE  IOS
PRNTR  NONE  IOS   IEEE  NONE  IO

Current Assignments:
CONIN:  = CRT
CONOUT: = CRT
AUXIN:  = Null Device
AUXOUT: = Null Device
LST:    = CEN

Enter new assignment or hit RETURN
```

The system prompts for a new device assignment. You can enter any valid device assignment. If you do not want to change any device assignments, press the RETURN key.

The following forms of the DEVICE command display the names and attributes of the physical devices and the current assignments of the logical devices in the system.

DEVICE NAMES<cr>

Lists the physical devices with a summary of the device characteristics.

DEVICE VALUES<cr>

Displays the current logical device assignments.

```
DEVICE CRT<cr>
```

Displays the attributes of the physical device CRT:

```
DEVICE CON<cr>
```

Displays the assignment of the logical device CON:

You can use DEVICE to assign a logical device to one or more physical devices, or disconnect a logical device from any physical device.

```
DEVICE CONOUT:=LPT,CRT<cr>
```

Assigns the system console output (CONOUT:) to the printer (LPT:) and the screen (CRT:).

```
DEVICE AUXIN:=CRT2 [XON,9600]<cr>
```

Assigns the auxiliary logical input device (AUXIN:) to the physical device CRT using protocol XON/XOFF and sets the baud rate for the device at 9600.

```
DEVICE LST:=NULL<cr>
```

Disconnects the list output logical device (LST:).

The following forms of the DEVICE command set the attributes of the physical device specified in the command

DEVICE Physical Device Attributes

Attributes and Their Meanings

XON Refers to the XON/XOFF communications protocol. This protocol uses two special characters in the ASCII character set called XON and XOFF. XON signals Transmission On, and XOFF signals Transmission Off. Before each character is output from the computer to the peripheral device, the computer checks to see if there is any incoming data from the peripheral. If the incoming character is XOFF, the computer suspends all further output until it receives an XON from the device, indicating that the device is again ready to receive more data.

NOXON Indicates no protocol and the computer sends data to the device whether or not the device is ready to receive it.

baud rate Is the speed of the device. The system accepts the following baud rates:

50	75	110	134
150	300	600	1200
1800	2400	3600	4800
7200	9600	19200	

DEVICE LPT [XON,9600] <cr>

Sets the XON/XOFF protocol for the physical device LPT and sets the baud rate to 9600.

The following forms of the DEVICE command display or set the current console size.

DEVICE CONSOLE [PAGE] <cr>

Displays the current console page width in columns and length in lines.

DEVICE CONSOLE [COLUMNS=40 LINES=16]<cr>

Sets the screen size to 40 columns and 16 lines.

DIR—*lists a directory of specified files and attributes*

The DIR command displays the names of files and the attributes associated with the files. DIR and DIRSYS are resident utilities; DIR with options is a transient utility.

Format:

DIR filespec

The DIR command lists the names of all files designated as directory (DIR attribute) files on the current default drive, under the currently active user number. If a drive is specified, the DIR command displays the names of all DIR files in the current user area on the specified drive. To locate a specific file, follow the DIR command with the filespec of the file being searched for. You can use the wildcard characters * and ? in the filespec; all filespecs that satisfy the match are listed.

Examples:

DIR<cr>

Displays all DIR files cataloged in user 0 on the default drive A.

DIR B:<cr>

Displays all DIR files for user 0 on drive B.

DIR B:X.BAS<cr>

Displays the name X.BAS if the file X.BAS is present on drive B.

DIR X*.C?D<cr>

Displays all DIR files for user 0 on drive A whose file name begins with the letter X, and whose three-character file type contains the first character C and last character D.

Format:

DIRSYS filespec

The DIRSYS works just like the DIR command except it displays the names of files in the current user number that have the System (SYS) attribute. However, even though you can run System (SYS) files that are stored in user 0 from any other user number, DIRSYS only displays user 0 files if the current user number is 0. DIRSYS accepts wildcards in the file specification. You can abbreviate the DIRSYS command to DIRS.

If no file names match the file specification, or if no files are cataloged in the directory of the specified drive, the DIR or DIRSYS command displays the message:

```
No File
```

If system (SYS) files match the file specification, DIR displays the message:

```
SYSTEM FILE(S) EXIST
```

If nonsystem (DIR) files match the file specification, DIRSYS displays the message:

```
NON-SYSTEM FILES(S) EXIST
```

The DIR command pauses after filling the screen. Press any key to continue the display.

NOTE

You can use the DEVICE command to change the number of columns displayed by DIR or DIRSYS.

Format:

DIR filespec [options]

The DIR command with options is an enhanced version of DIR. DIR command options display files in a variety of ways. For example, DIR options can search for files on any or all drives or for any or all user numbers.

DIR allows the option list to occur anywhere on the command tail; however, only one option list is allowed.

Options must be enclosed in square brackets. Multiple options can be specified, provided they are separated by commas or spaces. Options can be abbreviated to only one or two letters if the abbreviation unambiguously identifies the option.

If a directory listing overflows the screen, DIR automatically halts the display. Press any key to restart the display. Following are the DIR options that may be enclosed in brackets.

Options and Their Functions

ATT Displays the user-definable attributes F1, F2, F3, and F4 as 1, 2, 3, and 4, respectively.

DATE Displays files with date and time stamps. If time and date stamping is not active, DIR displays the message:

```
Date and Time Stamping Inactive
```

DIR Displays only files that have the DIR attribute.

DRIVE=ALL Displays files on all accessed drives. DISK is also acceptable in place of DRIVE in all the DRIVE options.

Example:

DIR [DRIVE=ALL] <cr>

Displays all the files under user 0 on all the drives in the drive search chain. (See the SETDEF command.)

DRIVE=(A,B,C,...,P) Displays files on the drives specified.

Example:

DIR [DRIVE=B] <cr>

Displays all the files on drive B.

DRIVE=d Displays files on the drive specified by DIR B:

Example:

DIR [DRIVE=(A,B)] <cr>

Displays all the files under user 0 on drive A and B.

EXCLUDE Displays the files on the default drive in the user areas that do not match the files specified in the command line.

Example:

DIR [EXCLUDE] *.COM<cr_

Lists all the files on the default drive and user 0 that do not have a file type of .COM.

FF Sends an initial form feed to the printer device if the printer has been activated by CTRL-P. If the LENGTH=*n* option is also specified, DIR issues a form feed every *n* lines. Otherwise, the FF option deactivates the default-paged output display.

Example:

DIR [DRIVE=B,FF] <cr>

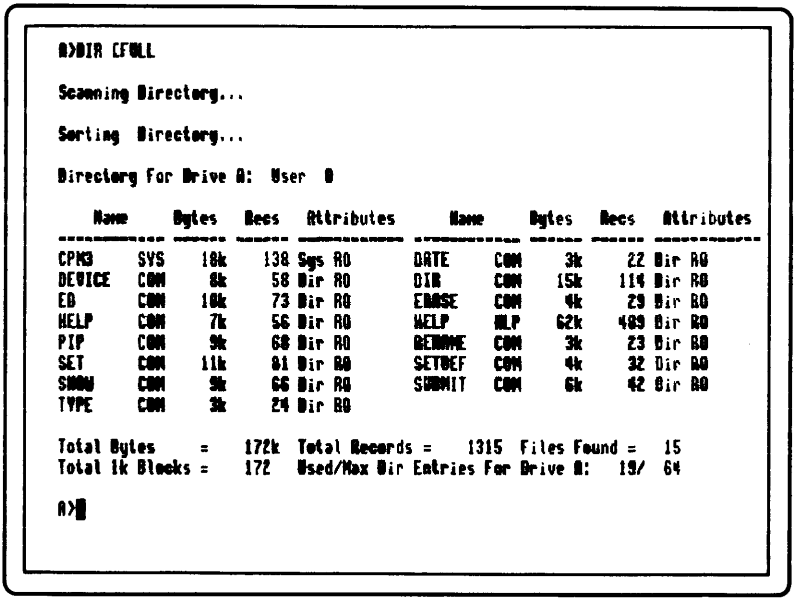
DIR sends a form feed to the printer before displaying the files on drive B.

FULL Shows the name and size of the specified file(s). The size is shown as the amount of space in kilobytes and the number of 128-byte records allocated to the file. FULL also shows the attributes of the file. (See the SETcommand for description of file attributes.) If there is a directory label on the drive, DIR shows the password-protection mode and the time stamps. If there is no directory label, DIR displays two file entries on a line, omitting the password and time stamp columns. The display is alphabetically sorted. FULL is the default output format for display.

Example:

DIR B: [FULL]<cr>

The following is sample output of the option display format shown in the above example:



```
A>DIR [FULL

Scanning Directory...

Sorting  Directory...

Directory For Drive A:  User  0

    Name       Bytes   Recs   Attributes      Name       Bytes   Recs   Attributes
------------ ------ ------ ------------ ------------ ------ ------ ------------
CPM3     SYS    18k    138 Sys RO       DATE     COM     3k     22 Dir RO
DEVICE   COM     8k     58 Dir RO       DIR      COM    15k    114 Dir RO
ED       COM    10k     73 Dir RO       ERASE    COM     4k     29 Dir RO
HELP     COM     7k     56 Dir RO       HELP     HLP    62k    489 Dir RO
PIP      COM     9k     68 Dir RO       RENAME   COM     3k     23 Dir RO
SET      COM    11k     81 Dir RO       SETDEF   COM     4k     32 Dir RO
SHOW     COM     9k     66 Dir RO       SUBMIT   COM     6k     42 Dir RO
TYPE     COM     3k     24 Dir RO

Total Bytes     =    172k  Total Records =    1315  Files Found =   15
Total 1k Blocks =    172   Used/Max Dir Entries For Drive A:   19/  64

A>█
```

LENGTH=*n* Displays *n* lines of output before inserting a table heading. *n* must be in the range between 5 and 65536. The default length is one full screen of information.

MESSAGE Displays the names of the specified drives and user numbers it is currently searching. If there are no files in the specified locations, DIR displays the "File Not Found" message.

Example:

```
DIR X.SUB [MESSAGE,USER=ALL,DRIVE=ALL]<cr>
```

Searches all drives under each user number for X.SUB. During the search, DIR displays the drives and user numbers.

NOPAGE Continuously scrolls information by on the screen. Does not wait for you to press a key to restart the scrolling movements.

NOSORT Displays files in the order it finds them on the disk. If this option is not included, DIR displays the files in alphabetical sort.

RO Displays only the files that have Read-Only attribute.

RW Displays only the files that are set to Read-Write.

Example:

```
DIR B: [RW,SYS]<cr>
```

The above example displays all the files on drive B with Read-Write and SYS attributes.

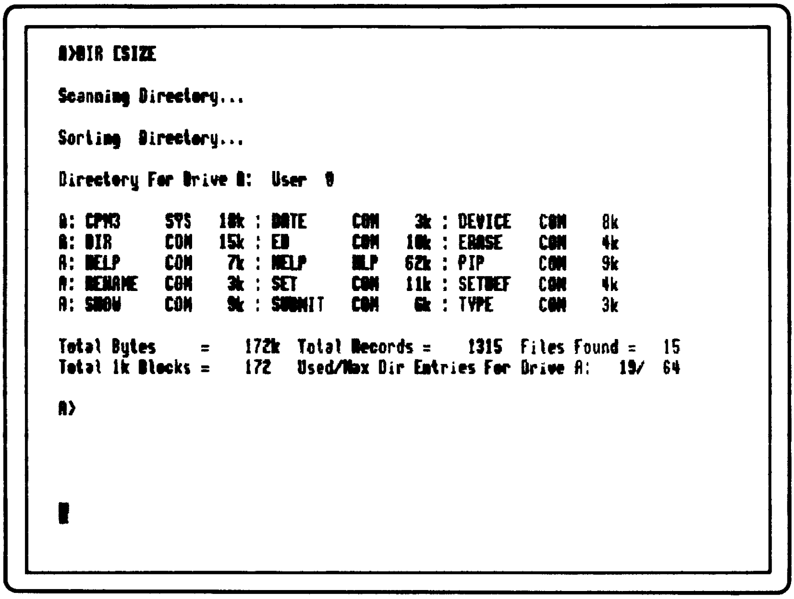
SIZE Displays the file name and file size in kilobytes.

Example:

```
DIR [SIZE]<cr>
```

The example above instructs DIR to list each file that resides on drive B with its size in kilobytes.

The following is a sample output of the [SIZE] option.



```
A>DIR [SIZE

Scanning Directory...

Sorting  Directory...

Directory For Drive A:  User  0

A: CPM3     SYS   18k : DATE     COM    3k : DEVICE   COM    8k
A: DIR      COM   15k : ED       COM   10k : ERASE    COM    4k
A: HELP     COM    7k : HELP     HLP   62k : PIP      COM    9k
A: RENAME   COM    3k : SET      COM   11k : SETDEF   COM    4k
A: SHOW     COM    9k : SUBMIT   COM    6k : TYPE     COM    3k

Total Bytes     =    172k  Total Records =    1315  Files Found =   15
Total 1k Blocks =    172   Used/Max Dir Entries For Drive A:    19/  64

A>
```

Both the FULL option and the SIZE option follow their display with two lines of totals. The first line displays the total number of kilobytes, the total number of records, and the total number of files for that drive and user area. The second line displays the total number of 1K blocks needed to store the listed files. The number of 1K blocks shows the amount of storage needed to store the files on a single-density diskette, or on any drive that has a block size of one kilobyte. The second line also shows the number of directory entries used per number of directory entries available on the drive. These totals are suppressed if only one file is found.

SYS Displays only the files that have the SYS attribute.

Examples:

DIR SYS<cr>

Displays all files for user 0 on drive A that have the system (SYS) attribute.

DIRS *.COM<cr>

This abbreviated form of the DIRSYS command displays all SYS files with file type COM on the default drive A.

DIR [USER=ALL,DRIVE=ALL,SYS] *.PLI *.COM *.ASM

Instructs DIR to list all the system files of type PLI, COM, and ASM on the system in the currently active drives for all the user numbers on the drives.

USER=ALL Displays all files under all the user numbers for the default drive.

Examples:

DIR B: [USER=ALL]<cr>

Displays all the files under each user number (0–15) on drive B.

DIR [DRIVE=ALL USER=ALL] TESTFILE.BOB<cr>

Above instructs DIR to display the file name TESTFILE.BOB if it is found on any logged-in drive for any user number.

USER=*n* Displays the files under the user number specified by *n*.

Example:

DIR [USER=2]<cr>

Displays all the files under user 2 on the default drive.

USER=(0,1,...,15) Displays files under the user numbers specified.

Example:

DIR B: [USER=(3,4,10)]<cr>

Displays all the files under user numbers 3, 4, and 10 on drive B.

DUMP—*displays disk file in hex*

DUMP displays the contents of the specified file in hexadecimal and ASCII format. Each line begins with an absolute byte address displayed in hexadecimal. This is followed by the next 16 consecutive bytes of the file, beginning at that address, also in hex.

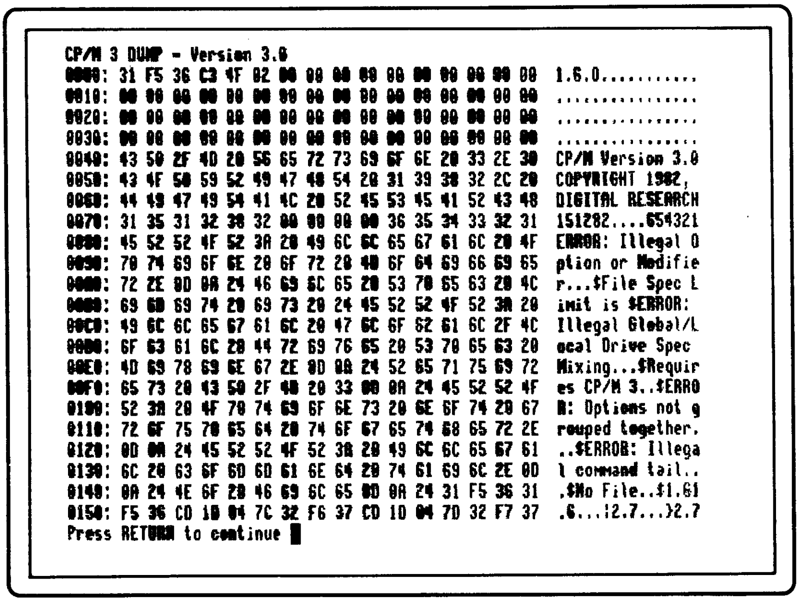
Format:

DUMP filespec

Example:

DUMP ABC.TEX<cr>

A sample console output looks like this:



```
CP/M 3 DUMP - Version 3.0
0000: 31 F5 36 C3 4F 02 00 00 00 00 00 00 00 00 00 00  1.6.0...........
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040: 43 50 2F 4D 20 56 65 72 73 69 6F 6E 20 33 2E 30  CP/M Version 3.0
0050: 43 4F 50 59 52 49 47 48 54 20 31 39 38 32 2C 20  COPYRIGHT 1982,
0060: 44 49 47 49 54 41 4C 20 52 45 53 45 41 52 43 48  DIGITAL RESEARCH
0070: 31 35 31 32 38 32 00 00 00 00 36 35 34 33 32 31  151282....654321
0080: 45 52 52 4F 52 3A 20 49 6C 6C 65 67 61 6C 20 4F  ERROR: Illegal O
0090: 70 74 69 6F 6E 20 6F 72 20 4D 6F 64 69 66 69 65  ption or Modifie
00A0: 72 2E 0D 0A 24 46 69 6C 65 20 53 70 65 63 20 4C  r...$File Spec L
00B0: 69 6D 69 74 20 69 73 20 24 45 52 52 4F 52 3A 20  imit is $ERROR:
00C0: 49 6C 6C 65 67 61 6C 20 47 6C 6F 62 61 6C 2F 4C  Illegal Global/L
00D0: 6F 63 61 6C 20 44 72 69 76 65 20 53 70 65 63 20  ocal Drive Spec
00E0: 4D 69 78 69 6E 67 2E 0D 0A 24 52 65 71 75 69 72  Mixing...$Requir
00F0: 65 73 20 43 50 2F 4D 20 33 0D 0A 24 45 52 52 4F  es CP/M 3..$ERRO
0100: 52 3A 20 4F 70 74 69 6F 6E 73 20 6E 6F 74 20 67  R: Options not g
0110: 72 6F 75 70 65 64 20 74 6F 67 65 74 68 65 72 2E  rouped together.
0120: 0D 0A 24 45 52 52 4F 52 3A 20 49 6C 6C 65 67 61  ..$ERROR: Illega
0130: 6C 20 63 6F 6D 6D 61 6E 64 20 74 61 69 6C 2E 0D  l command tail..
0140: 0A 24 4E 6F 20 46 69 6C 65 0D 0A 24 31 F5 36 31  .$No File..$1.61
0150: F5 36 CD 10 04 7C 32 F6 37 CD 10 04 7D 32 F7 37  .6...|2.7...}2.7
Press RETURN to continue █
```

ED—*CP/M Plus line editor*

The ED editing utility lets you create and edit disk files. ED is a line-oriented context editor. This means that you create and change character files on a line-by-line basis, or by referencing individual characters within a line.

Format:

ED filespec filespec

If you do not include the command tail "filespec filespec," as shown above, ED prompts you for the input and output filespecs as follows:

```
Enter Input File:
```

After you enter the input filespec, ED prompts again:

```
Enter Output File:
```

If you want ED to delete the input file and replace it with the edited file at the end of the edit, enter only RETURN. Otherwise, if you want to preserve the input file, enter a filespec for the output file.

If the output filespec consists only of a drive specifier, ED assumes the second filespec to be the same as the first.

If the file given by the filespec is not present on the disk, ED creates the file and displays the message:

```
NEW FILE
```

If you issue an ED command line that contains a file of type .BAK, ED creates and saves your new edited version of the .BAK file, and deletes your source file, leaving no backup. If you want to save the original .BAK file, use the RENAME command first to change the file type from .BAK, so that ED can rename it to .BAK as your backup file.

If you include the optional second filespec and give it the same name as the first filespec, ED again creates and saves your new edited version of the output filespec, but has to delete the original input filespec because it has the same name as the output file. You cannot, of course, have two files with the same name in the same user number on the same drive.

If the file given by the first filespec is already present, you must issue the A command to read portions of the file into the buffer. If the size of the file does not exceed the size of the buffer, the command A reads the entire file to the buffer.

The ED utility uses a portion of your user memory as the active text buffer where you add, delete, or alter the characters in the file. You use the A command to read all or a portion of the file into the buffer. Use the W or E command (described below) to write all or a portion of the characters from the buffer back to the file. A virtual character pointer, called CP, can be positioned anywhere in the text buffer.

You interact with the ED utility in either command or insert mode. ED displays the * prompt on the screen when it is in command mode. When the * appears, you can enter a single-letter command that reads text from the buffer, moves the character pointer, or changes the ED mode of operation. When in command mode, you can use the line-editing characters ^C, ^E, ^H, ^U, ^X, and ← to edit your input. In insert mode, however, you can use only ^H, ^U, ^X, and ←. Here is a summary of the ED commands:

Ed Commands and Their Actions

***n*A** Appends *n* lines from original file to memory buffer.

0A Appends file until buffer is one-half full.

#A Appends file until buffer is full (or end of file).

B, −B Move character pointer to beginning (B) or end (−B) of buffer.

***n*C, −*n*C** Move character pointer *n* characters forward (C) or back (−C) through buffer.

***n*D, −*n*D** Delete *n* characters before (−D) or from (D) the character pointer.

E Saves new file and returns to CP/M Plus.

Fstring^Z Finds character string.

H Saves the new file, then re-edits, using the new file as the original file.

I Enters insert mode; uses ^Z or ESCape to exit insert mode.

Istring^Z Inserts string at character pointer.

Jsearch_str^Zdel_to_str^Z Juxtaposes strings.

***n*K, −*n*K** Delete (kill) *n* lines from the character pointer.

***n*L, −*n*L, 0L** Move character pointer *n* lines.

***n*Mcommands** Execute commands *n* times.

n*, −*n Move character pointer *n* lines and display that line.

n: Moves to line *n*.

:ncommand Executes command through line *n*.

Nstring^Z Extends find string.

O Returns to original file.

***n*P, −*n*P** Move character pointer *n* lines forward and display *n* lines at console.

Q Abandons new file, returns to CP/M Plus.

R{**^Z**} Reads X$$$$$$$.LIB file into buffer.

Rfilespec{**^Z**} Reads filespec into buffer.

Sdelete string^Zinsert string{**^Z**} Substitutes string.

***n*T, −*n*T, 0T** Type *n* lines.

U, −U Uppercase translation.

V, −V, 0V Line numbering on/off, display free buffer space.

***n*W** Writes *n* lines to new file.

***n*X**{**^Z**} Writes or appends *n* lines to X$$$$$$$.LIB.

***n*Xfilespec**{**^Z**} Writes *n* lines to filespec or appends if previous x command applied to the same file.

0X{**^Z**} Deletes file X$$$$$$$.LIB.

0Xfilespec{ **^Z**} Deletes filespec.

*n***Z** Waits *n* seconds.

The I (Insert) command places ED in insert mode. In this mode, any characters you type are stored in sequence in the buffer starting at the current CP. Any single-letter commands typed in insert mode are not interpreted as commands, but are simply stored in the buffer. To return from insert mode to command mode, press ^Z or the ESC key. Note that you can always substitute the ESC key for ^Z in ED.

The single-letter commands are normally typed in lowercase. The commands that must be followed by a character sequence end with ^Z if they are to be followed by another command letter.

Any single-letter command typed in uppercase tells ED to internally translate to uppercase all characters up to the ^Z that ends the command.

When enabled, line numbers that appear on the left of the screen take the form:

nnnnn:

Where *nnnnn* is a number in the range 1 through 65535. Line numbers are displayed for your reference and are not contained in either the buffer or the character file. The screen line starts with :* when the character pointer is at the beginning or end of the buffer.

Examples:

```
ED MYPROG.PAS<cr>
```

If not already present, this command line creates the file MYPROG.PAS on drive A. The command prompt:

```
:*
```

appears on the screen. This tells you that the character pointer is at the beginning of the buffer. If the file is already present, issue the command:

```
$a
```

to fill the buffer. Then type the command:

```
On
```

to fill the screen with the first *n* lines of the buffer, where *n* is the current default page size. (See the DEVICE command to set the page size.)

Type the command:

```
e
```

to stop the ED utility when you are finished changing the character file. The ED utility leaves the original file unchanged as MYPROG.BAK and the altered file as MYPROG.PAS.

```
ED MYPROG.PAS B:NEWPROG.PAS<cr>
```

The original file is MYPROG.PAS on the default drive A. The original file remains unchanged when the ED utility finishes, with the altered file stored as NEWPROG.PAS on drive B.

B:ED MYPROG.PAS B:<cr>

The ED.COM file must be on drive B. The original file is MYPROG.PAS located on drive A. It remains unchanged, with the altered program stored on drive B as MYPROG.PAS.

ERASE—*erases a disk file*

The ERASE command removes one or more files from the directory of a disk. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA.

Format:

ERASE filespec [CONFIRM]

Use the ERASE command with care because all files that satisfy the file specification are removed from the disk directory.

Those ERASE commands that make use of wildcard characters require confirmation because they may erase an entire group of files. In this case, the computer prompts with the message:

```
Erase . . . (Y/N) ?
```

before erasing the files. Recheck the erase command carefully. Then respond with Y if you want to remove all matching files, and N if you want to avoid erasing any files.

If no files match the file specification, ERASE displays the following message:

```
No File
```

The CONFIRM option informs the system to prompt for verification before erasing each file that matches the filespec. You can abbreviate CONFIRM to C.

If you use the CONFIRM option with a wildcard filespec, ERASE prompts for confirmation for each file. You can selectively erase a file by responding Y to the CONFIRM message, or keep a file by responding N to the CONFIRM message.

Examples:

ERASE X.PAS<cr>

This command removes the file X.PAS from the disk in drive A.

ERA *.PRN<cr>

The system asks to confirm:

```
Erase * .PRN (Y/N) ?
```

All files with the file type PRN are removed from the disk in drive A.

ERA A:MY*.* [CONFIRM] <cr>

Each file on drive A with a file name that begins with MY is displayed with a question mark for confirmation. Type Y to erase the file displayed, N to keep the file.

ERA B:*.*<cr>

```
Erase B: * . * (Y/N) ?
```

All files on drive B are removed from the disk if you type Y.

GENCOM—*creates extension-compatible COM files*

The GENCOM command is a transient utility that creates an executable (.COM) file with attached RSX files. RSX (Resident System Extensions) files are discussed in detail in the Osborne Executive Technical Manual. GENCOM places a special header at the beginning of the output program file to indicate to the system that RSX loading is required. It can also set a flag to keep the program loader active.

The GENCOM command can also change a .COM file with RSXs to the original .COM file without the header and RSXs.

Format:

GENCOM filename

The above form of the GENCOM command takes a file that has already been processed by GENCOM and restores it to its original .COM file format. This form of the command assumes an input file type of .COM.

Example:

GENCOM MYPROG<cr>

GENCOM takes MYPROG.COM, strips off the header and deletes all attached RSXs to restore it to its original .COM format.

GENCOM has three options that help you attach or remove RSX files:

LOADER Sets a flag to keep the program loader active. This option is used only if no RSX files are attached to the .COM file.

NULL Indicates that only RSX files are specified. GENCOM creates a dummy .COM file for the RSX files. The output .COM file name is taken from the file name of the first RSX filespec.

SCB=(offset,value) Sets the System Control Block of the program, using the hex values specified by (offset,value).

Format:

GENCOM COM-filename RSX-filename...[option]

The above form of the GENCOM command adds and/or replaces RSX files to a file already processed by GENCOM.

GENCOM inspects the list of RSX files given in the command. If they are new, they are added to the file already processed by GENCOM. If they already exist, then GENCOM replaces the existing RSXs with the new RSX files. As above, GENCOM assumes a file type of .COM for the executable input file, and a file type of .RSX for all RSX files specified.

Example:

GENCOM MYPROG PROG1 PROG2<cr>

GENCOM looks at MYPROG.COM, which is already processed by GENCOM, to see if

PROG1.RSX and PROG2.RSX are already attached RSX files in the module. If either one is already attached, GENCOM replaces it with the new RSX module. Otherwise, GENCOM appends the specified RSX files to the COM file.

Format:

GENCOM RSX-filename...[NULL]

The above form of the GENCOM command attaches the RSX files to a dummy .COM file. GENCOM creates a .COM file with the file name of the first RSX-filename in the command tail. This format allows the system to load RSXs directly.

Example:

GENCOM PROG1 PROG2 [NULL] <cr>

Creates a file PROG1.COM with Resident System Extensions PROG1.RSX and PROG2.RSX.

Format:

GENCOM COM-filespec RSX-filespec...[option]

The above form of the GENCOM command creates a .COM file with a header and attached RSX module(s). A maximum of 15 RSX modules can be attached. GENCOM expects the first file name to be a .COM file and the following file names to be .RSX files. Note that the original .COM file is replaced by the newly created .COM file.

Example:

```
GENCOM MYPROG PROG1 PROG2<cr>
```

Generates a new file MYPROG.COM with attached RSXs PROG1 and PROG2.

Format:

```
GENCOM filename [ SCB=(offset,value),... | LOADER]
```

The above line attaches a GENCOM header record, with the SCB or LOADER flag set, to a file of type COM that contains no RSX. This form of the command does not attach RSXs to a file.

Examples:

```
GENCOM FILETWO [ loader] <cr>
```

Attaches a 256-byte header record to the file FILETWO.COM and sets the loader flag in the header record.

```
GENCOM FILEFOUR [ scb=(1,1)] <cr>
```

Causes the program loader to set byte 1 of the System Control Block to 1 when it loads FILEFOUR.COM.

NOTE

For more information, see the Digital Research GENCOM documentation or the Executive Technical Manual.

GET—*inputs console command from a disk file*

The GET command is a transient utility that lets you input CP/M Plus commands from a disk file. GET allows you to direct the system to take console input from a file for the next system command or user program that is entered. If you use the SYSTEM option, GET immediately takes console input from the file. Otherwise, it waits until you give a command to load and execute a transient program.

Format:

GET FILE filespec [options]

The above form of the GET command instructs the Osborne Executive Computer to get subsequent console input from a disk file. Console input is taken from a file until the program terminates. If the file is exhausted before program input is terminated, the program looks for subsequent input from the console. If the program terminates before exhausting all its input, the system reverts back to the console for console input.

When the SYSTEM option is used, the system immediately goes to the file specified for console input. If you omit the SYSTEM option, you can enter one system command to initiate a user program whose console input is taken from the file specified in the GET command. The system reverts to the console for input when it reaches the end of the file. The system can also be redirected to the console for console input if the GET CONSOLE command is a command line in the input file.

Example:

```
GET FILE XINPUT <cr>
MYPROG<cr>
```

The above sequence of commands tells the system to activate the GET utility. However, because SYSTEM is not specified, the system reads the next input line from the console and executes MYPROG. If the MYPROG program requires console input, it is taken from the file XINPUT. When MYPROG terminates, the system reverts to the console for console input.

The GET options are described below:

Options and Their Meanings

ECHO Specifies that the input is echoed to the console. This is the default option.

NO ECHO Specifies that the file input is not to be echoed to the console. The program output and system prompts are not affected by this option and are still echoed to the console.

SYSTEM Specifies that all system input is to be taken from the disk file specified in the command line. GET takes system and program input from the file until the file is exhausted or until GET reads a GET CONSOLE command from the file.

Example:

```
GET FILE XIN2 [ SYSTEM] <cr>
```

The above command immediately directs the system to get subsequent console input from file XIN2 because it includes the SYSTEM option. The system reverts to the console for console input when it reaches the end-of-file marker in XIN2. Or, XIN2 may redirect the system back to the console if it contains a GET CONSOLE command.

Format:

GET CONSOLE

The above form of the GETcommand tells the system to get console input from the console.

Example:

GET CONSOLE<cr>

This GETcommand tells the system to get console input from the console. You can use this command in a file (previously specified in a GET FILE command) which is already being read by the system for console input. It is used to redirect the console input to the console before the end-of-file is reached.

HELP—*provides information about CP/M Plus commands*

The HELP command is a transient utility that provides summarized information for all of the CP/M Plus commands described in this manual. HELP with no command tail displays a list of all the available topics. HELP with a topic in the command tail displays information about that topic, followed by any available subtopics. HELP with a topic and a subtopic displays information about the specific subtopic.

After HELP displays the information for your specified topic, it displays the special prompt

HELP> on your screen. Subtopics can then be accessed by preceding the subtopic with a period. The period causes the subtopic search to begin at the last known level. You can continue to specify topics for additional information, or simply press the RETURN key to return to the CP/M Plus system prompt.

You can abbreviate the names of topics and subtopics to the fewest letters that uniquely specify the topic name. Usually one or two letters is enough to specifically identify the topics.

When referencing a subtopic, you must type the topic name AND the subtopic, otherwise the HELP program cannot determine which main topic you are referencing. You can also enter a topic and subtopic following the program's internal prompt, HELP>, as shown below:

```
HELP>ED COMMANDS
```

This form of HELP displays information about commands internal to the editing program, ED.

Format:

```
HELP topic subtopic1 subtopic2 ... [option]
```

The above form of the HELP command displays the information for the specified topic and subtopics. Use the following two options with this form of the HELP command:

NOPAGE Disables the default-paged display of every *n* lines, where *n* is the number of lines per page as set by the system or as set by the user. To stop the display, press CTRL-S. To resume the display, press CTRL-Q. You can abbreviate NOPAGE to N. (See the DEVICE command for more information about setting the number of lines per page.)

LIST Same as NOPAGE, except that this option displays extra lines between headings.

Examples:

```
HELP<cr>
```

Displays a list of topics for which help is available.

```
HELP DATE<cr>
```

Displays general information about the DATE command. It also displays any available subtopics.

```
HELP DIR OPTIONS [ N] <cr>
```

The command above includes the subtopic options. In response, HELP displays information about options associated with the DIR command. The display is not in paged mode because the NO PAGE option is specified.

```
HELP ED<cr>
```

Displays general information about the ED utility.

```
HELP ED COMMANDS<cr>
```

Displays information about commands internal to ED. The above example can also be entered as:

```
HELP>ED.COMMANDS<cr>
```

Formats:

HELP [EXTRACT]

HELP [CREATE]

CP/M Plus is distributed with two related HELP files: HELP.COM and HELP.HLP. The HELP.COM file is the command file that processes the text of the HELP.HLP file and displays it on the screen. The HELP.HLP file is a text file to which you can add customized information, but you cannot directly edit the HELP.HLP file. You must use the HELP.COM file to convert HELP.HLP to a file named HELP.DAT before you can edit or add your own text.

This form of the HELP command has the following options:

EXTRACT Accesses the file HELP.HLP and creates a file called HELP.DAT, both on the default drive. You can now invoke a word-processing program to edit or add your own text to the HELP.DAT file. EXTRACTcan be abbreviated to E.

CREATE Accesses your edited HELP.DAT file on the default drive and builds a revised HELP.HLP file on the default drive. CREATE can be abbreviated to C.

You must add topics and subtopics to the HELP.DAT file in a specific format. The general format of a topic heading in the HELP.DAT file is shown below:

/ / /*n*Topicname<cr>

The three backslashes are the topic delimiters and must begin in column one. In the format statement above, *n* is a number in the range from 1 through 9 that signifies the level of the topic. A main topic always has a level number of 1. The first subtopic has a level number of 2. The next level of subtopic has a level number of 3, and so forth, up to a maximum of nine levels. "Topicname" is the name of your topic, and allows a maximum of twelve characters. The entire line is terminated with a carriage return.

Use the following guidelines to edit and insert text into the HELP.DAT file.

- Topics should be placed in alphabetical order.

- Subtopics should be placed in alphabetical order within their respective supertopic.

- Levels must be indicated by a number 1–9.

Some examples of topic and subtopic lines in the HELP.HLP file are shown below.

```
///1NEW UTILITY<cr>

///2COMMANDS<cr>

///3PARAMETERS<cr>

///2EXAMPLES<cr>
```

The first example shown above illustrates the format of a main topic line. The second example shows how to number the first subtopic of that main topic. The third example shows how the next-level subtopic under level 2 should be numbered. The fourth example shows how to return to the lower-level subtopic. Any topic name with a level number of 1 is a main topic. Any topic name with a level number of 2 is a subtopic within its main topic.

HEXCOM—*generates command file (.COM) from hex file*

The HEXCOM command is a transient utility that generates a command file (file type .COM) from a .HEX input file. It names the output file with the same file name as the input file but with file type .COM. HEXCOM always looks for a file with file type .HEX.

Format:

```
HEXCOM filename
```

Example:

```
HEXCOM B:PROGRAM<cr>
```

In the above command, HEXCOM generates a command file PROGRAM.COM from the input hex file PROGRAM.HEX.

INITDIR—*initializes disk directory*

The INITDIR command initializes a disk directory to allow date and time stamping of files on that disk.

Format:

INITDIR d:

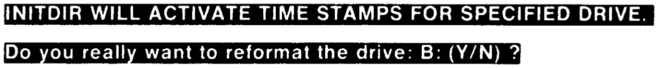
INITDIR is required to reformat the directory to enable time stamping, because date and time stamps require a different disk directory format. While reformatting the disk, INITDIR saves all of your files, then restores them when finished reformatting. If there is not enough directory space to save the files, INITDIR displays the following message:

INITDIR does not reformat the directory.

Example:

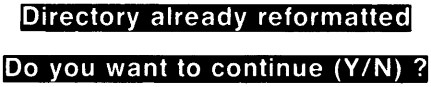
INITDIR B:<cr>

The system prompts to confirm:



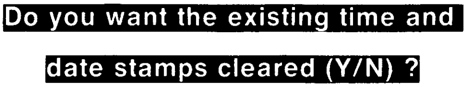
```
INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE.
Do you really want to reformat the drive: B: (Y/N) ?
```

If the disk had previously been formatted for time and date stamps, INITDIR displays the message:



```
Directory already reformatted
Do you want to continue (Y/N) ?
```

If INITDIR finds time and date stamps in directory, it displays the following message:



```
Do you want the existing time and
date stamps cleared (Y/N) ?
```

If you want to clear the date and time stamps on your disk, type Y for yes; otherwise, type N.

If the disk label is password-protected, INITDIR displays the message:

```
Directory is password protected

Password, please.
```

Enter the password to continue. If you do not enter the correct password, INITDIR terminates.

LIB—*library file utility*

A library is a disk file that contains a collection of commonly used object modules. Use the LIB utility to create libraries, or to append, replace, select, or delete modules from an existing library. You can also use LIB to obtain information about the contents of library files.

LIB creates and maintains library files that contain object modules in MicroSoft REL format. These modules are produced by Digital Research's relocatable macro assembler program, RMAC, or any other language translator that produces modules in MicroSoft REL format.

Link-80 links the object modules contained in a library to other object files. LINK-80 automatically selects from the library only those modules needed by the program being linked, and then forms an executable file with a file type of .COM.

The library file has the file type .REL or .IRL depending on the option you choose. Modules in a .REL library file must not contain backward references to modules that occur earlier in the library because LINK-80 currently makes only one pass through a library.

Format:

LIB filename [option]

Here are the options that can be specified in the LIB command:

Options and Their Meanings

I The INDEX option creates an indexed library file of type .IRL. LINK-80 searches faster on indexed libraries than on nonindexed libraries.

M The MODULE option displays module names.

P The PUBLICS option displays module names and the public variables for the new library file.

D The DUMP option displays the contents of object modules in ASCII form.

Examples:

LIB TEST4[P] <cr>

LIB TEST5[P] =FILE1,FILE2<cr>

The first example displays all modules and publics in TEST4.REL. The second example creates TEST5.REL from FILE1.REL and FILE2.REL and displays all modules and publics in TEST5.REL.

Format:

LIB filename[option] = filename<modifier>,filename<modifier>...

Unless otherwise specified, LIB assumes a file type of .REL for all source file names. When you follow a file name by a group of module names enclosed in parentheses, these modules are included in the new library file. If modules are not specified, LIB includes all modules from the source file in the new library file.

Use modifiers in the command line to instruct LIB to delete, replace, or select modules in a library file. Angle brackets (< >) enclose the modules to be deleted or replaced. Parentheses enclose the modules to be selected. Here are the modifiers:

Modifiers and Their Syntax

Delete <module–>

Replace <module=filename.REL>

If module name and file name are the same, this shorthand can be used:

<filename>

Select (modFIRST-modLAST,mod1,mod2, ...,modN)

Examples:

LIB FILE2=FILE3<MODA=><cr>

In this example, LIB creates FILE2.REL from FILE3.REL, omitting MODA which is a module in FILE3.REL.

```
LIB FILE6=FILE5<MODA=FILE8.REL><cr>
```

In this example, MODA is in the existing FILE5.REL. When LIB creates FILE6.REL from FILE5.REL, FILE8.REL replaces MODA.

```
LIB FILE6=FILE5<THISNAME><cr>
```

In this example, module THISNAME is in FILE5.REL. When LIB creates FILE6.REL from FILE5.REL, the file THISNAME.REL replaces the similarly named module THISNAME.

```
LIB TEST=TEST1(MOD1,MOD4),TEST2(C1-C4,C6)<cr>
```

In the above example, LIB creates a library file TEST.REL from modules in two source files. TEST1.REL contributes MOD1 and MOD4. LIB extracts modules C1, C4, and all the modules located between them, as well as module C6 from TEST2.REL.

```
LIB FILE1[I]=B:FILE2 (PLOTS,FIND,SEARCH-DISPLAY)<cr>
```

In this example, LIB creates FILE1.IRL on drive A from the selected modules PLOTS, FIND; and modules SEARCH through the module DISPLAY, in FILE2.REL on drive B.

LINK—*combines relocatable object modules*

The LINK command combines relocatable object modules, such as those produced by RMAC and PL/I-80, into a .COM file ready for execution. Relocatable files can contain external references and publics, and reference modules in library files. LINK searches the library files and includes the referenced modules in the output file. The LINK command is identical with the

LINK-80 utility. For more detailed information, see the LINK-80 documentation published by Digital Research.

Examples:

```
LINK m1,m2,m3<cr>
```

LINK-80 combines the separately compiled files m1, m2, and m3; resolves their external references; and produces the executable machine-code file M1.COM.

```
LINK m=m1,m2,m3<cr>
```

LINK-80 combines the separately compiled files m1, m2, and m3 and produces the executable machine-code file M.COM.

You can use LINK option switches to control the execution parameters of LINK-80. Link options follow the file specifications and are enclosed within square brackets. Multiple switches are separated by commas.

Format:

```
LINK filespec[ option] =
filespec[ option] ,filespec[ option] ...
```

Here are the options that can be used in the LINK command:

Options and Their Meanings

A Additional memory; reduces buffer space and writes temporary data to disk.

B BIOS link in banked CP/M Plus system. Aligns data segment on page boundary; puts length of code segment in header; defaults to .SPR file type.

Dhhhh Data origin; sets memory origin for common and data area.

G*n* Go; sets start address to label *n*.

Lhhhh Load; changes default load address of module to hhhh. Default 0100H.

Mhhhh Memory size; defines free memory requirements for MP/M modules.

NL No listing of symbol table at console.

NR No symbol table file.

Example:

```
LINK B:MYFILE[ NR]<cr>
```

LINK-80 on drive A uses as input MYFILE.REL on drive B and produces the executable machine-code file MYFILE.COM on drive B. The [NR] option specifies no symbol table file.

OC Output .COM command file. Default.

OP Output .PRL page-relocatable file for execution under MP/M in relocatable segment.

OR Output .RSP-resident system process file for execution under MP/M.

OS Output .SPR system page-relocatable file for execution under MP/M.

Phhhh Program origin; changes default program origin address to hhhh. Default is 0100H.

Q Lists symbols with leading question mark.

S Searches preceding file as a library.

Example:

```
LINK MYFIL,FILE5[s]<cr>
```

The [s] option tells LINK-80 to search FILE5 as a library. LINK-80 combines MYFILE.REL with the referenced subroutines contained in FILE5.REL on the default drive A and produces MYFILE.COM on drive A.

$Cd Destination of console messages d can be X (console), Y (printer), or Z (zero output). Default is X.

$Id Source of intermediate files; d is disk drive A–P. Default is current drive.

$Ld Source of library files; d is disk drive A–P. Default is current drive.

$Od Destination of object file; d can be Z or disk drive A–P. Default is to same drive as first file in the LINK-80 command.

$Sd Destination of symbol file; d can be Y or Z or disk drive A–P. Default is to same drive as first file in LINK-80 command.

Example:

```
LINK MYFILE, FILE1, FILE2[ S] , $CY IB LC OB SZ<cr>
```

This command tells LINK to link MYFILE.REL with FILE1.REL and FILE2.REL, search FILE2.REL as a library and send console messages to the printer, intermediate files to drive B, find the library file on drive C, put the object file on drive C, and suppress the symbol file output.

MAC—*macro assembler*

MAC, the CP/M Plus macro assembler, is a transient utility that reads assembly-language statements from a disk file of file type .ASM. MAC assembles the statements and produces three output files with the input file name, and output file types of .HEX, .PRN, and .SYM.

Format:

```
MAC filename $options
```

MAC assumes that filename.HEX contains Intel hexadecimal-format object code. You can debug the HEX file with a debugger, or link it with LINK-80 and execute it.

Filename.PRN contains an annotated source listing that can be printed or examined at the console. The .PRN file includes a 16-column-wide machine-language listing at the left side of the page that shows the values of literals, machine-code addresses, and generated machine code. An equals sign denotes literal addresses to eliminate confusion with machine-code addresses.

The output file filename.SYM contains a sorted list of symbols defined in the program.

Before invoking MAC, you must prepare a source program file with the file type .ASM containing assembly-language statements.

Example:

```
MAC SAMPLE<cr>
```

In the above example, MAC is invoked from drive A and operates on the file SAMPLE.ASM also on drive A.

You can direct the input and output of MAC using the options listed in the table below. Use a letter with the option to indicate the source and destination drives, console, printer, or zero output. Valid drive names are A through O. X directs output to the console. P directs output to the printer. Z specifies that output files will not be created. Here are the MAC options:

Options and Their Meanings

A Source drive for .ASM file.

H Destination drive for .HEX file.

L Source drive for macro-library LIB files called by the MACLIB statement.

P Destination drive for .PRN file (A,B,X,P,Z).

S Destination drive for .SYM file (A,B,X,P,Z).

Example:

MAC SAMPLE $PB AA HB SX<cr>

In this example, an assembly-option parameter list follows the MAC command and the source file name. The parameters direct the .PRN file to drive B, obtain the .ASM file from drive A, direct the .HEX file to drive B, and send the .SYM file to the console. You can use blanks between option parameters.

Here are the output file modifiers used with MAC:

Modifiers and Their Meanings

+**L** Lists input lines read from macro-library LIB files.

−**L** Suppresses listing (default).

+**M** Lists all macro lines as they are processed during assembly.

−**M** Suppresses all macro lines as they are read during assembly.

***M** Lists only hex generated by macro expansions.

+**Q** Lists all LOCAL symbols in the symbol list.

−**Q** Suppresses all LOCAL symbols in the symbol list (default).

+**S** Appends symbol file to print file.

−S Suppresses creation of symbol file.

+1 Produces a pass-1 listing for macro debugging in .PRN file.

−1 Suppresses listing on pass 1 (default).

PATCH—*installs Digital Research–distributed patches*

The PATCH command displays or installs patch number *n* to the CP/M Plus system or CP/M Plus command files.

Format:

PATCH filename *n*

Only CP/M Plus system files of file type .COM, .PRL, or .SPR can be patched with the PATCH command. If the file type is not specified, the PATCH utility looks for a file with a file type of .COM. The patch number *n* must be between 1 and 32 inclusive.

Example:

PATCH SHOW 2<cr>

The above command patches the system SHOW.COM file with patch number 2. The system displays the following query:

```
Do you want to indicate that Patch #2
has been installed for SHOW.COM?Y
```

If the patch is successful, the system displays the message:

```
Patch installed
```

If the patch is not successful, the system displays the following message:

```
Patch not Installed
```

One of the following error messages might be displayed:

```
* ERROR: Patch requires CP/M Plus.
* ERROR: Invalid filetype typ.
* ERROR: Serial Number mismatch.
* ERROR: Invalid patch number n.
```

PIP—*performs transfer of data between devices*

PIP (Peripheral Interchange Program) is a transient utility that copies one or more files from one disk and/or user number to another. PIP can rename a file after copying it; it can combine two or more files into one file; it can copy a character file from disk to the printer or auxiliary logical output device; it can create a file on disk from input to the console or other logical input device, and can transfer data from a logical input device to a logical output device; hence the name Peripheral Interchange Program.

PIP copies file attributes with the file. This includes R/W, R/O, SYS and DIR file attributes and the user-definable attributes F1 through F4. If a file is password-protected, you must enter the password in the command line following the file name and/or file type to which it belongs. If the password fails, the file is skipped and the failure noted.

When you specify a destination file with a password, PIP assigns that password to the destination file and automatically sets the password-protection mode to READ. This means that you need a password to read the file. (See the SET command.) When you specify a destination file with no password, PIP does not assign a password to the destination file. When you specify only a destination drive, PIP assigns the same password and password-protection mode to the destination file that was specified in the source file.

Before you start PIP, be sure that you have enough free space on your destination disk to hold the entire file or files you are copying. Even if you are replacing an old copy on the destination disk with a new copy, PIP still needs enough room for the new copy before it deletes the old copy. Use the DIR command to determine file size and the SHOW command to determine disk space. If there is not enough space, you can delete the old copy first by using the ERASE command.

Data is first copied to a temporary file to ensure that the entire data file can be constructed in the space available on the disk. PIP gives the temporary file the file name specified for the destination, with the file type .$$$. If the copy operation is successful, PIP changes the temporary file type .$$$ to the file type specified in the destination. If the copy operation succeeds and a file with the same name as the destination file already exists, the old file with the same name is erased before renaming the temporary file.

File attributes (SYS, DIR, RW, RO) are transferred with the files. If the existing destination file is set to Read-Only (RO), PIP asks you if you want to delete it. Answer Y or N. Use the [W] option to write over Read-Only files.

Format:

PIP

This form of the PIP command starts the PIP utility and lets you type multiple command lines while PIP remains in memory.

PIP displays an asterisk prompt on your screen when ready to accept input command lines. You can type any valid command line described below, following the asterisk prompt.

Terminate PIP by pressing only the RETURN key following the asterisk prompt. The empty command line tells PIP to discontinue operation and return to the CP/M Plus system prompt.

Example:

```
PIP<cr>
CP/M 3 PIP VERSION 3.0
*NEWFILE=FILE1,FILE2,FILE3<cr>
*APROG.COM=BPROG.COM<cr>
*A:=B:X.BAS<cr>
*B:=*.*<cr>
*<cr>
A>
```

The first command loads the PIP program. The PIP command input prompt * tells you that PIP is ready to accept commands. The effects of this sequence of commands are similar to those in the examples below, where the command line is included in the command tail, except that PIP is not loaded into memory for each command.

Format:

```
PIP d:=filespec[ options]
```

This form shows the simplest way to copy a file. PIP looks for the file named by the source filespec (to the right of the equals sign) on the default or optionally specified drive. PIP copies the file to the drive specified by d: and gives it the same name as the source filespec. If you want, you can use the [G*n*] option to read your source file from the user number specified by *n*. Several options can be combined together for the source file specification. The only option recognized for the destination file is [G*n*]. See PIP options below.

Examples:

```
PIP B:=A:*.COM<cr>
```

This command causes PIP to copy all the files on drive A with the file type COM to drive B.

```
PIP B:=A:PROG$$$$.*<cr>
PIP B:=A:PROG*.*<cr>
```

Either of the commands above cause PIP to copy all files whose file names begin with PROG from drive A to drive B.

```
PIP B:=A:*.*<cr>
```

This command causes PIP to copy all the files on drive A to drive B. You can use this command to make a backup copy of your distribution disk. Note, however, that this command does not copy the CP/M Plus system from the system tracks. Use COPYSYS to copy the operating system tracks.

```
PIP B:[G1]=A:*.BAS<cr>
```

This command causes PIP to copy all the files with a file type of BAS on drive A in the default user number (user 0) to drive B in user number 1. Recall that the DIR, TYPE, ERASE, and other commands only access files in the same user number from which they were invoked. (See the USER Command.)

Format:

```
PIP filespec=d: [options]<cr>
```

This form is a variation of the first. PIP looks for the file named by the destination filespec on the drive specified by d:, copies it to the default or optionally specified drive, and gives it the name specified by the destination filespec.

Examples:

```
PIP B:=A:OLDFILE.DAT<cr>

PIP B:OLDFILE.DAT=A:<cr>
```

Both forms of this command cause PIP to read the file OLDFILE.DAT from drive A and put an exact copy of it onto drive B. This is called the short form of PIP, because the source or destination names only a drive and does not include a filespec. When using this form, you cannot copy a file from one drive and user number to the same drive and user number. You must put the destination file on a different drive or in a different user number. See the section on PIP Options, and the USER Command. The second short form produces exactly the same result as the first one. PIP simply looks for the file OLDFILE.DATon drive A, the drive specified as the source.

Format:

PIP filespec=filespec[options] <cr>

This form shows how to rename the file after copying it. You can copy it to the same drive and user number, or to a different drive and/or user number. Rules for options are the same. PIP looks for the file specified by the source filespec, copies it to the location specified in the destination filespec, and gives it the name indicated by the destination filespec.

Examples:

PIP B:NEWFILE.DAT=A:OLDFILE.DAT<cr>

This command copies the file OLDFILE.DAT from drive A to drive B and renames it to NEWFILE.DAT. The file remains as OLDFILE.DAT on drive A. This is the long form of the PIP command because it names a file on both sides of the command line.

PIP NEWFILE.DAT=OLDFILE.DAT<cr>

Using this long form of PIP, you can copy a file from one drive and user number (usually user 0 because CP/M Plus automatically starts out in user 0, the default user number) to the same drive and user number. This gives you two copies of the same file on one drive and user number, each with a different name.

Remember that PIP always goes to and gets from the current default user number unless you specify otherwise with the [G*n*] option.

Format:

PIP d:=wildcard-filespec[options]

When you use a wildcard in the source specification, PIP copies matching files one by one to the destination drive, retaining the original name of each file. PIP displays the message COPYING followed by each filespec as the copy operation proceeds. PIP issues an error message and aborts the copy operation if the destination drive and user number are the same as those specified in the source.

Format:

PIP filespec=
filespec[options] ,filespec[options] ,...

This form of the PIP command lets you specify two or more files in the source. PIP copies the files specified in the source from left to right and combines them into one file with the name indicated by the destination file specification. This procedure is called file concatenation. You can use the [G*n*] option after the destination file to place it in the user number specified by *n*. You can specify one or more options for each source file.

Examples:

PIP NEWFILE=FILE1,FILE2,FILE3<cr>

The three files named FILE1, FILE2, and FILE3 are joined from left to right and copied to NEWFILE.$$$. NEWFILE.$$$ is renamed to

NEWFILE upon successful completion of the copy operation. All source and destination files are on the disk in the default drive and user number.

PIP B:X.BAS=Y.BAS,B:Z.BAS<cr>

The file Y.BAS on drive A is joined with Z.BAS from drive B and placed in the temporary file X.$$$ on drive B. The file X.$$$ is renamed to X.BAS on drive B when PIP runs to successful completion.

Format:

PIP filespec or device=
filespec or device[option]

AUX:	**AUX: [option]**
CON:	**CON: [option]**
PRN:	**NUL:**
LST:	**EOF:**

This form is a special case of the PIP command line that lets you copy a file from a disk to a device, from a device to a disk or from one device to another. The files must contain printable (ASCII) characters. Each peripheral device is assigned to a logical device that identifies a source device that can transmit data or a destination device that can receive data. (See the DEVICE command.) A colon follows each logical device name so it cannot be confused with a filespec. The valid devices for source and destination files are indicated in the format description above. Enter ^C to abort a copy operation that uses a logical device in the source or destination.

The logical device names are listed below:

CON: Console input or output device. When used as a source, usually the keyboard; when used as a destination, usually the screen.

AUX: Auxiliary Input or Output Device.

LST: The destination device assigned to the list output device, usually the printer.

The following three device names have special meanings:

NUL: A source device that produces 40 hexadecimal zeros.

EOF: A source device that produces a single ^Z, the CP/M Plus end-of-file mark.

PRN: The printer device with tab expansion to every eighth column, line numbers, and page ejects every 60th line.

Examples:

```
PIP PRN:=CON:,MYDATA.DAT<cr>
```

Characters are first read from the console input device, generally the keyboard, and sent directly to your printer device. You type a ^Z character to tell PIP that keyboard input is complete. At that time, PIP continues by reading character data from the file MYDATA.DATon drive B. Since PRN: is the destination device, tabs are expanded to 8 columns, line numbers are added, and page ejects occur every 60 lines.

Note that when the CON: device is the source you must enter both the carriage return (RETURN) and line feed (LF) keys for a new line.

```
PIP B:FUNFILE.SUE = CON:<cr>
```

Whatever you type at the console is written to the file FUNFILE.SUE on drive B. End the keyboard input by typing a ^Z.

```
PIP LST:=CON:<cr>
```

Whatever you type at the console keyboard is written to the list device, generally the printer. Terminate input with a ^Z.

```
PIP LST:=B:DRAFT.TXT[ T8] <cr>
```

The file DRAFT.TXTon drive B is written to the printer device. Any tab characters are expanded to the nearest column that is a multiple of 8.

```
PIP PRN:=B:DRAFT.TXT<cr>
```

The command above causes PIP to write the file DRAFT.TXT to the list device. It automatically expands the tabs, adds line numbers, and ejects pages after 60 lines.

Using Options with PIP

Options enable you to process your source file in special ways. You can expand tab characters, translate from upper- to lower-case letters, extract portions of your text, or verify that the copy is correct. Options can allow PIP to read a file with the system (SYS) attribute, cause PIP to write over Read-Only files, cause PIP to put a file into or copy it from a specified user number, translate from lower- to uppercase, and much more.

You can include PIP options following each source name. There is one valid option ([G*n*] —go to user number *n*) for the destination file specification. Options are enclosed in square brackets. Several options can be included for the source files. They can be packed together or separated by spaces.

The PIP options are listed below, using "*n*" to represent a number and "s" to represent a sequence of characters terminated by a ^Z. An option must immediately follow the file or device it affects, and be enclosed in square brackets [ ]. For those options that require a numeric value, no blanks can occur between the letter and the value.

You can include the [G*n*] option after a destination file specification. You can include a list of options after a source file or source device. An option list is a sequence of single letters and numeric values, optionally separated by blanks, and enclosed in square brackets [ ].

Many options force PIP to copy files character by character. In these cases, PIP looks for a ^Z character to determine where the end of the file is. All of the PIP options force a character transfer except the following:

A, C, G*n*, K, O, R, V, and W

Copying data to or from logical devices also forces a character transfer. You can terminate PIP operations by typing ^C. When concatenating files, PIP only searches the last record of a file for the ^Z end-of-file character. However, if PIP is doing a character transfer, it stops when it encounters a ^Z character.

Use the [O] option if you are concatenating machine-code files. The [O] option causes PIP to ignore embedded ^Z characters, normally used to indicate the end-of-file character in text files.

Following are the options which can be used with PIP:

Options and Their Functions

A Copies only the files that have been modified since the last copy. To back up only the files that have been modified since the last backup, use PIP with a wildcard filespec and the Archive option. PIP with the [A] option copies only the files that have been modified.

C Prompts for confirmation before performing each copy operation. Use the [C] option when you want to copy only some files of a particular file type.

D*n* Deletes any characters past column *n*. This parameter follows a source file that contains lines too long to be handled by the destination device; for example, an 80-character printer or narrow console. The number *n* should be the maximum column width of the destination device.

Example:

```
PIP CON:=WIDEFILE.BAS[ D80] <cr>
```

Writes the character file WIDEFILE.BAS from drive A to the console device, but deletes all characters following the 80th column position.

E Echoes transfer at console. When this parameter follows a source name, PIP displays the source data at the console as the copy is taking place. The source must contain character data.

Examples:

```
PIP B:=LETTER.TXT [ E] <cr>
```

The file LETTER.TXT from drive A is copied to LETTER.TXTon drive B. The LETTER.TXT file is also written to the screen as the copy operation proceeds.

```
PIP PROGRAM2.DAT=
PROGRAM1.DAT[ E V G3] <cr>
```

In this command, PIP copies the file named PROGRAM1.DATon drive A and echoes [E] the transfer to the console, verifies [V] that the two copies are exactly the same, and gets [G3] the file PROGRAM1.DAT from user 3 on drive A. Since there is no drive specified for the destination, PIP automatically copies the file to the default user number and drive.

F Filters out form-feeds. When this parameter follows a source name, PIP removes all form-feeds embedded in the source data. To change form-feeds set for one page length in the source file to another page length in the destination file, use the F option to delete the old form-feeds and a P option to simultaneously add new form-feeds to the destination file.

Example:

```
PIP LST:=B:LONGPAGE.TXT[FP65]<
```

Writes the file LONGPAGE.TXT from drive B to the printer device. As the file is written, form-feed characters are removed and reinserted at the beginning and every 65th line thereafter.

G*n* Gets source from or Goes to user number *n*. When this parameter follows a source name, PIP searches the directory of user number *n* for the source file. When it follows the destination name, PIP places the destination file in the user number specified by *n*. The number must be in the range 0 to 15.

Example:

```
PIP B:PROGRAM.BAK=
A:PROGRAM.DAT [G1]<cr>
```

The command above copies the file PROGRAM.DAT from user 1 to the current selected user number on drive B and renames the file type on drive B to PROGRAM.BAK.

H Hex data transfer. PIP checks all data for proper Intel hexadecimal-format file. The console displays error messages when errors occur.

I Ignore :00 records in the transfer of Intel hexadecimal-format file. The I option automatically sets the H option.

L Translates uppercase alphabetics in the source file to lowercase in the destination file. This parameter follows the source device or file name.

Example:

```
PIP NEWPROG.BAS=
CODE.BAS[ L] ,DATA.BAS[ U] <cr>
```

Constructs the file NEWPROG.BAS on drive A by joining the two files CODE.BAS and DATA.BAS from drive A. During the copy operation, CODE.BAS is translated to lowercase, while DATA.BAS is translated to uppercase.

N Adds line numbers to the destination file. When this parameter follows the source file name, PIP adds a line number to each line copied, starting with 1 and incrementing by one. A colon follows the line number. If N2 is specified, PIP adds leading zeroes to the line number and inserts a tab after the number. If the T parameter is also set, PIP expands the tab.

Example:

```
B>PIP LST:=PROGRAM.BAS[ NT8U] <cr>
```

Writes the file PROGRAM.BAS from drive B to the printer device. The N parameter tells PIP to number each line. The T8 parameter expands tabs to every eighth column. The U parameter translates lowercase letters to uppercase as the file is printed.

O Object file transfer for machine-code (noncharacter and therefore nonprintable) files. PIP ignores any ^Z end-of-file during concatenation and transfer. Use this option if you are combining or moving object-code files.

P*n* Sets page length. *n* specifies the number of lines per page. When this parameter modifies a source file, PIP includes a page eject at the beginning of the destination file and at every *n* lines. If *n* = 1 or is not specified, PIP inserts page ejects every 60 lines. When you also specify the F option, PIP ignores form-feeds in the source data and inserts new form-feeds in the destination data at the page length specified by *n*.

Qs Quits copying from the source device after the string s. When used with the S parameter, this parameter can extract a portion of a source file. The string must be terminated by ^Z.

R Reads system (SYS Attribute) files. Normally, PIP ignores files marked with the system attribute in the disk directory. But when this parameter follows a source file name, PIP copies system files, including their attributes, to the destination.

Ss Starts copying from the source device at the string s. The string argument must be terminated by ^Z. When used with the Q parameter, this parameter can extract a portion of a source file. Both start and quit strings are included in the destination file.

Example:

```
PIP PORTION.TXT=LETTER.TXT
[ SDear Sir^Z QSincerely^Z<cr>
```

Abstracts a portion of the LETTER.TXT file from drive A by searching for the character sequence "Dear Sir" before starting the copy operation. When found, the characters are copied to PORTION.TXTon drive A until the sequence "Sincerely" is found in the source file.

T*n* Expands tabs. When this parameter follows a source file name, PIP expands tab (^I) characters in the destination file. PIP replaces each ^I with enough spaces to position the next character in a column divisible by *n*.

U Translates lowercase alphabetic characters in the source file to uppercase in the destination file. This parameter follows the source device or file name.

V Verifies that data has been copied correctly. PIP compares the destination to the source data to ensure that the data has been written correctly. The destination must be a disk file.

Example:

```
B>PIP B:=A:*.COM[ VWR] <cr>
```

Copies all files with file type .COM from drive A to drive B. The V parameter tells PIP to read the destination files to ensure that data was correctly transferred. The W parameter lets PIP overwrite any destination files that are marked as RO (Read-Only).

The R parameter tells PIP to read files from drive A that are marked with the SYS (System) attribute.

W Writes over files with RO (Read-Only) attribute. Normally, if a PIP command tail includes an existing RO file as a destination, PIP sends a query to the console to make sure you want to write over the existing file. When this parameter follows a source name, PIP overwrites the RO file without a console exchange. If the command tail contains multiple source files, this parameter need follow only the last file in the list.

Z Zeroes the parity bit. When this parameter follows a source name, PIP sets the parity bit of each data byte in the destination file to zero. The source must contain character data.

PUT—*sends output to a specified file*

The PUTcommand is a transient utility that lets you direct console or printer output to a disk file. Use PUT to put console or printer output to a file during the execution of the next system command or user program. Or, PUTwith the SYSTEM option directs all subsequent console or printer output to a file.

Console or printer output is directed to the file until the program or command terminates. Then, output reverts to the console or printer.

When you use the SYSTEM option, all subsequent console/printer output is directed to the specified file until you enter a PUT CONSOLE or PUT PRINTER command.

Format:

PUT CONSOLE TO FILE filespec [option]

The above form of the PUT command tells the system to direct subsequent console output to a file. Here are the options which can be used with the PUT command.

Options and Their Meanings

ECHO Specifies that the output is echoed to the console. ECHO is the default option when you direct console output to a file.

NO ECHO Specifies that the file output is not to be echoed to the console.

FILTER Specifies filtering of control characters, which means that control characters are translated to printable characters. For example, an escape character is translated to ^[.

NO FILTER Means that PUT does not translate control characters. This is the default option.

SYSTEM Specifies that system output and program output is written to the file specified by filespec. Output is written to the file until a subsequent PUT CONSOLE command redirects console output back to the console.

Example:

PUT CONSOLE TO FILE XOUT [ECHO] <cr>

Directs console output to file XOUT with the output echoed to the console.

Format:

PUT PRINTER TO FILE filespec [option]

The above form of the PUT command directs printer output to a file.

The options are the same as in the PUT CONSOLE command, except that option NO ECHO is the default for the PUT PRINTER command. The printer output is echoed to the printer only if you specify the ECHO option.

Examples:

PUT PRINTER TO FILE XOUT<cr>
MYPROG<cr>

Directs the printer output of program MYPROG to file XOUT. The output is not echoed to the printer.

PUT PRINTER TO FILE XOUT2
[ECHO,SYSTEM] <cr>

Directs all printer output to file XOUT2 and to the console, and the PUT is in effect until you enter a PUT PRINTER OUTPUT TO PRINTER command.

The printer output can be directed to one or more files. The output to these files is terminated when you revert printer output to the printer using the command: PUT PRINTER TO PRINTER.

Format:

PUT CONSOLE TO CONSOLE [option]

The above form of the PUT command directs console output to the console.

Example:

```
PUT CONSOLE TO CONSOLE<cr>
```

Directs console output to the console.

Format:

```
PUT PRINTER TO PRINTER
```

The above form of the PUT command directs the printer output to the printer.

Example:

```
PUT PRINTER TO PRINTER<cr>
```

Directs printer output to the printer.

RENAME—*assigns a new name to a file*

The RENAME command lets you change the name of a file that is cataloged in the directory of a disk. It also lets you change several file names if you use wildcards in the filespecs. You can abbreviate RENAME to REN.

Format:

```
RENAME filespec=filespec
```

The new filespec (to the left of the equals sign) must not match the name of any existing file on the disk. The old filespec (to the right of the equals sign) identifies an existing file or files on the disk.

The RENAME command does not make a copy of the file, it only changes the name of the file.

If you omit the drive specifier, RENAME assumes that the file to rename is on the default drive. You can include a drive specifier as a part of either filespec. If both file specifications name a drive, it must be the same drive.

If the file given by the old filespec does not exist, RENAME displays the following message on the screen:

```
No File
```

If the file given by the new filespec is already present in the directory, RENAME displays the following message on the screen:

```
Not renamed: filename.typ already exists,
delete (Y/N)?
```

If you want to delete the old file, type Y to delete. Otherwise, type N to keep the old file and not rename the new file. If you use wildcards in the filespecs, the wildcards in the new filespec must correspond exactly to the wildcards in the old filespec.

Examples:

RENAME NEWASM.BAS=OLDFILE.BAS<cr>

The file OLDFILE.BAS changes to NEWASM.BAS on the default drive.

RENAME<cr>

The system prompts for the filespecs:

```
Enter New Name:
Enter Old Name:
File____is____renamed____on drive__
```

REN A:X.PAS=Y.PLI<cr>

The file Y.PLI changes to X.PAS on drive A.

RENAME S*.TEX=A*.TEX<cr>

Renames all the files matching the wildcard A*.TEX to files with file names matching the wildcard S*.TEX, respectively.

REN B:NEWLIST=B:OLDLIST<cr>

The file OLDLISTchanges to NEWLISTon drive B. Since the second drive specifier, B: is implied by the first one, it is unnecessary in this example. The command line above has the same effect as the following:

REN B:NEWLIST=OLDLIST
or
REN NEWLIST=B:OLDLIST

RMAC—*relocatable macro assembler*

RMAC is a relocatable macro assembler, assembling files of type .ASM into .REL files that can be linked to create .COM files.

Format:

RMAC filespec $options

The options of the RMAC command specify the destination of the output files. Here are the options that can be used with the RMAC command:

Option	d=output option
R– drive for REL file	(A–O, Z)
S– drive for SYM file	(A–O, X, P, Z)
P– drive for PRN file	(A–O, X, P, Z)

A–O specifies drive A–O.
X means output to the console.
P means output to the printer.
Z means zero output.

In the MAC command, the assembly parameter of H controls the destination of the .HEX file. In the RMAC command, this parameter is replaced by R which controls the destination of the .REL file; however, you cannot direct the .REL file to the console or printer (RX or RP) because the .REL file is not an ASCII (printable) file.

Example:

```
RMAC TEST $PX SB RB<cr>
```

In the above example RMAC assembles the file TEST.ASM from drive A, sends the listing file (TEST.PRN) to the console, puts the symbol file (TEST.SYM) on drive B and puts the relocatable object file (TEST.REL) on drive B.

SAVE—*saves the contents of memory to a file*

The SAVE command copies the contents of memory to a file. To use the SAVE utility, first issue the SAVE command, then run a program that reads a file into memory. When your program exits, it exits to the SAVE utility, which prompts you for the filespec to which memory is to be copied, and the beginning and ending address of the memory to be saved.

Format:

SAVE

Example:

SAVE<cr>

The above command activates the SAVE utility. Now enter the name of the program that loads a file into memory.

SID dump.com

Next, execute the program.

#gO

When the program exists, SAVE intercepts the return to the system and prompts you for the filespec and the bounds of memory to be saved.

```
SAVE Ver 3.0

File (or RETURN to exit)? dump2.com

Delete dump2.com? Y

From? 100

To? 400
```

The contents of memory from 100H (Hexadecimal) to 400H are copied to file DUMP2.COM.

SET*—initiates attributes for files*

The SETcommand initiates password protection and time stamping of files in the CP/M Plus operating system. It also sets file and drive attributes, such as the Read-Only, SYS, and user-definable attributes. It lets you label a disk and password-protect the label.

The SETcommand includes options that affect the disk directory, the drive, a file or set of files. The discussion of the SETcommand explicitly states which of the three categories are affected.

To enable time stamping of files, you must first run INITDIR to format the disk directory.

Format:

SET d: [NAME=diskname.typ]

The above SETcommand assigns a name to the disk in the specified or default drive.

CP/M Plus provides a facility for creating a directory label for each disk. The directory label can be assigned an eight-character name and a three-character type similar to a file name and file type. Label names make it easier to catalog disks and keep track of different disk directories. The default label name is LABEL.

Example:

SET [NAME=DISK100] <cr>

Labels the disk on the default drive DISK100.

Formats:

```
SET [PASSWORD=password]

SET [PASSWORD=<cr>
```

The first form of the above SETcommand assigns a password to the disk label. The second form of the command removes password protection from the label.

You can assign a password to the label. If the label has no password, any user who has access to the SET program can set other attributes to the disk that may make the disk inaccessible to you. However, if you assign a password to the label, then you must supply the password to set any of the functions controlled by the label. SETalways prompts for the password if the label is password-protected.

You can always determine if a disk is password-protected by using the SHOW command to display the label.

Example:

```
SET [PASSWORD=SECRET]<cr>
SET [PASSWORD=<cr>
```

The first command assigns the password SECRET to the disk label. The second command nullifies the existing password.

NOTE

If you use password protection on your disk, be sure to record the password. If you forget the password, you lose access to your disk and/or files.

Formats:

SET [PROTECT=ON]

SET [PROTECT=OFF]

The first SET command above turns on password protection for all the files on the disk. The password protection must be turned on before you can assign passwords to individual files or commands.

The second SET command disables password protection for the files on your disk.

After a password is assigned to the label and the PROTECT option is turned on, you are ready to assign passwords to your files.

SET filespec [PASSWORD=password]

The above SET command sets the password for the given filespec to the password indicated in the command tail. Passwords can be up to eight characters long. Lowercase letters are translated to uppercase.

You can use wildcards in the filespec. SET assigns the specified password to all files that match the wildcard filespec.

NOTE

Always record the passwords that you assign to your files. Without the password, you cannot access those files unless password protection is turned off for the whole disk. If you forget the password to the directory label, you cannot turn off the password protection for the disk.

Example:

```
SET MYFILE.TEX [ PASSWORD=MYFIL] <cr>
```

MYFIL is the password assigned to file MYFILE.TEX.

Formats:

```
SET filespec [ PROTECT=READ]

SET Filespec [ PROTECT=WRITE]

SET filespec [ PROTECT=DELETE]

SET filespec [ PROTECT=NONE]
```

You can assign one of four modes of password protection to your file. The protection modes are READ, WRITE, DELETE, and NONE and are described in the following table. Here are the password protection modes available in the SETcommand:

Modes and Their Protection

READ The password is required for reading, copying, writing, deleting, or renaming the file.

WRITE The password is required for writing, deleting, or renaming the file. You do not need a password to read the file.

DELETE The password is required only for deleting or renaming the file. You do not need a password to read or modify the file.

NONE No password exists for the file. If password exists, this modifier can be used to delete the password.

Example:

```
SET *.TEX [ PASSWORD=SECRET, PROTECT=WRITE] <cr>
```

Assigns the password SECRET to all the .TEX files on the default drive and current user number to prevent unauthorized editing.

Format:

```
SET [ DEFAULT=password]
```

The above set command assigns a default password for the system to use during your computer session. The system uses the default password to access password-protected files if you do not specify a password, or if you enter an incorrect password. The system lets you access the file if the default password matches the password assigned to the file.

Example:

SET [DEFAULT=dd] <cr>

Instructs the system to use "dd" as a password if you do not enter a password for a password-protected file.

Formats:

SET [CREATE=ON]

SET [ACCESS=ON]

SET [UPDATE=ON]

The above SETcommands allow you to keep a record of the time and date of file creation and update, or of the last access and update of your files. Here are the available options.

Options and Their Actions

[CREATE=ON] Turns on CREATE time stamps on the disk in the default drive. To record the creation time of a file, the CREATE option must have been turned on before the file is created.

[ACCESS=ON] Turns on ACCESS time stamps on the disk in the default drive. ACCESS and CREATE options are mutually exclusive. This means that only one can be in effect at a time. If you turn on the ACCESS time stamp on a disk that has the CREATE time stamp, the CREATE time stamp is automatically turned off.

[UPDATE=ON] Turns on UPDATE time stamps on the disk in the default drive. UPDATE time stamps record the time the file was last modified.

To enable time stamping, you must first run INITDIR to format the disk directory for time and date stamping.

Although there are three kinds of date/time stamps, two date/time stamps can be associated with a given file at one time. You can choose to have either a CREATE date or an ACCESS date for files on a particular disk.

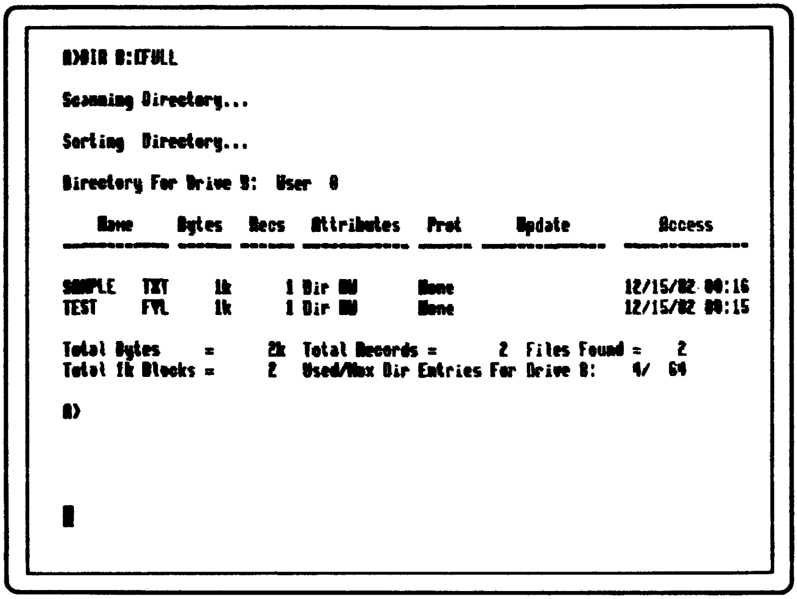
If you set both UPDATE and CREATE time stamps you will notice that editing a file changes both the UPDATE and CREATE time stamps. This is because ED does not update the original file but creates a new version with the name of the original file.

Examples:

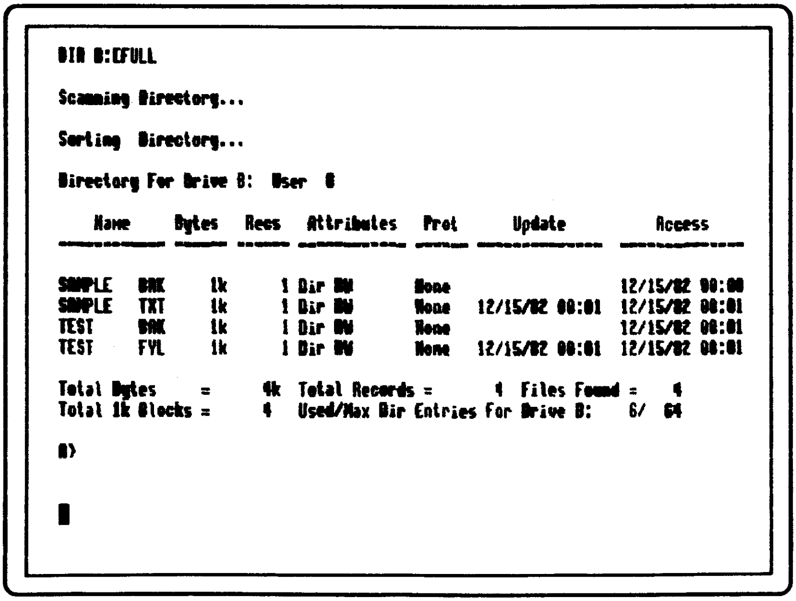
```
SET [ ACCESS=ON] <cr>
```

Once the ACCESS time-stamping option has been turned on, the DIR command used with the [FULL] option will display the following date and time stamps:

```
DIR [ FULL] <cr>
```



```
A>DIR B:[FULL

Scanning Directory...

Sorting  Directory...

Directory For Drive B:  User  0

    Name     Bytes  Recs  Attributes  Prot     Update          Access
  ---------- ------ ----- ----------- ------ -------------  --------------

SAMPLE  TXT    1k      1 Dir RW       None                   12/15/82 00:16
TEST    FYL    1k      1 Dir RW       None                   12/15/82 00:15

Total Bytes     =      2k  Total Records =       2  Files Found =    2
Total 1k Blocks =      2   Used/Max Dir Entries For Drive B:    4/  64

A>
```



```
DIR B:[FULL

Scanning Directory...

Sorting  Directory...

Directory For Drive B:  User  0

    Name     Bytes  Recs  Attributes  Prot     Update          Access
  ---------- ------ ----- ----------- ------ -------------  --------------

SAMPLE  BAK    1k      1 Dir RW       None                   12/15/82 00:00
SAMPLE  TXT    1k      1 Dir RW       None  12/15/82 00:01   12/15/82 00:01
TEST    BAK    1k      1 Dir RW       None                   12/15/82 00:01
TEST    FYL    1k      1 Dir RW       None  12/15/82 00:01   12/15/82 00:01

Total Bytes     =      4k  Total Records =       4  Files Found =    4
Total 1k Blocks =      4   Used/Max Dir Entries For Drive B:    6/  64

A>
```

The access time stamps displayed show the time the file was last displayed or edited. Note that displaying a file name in a directory listing does not consititute an access and is not recorded.

```
SET [ CREATE=ON,UPDATE=ON ] <cr>
```

The following DIR output shows how files with create and update time stamps are displayed.

```
DIR [ FULL ] <cr>
```

Format:

```
SET filespec [ Attribute ]
```

The above SET command sets the specified attributes of a file or a group of files. Here are the attributes that can be used in the SET command:

Attributes and Their Meanings

ARCHIVE=OFF Sets the archive attribute to OFF. This means that the file has not been backed up (archived).

ARCHIVE=ON Sets the archive attribute to ON. This means that the file has been backed up (archived). The archive attribute can be turned on explicitly by the SET command, or it can be turned on by PIP when copying a group of files with the PIP [A] option. This PIP option requires a file specification and copies only files that have been changed since the last time they were backed up with the PIP [A] option. PIP then sets the archive attribute for each file successfully copied. The archive attribute is displayed by both SHOW and DIR.

DIR Gives the file the DIR attribute.

RO Sets the file attribute to allow Read-Only access.

RW Sets the file attribute to allow Read-Write access.

SYS Gives the file the SYS attribute.

F1=ON or OFF Turns on or off the user-definable attribute F1.

F2=ON or OFF Turns on or off the user-definable file attribute F2.

F3=ON or OFF Turns on or off the user-definable file attribute F3.

F4=ON or OFF Turns on or off the user-definable file attribute F4.

Examples:

SET MYFILE.TEX [RO SYS] <cr>

Sets MYFILE.TEX to Read-Only and SYStem.

SET *.COM
[SYS,RO,PASS=123,PROT=READ] <cr>

This setting affords the most protection for all the .COM files on drive A. The password-protection mode is set to READ. Therefore, you cannot even read one of the .COM files without entering the password 123, unless the default password has been set to 123. Even if the correct password is entered, you still cannot write to the file because the file is Read-Only.

SET *.COM
[RW,PROTECT=NONE,DIR] <cr>

Reverses the protection and access attributes of the .COM files affected by the previous example. After executing the above command, there is no password protection; the files of type .COM can be read from or written to, and are set to .DIR files.

Formats:

SET d: [RO]

SET d: [RW]

The above SET commands set the specified drive to Read-Only or Read-Write. If a drive is set to Read-Only, PIP cannot copy a file to it, ERASE cannot delete a file from it, and RENAME cannot rename a file on it. You cannot perform any operation that requires writing to the disk. When the specified drive is set to Read-Write, you can read or write to the disk in that drive.

Example:

SET B: [RO] <cr>

Sets drive B to Read-Only.

SETDEF—*displays and sets drive and file search order*

The SETDEF command lets you display or define the d'sk search order, the temporary drive, and the file-type search order. The SETDEF definitions affect only the loading of programs and/or execution of SUBMIT (SUB) files.

The SETDEF command also lets you turn on or off the DISPLAY and PAGE modes for the system. When DISPLAY mode is on, the system displays the location and name of programs loaded or .SUB files executed. When PAGE MODE is on, CP/M Plus utilities stop after displaying one full screen of information. You can then press any key to continue the display.

Formats:

```
SETDEF
```

The above form of the SETDEF command displays the disk search order, the temporary drive, and the file-type search order.

```
SETDEF [ TEMPORARY=D: ]
```

The above form of the SETDEF command defines the disk drive to be used for temporary files. The default drive used for temporary files is also the system default drive.

Example:

```
SETDEF [ TEMPORARY=A: ] <cr>
```

Sets disk drive A as the drive to be used for temporary files.

Format:

```
SETDEF d:,d:,...
```

The above form of the SETDEF command defines the disks to be searched by the system for programs and/or SUBMIT files to be executed. The CP/M Plus default is to search only the default drive.

NOTE

** can be substituted for d: to indicate that the default drive is to be included in the drive search order.*

Example:

```
SETDEF B:,*<cr>
```

Tells the system to search for a program on drive B, then, if not found, search for it on the default drive.

Format:

```
SETDEF [ ORDER=(typ,typ)]
```

The above form of the SETDEF command defines the file-type search order to be used by system for program loading. The file type, indicated as typ in the syntax line, must be .COM or .SUB. THE CP/M Plus default search is for .COM files only.

Example:

```
SETDEF [ ORDER=(SUB,COM)] <cr>
```

Instructs the system to search for a .SUB file to execute. If no .SUB file is found, search for a .COM file.

Formats:

SETDEF [DISPLAY]

SETDEF [NO DISPLAY]

The above commands turn on or off the system display mode. When the display mode is set to ON, the system displays the drive, file name, file type (if any), and user number (if not the default user number) of the currently executing program or SUBMIT file.

Examples:

SETDEF [DISPLAY] <cr>

Turns on the system display mode. Henceforth, the system displays the name and location of programs loaded or submit files executed. For example, if you enter the PIP command after turning on the system display mode, CP/M Plus displays the following:

```
A > PIP

A:PIP COM

CP/M 3 PIP VERSION 3.0

*
```

This indicates that the file PIP.COM was loaded from drive A under the current user number. If the current user number is not 0, and if PIP.COM does not exist under the current user number, then the system displays the location of PIP.COM as follows:

```
4A > PIP

A:PIP COM (User 0)

CP/M 3 PIP VERSION 3.0

*
```

This indicates that PIP.COM was loaded from drive A under user number 0. This mode is in effect until you enter:

SETDEF [NO DISPLAY] <cr>

to turn off the system DISPLAY mode.

Formats:

SETDEF [PAGE]

SETDEF [NO PAGE]

The above commands turn on or off the system page mode. When the PAGE mode is set to ON, CP/M Plus utilities stop after displaying one full screen of information, called a console page. The utilities resume after you press any key. The default setting of the system page mode is ON.

Example:

SETDEF [NO PAGE] <cr>

Turns off the system page mode. CP/M Plus utilities do not pause after displaying a full console page, but continue to scroll.

SHOW—*displays information about files and diskette space*

The SHOW command displays information about the access mode and the amount of free disk space, the disk label, the current user number, the number of files for each user number on the disk, the number of free directory entries for the disk, and drive characteristics.

Format:

```
SHOW d:
```

The above form of the SHOW command displays the drive, the access mode for that drive, and the remaining space in kilobytes for the specified drive. SHOW without a drive specifier displays the information for all logged-in drives in the system.

Examples:

```
SHOW B: <cr>
     B: RW, Space: 9,488k

SHOW    <cr>

     A: RO, Space:     4k
     B: RW, Space: 9,488k
```

The first example shows that drive B has Read-Write access and 9,488K bytes of space left. The second example shows that drive A is Read-Only and has 4K bytes left and drive B is Read-Write and has 9,488K bytes left.

Format:

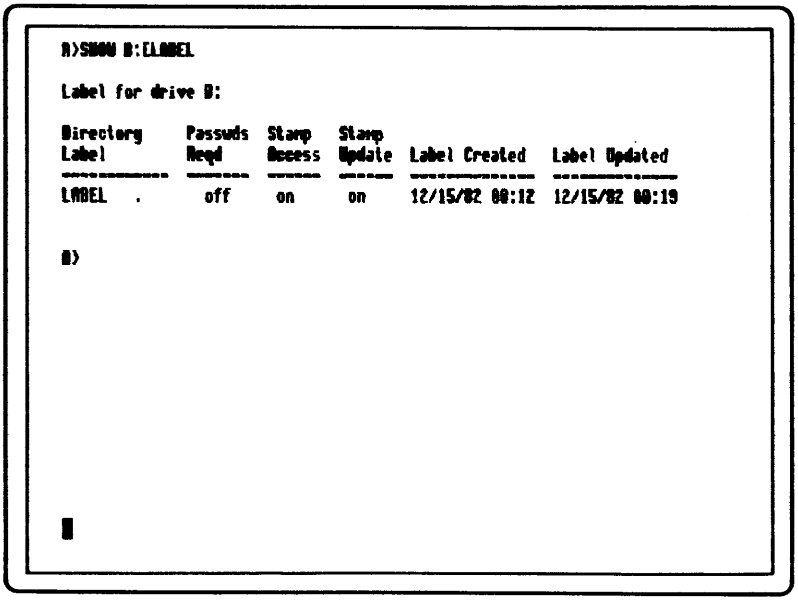
SHOW d: [LABEL]

The above form of the SHOW command displays disk label information.

Example:

SHOW B: [LABEL] <cr>

The above command displays the following for drive B:



```
A>SHOW B:[LABEL

Label for drive B:

Directory      Passwds  Stamp   Stamp
Label          Reqd     Access  Update  Label Created   Label Updated
------------   -------  ------  ------  --------------  --------------
LABEL    .       off     on      on     12/15/82 08:12  12/15/82 08:19


A>
```

The first column, Directory Label, displays the name assigned to that drive directory. The second column, Passwds Reqd, shows that password protection has been turned on for that drive.

As described in the SETcommand, each file can have up to two time stamps. The first of these time stamps can be either the creation date and time for the file or the time and date of the last access to the file. Access is defined as reading from or writing to the file. The third column of the SHOW LABEL output displays both the type of stamp and whether or not it is on. In the example above, creation time stamps are given to new files as shown by the Stamp Create column heading.

The fourth column displays the status of the second time stamp field. The update time stamps display the time and date of the last update to a file, that is, the last time someone wrote to the file. In the SHOW LABEL display, update time stamps are turned on.

In addition to showing the password protection and the active time stamps on a drive, SHOW LABEL also displays the date and time that the label was created and last updated.

Format:

```
SHOW d: [USERS]
```

The above command displays the current user number, all the users on the drive, and the number of files assigned to each user number.

Example:

```
SHOW USERS<cr>
```

```
Active User:      1
Active Files:     0      2      3      4
A: # of files:   95     40      1     26
A: Number of free directory entries:
350
A>
```

Format:

```
SHOW d: [DIR]
```

The above command displays the number of free directory entries on the specified drive.

Example:

```
SHOW C:[DIR]<cr>
```

```
C: Number of free directory entries:
24
A>
```

The above command shows that there are only 24 free directory entries in drive C.

Format:

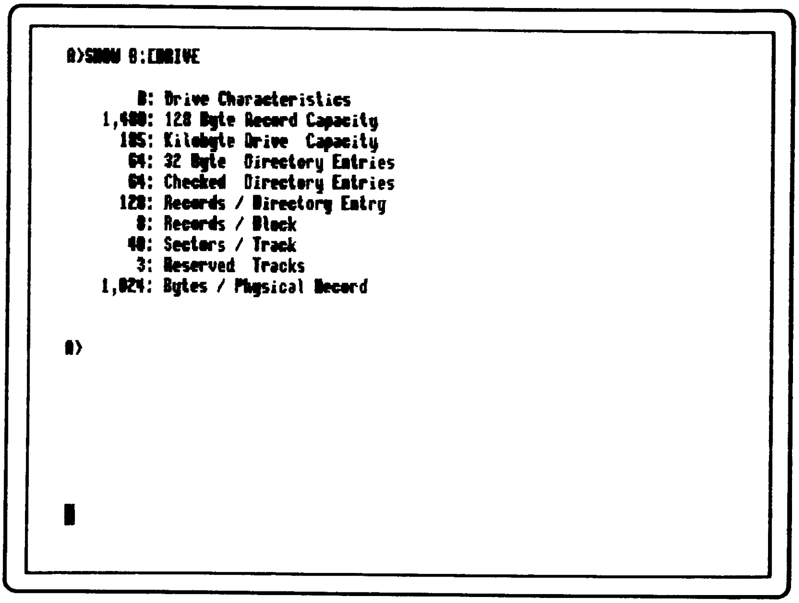
```
SHOW d: [DRIVE]
```

The above form of the SHOW command displays the drive characteristics of the specified drive.

Example:

```
SHOW [DRIVE]<cr>
```

The following is an example of the system display for the above command:



```
A>SHOW B:[DRIVE

        B: Drive Characteristics
    1,400: 128 Byte Record Capacity
      185: Kilobyte Drive  Capacity
       64: 32 Byte  Directory Entries
       64: Checked  Directory Entries
      128: Records / Directory Entry
        8: Records / Block
       40: Sectors / Track
        3: Reserved  Tracks
    1,024: Bytes / Physical Record


A>
```

SID—*symbolic instruction debugger*

The following discussion is a brief description of the SID command. The SID User's Guide describes SID in detail.

The SID symbolic debugger allows you to monitor and test programs developed for any 8080-compatible microprocessor. SID supports real-time breakpoints, fully monitored execution, symbolic disassembly, assembly, memory display and fill functions. Utility programs can be dynamically loaded with SID to provide traceback and histogram facilities.

Use SID commands to display memory and CPU registers, and direct the breakpoint operations during the debugging session.

Format:

SID

If invoked without a command tail, SID loads into memory without a test program. Use this form to examine memory or to write and test simple programs using the Assemble command. You must not use the SID commands G, T, or U described below until you have first loaded a test program.

Example:

SID<cr>

CP/M Plus loads SID from the default drive into memory. SID displays the # prompt when it is ready to accept commands.

Format:

SID filename,filename

A SID command line with a program file name (the first file name in the command line) loads both SID and the test program into memory. If the file type is omitted, .COM is assumed. SID optionally loads in a symbol table file specified by the second file name in the command tail. The symbol–file name needs no file type because SID assumes a file of file type .SYM. Use the C, G, T, or U command to begin execution of the test program under supervision of SID.

Examples:

B:SID SAMPLE.HEX<cr_

CP/M Plus loads SID and the program file SAMPLE.HEX into memory from drive B. SID displays:

```
NEXT MSZE  PC  END
nnnn mmmmm pppp eeee
```

where *nnnn* is a hexadecimal address of the next free location following the loaded program, and mmmm is the next location after the largest program. This is initially the same value as NEXT. pppp is the initial hexadecimal value of the program counter. eeee is the hexadecimal address of the logical end of the TPA.

#DFE00+ #128,+5

In the above example the first pound sign, #, is the SID prompt. This SID command, D, displays the values stored in memory starting at the value stored at address FE80(FE00 + #128) and ending with the value stored at address FE85 (FE80 + 5).

Use ^S to halt the screen display, ^Q to restart the display. Abort lengthy displays by typing any keyboard character. Use ^C to exit from SID.

SID can address absolute memory locations through symbolic expressions. A symbolic expression evaluates to either an address or a data item.

A symbolic expression can be a name from the .SYM file produced from your program by a CP/M macro assembler. When you precede the symbolic expression with a period, SID returns its address in hexadecimal. When you precede the symbolic expression with the at sign (@), SID returns the 16-bit value stored at that location and the next contiguous location. When you precede the symbolic expression with an equals sign, SID returns the 8-bit value stored at that location. For 2-byte expressions, this is the low byte because the 8080 microprocessor stores the low byte of a word value first.

A symbolic expression can be a literal value in hex, decimal, or ASCII, as indicated in the following list:

Hex SID uses literal hex values as given, but truncates any digits in excess of four on the left. The leftmost digit is the most significant digit. The rightmost digit is the least significant digit.

Decimal To indicate decimal values, precede them with a pound sign (#). Decimal values that evaluate to more than four hex digits are evaluted as the module of hex value FFFF. For example, #65534 = FFFEH, while #65536 = 0001H.

ASCII SID translates literal ASCII character strings between apostrophes to the hex value of the two rightmost ASCII characters. Blanks are evaluated as 20 hex.

You can combine symbolic expressions with the symbolic operators + or − to produce other symbolic expressions. Symbolic expressions combined in this way can be used to calculate the offset of an indirectly addressed data item, for example, a subscripted variable. A special up-arrow operator (↑) can reference the top-of-stack item. A string of *n* ^ operators can reference stack items without changing stack content or the stack pointer.

The table which follows lists the SID commands with their corresponding parameters and options. The actual command letter is printed in uppercase boldface. The parameters are in lowercase and follow the command letter. Optional items are in braces. Replace the arguments with the appropriate symbolic expressions as listed. Where two symbolic expressions are needed, SID can calculate the second one from the first using the symbolic operators decribed above.

Name	Syntax	Meaning
Assemble	**A**s	Enter assembly-language statements. s is the start address.
Call	**C**s {b{,d}}	Call to memory location from SID. s the called address, b is the value of the BC register pair, and d is the value of the DE register pair.
Display	**D**{W}{s}{,f}	Display memory in hex and ASCII. W is a 16-bit word format, s is the start address, and f is the finish address.
Load	**E**pgm-filespec {,sym-filespec}	Load program and symbol table for execution.
Load	**E*** sym-filespec	Load a symbol table file.

Name	Syntax	Meaning
Fill	**F**s, f, d	Fill memory with constant value. s is the start address, f is the finish address, and d is an 8-bit data item.
Go	**G**{p}{,a{,b}}	Begin execution. p is a start address, a is a temporary breakpoint, and b is a second temporary breakpoint.
Hex	**H** **H**a **H**ab	Display all symbols with addresses in hex. The first syntax displays hex, decimal, and ASCII values of a. The second syntax performs number and character conversion, where a is a symbolic expression, and the third syntax computes hex sum and difference of a and b, where a and b are symbolic expressions.
Input	**I**command tail	Input CCP command line.
List	**L**{s}{,f}	List 8080 mnemonic instructions. s is the start address, and f is the finish address.
Move	**M**s, h,d	Move memory block. s is the start address, h is the high address of the block, and d is the destination start address.
Pass	**P**{p{,c}}	Pass point set, reset, and display. p is a permanent breakpoint address, and c is initial value of the pass counter.
Read	**R**filespec{,d}	Read code/symbols. d is an offset to each address.
Set	**S**{W}s	Set memory values. s is address where value is sent, W is 16-bit word.
Trace	**T**{*n*{,c}}	Trace program execution. *n* is the number of program steps, and c is the utility entry address.

Name	Syntax	Meaning
Trace	**T**{W}{***n***{,c}}	Trace without call. W instructs SID not to trace subroutines, *n* is the number of program steps, and c is the utility entry address.
Untrace	**U**{W}{n{,c}}	Monitor execution without trace. *n* is the number of program steps, c is the utility entry address, W instructs SID not to trace subroutines.
Value	**V**	Display the value of the next available location in memory (NEXT), the location after the largest file read in (MSZE), the current value of the Program Counter (PC), and the address of the end of available memory (EWND).
Write	**W**filespec{,s,f}	Write the contents of a contiguous block of memory to filespec. s is the start address, f is the finish address.
Examine	**X**{f}{r}	Examine/alter CPU state. f is flag bit C, Z, M, E, or I; r is register A, B, D, H, S or P.

SID Utilities

The SID utilities HIST.UTL and TRACE.UTL are special programs that operate with SID to provide additional debugging facilities. The mechanisms for system initialization, data collection, and data display are described in the CP/M SID User's Guide. The following discussion illustrates how a utility is activated. You load the utility by naming it as a parameter when invoking SID:

SID filename.UTL<cr>

In the above example, file name is the name of the utility. Following the initial sign-on, the utility can prompt you for additional debugging parameters.

The HIST utility creates a histogram (bar graph) showing the relative frequency of execution of code within selected program segments of the test program. The HIST utility allows you to monitor those sections of code that execute most frequently.

On start-up, HIST prompts:

```
TYPE HISTOGRAM BOUNDS
```

Respond with

aaaa,bbbb

for a histogram between locations aaaa and bbbb inclusive. Collect data in U or T mode, then display results.

The TRACE utility obtains a backtrace of the instructions that led to a particular breakpoint address in a program under test. You can collect the addresses of up to 256 instructions between pass points in U or T modes.

SUBMIT—*allows batch processing of commands*

The SUBMIT command lets you execute a group or batch of commands from a SUBMIT file, which is a file containing a list of CP/M Plus commands that has a file type of .SUB.

Normally, you enter CP/M Plus commands a line at a time. If you must enter the same sequence of commands several times, you might find it easier to batch the commands together using the SUBMIT command. To do this, create a file using a text editor, and enter your commands into this file. The file is identified by the file name, and must have a file type of .SUB. When you issue the SUBMIT command, SUBMIT reads the file named by the filespec and prepares it for interpretation by CP/M Plus. When the preparation is complete, SUBMIT sends the file to CP/M Plus line by line, as if you were typing each command.

The SUBMIT command executes the commands from a SUB file as if you were entering the commands from the keyboard. The .SUB file can contain CP/M Plus commands, nested SUBMIT commands, and input data for a CP/M Plus command or program.

Formats:

SUBMIT filename argument argument ...

You can pass arguments to .SUB files at execution time. Each argument you enter is assigned to a parameter in the .SUB file. The first argument replaces every occurrence of $1 in the file, the second argument replaces parameter $2, and so on, up to parameter $9. For example, if the file START.SUB contains the following commands:

```
ERA $1.BAK
DIR $1
PIP $1=A:$2.COM
```

and you enter the command

```
SUBMIT START SAM TEX<cr>
```

the argument SAM is substituted for every $1 in the START.SUB file, and TEX for every occurrence of $2 in the START.SUB file. SUBMIT then creates a file with the parameter substitutions and executes this file. This file now contains the following commands:

```
ERA SAM.BAK
DIR SAM
PIP SAM=A:TEX.COM
```

If you include fewer arguments in the SUBMIT command than parameters in the .SUB file, the remaining parameters are not included in the commands. If you enter more arguments in the SUBMIT command than parameters in the .SUB file, the remaining arguments are ignored.

To include an actual dollar sign ($) in your .SUB file, type two dollar signs, $$. SUBMIT replaces them with a single dollar sign when it substitutes an argument for a parameter in the .SUB file. For example, if file AA.SUB contains line:

```
MAC $1 $$$2
```

and you enter the command

```
SUBMIT AA ZZ SZ<cr>
```

then the translated file contains the following:

```
MAC ZZ $SZ
```

Program Input Lines in a .SUB File

A .SUB file can contain program input lines. Any program input is preceded by a less than sign (<) as shown in the following example:

```
PIP
<B:=*.ASM
<CON:=DUMP.ASM
<
DIR
```

The three lines after PIP are input lines to the PIP command. The third line consists only of the < sign, indicating a carriage return. The carriage return causes PIP to return to the system to execute the final DIR command.

If the program terminates before using all of the input, SUBMIT ignores the excess input lines and displays the following warning message:

```
Warning: Program input ignored
```

If the program requires more input than is in the .SUB file, it expects you to enter the remaining input from the keyboard.

You can enter control characters in a .SUB file by using the normal convention of preceding the control character by an up-arrow, (^), followed by the letter to be converted to a control character. To enter an actual ^ character, use the combination ^^. This combination translates to a single ^ in the same manner that $$ translates to a single $.

The .SUB File

The .SUB file can contain the following types of lines:

- **Any valid CP/M Plus command**
- **Any valid CP/M Plus command with SUBMIT parameters**
- **Any data input line**
- **Any program input line with parameters ($1 to $9)**

CP/M Plus command lines cannot exceed 136 characters.

Example:

The following lines illustrate the variety of lines that can be entered in a .SUB file:

```
DIR
DIR *.BAK
MAC $1 $$$4
PIP LST:=$1.PRN [ T$2 $3 $5]
DIR *.ASM
PIP
<B:=*.ASM
<CON:=DUMP.ASM
<
DIR B:
```

You can then type:

```
SUBMIT<cr>
```

and the system prompts:

```
Enter File to Submit:
```

Enter the filespec and arguments here, such as:

```
START B TEX
```

Another example could be:

```
SUBMIT SUBA
```

Still another example using parameters is:

```
SUBMIT AA ZZ SZ
```

where AA is the .SUB file AA.SUB, ZZ is the argument to replace any occurrences of $1 in the AA.SUB file, and SZ is the argument to replace all occurrences of $2 in the AA.SUB file.

PROFILE.SUB—*automatically starts a command file*

Every time you turn on or reset your computer, CP/M Plus automatically looks for the special .SUB file named PROFILE.SUB to execute. If PROFILE.SUB does not exist on the default system drive and user number, then CP/M Plus resumes normal operation. If the PROFILE.SUB file exists, the system executes the commands in the file. This file is convenient to use if you regularly execute a set of commands before doing other work on the computer. For example, if you want to be sure that you always enter the current date and time on your computer before you enter any other commands, you can create the PROFILE.SUB file with a text editor, and enter the DATE command as follows:

```
DATE [SET] <cr>
```

Then, whenever you bring up the system, the system executes the DATE command and prompts you to enter the date and time. By using this facility, you can be sure to execute a regular sequence of commands before starting your normal session.

TYPE—*displays the contents of a named ASCII file*

The TYPE command displays the contents of an ASCII character file on your screen. TYPE displays the console listing in paged mode, which means that the console listing stops automatically after listing *n* lines of text, where *n* is usually the system default of 24 lines per page. (See the DEVICE command to set *n* to a different value.) Press ^C to exit back to the system. Paged mode is the default display mode.

Format:

TYPE filespec [NO PAGE]

The NO PAGE option, if present, displays the console listing continuously. If you do not enter a file specification in the TYPE command the system prompts for a file name with the message:

```
Enter filename:
```

Respond with the filespec of the file you want listed.

Tab characters occurring in the file named by the file specification are expanded to every eighth column position of your screen.

You can interrupt the listing, at any time during the display, by pressing ^S. Press ^Q to resume the listing.

Press ^C to exit back to the system.

Make sure the file specification identifies a file containing character data.

If the file named by the file specification is not present on the specified drive, TYPE displays the following message on your screen:

```
No File
```

To list the file to the printer as well as on the screen, type a ^P before entering the TYPE command line. To stop echoing console output at the printer, type a second ^P. The type command displays the contents of the file until the screen is filled. It then pauses until you press any key to continue the display.

Examples:

TYPE MYPROG.PLI<cr>

This command displays the contents of the file MYPROG.PLI on your screen 24 lines at a time.

TYPE B:THISFILE [NO PAGE] <cr>

This command continuously displays the contents of the file THISFILE from drive B on your screen.

USER—*establishes user number*

The USER command sets the current user number. The disk directory can be divided into distinct groups according to a user number. User numbers range from 0 through 15.

When CP/M Plus starts, 0 is the current user number. Any files you create under this user number are not generally accessible under any other user number, except through the PIP command, or the System (SYS) attribute as assigned with the SETcommand. (See the [G] option of the PIP Utility.)

Format:

USER (number)

Examples:

USER<cr>

The system command prompts for the user number, as follows:

```
Enter User#:
```

If you enter 5, for example, followed by RETURN the following message appears:

```
The current user number is now 5 on drive A
```

USER 3<cr>

This command changes the current user number to 3.

XREF—*provides a cross-reference of program variables*

The XREF command provides a cross-reference summary of variable usage in a program. XREF requires the .PRN and .SYM files produced by MAC or RMAC for the program.

The .SYM and .PRN files must have the same file name as the file name in the XREF command tail. XREF outputs a file of type .XRF.

Format:

XREF d: filename {$P}

Examples:

XREF b:MYPROG<cr>

In this example, XREF is on the A drive. XREF operates on the file MYPROG.SYM and MYPROG.PRN, which are on the B drive. XREF produces the file MYPROG.XRF on the B drive.

XREF b:MYPROG $P<cr>

In the above example, the $P option directs output to the printer.

CBASIC

Labels and Identifiers

CBASIC labels and identifiers must have 31 characters or less. If the last character is a %, then an integer numeric value is assumed. If the last character is a $, then a string variable is assumed. Otherwise, a real numeric variable is assumed.

Numbers:

Integer numbers can have values ranging between −32768 and 32767.

Real numbers are represented using standard decimal format; they can have up to 14 digits of precision. Larger numbers are represented using scientific notation. The mantissa has one digit in front of the decimal point. The exponent is a two-digit decimal number.

Line Numbers:

CBASIC statements do not need line numbers. Line numbers, where present, do not have to be in numeric sequence. Line numbers may be integers, real, or scientific-notation numbers. Line numbers with the same numeric value, but a different numeric type are treated as a different line number. For example, the following three line numbers have the same numeric value, but are treated as three different line numbers:

```
100
100.0
1.0E02
```

Expressions:

CBASIC evaluates expressions using the following evaluation hierarchy from highest to lowest:

1. Nested parentheses ( )

2. Exponent ^
3. Multiply *, divide /
4. Add +, subtract −, concatenate +, unary plus +, usary minus −
5. Relational operators LT or <, LE or < =, GT or >, GE or > =, EQ or =, NE or <>
6. NOT
7. AND
8. OR,XOR

Summary of CBASIC Statements

CBASIC statements are summarized below in alphabetic order. Each statement's format is given, along with an example.

CALL—*links to a subroutine*

The CALL statement calls an assembly-language subroutine. Also see SAVE, PEEK, and POKE statements.

Format:

CALL integer expression

The integer expression must evaluate to the absolute memory address of the entry point for the assembly-language subroutine being executed. An assembly-language return (RTN) instruction (executed out of the subroutine) will return execution to the next sequential CBASIC statement.

NOTE

CPU registers may be altered by the assembly-language subroutine.

Examples:

```
CALL 27248
CALL 2CA0H
CALL sub%
```

CHAIN — *transfers control from one program to another*

The CHAIN statement is used to transfer control from one executing CBASIC program to another.

Format:

CHAIN string expression

The string expression identifies the file name (and drive) for the next program to be executed. The selected file must be of type INT and it must exist on the specified drive. When no drive is specified, the active drive is assumed. A CHAIN statement causes the selected file to be loaded and the program held within the file to be executed. In addition, the return stack is reset, open files are closed, and a RESTORE statement is executed. A COMMON statement can be used to exchange data between chained programs.

Examples:

```
CHAIN "MAIN"
CHAIN DRIVE$ + ":"+ PRINT.MSG$
```

Sample Programs:

```
REM TRANSFERS CONTROL TO A PROGRAM OF
REM THE USER'S CHOICE DIRECT COMPILER
REM TO RESERVE EXTRA SPACE FOR
REM PROGRAM'S CONSTANT, CODE, DATA,
REM AND VARIABLE AREAS (32, 1000, 32, AND
REM 32 BYTES RESPECTIVELY) (TO PREVENT
REM OVERWRITING BY THE CHAINED
REM PROGRAM)

% CHAIN 32, 1000, 32, 32

INPUT "WOULD YOU LIKE TO RUN A PROGRAM (Y/N)?";RUN$

        IF RUN$ = "Y" THEN\
          INPUT "WHICH ONE?"; PROG.NAME$:\
CHAIN PROG.NAME$

  STOP

REM PROGRAM CHECKCHAIN—WHEN MAIN
REM PROGRAM CHAINS HERE A MESSAGE IS
REM PRINTED TO SHOW THE CHAIN WAS
REM SUCCESSFUL AND THEN CONTROL IS
REM TRANSFERRED BACK TO THE MAIN
REM PROGRAM

  PRINT "YOU HAVE SUCCESSFULLY CHAINED"
  PRINT "TO PROGRAM CHECKCHAIN"
  CHAIN "MAIN"
  STOP
```

CLOSE—*closes specified files*

The CLOSE statement closes open files.

Format:

CLOSE integer expression {,integer expression}

Each integer expression denotes an open file that is to be closed. Reference to a file that has not been opened will cause an error. When a file is closed, the file number is released, and the associated buffer space is returned to the system.

NOTE

CLOSE terminates any IF END statement that references the file being closed.

Examples:

CLOSE 1
CLOSE input.file.id%, temp.file.1%

Sample Program:

```
REM CREATE TWO NEW FILES, WRITE DATA TO
REM THEM, AND CLOSE THE FILES
  CREATE "NEWFILE.SEQ" AS 1
  CREATE "NEWFILE.RAN" RECL 25 AS 2
  FOR 1% = 1 TO 10
          PRINT #1; "RECORD NUMBER",1%
          PRINT #2,1%; "RECORD NUMBER",1%
  NEXT 1%
  CLOSE 1,2
```

COMMON—*specifies common variables*

The COMMON statement specifies simple and subscripted variables that are retained in a common area of memory and passed between chained programs.

Format:

COMMON variable {, variable}

NOTE

For a subscripted variable, the number of subscripts, not the actual subscript maximum dimensions, follows the parenthesis.

COMMON statements must be the first in a program, preceded only by REM statements or blank lines. Chained programs must begin with COMMON statements having coincident parameter lists. Common variables must be of the same type in all COMMON statements, and they must appear in the same sequence. Variable arrays must have the same number of dimensions, and each dimension must have the same maximum value.

Examples:

```
COMMON DATE$, NAME$,ACCOUNTS$(3)
COMMON SIZE%,ACCOUNT.LIMIT (2), COMPANY$
```

Sample Program:

```
REM TRANSFER CONTROLTO A PROGRAM OF
REM THE USER'S CHOICE

COMMON RET$

REM DIRECT COMPILER TO RESERVE EXTRA
REM SPACE FOR PROGRAM'S CONSTANT,
REM CODE, DATA, AND VARIABLE AREAS (32,
REM 1000, 32, AND 32 BYTES RESPECTIVELY)
REM (TO PREVENT OVERWRITING BY THE
REM CHAINED PROGRAM)

%CHAIN 32, 1000, 32, 32

  INPUT "WOULD YOU LIKE TO RUN A PROGRAM (Y/N)?";RUN$
            IF RUN$ = "Y" THEN \

        INPUT "WHICH ONE?"; PROG.NAME$:\
        INPUT "DO YOU WANT TO RETURN TO THIS PROGRAM?";RET$:\
        CHAIN PROG.NAME$

  STOP

REM PROGRAM CHECK2—WHEN MAIN
REM PROGRAM CHAINS HERE A MESSAGE
REM IS PRINTED TO SHOW THE CHAIN WAS
REM SUCCESSFUL AND THEN CONTROL IS
REM TRANSFERRED BACK TO THE MAIN
REM PROGRAM IF THE USER CHOSE THAT
REM OPTION

COMMON RET$

  PRINT "YOU HAVE SUCCESSFULLY CHAINED"
  PRINT "TO PROGRAM CHECK2"
  IF RET$ = "Y" THEN \
        CHAIN "MAIN2"

STOP
```

CONSOLE—*redirects print to the console*

The CONSOLE statement follows an LPRINTER statement and causes output to be diverted from the printer to the console. This statement can also be used to make console width adjustments.

Format:

CONSOLE

Following execution of a CONSOLE statement, PRINT statement output is directed to the console.

To adjust the console width, use the POKE statement. POKE the required character width to location 272. The new console width becomes effective after the next execution of a console statement. A zero width setting is considered infinite, so that new lines are never automatically started. The default console width is 80 characters.

Example:

```
CONSOLE
```

Sample Program:

```
LPRINTER
PRINT "THIS LINE WILL BE OUTPUT TO"
PRINT "THE PRINTER"

CONSOLE

PRINT "THIS LINE SHOULD APPEAR"
PRINT "ON THE CONSOLE"

LPRINTER WIDTH 30
PRINT "THE PRINTER WIDTH WAS SET TO 30"
PRINT "SO THIS SENTENCE";
PRINT "SHOULD BE PRINTED AS 4 LINES BY"
PRINT "THE PRINTER"

POKE 272, 30
CONSOLE

PRINT "THIS SHOULD APPEAR ON THE"
PRINT "CONSOLE SCREEN";
PRINT "THE POKE STATEMENT SET THE LINE"
PRINT "WIDTH TO 30"

STOP
```

CREATE—*creates a new file*

The CREATE statement is comparable to an OPEN statement except that it is used to activate a new file rather than an existing file. Any existing file with the same name is erased so that a new file can be created.

Format:

CREATE string expression
[RECL integer expression]
AS integer expression
[BUFF integer expression]
[RECS integer expression]

The expression following the keyword CREATE is a valid file name that identifies the file to be created. The integer expression following AS designates which of 20 available file numbers is assigned to the new file (see the READ#, PRINT#, IF END, and CLOSE statements). Each active file is assigned an identification number. This number is used in all subsequent references.

The integer expression following RECL specifies record length. When the length is specified, the file will contain fixed-length records that may be accessed randomly or sequentially. If no length is specified, then record length will vary, and the file must be accessed sequentially.

The BUFF and RECS portion of a CREATE statement either appear in conjunction, or are omitted. The integer expression following BUFF indicates how many disk sectors are to be maintained in memory. When using random access, you must specify one disk sector. A value of 1 is automatically assumed in the absence of the BUFF and RECS parameters. RECS identifies the size of a physical sector on the disk. This value is currently ignored and the system assumes a sector size of 128 bytes. (See the CLOSE statement for a CREATE programming example.)

DATA—*lists data constants*

A DATA statement defines constants. These constants are assigned to variables by READ statements.

Format:

DATA constant [,constant]

The DATA statement may be used to list string, integer, or real constants. DATA statements can be located anywhere in a program, except before a COMMON statement. Each DATA statement must occupy one line exclusively and cannot be continued on the next line. No other statement can appear on the same line with a DATA statement.

A list of constants is compiled from all DATA statements in a program. Constants are put into the list sequentially as they appear in the DATA statement(s) parameter list(s), in the same order that the DATA statements themselves appear. READ statements work down the list of constants, assigning the next sequential constant in the DATA list to the next variable in the READ statement parameter list. An attempt to read past the DATA statement list will cause a run-time error.

Example:

110 DATA 1, 2, 2.1, 22.1

Sample Program:

```
REM READ AND PRINT THE DATA LIST
DATA 123.987, 42, "MORE DATA", "EVEN MORE DATA"

READ REAL.NUM, INT,NUM%, STRING1$, STRING2$
PRINT REAL.NUM; STRING1$; INT.NUM%; STRING2$

STOP
```

DEF— *defines a line function*

The DEF statement defines either a single line function or multiple line function. The DEF keyword must appear before any reference to the actual function.

Format:

DEF FN.name [(parm {,parm})] =expression

This format illustrates a single line function. A function name always begins with FN.; the remainder of the function name may consist of any combination of letters, numbers, or decimal points including blanks. The function name must be a valid CBASIC variable name.

The function computes a value, which it assigns to the function name. Subsequently the function name is treated as a CBASIC program variable. The function name determines whether the function is of type real, integer, or string.

In a single line function the expression to the right of the equal sign is evaluated, and the result is returned via the function name.

Parameters ("parm") are listed within the expression. When the function is referenced in the body of the program, actual values must be specified for each parameter. These actual values are used when the expression is evaluated.

Format:

DEF FN.name [(parm {,parm})]

Multiple line functions begin with a DEF statement and end with a FEND statement. The DEF statement specifies the function name and any function parameters. Any group of statements can occur between the DEF and FEND statements. A RETURN statement must be present to terminate the multiple line function, in a manner analogous to a subroutine return. The value returned by the multiple line function is the last value assigned to the function name before the RETURN statement is executed.

Examples:

```
DEF FN.CIRCLE.AREA (R)=3.142 * (R^2)
THIS.AREA=FN.CIRCLE.AREA (5)
```

Sample Program:

```
DEF FN.CIRCLE.AREA (R)=3.142 * (R * R)
DEF FN.CYLINDER.VOLUME (RADIUS,HEIGHT)
BASE=FN,CIRCLE.AREA (RADIUS)
FN.CYLINDER.VOLUME=BASE * HEIGHT
RETURN
FEND

INPUT "CIRCLE RADIUS?";RAD
THIS.AREA=FN.CIRCLE.AREA (RAD)
```

```
INPUT "CYLINDER RADIUS?" RAD
INPUT "CYLINDER HEIGHT?"; HEIGHT
THAT.VOLUME=FN.CYLINDER.VOLUME (RAD,HEIGHT)
PRINT "THE CIRCLE'S AREA IS:", THIS AREA
PRINT "THE CYLINDER'S VOLUME IS:", THAT.VOLUME

STOP
```

DELETE—*erases file entry from the directory*

DELETE removes the indicated files from their respective directories.

Format:

DELETE <expression> {,<expression>}

Each numeric expression indicates the assigned number of an active file. Real values are converted to integer; string values cannot be used. Any IF END statement associated with the file number being deleted will no longer be valid.

Examples:

DELETE 3,2,1
DELETE FILE.NO%,OUTPUT FILE.NO%

DIM—*allocates storage for an array*

The DIM statement allocates storage for an array and defines the upper limit of each subscript; a lower bound limit of zero is assumed. Execution of each DIM statement allocates a new array. If the current array is numeric, it causes the previous array to be deleted, freeing space for the new one. Each element in a string array must be set to null before reexecution of DIM, to regain the maximum amount of storage.

Format:

DIM exp (subscript list) ,exp (subscript list)

Space is dynamically allocated for numeric and string arrays. Elements of string arrays may be any length up to 255 bytes, and change in length as they assume different values. Numeric arrays are initially set to zero while elements of string arrays are null.

The subscript list specifies the number of dimensions and extent of each dimension of the array being declared. The subscript list may not contain a reference to the array being dimensioned.

Examples:

```
DIM A (10)
DIM NAME$ (50), ADDRESS$ (100),
DIM A% (1,2,3,), SALES% (QUOTA%)
DIM X (A%(B%),C%,D%)
```

FEND—*teminates a multiple line-function definition*

CBASIC should never execute a FEND statement; all multiple line functions must end execution with a RETURN statement, otherwise an error will occur. See the DEF statement for more details.

Format:

FEND

FOR—*initiates a FOR/NEXT loop*

The FOR statement establishes a loop index, an initial index value, a termination index value, and an amount by which the index is increased on each iteration through the loop. A NEXT statement is used to terminate the loop.

Format:

```
FOR index=numeric exp TO numeric exp [STEP exp]
```

Index must be an unsubscripted variable. The expression following the equal sign is evaluated and then assigned to the index, thus establishing the initial index value for the FOR/NEXT loop.

The expression following TO is the loop termination value. A positive step value causes the loop to execute until the index is greater than the termination value. A negative step value executes the loop until the index is less than the termination value. The type of the termination expression must match the type of the index.

The expression following STEP is the loop increment value. When STEP is omitted, a value of 1 is assumed. The increment is added to the index on each execution of the loop prior to comparing the index with the termination value.

Program speed can be maintained by using an index and expressions of type integer, and by omitting the STEP increment expression when a value of 1 is intended.

NOTE

Statements within a FOR/NEXT loop are always executed at least once.

Examples:

FOR 1%=1 TO 1000
FOR J=12.0 TO 123.67 STEP 1.6

Sample Program:

```
PRINT "THIS PROGRAM FINDS THE AVERAGE"
PRINT "OF NUMBERS YOU INPUT"
INPUT "HOW MANY NUMBERS?", LAST %

TOTAL=0.0 REM INITIALLY SET TOTALTO ZERO
FOR INDEX%=1 TO LAST%
PRINT "NUMBER";INDEX%;
INPUT NEXT.NUMBER
TOTAL=TOTAL+NEXT.NUMBER
NEXT INDEX%
AVERAGE=TOTAL/LAST%
PRINT "THE AVERAGE IS";AVERAGE
STOP
```

GOSUB—*causes execution of a subroutine*

The GOSUB statement executes a subroutine. The subroutine is identified via a statement line number. Execution returns to the statement following GOSUB after the subroutine has completed execution.

Format:

GOSUB statement number

The statement number specifies the subroutine entry point.

NOTE

CBASIC can nest subroutines to a depth of 20.

Examples:

GOSUB 200
GOSUB 100.001

Sample Program:

```
REM USE A SUBROUTINE TO MAKE
REM CORRECTIONS

        INPUT "WHAT STRING WILL YOU PRINT?";STRING$
        GOSUB 300       REM MAKE CORRECTIONS
        GOSUB 350       REM PRINT THE STRING

        STOP

300 REM CORRECTIONS SUBROUTINE
        INPUT "WANT TO MAKE CORRECTIONS (Y/N)?";ANS$
        WHILE ANS$="Y"
                INPUT "NEW STRING?";STRING$
                INPUT "IS IT STILL WRONG(Y/N)";ANS$
        WEND
        RETURN

350 REM: PRINT SUBROUTINE
        LPRINTER
        PRINT STRING$
        CONSOLE

        RETURN
```

IF END#—*establishes a conditional file-access branch*

The IF END# statement prepares program logic for conditional execution of a branch during a subsequent file access, providing certain conditions are detected.

Format:

IF END# integer expression THEN statement number

The IF END# statement is unusual in that it prepares program logic for a future conditional branch. Only bookkeeping operations are performed when the IF END# statement is executed. The "integer expression" must be a number between 1 and 20. This is a file number that links the IF END# statement with subsequent file-access statements. The statement number specifies the program line to which the conditional branch will occur if certain conditions are encountered when the subsequent file-access statement is executed. These are the conditions that can cause the IF END# branch:

1. On reading from a file—if the End Of File is detected.
2. On writing to a file—if there is no more disk space.
3. On any file access—if the named file does not exist.

Any number of IF END# statements may appear in a program. The conditional branch specified by the most recent IF END# statement with the same file number gets executed.

IF END# must be the only statement on a line. It cannot be followed by a colon and additional statements, nor can other statements precede it on the same line.

NOTE

When a file is deleted or closed, IF END# statements having the deleted or closed file number are deactivated.

Examples:

```
IF END# 7 THEN 700
REM read a file from #7. On detecting the
REM end of file branch to statement 700
READ #7 ,A,B,C,D
```

Sample Program:

```
REM ADD A NEW RECORD TO NEWFILE.SEQ
REM: OPEN FILE — IF FILE DOES NOT EXIST
REM GO TO 900
          IF END# 10 THEN 900
               OPEN "NEWFILE.SEQ" AS 10

     REM: FIND END OF FILE
     IF END# 10 THEN 200

100 READ # 10; STRING$,NUMBER%
          GOTO 100

200 PRINT # 10;STRING$,NUMBER% +1
          CLOSE 10
          STOP

900 PRINT "ERROR—FILE DOES NOT EXIST"
          STOP
```

IF-THEN-ELSE—*provides a conditional branch*

IF-THEN-ELSE conditionally executes statements depending on the value of an expression. When the conditional expression following IF is true, the group of statements following THEN executes; otherwise the group of statements following ELSE is executed. An expression is "true" if it has a value other than zero.

Formats:

IF integer expression
THEN group of statements
[ELSE group of statements]

The ELSE portion of an IF statement is optional. When ELSE is omitted and the integer expression is false, execution continues with the next sequential statement.

NOTE

A group of statements consists of one or more CBASIC statements separated by colons. The following statements cannot be in this group: DATA, DEF, DIM, IF, and IF END.

The following format is compatible with other BASIC languages:

IF integer expression THEN
statement number

This form of the IF END statement treats the statement number as a GOTO statement. This form does not allow use of the ELSE option.

Sample Program:

```
REM THIS PROGRAM ASKS FOR USER'S NAME
REM AND CHECKS FOR ERRORS

100 INPUT "WHAT IS YOUR NAME?";LINE NAME$

        PRINT NAME$;
        INPUT "-IS CORRECT (Y/N)?";ANSWERS


200 IF ANSWER$ < > "Y" AND ANSWER$ < > "N"\ INVALID ANSWER
INPUT "TYPE " "Y" " FOR YES OR " "N" " FOR NO"; ANSWERS$:\
GOTO 200

IF ANSWER$ = "N" THEN GOTO 100 \ INCORRECT NAME
ELSE PRINT "THANK YOU"\  CORRECT
STOP
```

INITIALIZE—*Reinitializes the operating system*

The INITIALIZE statement must be executed whenever the diskette in a drive is replaced during program execution. Never change a diskette while any files are open. If you do, data may be lost from open files, and the directory will not be updated.

Example:

```
INITIALIZE
```

INPUT—*assigns data to variables*

The INPUT statement receives data from the console input device and assigns it to variables.

Format:

INPUT ["message";] variable {,variable}

The message is an optional string of characters that is printed as a prompt before the computer accepts data from the console; the message must end with a semicolon. In the absence of a prompt, a question mark appears which indicates that input data is expected. The prompt string (or question mark) is followed by a blank character. You must enter data in response to the prompt or question mark.

NOTE

Prompt strings are directed to the console input device even when an LPRINTER statement is in effect.

INPUT statement variables may be simple or subscripted, string or numeric. Data items you enter at the keyboard must be separated by commas. The last data item must be followed by a carriage return. Strings may be enclosed in quotation marks, in which case commas and leading blanks can be part of the string.

There must be a data item for each variable present in the INPUT statement; otherwise CBASIC will request that all data be reentered. The message "IMPROPER INPUT—REENTER" is displayed if too many or too few data items are entered.

You can enter a maximum of 255 characters in response to an INPUT statement. Data will be ignored after this 255-character limit is reached.

If **^Z** is the first character you entered, the program terminates as if a STOP statement had been executed. All CP/M line-editing functions such as **^U** and **^R** are active while data is being input in response to an INPUT statement. (See IF-THEN-ELSE for a sample program.)

Example:

```
INPUT "Enter three values:" ;A%,B%,C%
```

INPUT LINE—*assigns console entry to a string variable*

INPUT LINE is a special form of the INPUT statement that reads an entire keyboard entry and assigns it to a single string variable.

Format:

```
INPUT ["message";] LINE string variable
```

The prompt is handled as described for the INPUT statement. Only one string variable can follow the LINE keyword.

Entered data is assigned to the string variable. Input terminates with a carriage return. Commas and spaces count as characters to be included in the string.

You can enter a null string by responding to an INPUT LINE statement with a carriage return. Up to 255 characters can be entered; additional characters are ignored.

An INPUT LINE statement does not generate an "IMPROPER INPUT" message. (See IF-THEN-ELSE for a sample program.)

Examples:

```
INPUT "Abort (Y or N)?";LINE ANS$
INPUT "Press RETURN to continue"; LINE DUMMY$
```

LET—*assigns a value to a variable*

The LET statement evaluates an expression and assigns the result to a variable. The LET keyword is optional.

Format:

```
[LET] variable = expression
```

The variable may be simple or subscripted. You must specify a string variable for a string expression. Integer or real variables may be specified for numeric expressions. The type of a numeric expression will convert to agree with the type of the numeric variable.

Examples:

```
LET X=3
AMOUNT=COST*QTY%
NAME$(L%)=FIRST(j%)+LAST$(k%)
```

LPRINTER—*directs print from console to printer*

The LPRINTER statement directs PRINT statement output to the printer (see the CONSOLE statement).

Format:

LPRINTER [WIDTH integer expression]

The optional WIDTH parameter sets the printer line width. The initial width is set at 132 characters. A carriage return is automatically output following 132 characters or the specified line width. If you set the WIDTH to zero, an infinite width is assumed and no automatic carriage returns are output. (See the CONSOLE statement for a sample program.)

Examples:

LPRINTER

LPRINTER PRINTER.WIDTH%

LPRINTER.WIDTH 0

NEXT—*terminates a FOR/NEXT loop*

The NEXT statement terminates one or more FOR/NEXT loops. (See the FOR statement.)

Format:

NEXT [index {,index}]

A NEXT statement without any index parameters terminates the most recently encountered FOR statement loop. The index is optional; if present, it must match the index of the FOR statement for the loop being terminated. If the index does not match, an error will occur. You can terminate more than one FOR statement by listing multiple indexes in the NEXT statement. (See the FOR statement for a sample program.)

ON GOTO—*executes a GOTO statement with a choice of destinations*

The **ON GOTO** statement executes a GOTO, selecting a destination statement number that depends on the value of an expression.

Format:

ON integer expression GOTO
statement number list

The statement number list consists of one or more statement line numbers. The integer expression selects one of these statement numbers. If the integer expression is 1, then the first number is selected; if the integer expression is 2, then the second number is selected, and so on.

The integer expression must evaluate to a number ranging between 1 and the number of statement numbers in the statement number list. If the integer expression has any other value, then an error will occur. An error will also be reported if the statement number in the statement number list does not exist and the program logic, therefore, does not know where to branch.

Example:

ON I% GOTO 199, 200, 300

When the statement shown above is executed, if I % is 1, then a branch to line 199 will occur. If I % is 2, a branch to line 200 will occur; and if I % is 3, then a branch to line 300 occurs. In the context of this scenario, line numbers 199, 200, and 300 must exist, and I% must be evaluated as 1, 2, or 3. Here is another example:

On CODE% 3 GOTO 33.1, 33.2, 33.3

Sample Program:

```
REM THIS EITHER CHANGES OR PRINTS THE
REM INPUT DATA STRING UNTIL THE USER
REM TELLS IT TO QUIT

100      INPUT "INPUT DATA STRING:";
         STRING$

125      PRINT "YOUR OPTIONS ARE:"
         PRINT "   1........CHANGE THE STRING"
         PRINT "   2........PRINT THE STRING"
         PRINT "   3........QUIT"
         INPUT "WHICH DO YOU WANT?";NUMBER%
         IF NUMBER%0 AND NUMBER% <4 THEN\
                 ON NUMBER\ GOTO 100, 150, 175\
                 ELSE GOTO 125

150      LPRINTER: PRINT STRING$:CONSOLE

175      STOP
```

ON GOSUB—*executes a GOSUB statement with a choice of subroutines*

The ON GOSUB statement is similar to ON GOTO, described above, except that a subroutine is called at the specified statement.

Format:

ON integer expression
GOSUB statement number list

The statement number list consists of one or more line numbers that are subroutine entry points. The integer expression determines which subroutine gets called. If the integer expression is 1, the first subroutine is called. If the integer expression is 2, the second subroutine is called, and so on. The integer expression must have a value that ranges between 1 and the number of subroutine entry points in the statement number list.

Example:

ON I% GOSUB 199, 200, 300

Sample Program:

```
REM THIS PROGRAM EITHER CHANGES OR
REM PRINTS THE INPUT DATA STRING UNTIL
REM THE USER TELLS IT TO QUIT

  INPUT "INPUT DATA STRING:";STRING$
  MORE% = 1
  WHILE MORE% = 1

    PRINT "YOUR OPTIONS ARE:"
    PRINT " 1.......CHANGE THE STRING"
    PRINT " 2.......PRINT THE STRING"
    PRINT " 3.......QUIT"

    INPUT "WHICH DO YOU WANT?";NUMBER%

    IF NUMBER%>0 AND NUMBER%<4 THEN\
      ON NUMBER%GOSUB 210, 220, 230

  WEND
  STOP

210          INPUT "WHAT STRING WOULD YOU LIKE/";STRING$
             RETURN

220          LPRINTER:  PRINT STRING$:  CONSOLE
             RETURN

230          MORE% =0          REM NO MORE


STOP
```

This is almost identical to the ON GOTO example shown previously. In the ON GOSUB case, 199, 200, and 300 are line numbers within subroutines. After the selected subroutine executes, program logic will return to the statement directly following the ON GOSUB.

OPEN—*activates an existing file*

The OPEN statement is used to activate an existing file. The OPEN statement is similar to the CREATE statement, except that an existing file is activated rather than a new one.

Format:

OPEN string expression
[RECL integer expression]
AS integer expression [BUFF integer exp]
[RECS integer expression]

The string expression following the keyword OPEN is the name of the file to be activated. The integer expression following RECL specifies record length. When you specify the length, the file will contain fixed-length records that you can access randomly or sequentially. If no length is specified, the record length will vary.

NOTE

An IF END statement associated with the assigned file number will not be affected.

The BUFF and RECS portions of an OPEN statement must either be used in conjunction, or be omitted. The integer expression following BUFF indicates how many disk sectors are to be maintained in memory. When using random access you must specify one disk sector. A value of 1 is automatically assumed without the BUFF and RECS option. RECS identifies the size of a physical sector on the disk. This value is currently ignored. The system currently assumes a sector size of 128 bytes.

Examples:

```
OPEN "PAYROLL.DAT" AS 9
OPEN FILE.NAME$ AS FILE.NO%
OPEN WORK.FILE$(1%)\
RECL LENGTH%\
AS FILE.NO%\
BUFF BS% RECS 128
```

Sample Program:

```
REM OPEN NEWFILE.SEQ AND NEWFILE.RAN.
REM READ AND PRINT THE FIRST FILE OF
REM NEWFILE.SEQ AND THE FIFTH FILE OF
REM NEWFILE.RAN

    REM:  OPEN NEWFILE.SEQ—IF IT DOES
          NOT EXIST GO TO 900
          IF END #3 THEN 900
          OPEN "NEWFILE.SEQ" AS 3

    REM:  OPEN NEWFILE.RAN—IF IT DOES
          NOT EXIST GO TO 900
          IF END #4 THEN 900
          OPEN "NEWFILE.RAN" RECL 25 AS 4
          READ #3: STRING$, NUMBER%
      PRINT "FIRST RECORD OF
      NEWFILE.SEQ:",STRING$;NUMBER%
```

```
                READ #4,5;STRING$,NUMBER%
                PRINT "FIFTH RECORD OF
                NEWFILE.RAN:", STRING$ NUMBER%

                CLOSE 3, 4
                STOP
        900     PRINT "ERROR DOES NOT EXIST"
                STOP
```

OUT—*outputs an integer to an I/O device*

The OUT statement is used to output an integer to a selected input/output port.

Format:

OUT integer expression, integer expression

The low-order byte of the second expression is sent to the output port addressed by the low-order byte of the first expression.

Examples:

OUT 3, 80H

OUT TAPE.DATA%,NEXTCHAR%

POKE—*stores a byte in a selected location*

The POKE statement stores a byte of data in a memory location identified by its absolute memory address.

Format:

POKE integer expression, integer expression

The low-order byte of the second expression is stored at the memory location addressed by the first expression.

Examples:

POKE 100H, 225
POKE MSG.ID%, END.MARK%

Sample Program:

```
REM POKE OR PEEK AT LOCATIONS CHOSEN
REM BY USER

    MORE% = 1
    WHILE MORE%
        INPUT "TYPE POKE, PEEK, OR QUIT"; STRING$
        IF STRING$ = "PEEK" THEN\
              INPUT "LOCATION?"; LOC%:\
              VAL% = PEEK (LOC%):\
              PRINT VAL%
        IF STRING = "POKE" THEN\
              INPUT "LOCATION?";LOC%:\
              INPUT "VALUE?";VAL%:\
              POKE LOC%, VAL%
        IF STRING$ = "QUIT" THEN\
              MORE% = 0

    WEND
    STOP
```

PRINT—*outputs data to the console or printer*

The PRINT statement outputs data to the console or printer. (See the LPRINTER and CONSOLE statements.)

Format:

PRINT expression {delim expression} [delim]

Expressions in the PRINT statements parameter list are printed. Numbers are printed in 15-character columns and are left-justified. Scientific notation is used if a number will not fit within the 15-character column width.

Strings are printed without modifications. If the end of a line is reached part way through a numeric field, then the entire number moves to the next line. A string, on the other hand, may be broken by an automatic carriage return. The string is output until the end of the line is reached, and the remainder of the string is output on the next line.

The delimiter (delim) between expressions may be a comma or a semicolon. A comma causes automatic spacing to the next column divisible by 20. A semicolon causes one blank to be output following a numeric value and eliminates spacing following a string.

The delimeter at the end of a PRINT statement is optional; if present, the last delimeter has the same effect as that described earlier. No carriage return is output if the PRINT statement ends in a delimeter. A delimeter at the end of a statement allows the next PRINT statement to continue where the previous one left off. If you use no delimeter, a carriage return and line feed are output at the end of the print operation.

NOTE

A PRINT statement with no parameter list outputs a carriage return and a line feed.

Examples:

PRINT VALUE, AMOUNT
PRINT "the amount owed is" ; COST
PRINT A$; B$, C$,
PRINT

Sample Program:

```
REM PRINT X AND Y USING BOTH THE , AND ;
REM DELIMETERS
        X = 123.5463
        Y = 98777777655232536 47.5890
        PRINT "X =",X
        PRINT "Y =";Y; "X =";X
        STOP
```

PRINT# —*outputs data to a disk file*

The PRINT# statement functions like the PRINT statement described earlier, except that output is directed to a disk file.

Format:

PRINT# integer expression [,integer expression] ; expression {,expression}

The PRINT# statement consists of two parts: the file and record-selector portion, which is terminated with a semicolon, and a list of expressions. The expressions are separated by commas. No comma should appear at the end of the list, however.

The first expression is a number identifying the file to which data will be output; this file number must be active and can range from 1 to 20.

The second integer expression is optional; if present, it identifies the record number of a random-access file where data will be stored. When the second integer expression is present, the file must have been opened with the RECL parameter specifying record length.

Numbers are output to a diskette without any change in format. Strings are enclosed in quotation marks before being output. A string that is output to a diskette file may not contain quotation marks. Blanks are used to pad fixed-length records of a random-access file so that the line feed is the last character in the record. (See the CLOSE statement for a sample program.)

Examples:

```
PRINT# 1; A,B,C
PRINT#DATA%,REC.NO%
;NAME$,STREET$.CITY$,STATE$,ZIP
```

PRINT USING—*outputs formatted data*

The PRINT USING statement provides formatted output to the console or printer. You specify the format using a format expression.

Format:

PRINT USING string expression ;
expression list

The string expression consists of data fields that describe the format used to output the expression list. The string expression may also contain literal characters that are output as they occur.

The expression list is as described for the PRINT statement. String field specifications may be used in a PRINT USING statement. These PRINT USING field specifiers are described in the "Print Using Table" and below:

Literal Characters:

Characters that are not part of a string or numeric-field format are assumed to be literal characters. Blank spaces, for example, are frequently used to separate fields.

Examples:

"#### ####"	3 blanks separate two numeric fields.
"NO.*##########*"	The 3 characters NO. precede a numeric field.

\ treats the next character in the format expression as a literal character. A single literal character is defined using a \ character followed by the single literal character. A \ is generally used to print a control character as a literal.

Examples:

"\#"	Prints #
"\$\#\!"	Prints ##!
"\\\\"	Prints \\

Numeric Fields:

Numeric digit

An integer numeric field is specified by two or more # characters, one per character of the field width. Numbers are right-shifted within the numeric field, with blanks preceding the most significant digit.

Examples:

"###" specifies a 3-digit numeric field.

"######" specifies a 6-digit integer numeric field.

You specify a real numeric with a decimal point by placing a character within the # character string at the location where the decimal is to appear.

Example:

"###.##" specifies a 5-digit numeric field; 3 digits precede the decimal point and 2 follow.

You can format numeric fields with commas by placing one or more characters anywhere between the first and last # characters. The position of the , character in the numeric field specification is irrelevant; the commas are positioned after every third character.

Example:

"##,#####.##" specifies a number with 8 predecimal digit positions and 2 postdecimal digits. A , character prints every third digit to the left of the decimal point.

^ Exponential numeric format:

You specify exponential numeric format by appending one or more ^ characters at the end of the numeric field definition. The decimal-point position affects the exponent value. Four character positions are always added for the exponent.

Example:

"#,####^" specifies a number printed in exponential format with 1 predecimal digit and 4 postdecimal digits.

**** Fills a numeric field with leading asterisks:**

Two asterisks appearing at the beginning of a numeric field fill unused leading characters with asterisks.

Example:

"**######.##" specifies a numeric field with 8 predecimal digits and two postdecimal digits. * characters appear in any leading unused position.

$ Monetary format with dollar sign:

A $ sign will appear in front of the first non-blank digit if the numeric field definition begins with two $ characters. The $ character will not be printed for negative numbers and cannot be used with * fill characters.

Example:

"$$######.##" specifies a numeric field with 8 predecimal and 2 postdecimal digits. A $ sign precedes the first nonblank digit.

– Marks position for a number's sign (leading or trailing):

A leading negative sign is defined by a – character in the first-character position of the numeric field definition. A negative sign at the end of a numeric field is specified if the – character appears in the last-character position of the numeric field definition.

Examples:

"–######.##" specifies a numeric field with 6 predecimal digits preceded by a negative sign for negative numbers, and two postdecimal digits.

"######.##–" specifies a numeric field with 6 predecimal digits and two postdecimal digits followed by a negative sign for negative numbers.

String Variables:

& defines a variable-length string field.

A variable-length string field is specified by a single & character.

Examples:

"**&**" Specifies a string field of any length.

"**&&**" Specifies two variable-length string fields separated by a single blank character.

To define a fixed-length string field, place a / character in the first and last field positions. Any characters may appear between the first and last / characters.

Example:

"/...../" specifies a 7-character string field.

! specifies a single string character.

A! specifies a single-character string field.

Example:

"!" prints the first character of a string field.

Sample Program:

```
REM USE PRINT USING TO PRINTA PURCHASE
REM RECORD
    COMPANY$ = "SHANNON'S SUPPLIES"
    FIRST.NAMES$ = "LIST"
    MI$ = "G"
    LAST.NAMES$ = "PATRICK"

PARTNO1% = 1306 : PRICE1 = 1304.57 : QUANT1% =4
PARTNO2% = 1296 : PRICE2 = 23.99 : QUANT2 = 6
CHARGE1 = PRICE 1 * QUANT1%
CHARGE2 = PRICE2 * QUANT2%
TOTAL = CHARGE1 + CHARGE2

FOR$= "RECORD OF PURCHASE BY /FFFFF/!. /LLLLLLLLL/"
FROM$ = "FROM &"
ITEMS$ = "PART /##### : ## AT ##,###.## AMOUNT:/ $$##,###.##"

PRINT USING FOR$; FIRST.NAMES$,MI$,LAST.NAMES$
PRINT USING FROM$; COMPANY$
PRINT
PRINT USING ITEM$:PARTNO1%, QUANT1%,PRICE1,CHARGE1
PRINT USING ITEM$;PARTNO2%, QUANT2%,PRICE2,CHARGE2
PRINT
PRINT USING "!.!.!. OWES A TOTAL OF **###,###.##";\
        FIRST.NAME$,MI$,LAST.NAME$,TOTAL

STOP
```

String-Field Specifications in "PRINT USING" Statements

	FIELD	FORMAT	EXAMPLE	COMPONENT
LITERAL CHARACTER	A single literal character	\x where x can be any character.	"\#" prints # "\#\#\!" prints ##! "\\\\" prints \\	\ is generally used to print a control character as a literal.
	Literal text and field separators	Any character not part of a string or numeric field format.	"#### ####. Three blank spaces separate two numeric fields "NO:###". The three characters NO: precede a numeric field.	Most frequently, literal blank spaces are used to separate fields.
NUMERIC FIELDS	Simple integer numeric	Two or more # characters, one # character per field width.	"###" specifies a 3-digit integer numeric field. "######" specifies a 6-digit integer numeric field.	Numbers are right shifted within the numeric field with blanks preceding the most significant digit.
	Real numeric with decimal point	Place . character within # string at desired decimal point location.	"###.##" specifies a 5-digit numeric field. 3 digits precede the decimal point, 2 follow.	Numbers are rounded.
	Numeric fields with characters	Place one or more , characters anywhere between the first and last # characters.	"##,,#####,##" specifies a number with 7 predecimal and 2 postdecimal digits. A , occurs before every third digit to the left of the decimal point.	The position of , characters in the field specification is not relevant.
	Numeric field with exponential notation	Append one or more ↑ characters at the end of the field definition.	"#.####↑" specifies a number printed in exponential format with one predecimal digit and four postdecimal digits.	The decimal point position effects the exponent value. Four character positions are always added for the exponent.
	Numeric field with * in leading numeric character position	Add ** to the front of the field specification.	"**######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, plus * characters in leading unused character positions.	This is used in financial printouts.
NUMERIC FIELDS	Numeric field with $ character	Add $$ to the front of the field specification.	"$$######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, and a $ character preceding the first nonblank digit.	$ is not printed for negative numbers. $$ cannot be used with **.
	Numeric field with - sign in first character position Numeric field with - sign in last character position	Begin field specification with - character. End field specification with - character.	"-######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, and a - sign in the first character position for negative numbers. "#####.##-" specifies a - sign in the last character position for the same field.	Normally the - sign precedes the first non-blank character. The leading - cannot be used with ** or $$ options.
	Variable-length string field	A single & character.	"&" prints a string field of any length.	
	Fixed-length string field	character in first and last character position.	" ..... " specifies a 7-character string field.	Any characters may appear between the first and last .
	Single-string field character	A single ! character.	"!" prints the first character of a string field.	

From *CBASIC™ User Guide* by Adam Osborne, Gordon Eubanks, and Martin McNiff. Copyright 1981 McGraw-Hill Inc. Used with permission of OSBORNE McGraw-Hill

PRINT USING#—*outputs formatted data to disk file*

The PRINT USING# statement outputs formatted data to disk files. See the PRINT# and PRINT USING statements for details on formatted printing and printing to disk files. (See table on previous page.)

RANDOMIZE—*seeds a random-number generator*

The RANDOMIZE statement is used to seed the random-number generator.

Format:

RANDOMIZE

The time an operator takes to respond to an INPUT statement is used to seed the random-number generator. Thus, an INPUT statement must be executed before the RANDOMIZE statement.

Example:

RANDOMIZE

Sample Program:

```
REM GUESSING GAME

    INPUT "GUESS A NUMBER BETWEEN 0
      AND 100"; GUESS%

    RANDOMIZE
    NUMBER% = 100 * RND

    COUNT% = 1
    WHILE GUESS% <> NUMBER%
```

```
          IF GUESS%>NUMBER% THEN PRINT "TOO LARGE"\
          ELSE PRINT "TOO SMALL"

          INPUT "NEXT GUESS?"; GUESS%
          COUNT% = COUNT% + 1
      WEND

      PRINT "YOU GUESSED IT IN";COUNT%; "TRIES"
      INPUT "DO YOU WANT TO TRY AGAIN (Y/N)?"; AGAIN$
      IF AGAIN$ = "Y" THEN GOTO 100
      STOP
```

READ—*assigns values from DATA statements to variables*

The READ statement assigns values taken from DATA statements to variables in the READ statement parameter list. (See the DATA and RESTORE statements.)

Format:

READ variable {,variable}

The next unused item from the list of constants compiled by concatenating all data-statement parameters is assigned to the next variable in the READ statement parameter list. (See the DATA statement for a sample program.)

Examples:

READ COST1,COST2,COST3
READ TABLE (I%)

READ#—*reads data from disk files*

The READ# statement is used to read data from disk files.

Format:

READ# integer exp [,integer exp];
variable {,variable}

The first expression is the file number for which data will be read; this file number must be active and range from 1 to 20. The second integer expression is optional and identifies the record number of a random-access file from which data will be read. If the second integer expression is present, the file must have been opened with the RECL parameter specifying record length.

When an attempt is made to read past the end of a file, execution continues with the statement number specified by the most recently executed IF END# statement having the same file number. If no IF END statement was executed, an error occurs. (See the OPEN statement for a sample program.)

READ# LINE—*reads a disk file string and assigns it to a variable*

The READ# LINE statement reads a record from a selected diskette file and assigns it to a string variable. The READ# LINE statement acts like the READ# statement.

Format:

READ# integer exp [,integer exp];
LINE string variable

The first integer expression selects the file. The second integer expression, if present, selects the record for a random-access file. These expressions are used as described for the READ# statement.

The string read from the selected record is assigned to the string variable. If the length of the record is greater than 255 bytes, the record is shortened to 255 characters and a warning is printed on the console.

REM—*indicates a remark*

The REM statement is used to document programs. This statement is ignored by the CBASIC compiler.

Format:

REM any character

CBASIC object size is not affected by REM statements, since these statements are ignored by the compiler. REM statements can be split across lines using a \ character. If a REM statement shares a line with other statements, then REM must be the last statement on the line.

RESTORE—*resets the data-list pointer*

The RESTORE statement is used to reset the pointer into the DATA statement's list of constants so that the next value read by a READ statement will be the first item in the first DATA statement.

Format:

RESTORE

Sample Program:

```
REM READ AND PRINT THE DATA LIST
DATA 123.987,42 "MORE DATA", "EVEN MORE DATA"

READ REAL.NUM, INT.NUM%, STRING1$, STRING2$
PRINT REAL.NUM; STRING1$, INT.NUM%; STRING2$

RESTORE
READ REAL.AGAIN, INT.AGAIN%
PRINT REAL.AGAIN, INT.AGAIN%
STOP
```

RETURN—*returns program execution from a subroutine to the calling program*

The RETURN statement causes execution of the program to continue with the statement following the most recently executed GOSUB. The RETURN statement also causes an exit from a user-defined function.

Format:

RETURN

Each GOSUB statement saves the location of the statement following it. Program execution returns to the saved location after the subroutine has completed execution. (See the GOSUB, ON GOSUB, and DEF statements.)

SAVEMEM—*reserves space for a file*

The SAVEMEM statement loads an assembly-language program from diskette into memory.

Format:

SAVEMEM integer constant, string expression

The integer constant specifies the number of bytes of memory to reserve for the file being loaded. Space is reserved at the top of available memory. You can calculate the beginning address of the reserved area by subtracting the integer constant from the largest available address.

The string expression identifies the file to be read. Records are read from the file until an end of file is encountered or until the next record to be read would exceed the specified memory size. If the string expression is null or if the integer constant is less than 128, space is reserved, but no file is loaded.

Only one SAVEMEM statement may appear in a program; if the first program executed contains a SAVEMEM statement, any chained program associated with it must also include a SAVEMEM statement. The associated chained programs must have the same integer constant but a different file may be loaded by each chained program.

Examples:

```
SAVEMEM 1028, "IOPACK.COM"
SAVEMEM 128,""
```

STOP—*stops program execution*

The STOP statement causes program execution to stop. Control is returned to the operating system (CP/M).

Format:

```
STOP
```

NOTE

Any OPEN files will be closed.

WEND—*terminates a WHILE loop*

The WEND statement terminates a WHILE statement loop (see the WHILE statement).

Format:

WEND

WHILE—*initiates the WHILE-WEND loop*

The WHILE statement controls looping between the WHILE statement and its corresponding WEND statement.

Format:

WHILE integer expression

Looping continues until the integer expression is a 0 (false). The loop may contain any number of statements, including additional WHILE statements. If the expression is false initially, then no statements in the loop execute. (See RANDOMIZE for a sample program.)

Examples:

```
WHILE AMOUNT < = MAX
WHILE -1 REM LOOP FOREVER
```

CBASIC Functions

There are three kinds of functions: numeric, string, and disk. The function name further identifies it as an integer, a real, or a string function. If the function requires an expression, then the normal CBASIC expression rules apply. If an integer expression is required, real values are automatically converted to integers. Likewise, integer expressions are converted to real expressions where necessary. Numeric expressions in string functions generate a run-time error; so do string expressions in numeric functions. Avoid conversions by using the proper form for the expression since this improves program execution speed.

ABS—*returns absolute value*

The ABS function returns the absolute value of the argument.

Format:

ABS (numeric expression)

Examples:

```
X=ABS(Y)
DIFF=ABS(COST-PROFIT)
```

ASC—*returns an ASCII value*

The ASC function returns the ASCII integer value for the first character of the argument. Only the first character is considered. If the expression is evaluated as a null string or if the argument is numeric, an error will occur.

Format:

ASC (string expression)

Examples:

```
I% = ASC(STRING$)
FIRST% = ASC(FIRST.NAME$)
```

ATN—*returns the arctangent of the argument*

The ATN function is used to return the arctangent of an argument. The argument must be expressed in radians. The value returned is a real number. The arctangent returned can be used to compute various inverse trigonometric functions.

Format:

ATN (numeric expression)

Examples:

```
ANGLE = ATN(X)
ASIN = ATN(X/SQR(1.0 - X*X))
```

CHR$—*returns the ASCII string equivalent of the argument*

The CHR$ function converts the argument to its one-character ASCII string equivalent. The CHR$ function can be used to send control characters to an output device. If the argument is greater than 255, the high-order byte is ignored.

Format:

CHR$ (numeric expression)

Examples:

```
BELL$ = CHR$(7)
STOP.CHAR$ = CHR$(STOP%)
```

COMMAND$—*returns the command line*

The COMMAND$ function returns the command line used to execute the current program. The name of the program being executed will not be included in the string that is returned. If the TRACE option is used with CRUN, the word TRACE and the associated line numbers will not be included.

Format:

```
COMMAND$
```

The COMMAND$ function allows options to be passed to the CBASIC program when it is executed. The COMMAND$ function may be used anywhere and anytime within a program, including programs loaded by a CHAIN statement.

CONCHAR%—*returns the binary equivalent of a character input at the console device*

The CONCHAR% function reads one character from the console input device and returns an integer equal to the binary representation of that character.

Format:

```
CONCHAR%
```

The character that is read back is echoed to the console display device by the CONCHAR% function. If no character has been entered at the console, a zero is returned. (See the CONSTAT% function.)

Examples:

```
ANS% = CONCHAR%
IF CONCHAR% = ESC% THEN DONE%
= TRUE%
```

CONSTAT%—*returns console status*

The CONSTAT% function returns an integer expression to indicate console status. A logical true (−1) signifies that the console is ready. If a logical false (0) is returned, the console device does not have a character ready.

Format:

```
CONSTAT%
```

Examples:

```
IF CONSTAT% THEN ANS% = CONCHAR%
X% = CONSTAT%
```

COS—*returns the cosine of the argument*

The COS function returns the cosine of the argument. The argument must be expressed in radians, and the value returned is a real number.

Format:

```
COS (numeric expression)
```

Example:

```
X = COS(ANGLE)
```

EXP—*returns the exponent of the argument*

The EXP function returns the value of the constant raised to the power of the argument. The value returned is a real number.

Format:

EXP (numeric expression)

Examples:

```
POWER=EXP(X*X-Y*Y)
E=EXP(1)
```

FLOAT—*converts the argument to a real number*

The FLOAT function converts the argument to a real number. If the argument is already a real number, FLOATconverts it to an integer and then back to a real number.

Format:

FLOAT (numeric expression)

Example:

```
X=SIN(FLOAT(I%))
```

FRE—*returns the amount of available data memory*

The FRE function returns the number of available memory bytes in the dynamic or free storage area. The value FRE returns is a real number. Free storage may consist of two or more noncontiguous memory blocks.

Format:

FRE

Example:

SPACE.REMAINING=FRE

INP — *returns a byte from an I/O port*

The INP function returns a byte from a selected input/output port. An integer that is the value read from the port addressed by the argument is returned.

Format:

INP (integer expression)

Examples:

DEV.1%=INPUT
TAPE.STATUS%=INP (TAPE.SP%)

INT — *returns the integer portion of the argument*

The INT function truncates the fractional portion of the argument and returns the integer portion. The value returned by INT is a floating-point number; if the argument is an integer, it is converted to a real value.

Format:

INT(numeric expression)

Examples:

```
DOLLARS=INT(TOTAL.DUE)
CENTS=TOTAL-INT(TOTAL.DUE)
```

INT%—*converts the argument to an integer*

The INT% function converts the argument to an integer. The difference between INT and INT% is that the result of INT is a real number, while the result of INT% is an integer. If the argument is an integer, it is converted to a real number and then back to an integer.

Format:

INT% (numeric expression)

Examples:

```
K%=INT%(SIZE)
LENGTH=DIMENSION (INT%(X))
```

LEFT$—*returns leftmost characters of the argument*

The LEFT$ function returns characters from the left of a string argument.

Format:

LEFT$ (string expression, numeric expression)

The numeric expression specifies the number of characters returned. If the number of characters that is to be returned is greater than the length of the string, the entire string is returned. If the numeric expression is zero, a null string is returned. If the expression is negative, an error will occur.

Examples:

```
RESPONSE=LEFT$(ANS$,1)
PRINT LEFT$
(NAMES$,MAX.NAME.LENGTH%)
```

LEN—*returns the length of an argument*

The LEN function returns the length of a string argument. The value returned by LEN is an integer, a zero is returned if the value is a null string. An error occurs if the argument is numeric.

Format:

LEN (string expression)

Examples:

```
LENGTH.NAME%=LEN(NAME$)
IF LEN(TEMP$) >= MAX.L% THEN GOTO 100.00
```

LOG—*returns the natural logarithm of the argument*

The LOG function returns the natural or Naperian logarithm of the argument. The LOG function can be used to calculate logarithms to other bases. The value returned is real. Integer arguments are converted to real numbers before the logarithm is computed. An error occurs if the argument is negative or zero.

Format:

LOG (numeric expression)

Examples:

```
BASE.TEN.L=LOG(X)/LOG(10)
Z=LOG(W)
```

MATCH—*searches a string for a match*

The MATCH function returns the position of the first occurrence of a pattern string within a source string, beginning at a specified position in the source string.

Format:

MATCH (string exp, string exp, numeric exp)

The first string expression argument is the pattern, the second expression argument is the source string. The integer argument specifies the character position in the source string where the search is to begin. The search progresses from the starting position to the end of the source string, attempting to match the pattern specified in the first expression. If a match occurs, the position of the first matching character in the source string is returned. If no match occurs, a zero is returned.

The following special pattern-matching characters are provided:

A pound sign matches any numeric digit (0–9).

! An exclamation mark matches any uppercase or lowercase letter (A–Z and a–z).

? A question mark matches any character.

\ A backslash indicates that the character following the backslash does not have a special meaning. Thus, a backslash before a #,!,?, or another backslash character will override the definition above and result in the character being treated as a normal pattern character.

If either string expression is null, a zero is returned. If the beginning point for the match is greater than the length of the source string, a zero is returned. If the numeric expression is zero or negative, an error will occur.

Examples:

MATCH ("CDE", "ABCDEFGHI", 1) returns 3

MATCH ("CDE", "ABCDEFGHI", 3) returns 3

MATCH ("CDE", "ABCDEFGHI", 4) returns 0

MATCH ("D?F", "ABCDEFGHI", 1) returns 1

MATCH ("\ #1 \ \ \ ?", "1#1\ ?2#", 1) returns 2

MID$—*returns a portion of a string*

The MID$ function returns a string that may be any portion of another string. MID$ can accomplish the same function as either RIGHT$ or LEFT$, but, in addition, MID$ can return a string from the middle of another string.

Format:

MID$ (string exp, numeric exp, numeric exp)

A portion of the first string expression is returned. The second expression specifies the starting position of the string to be returned. The third expression specifies the length of the string to be returned. If the third expression specifies a character position beyond the end of the string, then characters from the position specified by the second argument, up to the end of the string, are returned. A null string is returned if the starting position specified by the second expression is greater than the length of the string, or if the third expression is zero.

NOTE

An error occurs if either numeric expression is negative, or if the second one is zero.

Examples:

MIDDLE$=MID$(NAME$,START.MN%, LENGTH.MN%)
DAY$=MID$("MOTUWETHFRSASU", DAY% 3-2,3)

PEEK—*returns the contents of a selected memory location*

The PEEK function returns the contents of the memory location addressed by the function argument.

Format:

PEEK (numeric expression)

The numeric expression is a memory address. The PEEK function returns an integer value equal to the contents of the addressed memory location. The memory location must be within the address space of the computer, or the results will be meaningless. For memory locations greater than 32767, the address must be negative; in this case the address could be expressed in hexadecimal notation for clarity.

Examples:

BDOS%=PEEK(6) + PEEK(7) 256
ENTRY%=PEEK(OEOOH)
PARM1%=PEEK(LOC.P1%)

POS—*returns the column position of the next print character*

The POS function returns the character number for the next character to be displayed or printed on the current line.

Format:

POS

POS returns the number of characters to the output device. Cursor control characters are also counted even when the cursor has not been advanced. Thus, if the cursor has been positioned by special characters or nonprinting control characters have been output, the POS function does not return the actual cursor position.

Examples:

```
PRINT TAB (POS+3);"#"
LOC.CURSOR%=POS
```

RENAME—*changes the name of a disk file*

The RENAME function changes the name of a disk file.

Format:

RENAME (string expression,
string expression)

The first string expression is the new file name. The second string expression is the current file name. The RENAME function returns an integer value of 0 (false) if the RENAME cannot be accomplished and a −1 (true) if the RENAME is successful.

NOTE

The file being renamed must not be open or an error will occur when the file is closed.

Examples:

```
IF RENAME ("MASTER.CUR", "MASTER.TMP")=O THEN GOTO 500
X%=RENAME(NEW.NAME$, OLD.NAME$)
```

RIGHT$—*returns the rightmost character of the argument*

The RIGHT$ function returns the rightmost characters of a string.

Format:

**RIGHT$ (string expression,
numeric expression)**

The numeric expression specifies the number of rightmost characters of the string to be returned. If the number of characters to be returned is greater than the length of the string expression, then the entire first argument is returned. If the numeric expression is 0, a null string is returned. A negative numeric expression causes an error to occur.

Examples:

```
CHECK.DIGIT$=RIGHT$(ACCOUNT.NOS$,1)
LAST.NAME=RIGHT$(NAME$,LEN(NAME$)-LEN(FIRST.NAME$)
```

RND—*returns a random number*

The RND function returns a random number between 0 and 1.

Format:

RND

The RND function generates the next random number in a sequence that is based on the current seed. The value returned is a real number. The RANDOMIZE statement must be executed to generate a seed and a random-number sequence.

Example:

PRINT RND

SADD—*returns the starting memory address of a string*

The SADD function returns the starting memory address of the string currently assigned to the string variable. The first byte of the string is the length of the string. Subsequent bytes hold characters of the string.

Format:

SADD (string variable)

The string variable cannot be a string expression. The value returned by SADD is an integer. A zero is returned for a null string unless the null strings have previously been assigned a test value.

Examples:

```
LOC.PARM%=SADD(NAME$)
PTR%+SADD(A$)
```

SGN—*returns the sign of the argument*

The SGN function returns −1, 0, or 1, depending on whether the argument is negative, zero, or positive respectively.

Format:

SGN (numeric expression)

NOTE

The value returned is an integer.

Examples:

```
IF SGN(TOTAL)=-1 THEN GOTO 200.20
ON SGN(X) + 2 GOTO 10, 20, 30
```

SIN—*returns the sine of the argument*

The SIN function returns the sine of the argument.

Format:

SIN (numeric expression)

The argument must be expressed in radians. The value returned is a real number.

Example:

X=SIN(ANGLE)

SIZE—*returns the amount of reserved space*

The SIZE function returns the amount of space reserved by a file or a group of files.

Format:

SIZE (string expression)

The string expression may be an ambiguous file name or the name of one file within the directory. Ambiguous file names allow the programmer to determine the space being used by all of the files whose names have common characters. If no files within the directory match the expression, a 0 is returned. The size function returns an integer; the value returned is the number of blocks that the file is using. Each block is 128 bytes.

Examples:

```
AMOUNT$+SIZE(WORKFILE)
I%=SIZE("*.BAK")
FREE.SPACE%=DISK.CAPACITY%- SIZE("*.*")
```

SQR—*returns the square root of the argument*

The SQR function returns the square root of the argument.

Format:

SQR (real expression)

The SQR function returns a real number. If the argument is negative, a warning appears on the console, and the absolute value of the argument is used in calculating the square root.

Examples:

```
HYP=SQR(X*X + Y*Y)
PRINT SQR(X)
```

STR$—*returns a character string*

The STR$ function returns a text character that is the ASCII equivalent of the argument.

Format:

STR$ (real expression)

The STR$ function converts a real number into its string equivalent. For example, the real number 1.234 would convert to the string "1.234".

Examples:

```
A$=STR$(X)
PRINT STR$ (ZIP)
```

TAB—*positions the cursor*

The TAB function is used to position the cursor at the character position the argument specifies.

Format:

TAB (integer expression)

The TAB function can be used only in PRINT statements. The cursor or printer, depending on whether LPRINTER is in effect, moves to the right of the selected column. The TAB does not move backwards. If the expression is less than the current cursor position, a new line starts and the TAB then occurs.

If the argument exceeds the line width of the device receiving output, an error occurs. A semicolon should be used following a TAB; a comma could cause additional spacing.

Examples:

```
PRINT TAB(START.COL%); NAME$
PRINT X;TAB(30);Y
```

NOTE

The TAB function counts characters output since the last carriage return and line feed. This could differ from the current cursor line position. Differences result if the program has output nonprinting control characters or characters that move the cursor.

UCASE$—*converts a string to uppercase characters*

The UCASE$ function is used to convert a string to its uppercase equivalent.

Format:

```
UCASE$ (string expression)
```

The value UCASE$ returns is a string. All lowercase alphabetic characters in the argument are converted to uppercase characters. Other characters remain unaltered.

Examples:

```
U.CITY$=UCASE$(CITY$)
X$=UCASE$(X$)
```

VAL—*converts a string to a real number*

The VAL function converts a string to a real number.

Format:

VAL (string expression)

The VAL function converts a string to its real numeric equivalent. The string must be a valid text representation of a real number. For example, the string "1.234" is converted into the real number 1.234. The first character of the string must be a number or a plus or minus sign. An error is generated if the string includes any character that is not part of a valid real number. Zero is returned for a null string.

Examples:

```
X=VAL(A$)
PI=VAL("3.1416")
```

Compiler Directives

Compiler directives control the compiler (CBAS2.COM). Except for the END statement, all directives are preceded by a percent sign in the first column. The compiler ignores any nondirective characters on the same line.

%LIST Begins listing a source file when it is encountered during compilation.

%NOLIST Terminates listing of the source file initiated by %LIST.

%PAGE <constant> Sets page length output to the printer. The default page length is 64.

%EJECT Positions listing sent to the printer and disk at the top of the following page.

%INCLUDE <file name> Causes named file to be compiled into the source file. %INCLUDE may be preceded by a drive identifier if needed. The file referenced by %INCLUDE must be of type .BAS. Each statement number in the referenced file is indicated by an equal sign after its assigned statement number. %INCLUDE(s) may be nested up to six levels.

%CHAIN <constant>, <constant>, <constant> Sets the size of the main programs—constant, code, data, and variable area—to prevent overwrites. The first constant is the area used for real constants, the second is the size of the code area, the third is used to store value from DATA statements, and the fourth is the size of the area used to store variables.

[statement number >] END The END statement is optionally used to terminate reading of the source file. This statement must be the first on a line to cause all statements following it to be ignored.

Compiler Toggles

You set compiler toggle switches by following the program file name with a blank, a dollar sign, and the desired letters as follows:

B Suppresses program listing to the console during compilation. (Default is OFF.)

C Suppresses generation of an .INT file. Saves time when compiling for errors. (Default is OFF.)

D Suppresses translation of lowercase to uppercase letters. (Default is OFF.)

E Lists line numbers where errors have occurred. Toggle E must be ON when you use TRACE. (DEFAULT IS OFF.)

F Sends the compiled output listing to the list device as well as the console. (Default is OFF.)

G Sends a compiled output listing with the same name as the source, but of type .LST, to a disk file. A drive identifier may be appended in parentheses after the toggle code. (Default is OFF.)

Examples:

```
B:CBAS2 A:ACCOUNTS $BGF
CBAS2 PAYROLL $(G)EC
```

Cross-Reference Lister (XREF.COM)

The utility program XREF.COM creates a disk file that lists alphabetically every identifier used in a CBASIC program. The list indicates the usage of the identifiers (function, parameter, or global) and lists each line on which the identifier was used. Functions appear first in the list, with associated parameters and local variables immediately following.

Format:

```
XREF < file name > [disk ref] [$<toggles>] ['<title>']
```

The named source file must be of type .BAS. By default, the listing is sent to the disk on which the source file is located. A drive identifier after the file name sends the cross-reference file to the drive specified.

A file created by XREF has the same name as the source, but adds the extension .XRF. The standard output is 132 columns wide. You can use toggles in the XREF command to direct output of the cross-reference listing. A blank space, a dollar sign, and the following toggle letter codes direct output of the XREF file:

A Outputs the cross-reference file to the list device and disk file.

B Suppresses output to the disk. No output is produced if you specify only B.

C Suppresses output to the disk but permits output to the list device. This toggle has the same effect as specifying A and B.

D Produces 80-column output instead of 132.

E Produces output of identifiers and their usage without line numbers.

F(*n*) Lets you change the default page length (60 lines) to *n*.

G Suppresses form-feeds and printing of heading lines.

H Suppresses translation of lowercase to uppercase letters.

The optional 'TITLE' field must be the last on the command line. The characters indicated between the apostrophes are printed as the heading for each page. You can specify up to 30 characters when the output width is set to 132, and 20 if the column width is set to 80.

TRACE

The TRACE option allows run-time debugging of a program by printing the compiler-assigned line number of each statement as it is executed.

Format:

```
CRUN2 <filename> [TRACE] <Ln1>[,<Ln2>]]]
```

You must have used the E toggle in the source program in order for TRACE to work. The first number specifies the line number where tracing should begin and the second number specifies where tracing should end. You can use the first number alone to begin tracing at any desired line. If you don't specify a line number, the entire program is traced.

Example:

```
CRUN2 EXAMPLE TRACE 1,3
```

would cause the following output:

```
AT LINE 0001
AT LINE 0002
AT LINE 0003
```

MBASIC

Special Control Characters

^A	enters EDIT mode on current line or previously typed line.
^C	interrupts program execution; returns to BASIC command level.
^G	rings bell.
^H	deletes last character entered.
^I	advances to next tab stop (every 8 columns).
^J	divides logical line into physical lines.
^O	halts/resumes program output.
^R	retypes current line.
^S	suspends program execution.
^Q	resumes execution following **^S**.
^U	deletes current line.
^X	deletes current line.
<RETURN>	ends every program line.
<ESC>	leaves EDIT mode subcommands.
.	defines current line for EDIT, RENUM, DELETE, LIST, LLIST.
&O or **&**	serves as a prefix for octal constant.
&H	serves as a prefix for hexadecimal constant.
:	separates statements entered on the same line.
?	serves as a PRINT statement equivalent.

MBASIC Commands

AUTO — *automatically generates line numbers*

The AUTO command sequentially numbers program lines following each carriage return.

Format:

AUTO [line number [, increment]]

Line numbering begins with the number following AUTO and is incremented as specified by the next number; the default value for both is 10. The increment value specified by the most recent AUTO command is adopted when you use a comma but no increment value.

An asterisk appears when AUTO generates an existing line number indicating that the contents of the existing line will be replaced. A carriage return typed after the asterisk skips the existing line and generates the next line number. **^C** terminates AUTO without saving the current line and returns to the command level.

Examples:

AUTO 15,5	Generates line numbers 15,20,25,30,35. . .
AUTO	Generates line numbers 10,20,30,40,50. . .

CLEAR—*reallocates memory space*

The CLEAR command is used to set numeric variables to zero, string variables to null, and, optionally, to set the end of memory and amount of stack space.

Format:

CLEAR [,[<expression>] [,<expression>]]

The first expression, if present, sets the highest memory location available for use. The second expression reserves stack space. The default stack space setting is either 256 bytes or an eighth of the available memory, depending on which is smaller.

Examples:

CLEAR ,32554	Sets highest memory to 32554
CLEAR ,,2999	Sets stack size to 2999
CLEAR ,32554,2999	Sets highest memory to 32554 and stack size to 2999

CONT — *resumes program execution*

The CONT command causes execution of the current program to continue following **^C**, STOP, or END statement.

Format:

CONT

Execution continues where it left off when a CONT command is issued. If execution halted following a prompt for an INPUT statement, the prompt is redisplayed. CONT is usually used with STOP to examine and change values or to resume execution following an error.

Example:

(See STOP statement.)

DELETE — *deletes program lines*

The DELETE command erases program lines between the first and second line numbers listed.

Format:

DELETE [<first line number>]
[<second line number>]

You can delete program lines from the beginning of the file to any desired line number by following DELETE with a hyphen and the last line to be eliminated. The command level always returns following execution of a DELETE command. If a nonexistent line number is referenced, an "Illegal function call" error occurs.

Examples:

```
DELETE 10
DELETE 10–50
DELETE –100
```

EDIT—*enters edit mode*

The EDIT command lets you edit a specified program line. Once in the edit mode you use subcommands to move the cursor, insert, delete, replace, or search for data.

Format:

EDIT <line number>

When you issue the EDIT command, the specified line number is listed followed by a space. Alternatively, **^A** can be used to enter the edit mode for the line currently being written. When **^A** is issued, the edit mode is indicated by a carriage return and an exclamation mark. If a syntax error occurs during program execution, the edit mode is automatically entered.

A number may precede many of these subcommands, indicating that the command should execute that many times; (*n*) represents any number. The editing subcommands are summarized below:

(*n*) SPACE moves the cursor right (*n*) number of characters.

(*n*) ← or **^H** moves cursor left (*n*) number of characters (destructive in insert mode).

(*n*) D deletes (*n*) number of characters to the right of the cursor.

I (string) inserts the specified (string) at the cursor position.

H (string) deletes all characters right of the cursor and inserts the specified (string).

X (string) moves the cursor to the end of the line and inserts the specified (string).

(*n*) S (target) finds the (*n*)th occurrence of the specified (target) character.

(*n*) K (target) deletes all characters to the (*n*)th occurrence of the specified (target) character.

(*n*) C (list) changes the next (*n*) characters to the specified (list) of characters.

A restores original line for reediting.

L lists the remainder of the line and repositions the cursor at the beginning.

Q abandons the edit mode and returns to the command level without saving changes.

RETURN saves the newly edited line and exits from the edit mode.

E saves the newly edited line without displaying the remainder.

NOTE

If you enter an unrecognized edit subcommand or parameter, the bell will ring.

FILES—*lists files on the currently active drive*

The FILES command is used like the CP/M DIR and DIRSYS commands to view a directory of the files residing on the current disk.

Format:

```
FILES [<filename>]
```

You can list all files or a specific file on the active drive, or you can view a category of files. The standard CP/M wildcard conventions can be used.

Examples:

```
FILES
Files *.BAS
Files MBASIC.BA?
```

LIST—*lists program lines*

The LIST command causes specified program lines to be listed on the screen.

Format:

```
LIST [<line number>[ -[<line number>]]]
```

Typing the LIST command while you are in the command mode provides a listing of all program lines in memory. A single line number or a range of lines between a first and second line number may be listed. A hyphen used in place of either the first or the second line number lists all program lines beginning or ending at the indicated line number.

Examples:

```
LIST
LIST 10
LIST 10–100
LIST –50
LIST 50–
```

LLIST—*prints program lines*

The LLIST command causes specified program lines to be listed at the printer instead of at the console.

Format:

```
LLIST [<line number>[ -[<line number>]]]
```

The LLIST command functions exactly as the previously described LIST command, except that the printer serves as the output device.

LOAD—*loads a program file*

The LOAD command causes a named disk file to be loaded in memory.

Format:

```
LOAD <file name> [,R]
```

The file name is the name under which the file was saved and whose default extension, .BAS, is supplied by CP/M. When LOAD is issued, all files are closed and memory is cleared before the specified file is loaded. The R option leaves data files open and automatically runs the newly loaded program. You use the R option to chain programs that share common data.

Example:

```
LOAD "NEWFYL", R Assumes file type of .BAS
```

MERGE—*combines two programs*

The MERGE command references and combines a disk file with the program in memory.

Format:

```
MERGE <filename>
```

The file name is the name you assigned to the file when you saved it. Only files saved in ASCII format may be merged, or else a "Bad file mode" error will occur. CP/M automatically furnishes a .BAS-type extension. Lines in the file being merged replace those in the calling file if line numbers are duplicated. The command mode returns following a MERGE command.

Example:

```
MERGE "DATAFYL"
```

NAME—*renames a file*

The NAME command renames a disk file.

Format:

NAME <old file name> AS <new file name>

You list the name of the file to be renamed first, followed by AS and the new name that will identify the file. If the old file name does not exist, or if the new file name is already being used, an error occurs.

Example:

```
NAME "OLDFYL" AS "NEWFYL"
```

NEW—*deletes current program from memory*

The NEW command deletes all variables and clears the current program from memory.

Format:

NEW

Issue NEW to clear memory and prepare for a new program to be loaded. This command is usually used on unwanted files or ones that have already been saved. The command mode returns following the NEW command.

Example:

```
NEW
```

NULL—*sets the number of nulls to follow each program line*

The NULL command establishes the number of nulls that are to follow each line in the program.

Format:

NULL <integer expression>

The integer expression signifies the number of nulls output after each line. The default value is zero. You should use 0 or 1 for most 1200-baud or parallel printers, 2 or 3 for 30-cps hard copy printers, and 3 or larger for 10-character-per-second tape punches or typewriter printers.

Examples:

NULL 2

RENUM—*renumbers program lines*

The RENUM command sequentially renumbers program lines.

Format:

RENUM [[<new number>] [,[<old number>] [,<increment.]]]

The new line number replaces the old line number, which is incremented as specified. By default, line numbering begins with the first line number of the program, starting with line 10 and incremented by 10. The newly specified line numbering is also supplied for all existing statements that reference lines by their old numbers. Line numbers cannot be renumbered out of sequence or extend beyond 65529 or an "Illegal function call" will occur.

Examples:

```
RENUM
RENUM 300,,50
RENUM 1000,900, 20
```

RESET—*closes files and updates the directory*

The RESETcommand closes all open disk files and updates the directory information.

Format:

```
RESET
```

You should normally issue RESET before removing a diskette to ensure that all files have been closed and to maintain an accurate directory.

RUN—*executes a program*

The RUN command executes either the program currently in memory or a program stored in a disk file.

Format:

```
RUN <line number>
```

The above format executes the program in memory. Optionally, the program may be executed from a specific line number specified after the command.

Format:

RUN <file name> [,R]

This form of the RUN command executes a particular program stored in a disk file. When the file is loaded, all other files are closed and everything residing in memory clears. The R option can be used to leave data files in memory in an open condition.

Examples:

```
RUN 100
RUN"ANYFILE",R
```

SAVE—*saves named file*

The SAVE command saves the contents of memory to a named disk file.

Format:

SAVE"<file name>"[,A] [,P]

You can save the contents of memory under any valid file name. CP/M automatically supplies a default-type extension of .BAS if you don't supply it. If the name you assign to the file already exists, the contents of the existing file are written over.

Files are saved in binary format; however, you can save the file in ASCII format by placing a comma and the letter A after the file name. You can protect the file from unauthorized access by appending P to the end of the statement. The P option prevents lines in the program from being subsequently listed or edited.

Examples:

SAVE "COMFYL", A	Saves ASCII
SAVE "PROGRAM", P	Saves protected
SAVE "BINARY"	Saves binary

SYSTEM—*closes files, then relinquishes control to CP/M*

The SYSTEM command causes all open files to be closed, updates the directory, and returns to the CP/M command level. Note: **^C** does not exit from MBASIC-80 to CP/M.

Format:

SYSTEM

TRON/TROFF—*enables/disables tracing*

The TRON command is used to turn tracing ON. When tracing is ON, the line number currently being executed appears within brackets. Execution of a TROFF or NEW command turns tracing OFF.

Formats:

TRON

TROFF

Example:

```
TRON
AUTO 10,10
10 A=5
20 PRINT A
30 END
40 ^C
RUN
[10] [20] 5
[30]
TROFF
```

WIDTH—*sets line width*

The WIDTH command establishes the length of lines displayed on the screen or optionally output to the printer.

Format:

```
WIDTH [LPRINT] <integer expression>
```

This command affects the width of lines displayed at the console. The optional LPRINT clause sets the width of lines output to the printer. The integer expression specifies the number of characters, ranging between 15 and 255, that occupy a line; the default line width is 72. A line width of 255 is considered infinite, so carriage returns do not occur automatically at the end of lines. When you position the printhead or cursor using the POS or LPOS functions, it returns to zero after reaching position 255.

Examples:

```
WIDTH 52
WIDTH LPRINT 255
```

MBASIC Statements

CALL—*links to an assembly-language subroutine*

The CALL statement transfers program flow to an assembly-language subroutine.

Format:

```
CALL <variable name> [(<argument list>)]
```

The variable name contains the starting memory address for the subroutine, and it cannot be an array variable name. The argument list holds arguments passed to the subroutine. The argument list cannot contain literals.

Example:

```
10 ROUTINE=&HD000
20 CALL ROUTINE (I,J,K)
```

CHAIN—*passes variables between programs*

A CHAIN statement transfers control and then passes variables to a referenced program.

Format:

```
CHAIN [MERGE] <filename> [,[<line number>] [,ALL]
[,DELETE <range>]]
```

The file name identifies the program being referenced. The line number or expression evaluates to the line where execution of the called program begins. If you omit the line number or expression, execution begins at the first line.

If you use the ALL option, all variables are passed to the called program. You have to use a COMMON statement in conjunction with CHAIN when passing partial variables.

The MERGE option lets you bring in a subroutine as an overlay. A MERGE operation is performed with the current and the called programs. The called program must be an ASCII file if it is to be merged.

To remove an overlay so another overlay can be brought in, use the DELETE option. The line numbers specified in the range are modified accordingly by RENUM.

NOTE

The MERGE option leaves files open and preserves the current OPTION BASE setting. If MERGE is omitted, CHAIN does not preserve variable types or user-defined functions.

Examples:

```
CHAIN "PROG"
CHAIN "PROG", 30, ALL
CHAIN MERGE "PROG", 30
CHAIN MERGE "PROG",30, DELETE, 30–110
```

CLOSE—*closes disk files*

The CLOSE statement concludes input and output to disk files.

Format:

```
CLOSE [ [#] <file number> [,[#] <filenumber...>]]
```

The same number under which a file was opened is used to close the file. The closed file's number is released and may subsequently be used to open another file. If you don't specify a file number, all files are closed.

A CLOSE statement issued for a sequential output file writes the final buffer of output.

NOTE

Files automatically close when the computer encounters an END statement or a NEW command.

Examples:

```
CLOSE #1
CLOSE
CLOSE FILE.1,FILE.2
```

Where FILE.1 and FILE.2 are valid variables that contain the appropriate file numbers.

COMMON—*specifies common variables*

The COMMON statement specifies simple and subscripted variables that are retained in a common area of memory and are passed between chained programs.

Format:

COMMON <list of variables>

Common statements should appear at the beginning of the program. You cannot use the same variable in more than one COMMON statement. To specify an array variable, append opened and closed parentheses "()" to the variable name.

NOTE

If you use the BASIC compiler, you must declare common arrays in preceding DIM statements.

Example:

```
10 COMMON A,B,C,D(),G$
20 CHAIN "PROGRAM.TYP", 10
```

DATA—*holds a data list within a program*

DATA statements define string and numeric constants, which are stored until you assign them to variables through READ statements.

Format:

DATA <list of constants>

You can use DATA statements anywhere in a program to list string, integer, or real constants. Any number of DATA statements are permissible, each occupying one line exclusively. You can list as many constants, delimited by commas, that can fit on a line. The constants from all of the DATA statements are stored in memory in the same sequential order in which they appear in the program. These constants are then assigned to variables within the READ statements being executed.

Constants in DATA statements and variables in READ statements that reference them, must be of the same type. The list of constants can contain any combination of numeric formats except numeric expressions. String constants that contain commas, colons or leading/trailing blanks must be surrounded by quotation marks. An attempt to read past the available data constants will cause an error.

NOTE

The RESTORE statement can be used to reset the data pointer, causing the data to be reread.

Example:

```
110 DATA 1, 2, 2.1, 22.1
111 DATA "Sample, with comma", Sample
without comma
```

DEF FN—*defines user-written function*

The DEF FN statement is used to define and name a function that the user writes.

Format:

```
DEF FN <name> [(<parameter list>)] =<function definition>
```

The function name must be a valid variable name preceded by the letters FN. The parameter list contains variable names that correspond to those used by an expression in the function definition. When the named function is referenced, the values in the parameter list are substituted for the variable names in the function definition. Each variable name in the parameter list must be set off from the next by a comma.

The function definition consists of an expression, that when referenced, performs the required operation. Variables in the expression are interpreted only in the context of the function. If the variable is not explicitly stated in the parameter list, the current value of the variable is assumed.

User-defined functions can be either string or numeric. When a type is specified in the function name, the value of the expression is forced to match it before returning to the calling file. If the type of the function name and the argument are different, then a "TYPE MISMATCH" error will occur. An error also occurs if the DEF FN is not executed prior to the function it defines.

Example:

```
100 DEF FNEF (X,Y)=X^2+Y^2
110 A=FNEF (3,4)      (A would equal 25 in
                       this example.)
```

DEF INT/SNG/DBL/STR—*defines variable types*

The DEF statement identifies variable names that begin with a predetermined letter as being either integer, single- or double-precision, or string-type variables.

Format:

DEF <type> <range(s) of letters>

The type is defined as INT (integer), SNG (single-precision), DBL (double-precision), or STR (string). The range of letters listed after the type declaration identify variable names that begin with the indicated letters as being a specific type of variable. However, type-declaration characters have precedence over these DEF statements in the typing of variables. Undefined variables without declaration characters are assumed to be single-precision variables.

Example:

```
100 DEFSTR A–F
110 DEFDBL G
120 DEFINT H–M, N–S
```

DEF USR—*defines assembly subroutine entry point*

The DEF USR statement defines the starting address of an assembly-language subroutine.

Format:

DEF USR [<digit>] =<starting address>

A digit from 0 to 9 corresponds to the number of the USR routine whose starting address follows; USR 0 is assumed if no digit is specified. You can use any number of DEF USR statements in a program to redefine subroutine starting addresses so as many subroutines as necessary can be accessed.

Example:

```
100 DEF USR9=2200
110 A=USR9 (X^2/2.14)
```

DIM—*allocates storage for an array*

The DIM statement allocates storage for an array and defines the upper limit of each subscript; a lower-bound limit of zero is assumed.

Format:

DIM <list of subscripts>

The subscript list indicates the number and extent of dimensions for the array being declared. The minimum value of a subscript is considered to be 0 unless an OPTION BASE statement designates otherwise. Subscripted variables that are not dimensioned have an upper-bound limit of 10. If a subscript larger than the value specified in the DIM statement is encountered, a "Subscript out of range" error occurs.

Example:

```
10 DIM X(20)
20 FOR Y=0 to 20
30 READ X (Y)
40 NEXT Y
```

END—*terminates program execution*

The END statement halts program execution, closes all files, and returns to the command level.

Format:

END

You may place END anywhere in the program to terminate program execution. An END statement is not required at the end of a file.

Example:

```
10 IF A=20 THEN END ELSE GOTO 20
```

ERASE—*eliminates specified arrays*

The ERASE statement erases previously defined arrays. Following their erasure, the space occupied by the arrays is released, and you can use it to redimension the arrays.

Format:

ERASE <array variable list>

If you make an attempt to redimension an array without first erasing it, a "Redimensioned array" error will occur.

Example:

```
10 ERASE X,Y
20 DIM Y(50)
```

ERROR—*allows user-defined error codes or simulates error*

The ERROR statement allows error codes to be defined, or may be used to simulate the occurrence of a specified error.

Format:

ERROR <integer expression>

The value of the integer expression must be between 0 and 255. To define an error code, specify a value greater than any existing MBASIC error codes. An error-trapping routine can then handle the user-defined error code. Errors are simulated, with an integer expression corresponding to the desired MBASIC error code, which causes the associated error message to be printed. If the ERROR statement references a code for which no error message has been designed, an "Unprintable error" message appears. If no error-trap routine is associated with the error, then an error message appears and execution is terminated.

Examples:

```
10 ON ERROR GOTO 100
20 INPUT "TYPE A NUMBER FROM 1 TO 10"; N
30 IF N<1 or N>10 THEN ERROR 200
```

ERR and ERL—*serve as variables in error routine*

The ERR and ERL variables direct program flow in an error-handling subroutine. The ERR variable contains the error code, and ERL lists the line number where the error was detected.

Format:

```
IF ERR = error code THEN ...
IF ERL = line number THEN...
```

Since ERR and ERL are reserved variables, they cannot be placed to the left of the equal sign in a LET statement. The applicable error codes are listed in the section on MBASIC error messages.

FIELD—*defines a field in a random file buffer*

The field statement allocates space for variables in a random file buffer. A FIELD statement must execute before you can use a GET or PUT statement to extract or insert data in the random file buffer.

Format:

```
FIELD[#] <filenumber>,<field width>AS <string variable>
```

The same number used to open the file identifies the file whose variable fields are to be defined. The number of character spaces to be reserved in the buffer for the string variable is determined by the field width. The total number of bytes allocated by the field width cannot be larger than the record length specified when the file was opened. As many field statements as necessary can execute for the same file, with all being in effect simultaneously.

NOTE

Variables whose fields have been allocated should not be used in INPUTor LET statements. Once a variable field is defined, any subsequent input will move the pointer in the random file buffer.

Example:

```
FIELD #1,5 AS FIRST$, 10 AS SECOND$
```

FOR . . . NEXT —*establishes loop parameters*

The FOR statement defines an index, establishes an initial and terminal index value, and initiates a loop. The NEXTstatement diverts execution back to the corresponding FOR statement.

Format:

```
FOR<index = <initial TO<terminal [ STEP< degree >]

.
.
.
NEXT [<index ] [,< index ...]
```

The index is used as the counter and must be an unsubscripted variable. The initial value of the counter is established by the first numeric expression, with the terminal value being specified by the second expression. The counter is either increased or decreased each time the loop is completed and incremented by one unless modified by STEP.

You can use the STEP option to increment the index by degree. A positive STEP value causes the loop to be executed until the index value exceeds the terminal value. STEP can be negative, however, in which case the index is decremented until the counter value is less than the initial value. You can nest FOR/NEXT loops, as long as each has its own unique index variable.

The NEXT statement causes the associated FOR statement to execute until the loop is terminated. If you use an index variable, it must match the index variable in the corresponding FOR statement. More than one FOR/NEXT loop may be terminated by a single NEXT statement listing all the index variables. If no index variable is listed, the most recently encountered FOR statement is terminated. When you nest FOR/NEXT loops, the NEXT statement for the inside loop must appear before that for the outside loop.

Example:

```
10 A=1
20 FOR A=1 to 1000 STEP 2          <---.
30 PRINT A                             : Loop
40 NEXT A                          ----
```

GET—*reads record from random disk file*

The GET statement reads a random-disk file record into a random buffer.

Format:

GET [#] <file number>[,<record number>]

The file number is the number you assign to the file when you open it. The specified record number is read into the buffer, where an INPUT# or LINE INPUT# statement can subsequently access it. If no record number is indicated, the next sequential record following that most recently read is assumed. The highest record number that can be read is 32767.

Example:

```
GET #1, 17*I+1
```

GOSUB . . . RETURN—*executes a subroutine*

The GOSUB statement directs program execution to the beginning line number of a subroutine. The RETURN statement causes execution to continue with the statement after the most recent GOSUB.

Format:

```
GOSUB <line number>
.
.
.
RETURN
```

A subroutine can be referenced many times, and, if necessary, from within another subroutine. Nesting of subroutines is limited only by available memory. More than one RETURN statement may be located in a subroutine, each executing a return where needed. Subroutines should be preceded by a STOP, END, or GOTO statement to prevent unintentional execution.

Example:

```
10 GOSUB 40
20 PRINT "Executing main program again"
30 END
40 PRINT "This is the SUBROUTINE" <--------.
50 RETURN             <-------- Subroutine
```

GOTO—*branches as specified*

The GOTO statement causes program execution to continue, beginning at a specified line.

Format:

```
GOTO <line number>
```

Program execution is diverted to the indicated line number and continues with the first executable statement.

Example:

```
10 READ A
20 PRINT "A=";A,
30 B=3.14 * A^2
40 PRINT "AREA=" ;B
50 GOTO 10
60 DATA 1, 2, 3, 4, 5
```

Note: Endless loop caused by GOTO 10

IF(GOTO)-THEN-ELSE—*directs conditional branch*

An IF-THEN-ELSE statement either branches to a particular line number or executes a group of statements, depending on the value of an expression. An IF-GOTO-ELSE statement functions in the same manner but is strictly for branching to a specific line number.

Formats:

```
IF <expression> THEN [<statements>] or
              <line number>
 [ELSE] [<statements>] or <line number>

   IF <expression> GOTO <line number>
  [ELSE] <statements> or <line number>
```

When the conditional expression following IF is true (not zero), execution of the line number or group of statements following THEN (or line number after GOTO) takes place. When the expression is false (zero), the next executable statement is processed. The ELSE option may execute a group of statements or branch to a specific line.

Nesting of IF (THEN, GOTO) ELSE statements is limited only by line length. When you use multiple ELSE clauses, the closest THEN or GOTO statement is matched.

Examples:

```
IF STATE > 49 THEN HAWAII = NAMEN
IF STATE = 48 THEN ALASKA = NAMEN
  ELSE GOTO 410
IF (HOT=TRUE) GOTO 7734
```

INPUT—*assigns data to variables*

The INPUT statement prompts for data on the screen to be entered and, subsequently, assigned to variables.

Format:

```
INPUT [;] [<"prompt string">;]
        <list of variables>
```

A semicolon following INPUT prevents a carriage return entered via the keyboard from being echoed as a line feed. The prompt string is an optional message (in quotes) that requests the appropriate data during program execution. In the absence of a prompt, a question mark is displayed indicating that data is expected. The semicolon after the prompt string is necessary, but you can use a comma in its place to suppress the prompt question mark.

Variables to be defined are listed at the end of the INPUT statement. The data items supplied for a given variable are separated by commas. Data entered in response to the prompt is assigned sequentially to the variable list. The number of data items must correspond to the listed number of variables. The type of data item must agree with the type of variable. If the data entered is the wrong type or does not match the number of listed variables, the message "?Redo from start" appears.

Example:

```
10 INPUT "Pick a number and get one   free"; A
20 PRINT A "PLUS 1 IS"; A+1
30 END
```

INPUT# — *assigns stored data to variables*

The INPUT# statement reads data from a disk file and assigns it to variables.

Format:

INPUT# <file number>, <variable list>

The file number identifies which disk file contains the data to be assigned to the variable list. The data items should be stored in the file as if entered at the keyboard. Numeric values are delimited by a space, carriage return, linefeed, or comma. String items are delimited in the same way but may be enclosed in quotes if one of these delimiters is intended for the string. A data item cannot exceed 255 characters. If end-of-file is reached while inputting data, remaining item(s) are ignored.

Example:

```
10 OPEN "I", #1, "DATA"
20 INPUT#1,N$, D#, H$
30 IF RIGHT$ (H$,2)= "78" THEN PRINT N$
40 GOTO 20
```

INSTR—*searches for a given string*

INSTR looks for the first occurrence of a string in another string and returns its position. The string to be searched is listed first, followed by the string portion being searched for. An optional offset can precede both of these strings to define the starting position of the search. INSTR returns a 0 if starting position is null or greater than the string being searched, or if the string being searched for cannot be found. If the string being searched for is null, a 1 is returned.

Format:

```
INSTR ([position],<search string>,
    <string being searched>)
```

Example:

```
10 A$ = "SEARCH STRING"
20 B$ = "S"
30 PRINT INSTR (4,A$,B$)
RUN
8
```

INT—*returns the integer portion of the argument*

The INT function eliminates the fractional portion of the argument and returns the integer portion. If the argument is a real number, it is first converted to an integer and then converted back to a real number.

Format:

```
INT (<numeric expression>)
```

Example:

```
PRINT INT (9.8)
```

KILL—*deletes a disk file*

The KILL statement deletes a given file from the current diskette.

Format:

```
KILL <filename>
```

Only closed files may be eliminated with the KILL. If KILL is specified for an open file, a "file already open" error will occur. You can expunge any type of file with the KILL statement.

Example:

```
100 KILL "DATA.FYL"
110 KILL "data.fyl" <------ Useful for
                            eliminating
                            lowercase
                            file names
```

LET—*assigns value to variable*

The LET statement assigns the value of an expression to a variable.

Format:

```
[LET] <variable>=<expression>
```

The word LET is optional and may be omitted; the equal sign itself denotes an assignment.

Example:

```
10 LET A=12
20 B=A+1
```

LINE INPUT—*assigns data to a string variable*

The LINE INPUT statement asks for a line of data that, when entered via the keyboard, is assigned to a string variable.

Format:

```
LINE INPUT [;] [,"prompt string"]
        <string variable>
```

LINE INPUT may be followed by a semicolon, in which case the carriage return used to enter the line is not echoed. The prompt string is a literal that appears on the screen asking for the line to be input. The line entered at the keyboard can be up to 254 characters long. The input line is entered by following it with a carriage return. You can use ^C to escape from a LINE INPUT statement and return to the command level, if necessary; CONT resumes execution at the LINE INPUT.

Example:

```
10 LINE INPUT, A$
20 LINE INPUT; "NAME";N$
```

LINE INPUT#—*reads line from disk file*

The LINE INPUT# statement reads a line up to 254 characters long from a sequential disk file and assigns this data to a string variable.

Format:

LINE INPUT# <file number>, <string variable>

The file number identifies the file containing the data to be assigned to the string variable. The data for one LINE INPUT# statement is delimited from the next by a carriage-return linefeed. You usually use LINE INPUT# when you've broken lines of a data file into fields or when a program is reading an ASCII-saved program file into another program.

Example:

```
10 LINE INPUT#2, B$
```

LPRINT—*outputs data to the line printer*

The LPRINTand LPRINT USING statements perform the same action as the PRINT [USING] statements, except that data is output to the printer instead of on the screen.

Format:

LPRINT [<list of expressions>]

Lines printed through the LPRINTstatement assume a 132-character-wide line.

LSET—*stores left-justified data in random file buffer*

The LSETstatement transfers data from memory to a random file buffer, or you can use it with a nonfielded string variable to left-justify a string in a given field.

Format:

LSET <string variable> = <string expression>

LSET left-justifies a string expression in a corresponding string variable. If the string exceeds the field length, the material on the right is truncated. If numeric values are to be affected, you must first convert them to a string, using an MK (I,S,D)$ function.

Example:

```
10 LSET A$= MKS$(MAX)
20 LSET B$= "JOHN DOE"
```

MID$—*replaces portion of a string*

The MID$ statement substitutes characters in an existing string with replacement characters.

Format:

MID$ (<string exp1>,n [,m])=<string exp2>

The characters in the first string expression, beginning at the nth character position, will be replaced with the characters in the second expression. You can use the m parameter to signify the number of characters involved in the replacement. A MID$-initiated replacement may never exceed the character length of the original expression.

Example:

```
10 A$="1981"
20 MID$ (A$,4)="2" or
MID$ (A$,3,2) = "82"
30 PRINT A$
```

ON ERROR GOTO—*enables error-trap routine*

The ON ERROR GOTO statement executes an error-handling subroutine when an error is encountered.

Format:

ON ERROR GOTO <line number>

ON ERROR GOTO enables error trapping. When an error is encountered following this statement, the error-handling subroutine beginning at the specified line number executes. To disable error trapping so subsequent errors halt execution and display the appropriate error message, execute an ON ERROR GOTO O statement. You can disable error trapping within an error-handling subroutine when you expect errors for which you haven't formulated a recovery action.

Example:

```
10 ON ERROR GOTO 500
```

ON GOTO (GOSUB)—*conditional branch*

The ON GOTO statement directs program execution to one of several line numbers, depending on the value of an expression. The ON GOSUB statement conditionally executes a subroutine, beginning at one of several line numbers, depending on the value of an expression.

Formats:

ON <expression> GOTO <list of line numbers>

ON <expression> GOSUB <list of line numbers>

The expression value determines which of the line numbers will execute: fractions are rounded. If the expression is zero, or greater than the number of items in the list (without exceeding 255), the next valid statement is executed.
If the value of the expression is negative or greater than 255, an "illegal function call" error occurs. Use a RETURN statement to terminate each subroutine.

Example:

```
10 ON A% GOTO 100,200,300,
20 ON B% GOSUB 400, 100, 200
```

OPEN—*activates an existing file*

The OPEN statement allows I/O operations to be performed on a named file. When this statement is executed, buffer space is allocated for the indicated file, and the mode of access to be used with the buffer is defined.

Format:

```
OPEN <mode>, [#] <file number>, <file name>, [<reclen>]
```

The first character of a string expression indicates the desired mode of access. The letter O indicates a sequential output mode, I indicates a sequential input mode, and R indicates either a random input or output mode of access. The file number may be 1, 2 or 3. To open files numbered from 4 to 15, reinitialize with A>MBASIC /F#(4–15). In all cases the file must be currently unoccupied. You will subsequently use the assigned file number to access the file. The file name is the name you gave to the file when you saved it. The default record length is 128 bytes, but you can alter it by specifying a length at the end of the statement. A file you open for sequential input or random access can occupy more than one file number; however, a file opened for output can be associated with only one file number.

OPTION BASE—*declares subscript array value*

The OPTION BASE statement declares the minimum value for an array subscript. The default base value is 0, and the highest base value possible is 1.

Format:

OPTION BASE (1 or 0)

Example:

OPTION BASE 1

OUT—*sends byte to port*

The OUT statement outputs a specific byte of data to a hardware output port.

Format:

OUT <port>, <byte>

The integer expression, representing the port and byte of data to be output, must range between 0 and 255.

Example:

```
10 OUT 20, 100
```

POKE *— writes byte in memory*

The POKE statement writes a specific byte of data to memory address.

Format:

POKE <address>, <byte>

The memory address to be referenced precedes the byte of data to be written. The address must be in the range of 0 to 65536, and the specified byte must range between 0 and 255. POKE is usually used in association with the PEEK function.

Example:

```
10 POKE &H5A00, &HFF
```

PRINT*— outputs data to the console*

The value of each expression listed in a PRINT statement is displayed on the screen.

Format:

PRINT [<list of expressions>]

Numeric and string expressions listed in the PRINTstatement are evaluated and then printed on the screen. String expressions must be surrounded by quotation marks. If no expression is listed, a blank line is output.

You dictate the format of printed values through the use of punctuation characters. Each line is divided into print fields of 14 spaces each. A carriage return is output at the end of each PRINT statement line unless you suppress it with a comma or a semicolon. A comma after an expression indicates that the next value printed should start in the next field. Spaces or a semicolon after an expression mean that the next value to be printed should continue immediately after the value before it. When a comma or semicolon terminates the statement, the values printed for one PRINT statement may continue with those from another.

Positive numbers are printed with a leading space, while negative numbers are preceded by a minus sign; all numbers are followed by a space. Single-precision numbers that can be represented with less than 6 digits or double-precision numbers that can be represented with less than 16 digits are so printed.

NOTE

A question mark may be substituted for the word PRINT.

Example:

```
10 A=10
20 PRINT "TEN PLUS ONE EQUALS";
  A+1
(or ? "TEN PLUS ONE EQUALS"; A+1)
  30 END
```

PRINT USING—*prints formatted data*

The PRINT USING statement outputs formatted data on the screen.

Format:

PRINT USING <string expression>;
<expression list>

The string expression consists of data-field specifiers and possible literals enclosed in quotation marks. These field specifiers define the format to be used when values for the expression list are output. Each expression in the list is delimited by a semicolon and will be printed in the specified format. String-field specifications used to define formatting are described below:

Literal Characters:

Characters that are not part of a string- or numeric-field format are assumed to be literal characters. Blank spaces, for example, are frequently used to separate fields. You can explicitly identify a character as a literal by preceding it with a slash character (\).

Examples:

```
PRINT USING "### ###";A$
PRINT USING "NO: ###";B$
PRINT USING "\$"; C
```

Numeric Fields:

Numeric integer fields are indicated by one # character per digit. Specify a real numeric with a decimal point by placing the period where you need it. You can format numeric fields with commas by positioning them anywhere in the field. The position of the comma is irrelevant, since it will appear after every third character.

Examples:

```
PRINT USING "###.##"; A
PRINT USING "####,####"; B
```

You can specify exponential numeric format by appending one or more carets (^) to the end of the numeric-field definition. The decimal-point position affects the exponent's value, since four character positions are always added for the exponent.

Example:

```
        PRINT USING "#.####^"; A
```

You can pad a numeric field with leading asterisks. A dollar sign will appear in front of the first nonblank digit if the numeric-field definition begins with two dollar signs ($$). You can place a leading or trailing minus sign wherever you want it.

Examples:

```
PRINT USING "**#####.##";A
PRINT USING "$$####.##"; B
PRINT USING "-#####"; C
```

String Variables:

Specify variable-length fields with ampersands. A fixed-length field is defined by a slash (/) before and after it. An exclamation indicates a single-character string field.

PRINT USING# *—outputs data to disk*

The PRINT# and PRINT USING# statements function like the PRINTand PRINT USING statements, except that the data is written to a sequential disk file.

Format:

PRINT# <filenumber>,
[USING <string exp>;] <list of exps>

The file number is the number you assigned to the file when you opened it. The string expression consists of the field specifiers you need for formatting. The numeric or string expressions in the list will be written to the specified disk file. An exact image of the data is output, so pay careful attention to the use of formatting specifiers.

PUT *—writes data from buffer to disk*

The PUTstatement writes a record from a random buffer to a random disk file.

Format:

PUT [#] <file number> [,<recordnumber>]

The file number assigned to the disk file when you opened it references the file in this statement. The file number is followed by the number of the record to be written. The record number can range between 1 and 32767. You usually use a PUTstatement to recover data stored in the buffer by a PRINT#, PRINT USING#, or WRITE# statement.

Example:

```
PUT #3,4
```

RANDOMIZE—*seeds random-number generator*

The RANDOMIZE statement is used to seed the random-number generator.

Format:

RANDOMIZE [<expression>]

The expression indicates the number to be seeded and may range from −32768 to 32768. If you omit it, program execution stops, the message "Random Number Seed (−32768 to 32767)?" is displayed; execution resumes when you supply this number. If not reseeded, the RND function will return the same sequence of random numbers.

Example:

```
LIST
5 PRINT"START?";
10 DUMMY$=IHKEY$
15 S=S+1
20 IF S=32769! THEN S=-32768!
25 IF DUMMY$<>"Y" THEN 10
30 RANDOMIZE S
35 FOR A=1 TO 50 STEP 2
40 PRINT INT(RND*100);
45 NEXT A
```

READ—*assigns values to variables*

READ statements extract values from DATA statements and assign them sequentially to variables.

Format:

READ <list of variables>

The numeric and string values in DATA statements are assigned sequentially to READ statement variables in the same order in which they appear in the program. The type of listed variables must match the data values used. One READ statement may read more than one DATA statement, and a single data statement may supply variables for multiple READ statements. If there are more READ variables than DATA items, an "Out of data" error occurs. If there are fewer READ variables than DATA items, variables are read from the next READ statement, if present, or are ignored.

NOTE

Use RESTORE statement for reading data from the start of a program.

Example:

```
10 FOR A=1 TO 10
20 READ B (A)
30 NEXT A
40 DATA 1, 2, 3, 4, 5, 6, 7
50 DATA 8, 9, 10
```

REM—*indicates a remark*

All characters to the right of a REM statement are considered remarks and are ignored by program execution.

Format:

```
REM <any characters>
```

You can continue a REM statement on the following line by using a backslash. You can use these statements as a target when branching execution through the use of a GOTO or GOSUB statement. Indicate remarks in midline with an apostrophe (') instead of the statement word REM.

Example:

```
REM COMPUTING THE MEAN
REM EQUIVALENT
REM ' MEANING THE COMPUTER
REM EQUIVALENT
```

RESTORE—*restores data-list pointer*

The RESTORE statement provides a method by which DATA statements can be read, beginning at any location in the program.

Format:

```
RESTORE [<line number>]
```

Execution of the RESTORE statement causes subsequent READ statements to access data beginning with the first DATA statement or a specified line number.

Examples:

```
RESTORE
RESTORE 20
```

RESUME—*resumes execution after an error routine*

A RESUME statement is located in an error-trap routine to cause the resumption of execution following an error-recovery procedure.

Format:

RESUME [0] [NEXT] [<line number>]

Both RESUME and RESUME 0 cause execution to resume with the statement where the error occurred. RESUME NEXT resumes execution with the statement following the one that caused the error. You can specify a line number where execution should resume.

Example:

```
ON ERROR GOTO 1000
```

RSET—*stores right-justified data in a random file buffer*

The RSET statement transfers data from memory to a random file buffer, or you can use it with a nonfielded string variable to right-justify a string in a given field.

Format:

RSET <string variable> = <string expression>

RSET right-justifies a string expression in a corresponding string variable. If the string exceeds the field length, characters are dropped from the right. Numeric values must be converted to string in order to be affected.

Example:

```
90 A$=SPACES (10)
100 RSET A$=B$
```

STOP—*terminates program execution*

When the STOP statement is encountered, program execution is halted and control returns to the command level.

Format:

```
STOP
```

A STOP statement may appear anywhere in a program to terminate execution. When execution halts with STOP, the message "Break in line ______" appears, indicating the number of the last line executed. Execution may be continued with a CONT statement.

Example:

```
STOP
```

SWAP—*exchanges variable values*

A SWAP statement is used to exchange the value of one variable for that of another.

Format:

```
SWAP <variable>, <variable>
```

Single-precision, double-precision, string, or integer variables may be exchanged. The swapped variables must, however, be of the same type or a "Type mismatch" error will occur.

Example:

```
SWAP A$,B$
```

WAIT—*suspends execution and monitors port status*

The WAIT statement temporarily halts program execution while the status of a hardware port is evaluated. Execution resumes when a particular bit pattern is achieved.

Format:

```
WAIT <port number>, I[,J]
```

The port number is exclusive OR'ed with the expression J, and then AND'ed with I. If J is omitted, its value is assumed to be 1. The port data is read repeatedly until the value is other than 0, in which case the next executable statement is processed.

NOTE

If an infinite loop occurs, the machine must be reset.

Examples:

```
WAIT 32, 3
WAIT 32, 3,2
```

WHILE/WEND—*performs a conditional loop*

The WHILE portion of the statement initiates the loop, which is repeated until the value of the expression is false (0). The WEND portion of the statement terminates the loop when the expression is deemed true (not zero).

Format:

```
WHILE <expression>
.
.
.
WEND
```

Looping continues until the value of the expression is zero. The loop may contain any number of statements, including other WHILE/WEND statements. When WHILE/WEND statements are nested, each WEND matches the most recent WHILE. When the expression is evaluated as other than zero, execution continues with the statement following WEND.

Example:

```
WHILE AMOUNT <=MAX    <---------
WEND                  ---------:  Loop
```

WRITE—*outputs data at console*

The WRITE statement sends specified data to the console.

Format:

```
WRITE [<list of expressions>]
```

Numeric and/or string expressions in the list are evaluated and displayed on the screen. Each expression in the list must be separated from the next with a comma. If the list of expressions is omitted, a blank line is output. A carriage return is supplied following the last item in the list.

Example:

```
10 A=10: B=20: C$= "THE END"
20 WRITE A,B,C$
```

WRITE# —*writes data to disk*

The WRITE# statement outputs specified data to a named sequential disk file.

Format:

WRITE# <filenumber>, <list of expressions>

You must open a file in the output mode before you can write data to it. The file to be written to is subsequently referenced by the same number you assigned the file when it was opened. Commas in the list delimit string and numeric expressions.

The WRITE# statement performs essentially the same function as the PRINT# statement, except that delimiters are supplied when the data is written to disk.

Example:

```
WRITE #1,A$,B$
```

MBASIC Functions

ABS—*returns absolute value*

The ABS function returns the absolute value of a numeric expression.

Format:

```
ABS (<numeric expression>)
```

Example:

```
COST-ABS (5*40 (-5))
PRINT ABS(COST-PROFIT)
```

ASC—*returns an ASCII value*

The ASC function returns the ASCII integer value for the first character of the argument. Only the first character is considered. If the expression is evaluated as a null string, or if the argument is numeric, an error will occur.

Format:

```
ASC (<string expression>)
```

Example:

```
I%=ASC (STRING$)
FIRST% = ASC (FIRST.AME$)
```

ATN—*returns arctangent of the argument*

The ATN function is used to return the arctangent of an argument. The argument must be expressed in radians. The value returned is a real number. The arctangent returned can be used to compute various inverse trigonometric functions.

Format:

ATN (numeric expression)

Examples:

```
ANGLE = ATN (X)
ASIN = ATN (X/SQR(1.0-X*X))
```

CDBL—*converts argument to double precision*

The CBDL function converts a numeric expression to a double-precision argument.

Format:

CDBL (<numeric expression>)

Example:

```
10 A = 126.67
20 PRINT A; CDBL (A)
```

CHR$—*returns ASCII string equivalent of the argument*

The CHR$ function converts the argument to a single-character string. The argument is assumed to be an ASCII code. The function returns the ASCII character represented by the argument. If the argument is greater than 255, the high-order byte is ignored.

Format:

CHR$ (<numeric expression>)

Example:

BELL$ = CHR$ (7)

CINT—*converts the argument to an integer*

The CINT function converts a numeric expression to an integer by rounding the fractional portion. An overflow error occurs if the numeric expression is less than −32768 or larger than 32767.

Format:

CINT (<numeric expression>)

Example:

PRINT CINT (9.6)

COS—*returns the cosine of the argument*

The COS function returns the cosine of the argument. The argument must be expressed in radians. The value returned is a real number.

Format:

COS (numeric expression)

Example:

```
10 A=3*COS (.5)
20 PRINT A
```

CSNG—*converts the argument to single precision*

The CSNG function converts a numeric expression to a single-precision number.

Format:

```
CSNG (<numeric expression>)
```

Example:

```
10 A# = 123.345
20 PRINT A#; CSNG (A#)
```

CVI, CVS, CVD—*convert string value to numeric value*

The CV(I,S,D) functions convert string to numeric values. CVI converts a two-byte string value to an integer, CVS converts a four-byte string to a single-precision number, and CVD converts an eight-byte string to a double-precision number.

Formats:

```
CVI (2-byte string)
CVS (4-byte string)
CVD (8-byte string)
```

Example:

```
10 FIELD #1, 4 AS N$, 12 AS B$,...
20 GET #1
30 Y=CVS (N$)
```

EOF—*checks for end of file*

The EOF function returns −1 (true) if the end of a sequential file has been reached. This function is used to test for the end of file before inputting from it, to avoid "Input past end" errors.

Format:

EOF (<file number>)

Example:

```
10 OPEN "I" ,2, "DATA"
20 B=0
30 IF EOF (2) THEN 100
40 INPUT #2, M(B)
50 B=B+1:GOTO 30
```

EXP—*returns the exponent of the argument*

The EXP function returns the value of the constant raised to the power of the argument. The value returned is a real number, even if the value is an integer.

Format:

EXP (<numeric expression>)

Example:

```
10 X = 2
20 PRINT EXP (X-1)
```

FIX—*returns integer portion of the argument*

The FIX function returns the truncated integer portion of a numeric expression. FIX performs the same function as the formula SGN (X) * INT(ABS (X)), except that the next lower number for negative X is not returned.

Format:

FIX (numeric expression)

Example:

PRINT FIX (49.75)

FRE—*shows amount of data memory area remaining*

The FRE function returns the number of available memory bytes in the dynamic, or free, storage area. The value FRE returns is a real number. Free storage may consist of two or more noncontiguous memory blocks. Using FRE(" ") forces a "garbage collection" before returning the number of free bytes (may take 1 to 1½ minutes).

Format:

FRE (0 or string expression)

Example:

PRINT FRE (0)

HEX$—*returns the hex value of the argument*

The HEX$ function returns a string representing the hexadecimal value of the decimal argument. The argument is rounded to an integer before the hexadecimal value is computed.

Format:

HEX$ (<numeric expression>)

Example:

```
10 INPUT X
20 A$ = HEX$ (X)
30 PRINT X "DECIMAL IS" A$
  "HEXADECIMAL"
```

INKEY$—*returns character string from console*

The INKEY$ function returns either a one-character string which is read from the console or a null string if no character is pending. All characters are passed to the program, except for ^C, which causes the program to be terminated.

Format:

INKEY$

Example:

```
1000 'TIMED INPUT SUBROUTINE
1010 RESPONSE$=" "
1020 FOR I%=1 TO TIMELIMIT%
1025 A$ = " "
1030 A$=INKEY$ : IF LEN (A$) = 0
  THEN 1060
1040 IF ASC (A$) = 13 THEN TIMEOUT%
  = 0 : RETURN
1050 RESPONSE$ = RESPONSE$ + A$
1060 NEXT I%
1070 TIMEOUT% = 1 : RETURN
```

INP—*returns a byte from an I/O port*

The INP function returns a byte from a selected input/output port. INP returns an integer that is the value read from the port addressed by the integer expression.

Format:

```
INP (integer expression (0–255))
```

Example:

```
100 A=INP (255)
```

INPUT$—*returns string read from console or disk*

The INPUT$ function returns a string of characters read from the terminal or, optionally, from a disk file. If the terminal is the source of the input, no characters are echoed, and all control characters with the exception of ^C are passed through to the program.

Format:

INPUT$ (numeric expression) [,#file number]

Example:

```
10 PRINT "Enter P to continue or S to stop"
20 X$=INPUT$ (1)
30 IF X$ = "P" THEN 500
40 IF X$ = "S" THEN GOTO STOP ELSE 10
```

LEFT$—*returns leftmost characters*

The LEFT$ function returns the leftmost characters of the argument. The numeric expression specifies the number of characters to be returned. If the number of characters to be returned is greater than the length of the string expression, the entire first argument is returned. If the numeric expression is zero, a null string is returned; if it is negative, an error will occur.

Format:

```
LEFT$(<string expression>,
   <integer expression>)
```

Example:

```
10 A$ = "LEFTMOST"
20 B$ = LEFT$ (A$,4)
30 PRINT B$
RUN
LEFT
OK
```

LEN—*returns the number of characters in a string*

The LEN function returns the number of characters in a string expression. It returns 0 for a null string.

Format:

```
LEN (<string expression>)
```

Example:

```
10 X$ = "NUMBER OF CHARACTERS"
20 PRINT LEN (X$)
```

LOC—*shows next random record or sequential sector*

The LOC function, when invoked for a random file, returns the next default record number to be accessed. When you use LOC for a sequential file, the number of sectors read from or written to the file since its return is shown.

Format:

```
LOC (<file number>)
```

Example:

```
10 IF LOC (1) > 50 THEN STOP
```

LOG—*returns the natural logarithm of the argument*

The LOG function returns the the natural logarithm of the specified argument. The argument must be greater than zero, or an error will occur. You can use this function to calculate logarithms to other bases.

Format:

LOG (<numeric expression>)

Example:

PRINT LOG (45/7)

LPOS—*returns print strike position*

The LPOS function returns the assumed current position of the line printer hammer within the line printer buffer. LPOS does not necessarily give the physical position of the printhead. A numeric expression is used as a dummy argument.

Format:

LPOS (<numeric expression>)

Example:

```
10 IF LPOS (X) > 60 THEN LPRINT CHR$ (13)
```

MID$—*returns a portion of a string*

The MID$ function returns a string that may be any portion of another string. MID$ can accomplish the same function as either RIGHT$ or LEFT$, but, in addition, MID$ can return a string from the middle of another string.

Format:

MID$ (<string expression>,
<integer expression>,[integer])

A portion of the string expression is returned, beginning at the position defined by the integer expression that follows it. An optional integer expression specifies the length of the string to be returned. If the last expression specifies a character position beyond the end of the string, then characters from the position specified by the first integer expression up to the end of the string are returned. A null string is returned if the starting position specified by the second expression is greater than the length of the string, or if the third expression is zero.

Example:

```
10 A$ = "EXTRACT"
20 B$ = "SEARCH STRING EXPRESSION"
30 PRINT A$; MID$ (B$,8,6)
```

MKI$, MKS$, MKD$—*convert numeric value to string*

The MK(I,S,D)$ functions convert numeric values to string values. MKI$ converts an integer to a two-byte string, MKS$ converts a single-precision number to a four-byte string, and MKD$ converts a double-precision number to an eight-byte string. These functions must be used on numeric values that are placed in a random file buffer with an LSET or RSET statement.

Formats:

```
MKI$ (<integer expression>)
MKS$ (<single-precision expression>)
MKD$ (<double-precision expression>)
```

Example:

```
10 AMT = (K+T)
20 FIELD #1, 8 AS D$, 20 AS N$
30 LSET D$ = MKS$(AMT)
40 LSET N$ = A$
50 PUT #1
```

OCT$—*returns octal value of the argument*

The OCT function returns a string that represents the octal value of the decimal argument. The argument is rounded to an integer before being evaluated.

Format:

```
OCT$ (<numeric expression>)
```

Example:

```
PRINT OCT$ (24)
30
```

PEEK—*returns contents of memory location*

The PEEK function returns the contents of a selected memory location. The numeric expression represents the memory location to be examined. PEEK returns an integer value equal to the contents of the specified memory location. For memory locations greater than 32767 the argument must be negative, in which case you should express the argument in hexadecimal notation for clarity.

Format:

PEEK <numeric expression>

Example:

```
BDOS% = PEEK (6)+PEEK (7)*256
```

POS—*returns position of the next print character*

The POS function returns the current cursor position. The leftmost column is position 1.

Format:

POS (<numeric expression>)

Example:

```
IF POS (X)>60 THEN PRINT CHR$ (13)
```

RIGHT$—*returns rightmost characters of a string*

The RIGHT$ function returns the rightmost characters of the argument. The numeric expression specifies the number of characters to be returned. If the number of characters to be returned is greater than the length of the string expression, the entire first argument is returned. If the numeric expression is zero, a null string is returned; a negative numeric expression causes an error to occur.

Format:

RIGHT$ (<string expression>,
<integer expression>)

Example:

```
10 A$ = "STRING EXPRESSION"
20 B$ = right$ (A$,4)
30 PRINT B$
RUN
SION
```

RND—*returns a random number*

The RND function returns a random number between 0 and 1. The RND function generates the next random number in a sequence based on the current seed. The value returned is a real number. The RANDOMIZE statement must be executed to generate a seed and a random number sequence. An optional numeric expression may direct the sequence. If the expression is less than 0, then the sequence repeats; if it is greater than 0 or omitted, the next random number in the sequence is generated. If the expression equals 0, then the same sequence repeats.

Format:

```
RND ([numeric expression])
```

Example:

```
10 FOR I = 1 to 5
20 PRINT INT(RND*100);
30 NEXT
RUN
24  30  31  51  5
Ok
```

SGN—*returns the sign of the argument*

The SGN function returns −1, 0, or 1, depending on whether the argument is negative, zero, or positive, respectively.

Format:

SGN (<numeric expression>)

Example:

```
IF SGN(TOTAL) = −1 THEN GOSUB 200
ON SGN(X) + 2 GOTO 10, 20, 30
```

SIN—*returns the sine of the argument*

The SIN function returns the sine of a numeric expression in radians. The expression is calculated in single precision.

Format:

SIN (<numeric expression>)

Example:

```
PRINT SIN (2.7)
```

SPACE$—*returns a specified number of spaces*

The SPACE$ function returns a string of spaces of a specified length. The argument representing the length must range between 0 and 255 and will be rounded to an integer.

Format:

SPACE$(<numeric expression>)

Example:

```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
```

SPC—*outputs blanks*

The SPC function outputs a specified number of blanks in conjunction with PRINT or LPRINT statements. The number specified can range between 0 and 255.

Format:

```
SPC (<integer expression>)
```

Example:

```
"SPACED" SPC (20) "OUT"
```

SQR—*returns the square root*

The SQR function returns the square root of an argument which must be larger than 0.

Format:

```
SQR (<numeric expression>)
```

Example:

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X, SQR (X)
30 NEXT
```

STR$—*returns string representation*

The STR$ function returns the string representation of the argument.

Format:

STR$ (<numeric expression>)

Example:

```
PRINT STR$ (35)
```

STRING$—*returns a string*

The STRING$ function returns a string whose length is specified by the first integer expression. The characters in the returned string have the same ASCII code as specified by the second integer expression or begin with the first character of the specified string expression.

Format:

STRING$ <integer exp>,
<integer or string exp>

Example:

```
A$ = STRING$(75, "A")
B$ = STRING$ (75, 25)
```

TAB—*positions cursor*

The TAB function positions the cursor at the column specified by the argument. If the cursor is already positioned beyond the specified column, the cursor moves to that position on the next line. TAB may be used only in PRINT statements and must range between 1 and 255.

Format:

TAB (<integer expression>)

Example:

PRINT TAB (20),A$

TAN—*returns the tangent in radians*

The TAN function returns the tangent of a numeric expression in radians, which calculates as a single-precision expression. If an overflow occurs, the "OVERFLOW" message is displayed, the machine infinity is supplied, and execution continues.

Format:

TAN (<numeric expression>)

Example:

A= TAN (3.14)

USR—*calls a user routine*

The USR function calls an assembly-language subroutine identified by the specified argument. The user number, which may range from 0 to 9, corresponds to the user number the DEF USR statement assigns. User number 0 is assumed if you don't specify one.

Format:

USR [user number] (<numeric expression>)

Example:

X = USR9 (Y)

VAL—*converts a string to a real number*

The VAL function converts a string into a real number. The real number VAL returns equals the number the string expression represents. This conversion is equivalent to numeric keyboard input in response to an INPUT statement.

Format:

VAL (string expression)

Example:

PRINT VAL ("3.1")

VARPTR—*returns starting address of the argument*

The VARPTR function returns the beginning address of a variable name or a disk I/O buffer assigned to a file number. Any type of variable name (numeric, string, array) may be examined, with the address returned being in the range 32767 to −32768. When a negative address is returned, add it to 65536 to obtain the actual address. This function is usually for finding the address of a variable or array and then for passing it to an assembly-language subroutine.

Format:

VARPTR (<variable name>) or (#<file number>)

Example:

A= VARPTR (X)
B= VARPTR #(9)

Initialization Options

/F:# if you want to open more than 3 MBASIC files at one time. The maximum number is 15.

/M:# (decimal or hex) sets the highest memory location to be used by MBASIC; when you want to leave some empty RAM space for other purposes.

/S:### must be used to access random files larger than 128 bytes per record.

System Specifications

This section examines the more technical aspects of the Executive computer. For a more elaborate exploration of the hardware and software, refer to the Executive Technical Manual.

Standard Hardware Features

- Z80A central processor
- 124K total RAM
- 4K RAM for two character sets (128 characters/set)
- 4K x 12 RAM for video memory
- 8K ROM
- 2K bytes scratchpad RAM
- 7-inch amber video monitor
- Two double-density disk drives (185K capacity each)
- Two RS-232-C serial ports
- Parallel port: IEEE 488 or Centronics protocols
- External video connector
- Composite video connector (RS-470)
- Full upper/lowercase keyboard, with numeric keypad

Standard Software

- CP/M Plus Operating System (banked version)
- UCSD p-System
- WordStar 3.4 (with MailMerge)
- SuperCalc 1.12
- CP/M Utilities
- Executive Data Base Manager
- Microsoft Disk BASIC 5.3
- CBASIC 2.37
- Osborne Executive Utilities:

    XDIR
    COPY 1.1
    COPYSYS 1.0
    CHARGEN 1.0
    SETUP 1.0

Main Processor Board

Z80A CPU

- Data word width: 8 bits
- 4 MHz primary clock

Memory

- 124K RAM (bank-switched)
- 4K video RAM, 4K of video-attribute RAM
- 8K EPROM
- 4K character-font RAM
- 2K scratchpad RAM
- RAM cycle time: 200 ns
- ROM cycle time: 300 ns

Ports

- Z80A SIO/2 serial communications controller, two standard female DB-25 connectors provided, with a full range (50–9600) of baud rates, software-selectable
- Motorola 6821 PIA (Parallel Interface Adaptor) IEEE-488 may be configured as Centronics-type port
- DMA port—internal

Display

- 7-inch amber screen
- Two 128-character sets (English standard, plus user-defined alternate)
- Writable character font
- 24 lines of 80 characters each
- 8-by-10 dot matrix for each character cell
- 12-bit-wide display memory—7-bit ASCII, plus bit for each of the following attributes:

    Reverse video
    Blink
    Full/half intensity

Underscore
Alternate character set

- 4K memory-mapped display
- Console can be configured via software to emulate most popular terminals
- Cursor definition can be selected by user (block, underline, blinking, or invisible)

Controls

- 69-key detachable keyboard, including 12-key numeric keypad
- Power On/Off switch on front panel
- RESET pushbutton on front panel
- Video brightness and contrast controls on front panel

Disk Drives

- Two half-height drives per unit
- Media: 5.25-inch diskettes, double-density, soft-sectored; Formats supported:

  Osborne 1, single or double density
  IBM PC (CP/M 86)
  DEC VT180
  Xerox 820-1, single density
  Cromemco mini-disk
  UCSD p-System Universal Format

- Seek time: 20 milliseconds, track to track
- Average rotational speed: 200 milliseconds ± 2%
- Recording standards: FM (single density) or MFM (double density)
- 1797 Floppy Disk Controller
- ROM disk routine access for non-CP/M use
- Sense density routines return sector size and number of sectors per track
  Read/write one or more physical sectors at a time

Diskette Capacity (formatted)

(1K = 1024 bytes)	OCC Single Density	OCC Double Density	Xerox Single Density	IBM Double Density	DEC Double Density
Bytes per diskette	100K	200K	90K	160K	180K
Bytes of CP/M space	92K	185K	83	156	171
Bytes per sector	256	1024	128	512	512
Sectors per track	10	5	18	8	9
Tracks per diskette	40	40	40	40	40
Sector translate	2 to 1	2 to 1	5 to 1	1 to 1	2 to 1

Size of Unit (measured with case closed)

Width	20.5 inches	52.07 cm
Height	9 inches	22.86 cm
Depth	13 inches	33.02 cm

Weight

28.5 lb (shipping weight 37 lb)

Portability

Leather handle for carrying
Weather resistant when unit is closed
Power cord stows within case
High-impact plastic case meets UL 94VO flame-retardance specifications

Power Consumption

110/220 volts AC, 50/60 Hz
55 watts maximum

Environmental

Maximum operating ambient air temperature: 95°F (35°C)
Minimum operating ambient air temperature: 32°F (0°C)
Humidity: 95% relative, noncondensing

Safety

Cooling fan
Three-wire, grounded power plug
Three primary-circuit protection points
Switchable fuse mounted on AC panel
Thermal cut-off opens when internal temperature exceeds 150°F
Power supply shuts off under overload conditions

Certifications

FCC UL VDE IEC CSA

Memory Organization

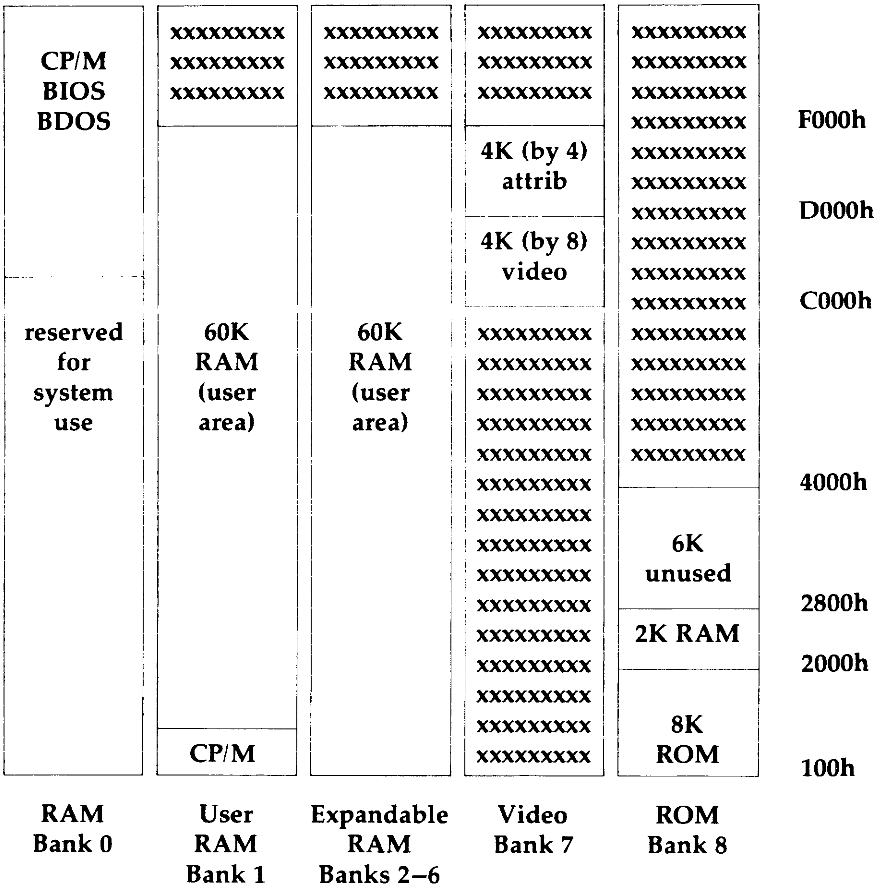
The standard 124K memory of the Executive is organized into 9 banks shared by system routines and user programs. The majority of memory consists of "Random-Access Memory" (RAM) with 8K for "Read-Only Memory" (ROM). Banks 0, 7, and 8 are reserved for system use; Bank 1 is for user programs; and Banks 2 through 6 are not currently implemented.

A bank is simply a range of addresses in memory. Organizing memory into banks allows the CPU to address more memory than would otherwise be available. This is accomplished by bringing in segments of memory as they are needed and hiding those portions not being used. These banks can consist of RAM or ROM, or a combination of both.

Memory Layout

The following memory map shows how the banks are organized.

OSBORNE EXECUTIVE MEMORY MAP



Note: xxxx means that address space is unused by this bank and another bank is seen here.

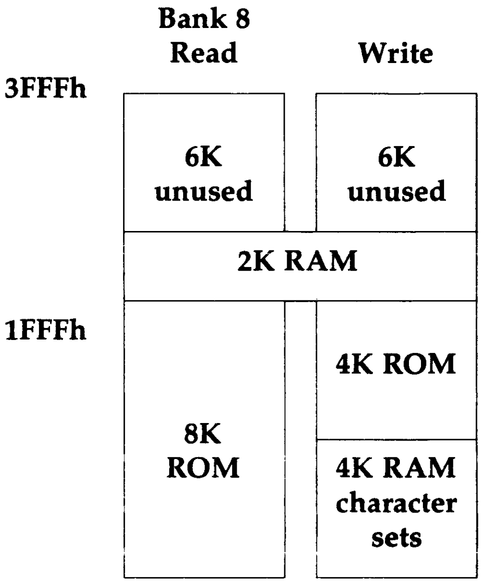
Bank 0 consists of 64K of RAM which is reserved for system use, meaning that it is not available for user programs. It contains BIOS, a portion of BDOS, disk and interrupt buffers. The top 4K is used by the system to control bank switching and data handling, and thus is never shadowed by any other bank.

Bank 1 consists of 60K of memory, with the area between 100h and EFFFh available for user programs. The bottom 256-byte area, from 0 to 100h, is known as Page Zero and contains various, critical system address pointers.

Banks 2 through 6 are not implemented. It is possible to have up to 32 user banks by devising a different memory scheme, providing there is sufficient hardware memory available. (Memory expansion is discussed in the Executive Technical Manual.)

Bank 7 contains the video screen image. The lower portion, from C000 through CFFF, contains bits 0 through 7 of the video memory, plus an area reserved for the Monitor and the DMA port. The upper portion, from D000 through DFFF contains the attribute bits. (A more complete description of the video memory appears in a later section.)

Bank 8 is 16K and occupies memory between addresses 0000 through 3FFF. The memory in Bank 8 is treated differently depending on whether the CPU is reading from or writing to it. When read by the CPU, Bank 8 contains 8K of ROM in the lower portion and 2K of RAM above it. The 2K of RAM is used for temporary storage. The remaining upper 6K is unused. When being written to, the bottom 4K is RAM and is where the two character sets (2K each) reside. Above the area reserved for the character set, ROM is 4K of unused address space. The top 6K of Bank 8 is unused but will still overlay memory with the rest of the bank. The following illustration shows how Bank 8 appears differently in read and write modes:



NOTE

The Technical Manual explains how the memory in Bank 8 can be expanded by substituting two 8K ROMs or two 16K ROMs, and using the appropriate jumpers to select them.

Memory Priority

Banks of memory can be thought of as pages that overlay each other as they are being used. A priority scheme determines which banks take precedence. Bank 0 is always enabled. Banks 1 through 8 are enabled by writing a data byte to port 0, with the bank(s) to be enabled identified by their corresponding bit being toggled ON. Here are the bits associated with each bank.

Bank		**Bit**	
8		7	
7		6	
6		5	
5		4	
4	not used	3	not used
3		2	
2		1	
1		0	
0		(always enabled)	

Those banks that correspond to higher-order bits (bits 6 and 7, for example) will have higher priority than banks corresponding to lower-order bits (bits 0 and 1). When two banks are selected that shadow each other, the bank with the higher priority is present in the area. Here is an example:

Data Byte Output to Port 0

7	6	5	4	3	2	1	0	Bits	
1	1	0	0	0	0	0	1		
8	7	6	5	4	3	2	1	0	Banks

In the example above, Banks 0, 1, 7, and 8 are enabled. Bank 8 has the highest priority, so its 16K of ROM and RAM will overlay the CP/M and user areas of Bank 1, and part of the system area of Bank 0. Likewise, the 8K of video memory of Bank 7 will overlay the corresponding locations of Banks 0 and 1, leaving a window into Bank 1 from 4000 to BFFF available for user programs. The following illustration shows what portions of memory would be enabled:

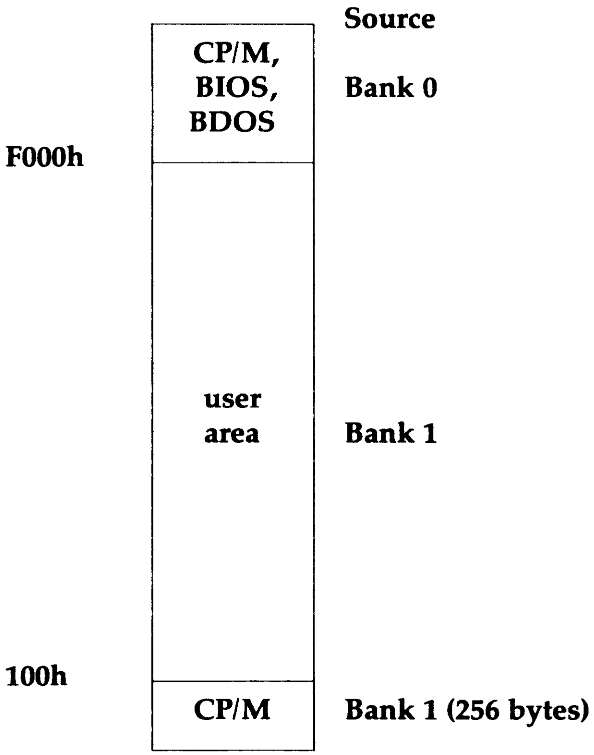
Effective Memory Area (using data byte shown)

		Source
	CP/M, BIOS, BDOS	Bank 0
E000h		
	4K (by 4) attrib	Bank 7
D000h		
	4K (by 8) video	Bank 7
C000h		
	user area	Bank 1
4000h		
	unused	Bank 8
2800h		
	RAM	Bank 8
2000h		
	ROM	Bank 8

As another example, suppose we output the following data by to port 0.

7	6	5	4	3	2	1	0	Bits	
0	0	0	0	0	0	0	1		
8	7	6	5	4	3	2	1	0	Banks

With this byte, we will have enabled Bank 1; and since Bank 0 is always enabled, the effective memory area will look like this:



The previous example shows the memory that is enabled during normal use of the CP/M operating system.

Character Generation

A character set is a collection of characters in a particular style. The Executive is capable of storing two character sets of up to 128 characters each. The default characters are Standard English (ASCII) and reside in the lower 2K of Bank 8. However, any set of characters can be defined through the CHARGEN program described in Chapter 2.

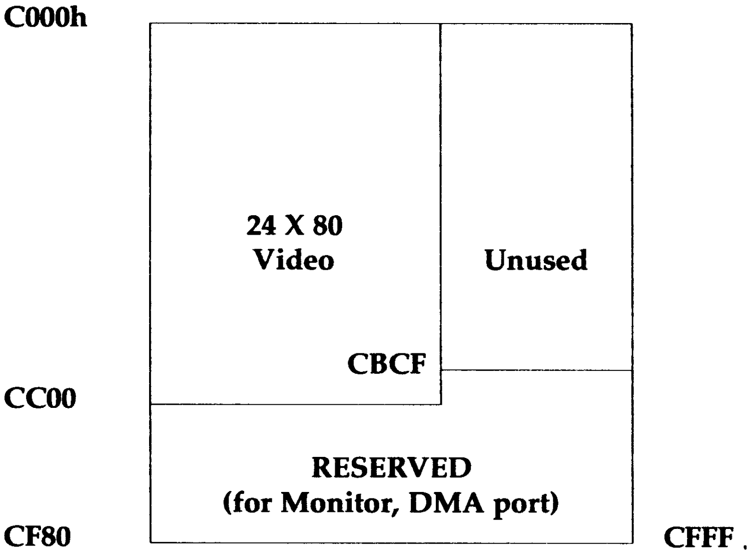
The alternate character set can be enabled for a particular character by toggling a bit in video RAM in Bank 7. As we mentioned earlier, reading and writing to this area involves two types of memory: reading accesses the ROM, writing accesses the character-definition RAM. Each character set is selected by the "alternate character set bit" (bit 4 of the attribute RAM in Bank 7); a 0 in this bit directs the system to use the character set stored in the lower 2K; a 1 in this bit directs the system to use the alternate character set in the upper 2K.

Memory-Mapped Video

Video memory is a 128-by-32 matrix, though only a portion is used as display. The actual display is an 80-by-24 matrix; the remainder is reserved for system use.

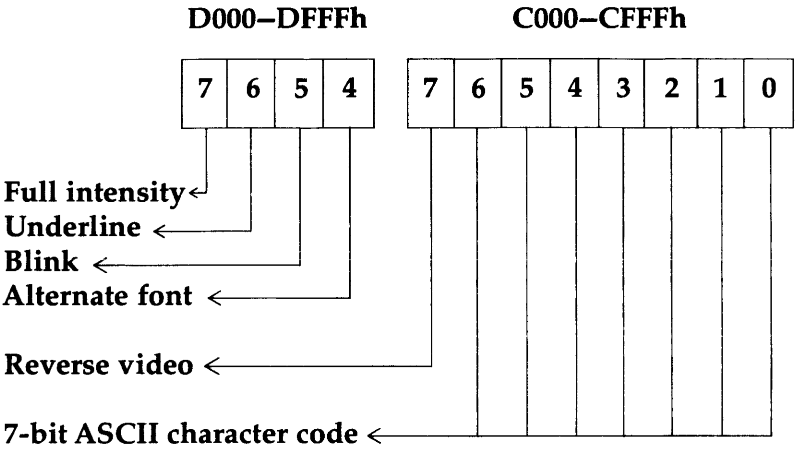
Memory controlling the video display resides in Bank 7. The particular characters used in the display are governed by the character font, which can be written to Bank 8, at addresses 0000–0FFFh.

The layout of the video portion of Bank 7 is as follows:



Note: In the above diagram, location C000 corresponds to the upper left corner of the screen; location CBCF corresponds to the lower right corner.

Twelve bits are associated with each character shown on the video screen; 7 of these are the ASCII character code, and the remaining 5 are attribute bits. Of these 5 attribute bits, one is stored in bit 7 of the video RAM (from C000 through CFFFh), and the remaining 4 bits are stored in the attribute area (from D000 through DFFFh), as shown in the following diagram:



The full-intensity bit is enabled for full or dim character display. A 1 means that the user will see a character of full intensity; a 0 means that the character will be half intensity.

If the underline bit is 1, then the character will be underlined; if it is 0, the character will appear without the underline.

The blink bit is used to continuously turn ON and OFF the display of a character. Blinking is enabled if the bit contains a 1, and disabled by a 0. A character will blink about twice a second.

The alternate-font bit is used to determine which 2K of the character-set RAM should be accessed. If this bit is 0, the character set in the lower 2K is used; if this bit is 1, the character set in the upper 2K is used. Note that a 256-character set can occupy that 4K of RAM (which might be the case for a language other than English); use of the alternate-font bit can be used as an index when addressing these two areas of memory.

If reverse video is enabled, characters on the screen will appear to be dark against a light rectangular background. A 1 in this bit enables reverse video; a 0 means that the character will appear as a light character against a dark background. If blinking is enabled along with reverse video, the character so identified will switch between normal and reverse representation.

A proprietary feature of the Executive allows 12-bit block moves. When moving data in the video memory (C000–CFFF) with LDIR and LDDR instructions, both the character and attribute bits are moved as a unit. Moves outside the video RAM area are accomplished in 8-bit segments.

The Executive utilizes software scrolling. This feature means that the top line of the screen display will always be at address C000h. (The next line starts at C080h; the 24th line—the bottom one—starts at CB80h.) Each line occupies 128 addresses. Lines 25 through 32 (decimal) of the display are never shown on the screen, though allocated in video RAM; neither are columns 81 through 128 of each line.

The normal method of enabling the video bank for a user-defined video driver is shown below:

```
IN     0          ;Get current banks enabled
STA    TEMP       ;Save in temp location
OR     A,40H      ;Enable video bank
OUT    0          ;
.
.                 ;User video driver here
.
LDA    TEMP       ;Restore banks
OUT    0
```

I/O DEVICES

The Osborne Executive is designed to maintain communication between most popular devices through the connectors mounted on the front panel.

- Printer port
- Modem port
- IEEE-488 port
- Keyboard port
- External video port
- Composite video connector

The serial-modem port A (DCE) and printer port B (DTE) are both designed around the RS-232C standard, both using DB25S connectors. Serial data is transmitted in 8-bit segments using 1 stop bit, 1 start bit, and no parity. The only difference between these two connectors is in the way the signals are routed. Any RS-232-compatible device can communicate through these connectors if the pinouts are arranged correctly and the signals are compatible.

The parallel IEEE-488 port is most commonly used to interface with a parallel printer, but any compatible device disk drive, teletype, scientific equipment, etc.) can be connected. Again, the only criteria is that the pinouts must be checked for compatibility against the pinouts listed for those ports. Note that not all manufacturers fully comply with the RS-232C and IEEE-488 standards; the device may need further modifications to be used with the Executive.

The keyboard port is intended primarily for the Osborne-supplied keyboard. However, optional keyboards may be developed by independent manufacturers for use with the Executive.

The composite video connector is an RCA phono jack that provides RS-470-compatible signals. This connector is the preferred method of attaching an external monitor.

The external video connector is also used for connecting an external monitor. Power must be OFF before removing or connecting the jumper plug that is installed here. Do not lose the jumper plug; the built-in monitor will not operate without it.

WARNING

Turning the power switch ON with nothing in the video port may cause damage to the Osborne Executive.

Port Selection

Peripheral devices are selected by an I/O Decoder inside the computer that determines which device is allowed bus access. Only one device can gain access at a given time. Devices directly controlled by the decoder are:

Disk Controller
RS-232-C interface
IEEE-488
Keyboard
Real-time Clock
System Configuration

The I/O Decoder, via the Peripheral Interface Adapter (PIA), controls the following functions:

Selection of disk drive A or B
Priority control of the internal memory
RS-232 RI and DSR lines
TX and RX clock select (internal/external)
Double-density select
Speaker
50/60 Hz monitor select

In addition to controlling peripheral device selection, the system configuration PIA also provides clock selection, which determines whether the peripheral devices will be controlled by the computer's internal, programmable clock generator or by the peripheral device. The port assignments are as follows:

PORT RANGE	DESCRIPTION
00–03	6821 system ports (PIA)
04–07	8253 timer (baud rates)
08–0B	1797 disk controller
0C–0F	Z80-SIO/2 serial interface
10–13	6821 parallel-port interface (488)
14	keyboard
18–1B	real-time clock overflow
1C–1F	video enable

Note: Only bits 4 and 7 are used for I/O address decoding.

Serial RS-232 Pinouts

PRINTER (SERIAL PORT) DB25S CONNECTOR (DTE)

Pin #	Signal	Direction
1	Chassis Ground	NA
2	Transmit Data (TXD)	in
3	Receive Data (RXD)	out
4	Request to Send (RTS)	in
5	Clear to Send (CTS)	out
6	Data Set Ready (DSR)	out
7	Signal Ground	NA
8	Data Carrier Detect (DCD)	out
20	Data Terminal Ready (DTR)	in

MODEM (SERIAL PORT) DB25S CONNECTOR (DCE)

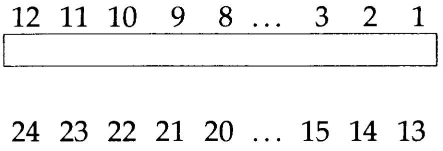
Pin #	Signal	Direction
1	Chassis Ground	NA
2	Transmit Data (TXD)	out
3	Receive Data (RXD)	in
4	Request to Send (RTS)	out
5	Clear to Send (CTS)	in
6	Data Set Ready (DSR)	in
7	Signal Ground	NA
8	Data Carrier Detect (DCD)	in
11	+12V	out
10	Modem Control Bit (MCB)	out
15	Transmit Clock (TXCLK)	in
17	Receive Clock (RXCLK)	in
20	Data Terminal Ready (DTR)	out
22	Modem Ring Indicator (RI)	in
24	Transmit Clock (TXCLK)	out

The IEEE-488/Centronics Interface

Any IEEE-488 compatible device can connect to the Osborne Executive. The following table shows the pin assignments for the IEEE-488 edge connector.

Pin #	SIGNAL	
1	Data bit 1	DIO1
2	Data bit 2	DIO2
3	Data bit 3	DIO3
4	Data bit 4	DIO4
5	End or Identify	EOI
6	Data Valid	DAV
7	Not Read for Data	NRFD
8	No Data Accepted	NDAC
9	Interface Clear	IFC
10	Service Request	SRQ
11	Attention	ATN
12	Cable Shield + GND	SHIELD
13	Data bit 5	DIO5
14	Data bit 6	DIO6
15	Data bit 7	DIO7
16	Data bit 8	DIO8
17	Remote Enable	REN
18	Signal Ground for DAV	
19	Signal Ground for NRFD	
20	Signal Ground for NDAC	
21	Signal Ground for IFC	
22	Signal Ground for SRQ	
23	Signal Ground for ATN	
24	Signal Ground for Logic	

The pin connections on the Osborne Executive are counted from the top right to the lower left.



Hewlett-Packard computers and Commodore's PET and CBM use the IEEE-488 interface-to-interface peripherals. Units designed for these computers will work with the Executive. The user should purchase a cable with the pin connectors indicated above, then run SETUP to configure the system properly. Though the Setup program can be used to select the appropriate device, the IEEE-488 device address in BIOS is currently set to 4 and can be changed by modifying location FE00h.

The Osborne Executive's system software includes a simplified way to work with the IEEE-488 port. The following routines must be employed in an intelligent sequence to use the IEEE interface properly. These commands are:

Control out
Status in
Go to standby
Take control
Output interface message
Output device message
Input device message
Input parallel-poll message

IEEE-488 commands use no RAM other than the CP/M stack. Each command routine in ROM determines the status of the port by reading the status of the 6821 PIA chip. The PIA transmits signals in both directions. Therefore, to reduce the overhead in determining the current direction the PIA is attempting to communicate, it is always left in one of two modes: source handshake mode, or accepter handshake mode. (The PIA specification sheet will be helpful in determining these modes.)

Several of the IEEE commands require the PIA to be in source handshake mode when called. The PIA is normally in source handshake mode following the completion of any IEEE bus information transfer, so this is not a major restriction. For instance, both the Status-In and the Parallel-Poll commands require the PIA to be in source mode, which means that you can perform the detection-of-device requests using either serial poll or parallel poll only when the interface is idle.

If you want to send or receive data on the IEEE bus, two steps must be performed. First, the bus must be configured by using the Output Interface Message. Then data is sent or received on the bus, using the Ouput Device Message or the Input Device Message.

To send data to a device on the IEEE bus, the controller makes the device a LISTENER, assumes the role of TALKER, and sends the data. To receive data from an external device, the controller must first make the device a TALKER and then assume the role of LISTENER. After this, the controller goes on standby and allows the CPU and the device to communicate at their own rate.

The controller can regain control asynchronously by setting the ATN sign to true. But if a device-dependent message is true when ATN becomes true, other devices on the IEEE bus can misinterpret the interrupted byte as an interface message and cause problems. Avoid this problem by taking control synchronously. If high-speed transfer of data between devices is not required and the computer can be tied up during the transfer, it is better to make the controller listen to the transfer while discarding the data. This procedure allows the controller to count transfers, look for EOI signals, or timeout the TALKER before regaining control.

The Centronics Interface

To connect a Centronics printer to your Osborne Executive, you must obtain a cable with the following connections:

IEEE Edge Connector		**Centronics Connector**
pin 1	data 0	2
2	data 1	3
3	data 2	4
4	data 3	5
5	data 4	6
6	data 5	7
7	data 6	8
8	data 7	9
11	out strobe	1
12	ground	19
15	busy	11
17	input ready	14 (not necessary on Epson)
19	in strobe	13

Note: Refer to the earlier diagram showing the pin numbering on the Osborne Executive IEEE-488 port.

Generally speaking, the above signal definitions apply to most parallel printers, although the connector on the printer end may be entirely different. One area that has provided confusion is the Centronics protocol definition. A number of printer manufacturers, Centronics included, do not use the full set of ground hookups defined, and in some cases, printer manufacturers who claim Centronics compatibility do not always attain it. If your dealer can't help you in connecting a printer to your Executive, write to Osborne Computer Corp., Customer Service Dept., 26538 Danti Court, Hayward, Calif. 94545, with a full description of the printer you are trying to connect, and we will attempt to help you get it working.

The Keyboard

The keyboard port is a read-only port. It is tested with an IN instruction for port 14, which looks at an 8-by-8 matrix describing the keyboard. The keyboard matrix is shown below.

COLUMN

ROW

	PIN #	10	11	12	13	14	15	16	17
	COLUMN #	0	1	2	3	4	5	6	7
3	0	ESC	TAB	CTRL	*	SHIFT	RETURN	∵	[ ]
8	1	! 1	@ 2	# 3	$ 4	% 5	^ 6	& 7	* 8
6	2	Q	W	E	R	T	Y	U	I
4	3	A	S	D	F	BELL G	H	J	K
2	4	Z	X	C	V	B	N	M	< ,
7	5	⇧	⇦	) 0	SPACE BAR	> .	P	O	9
5	6	⇨	⇩	– -	? /	: ;	| \	L	+ =
9	7	*	*	*	CAPS LOCK	*	*	*	*

PIN # ROW #

The 8 bits of Register A are set to test any of the eight rows in the matrix.

Register A

0	0	0	0	1	0	0	0	
7	6	5	4	3	2	1	0	Bits

In the previous example, row 3 is flagged for testing. If Register A is returned with the byte shown below when the IN instruction is performed,

Register A

1	1	1	1	1	0	1	1	
7	6	5	4	3	2	1	0	Bits

the decode tables in Bank 8 will interpret the keys pressed to be D. A 1 implies the key was not pressed, and a 0 implies the key was pressed.

Addressing the video screen has already been discussed in an earlier section of this chapter.

Console Commands

Cursor positioning is user-programmable. This means that if you want to put the cursor in column x of line y, send the following command sequence to the console:

<ESC> = <row y> <column x>

Here, <row y> is the number of the row where the cursor is to appear, and <column x> is the column number. Both numbers should be offset by 32. In BASIC, the sequence could be coded thus:

```
10 PRINT CHR$(27)+"="+CHR$(rowy+32)+CHR$(colx+32);
```

Alternatively, to place the cursor in a desired location, an assembly-language routine such as the following would reposition the screen.

```
         BDOS    EQU   0005h
         ESC     EQU   27
         PLACE   EQU   yyxx      ;you fill in
SETCUR:  MVI     E,ESC
         CALL    OUTCH           ;send ESCAPE
         MVI     E,'='
         CALL    OUTCH           ;send SETCUR
         LXI     H,PLACE
         LXI     D,2020h         ;bias for YX
         DAD     D               ;add bias to position
         PUSH    H               ;save column s
         MOV     E,H             ;prepare row y
         CALL    OUTCH           ;send row y
         POP     D               ;prepare column x
         CALL    OUTCH 'RET      ;send column x and
                                 return
OUTCH:   MVI     C,6             ;CP/M write to
                                 console
                                 ;function
         CALL    BDOS            ;call CP/M
         RET
```

Other video attributes can be controlled by issuing the appropriate sequence of characters. The Executive Monitor is patterned after the Televideo 950, and so uses many of the same screen control characters as listed below.

Keyboard Sequence	Decimal Sequence (BASIC)	Hex Sequence (Assembly)	Action
CTRL G	07	07	Rings the bell
CTRL H	08	08	Cursor left, no erasure
CTRL I	09	09	TAB
CTRL J	10	0A	Line feed
CTRL K	11	0B	Cursor up
CTRL L	12	0C	Cursor right
CTRL V	22	16	Cursor down
CTRL Z	26	1A	Clear screen
CTRL ^	30	1E	Home cursor
CTRL _	31	1F	New line
CTRL TAB			Toggle alternate char set
ESC)	27 41	1B 29	Start dim mode
ESC (	27 40	1B 28	End dim mode
ESC Z	27 90	1B 5A	Clear screen
ESC "	27 34	1B 22	Unlock keyboard
ESC #	27 35	1B 23	Lock keyboard

ESC =p1,p2	27 61	1B 3D	Load cursor row, column:
			p1 = ypos + 32
			p2 = xpos + 32

ESC Q	27 81	1B 51	Char insert
ESC W	27 87	1B 57	Char delete
ESC E	27 69	1B 45	Line insert
ESC R	27 82	1B 52	Line delete
ESC T	27 84	1B 54	Clear to end of line
ESC Y	27 89	1B 59	Clear to end of page

Keyboard Sequence	Decimal Sequence (BASIC)	Hex Sequence (Assembly)	Action
ESC ^	27 94	1B 5E	Start blink
ESC q	27 113	1B 71	End blink
ESC j	27 98	1B 62	Start inverse
ESC k	27 107	1B 6B	End inverse
ESC l	27 108	1B 6C	Start underline
ESC m	27 109	1B 6D	End underline
ESC <	27 60	1B 3C	Key Click ON
ESC >	27 62	1B 3E	Key Click OFF
ESC a	27 97	1B 61	Start Alternate Char
ESC A	27 65	1B 41	End Alternate Char
ESC e	27 101	1B 65	Key Translate Enable
ESC f	27 102	1B 66	Key Translate Disable
ESC z 'b'	27 122	1B 7A	Define window where b=window number (ASCII):
	27 119 48	1B 77 30	0
	27 119 49	1B 77 31	1
	27 119 50	1B 77 32	2
	27 119 51	1B 77 33	3
	27 119 52	1B 77 34	4
	27 119 53	1B 77 35	5
	27 119 54	1B 77 36	6
	27 119 55	1B 77 37	7
	27 119 56	1B 77 38	8
	27 119 57	1B 77 39	9
			followed by beginning row and column, then ending row and column numbers with decimal offset (usually offset=32).

Keyboard Sequence	Decimal Sequence (BASIC)	Hex Sequence (Assembly)	Action
ESC s 'b'	27 115 b	1B 77 b	Set window where b = the same values as for "define window" above
ESC .#	27 46	1B 2E	Define cursor type where: #=0 invisible #=1 blinking block #=2 steady block #=3 blinking underline #=4 steady underline
ESC o	27 111	1B 6F	Set X/Y offset
ESC b	27 98	1B 62	Set reverse video
ESC d			Return normal video
ESC x p	27 120	1B 78	Define background where p = ASCII char:
	27 71 48	1B 47 30	0 for normal
	27 71 49	1B 47 31	1 for dim
	27 71 50	1B 47 32	2 for blink
	27 71 51	1B 47 33	3 for dim blink
	27 71 52	1B 47 34	4 for reverse
	27 71 53	1B 47 35	5 for reverse dim
	27 71 54	1B 47 36	6 for reverse blink
	27 71 55	1B 47 37	7 for reverse dim blink
	27 71 56	1B 47 38	8 for underline
	27 71 57	1B 47 39	9 for underline dim
	27 71 58	1B 47 3A	: for underline blink
	27 71 59	1B 47 3B	; for underline dim blink
	27 71 60	1B 47 3C	< for underline reverse

Keyboard Sequence	Decimal Sequence (BASIC)	Hex Sequence (Assembly)	Action
	27 71 61	1B 47 3D	= underline reverse dim
	27 71 62	1B 47 3E	> underline reverse blink
	27 71 63	1B 47 3F	? underline reverse dim blink
ESC g			Graphics ON
ESC G			Graphics OFF

To clear the screen in a BASIC program, use the following "PRINT" statement.

```
PRINT CHR$(&H1A)
```

In assembly language, place a 1Ah in the E register and use CALL OUTCH to clear the screen.

Direct Screen Manipulation

We mentioned earlier that the video display is mapped using RAM memory beginning at C000h. The video memory is actually two paired sections of Bank 7 that allows each character to have 7 character bits and 5 attribute bits.

Remember that each character is actually considered to be 12 bits wide; 8 bits, consisting of 7 ASCII character bits and one attribute bit, are stored in the video RAM area of Bank 7 (from C000 through CFFFh), while the remaining 4 attribute bits are stored in the attribute area of the same bank (from D000 through DFFFh).

By directly manipulating the relevant bits for each character, individual memory locations can be changed without having to use the output sequences demonstrated earlier. To get to the attribute bits, use the assembly-language code shown below.

```
IN      0       ;get bank state
STA     TEMP    ;save in temp location
ORI     40h     ;shadow in video memory
OUT     0
```

Now read the memory location you wish to change, and set or clear the bits you wish to change. Note that 1=on and 0=off for all attributes.

```
LDA     TEMP    ;shadow out video
OUT     0
```

The nice thing about this method of addressing the video attributes is that instead of having to write to each individual location that should be affected by the particular attribute(s) a "template" can be constructed of the screen areas that are to be changed. Once set, these areas will remain so affected (e.g., blinking, etc.) as new characters are written into them. However, these areas should be reset to default values before you leave this template program; or you will have to press RESET to get the default attributes.

Configurable Tables and Pointers

The following section describes four tables and pointers that you may change if you have user-specific keyboard and console definitions.

The keyboard translation table starts at address 2015h in Bank 8. It includes the key code table, the shift table, the control key table, the Alpha lock table, and the control shift table.

Keyboard Table

(starting location = KYCDTB)

KYCD-TB		ESC	TAB	ignored	ignored
	+ 4	ignored	<cr>	’	[
	+ 8	1	2	3	4
	+ C	5	6	7	8
	+ 10h	q	w	e	r
	+ 14	t	y	u	i
	+ 18	a	s	d	f
	+ 1C	g	h	j	k
	+ 20	z	x	c	v
	+ 24	b	n	m	,
	+ 28	8Ah	8Dh	0	<space>
	+ 2C	.	p	o	9
	+ 30	8Bh	8Ch	_	/
	+ 34	;	\	l	=
	+ 38	ignored	ignored	ignored	ignored
	+ 3C	<cr>	ignored	ignored	ignored

Shift Key Table

(used when shift key is down)

SHFT-TB	+ 40	ESC	TAB	ignored	ignored
	+ 44	ignored	<cr>	”	]
	+ 48	!	@	#	$
	+ 4C	%	^	&	*
	+ 50	Q	W	E	R
	+ 54	T	Y	U	I
	+ 58	A	S	D	F
	+ 5C	G	H	J	K
	+ 60	Z	X	C	V
	+ 64	B	N	M	<
	+ 68	8Ah	8Dh	)	<space>
	+ 6C	>	P	O	(
	+ 70	8Bh	8Ch	_	?
	+ 74	:	¦	L	+
	+ 78	ignored	ignored	ignored	ignored
	+ 7C	<cr>	ignored	ignored	ignored

Control Key Table

(used when control key is down)

CT TBl	+ 80h	ESC	TAB	ignored	ignored
	+ 84	ignored	<cr>	ignored	^[
	+ 88	81h	82h	83h	84h
	+ 8C	85h	86h	87h	88h
	+ 90	^Q	^W	^E	^R
	+ 94	^T	^Y	^U	^I
	+ 98	^A	^S	^D	^F
	+ 9C	^G	^H	^J	^K
	+ A0	^Z	^X	^C	^V
	+ A4	^B	^N	^M	{
	+ A8	8Ah	7Fh	80h	Note: ^Back Arrow=DEL
	+ AC	}	^P	^10	89h
	+ B0	8Bh	8Ch	^_	~
	+ B4	ignored	^\	^L	`
	+ B8	ignored	ignored	ignored	ignored
	+ BC	<cr>	ignored	ignored	ignored

Alpha Lock Table

(used when Alpha Lock key is down)

AL-TB	+ C0	ESC	TAB	ignored	ignored
	+ C4	ignored	<cr>	’	[
	+ C8	1	2	3	4
	+ CC	5	6	7	8
	+ D0	Q	W	E	R
	+ D4	T	Y	U	I
	+ D8	A	S	D	F
	+ DC	G	H	J	K
	+ E0	Z	X	C	V
	+ E4	B	N	M	,
	+ E8	8Ah	8Dh	0	
	+ EC	.	P	O	9
	+ F0	8Bh	8Ch	_	/
	+ F4	;	\	L	=
	+ F8	ignored	ignored	ignored	ignored
	+ FC	<cr>	ignored	ignored	ignored

Control/Shift Table
(used when Control and Shift keys
are pressed simultaneously)

CS-TB	+ 100	ESC	TAB	ignored	ignored
	+ 104	ignored	<cr>	ignored	^]
	+ 108	81h	^@	ignored	84h
	+ 10C	ignored	^^	87h	88h
	+ 110	^Q	^W	ignored	^R
	+ 114	ignored	^Y	^U	^I
	+ 118	^A	^S	ignored	^F
	+ 11C	ignored	^H	^J	^K
	+ 120	^Z	^X	ignored	^V
	+ 124	ignored	^N	^M	{
	+ 128	8Ah	7Fh	ignored	<space>
	+ 12C	}	^P	^D	{
	+ 130	8Bh	8Ch	^_	7Fh
	+ 134	^_	^\	^L	'
	+ 138	ignored	ignored	ignored	ignored
	+ 13C	<cr>	ignored	ignored	ignored

Other control pointers are stored in the same area. These include CTBLFLAG, which determines whether the system uses ROM tables (=0) or application tables (=1); ROMTBLS, a pointer to the ROM structure; APPTBLS, a pointer to the application structure; CURPOS, which points to the current cursor position; and BACKGND, which is the background attribute flag (bit 7 + reverse video, bit 6 + full intensity, bit 5 = underline, and bit 4 = blink).

The function key translate pointers and tables are in the same area. The pointers include FKEYFLAG (=0 if the program is to use the ROM table, =1 if the application tables should be used, or =2 if neither is to be used); ROMFKEY and APPFKEY, which point, respectively, to the ROM and Application function key tables.

Interrupt Processing

Like the table pointers above, the interrupt routine uses a table to identify the source of the interrupt. The interrupt routine handler polls the devices to determine the source of the interrupt, and uses a table to decide where to go to service the interrupt. The table is stored in ROM, and read into location 23F3H of Bank 8 RAM. The Interrupt Table contains 3 bytes of information for each of the 14 possible devices; the first byte indicates which bank contains the appropriate servicing routine, and the next two indicate the address within that bank where the service routine can be found.

The Interrupt Table (INTTBL) contains service routine locations for the following devices (in order).

SIO TX channel B
SIO External/status channel B
SIO RX channel B
SIO RX channel B special condition
SIO TX channel A
SIO External/status channel A
SIO RX channel A
SIO RX channel A special condition
Real-time clock (tick) interrupt
Parallel entry
Intelligent keyboard
DMA entry
Floppy disk controller
System reset

Software Error Messages

This section provides a complete listing and explanation of CP/M Plus, Wordstar, SuperCalc, CBASIC, and MBASIC error messages.

ROM Informational and Error Messages

Following are the messages displayed by the Executive ROM:

```
PERFORMING SELF-TEST
```

This informational message is displayed while the power ON diagnostics are executing.

```
COMPUTER NOT WORKING PROPERLY
PLEASE CONTACT AN OSBORNE DEALER
PROCEED AT YOUR OWN RISK!
```

This message is displayed if the Executive fails any portion of the diagnostics test.

```
LOADING SYSTEM
```

This informational message is displayed while the boot process is in progress.

```
DISK IS NOT A VALID SYSTEM DISKETTE.
PRESS RETURN TO TRY AGAIN.
```

This message appears when there is not a valid operating system contained on the system tracks of the diskette being booted.

```
BOOT ERROR. PRESS RETURN TO TRY AGAIN.
```

This message is displayed if a hard disk error is encountered.

CP/M Plus Error Messages

Messages come from several different sources. CP/M Plus displays error messages when there are errors in calls to the Basic Disk Operating System (BDOS). CP/M Plus also displays messages when there are errors in command lines. Each utility supplied with CP/M Plus has its own set of messages. The following table lists CP/M messages and utility messages.

`ABORTED`

PIP. You stopped PIP operation by pressing a key.

`Assign a password to this file`

SET. Password protection is enabled for this file but no password has been assigned.

`BAD DELIMITER`

SET. Check command line for typing errors.

`Bad Load`

CCP error message, or SAVE error message.

`Baud rate cannot be set for this device`

DEVICE. Only physical devices that have the "SOFT-BAUD" attribute may have their baud rates changed. To check the attributes of the physical device, type "DEVICE physical."

`Bdos Err On:`

CP/M replaces : with the drive specification of the drive where the error occurred. This message appears when CP/M finds no disk in the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and try again.

Bdos Err On: Bad Sector

This message appears when CP/M finds no disk in the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and try again. This could also indicate a hardware problem or a worn or improperly formatted disk. Press C to terminate the program and return to CP/M, or press the return key to ignore the error.

Bdos Err On: File

Bdos Err On: File R/O

You tried to erase, rename, or set file attributes on a Read-Only file. The file should first be set to Read-Write (RW) with the command "SET filespec RW."

Bdos Err On x: R/O

Drive has been assigned Read-Only status with a SETcommand. CP/M terminates the current program as soon as you press any key.

Bdos Err on x: Select

CP/M received a command line specifying a nonexistent drive, the disk in the drive is improperly formatted, or the disk is Read-Only. CP/M terminates the current program as soon as you press any key. Press RETURN or CTRL-C to recover.

Break "x" at c

ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.

#	Search failure. ED cannot find the string specified in an F, S, or N command.

? Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.

O The file specified in an R command cannot be found.

> Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.

E Command aborted. A keystroke at the console aborted command execution.

F Disk or directory full. This error is followed by either the disk- or directory-full message. Refer to recovery procedures listed under these messages.

Cannot close file

GENCOM. CP/M cannot close an output file. This usually occurs because the disk is write-protected.

CANNOT CLOSE DESTINATION FILE – { filespec }

PIP. An output file cannot be closed. You should take appropriate action after checking to see if the correct disk is in the drive and that the disk is not write-protected.

Cannot close, R/O

CANNOT CLOSE FILES

CP/M cannot write to the file. This usually occurs because the disk is write-protected.

SUBMIT. This error can occur during SUBMIT file processing. Check if the correct system disk is in the A drive and that the disk is not write-protected. The SUBMIT job can be restarted after rebooting CP/M.

Cannot delete file

GENCOM. CP/M cannot delete a file. Check to see if the .COM file is Read-Only or password-protected.

Cannot have both create and access time stamps

SET. CP/M Plus supports either create or access time stamps, but not both.

Cannot label a drive with a file referenced

SET. SET does not allow mixing of files and drives.

CANNOT READ

PIP. PIP cannot read the specified source. Reader may not be implemented.

Cannot set both RO and RW

SET. A file cannot be set to both Read-Only and Read-Write.

Cannot set both SYS and DIR

SET. A file cannot be set to both SYS and DIR.

CANNOT WRITE

PIP. The destination specified in the PIP command is illegal. You probably specified an input device as a destination.

Checksum error

PIP. A hex-record checksum error was encountered. The hex record that produced the error must be corrected, probably by recreating the hex file.

```
Comma/space needed
```

SHOW. The parser detected the concatenation of two drive options. For example: "d:e:".

```
Command Buffer Overflow
```

SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.

```
Command too long
```

SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.

```
CORRECT ERROR, TYPE RETURN OR CTRL-Z
```

PIP. A hex-record checksum was encountered during the transfer of a hex file. The hex file with the checksum error should be corrected, probably by recreating the hex file.

```
DESTINATION IS R/O, DELETE (Y/N)?
```

PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y, the destination file is deleted before the file copy is done.

```
Directory full
```

```
DIRECTORY FULL
```

ED. There is not enough directory space for the file being written to the destination disk. You can use the OXfilespec command to erase any unnecessary files on the disk without leaving the editor.

SUBMIT. There is not enough directory space on the temporary file drive to write the temporary file used for processing SUBMIT files. Use the SETDEF command to determine which drive is the temporary file drive. Use the ERASE command to erase unnecessary files or set the temporary file drive to a different drive and retry.

LIB-80, LINK-80. There is no directory space for the output or intermediate files. Use the ERASE command to remove unnecessary files.

GENCPM. There is no directory space for the CPM3.'SYS.

HEXCOM. There is no directory space for the output COM file.

Disk full

ED. There is not enough disk space for the output file. This error can occur on the W, E, H, or X commands. If it occurs with X command, you can repeat the command, prefixing the file name with a different drive.

DISK READ ERROR – { filespec }

PIP. The input disk file specified in a PIP command cannot be read properly. This is usually the result of an unexpected end-of-file. Correct the problem in your file.

DISK WRITE ERROR – { filespec }

PIP. A disk write operation cannot be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space and then execute PIP again.

SUBMIT. The SUBMIT program cannot write the $$$.SUB file to the disk. Erase some files, or select a new disk and try again.

```
End of line expected
```

DEVICE, GET, SETDEF, and PUT. The command typed does not have any further parameters. An end-of-line was expected. Any further characters on the line were ignored.

```
ERROR: Bad close
```

SAVE. An error occurred during the attempt to close the file, probably because the file is write-protected.

```
ERROR: BAD PARAMETER
```

PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly.

```
ERROR – Close operation failed
```

COPYSYS. There was a problem in closing the file at the end of the file copy operation.

```
* * ERROR: False password { password }
```

PATCH. The wrong file password was typed.

```
ERROR: Illegal filename
```

SAVE. There is an error in the filespec on the command line.

```
* * ERROR: Illegal patch number: { ## }
```

PATCH. The patch number is too large (greater than 32), or it is not a true number.

```
* * ERROR: Invalid serial file type: { typ }
```

PATCH. The file type entered is not valid for PATCH.

```
* * ERROR: No file: { filename.typ }
```

PATCH. The file was not found in the file search process.

```
ERROR: No source file on disk
```

COPYSYS. The file CPM3.SYS is not on the disk specified.

```
ERROR – No directory space
```

COPYSYS and SAVE. There is insufficient space in the directory track for another entry.

```
ERROR: No disk space
```

SAVE. While writing to the file, the disk became full.

```
Error on line nnn message
```

SUBMIT. The SUBMIT program displays its messages in the format shown above, where *nnn* represents the line number of the SUBMIT file. Refer to the message following the line number.

```
ERROR – Out of data space
```

COPYSYS. The destination drive ran out of space during the transfer of the CPM3.SYS file.

```
* * ERROR: PATCH requires CP/M Plus
```

PATCH. PATCH is being used under another operating system.

```
* * ERROR: Serial number does not match
```

PATCH. You are attempting to patch a non-CP/M Plus utility.

* * FALSE PASSWORD * *

DUMP. The password typed on the command line is not correct.

FILE ERROR

ED. Disk or directory is full, and ED cannot write anything more on the disk. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.

FILE EXISTS

You have asked CP/M to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.

REN. The new name specified is the name of the file that already exists. You cannot rename a file with the name of an existing file. If you want to replace an existing file with a newer version of the same file, either rename or erase the existing file, or use the PIP utility.

File exists, erase it

ED. The destination file name already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file.

* * FILE IS READ-ONLY * *

ED. The file specified in the command to invoke ED has the Read-Only attribute. ED can read the file so that the user can examine it, but ED cannot change a Read-Only file.

PUT. The file specified to receive the output is a Read-Only file. It cannot be written to.

File not found

FILE NOT FOUND – filespec

DUMP, ED, GENCOM, GET, PIP, SET. An input file that you have specified does not exist. Check that you have entered the correct drive specification or that you have the correct disk in the drive.

First submitted file must be a COM file.

GENCOM. A COM file is expected as the first file in the command tail. The only time GENCOM does not expect to see a COM file in the first position of the command tail is when the NULL option is specified.

FIRST COMMON NOT LARGEST:

LINK-80. A subsequent COMMON declaration is larger than the first COMMON declaration for the indicated block. Check that the files being linked are in the proper order, or that the modules in a library are in the proper order.

HELP.DAT not on current drive.

HELP. HELP cannot find HELP.DAT file to process.

Illegal command tail.

DIR. The command line has an invalid format or option.

Illegal Format Value.

DIR. Only SIZE and FULL options can be used for display formats.

{ filename } ? File invalid.

SET. There is something wrong with the input file name and it could not be parsed.

Invalid Assignment

SET. You specified an invalid drive or file assignment, or misspelled a device name. This error message might be followed by a list of the valid file assignments that can follow a file name. If an invalid drive assignment was attempted, the message "Use: d:=RO" is displayed, showing the proper syntax for drive assignments.

Invalid control character

SUBMIT. The only valid control characters in the SUBMIT files of type SUB are ^A through ^Z. Note that in a SUBMIT file, the control character is represented by typing the circumflex, (^), not by pressing the CTRL key.

Invalid command

GETand PUT. The string or substring typed in the command line was not recognized as a valid command in the context used.

Invalid delimiter

DEVICE, SETDEF, GET, and PUT. The delimiter —([), (]), (,), ( ), or (=)—was not valid at the location used. For example, a [was used where an = should have been used.

INVALID DIGIT – { filespec }

PIP. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected, probably by recreating the hex file.

Invalid Disk Assignment

SET. Might appear if you follow the drive specification with anything except =RO.

INVALID DISK SELECT

CP/M received a command line specifying a nonexistent drive, or the disk in the drive is improperly formatted. CP/M terminates the current program as soon as you press any key.

Invalid drive

SETDEF. The specified drive was not a valid drive. Drives recognized by SETDEF are "*" (default drive) and "A" to "P."

INVALID DRIVE NAME (Use A, B, C, or D)

GENCPM. GENCPM recognizes only drives A, B, C and D as valid destinations for system generation.

Invalid File Indicator

SET. Appears if you do not specify RO, RW, DIR, or SYS.

Invalid file name

GENCOM and SUBMIT. The file name typed does not conform to the normal CP/M naming conventions for files.

Invalid file specification

GETand PUT. The file name typed does not conform to the normal CP/M naming conventions for files.

INVALID FORMAT

PIP. The format of your PIP command is illegal. See the description of the PIP command.

Invalid number

DEVICE. A number was expected but not found, or number was out of range (numbers must be from 0 to 255).

Invalid option

DEVICE and GET. An option was expected and the string found was not a device option or was not valid in the context used.

SETDEF. The option typed in the command line is not a valid option. Valid options are TEMPORARY, ORDER, PRL, and NOPRL.

Invalid option or modifier

PUT. The option typed is not a valid option.

Invalid physical device

DEVICE. A physical device name was expected. The name found in the command string does not correspond to any physical device name in the BIOS.

Invalid RSX type

GENCOM. File type must be "RSX".

INVALID SEPARATOR

PIP. You have placed an invalid character for a separator between two input file names.

Invalid type for ORDER option

SETDEF. The type specified in the command line was not COM or SUB.

INVALID USER NUMBER

PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

Label error

SET. SET failed to find any label for the drive.

More than four drives specified

SETDEF. More than four drives were specified for the drive search chain.

MULTIPLE DEFINITION:

LINK-80. The specified symbol is defined in more than one of the modules being linked.

***n*?**

USER. You specified a number greater than 15 for a user area number. For example, if you type USER 18, the screen displays 18?.

No directory label exists.

SHOW. The LABEL option was requested but the disk has no label.

No directory space

NO DIRECTORY SPACE – filespec

COPYSYS, GENCOM, MAC, PIP, RMAC, AND SAVE. There is not enough directory space for the output file. Use the ERASE command to remove unnecessary files on the disk and try again.

No more RSX files to be used

GENCOM. GENCOM has removed all the errant input files and none are left.

No SUB file found

SUBMIT. The SUB file typed in the command line cannot be found in the drive search process.

Not enough directory space

INITDIR. There are not enough remaining directory slots to allow for the time and date extension.

NO FILE – { filespec }

DIR, ERA, REN, PIP. CP/M cannot find the specified file, or no files exist.

NO INPUT FILE PRESENT ON DISK

DUMP. The file you requested does not exist.

No memory

There is not enough memory (buffer) available for loading the program specified.

No options specified

SET. SET needs to know what to do with the specified file(s) or drive(s).

* * NO RECORDS EXIST * *

DUMP. Only a directory space exists for the file.

NO SOURCE1 FILE

COMPARE. The first file specified cannot be found.

NO SOURCE2 FILE

COMPARE. The second file specified cannot be found.

NO SOURCE FILE ON DISK

GENCPM. Cannot find CP/M either in CPM3.SYS form or on the system tracks of the source disk.

NO SPACE

SAVE. Too many files are already on the disk, or no room is left on the disk to save the information.

No SUB file present

SUBMIT. For SUBMIT to operate properly, you must create a file with file type of SUB. The SUB file contains usual CP/M commands. Use one command per line.

NON-SYSTEM FILE(S) EXIST

DIRS. If non-system (DIR) files reside on the specified drive, DIRS displays this message.

NOT A CHARACTER SOURCE

PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.

* * NOT DELETED * *

PIP. PIP did not delete the file, which may have had the RO attribute.

NOT FOUND

PIP. PIP cannot find the specified file.

Parameter error

SUBMIT. Within the SUBMIT file of type SUB, valid parameters are $0 through $9.

PARAMETER ERROR. TYPE RETURN TO IGNORE

GENCPM. If you press return, GENCPM proceeds without processing the invalid parameter.

QUIT NOT FOUND

PIP. The string argument to a Q parameter was not found in your input file.

Read error

TYPE. An error occurred when reading the file specified in the type command. Check the disk and try again. The SET filespec command can diagnose trouble.

READER STOPPING

PIP. Reader Operation interrupted.

Record Too Long

PIP. PIP cannot process a record longer than 128 bytes.

Requires 3.0 or higher

SETand SHOW. This version of SETor SHOW must be run only under CP/M 3.0 or higher.

Requires CP/M 2.0 or newer for operation

PIP. This version of PIP requires the facilities of CP/M 2.0 or newer version.

START NOT FOUND

PIP. The string argument to an S parameter cannot be found in the source file.

SOURCE FILE INCOMPLETE

GENCPM. GENCPM cannot use your CP/M source file.

SYSTEM FILE(S) EXIST

DIR. If system (SYS) files reside on the specified drive, DIR displays this message.

"SYSTEM" FILE NOT ACCESSIBLE

You tried to access a file set to SYS with the SET command.

There are not enough available RSX slots

GENCOM. The number of input RSXs plus the number of used RSXs exceeds 15 slots.

This file was not used

GENCOM. In reference to the last error, GENCOM removes the errant file from the input list and proceeds.

*** * TOO MANY FILES * ***

SET. There is not enough memory for SET to sort the files specified, or more than 512 files were specified.

UNEXPECTED END OF HEX FILE – { filespec }

PIP. An end-of-file was encountered prior to a termination hex record. The hex file without a termination record should be corrected, probably by recreating the hex file.

Unrecognized Destination

PIP. Check command line for valid destination.

Unrecognized option

SHOW. An option typed in the command line is not valid for the SHOW command.

Use: SET d = RO

SET. An invalid SET drive command was given. The only valid drive assignment in SET is SET d:=RO.

VERIFY ERROR: – { filespec }

PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination disk or drive.

WRONG CP/M VERSION (REQUIRES 3.0)

You are trying to reference a version of the CP/M Operating System other than the one the Executive is configured for.

XSUB ACTIVE

SUBMIT. XSUB has been invoked.

XSUB ALREADY PRESENT

SUBMIT. XSUB is already active in memory.

Your input?

If CP/M cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command name correctly, or that the command you requested exists as a .COM file on the default or specified disk.

YOUR INPUT?

You did not specify how many pages of memory to save, or the file reference you specified makes no sense; you typed an ambiguous file reference.

BIOS ERROR MESSAGES:

In addition to the standard CP/M Plus error messages, there are others that address specific conditions inherent to our implementation of BIOS. Error messages pertaining to disk access and device selection have been included.

DISK ERROR MESSAGES:

The following disk error messages may occur if an error is encountered during a read or write operation to the disk drives. These messages will not be displayed if the @ERMDE flag (a function of the SCB) is set indicating that no errors are to be passed to the user. Also, since console interaction for disk error messages is via CONIN and CONOUT, these messages will not appear in a file if CONOUT is being PUT to a file. Here are the disk error messages:

```
BIOS error selecting drive X: not ready or not formatted.
```

This message occurs when an attempt is made to activate a drive containing a diskette with an unrecognizable format. Another message asks whether or not to try logging the drive again:

```
Retry this operation (Y or N)?
```

Type Y for yes if you want to retry or N for no; the word YES or NO will be displayed depending on your choice.

```
BIOS error reading drive X: ZZZZZ,
(code = AA/BB-CCCCCCCCCCCCCCCC)
```

This error message is displayed when an error is encountered during a disk read operation.

```
BIOS error writing drive X: ZZZZZ,
(code = AA/BB – CCCCCCCCCCCCCCCC)
```

This error message displays when an error occurs during a disk write operation.

The X in the above messages represents the drive identifier. The ZZZZZ represents one or more of the following messages:

```
not-ready
write-protected
fault
not-found
crc-error
lost-data
drq
busy
```

The dim numbers displayed in parentheses contain hex codes interpreted as follows:

AA — version of the floppy disk status (indicated by ZZZZZ).

BB — number of sectors attempted this transfer.

CC — contents of the internal table used by the ROM. The first byte corresponds to the memory bank, the second and third bytes are the DMA address, the forth byte represents the starting sector, the fifth and sixth bytes are the current track, the seventh byte is the internal unit number, and the last byte is the media flag.

Another message asks if the operation should be tried again:

```
Retry this operation (Y or N)?
```

If a retry is attempted, then the media change flag is queried to check if the diskette has been changed. When a media change is detected, the following message is displayed:

```
WARNING: data on disk inserted may be destroyed.
Continue (Y or N)?
```

The "WARNING" portion of the message will blink. If the retry is still attempted, then the media change flag is reset and the operation is retried. If the diskette has been changed, the directory will be destroyed.

DEVICE-NOT-READY ERROR MESSAGES:

The "device not ready" message occurs when an attempt is made to output data to a port assigned to a logical device where the physical device is not connected. The error message will not appear if the application program checks the status of devices before output; WordStar does this for the list output. Here is the message:

```
BIOS error device ______________ not ready,
unassign this device (Y or N)?
```

This message occurs when a device such as a printer is not attached to the assigned port (i.e., the LST: device). The name of the device is displayed.

If the device is available, connect it to the appropriate port and type N to begin output. If you type Y, the device is unassigned and must be reassigned using the SETUP program or the CP/M DEVICE utility.

WordStar Error Messages

```
@ @ @ @

File WSMSGS.OVR not found. Menus & messages will

display as @ @ @ @ only
```

Indicates that the message file cannot be located and most messages will appear as @@@@; under this condition the help level is automatically set to 0. Under some circumstances, you may not be able to continue editing.

EDIT FUNCTION ERROR MESSAGES:

```
* * * INTERRUPTED * * * Press ESCAPE key [ ]
```

This message occurs when a function is in progress and the interrupt command **^U** is typed.

```
* * * NOT FOUND: string * * * Press ESCAPE Key [ ]
```

This message indicates that a search initiated by a FIND (^QF), REPLACE (^QA), or FIND, REPLACE AGAIN (^L) command could not locate the specified string.

```
* * * ERROR E5: THAT MARKER NOT SET * * *

Press ESCAPE Key [ ]
```

Reference to a marker which you did not set during the current editing session.

```
* * * ERROR E6: BLOCK BEGINNING NOT MARKED

(OR MARKER IS UNDISPLAYED) * * *

Press ESCAPE Key [ ]
```

Reference to a beginning block marker that has either not been set or is currently hidden.

```
* * * ERROR E7: BLOCK END NOT MARKED
(OR MARKER IS UNDISPLAYED) * * *
Press ESCAPE Key [ ]
```

Reference to an end block marker that has either not been set or is currently hidden.

```
* * * ERROR E8: BLOCK END MARKER BEFORE BLOCK
BEGINNING MARKER * * * Press ESCAPE Key [ ]
```

The block-end marker has been set before the block-beginning marker.

```
* * * ERROR E9: BLOCK TOO LONG—MOVE OR COPY IN
TWO SMALLER BLOCKS * * * Press ESCAPE Key [ ]
```

The size of the text in the currently marked block exceeds the amount that WordStar can handle. Divide the block and move it in portions. There is no limit to the size of blocks that can be written.

```
* * * ERROR E10: CURSOR NOT IN RANGE FOR
COLUMN MOVE/COPY * * * Press ESCAPE Key [ ]
```

A column block move or copy cannot be accomplished because the cursor lies in a negative print position or past column 240.

```
* * * ERROR E11: THAT FILE EXISTS ON DESTINATION
DISK, DELETE EXISTING FILE FIRST, OR USE A DIFFERENT
DISKETTE. * * * Press ESCAPE Key [ ]
```

This message is caused by an attempted file transfer to a diskette that already contains a file of the same name. If such a transfer were successful, the original file would be replaced.

```
* * * ERROR E12: DISK FULL * * *
Press ESCAPE Key [ ]
```

Indicates that the capacity of the diskette to which data is being sent has been reached. You should take preventive measures to ensure that this catastrophic error does not occur. If this error occurs while you are moving the cursor toward the beginning of a large file, try moving the cursor to the end of the file and then saving with ^KS. If the error occurred while you were saving a file, press ESCAPE and delete some files with ^KJ, or try writing a portion of the file to the A drive.

```
* * * ERROR E13: COLUMN READ/WRITE NOT
ALLOWED * * * Press ESCAPE Key [ ]
```

This message indicates that reading and writing of blocked columns between files is not currently implemented. You can accomplish this maneuver by reading or writing a regular block containing the column and then copying or moving the column and erasing the remainder.

NOTE

You should note internal errors 115, 116, 117, 118, 119, and 136, and if you can reproduce them, you should report them to Osborne Computer Corporation and MicroPro International.

Utility Error Messages

SETUP ERROR MESSAGES:

```
READ ERROR
Press ESC key
```

There is a problem reading or writing the system to or from the diskette in the specified drive. Press ESC to continue.

```
WRITE ERROR
Press ESC key
```

There is a problem saving the system to the diskette in the specified drive. Press ESC to continue.

```
NOT A SYSTEM DISKETTE
You can only set up diskettes with
initialized system tracks.
Press ESC key
```

An attempt has been made to write the system tracks to a diskette which does not contain the CP/M Plus operating system. Press ESC to continue.

```
MISMATCHED VERSION NUMBER
Destination version doesn't match source version.
Press ESC key
```

You are trying to save the system to a diskette with a different BIOS version number than the source. The distinction is made because I/O addresses may change from one version to the next. Press ESC to continue.

```
ERROR EXECUTIVE SETUP

This setup program is designed for the Osborne Executive.

Your machine is not a valid Executive.

Please use the setup program designed for your machine.

Press ESC key
```

You are trying to run the Executive SETUP program on a machine other than it was designed for (such as the Osborne 1). Press ESC to continue.

```
INVALID VERSION NUMBER

The system on your boot disk is incompatible with this

version of setup. To use this setup, boot again with an

up-to-date system diskette.

Press ESC key
```

You are trying to run the SETUP program using an earlier version of the operating system. You must start the computer using the latest version of CP/M Plus. Press the ESC key to continue.

```
INVALID VERSION NUMBER

Source is not up to date for this version of setup.

Press ESC key
```

Indicates that you are trying to read an outdated operating system from the diskette in the source drive. Press ESC to continue.

```
Setup has not been saved yet.

Are you sure that you want to quit (Y or N)
```

Appears when you try to exit from the SETUP program without saving the current settings. Press Y for yes, to confirm your intention of leaving the setup without saving the current settings. Press N for no, if you are not ready to leave the SETUP program.

COPY ERROR MESSAGES:

```
COPY requires CP/M 3.X or greater.
```

You have attempted to run the COPY program using an invalid CP/M operating system.

```
Fatal read error occurred. Make sure Source disk is in
place then try again.
Press ESC to continue.
```

Indicates that the copy process cannot be implemented because of a problem reading the source diskette. This message usually occurs when there is no diskette in the source drive. Press ESC to continue.

```
Read error occurred on track ____ of source diskette.
Continue (Y/N)
```

Displays when a hard error occurs while reading the source diskette. Press Y for yes, and the copy process continues starting with the next track. Press N for no to abort the copy process.

```
Copy completed with READ errors.
Destination disk may contain bad data.
Press ESC to continue.
```

Informs you that there were read errors encountered during the copy operation and that the destination diskette may contain corrupted or incomplete information. Press ESC to continue.

```
Write error occurred. Retry with a different
destination disk.
Press ESC to continue.
```

A write error occurred while saving to the destination diskette. In this case, the information on the destination diskette is known to be bad and the copy process is aborted. Press ESC to continue.

```
Diskette in Drive X contains information.
Is it OK to overwrite it (Y/N)?
```

The destination diskette for a copy or format operation contains information. You are given the choice of writing on top of the data (Y) or aborting the operation (N) and using another diskette.

```
Diskette in drive X may contain information.
Is it OK to overwrite it (Y/N)?
```

Information has been detected on the destination diskette. You may choose to write over the existing data or not.

```
COPY ABORTED.
Press RETURN to continue.
```

You have aborted the copy operation. You can abort the copy process at any time by pressing Q. Press RETURN to continue.

```
Source diskette is not formatted
with a standard Osborne format.
Press RETURN to continue.
```

The diskette that you are trying to copy is not in a format that the Executive can read. Press RETURN to continue.

```
FORMAT ERROR.
Press RETURN to continue.
```

Occurs when the Executive has a problem formatting the diskette in the specified drive and the format operation is aborted. Press RETURN to continue.

```
FORMAT ABORTED.
Press RETURN to continue.
```

The formatting process has been aborted. Formatting can be abandoned at any time by pressing Q. Press RETURN to continue.

MESSAGES FOR THE COPYSYS PROGRAM:

```
COPYSYS requires CP/M 3.X or greater.
```

You are attempting to run the COPYSYS program using the wrong version of the CP/M operating system.

```
Diskette in Drive X is not a system diskette.
System get aborted. Press RETURN to continue.
```

You are attempting to read the operating system from a diskette without a valid CP/M Plus system. Press RETURN to continue.

```
A disk error occurred while reading the system.
System get aborted. Press RETURN to continue.
```

An error was detected while reading the system from the source diskette. Press RETURN to continue.

```
A disk error occurred while writing the system
System save aborted. Press RETURN to continue.
```

Occurs when there is a problem writing the system to the destination diskette. Press RETURN to continue.

```
CPM3.SYS not found. System get aborted.
Press RETURN to continue.
```

Indicates that the companion system file CPM3.SYS is not on the diskette designated as source. This system file is a necessary part of the operating system. Press RETURN to continue.

```
Insufficient space on drive X for CPM3.SYS.
System save aborted. Press RETURN to continue.
```

There is not enough room on the destination diskette for the system file CPM3.SYS.

```
CPM3.SYS is already on drive X.
Do you want to overwrite it (Y/N)?
```

Indicates that the system file CPM3.SYS is already located on the destination diskette. You can replace it by pressing Y or leave it as it is by pressing N.

```
System disk must be OCC double-density format.
System save aborted. Press RETURN to continue.
```

Appears when the Executive detects that the source or destination diskettes are not a double-density format. Press RETURN to continue.

MESSAGES FOR CHARGEN:

```
File not found. Press any key to continue.
```

Displayed on a file read if the file cannot be found. Make sure you have the right disk in place. Pressing any key returns you to the main menu.

```
EOF in file read. Press any key to continue.
```

Displayed when an end-of-file condition occurs while reading the specified hex file before a whole character set has been read. Pressing any character returns you to the main menu. A whole character set consists of 128 lines of 16 data bytes each. The portion of the hex file actually read can still be edited or rewritten.

```
File already exists. OK to continue? (Y/N)
```

Displayed on a hex file if the specified file already exists. A Y(es) answer results in the old file being erased and the new one written. A N(o) answer results in the file write being aborted.

```
Full directory on file creation –
Press any character to continue.
```

Displayed on a hex file write if the file cannot be written because the directory is full. Pressing any character returns you to the main menu.

```
Drive must be A or B
```

Displayed when you enter a file name specifying a drive which is not A or B (e.g., C:F00.CHR).

```
File name cannot be blank
```

Displayed when you enter a blank file name (e.g., B:.CHR).

```
File type too long
```

Displayed when you enter a file type which is too long (i.e., more than 3 characters).

```
Blank file type
```

Displayed when you enter a file type which is blank (e.g., B:F00.).

```
File name too long
```

Displayed when you enter a file name which is too long (i.e., more than 8 characters).

```
X is invalid in a file name
```

Displayed when you enter a file name with an invalid character (e.g., a blank). X is replaced with the invalid character in the actual message displayed. Note: the file is still written with the invalid name.

```
X is invalid in a file type
```

Displayed when you enter a file type with an invalid character (e.g., a blank). X is replaced with the invalid character in the actual message displayed. Note: the file is still written with the invalid name.

```
No disk in drive or unformatted
```

Displayed when reading or writing the system tracks and there is no disk in the drive or it is unformatted.

```
Disk is wrong density
```

Displayed when reading or writing the system tracks and the disk in the drive is not OCC double-density format.

Hard write error

Displayed when writing to the system tracks and a write error occurs.

Hard read error

Displayed when reading from the system tracks and a read error occurs.

FILE NAME ERRORS:

filename.typ NOT FOUND

The file name you supplied for a FILE NAME? prompt cannot be located.

INVALID FILE NAME: string entered

The string you entered in response to a FILE NAME? prompt is invalid.

Can't edit a file type.BAK or. $ $ $ – rEname or cOpy the file before editing

This message occurs when you make an attempt to edit (D or N) a WordStar backup or temporary file. The file can be renamed or read into another file if your intent is to edit the file.

File WS.COM Not Found – Can't Run program unless WS.COM is available

The message means the main WordStar program (WS.COM or other if changed through installation) cannot be found.

File x:filename.typ ALREADY EXISTS

The name being assigned to a file through the RENAME command E already exists; choose a different name or rename the original.

FILE x:filename.typ NOT ON SAME DRIVE

This message tells you that a file cannot be renamed from one drive to another.

FILE x:filename.typ EXISTS–OVERWRITE? (Y/N): [ ]

This message appears when you try to copy a file to an already existing file name. If you want to copy over the original file, simply type Y for yes.

MailMerge Error and Warning Messages

MailMerge has several warning and error messages. An error message always appears when an invalid DOTcommand is encountered. Error messages also occur when a referenced file cannot be found, or when the contents of a data file do not correspond to the keywords listed by .RV in the document file.

MailMerge displays a warning message on encountering certain special conditions. Processing continues even though the warning messages are displayed. Errors indicated on the screen remain in the file being processed.

Some messages inform you of conditions that might not have adverse consequences but that you should nevertheless consider. Review all messages that accumulate on the screen and note any conditions that require action. Edit the file to correct any mistakes that you might have made while creating the file.

The following warning and error messages appear when problems are detected:

```
* * * Invalid DOT command ignored
```

A DOT command not used in its proper form or context will cause this error message. The DOT command in question is displayed on the following line. A more specific error message may accompany this message in some cases.

```
* * * Insert diskette with file (:) filename.typ
then press RETURN
```

Requests insertion of the diskette containing the named file into the indicated drive (shown after the message). This message results when you process a .DF or .FI command with the word CHANGE after the file name.

```
* * * Cannot change disk in drive (:), request ignored
```

Displayed when you have specified a diskette change for the drive holding the program files. To avoid this type of error, limit a diskette change to the B disk drive. An attempt to access the specified file will take place, just in case the file is on the specified drive (or the logged drive if you did not specify one). If the file cannot be found, then the following message is displayed:

```
* * * file (:) filename.typ not found
```

Indicates that the file name called for could not be found in the specified drive (or the logged drive if no drive was specified). Both drives are searched in order to locate the specified file. If the file cannot be found, processing will continue without it.

* * * but found, and will use, (:) filename.typ

When a file cannot be located on the specified drive, a search of the other drive occurs. The message above is displayed if a file with the specified name is found. This message tells you where the file was located, allowing you to determine if this is the file you intended to reference.

* * * No .DF before .RV

No data file with the name you specified in .DF could be found, or the .DF command itself could not be found. If a "file not found" message does not accompany this message, then check the file to make sure that a .DF was used and that it was placed before .RV. Printing of the document file will continue without data for the keywords.

* * * WARNING: Overlong data value truncated

Displayed when data items contain more than the maximum 200 allowable characters. Only the first 200 characters are used, the excess may be omitted or may be used as data for the next variable read by .RV. This message might indicate an error in the format of the data file.

* * * Invalid variable name in .RV command ignored

Means that one or more of the keywords identified in the .RV command were not in their valid form. Ampersands should not be used in the keywords listed by .RV.

* * * WARNING: data exhausted, null value(s) used

Displayed when data from the data file is exhausted before all the keywords listed in .RV have been read. The variable identifiers for which there is no data will be assigned a null value consisting of no characters. Printing will usually stop before the next copy is printed after this message is displayed.

This type of error usually occurs when the last record of a data file is being processed. The error condition might be present anywhere in the data file, even though the error is not detected until processing of the file is almost complete. This type of error is typically caused by a lack of proper commas and carriage returns. Sometimes the error is caused by the presence of the wrong data file.

NOTE

When a document file containing .DF or .RP is processed, MailMerge searches ahead after each printed document looking for data to be used in the next document. If no data item is encountered, then processing will terminate without the above message being printed.

MISCELLANEOUS ERRORS:

```
* * * ERROR E38 (-42): BAD OVERLAY FILE, OR
WRONG VERSION OVERLAY FILE * * *
Press ESCAPE Key [    ]
```

```
* * * ERROR E43 (44): WRONG VERSION OVERLAY
FILE * * *
```

Occurs when the wrong version of an overlay file (.OVR) is being used; sometimes caused by a damaged diskette.

```
E46: Overlay file WSOVLY.OVR Not found * * *
Press ESCAPE Key [ ]
```

The file named WSOVLY.OVR is missing from the current version of WordStar.

```
* * * ERROR E47: FILE MERGEPRN.OVR NOT FOUND
(The separately supplied file MERGEPRIN.OVR is required
for use of MailMerge) * * * Press ESCAPE Key [ ]
```

This message appears when a MailMerge operation is attempted and the MailMerge program file (MERGEPRIN.OVR) cannot be located on the currently logged drive.

```
* * * ERROR 52: PROGRAM IS AN EMPTY FILE!? * * *
Press ESCAPE Key [ ]
```

This message appears when an invalid program is referenced through the RUN-A-PROGRAM command, R.

```
* * * ERROR E53: PROGRAM TOO BIG FOR
MEMORY AVAILABLE UNDER WordStar * * *
Press ESCAPE Key [ ]
```

The program trying to be run is too large to be run through WordStar.

WARNINGS:

```
* * * WARNING: WORD TOO LONG TO FIT MARGINS
```

This warning appears when too many characters are strung together on a line.

CAN'T DISPLAY PAGE BREAKS IN A NON-DOCUMENT FILE

The Page-Break display command, ^OP, was issued and is not valid in the nondocument mode.

PUT AT FILE BEGINNING FOR CORRECT PAGE-BREAK DISPLAY

Indicates that the DOT commands being used should be located at the beginning of the file so that the page breaks can be determined accurately.

?

A lingering question mark appears in the rightmost flag column when you specify an erroneous or incomplete DOT command, a missing numeric argument, an unrecognizable code, or an excessive number.

*** * * WARNING: WRONG VERSION OF WSMSGS.OVR – SOME MESSAGES MAY BE INCORRECT * * ***

This warning is telling you that the version of the message file (WSMSGS.OVR) you are using is incompatible with the version of WS.COM being used.

*** * * WARNING: DISK FULL, DELETING OLD .BAK FILE TO MAKE SPACE (NORMALLY. THE PREVIOUS BACKUP FILE IS DELETED ONLY AFTER EDIT IS SUCCESSFULLY COMPLETED).**

This warning message informs you that the diskette being written to is becoming full. You should take action immediately to avoid serious complications.

```
WARNING: You are editing the same file as you are printing.
WordStar will not allow you to save the edited version until
the print has completed or has been abandoned.
```

Indicates that the file being printed cannot be simultaneously edited.

NOTE

Occasionally the Osborne 1 beeps and the screen fills with lines of exclamation points. This condition occurs when you are issuing more commands than the computer can handle.

PRINTING MESSAGES:

```
filename.typ NOT FOUND
```

The file named for printing could not be located on the logged or indicated drive.

```
INVALID FILE NAME: string
```

The string you entered in response to a print prompt is invalid.

```
WARNING: You are printing the same file as you are editing.
The last saved version will be printed, not reflecting unsaved
changes. Furthermore, WordStar will not allow you to save
the edited version while the print is in progress.
```

Indicates that the file referenced for printing is currently being edited and the backup version will be printed if it is available.

*** * * PRINT OUTPUT DISK FULL. PRINT PAUSED. * * ***

Occurs when the diskette onto which the print-output file is being written becomes full. Delete unneeded files when applicable.

INFORMATION MESSAGES:

FINISHING PRINT BEFORE EXIT (type ^ U to cancel exit command) . . .

Occurs when you try to exit to CP/M while a print operation is in progress.

FINISHING PRINT OF SAME FILE BEFORE SAVING (type ^ U to cancel Save command) . . .

Occurs when you try to save a file that is currently being printed.

FINISHING PRINT OF .BAK FILE BEFORE SAVING (type ^ U to cancel Save command) . . .

Occurs when you try to save a file you are editing while its backup version is printing.

NOTE

If a FATAL error occurs, call your authorized Osborne dealer.

SuperCalc Error Messages—Causes and Cures

The following material provides you with a detailed description of the error messages that you may receive while using the SuperCalc program. They are discussed in alphabetical order. For each error message, we have included a brief explanation of its cause and a procedure for correcting the situation that resulted in the message.

Here is a list of errors considered:

Column ERROR

Incorrect specification of a column. Correct specification is a letter from A to Z or two letters from AA to BK. To correct: use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

Disk FULL

The disk designated to receive the file does not have enough space. The SuperCalc program will ask if you want to redo the operation (Y) or not (N). If you want to redo it, remove the disk and insert another one that has enough space; then press **Y**. If you press **N**, the operation is aborted, and you return to the SuperCalc program.

Drive not ready

This is a system error message from the BIOS portion of your CP/M operating system. It is possible that the drive will become ready and that retrying will work. Check to make sure that the disk drive is closed.

File NOT on disk

This occurs with the load command. The file name given is not found on the disk drive specified or implied in the entry. Check your command entry.

1. Check the drive designation. If you did not specify one, the SuperCalc program assumes you mean the current default drive.
2. Check the spelling of the file name.
3. Check to see that the correct diskette is in the drive.

In cases 1 or 2, use the in-line editor to correct the drive designation or the file name and reenter the command.

In case 3, either place the correct disk in the drive or, if this is not feasible, cancel the commmand with **^Z**.

Formula ERROR

There are two possible causes.

1. You entered text without a leading ". SuperCalc assumes that you intended to enter a formula, and it cannot make sense out of the entry as a formula.
2. There is some error in the way you specified a formula. Check it for correct specification of function name, correct use of expressions, balanced parentheses, valid cell names, etc.

To correct: use the in-line editor to correct your entry and reenter, or cancel the entry with ^Z.

Memory FULL

Too much content in the worksheet. (This is a different case from Worksheet Full, described below, in which there are too many cell stubs on the worksheet.) To correct: blank any contents that you can spare. If you can, move material to the upper left of the worksheet, trying to preserve a roughly rectangular shape. Save the worksheet, ZAP the screen, and reload the worksheet.

If this does not free enough space, then you must break the worksheet into convenient portions for future work. To do this, ZAP the screen and reload selected portions of the saved worksheet. Build two or more worksheets out of these portions, saving them as separate worksheets.

Overlay ERROR

This is a serious error that prevents the SuperCalc program from being used. There are two possible causes:

> The SuperCalc program has (1) not been "installed" or (2) has been "installed" incorrectly. Installing the SuperCalc program means customizing it for your computer system. This customizing involves specifying the terminal that you are using, the disk drives available, the memory space available, and the version of the CP/M operating system that you use. SuperCalc comes correctly installed for the Osborne 1. See the Appendix for information on SuperCalc Installation procedures.

To correct:

1. If you are installing the SuperCalc program yourself, reinstall it, checking the installa-

tion documentation carefully as you proceed.

2. Consult your authorized Osborne 1 dealer for information and assistance.

Protected Entry

This message can appear as the result of an error, or it may appear as an informational note. If the message is the result of an error, it will appear during data entry or the edit command. You are attempting to enter data into an active cell that is protected. You must either remove the data from the entry line or cancel the edit command.

This message may appear as an informational note during a blank, copy, load, or replicate command. If there are protected cells in the area being blanked or in the destination area of the copy, load, or replicate command, the protected cells in the area remain unchanged; the other cells in the area have been changed. If you meant to leave the protected cells unchanged, all is well. If not, you may wish to unprotect them and redo the command.

Range ERROR

Incorrect specification of a range. A range may be a single cell, a partial column, or a partial row. To correct: use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

Row ERROR

Incorrect specification for a row. Correct specification is a number from 1 to 254. Use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

Replicate Definition ERROR

The destination may be specified incorrectly, or the destination area may be too small.

1. Specification error for the destination.
   a. If the source is a single cell, the destination should be specified as a partial column or partial row.
   b. If the source is a partial column, the destination should be specified as cells on the upper row of the destination. This will look like a partial row.
   c. If the source is a partial row, the destination should be specified as cells in the column on the left of the destination. This will look like a partial column.
2. Destination area is too small (will not fit).

Given the size of the source and the location of the destination, the result will not fit within the worksheet boundaries. Correct the specification using the in-line editor and reenter the command, or cancel the command with ^Z. (Note: SuperCalc caught the error before attempting to execute the command.)

Window Parameter ERROR

This occurs during the window command when you attempt to split the screen when the active cell is at the left or right edge or the top or bottom row of the display screen. Because of the way that the command works, the split cannot take place at the edges of the screen.

Either move the active cell away from the edge of the display window or scroll the screen to provide an additional column or row between the edge and the location you desire for the split.

Worksheet FULL

The worksheet is too large in size; there are too many cell stubs. (This is different from the case described above in Memory Full, where the worksheet has too much content.)

If you can, blank any unnecessary contents and move the other contents to the upper left, trying to preserve a roughly rectangular shape. Then save the worksheet, ZAP the screen, and reload.

"Memory Use—Hints and Concepts" in the SuperCalc reference section explains how it is possible to unintentionally create many more cell stubs than necessary. You may get a Worksheet Full message even though you have few contents and they are at the upper left. In such a case, saving the worksheet, ZAPping the screen, and reloading the worksheet will get rid of unnecessary cell stubs.

SUPERDATA INTERCHANGE ERROR AND WARNING MESSAGES:

If you enter an incorrect file name or select the wrong conversion option, the SDI program may not be able to carry out your instructions. Regardless of which conversion option you choose, the SDI program checks to see that:

1. The source file exists.
2. The source file structure is correct for the option selected.

3. The destination file exists. The SDI program creates a file if it does not exist or notifies the operator if it does exist.

The error messages provided if an error is detected are:

1. If the file cannot be found, this message is displayed:

```
AN ERROR HAS OCCURRED

THE ERROR IS:

SOURCE FILE NOT FOUND

PRESS RETURN TO CONTINUE
```

2. If the source file structure is not correct for the option selected, this message is displayed for options A and C:

```
AN ERROR HAS OCCURRED

THE ERROR IS:

FILE IS NOT A SUPERCALC FILE

PRESS RETURN TO CONTINUE
```

or for option B:

```
AN ERROR HAS OCCURRED

THE ERROR IS:

FILE IS NOT A COMMA SEPARATED VALUE FILE

PRESS RETURN TO CONTINUE
```

or for option D:

```
AN ERROR HAS OCCURRED

THE ERROR IS:

FILE IS NOT A SUPERDATA FORMAT FILE

PRESS RETURN TO CONTINUE
```

3. If the destination file already exists, this message is displayed:

   ```
   FILE ALREADY EXISTS!
   OKAY TO OVERWRITE (Y/N)?
   ```

   If a "Y" is entered, the new data will overwrite the existing data file. If an "N" is entered, the conversion process will be abandoned and you will be returned to the SDI menu.

ADDITIONAL ERROR MESSAGES AND THEIR CAUSES:

```
FILE IS EMPTY
```

The file was opened for conversion and found to contain no data at all (only a directory entry exists).

```
FILE NOT OPENED
```

This is really a warning issued when you select not to overwrite a file that already exists.

```
MULTIPLE ORIGINS IN A TUPLE
```

A "tuple" is the data between one BOT statement and another. The "cell origin" data item can only occur **after** a BOT, and another BOT **must** occur before another cell origin."

```
IMPROPERLY FORMED STRING IN ORIGIN HEADER
```

This usually means that the colon (:) is missing in a cell-origin data item.

NO ROW FOUND IN ORIGIN HEADER

This means that the first item only (column) was found in a cell-origin data item. The colon (:) was also found. The row specifier after the colon was absent.

WARNINGS ISSUED BY SUPERDATA INTERCHANGE:

These warnings are issued by the SDI program to inform you when an unusual action has taken place or when data is found that has not been correctly formed. In the latter case, usually no action is taken by the SDI program. In either case, the converted file may be damaged and/or inaccurate.

ROW OUT OF RANGE

COLUMN OUT OF RANGE

The row or column in a ROW-FORMAT or COL-FORMAT is out of the possible range. The range is (1–63) for columns and (1–254) for rows.

NULL CELL CANNOT HAVE FORMULA

The data item following a value of NULL **can** be a format string to format the cell. However, it **cannot** be a formula data item.

NO VALUES IN ORIGIN HEADER

The data value indicated a cell-origin data value. However, the cell origin string was null or empty (" ").

BAD INTEGER NUMBER

An integer value that was found in a file is bad. This value either contains characters other than 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9, or contains control characters. Also, the file itself may be damaged.

FORMAT ERROR

The string of characters in a format string used with COL-FORMAT, ROW-FORMAT, GDISP-FORMAT, or a format data item contains invalid characters. Spaces may have been inadvertently placed in the string. Valid characters are: TL, TR, R, L, $, *, G, D, E, and I.

CBASIC Error Messages

CBASIC COMPILER ERRORS:

The following compiler error messages can appear during compilation of a source file:

NO SOURCE FILE: < filename > .BAS-

The source file could not be found on the indicated drive.

OUT OF DISK SPACE

The compiler encountered insufficient disk space while writing the .INT or .LST file.

OUT OF DIRECTORY SPACE

The compiler ran out of directory entries while attempting to create or extend an .INT or .LST file.

BDOS ERROR ON (A,B)

This CP/M error message indicates that an error occurred while the computer was reading from or writing to a disk file.

PROGRAM CONTAINS *n* UNMATCHED FOR STATEMENT(S)

n FOR statements have no associated NEXT statements.

PROGRAM CONTAINS *n* UNMATCHED WHILE STATEMENT(S)

n WHILE statements have no associated WEND statements.

PROGRAM CONTAINS 1 UNMATCHED DEF STATEMENT

A multiple line function was not terminated with a FEND statement, possibly causing further errors.

WARNING INVALID CHARACTER IGNORED

An invalid character was detected in the last line, then was replaced by a question mark and ignored.

INCLUDE NESTING TOO DEEP NEAR LINE *n*

An INCLUDE statement exceeded the maximum nesting level near line *n*.

COMPILER ERROR CODES:

The following two-letter error codes display with the line number and position of the error:

BF

Invalid branch into a multiple line from outside of the function.

BN

Invalid numeric constant was encountered.

CI

Improper file name used in an %INCLUDE directive.

CS

The COMMON statement was not the first program statement preceded only by a directive, remark, or blank line.

CV

A subscripted variable in a COMMON statement was not properly defined.

DL

Duplication of the same line number, an undefined function, or a DIM statement that does not precede all referenced arrays, was detected.

DP

A DIM variable was previously defined in another DIM statement or was used as a simple variable.

FA

A function name not used in the function was encountered to the left of the equal sign in an assignment statement.

FD

A function name is the same in two DEF statements.

FE

An incorrect mixed-mode expression exists in a FOR statement, usually the expression following TO is involved.

FI

The FOR loop index is not an unsubscripted-numeric variable expression.

FN

An incorrect number of parameters are used in the function reference.

FP

The function-reference parameter type does not match that in the DEF statement.

FU

An undefined function has been referenced.

IE

The IF statement expression is erroneously evaluated as type string.

IF

The FILE statement variable is type numeric instead of type string.

IP

An input prompt string was not enclosed in quotes.

IS

A subscripted variable was not dimensioned before it was referenced.

IT

Indicates that an invalid compiler directive was issued.

IU

A DEF-statement defined array was not subscripted.

MC

A variable was defined more than once in a COMMON statement.

MF

The expression is evaluated as type string instead of type numeric.

MM

An invalid mixed mode was encountered, usually caused by a mixture of string and numeric types in an expression.

MS

A numeric instead of a string expression was used.

ND

A DEF statement could not be found for a corresponding FEND statement.

NI

A NEXT variable reference did not match that referenced by the associated FOR statement.

NU

A NEXTstatement occurred without an associated FOR statement.

OF

An illegal branch from within a line function was attempted.

OO

The ON statement limit of 40 was exceeded.

PM

A DEF statement was encountered within a multiple line function. Functions cannot be nested.

SE

A syntax error occurred in the source line, usually as the result of an improperly formed statement or misspelled keyword.

SF

A numeric instead of a string expression was used in a SAVEMEM statement. Check for quotes around string constants.

SN

An incorrect number of subscripts were found in a subscripted variable, or a DIM variable was previously used with a different number of dimensions.

SO

A statement that is too complex should be simplified in order to be compiled.

TO

Indicates a symbol table overflow, meaning the program is too large for the current Osborne 1 memory configuration.

UL

You have referenced a nonexistent line number.

US

A string was terminated with a carriage return, rather than quotes.

VO

Variable names are too long for one statement.

WE

The expression following the WHILE statement is not numeric.

WN

The nesting level of WHILE statements (12) has been exceeded.

WU

A WEND without an associated WHILE statement was encountered.

RUN-TIME ERRORS:

The following run-time error messages are displayed below the most recent screen line, to indicate conditions which usually terminate program execution.

NO INTERMEDIATE FILE

A file name of type .INT could not be located on the specified drive.

IMPROPER INPUT-REENTER

The fields you entered at the keyboard do not match those specified in the INPUT statement.

WARNING CODES:

Two-letter codes preceded by the word WARNING indicate errors that do not prevent execution of a program but should be attended to. These codes are:

DZ

A number divided by zero resulted in the largest CBASIC number.

FL

A field length greater than 255 bytes was encountered during a READ LINE; the remainder is ignored.

LN

A LOG function argument was zero or negative; the value of the argument is returned.

NE

A negative number before the raise to a power operator (^) was encountered, resulting in the absolute value of the parameter being calculated.

OF

A real-variable calculation produced an overflow. The result is set to the largest valid CBASIC real number. Overflow is not detected with integer arithmetic.

SQ

A negative number was specified in the SQR function. The absolute value is used.

RUN-TIME ERROR CODES:

The following two-letter codes are preceded by the word ERROR and cause execution to terminate:

AC

An ASC function string argument was evaluated as a null string.

BN

The BUFF value in either the OPEN or CREATE statement is less than 1 or greater than 52.

CC

The CHAINed program code area is greater than the calling program's code area. Use %CHAIN for adjustment.

CD

The CHAINed program data area is greater than the calling program's data area. Use %CHAIN for adjustment.

CE

The file being closed could not be found in the directory.

CF

The CHAINed program constant area is greater than the calling program's constant area. Use %CHAIN for adjustment.

CP

The CHAINed program variable storage area is greater than the calling program's variable storage area. Use %CHAIN.

CS

The CHAINed program reserved a different amount of memory with a SAVEMEM statement than the calling program.

CU

An inactive file number was specified in the CLOSE statement.

DF

An already active file number was specified in an OPEN or CREATE statement.

DU

An inactive file number was specified by a DELETE statement.

DW

Indicates a write to a file for which no IF END statement has been executed; may occur if the disk directory is full.

EF

Indicates a read past an end of file for which no IF END statement has been executed.

ER

A write to a record whose length exceeds the maximum record length specified by an OPEN, CREATE, or FILE statement was attempted.

FR

The renamed file name already exists.

FU

A read or write operation to an inactive file was attempted.

IF

An invalid file name was specified.

IR

A record number of zero was specified.

IV

Execution of an INT file created by a version 1 compiler was attempted. Recompile using version 2 compiler.

IX

A FEND statement was encountered before execution of a RETURN statement.

ME

A full directory resulted in an error while you were creating or extending a file.

MP

A third MATCH function parameter was zero or negative.

NF

A file number less than 1 or greater than 20 was specified, or a file statement was executed when 20 files were already active.

NM

Not enough memory was available to load the program.

NN

A PRINT-USING statement could not print a number because no numeric data field could be found in the USING string.

OD

A READ statement was executed with no corresponding data.

OE

Invalid execution of an OPEN statement for a nonexistent file when no prior IF END statement had executed.

OI

An ON GOSUB expression, or an ON GOTO statement was evaluated as a number less than 1, or greater than the number of line numbers in the statement.

OM

Current program exceeded available memory. Close unneeded opened files, nullify unused strings, and read data from a disk file.

QE

A PRINT string contained a quotation mark and could not be written to the specified file.

RB

Attempted random access to a file activated with BUFF where more than one buffer was specified.

RE

Attempted read past the end of a record in a fixed file.

RG

A RETURN was issued for which there was no associated GOSUB statement.

RU

A random read or print to a file that was not fixed was attempted.

SB

An ARRAY subscript exceeded the defined boundaries.

SL

A string longer than 255 bytes resulted from a concatenation operation.

SO

The file specified in SAVEMEM was not on the indicated disk.

SS

The second parameter in the MID$ function, or the last parameter in the LEFT$ or RIGHT$, was negative or zero.

TL

The TAB statement parameter was less than 1 or greater than the current line width.

UN

A PRINT USING statement contained a null edit string, or an escape character (\) was the last in an edit string.

WR

An attempt was made to write to a file after it had been read but before it had been read to the end of the file.

MBASIC Error Messages

1:NF

NEXT without FOR

A NEXT statement variable was encountered without a corresponding FOR statement variable.

2:SN

Syntax Error

An incorrect sequence of characters was encountered.

3:RG

Return without GOSUB

A RETURN statement was encountered for which no previously executed GOSUB could be matched.

4:OD

Out of data

Indicates that a READ statement was executed for which no unread DATA statements could be found.

5:FC

Illegal function call

possibly as the result of:

a) Too large or negative subscript
b) Negative or zero LOG argument
c) Negative argument to SQR
d) Negative mantissa with noninteger exponent
e) Illegal call to USR function with no defined starting address
f) Improper argument to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, TAB, SPC, STRING$, SPACE$, INSTR, ON . . . GOTO

6:OV

Overflow

Result of calculation exceeds upper limit of MBASIC 80 number format and thus cannot be displayed.

7:OM

Out of memory

Indicates that the program is too long, or has too many loops, variables, or expressions.

8:UL

Undefined line

A nonexistent line number was referenced in a GOTO, GOSUB, IF . . . THEN . . . ELSE, or DELETE statement.

9:BS

Subscript out of range

The array element being referenced is outside the array dimensions or has the wrong number of subscripts.

10:DD

Redimensioned array

Indicates that more than one DIM statement was given for the same array. May also be caused when a DIM statement is given for an array after the default dimension of 10 has already been established.

11:/O

Division by zero

Division by zero was encountered in an expression, or the operation of involution resulted in zero being raised to a negative power.

12:ID

Illegal direct

Execution of a statement that is illegal in direct mode was attempted.

19

No RESUME

No RESUME statement has been specified for the error-trapping routine in progress.

20

RESUME without error

A RESUME statement has been encountered before an error-trapping routine has been entered.

21

Unprintable error

An error for which no message has been defined has been encountered.

22

Missing operand

An expression containing an operator was encountered that had no operand following it.

23

Line buffer overflow

Indicates that the line being input contains too many characters.

26

FOR without NEXT

A FOR statement for which no corresponding NEXT statement has been specified has been encountered.

29

WHILE without WEND

A WHILE statement for which no corresponding WEND statement has been specified has been encountered.

30

WEND without WHILE

A WEND statement for which no corresponding WHILE statement has been specified has been encountered.

50

Field overflow

The record length allocated by a FIELD statement for a random file is being exceeded.

51

Internal error

An internal malfunction has occurred; report this error to your dealer.

52

Bad file number

The file number being used to reference a file is either out of the range specified at initialization or is a file number that has not been OPENed.

53

File not found

The file being referenced by a LOAD, KILL, or OPEN statement does not exist on the currently logged drive.

54

Bad file mode

An illegal attempt has been made to LOAD a sequential file using PUT, GET, or LOF. Can also be caused by execution of an OPEN statement for which a file mode other than I, O, or R has been specified.

55

File already open

A sequential output mode OPEN statement is issued for an already open file. Can also be caused by a KILL statement issued for an open file.

57

Disk I/O error

A fatal error occurred during an input or output operation.

58

File already exists

The file name supplied in the NAME statement already exists on the specified drive.

61

Disk full

The message shows that there is no space left on the currently logged drive.

62

Input past end

Indicates that an INPUTstatement was issued after all the data in the file had been used, or for a null file.

NOTE

Use the EOF function to detect the end of file.

63

Bad record number

The record number specified in the PUTor GET statement is either greater than the maximum allowed (32767) or equal to zero.

64

Bad file name

An invalid file name was used in a LOAD, SAVE, KILL, or OPEN statement.

66

Direct statement in file

A direct statement used to load an ASCII-format file caused execution of the LOAD statement to be terminated.

67

Too many files

Attempt to create a new file using SAVE or OPEN cannot be accomplished because all 255 directory entries are full.

APPENDIX A

ASCII and Hexadecimal Conversions

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table 1 defines ASCII symbols, then Table 2 lists the ASCII and hexadecimal conversions. The table includes binary, decimal, hexadecimal, and ASCII conversions.

TABLE 1. ASCII SYMBOLS

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line-feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	space
ESC	escape	STX	start of text
ETB	end of transmission	SUB	substitute
ETX	end of text	SYN	synchronous idle
FF	form-feed	US	unit separator
		VT	vertical tabulation

TABLE 2. ASCII CONVERSION TABLE

Decimal	Hexadecimal	Binary	ASCII	
0	0	0000000	NUL	
1	1	0000001	SOH	(CTRL-A)
2	2	0000010	STX	(CTRL-B)
3	3	0000011	ETX	(CTRL-C)
4	4	0000100	EOT	(CTRL-D)
5	5	0000101	ENQ	(CTRL-E)
6	6	0000110	ACK	(CTRL-F)
7	7	0000111	BEL	(CTRL-G)
8	8	0001000	BS	(CTRL-H)
9	9	0001001	HT	(CTRL-I)
10	A	0001010	LF	(CTRL-J)
11	B	0001011	VT	(CTRL-K)
12	C	0001100	FF	(CTRL-L)
13	D	0001101	CR	(CTRL-M)
14	E	0001110	SO	(CTRL-N)
15	F	0001111	SI	(CTRL-O)
16	10	0010000	DLE	(CTRL-P)
17	11	0010001	DC1	(CTRL-Q)
18	12	0010010	DC2	(CTRL-R)
19	13	0010011	DC3	(CTRL-S)
20	14	0010100	DC4	(CTRL-T)
21	15	0010101	NAK	(CTRL-U)
22	16	0010110	SYN	(CTRL-V)
23	17	0010111	ETB	(CTRL-W)
24	18	0011000	CAN	(CTRL-X)
25	19	0011001	EM	(CTRL-Y)
26	1A	0011010	SUB	(CTRL-Z)
27	1B	0011011	ESC	(CTRL-[)
28	1C	0011100	FS	(CTRL-\)
29	1D	0011101	GS	(CTRL-])
30	1E	0011110	RS	(CTRL-^)
31	1F	0011111	US	(CTRL-_)
32	20	0100000	(SPACE)	
33	21	0100001	!	
34	22	0100010	"	

TABLE 2. ASCII CONVERSION TABLE (continued)

Decimal	Hexadecimal	Binary	ASCII
35	23	0100011	#
36	24	0100100	$
37	25	0100101	%
38	26	0100110	&
39	27	0100111	'
40	28	0101000	(
41	29	0101001	)
42	2A	0101010	*
43	2B	0101011	+
44	2C	0101100	'
45	2D	0101101	-
46	2E	0101110	.
47	2F	0101111	/
48	30	0110000	0
49	31	0110001	1
50	32	0110010	2
51	33	0110011	3
52	34	0110100	4
53	35	0110101	5
54	36	0110110	6
55	37	0110111	7
56	38	0111000	8
57	39	0111001	9
58	3A	0111010	:
59	3B	0111011	;
60	3C	0111100	<
61	3D	0111101	=
62	3E	0111110	>
63	3F	0111111	?
64	40	1000000	@
65	41	1000001	A
66	42	1000010	B
67	43	1000011	C
68	44	1000100	D
69	45	1000101	E

TABLE 2. ASCII CONVERSION TABLE (continued)

Decimal	Hexadecimal	Binary	ASCII
70	46	1000110	F
71	47	1000111	G
72	48	1001000	H
73	49	1001001	I
74	4A	1001010	J
75	4B	1001011	K
76	4C	1001100	L
77	4D	1001101	M
78	4E	1001110	N
79	4F	1001111	O
80	50	1010000	P
81	51	1010001	Q
82	52	1010010	R
83	53	1010011	S
84	54	1010100	T
85	55	1010101	U
86	56	1010110	V
87	57	1010111	W
88	58	1011000	X
89	59	1011001	Y
90	5A	1011010	Z
91	5B	1011011	[
92	5C	1011100	\
93	5D	1011101	]
94	5E	1011110	^
95	5F	1011111	<
96	60	1100000	’
97	61	1100001	a
98	62	1100010	b
99	63	1100011	c
100	64	1100100	d
101	65	1100101	e
102	66	1100110	f
103	67	1100111	g
104	68	1101000	h

TABLE 2. ASCII CONVERSION TABLE (continued)

Decimal	Hexadecimal	Binary	ASCII
105	69	1101001	i
106	6A	1101010	j
107	6B	1101011	k
108	6C	1101100	l
109	6D	1101101	m
110	6E	1101110	n
111	6F	1101111	o
112	70	1110000	p
113	71	1110001	q
114	72	1110010	r
115	73	1110011	s
116	74	1110100	t
117	75	1110101	u
118	76	1110110	v
119	77	1110111	w
120	78	1111000	x
121	79	1111001	y
122	7A	1111010	z
123	7B	1111011	{
124	7C	1111100	|
125	7D	1111101	}
126	7E	1111110	~
127	7F	1111111	DEL