EDIT
Advanced 8080 Editor
User's Manual

Describes EDIT, Release 1.0

Processor Technology Corporation

7100 Johnson Industrial Drive
Pleasanton, CA 94566
Telephone (415) 829-2600

First Edition, First Printing, June 1978
Manual Part No. 727141

IMPORTANT NOTICE

EDIT was derived from EDIT3, a product of LSM Engineering. Major portions of this manual are subject to copyright by LSM Engineering, and are used here with permission of the author.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION 1

INTRODUCTION

1.1 CAPABILITIES

EDIT is a text editor program that allows for the creation or modification of ASCII files such as source files coded in FORTRAN, BASIC, or Assembly Language. This program allows editing on character, string, line and page levels; at any of these levels additions, insertions, substitutions and deletions of text may be made. Additionally, EDIT offers the option to retain a command string as a macro and execute it repeatedly.

The EDIT program itself resides in low memory and requires approximately 4K. EDIT uses the remaining portion of memory as the text buffer area, reserving a few areas: one for use as a command buffer, another for use as a macro buffer, and others for cassette Input/Output.

EDIT receives its text from two sources: it reads data from multiple-block tape files (see Appendix 2) or allows creation of new text on-line from an operator's keyboard. An input file is treated as a continuous string of characters, usually organized into pages. Upon command, EDIT will read one page of text from a tape file: that is, reading will progress until a page terminator (formfeed character) is encountered in the file, or until the buffer is full. Text may be stored, at the conclusion of editing, with or without the page terminator. (A text without terminators will be regarded by EDIT as a single page.)

Notice that when you use EDIT, you will most likely be using two cassette recorders, unit 1 for input, unit 2 for output. Details on how to connect the cassette recorder to the computer, itself, are to be found in Section 7 of your Sol manual. (If you intend to use only one cassette recorder for both input and output, it will be necessary for you to move the cables from jack to jack.) Section 14 of this manual provides some useful information about working with cassette recorders, in general.

It is possible to use this cassette version of EDIT to write files compatible with the disk system. The name of any such file should be PTFIL, and its block size should be 1024. (See Section 5.1.2 on block size.) A file which has already been written with a different name and/or block size may be edited to incorporate these characteristics. (See Section 5 about tape file input and output.)

1.2 CONVENTIONS

The conventions listed on the next page are used to clarify the commands and examples within this manual:

1. The ESCape key, entered by the operator, is echoed to the Video Display, and is represented in the examples by a dollar symbol ($).

2. Control characters in this manual are represented by a "^" followed by the character depressed in conjunction with the control key, e.g., ^P denotes a control-P. These representations do not correspond to the way in which control characters actually appear on the Video Display, where each such character, if it is echoed at all, has a symbolic counterpart, e.g., ^A appears as #.

NOTE: Some control characters (such as ^P) are commands and are not echoed to the console.

3. Whenever a string parameter is part of a command format, it is represented in lower case (e.g., Sstring$ represents a command which might actually be entered as Sgoto, or SBLACK, followed by an ESCape.) In the text that accompanies examples a string is set off by quotation marks. For example, the command CAB$CD$$ appears expanded in the explanation as: C "AB" to "CD".

4. Lower case n represents a positive or negative decimal integer which, when it precedes a command, is related to that command in a certain quantitative way: for example, it can indicate the number of times an action is to be performed. The legal values of a quantifier are given in connection with a specific command when it is discussed. (-65535 to 65535 is the maximum range.) A plus sign before a quantifier is for clarification; its use is optional.

5. "<" and ">" are meta-symbols (except as noted) and the enclosed string indicates the type of string desired,rather than an actual string.

6. A <cr> in an example indicates the insertion of a carriage-return.

7. EDIT may be run under either the Processor Technology SOLOS or CUTER monitor programs. "SOLOS/CUTER" is used in this manual to refer to whichever monitor is in use.

1.3 DEFINITIONS

An input file is a continuous string of characters that EDIT reads, page by page, from a tape file or receives on-line from the operator's keyboard. A formfeed character marks the end of a page; a page includes all of the characters up to, but not including, the formfeed. The formfeed is not retained within the text buffer; it may be specified that it be written to the output file.

A page may be segmented into lines. Each line is a string of characters up to and including the carriage return. A line-feed is assumed after each carriage return, although it does not actually appear as a character within the file.

EDIT maintains a character pointer (CP) within the text buffer. This pointer is moved through the text by various commands; it should be regarded as always pointing between two characters, rather than at a particular character!

SECTION 2

ACCESSING THE EDIT PROGRAM

2.1 LOADING

EDIT is loaded at location 0000 through approximately 1100 Hex. To read EDIT off of its cassette tape from SOLOS/CUTER, type either XEQ EDIT <cr> or GET EDIT <cr> EX 0 <cr>.

If you follow the advice given in Section 14 and CAtalog the tape before using it, you will notice that two other programs are recorded after EDIT. PACK and UNPAC are described in Appendix 2; they enable you to 1) take a single-block file created by some other program and convert it to multiple-block structure so that it can be EDITed, or 2) take a multiple-block file created by EDIT and convert it to single-block structure for use in another program, e.g., ALS-8 or Software #1.

2.2 STARTING UP

EDIT is started at location 0000 Hex; whereupon, the EDIT program clears its scratch area, macro, and command buffers, then searches contiguous random access memory (RAM) to find the last available location that the text buffer can occupy.

EDIT then displays:

```
EDIT X.Y ZZZZZ
:
```

Where X.Y is the version of EDIT
ZZZZZ is the total character space (decimal) available in the text buffer
: is EDIT's prompt

If ZZZZZ is a smaller number than you would expect to find available, given the amount of memory in your computer, EDIT has probably encountered either read-only memory, or a bad random-access memory location, and discontinued its search for text buffer space.

The text buffer initially takes up all available memory space. The contents of memory (routines, data, etc.) are not altered until text is entered. The last location for memory usage may be specified at any time after initialization is complete. (The size command is described in Section 11.)

In case of accidental exit, EDIT may be restarted by executing address 0000H, bypassing all clearing and initialization. If memory has not been changed externally, the text buffer will

remain intact. (There are commands which enable you to leave EDIT intentionally, to execute some other routine or to return to SOLOS/CUTER. The G and H commands, which serve these functions, are described in Section 10 of this manual.)

SECTION 3

LANGUAGE ELEMENTS

3.1 COMMAND FORMAT

A single instruction to EDIT has one of the formats listed below. (Spaces are not part of any EDIT commands, except within strings, and are used in this manual only for clarity.)

1. <command>
2. n <command>
3. <command> string $
4. <command> string1 $ string2 $
5. <command> <hex address> $

The command portion is a one or two character mnemonic (such as W or X?). Commands may be upper or lower case.

A signed integer, the n parameter, may be used to quantify a command, that is, to indicate how many times a command is to be executed or how many characters are involved. EDIT interprets a quantifier as a decimal number. Decimal values may range from -65535 to +65535. (If the operator enters a quantifier where it is not required, EDIT executes the command, ignores the quantifier, and returns no error message. If a negative quantifier is specified, but is meaningless to the command, it is taken to be positive; no error message is displayed.)

Some commands require at least one string. A string may be zero or more characters in length, and is terminated by the ESCape key (echoed as the $ symbol on the operator's Video Display). A string may include the carriage return character.

EDIT begins execution of most commands when two consecutive ESCapes are entered. (One of these may be a string terminator.) The commands ^P, ^R, and ^T are exceptions to this rule: they are executed immediately upon being entered by the operator.

NOTE: If you have typed in a command and it appears not to have been accepted properly, make sure that you have actually typed in ESCapes, rather than dollar signs, as command terminators.

EXAMPLES:

: 5L$$ The command to move the cursor to the beginning of the fifth line following its present line position must be terminated by two ESCapes.

: SLXI$$ The command to search for the string "LXI" requires one ESCape as a string terminator, and a second as a command terminator.

3.2 COMMAND SUMMARY

Here is an overview of the commands to be used with EDIT. Note that each command is accompanied by 1) a very brief explanation of its function, and 2) a reference pointing to the section of this manual where a more complete explanation may be found.

The abbreviation, "CP",here and elsewhere in the manual, stands for Character Pointer, as defined in Section 1.3, above.

There are a number of search commands which are listed under COMBINED TAPE INPUT AND OUTPUT, rather than under CP CONTROL, because they have tape input/output functions associated with them.

COMMAND		REFERENCE
SPECIAL CHARACTERS (immediate execution)		
DELete	Deletes the last character entered.	3.4.1
MODE SELECT	Cancels a current command string, or halts its execution.	3.4.2
^T	Turns off tab simulation, if on; turns it on, if off. Tabs are predefined at 8-space intervals. There is no provision for changing them.	4.
SPACE-BAR	Alternately holds and continues output to the console display during TYPE command execution.	4.
^P	Prints the last command string.	12.2
^R	Re-executes the last command string.	12.3
CONSOLE INPUT/OUTPUT		
V	Prints entire buffer on Video Display.	4.
nV	Prints n lines, beginning at the CP, on the Video Display.	4.
T	Sends buffer to the current pseudo-port.	4.
nT	Sends n lines, beginning at the CP, to the current pseudo-port.	4.

nTN	Specifies number of nulls to be sent after each carriage return/line feed on subsequent T commands.	4.
nTW	Specifies the width of an output line (n characters) on subsequent T commands.	4.

OPENING AND CLOSING FILES

< <filename> $	Opens a file for input.	5.1.1
> <filename> $	Opens a file for output.	
n;	Sets block size.	5.1.2
<=	Prints name of current input file.	5.1.3
>=	Prints name of current output file.	
<$	Closes current input file.	5.1.4
>$	Closes current output file.	

INPUT FROM A TAPE FILE

Y	Clears the previous contents of the text buffer, without writing it, and reads ("yanks") the next page.	5.2.1
A	Reads the next page, and appends the input to the current contents of the text buffer.	5.2.2

OUTPUT TO A TAPE FILE

P	Writes the entire text buffer with a final formfeed.	5.3.1
nP	Writes n lines from the CP and a final formfeed.	5.3.1
PW	Writes the entire text buffer without a final formfeed.	5.3.2
nPW	Writes n lines from the CP and no final formfeed.	5.3.2
PE	Writes the entire text buffer with a file terminator. Closes the file.	5.3.3
nPE	Writes n lines from the CP with a file terminator. Closes the file.	5.3.3

COMBINED TAPE INPUT AND OUTPUT

E	Copies current buffer and remainder of input file to output file; closes output file.	5.4.1
nR	Executes, n times, the sequence: P, then Y. (See "INPUT...","OUTPUT...")	5.4.2

Nstring$	Searches buffer for "string"; continues search, page by page, each time writing out the buffer (P) before proceeding to the next page (Y).	7.3
Qstring$	Like Nstring$, but does not write out the text buffer before proceeding to the next page.	7.3
Ostring1$string2$	Changes "string1" to "string2", searching the whole file by doing P's and Y's, as necessary.	8.2.3
Ostring1$$	Deletes "string1", searching the file by doing P's and Y's, as necessary.	8.2.3
?	Rewinds the input tape	5.5

BUFFER INFORMATION

=	Displays the total number of lines/ characters in the text buffer.	6.
@	Displays the number of the line in which the CP resides.	6.
W	Returns the number of characters in the line at which --actually, immediately before which-- the CP resides.	6.
: <hex address> $	Specifies the last available memory location for the text buffer.	11.

CP CONTROL

B	Moves the CP to the beginning of the text buffer.	7.1
Z	Moves the CP to the end of the text buffer.	7.1
L	Moves the CP to the beginning of the current line.	7.2
+ or - nL	Moves the CP forward (+) or backward (-) n lines.	7.2
nJ	Moves the CP to the beginning of the nth line.	7.2
+ or - nM	Moves the CP forward (+) or backward (-) n characters.	7.3
Sstring$	Moves the CP to the first character after "string".	7.3

Also see N and Q commands in combined tape input and output, above.

ADDITION/INSERTION

Istring$	Inserts a string of characters at the CP.	8.1.1
nI	Inserts a decimal value of a single character at the CP.	8.1.2

DELETION/SUBSTITUTION

K	Deletes the entire line, no matter where the CP is located on it.	8.2.1
nK	Deletes from the current CP forward over n carriage returns.	8.2.1.
+ or - nD	Deletes forward (+) or backward (-) n characters.	8.2.2
Cstring1$string2$	Changes "string1" to "string2".	8.2.3
Cstring$$	Deletes "string" (changes it it to null).	8.2.3

Also see O commands in COMBINED TAPE INPUT AND OUTPUT, above.

MACROS

XM <command string> $$	Defines a macro command string.	9.2
nX	Executes the macro n times.	9.3
XD	Deletes the macro.	9.4
X?	Prints the macro.	9.5

LEAVING EDIT

G <hex address> $	Goes to an external user routine.	10.2
H	Close files; return to SOLOS/CUTER.	10.3

3.3 COMMAND STRINGS

EDIT is able to execute, not only a single command, but also a group of commands entered as a series, before returning to the command entry state. As each command is entered into what becomes a command string, it is placed into the command buffer. An ESCape between commands in a command string is optional, unless either 1) a command includes a string parameter, or 2) the following command might otherwise be construed as part of the first command (e.g., The command string X$D must be differentiated from the single command XD).

EXAMPLES:

: Y55L5T$$	This command string consists of three commands: Y, 55L, and 5T. The command string is terminated by two ESCapes.

: SADD$L1TC1$2$$	This command string consists of four commands: S "ADD", L, 1T, and C "1" to "2". Note that only the strings, in the S and C commands, require the ESCape separator, and that when a character string is the final item in a command string, only one additional ESCape is needed.
: SFILE$ <cr> L <cr> :	This command string consists of four commands: S "FILE", L and 5T. A carriage return may be used as a visual separator between commands in a command string; such carriage returns do not affect the way that the command string is executed.

3.4 COMMAND KEYING ERRORS

EDIT executes a command string one command at a time. If EDIT encounters a command that cannot be executed, it will print out an error message and the unexecuted portion of the command string, and it will clear the command buffer. EDIT will ignore a quantifier if it appears without a command.

EXAMPLES:

: YABCZZ33LK$$ String not found ? ? ? CZZ33LK$$:	Six commands: Y, A, B, C "ZZ" to "33", L, and K. EDIT types an error message ("String not found" and "? ? ?"), the unexecuted portion of the command string and the prompt symbol.

CAUTION: A carriage return within a text string or between the characters of a two-character command is not ignored.

AVOID:

: X <cr> D$$	EDIT executes the XD (a legal command) as X, also a legal command, then deletes one character.

If an error is made while keying a command, it may be corrected in one of the following ways:

KEY	ACTION TAKEN
DELete	Deletes last character entered. Cursor backspaces.
MODE SELECT	Cancels a current command string or halts its execution.

3.4.1 Character Level: DELete

As commands are entered, they are stored in the command buffer. DELete deletes the last character entered in the command buffer. Several DELetes may be entered to delete several characters. If the deletion empties the command buffer, EDIT issues the prompt.

3.4.2 Command Level: MODE SELECT

Before a command string is terminated, it can be canceled by issuing a MODE SELECT. EDIT stops, empties the command buffer, and issues the prompt symbol. If the interrupted command is performing I/Ø, the Character Pointer is set to the beginning of the text buffer; otherwise, it is left in its current position (nP, nPW, or nPE), or at the end of the previous text (A).

3.4.3 Command Buffer

The command buffer is of sufficient length (124 characters) to accommodate long command strings; if the command buffer length is exceeded, the command input, up to that point, will be executed. The exception to this rule is that the insert command will allow input until the text buffer is filled.

3.5 SPECIAL HANDLING

Several input characters receive special handling by EDIT, depending upon whether the character originates from the console or from a tape file. Where the word "normal" appears in the table below, it is used to indicate that a particular control character, where it is read or inserted into the text buffer, is regarded as though it were any other text character, i.e., it does not initiate any action by the program or the system. Exceptions are noted below.

CHARACTER	HEX	FROM KEYBOARD	FROM TAPE FILE
MODE SELECT or null, or ^@	ØØ	Executed, not echoed, not retained	Ignored
^A	Ø1	Echoed as #, retained as ^A in the command string	Normal
LINEFEED or ^J	ØA	Not executed	Ignored
DEL or RUBOUT	7F	Executed, not retained	Ignored
^T		Executed, not retained	Normal character (no tab)

ESCape, SHIFT-^K	1B	Echoed as $, retained as ESC in the command string	Ignored
carriage-return, ^M or <cr>	ØD	Echoes as CR/LF, retained as CR	Normal
^L or formfeed	ØC	Echoed as special character, retained as formfeed.	Terminates page, then is discarded
^P	1Ø	Executed and not retained if it is the first character of the command. Otherwise normal.	Normal
^R	12	Executed and not retained if it is the first character of the command. Otherwise normal.	Normal

On output to the Video Display, EDIT provides a linefeed character after each carriage return.

Null, ^A, ESCape and DELete cannot be inserted into the text; control-T can be inserted using the I command and a quantifier. All other characters of this set may be keyed in directly.

Note that SHIFT-3 (the # symbol) and SHIFT-4 (the $ symbol) may also be used as regular characters.

SECTION 4

CONSOLE INPUT/OUTPUT

All manual (as opposed to tape file) input, including entering commands and new text, is taken from the current SOLOS/CUTER input pseudo-port. In this manual console input is assumed to be from a keyboard. All input is echoed on the Video Display.

Non-tape output can be directed to two different places: either the Video Display or the current SOLOS/CUTER output pseudo-port. It is expected that all normal editing will be done on the Video Display and that information and error messages will be sent there. The output to the current pseudo-port is intended primarily to be used to get hard copy output of a text file, where a printer is connected to the port. In this manual, output is assumed to go to the Video Display.

The speed of output to the Video Display is altered if a key representing a digit is hit during printing. The digit 1 causes output to be fastest (no delay), whereas 9 causes it to be slowest. Output to the Video Display is suspended temporarily when the user hits the space bar during printing; it is made to resume when he hits the space bar or another key. Output may be aborted with the MODE SELECT key.

COMMAND	EFFECT
V	Print entire buffer on Video Display.
nV	Print n lines following the Character Pointer on the Video Display. Where n exceeds the number of lines that exist in the buffer past the CP, print as many lines as exist.
T	Send buffer to current pseudo-port.
nT	Send n lines following the CP to the current pseudo-port.
nTN	Specify number of nulls (n) to be sent after each carriage return/linefeed for the subsequent T command.
nTW	Specify output line to be n characters wide for subsequent T commands.
^T	Toggle tabbing mode (on/off)

EXAMPLES:

```
: 2V$$                  The command 2V causes two lines, start-
DOIT:  STA DPEX         ing at the Character Pointer, to be
       LDA    FLAG      printed on the Video Display.

: =1ØV$$                This command string has two commands
                        in it (= and 1ØV).
2/24                    There are two lines (and twenty-four
TUIT:  ANA B            characters) in the buffer. n exceeds
       STA TABLE        this number, even assuming that the CP
                        is positioned at the beginning of the
                        text.  Therefore, only two of the ten
                        requested  lines are printed.

: SLDA$1T$$             After a SEARCH for "LDA," typing starts
        PINK            at the current position of the CP.
: SLDA$L1T$$            To examine the entire line, issue the L
        LDA PINK        command before the nT command.
```

When the tabbing mode is on, all ^I's (Ø9H) sent to the current pseudo-port via the T command cause spaces to be printed until the next tab stop is reached. Tabs are defined at columns 1,9, 17,25,etc. There is no provision for changing these settings. When tabbing is off, ^I's are sent normally. To see how the tabbing looks, set the current pseudo-port to the Video Display, and then use the T command, rather than the V command, to print.

SECTION 5

TAPE FILE INPUT/OUTPUT

5.1 OPENING AND CLOSING TAPE FILES

COMMAND	EFFECT
< <filename> $	Opens the file <filename> for input from tape unit 1.
> <filename> $	Opens the file <filename> for output to tape unit 2.
n;	Sets to n the tape block size for output files. (n is a quantifier.)
<=	Prints the name of the current input file.
>=	Prints the name of the current output file.
<$	Closes the current input file.
>$	Closes the current output file.

NOTE: In these commands the first "<" or ">" is a literal part of the command, not a meta-symbol.

To copy an existing file and change its name, open an output file with the desired filename, and transfer the information in your original file with a succession of Yanks and Puts. If the file is intended to be compatible with the disk system, its name should be PTFIL. (Remember that its block size should also be 1024. The procedure for creating such a file is outlined in Section 5.1.2, below.)

5.1.1 Opening: < <filename> $ or > <filename> $

The < command opens the specified file for input operations. The input file must be on a cassette in tape unit 1. There is an immediate search to make sure that the file exists. The > command opens the specified file for output operations. The output file will be written to tape unit 2. The cassette should be positioned past the leader and placed in RECORD mode. Any file previously being used for the same function (input or output, depending what kind of file has been opened) will be closed, if necessary. Because tape input/output is buffered, tape movement will not necessarily occur instantaneously when a command is entered.

EXAMPLES:

```
: <TUNA$$          Opens the file TUNA (on unit 1) for Y, A, E,
:                  R, N, O, AND Q commands.

: >FISH$$          Opens the file FISH (on unit 2) for P, E, R,
:                  N, O, AND Q commands.
```

A legal filename consists of up to five ASCII characters, none of which may be a blank or a slash. (See Appendix 1 for a list of ASCII characters.) If you open a file for input and the file does not exist on the tape being read--if, for example, the filename is misspelled in the command--EDIT will read through the whole tape in search of a matching header, and never return an error message. The MODE SELECT key can be used to abort a read; the message "Tape read error" will be displayed, and the prompt (:) will reappear on the screen. Enter the command again, using the correct filename.

5.1.2 Setting Block Size: n;

Output files are written on the tape as a series of blocks. The ; command sets the number of bytes to be in each block. Default and maximum is 1024. For BASIC files this parameter should be set to 256. The larger the block size, the quicker tape access is, by virtue of the reduced number of between-block gaps. If size 0 or no size is specified, the block size is set to the default value. The block size must be set before the output file is opened.

If the output file is intended for use in ALS-8 or Software #1, you will need to use the PACK program (described in Appendix 2) to change the file structure.

If the file which you are creating is intended for use in the disk system, its block size must be set to 1024. The following procedure will result in the "copying" of an existing, but not properly structured, file into a file with the desired characteristics:

1. Open the existing file for input. (See 5.1.1)
2. Set block size to 1024 bytes.
3. Open for output a file called PTFIL.
4. Use Y, followed by E, to copy the file.

5.1.3 Printing: <= or >=

The <= command prints the current input <filename>; the >= command prints the current output <filename>. If no such file is open, EDIT will respond with a blank line.

EXAMPLE:

```
: <=$$
TUNA                The last < command was "<TUNA$".
:

: >=$$
FISH                The last > command was ">FISH$".
:

: >=$$
                    No output file is open.
:
```

5.1.4 Closing: <$ or >$

The <$ command closes the current input file. The >$ command closes the current output file. If the corresponding file was not open, then the command has no effect.

EXAMPLE:

```
: <$<=$$            Closes the input file and then tries to print
                    its name.
:
```

Any file written in EDIT must be either closed or endfiled before an attempt is made to read from it. The Put and Endfile (PE) command will automatically close a file; the command to close a file will automatically endfile it. Also, leaving EDIT will result in the closing of all files open at the time that the command is entered.

5.2 INPUT FROM A TAPE FILE

The following commands allow input from a file that is opened:

COMMAND	EFFECT
Y	Reads the next page into the text buffer and overwrites the previous buffer contents.
A	Reads the next page into the text buffer, appending the input to the end of the current buffer contents.
Q	See Section 7.3

Any quantifier preceding these commands is ignored. Note that the formfeed page terminator is not retained within the text buffer and all input characters are masked to 7 bits.

5.2.1 YANK: Y

The YANK (Y) command always destroys the current contents of the text buffer, and attempts to read in a page from the assigned file. The CP is positioned before the first character of the new page. If no page is available, EDIT issues the appropriate error message; EDIT does not complain if the next page is empty (formfeed only).

EXAMPLE:

```
YY$$        YANKS one page, then immediately YANKS another page
            from the assigned file.
            The first page of text YANKED was destroyed but
            the next page is available for editing.
```

If the input fills the text buffer before a formfeed is encountered, EDIT types the "Buffer full" error message. A small amount of space, from 133 to 265 bytes, has been reserved in the text buffer, so that some editing may still be done before the text, or part of it, is written out (See also section 5.2.2).

5.2.2 APPEND: A

The APPEND (A) command does not destroy the previous contents of the text buffer. It appends the subsequent page to the current contents. Unless the formfeed character is inserted, the two pages are now concatenated. The CP is positioned before the first character of the appended page.

EXAMPLE:

```
: YAA$$     YANK one page, then immediately APPEND two addi-
            tional pages.
```

If the input fills the text buffer before an EOF or a formfeed is encountered, EDIT types a message indicating that the buffer is full; the CP is moved to the beginning of the text buffer.

Some text may be written out of the text buffer, and then deleted from it, to clear space. Some text should be deleted in this manner if another APPEND is desired.

EXAMPLE:

```
: AAAAA$$   Attempt to APPEND five additional pages.
Buffer full
? ? ?
AAA$$       EDIT types out the unexecuted portion of the
            command string.
```

5.3 OUTPUT TO A TAPE FILE

The following commands allow output of a given number of lines or the entire text buffer:

COMMAND	EFFECT
P	Writes the entire text buffer with a final formfeed.
nP	Writes n lines, starting at the CP, and a final formfeed.
PW	Writes the entire text buffer without a final formfeed.
nPW	Writes n lines, starting at the CP, without a final formfeed.
PE	Writes the entire text buffer and endfiles.
nPE	Writes n lines, starting at the CP, and endfiles.

The first and last text characters are the limits of the above commands. Under these commands, the CP is never moved, although it is used to locate where writing should start when n is specified.

All output characters have the high bit set equal to zero (no parity).

5.3.1 PUT: P

The entire contents of the text buffer are written to the assigned file and formfeed, specifying a page, is written as the final character. If the text buffer is empty, only the formfeed is written. If n is specified, EDIT writes n lines, starting at the current CP, and a final formfeed; note that the CP is not moved. If n is Ø, only a formfeed is written.

EXAMPLE:

```
: 2ØP$$        Twenty lines and a final formfeed are written.
```

5.3.2 PUT Without A Formfeed: PW

The PW command is like the P command, except that the entire buffer (PW), or the specified number of lines from the current CP (nPW) are written without the final formfeed.

EXAMPLE:

```
: Y$$          Four commands (Y, PW, Y and P) that combine two
: PW$$         pages into one.  The command string YAP$$ could
: Y$$          also be used.
: P$$
```

5.3.3 PUT and Endfile: PE

The PE command is like the P command, except that the entire text buffer (PE) or the specified number of lines from the current CP (nPE) are written and the file is endfiled. No final formfeed is written. If n is Ø, then the file is just endfiled.

Remember that a file should be either closed, or endfiled, or both, if it is to be used later as an input file.

5.4 COMBINED TAPE INPUT AND OUTPUT

COMMAND	EFFECT
E	Copies current buffer and remainder of input file to the output file and closes the output file.
nR	R does a P (Put) followed by a Y (Yank); n indicates how many times R is to be performed.
N	See Section 7.3
O	See Section 8.2.3

5.4.1 End: E

E copies the input file page by page, preserving page structure until the end of file. When the end of file is encountered, E endfiles and closes the output file.

EXAMPLE:

: YCMVIMOVE$$	Reads a page, changes the first occur-
EOF	rence of MVI to MOV, copies input file
:	to output file and endfiles the latter.

5.4.2 Put and Yank: nR

R does a Put, followed by a Yank, as many times as the quantifier n indicates. If the end of file is encountered during the Yank portion of the command, the R command is terminated immediately and EOF is printed. In such a case, the last page of text has NOT been written to the output file.

EXAMPLE:

:Y5RSend$-3DE$$	Yanks a page, does a Put followed by a
EOF	Yank, etc., until EOF is encountered
:	during a Yank. The last page has not been written out.

5.5 REWINDING A TAPE

COMMAND	EFFECT
?	Powers tape unit 1 and issues a rewind message.

It is assumed that only input files will need to be rewound, so only unit 1 can be powered. The ? command causes tape unit 1 to be powered and the message,"Rewind tape unit 1..." to be returned.Once the tape has been rewound, enter a carriage return to remove power from the tape unit.

An input file should be opened again, after rewinding, before the next attempt to read from it.

SECTION 6

BUFFER CONTENTS

The commands listed below provide information about the text buffer and the position of the Character Pointer.

COMMAND	EFFECT
=	Returns the total number of lines/characters in the text buffer.
@	Returns the number of the line in which the CP is currently positioned.
W	Returns the number of the character immediately ahead of which the CP is positioned.

EDIT returns each value as a decimal number. If the text buffer is empty, the = command returns a Ø/Ø value (no lines, no characters); the @ command returns 1 (CP is in the first line), and the W command returns Ø.
If a quantifier precedes any of these commands, it is ignored.

EXAMPLES:

```
: =$$
2370/16878          There are currently 2370 lines and 16878 char-
:                   acters in the text buffer.

: Y=$$              As part of a command string of two commands:
33/1109             YANK, total lines and total characters.
:

: @W$$              Where is the CP?
206                 On line 206, character 15.
15
:
```

SECTION 7

CONTROLLING THE CHARACTER POINTER

Several commands are available for positioning the Character Pointer at the page, line or character level.

7.1 PAGE LEVEL: B and Z

The following commands move the Character Pointer to a specified position relative to the entire page of text being edited.

COMMAND	EFFECT
B	Moves the CP to the beginning of the text buffer (before the first text character).
Z	Moves the CP to the end of the text buffer (after the last text character).

Any quantifiers used with the B and Z commands are ignored: EDIT executes the command once, and issues no error message.

EXAMPLE:

```
: @B2V$$
287
       LDA FLAG
       ORA A
```

The @ command finds the line position of the CP (287); the B command moves it to the beginning of the buffer and the 2V command types the first two lines in the buffer.

7.2 LINE LEVEL: L and J

The following commands move the Character Pointer to a given line.

COMMAND	
L	Moves the CP to the beginning of the current line.
+nL	Moves the CP forward n lines (over n carriage returns).
-nL	Moves the CP backward n lines (over n carriage returns).
nJ	Moves the CP to the beginning of the nth line.

The LINE (L) commands position the CP before the first character of a particular line. Where a quantifier is supplied, it indicates how many carriage returns the CP must encounter in order to reach the appropriate line. In the absence of a quantifier, the current line is assumed.

The L command moves the CP to the beginning of the correct line. EDIT moves the CP back to the last carriage return and then forward one position.

When moving backward (the minus sign is required) through the text, EDIT proceeds by counting n+1 carriage returns back, and 1 position forward. The beginning of the buffer is equivalent to a carriage return). If n directs the CP beyond the limits of the buffer, the CP is left positioned after the last character in the buffer (+n) or before the first (-n).

EXAMPLE:

```
: @-3L@$$       CP starts in line 8 and moves backward 3 lines to
8               the beginning of line 5.
5
:
```

The JUMP (J) command moves the CP to the beginning of the text buffer, then forward over n-1 carriage returns, finally positioning the CP before the first character of the nth line. If n is omitted, the CP is positioned at the beginning of the text buffer. If line n does not exist, EDIT issues the "? ? ?" error message.

EXAMPLE:

```
: @32J@$$       The CP located in line 47 moves to the beginning
47              of the buffer, then jumps to the 32nd line.
32
```

7.3 CHARACTER LEVEL: M, Sstring$, Qstring$, and Nstring$

The following commands move the Character Pointer to a given character position, either relative to where the CP resides when the command is given, or to a group of characters (i.e., a character string).

COMMAND	EFFECT
+nM	Moves the CP forward n characters.
-nM	Moves the CP backward n characters.
Sstring$	Searches the buffer for "string" and positions the CP after the last character of "string."
Qstring$	Searches the file, page by page, for "string." Positions the CP after the last character of "string."
Nstring$	Like Qstring$, but copies the input file to the output file during the search.

The MOVE (+ or - M) command moves the CP forward or backward by character through the text buffer. If n is omitted, the CP is moved one position. If n is specified, the CP moves forward n characters and is positioned before the n+1th character. If n is negative, the CP moves backward characters and is positioned before the nth character back from the current position. If n exceeds the limits of the text, the CP is left positioned after the last text character (+n), for example, after the final carriage return or before the first character (-n) in the text buffer.

EXAMPLE:

```
: 1V5M1V$$                  Three commands:  1V, 5M, and 1V. The
LXI  D,FUN                  CP moves from the beginning of the line
D,FUN                       to the D.
:
```

The SEARCH (Sstring$) command moves the CP forward from its current position while searching for "string" in the text buffer. If "string" is found, the CP is positioned after the last character in "string". The search ends when the first occurrence is encountered. If the text buffer is searched to its end and "string" is not found, EDIT types the "String not found" error message and positions the CP at the start of the text buffer.

```
: 1VSADC$1V$$               Three commands:  1V, SEARCH "ADC" and 1V.
      ADC L                 After finding the string, the CP is posi-
 L                          tioned after the 'C'; the next command
:                           (1V) types from the CP to the end of the
                            line.
```

The SEARCH string may include mask character positions. Enter ^A to mark a character position that will be found to match any other character; ^A is echoed as #.

EXAMPLE:

```
: SA#B#C$L1V$$          The command searches for the occurrence of
A2BXCL 123              A, B and C, each separated by any one char-
:                       acter, and types the line.
```

The N and Q commands are similar to S, except that they search the entire file, page by page. If an end-of-file is reached and "string" is not in the buffer, then the "String not found" message is printed. The N command copies the input file to the output file as it searches, and closes the output file if the search is unsuccessful.

SECTION 8

ALTERING BUFFER CONTENTS

8.1 ADDITION/INSERTION

The following commands allow individual characters and text to be inserted into the text buffer:

COMMAND	EFFECT
Istring$	Inserts a string of characters at the CP.
nI	Inserts the character whose decimal ASCII value is n at the CP.

8.1.1 Text: Istring$

The "string" may be null, one character in length, several lines, or the entire text buffer. The CP is left positioned after the last character of the "string".

EXAMPLE:

```
: I     LDA     UP          The string is comprised of
        MOV     M,A         three lines which are in-
        INX     H           serted.
$$
```

A "long" insertion operation proceeds to fill the command buffer before entering the text into the text buffer; as each additional line is typed (a line is defined as ending in a carriage return), it is entered into the text buffer. At this point cancelling (^@) will only cancel within the line being entered and rubout will only delete characters back to the beginning of the last line started.

8.1.2 Single Character: nI

This command makes it possible to insert into text a character that the keyboard does not have (such as a lower case character) or one that receives special handling. The numeric value of a single character is masked to 7 bits and inserted; the CP is left positioned after the character. The n parameter, which may range from 2 to 127, corresponds to the decimal ASCII representation of the character to be inserted. Appendix A has a list of ASCII characters. Formfeeds (ØCH) may be inserted to divide the text buffer into two or more pages.

Exceptions: null (ØØH), EOF (Ø1H), ESCape (1BH),and DELete (17FH) are ignored.

EXAMPLES:

```
: 16I$$          Inserts the decimal value of ^P into the text
:                buffer at the current Character Position.
```

EXAMPLE OF ERROR:

```
: 128IL1V$$      Attempted to enter three commands: inserting
                 the decimal integer 128, L and 1V, but EDIT
? ? ?            masks the converted integer to 7 bits
1281L1V$$        which become Ø, aborts command string proces-
                 sing with an error message, since the insert
                 command would have tried to insert a null.
```

8.2 DELETION/SUBSTITUTION

Several commands are available to delete text at either the line, character or string level. Substitution of text or a null string (effective deletion) is available only at the string level.

8.2.1 Line Level: K

The following command:

COMMAND	EFFECT
K	Deletes the entire line including the carriage return.
+nK	Deletes from the CP, forward over n carriage returns.

The KILL (K) command counts carriage returns and regards a line as all the characters up to and including the carriage return. The K alone deletes the entire line, no matter where the CP is located on it. If n is specified, EDIT deletes the remainder of the current line, as well as all text encountered for the next n carriage returns. The limits of the buffer are the limits of the K command.

EXAMPLES:

```
: 4J1V$$              Jumps to the 4th line and displays it.
        STA TABF
: B3K1V$$             Moves the CP to the beginning of the buffer,
        STA TABF      KILLS three lines; the old line 4 is now the
                      first line in the buffer.
```

8.2.2 Character Level: D

The following command:

COMMAND	EFFECT
+nD	Deletes from the CP forward n characters.
-nD	Deletes from the CP backward n characters.

The DELETE (D) command, like the MOVE (M) command, is easier to understand if the CP is regarded as pointing between two characters. The limits of the buffer are the limits of the command.

If n is Ø or omitted, one character is deleted.

Since a carriage return is one character, this command may also be used to join lines by deleting the carriage return.

EXAMPLES:

`: 1VS $1DL1V$$`
`JMP @1`
`JMP 1`
`:`
	Five commands: 1V, SEARCH for a space, 1D, L and 1V. The command string types the current line, finds the space, deletes the following character, moves the CP to the start of the line and displays the entire line.

`: B2V$$`
`XCHG`
`RET`
	The command string displays the original two lines.

`: SG$1DL1V$$`
`XCHG RET`
`:`
	Four commands: SEARCH for "G", 1D, L and 1V. After a search for the "G", the next character, a carriage return, is deleted, so that the two lines previously separated by the carriage return are concatenated.

8.2.3 String Level: C or O

The search for and substitution of one string for another is commanded by:

COMMAND	EFFECT
Cstring1$string2	Changes "string1" to "string2".
Cstring1$$	Deletes "string1" (changes it to null).
Ostring1$string2$	Changes "string1" to "string2," searching the entire file while doing P's (Puts) and Y's (Yanks)
Ostring1$$	Deletes "string1," searching the entire file, if necessary.

The CHANGE (Cstring1$string2$) command moves the CP forward from its current position while searching for the first occurrence of "string1". When located, the string is deleted and replaced with "string2"; the CP is positioned after the last character of "string2". If the end of the text buffer is reached before "string1" is found, EDIT types the "String not found" error message, and moves the CP to the start of the text buffer.

Note that if "string2" is null (Cstring$$), "string1" is deleted. This form of the CHANGE command will automatically be the last command in a command string since the two ESCapes force execution of the command string.

Any quantifier preceding C is ignored.

EXAMPLES:

```
: CFILE$TEST$L1V$$          Three commands:  CHANGE "FILE" to
      LDA       TEST        "TEST", L and 1V.  After replacing
:                           "FILE" with "TEST", EDIT types the line.

: ABC69$DE$$                The first two commands (A and B) are
                            executed:  the search for "69" was
                            unsuccessful.
String not found

? ? ?
C69$DE$$
```

The search string ("string1") may contain mask character positions. ^A (echoed as #) should occupy the mask character position or positions in the search string.

EXAMPLE:

: CA##B$BA$$	The command searches for the first occurrence of A and B separated by any two characters and changes those four to BA.

The substitute string ("string2") must not contain any mask characters, or a command error will result.

The O command is similar to the C command, except that if "stringl" is not found in the buffer, a P (Put) followed by a Y (Yank) is done and the search is continued. If the search is unsuccessful, then the message is printed and the output file is closed.

SECTION 9

MACROS

9.1 SUMMARY OF MACRO COMMANDS

EDIT offers the capability of retaining a command string and executing the string repeatedly by issuing a single command. The command string is stored in the macro buffer. The following commands are associated with macro handling (note that spaces are not part of the command definition, and are used here only for clarity):

COMMAND	EFFECT
XM <command string> $$	Defines a macro command string
nX	Executes the macro n times.
XD	Deletes the macro.
X?	Prints out the current macro.

9.2 DEFINING: XM <command string> $$

Any previous macro is destroyed by the definition of a new macro. Any quantifier is ignored when this command is executed. A macro definition becomes the last command in a command string since it is terminated by two ESCapes. The macro command string length is limited to 122 characters.

EXAMPLES:

```
: XMYS$1 $-3D$$   Defines the macro as a command string contain-
:                 ing three commands:  Y, S "$1 " and -3D.


: <TEST$XMY=$$    The command string opens the disk
:                 file TEST for input and then defines a
                  macro containing two commands:  YANK and text
                  buffer information (how many lines/characters)
```

9.3 EXECUTING: X

The commands X, 0X and 1X are equivalent: the command string currently retained in the macro buffer is executed once. If n is greater than 1, the macro is executed n times.

Any positioning of the CP corresponds to the specific commands that are contained in the macro. Any error situation caused by an individual command in the macro causes EDIT to halt execution, type the appropriate error message, and return control to the operator. If a macro is undefined or contains the X command (recursive execution of the macro), EDIT types "Macro error." A macro must not contain other macro commands.

EXAMPLE:

```
: XMYAP$$
: 10X$$       Where the macro buffer contains the command
:             string, YAP, executing the command successfully
              ten times produces 10 pages from the 20 that
              are read in.
```

As with any command, MODE SELECT or ^@ may be used to halt execution.

9.4 DELETING: XD

The macro might best be deleted in cases where a powerful macro would destroy the contents of the text buffer, if the X command were issued inadvertently.

EXAMPLE:

```
: XD$$        Deletes the macro.
```

9.5 PRINTING: X?

Printing the current macro can save reentering a complicated command string, or can be used to determine exactly what effect the last X command had.

EXAMPLE:

```
: X?$$
15L15V$$          The last command that defined a macro was:
                   XM15L15V$$
```

SECTION 10

LEAVING EDIT

10.1 SUMMARY OF EXIT COMMANDS

The user may leave EDIT, using the G or the H command, to execute another routine anywhere in memory, or to return to SOLOS/CUTER. This procedure is comparable to the use of a CALL instruction. After either the G or the H command, executing address 0000 Hex will result in a safe re-entry to EDIT, with the buffer intact, unless it has been changed externally.

COMMAND	EFFECT
G <hex address> $	Generates a return address and goes to the routine at the given location.
H	Close EDIT files and return to SOLOS/CUTER.

10.2 GO TO USER ROUTINE: G <hex address> $

Leading zeros in the hex address are not required. If G is not the last command in the command string, subsequent commands are held until a proper return to EDIT is made. The return location is placed on the EDIT stack; therefore, if the stack is not modified, a RET instruction in the external routine is all that is required to return to command string processing.

10.3 HALTING: H

This is the normal method of leaving EDIT. Any files opened by EDIT will be closed. If H is issued accidentally, the restart address (0000H) may be executed to recover the text buffer. Because the input file is closed, and the output file is endfiled and closed by this command, the previous state of the output file, and the position of the input file, can not be recovered by an execution of the restart address.

SECTION 11

SETTING THE TEXT BUFFER LIMIT

The text buffer uses the highest portion of memory related to EDIT. At system start up, EDIT takes as much contiguous RAM memory as it can find for the text buffer. This area is tested, but left unchanged. If the operator wishes , at any time, to specify the last location to be available for text, the size command may be issued. Issuing the size command, to limit the text buffer area prior to putting in text, will prevent the possible destruction of memory contents above the specified last available memory location.

COMMAND	EFFECT
: <hex address> $	Specifies the last available memory location for the text buffer.

The hex address must be larger than the present last location used for text and less than where it found the end of contiguous memory, or EDIT issues the "? ? ?" error message.

After the size command is issued, EDIT will respond with total buffer character space (decimal) available.

EXAMPLE:

```
: :1FFF$$
3446
```

Limit the text buffer not to expand past memory location 1FFF hex. EDIT indicates that there is now space for 3446 characters in the text buffer.

SECTION 12

IMMEDIATE COMMANDS

12.1 SUMMARY OF IMMEDIATE COMMANDS

There are three single-character commands which are executed immediately upon being entered. They do not affect the command buffer.

COMMAND	EFFECT
^P	Reprint the last command string entered.
^R	Re-execute the last command string entered.
^T	Toggle the tabbing switch. See Section 4.

12.2 PRINT LAST COMMAND STRING: ^P

If ^P is entered as the first character after execution of a command string, that command string is printed. The ^P is not echoed and not retained. If ^P is entered other than as the first character, it is echoed and treated normally. This command is useful as a means of seeing what was just executed, and also as a prelude to the ^R command.

12.3 RE-EXECUTE LAST COMMAND STRING: ^R

If ^R is entered as the first character after execution of a command string, that command string is re-executed. The ^R is not echoed and not retained. If ^R is entered other than as the first character, it is echoed and treated normally. This feature allows the contents of the command buffer to be treated like a second macro.

NOTE: A ^P or ^R may be entered successfully after a ^P or ^R. ^P and ^R will not work after an error has occurred, because the command buffer is cleared when there has been an error.

SECTION 13

ERROR MESSAGES

When an error is encountered, EDIT types an error message and the unexecuted remainder of the command string, beginning with the command in error.

ERROR MESSAGE	EXPLANATION
Unknown command	EDIT did not recognize the character just entered as a valid command.
? ? ? <remainder of command string>	EDIT encountered a command that it was not able to execute. Check the command.
String not found ? ? ? <remainder of command string>	The string, as specified by the operator, was not found. Check the command string itself.
Buffer full ? ? ? <remainder of command string>	The capacity of the text buffer was exceeded--a result of inputting to the limit of the text buffer before a form feed was encountered. The text in the text buffer may be edited and output. This message is also typed when the APPEND Command is issued in spite of an already full text buffer.
Macro error ? ? ? <remainder of command string>	At the time the X command was issued, the macro buffer was empty, or the macro buffer contained the X command. Define a macro that does not include an X command.
EOF	A YANK or APPEND command has found the end of file.
No input file open No output file open	A P, Y or A command was issued and there was no corresponding tape file open.

Error in read from tape	Either there is some problem with the input tape (e.g., a framing error, an over-run error) or MODE SELECT was hit during a read.

NOTE: If a macro command is the first command in the <remainder of command string> message, the error message is expanded to show the macro command quantifier (n), the unexecuted-remainder of the macro command string (in brackets), and the remainder of the command string. The quantifier given for the macro is the decimal number of macro execution attempts left at the point where the condition occurred to end macro command string execution.

EXAMPLE:

n[<remainder of macro command string>$$]<remainder of command string>$$

SECTION 14

ABOUT CASSETTE RECORDERS AND CASSETTE FILES

Successful and reliable results with cassette recorders and cassette files requires a good deal of care. You need to use consistent and careful methods, and you need to know what to expect, when you try to read a manufacturer's tape, or your own. The following methods are recommended:

1) Use only a recorder recommended for digital usage. For use with the Processor Technology Sol or CUTS, the Panasonic RQ-413AS or Realistic CTR-21 is recommended.

2) Keep the recorder at least a foot away from equipment containing power transformers or other equipment which might generate magnetic fields, picked up by the recorder as hum.

3) Keep the tape heads cleaned and demagnetized in accordance with the manufacturer's instructions.

4) Use high quality brand-name tape, preferably low noise, high output tape. Poor tape can give poor results, and rapidly wear down a recorder's tape heads.

5) Bulk erase tapes before reusing. It can be hard to find the file you want in a jumble of old file pieces. Bulk erasing also decreases the noise level of the tape.

6) Keep cassettes in their protective plastic covers, in a cool place, when not in use. Cassettes are vulnerable to dirt, high temperature, liquids, and physical abuse.

7) Experimentally determine the most reliable settings for volume and tone controls, and use these settings only.

8) On some cassette recorders, the microphone can be live while recording through the AUX input. Deactivate the mike in accordance with the manufacturer's instructions. In some cases this can be done by inserting a dummy plug into the microphone jack.

9) If you record more than one file on a side, SAVE an empty file, named "END" for example, after the last file of interest. Once you read its name, you will know not to search beyond it for files you are seeking. One way to avoid having to search for files is to record only one file per cassette, at the beginning of the tape, if you can afford the extra cassettes.

10) Do not record on the first or last minute of tape on a side. The tape at the ends gets the most physical abuse. Do not be impatient when trying to read the first file on a tape. You, or the manufacturer of a pre-recorded program, may have recorded a lot of empty tape at the beginning.

11) Record a file more than once, before it leaves memory. This redundancy can protect you from bad tape, equipment malfunction, and accidental erasure.

12) Most cassette recorders have a feature that allows you to protect a cassette from accidental erasure. On the edge of the cassette opposite the exposed tape are two small cavities covered by plastic tabs, one at each end of the cassette. If one of the tabs is broken out, then one side of the cassette is "write protected." An interlock in the recorder will not allow you to press the record button. A piece of tape over the cavity will remove this protection.

13) Use the tape counter to keep track of the position of files on the cassette. Always rewind the cassette and set the counter to zero when first putting a cassette into the recorder. Time the first 30 seconds and note the reading of the counter. Always begin recording after this count on all cassettes. Record the beginning and ending count of each file for later reference. Before recording a new file after other files, advance a few counts beyond the end of the last file to insure that it will not be written over.

14) The SOLOS/CUTER command CATalog can be used to generate a list of all files on a cassette. In SOLOS/CUTER, type CAT <CR>, rewind to the beginning of the tape, and press PLAY on the recorder. As the header of each file is read, information will be displayed on the screen. If you have recorded the empty file called END, as suggested, you will know when to search no further. If you write down the the catalog information along with the tape counter readings and a brief description of the file, you will be able to locate any file quickly.

15) Before beginning work after any modification to the system, test by SAVEing and GETting a short test program. This could prevent the loss of much work.

In addition to using the above procedures methodically, you need to know the various ways in which programs may be recorded on tapes you have purchased:

1) If you cannot read a file consistently, and suspect the tape itself, do not despair. The same file may have been recorded elsewhere on the tape. Processor Technology often records a second version, later on the same side of the tape. When you first get a tape, CATalog it with SOLOS or CUTER so you will know exactly what it contains. Write down the tape counter readings at the same time.

2) An empty file named END is sometimes placed at the end of the recorded portion of a tape. When SOLOS CATalogs a file, the file header information is displayed as soon as the beginning part of the file passes the tape head, but nothing is displayed when the end of the program passes by. If another filename such as END is displayed, you know you have just passed the end of the previous file.

3) Some of the programs supplied by Processor Technology contain a checksum test within their code, in addition to the checksum test which SOLOS performs. When a program containing this test is first executed after loading, the checksum test reads all of the program in memory, and calculates a checksum number which is compared with a correct value. If the numbers match, the program in memory is correct. Nothing is displayed when the numbers match, but if they do not match, the message CHECKSUM TEST FAILED, or a similar message, is displayed. The message may be followed by two numbers, representing the correct and incorrect checksum numbers.

Even though the checksum test was failed, it may be possible to enter the program anyway by typing the carriage return key. The bad data may not even be apparent, if it is in a portion of the program you do not use. It is best, however, to try to find and correct the problem causing the error so the checksum test is passed. The error can be caused by the cassette inteface circuitry, bad memory locations, bad tape, a faulty recording, improper alignment or settings on the cassette recorder, or other equipment problems.

APPENDIX 1
TABLE OF ASCII CODES (Zero Parity)

Paper tape 1 2 3 . 4 5 6 7 P	Upper Octal	Octal	Decimal	Hex	Character		
.	0000	000	0	00	ctrl @	NUL	
● .	0004	001	1	01	ctrl A	SOH	Start of Heading
● .	0010	002	2	02	ctrl B	STX	Start of Text
●● .	0014	003	3	03	ctrl C	ETX	End of Text
●.	0020	004	4	04	ctrl D	EOT	End of Xmit
● ●.	0024	005	5	05	ctrl E	ENQ	Enquiry
●●.	0030	006	6	06	ctrl F	ACK	Acknowledge
●●●.	0034	007	7	07	ctrl G	BEL	Audible Signal
.●	0040	010	8	08	ctrl H	BS	Back Space
● .●	0044	011	9	09	ctrl I	HT	Horizontal Tab
● .●	0050	012	10	0A	ctrl J	LF	Line Feed
●● .●	0054	013	11	0B	ctrl K	VT	Vertical Tab
●.●	0060	014	12	0C	ctrl L	FF	Form Feed
● ●.●	0064	015	13	0D	ctrl M	CR	Carriage Return
●●.●	0070	016	14	0E	ctrl N	SO	Shift Out
●●●.●	0074	017	15	0F	ctrl O	SI	Shift In
. ●	0100	020	16	10	ctrl P	DLE	Data Line Escape
● . ●	0104	021	17	11	ctrl Q	DC1	X On
● . ●	0110	022	18	12	ctrl R	DC2	Aux On
●● . ●	0114	023	19	13	ctrl S	DC3	X Off
●. ●	0120	024	20	14	ctrl T	DC4	Aux Off
● ●. ●	0124	025	21	15	ctrl U	NAK	Negative Acknowledge
●●. ●	0130	026	22	16	ctrl V	SYN	Synchronous File
●●●. ●	0134	027	23	17	ctrl W	ETB	End of Xmit Block
.●●	0140	030	24	18	ctrl X	CAN	Cancel
● .●●	0144	031	25	19	ctrl Y	EM	End of Medium
● .●●	0150	032	26	1A	ctrl Z	SUB	Substitute
●● .●●	0154	033	27	1B	ctrl [	ESC	Escape
●.●●	0160	034	28	1C	ctrl \	FS	File Separator
● ●.●●	0164	035	29	1D	ctrl]	GS	Group Separator
●●.●●	0170	036	30	1E	ctrl ^	RS	Record Separator
●●●.●●	0174	037	31	1F	ctrl _	US	Unit Separator
. ●	0200	040	32	20	Space		
● . ●	0204	041	33	21	!		
● . ●	0210	042	34	22	"		
●● . ●	0214	043	35	23	#		
●. ●	0220	044	36	24	$		
● ●. ●	0224	045	37	25	%		
●●. ●	0230	046	38	26	&		
●●●. ●	0234	047	39	27	'		
.● ●	0240	050	40	28	(		
● .● ●	0244	051	41	29	)		
● .● ●	0250	052	42	2A	*		
●● .● ●	0254	053	43	2B	+		
●.● ●	0260	054	44	2C	,		
● ●.● ●	0264	055	45	2D	-		
●●.● ●	0270	056	46	2E	.		
●●●.● ●	0274	057	47	2F	/		
. ●●	0300	060	48	30	0		
● . ●●	0304	061	49	31	1		
● . ●●	0310	062	50	32	2		
●● . ●●	0314	063	51	33	3		
●. ●●	0320	064	52	34	4		
● ●. ●●	0324	065	53	35	5		
●●. ●●	0330	066	54	36	6		
●●●. ●●	0334	067	55	37	7		
.●●●	0340	070	56	38	8		
● .●●●	0344	071	57	39	9		
● .●●●	0350	072	58	3A	:		
●● .●●●	0354	073	59	3B	;		
●.●●●	0360	074	60	3C	<		
● ●.●●●	0364	075	61	3D	=		
●●.●●●	0370	076	62	3E	>		
●●●.●●●	0374	077	63	3F	?		

APPENDIX 1
TABLE OF ASCII CODES *(Cont'd)* (Zero Parity)

Paper tape 1 2 3 . 4 5 6 7 P	Upper Octal	Octal	Decimal	Hex	Character	
. . . . ●	0400	100	64	40	@	
● . . . . . . ●	0404	101	65	41	A	
. ● . . . . . ●	0410	102	66	42	B	
● ● . . . . . ●	0414	103	67	43	C	
. . ● . . . . ●	0420	104	68	44	D	
● . ● . . . . ●	0424	105	69	45	E	
. ● ● . . . . ●	0430	106	70	46	F	
● ● ● . . . . ●	0434	107	71	47	G	
. . . ● . . . ●	0440	110	72	48	H	
● . . ● . . . ●	0444	111	73	49	I	
. ● . ● . . . ●	0450	112	74	4A	J	
● ● . ● . . . ●	0454	113	75	4B	K	
. . ● ● . . . ●	0460	114	76	4C	L	
● . ● ● . . . ●	0464	115	77	4D	M	
. ● ● ● . . . ●	0470	116	78	4E	N	
● ● ● ● . . . ●	0474	117	79	4F	O	
. . . . ● . . ●	0500	120	80	50	P	
● . . . ● . . ●	0504	121	81	51	Q	
. ● . . ● . . ●	0510	122	82	52	R	
● ● . . ● . . ●	0514	123	83	53	S	
. . ● . ● . . ●	0520	124	84	54	T	
● . ● . ● . . ●	0524	125	85	55	U	
. ● ● . ● . . ●	0530	126	86	56	V	
● ● ● . ● . . ●	0534	127	87	57	W	
. . . ● ● . . ●	0540	130	88	58	X	
● . . ● ● . . ●	0544	131	89	59	Y	
. ● . ● ● . . ●	0550	132	90	5A	Z	
● ● . ● ● . . ●	0554	133	91	5B	[	shift K
. . ● ● ● . . ●	0560	134	92	5C	\	shift L
● . ● ● ● . . ●	0564	135	93	5D	]	shift M
. ● ● ● ● . . ●	0570	136	94	5E	^	shift N
● ● ● ● ● . . ●	0574	137	95	5F	_	shift O
. . . . . ● ●	0600	140	96	60	`	
● . . . . ● ●	0604	141	97	61	a	
. ● . . . ● ●	0610	142	98	62	b	
● ● . . . ● ●	0614	143	99	63	c	
. . ● . . ● ●	0620	144	100	64	d	
● . ● . . ● ●	0624	145	101	65	e	
. ● ● . . ● ●	0630	146	102	66	f	
● ● ● . . ● ●	0634	147	103	67	g	
. . . ● ● ● ●	0640	150	104	68	h	
● . . ● ● ● ●	0644	151	105	69	i	
. ● . ● ● ● ●	0650	152	106	6A	j	
● ● . ● ● ● ●	0654	153	107	6B	k	
. . ● ● ● ● ●	0660	154	108	6C	l	
● . ● ● ● ● ●	0664	155	109	6D	m	
. ● ● ● ● ● ●	0670	156	110	6E	n	
● ● ● ● ● ● ●	0674	157	111	6F	o	
. . . . ● ● ●	0700	160	112	70	p	
● . . . ● ● ●	0704	161	113	71	q	
. ● . . ● ● ●	0710	162	114	72	r	
● ● . . ● ● ●	0714	163	115	73	s	
. . ● . ● ● ●	0720	164	116	74	t	
● . ● . ● ● ●	0724	165	117	75	u	
. ● ● . ● ● ●	0730	166	118	76	v	
● ● ● . ● ● ●	0734	167	119	77	w	
. . . ● ● ● ●	0740	170	120	78	x	
● . . ● ● ● ●	0744	171	121	79	y	
. ● . ● ● ● ●	0750	172	122	7A	z	
● ● . ● ● ● ●	0754	173	123	7B	{	
. . ● ● ● ● ●	0760	174	124	7C	|	
● . ● ● ● ● ●	0764	175	125	7D	}	Alt Mode
. ● ● ● ● ● ●	0770	176	126	7E	~	Prefix
● ● ● ● ● ● ●	0774	177	127	7F	DEL	Rubout

APPENDIX 2

PACK AND UNPAC

In Section 5.2.7 of the SOLOS/CUTER User's Manual, there is a discussion of the two types of files generated and utilized by SOLOS/CUTER. It is important to remember that any given item of software may be able to handle only one of these file structures, and that a program that handles multiple-block files may require that the blocks be of a particular size or in a particular range of sizes. The PACK and UNPAC programs recorded on the same cassette as the software you have purchased convert files of either type into the opposite type; PACK converts a multiple-block file to single-block format, and UNPAC converts a single-block file to multiple-block format.

The following chart is a summary of the file requirements of cassette software. SB stands for single-block, and MB stands for multiple-block. If MB is followed by a slash and a number, that number indicates the block size required or generated by the program.

ITEM	INPUT FILE	OUTPUT FILE
ALS-8*	SB	SB
ASSM	MB/ less than or equal to 1024	No text output file
Ext. BASIC	MB/ 256	MB/ 256
EDIT	MB/ 256 to 1024	MB/ 256 to 1024
SOFTWARE #1	SB	SB

* uses SOLOS/CUTER SAVE and GET commands

If you want to use the output file from one of these programs as the input file for another, and if the file does not have the structure required by the second program, you will need to use PACK or UNPAC to create a file of the correct structure. (Actually, the original file will not be altered; each of these programs reads a file from tape unit 1 and RECORDS A DIFFERENTLY FORMATTED VERSION of the same file on tape unit 2.)

PACK and UNPAC can be used to convert a multiple-block file having one block size into a multiple-block file with another block size. First PACK the file, and then UNPAC it, specifying the block size that you want. (If you have the EDIT program, you will probably find it more convenient to use the n; command than to execute PACK, then UNPAC, for this purpose.)

PACK and UNPAC have only one error message:

Read error - Bad tape file.

This message is displayed during the reading of a file from unit 1; it usually indicates that the file was recorded incorrectly, or that a MODE SELect or CTRL-@ was entered from the keyboard while the tape was being read.

PACK

PACK reads a multiple-block text file from tape unit 1 and writes it out as a single-block file on tape unit 2. When you execute PACK, either by typing XEQ PACK<CR> or the sequence GET PACK<CR> followed by EX 0<CR>, the screen will display:

Multi- to single- block converter

Enter name of multi-block file:

Type the name of the multiple-block file that you want to PACK. PACK will also give this name to the single-block file that it creates. A file name should contain one to five characters, no blanks or slashes; when you have typed the whole name, hit the carriage return key. If you make a typing error BEFORE YOU HIT THE RETURN KEY, use DELete to erase the last character typed. If you hit the return and then discover a typing error in the file name, you can use the ESCape key to restart the series of questions.

Should output file be in ALS-8 format? (Y/N):

If the file is intended for use in the ALS-8 system, answer Y; otherwise, answer N. In ALS-8 format each line of a file begins with a count of the number of bytes in that line. Note that the input file for PACK will never be in ALS-8 format, because ALS-8 format is not used for multiple-block files.

Set up tapes. (Hit return when ready)

Insert the tape containing the multiple-block file in tape unit 1, and put the recorder in PLAY mode. Insert the tape that will contain the single-block file in tape unit 2, and put that recorder in RECORD mode. When you hit the return key, the program will begin reading from tape unit 1 and writing to tape unit 2.

At any time before you hit this final carriage return, you can restart the question-answer section of PACK by hitting the ESCape key. Once you have given the carriage return, however, the only way to interrupt the activity of the program is to use MODE SELECT or CTRL-@ to abort it and return to SOLOS/CUTER. If you abort the program and want to execute it again, you will have to reload it from cassette.

UNPAC

UNPAC reads a single-block text file from tape unit 1 and writes it out as a multiple-block file on tape unit 2. When you execute UNPAC by typing either XEQ UNPAC<CR> or the sequence GET UNPAC<CR> followed by EX Ø, the screen will display:

```
Single- to multi- block converter

Enter name of single-block file:
```

Enter the name of the file that you want to UNPAC. The same name will be given to the multiple-block output file. The file name specifications are the same as those for PACK, and the DELete and ESCape keys serve the same purpose as in PACK (see above).

```
Is input file in ALS-8 format? (Y/N):
```

If the input file is in ALS-8 format, answer Y; otherwise, answer N. The output file will not be in ALS-8 format. (The ALS-8 byte counts will be stripped from the input file before it is written out to the output file.)

```
Enter desired block size for output file:
```

Enter the block size (in number of bytes per block) as a decimal number, and then type a carriage return.

```
Set up tapes.  (Hit return when ready)
```

See the explanation under PACK, above. The ESCape and MODE keys function as they do with PACK; like PACK, UNPAC must be reloaded from cassette in order to be executed after MODE SELECT or Control-@ has been used.