

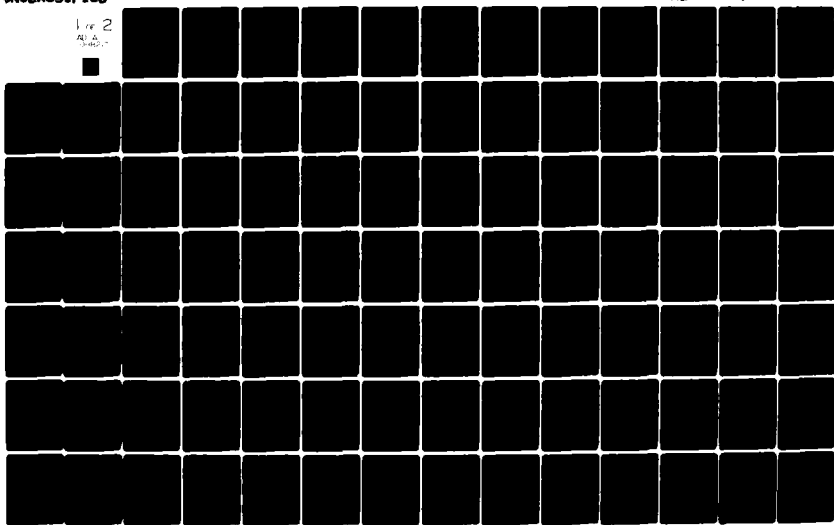
AD-A086 217

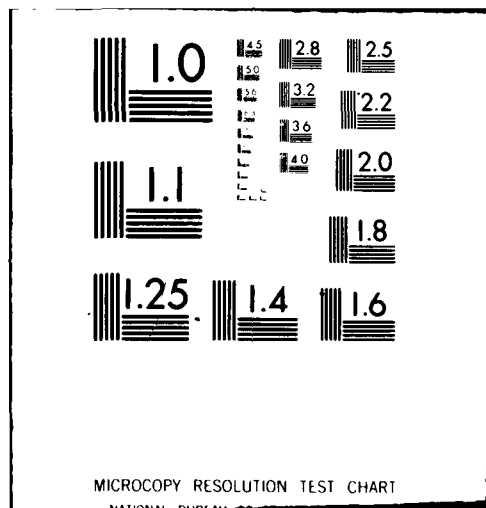
ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/6 9/2
TFS--THE TEXT FORMATTING SYSTEM. A TEXT FORMATTER DESIGNED TO R--ETC(U)
JUL 80 R L CONN

UNCLASSIFIED

NL

1 of 2
ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED





Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (12) 110	2. GOVT ACCESSION NO. AD-A088217	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 TFS -- The Text Formatting System, A Text Formatter Designed to Run Under the CP/M Operating System.		5. TYPE OF REPORT & PERIOD COVERED Final 5/79 to 5/80
7. AUTHOR(s) (10) Richard L. Conn		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS US Army Satellite Communications Agency USA CORADCOM, Attn: DRCPM-SC-4G Ft Monmouth, NJ 07703		8. CONTRACT OR GRANT NUMBER(s) (16) (17) 33
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Elt: 6-11-01-A Proj: 1LI 61101 A91A Task: 33 Work Unit: 131		11. CONTROLLING OFFICE NAME AND ADDRESS -Same as 9-
12. REPORT DATE (11) 25 July 1980		13. NUMBER OF PAGES 104
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (9) Final rept. - May 79 - May 80		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Distribution Unlimited		
18. SUPPLEMENTARY NOTES Source Listing of TFS.ASM is included as part of this report.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Z80 microprocessor, 8080 microprocessor, CP/M, text formatting, word processing, floppy disk		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Text Formatting System (TFS) described by this document is a utility program which runs on the CP/M operating system, Version 1.4. TFS produces a modified listing of the contents of a CP/M text file either on a floppy disk or on a printer page. TFS is a program which interprets control specifications within the text of the file to be formatted and interprets them to produce the formatted listing. These control specifications instruct TFS to perform operations such as		

DTIC
ELECTE
AUG 19 1980
A

80 8 13 059

AD A088217

DDC FILE COPY

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

04 LOOO

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

centering, underlining, right justification, starting a paragraph, pagination, and others.

Unlike other more conventional formatters, however, TFS constructions make up a quasi-programming language. Facilities are included for MACRO definition and application, looping, insertion of data and commands from disk files and incorporation into the document, and interpretive interaction with the user from the console. ←

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TFS -- The Text Formatting System

A Text Formatter Designed to Run Under the CP/M Operating System

by
Richard L Conn

25 July 1980

Requestion For	
FILE NO.	☒
REV. NO.	☐
UNCLASSIFIED	☐
CLASSIFIED	☐
by	
DATE	
APPROVED BY	
DATE	
Dist.	10001
A	

TFS -- The Text Formatting System

Section I -- TFS Reference Manual (Files TFSRM0.TFS to TFSRM8.TFS)

Section II -- TFS HELP File Listing (File TFS.HLP)

Section III -- TFS Source File Listing (File TFS.ASM)

TFS — The Text Formatting System

A Text Formatter Designed to Run Under the CP/M Operating System

by

Richard L Conn

25 July 1980

TFS — The Text Formatting System

Table of Contents

Ch	Pg	Title	Page
1	1	Chapter 1: Introduction to the Text Formatting System	1
1	1	TFS IN GENERAL	1
1	2	TFS C-SPECS	2
1	3	THE TFS COMMAND	3
2	1	Chapter 2: Output Control C-specs	4
2	2	LINE OUTPUT	5
2	3	PARAGRAPHS	6
2	4	HEADINGS, FOOTINGS, and CHAPTERS	7
2	4	CENTERING	7
2	5	TABBING	8
2	5	NUMBERING and BACKSPACING	8
2	7	REGISTERS and REGISTER NUMBERING	10
2	7	COPYING	10
3	1	Chapter 3: Parameter Set C-specs	12
4	1	Chapter 4: Data File Manipulation C-specs	14
4	1	OPENING and CLOSING DATA FILES	14
4	2	READING TFS DATA FILES	15
4	2	EXAMPLES of the USE of TFS DATA FILES	15
5	1	Chapter 5: Miscellaneous C-specs, including Macros	17
5	2	KEYBOARD INPUT	18
5	3	APPENDING FILES	19
5	3	MACROS	19
6	1	Chapter 6: TFS User and Error Messages	20
6	1	TFS USER MESSAGES	20
6	2	TFS ERROR MESSAGES	21
7	1	Chapter 7: A Sample TFS Source File	24
8	1	Chapter 8: Summary of the TFS Commands	26

TFS — The Text Formatting System

CHAPTER 1

Introduction to the Text Formatting System

The Text Formatting System, hereafter referred to as TFS, is a program which produces a modified printer listing of the contents of a CP/M text file. This listing is formatted; that is, it is modified according to control specifications contained within the file itself. TFS is a program which interprets these control specifications and produces the formatted listing.

TFS IN GENERAL

These control specifications, hereafter referred to as C-specs, are commands of the form '%NAME', where '%' designates that a command name follows and 'NAME', a character string of up to four characters, is the name of the C-spec. For example, the C-spec which turns on underlining in TFS is '%UL'; every word which follows a %UL is underlined until another %UL is encountered.

Naturally, as one may notice at this point, a string starting with a '%' designates a C-spec, and this seems to prohibit the use of the '%' character as the first character of a word. This assumption, however, is not the case. An escape sequence, '%%', has been designated to eliminate this problem. Any word starting with a '%%' is displayed as that word beginning with a single '%'; for example, '%this' is printed by TFS as '%this'.

A word, as defined by TFS, is a string of characters delimited by either two blanks, the beginning of a line and a blank, the beginning of a line and a carriage return, or a blank and a carriage return. Hence, a word is any string like 'string', where 'string' is delimited by blanks like ' string ', and words may not cross line boundaries in an CP/M text file, since CP/M defines each line of its files to be terminated by a carriage return character (ASCII 0D hexadecimal) and a line feed character (ASCII 0A hexadecimal). TFS is a word-oriented text formatter. It places words in an output line until the line is filled, and then it prints the line. TFS analyzes each word as it reads the word from the CP/M text file, checks to see if the word is a C-spec, formats the word if it is not a C-spec and executes the C-spec if it is, and then continues by reading the next available word in the file. TFS continues until it reaches the end of the file.

TFS operates in basically two modes: (1) ASIS mode (see the %ASIS command)

TFS — The Text Formatting System

and (2) normal mode. In normal mode, TFS will read words from the CP/M text file, building the current output line as it goes. All output lines are of a definite length, defined either by default or explicitly by the user, and TFS puts the words it reads into the current output line buffer until it reads a word too large to fit in the remaining space. If there are no words in the line at this time, it will force the word into the line and print the line; otherwise, it will output the line. In outputting a line with right justification, TFS inserts blanks between words in the output line buffer until there is a specified number of characters in the line (the length of the line). It will insert these blanks starting at the right end of the line, going to the left. As a result, the left and right margins of a page produced by TFS line up if right justification is employed. If right justification is not employed, TFS simply outputs the line without filling it. Right justification is the default.

While creating the output line in normal mode, TFS follows the convention that all words ending in a special character ('.', '!', ':', or '?') are followed by two blanks; all other words are followed by one blank. It is felt that this convention improves readability of the line.

In ASIS mode, TFS simply outputs the lines following the %ASIS C-spec exactly as they appear. Refer to the documentation on the %ASIS command.

TFS C-SPECS

As mentioned earlier, TFS recognizes many commands (or C-specs) during the formatting of an ARIAN file. These C-specs all take the form of '%NAME', where '%' indicates that a C-spec name follows, and 'NAME' is the name of the C-spec. 'NAME' may contain any combination of upper- and lower-case characters; TFS converts all lower-case characters in the name of a C-spec to upper-case. The following is a list of all C-specs recognized by TFS; the chapters of this manual will describe these C-specs in detail.

Type of C-spec: Output Control

UL, RJ, NORJ, ASIS, BR, CR, AP, P, PX, PAGE, SKIP n, HEAD text,
FOOT text, T n, C text, CB, CH n text, COPY n, ENDC, LOOP, ENDL,
TP n, LEX, BS, N, R r, and DR r

Type of C-spec: Parameter Set

PAP n m, PARX n m, LMAR n, LLEN n, LINE n m, PGON n,
PGOF, PNUM n, SP n, BLK c, SETN n, SETR r n, INCR r, and CLRR

Type of C-spec: Data File Manipulation

OPEN filename, CLOS, and READ

Type of C-spec: Miscellaneous

SAV, RES, PAUS text, REM text, APND filename, MAC, ENDM, KB text,
EXIT, STOP, HALT

TFS — The Text Formatting System

THE TFS COMMAND

TFS is invoked by typing the TFS command. This command causes the TFS program to be loaded into memory and executed.

The TFS command has four options. These options permit the user to: (1) have TFS stop after printing each page, thereby allowing the user to change paper, (2) to display the formatted output on the user's CRT one line at a time, (3) to have TFS skip to a specified page and start printing the report at that page, and (4) to have TFS send the formatted output to disk. '/P' invokes the first option, while '/Sn', where 'n' is a page number (1-9999), causes TFS to skip to the specified page and start printing on this page. '/V' invokes the second option. As the user may suspect, /V and /Sn and well as /P and /Sn may be combined in the command line. It makes no sense to combine /V and /P, since /V stops after every line anyway. '/D' invokes the fourth option; when invoked, the file 'filename.DOC' is generated, and the user is asked which disk he wishes it to be placed on. /D overrides /V if both options are specified.

The TFS command line, therefore, takes the form —

TFS d:filename.ext /o

where 'd:' is optional and may specify a drive, '.ext' is optional ('.TFS' is the default), and '/o' is an option. If an invalid option is specified, TFS will display the valid options and abort; hence, if the user desires a brief on-line memory refresh of the TFS options, he may type something like '/?' and receive the valid option list.

CHAPTER 2

Output Control C-specs

The output control C-specs control the output of TFS directly. They include:

- 1) Line control C-specs, such as %ASIS, %BR, %CR, and %SKIP.
- 2) Page control C-specs, such as %PAGE, %TP, and %CH.
- 3) Format control C-specs, such as %UL, %HEAD, %FOOT, %C, %CB, %RJ, %NORJ, %BS, %T, %N, %R, and %DR.
- 4) Paragraph control C-specs, such as %AP, %P and %PX.
- 5) and the Copy control C-specs, %COPY, %ENDC, %LOOP, %ENDL, and %LEX.

The %ASIS C-spec instructs TFS to display the following lines exactly as they are without filling the output lines. Only the spacing control is carried over the %ASIS C-spec; for instance, if the output is double-spaced when the %ASIS is encountered, the lines within the %ASIS block are also double-spaced.

An %ASIS block consists of a line of the CP/M text file to be formatted which contains the C-spec '%ASIS' followed by an optional comment, the lines to be printed "asis", and a terminating line beginning with the character '%'. The terminating line must start with the '%' character in the first valid character position of the line. For example, a typical %ASIS block would be:

```
%ASIS [comment]
[text to be displayed without formatting]
%ASIS [comment]
```

The only restriction placed on an '%ASIS' block is that the '%' character may not be placed in the first valid character position of a line without terminating the '%ASIS' block.

All text within an %ASIS block is displayed in the same form as it exists within the source text, with the exception that spacing is carried over from the last %SP command. Also, tabs are expanded within the %ASIS block, so it appears

TFS -- The Text Formatting System

as it was typed into the source text file by using an editor like ED.

The '%UL' C-spec is used to start and stop the underlining process. The group of characters to be underlined are enclosed in '%UL' C-specs. For example, a typical use of %UL is:

This text will not be underlined. %UL This text will be underlined. %UL This text will not.

LINE OUTPUT

Lines may be terminated in one of three ways: (1) automatically when TFS determines that the next word will not fit in the current output line, (2) in an implied way when certain TFS C-specs, such as %P, force the output of the current line by the nature of their function, and (3) explicitly in the use of the %BR and %CR C-specs.

%BR (break) and %CR (carriage return) force the output of whatever is in the output line buffer. If more than 10 characters are required to fill the line to make the margins line up, the line is not filled -- it is output exactly as it exists in the buffer. Otherwise, the line is filled as described earlier.

%BR breaks the output line. If the output line buffer is empty, nothing happens on the printer; if it contains something, it is printed, and a carriage return/line feed is output. %CR always performs a carriage return/line feed. If the output buffer is not empty, it acts like a %BR; if the buffer is empty, it outputs just a carriage return/line feed. Hence, if the user wishes to skip down one or two lines, he may insert two %CR's at the appropriate place.

Along the same lines as %BR and %CR, the %SKIP C-spec also can be used to terminate a line. %SKIP is always to be followed by a number 1-99; it acts like the specified number of %CR's. Unlike %CR, however, if the end of the page is encountered before the skipping is completed, a page eject is done and the skipping is stopped. For example, if only three lines are left on the page and a %SKIP 10 is encountered, then only a page eject will be done.

If the user wishes to skip down for the purpose of inserting a diagram in the text, %SKIP alone may not be adequate. If the diagram is to require ten lines and only five lines are left on the page, %SKIP will leave only five lines for the diagram.

To get around this problem, the %TP (test page) C-spec is implemented. This C-spec, which is also always followed by a number from 1-99, tests to see if the specified number of physical lines is left on the current page, and, if such is not the case, it forces a page eject. Hence, to ensure leaving ten lines for the diagram, a %TP 10 followed by a %SKIP 10 may be used. If the ten lines are not available on the current page, a page eject is done followed by the skip; otherwise, just the skip is done.

Another C-spec available to the user is the %PAGE C-spec, which forces a

TFS — The Text Formatting System

page eject. The advantages of this C-spec are obvious.

The normal output of TFS is right- and left-justified, and TFS provides two control C-specs which may be used to selectively engage and disengage the right justification feature. %RJ engages right justification, and %NORJ disengages it. Right justification is the default.

PARAGRAPHS

TFS supports basically two types of paragraphs -- normal, indented paragraphs and exdented paragraphs. Indented paragraphs are as one would expect a paragraph to be. The paragraph starts with the first word indented a specified number of characters in from the left margin. TFS permits indentation of from 0 to 99 characters.

Exdented paragraphs are displayed as the first line extending a specified number of characters to the left of the left margin. The lists presented in this manual are formed using exdented paragraphs. Obviously, the left margin must be greater in length than the number of characters to be extended; if such is not the case, an error message is printed in the output. It makes no sense to extend a line beyond the first character space the printer can print in.

The %PAR and %PARX C-specs define the characteristics of the paragraphs to follow. These C-specs, which are described in detail later, set the number of characters to be indented and exdented.

The %P and %PX C-specs tell TFS that an indented or exdented paragraph starts with the next word. The current output line is broken (%BR) and the new paragraph is started when these C-specs are encountered.

Another feature of TFS is the automatic paragraphing facility. This feature, invoked by the %AP C-spec, makes any line of source text not in an %ASIS block whose first character is a space or tab the first line of a new indented paragraph. %AP is a toggle C-spec; that is, if automatic paragraphing is off, %AP engages it, and if automatic paragraphing is on, %AP disengages it. Hence, the following example demonstrates the use of %AP --

%AP

This is the first line
of a new paragraph. This paragraph
will be formatted as a paragraph
until another line beginning with a space
or tab character is encountered.

This is the second paragraph.

This is the third.

%AP %REM Automatic paragraphing is now off.

TFS — The Text Formatting System

HEADINGS, FOOTINGS, and CHAPTERS

The %HEAD C-spec permits the user to place a heading at the top of each page. This C-spec takes the form of '%HEAD text' on one line of the file. Only the %T, %R, %D (%DR), and %% C-specs may be placed in the text following the %HEAD C-spec, and all of the text following this C-spec to the end of the line in the file is used as the heading.

When a page eject occurs and %PGON and %HEAD are in effect, the first printed line contains the page number. This is followed by the number of blank lines specified by the line spacing C-spec (%SP n — see later) and the heading followed by one additional blank spaced line. For instance, if the output is double spaced, one blank line follows the line containing the page number and three blank lines (1 blank, 1 blank for the next normal line in spacing, and 1 blank to follow that line) follow the heading. If %PGON is not in effect, then the heading only will appear at the top of the page. The %FOOT C-spec is used to place a footer at the bottom of a page. It is structured like %HEAD, and the footer appears after the last text line on the page. At least three blank lines must be present in the bottom margin.

As mentioned above, the %T, %R, %D, and %% C-specs are the only C-specs which may appear in header and footer specifications. The %T C-spec is defined exactly as it normally is; it takes the form of '%T n', and its effect is to tabulate to the specified column. The %R and %D C-specs function as %R and %DR normally function; each is followed by the number of the register to display. %R displays the value of the specified register and increments it, while %D displays the value of the specified register and does not increment it. The %% C-spec is unique to header and footer entries. This C-spec produces the number of the current page (four digits). Hence, the normal page numbering scheme with the %PGON C-spec need not be used, and the user may create his own headers and footers to include the current page number as he desires. A typical footer may look like:

%FOOT Chapter %D 0 %T 60 Page %%

The chapter C-spec, %CH, is of the form '%CH n text', where 'n' is the chapter number 1-99 and 'text' is the chapter title. %CH forces a page eject, skips down 10 physical lines, centers the word 'CHAPTER' and the chapter number, skips down two blank spaced lines (3 physical lines if double spaced, 1 physical line if single spaced, 5 if triple, etc.), and centers the text of the chapter title. As with all centering, the next word after the %CH C-spec must be a C-spec which breaks the output line with a carriage return (see the section on centering). For example, such a C-spec may be %CR, %P, or %PX.

CENTERING

Centering is done explicitly in TFS by using the %C and %CB C-specs and implicitly by using the %CH C-spec. Centering always involves breaking the current line and starting a new line. The %C C-spec is of the form '%C text',

TFS — The Text Formatting System

where text is terminated by the end of the current line in the CP/M text file. When %C is encountered the output buffer is broken, and the centering is done on the next physical line of the printed output. Since spacing is done after a line is printed, centered lines in a single-spaced section of text will be on the next physical line and centered lines on double-spaced sections of text will be on the second physical line following the broken line.

The %CB (Center Block) C-spec is used to center a block of lines. Upon encountering %CB, the current output line is broken and centering begins with the word following the %CB. The lines are centered according to their physical composition in the source file; i.e., each line in the source file is centered by itself without the normal TFS function of searching for the next word regardless of source line boundaries. Centering continues until another %CB is encountered, at which time the current output line is centered and the following word is the next word to be processed in a normal TFS manner.

The normal TFS commands, including macros and underlining, are valid during centering, so centered text may be processed normally in addition to being centered. Note, however, that C-specs like %SKIP are also engaged and may impact on the centering process.

The following is an example of the two center C-specs of TFS —

```
%c This line will be centered
This line will not. %cb Centering starts now.
This line will be centered.
As will this one.
This, also. %cb This will not.
```

TABBING

Tabulation can be done explicitly by using the %T C-spec. This C-spec is of the form '%T n', where n is the number of the column to tab to. If the output line pointer is already beyond this column, no tabulation will be done and an error message will appear on the console.

NUMBERING and BACKSPACING

Two C-specs included at this time are %BS and %N. These C-specs give the user some additional control over the format of the output that is particularly advantageous.

%BS is the backspace C-spec. It has no arguments, and its function is to perform a backspace in the output line buffer. As each word is stored in the output line buffer, it is followed by one or two blanks. If it does not end in a terminating character like '.' (specified earlier), it is followed by one blank; if it ends in such a character it is followed by two blanks. %BS backs up the pointer which points to the next available character position in the buffer; hence, it erases a blank following the last word placed in the buffer.

TFS — The Text Formatting System

`%BS` effectively concatenates the next word to be placed in the buffer with the last word placed in the buffer. This is particularly useful in cases where a C-spec which affects its arguments globally must be restrained.

As a case in point, `%UL` is such a C-spec. If the user wishes to underline a word, for instance, and terminate the word with a piece of punctuation which is not underlined, `%BS` makes this possible. For example,

```
%UL TFS %UL %BS .
```

makes the string 'TFS.' possible.

Two problems with using `%BS` should be noted at this time. The first problem is that if the string to be concatenated to the word in the buffer would result in a line overflow, TFS will not permit the concatenation to occur. For example, if 'stand' is concatenated to 'under' and there are only two spaces left in the output line buffer (i.e., 'understand' will overflow the right margin by three spaces), concatenation will not occur and 'stand' will appear as the first word of the next line. Hence, as a general rule, it is best to use `%BS` to append a single character to the last word in the output buffer. One may safely append larger strings only if he is sure that an overflow will not occur.

The second problem is that `%BS` will not work if the output buffer is empty. No significant error will occur, but the concatenation to the desired word may not occur. Specifically, if the word to be appended to was the last word of the line just printed, then the backspace will be ineffective.

Numbering is the second item covered in this section. It was included primarily because backspacing often accompanies numbering in TFS, such as in the numbering of list items.

The `%N` C-spec is used to enable automatic numbering. TFS supplies a number buffer to the user; this buffer is initialized to 1 when TFS is first invoked and can be set by the `%SETN` C-spec (see later). Whenever `%N` is encountered the value in this buffer (1-99) is placed in the output buffer as a two-character word. If the value is between 1 and 9, the first character is a blank. After the word is placed in the output buffer, the value of the number buffer is incremented. Hence, successive occurrences of `%N` result in successive numbers being placed in the output line, like '`%SETN 1 %N %N %N`' results in 1 2 3. This is particularly useful in producing numbered lists, where the user may wish to insert an element into a list at a later time and does not wish to manually renumber the lists in the CP/M text file. All lists in this manual are produced by using `%N` followed by a `%BS` and a `' '` as the first words in an extended paragraph (see the list at the beginning of this chapter).

TFS — The Text Formatting System

REGISTERS and REGISTER NUMBERING

In addition to the number buffer, TFS supplies 100 registers to the user. These registers are referenced by number, their numbers ranging from 0 to 99. They may be set, incremented, and displayed by the user at his convenience.

All registers are initialized to 1 when TFS is invoked and whenever the %CLRR C-spec (see later) is encountered. Two C-specs are available to display the contents of a register — %R and %DR. '%R n' displays the contents of register n as a two character number (blank filled) and increments the register, while '%DR n' displays the contents of the register and does not increment it.

%R and %DR functions are also available in headers and footers; the C-specs in this case are %R and %D.

Two other C-specs, %INCR and %SETR, are used to increment a register and set its value. These are discussed later.

COPYING

The last C-specs to be described in this chapter are %COPY, %ENDC, %LOOP, %ENDL, and %LEX. These C-specs allow the user to copy sections of the file up to 99 times. With these C-specs, all or selected sections of the CP/M text file may be duplicated in the output.

%COPY takes the form '%COPY n', where 'n' is the number of copies (1-99) to be made. The first word of the block to be copied is the first word of the line following the line containing the %COPY C-spec. For example,

```
%COPY 2 %P This
will be copied twice. %ENDC This once.
```

The entire file may be copied by placing %COPY after the last macro definition in the file (see the section on macros later) and by placing %ENDC as the last word in the file.

The %LOOP and %ENDL C-specs perform the same type of function as %COPY and %ENDC, but they establish an "infinite" loop. This loop can only be terminated by a loop exit C-spec, %LEX, or a TFS exit C-spec, %EXIT. The %LOOP and %ENDL C-specs find their value in applications where the user doesn't know how many copies he wants, the user wants more than 99 copies, and the user is generating his copies from a data file and he is terminating the process by placing a %LEX or an %EXIT C-spec in the data file. All words following %LEX in a Data File record are ignored.

%LOOP and %ENDL, like %COPY and %ENDC, bracket the text to be repeated. %LEX must appear somewhere within a loop bracketed by %LOOP and %ENDL; an error is given and processing is terminated if %LEX does not appear within such a loop.

TFS — The Text Formatting System

Loops may not extend beyond the resident TFS source. That is, a loop may not be begun in one TFS source file, an append be performed, and the loop is terminated in the appended file. Any attempt to do so results in a TFS fatal error.

CHAPTER 3

Parameter Set C-specs

The Parameter Set C-specs are used to assign values to the various control settings used by TFS. These C-specs include:

- 1) Paragraph parameter C-specs, such as %PAR and %PARX,
- 2) Line parameter C-specs, such as %LMAR and %LLEN,
- 3) Page parameter C-specs, such as %LINE, %PGON, %PGOF, and %PNUM,
- 4) the Spacing parameter C-spec, %SP,
- 5) the significant blank character definition C-spec, %BLK,
- 6) and the N and R parameter C-specs, %SETN, %CLRR, %INCR, and %SETR.

All Parameter Set C-specs except %PNUM (see below) are effective immediately. For example, once %PAR is used, all subsequent paragraphs created by %P are affected; additionally, the current paragraph is also affected immediately.

The paragraph parameter C-specs set the indentation or exdentation number and the number of spaced lines to be placed between paragraphs. Both %PAR and %PARX have two numeric arguments -- the first sets the indentation or exdentation and the second sets the number of spaced lines to be placed between paragraphs. As in most numeric parameters, these may take on the values from 1 to 99. For example, '%PAR 5 1' establishes an indentation of 5 spaces and the number of spaced lines to 1. If the output is double spaced, three blank lines (1 associated with the last line of the paragraph and 2 associated with the skipped line) are placed between each paragraph. Indented and exdented paragraphs are discussed in the previous chapter.

The %LMAR and %LLEN C-specs set the location of the left margin and the length of the output line. '%LMAR n' sets the left margin at 'n' characters right of the physical left end of the carriage; '%LLEN n' sets the length of the line to 'n' characters, starting at the current position of the left margin.

TFS — The Text Formatting System

The effects of these C-specs are order-dependent to some extent. Since %LLEN sets the position of the right margin based upon its argument and the position of the left margin and %LMAR ignores the length of the line, if an %LMAR C-spec is executed after a %LLEN C-spec, the new line length is the difference between the old line length and the new left margin. For example, if '%LLEN 80 %LMAR 10' is encountered in the text, the new line length would be 70. However, if '%LMAR 10 %LLEN 80' is encountered, the new line length would be 80. The %LLEN C-spec adds its argument to the current position of the left margin.

The %LINE C-spec sets the format of the output page. %LINE has two arguments — the number of physical text lines on the output page and the number of physical lines on the output page. For instance, '%LINE 40 51' specifies that there are to be 40 physical text lines (including spacing) on a page and 51 physical lines on a page. Hence, when TFS is started and this instruction is encountered, TFS starts counting down from 40 with the line on which the user set the Top-of-Form, and, when it has counted 40 lines, it skips down 11 (51-40) for the next page.

The %PGON and %PGOF turn on and off the automatic page numbering facilities of TFS. %PGON has one argument, the column number from which the page numbers will be right-justified. For example, '%PGON 60' right-justifies the page numbers in column 60, so page 1 will print the 1 in column 60 and page 10 will print the 0 in column 60 and the 1 in 59. %PGOF turns off the automatic page numbering facility. Note that paging itself is always engaged with TFS.

%PNUM sets the number of the next page to be printed to the value of its argument. Unlike the other C-specs, %PNUM is not effective immediately — its value applies to the next page rather than the current page. Hence, '%PNUM 5 %PAGE' would result in the page supplied by the page eject (%PAGE) command having a number of 5. %PGON must be in effect for this C-spec to work.

%SP sets the spacing of the line. This C-spec is of the form '%SP n', where 'n' may take on values from 1-99. '%SP 2' sets double spacing, '%SP 1' sets single spacing, etc. '%SP 0' is not recommended since results are sometimes unpredictable.

%BLK defines the significant blank character. This C-spec is of the form '%BLK c', where 'c' is a single character. Once this C-spec is employed, whenever the character 'c' is encountered, a blank is printed in its place. This significant blank character may be redefined at any time. When TFS is first invoked, no significant blank character is defined.

%SETN n sets the value of the number buffer to the specified value (0-99). This C-spec is used to initialize the number buffer for subsequent uses of the %N C-spec (described earlier).

The final three C-specs in this chapter are %CLRR, %INCR, and %SETR. They manipulate the 100 registers of TFS without printing any values. %CLRR takes the simple form of '%CLRR', and this command sets the values of all the registers to 1. %INCR takes the form of '%INCR r', and it increments the value of register r. Finally, %SETR takes the form of '%SETR r n', and it sets the value of register r to n.

TFS — The Text Formatting System

CHAPTER 4

Data File Manipulation C-specs

TFS supports two basic designations of files — TFS Source Files and TFS Data Files. Physically, there is no difference between these types of files; both may be created by CP/M in the same way, and they may be stored on disk and referenced by TFS. The difference between these two files is in their application. TFS Source Files are processed directly and normally by TFS. TFS Data Files, however, are processed indirectly by TFS; while processing a TFS Source File, elements from the TFS Data File are inserted into the output listing one record at a time, a record being defined as one line of a TFS Data File.

The TFS C-specs which are used to manipulate TFS Data Files are:

- 1) The %OPEN C-spec, which is used to open a Data File,
- 2) The %CLOS C-spec, which is used to close a Data File, and
- 3) The %READ C-spec, which is used to read a record from a Data File.

OPENING and CLOSING DATA FILES

Before a TFS Data File may be used, it must be opened. This function serves to locate the data file, initialize the resident buffer for it, and set a number of internal variables. This is done by using the %OPEN C-spec. This C-spec is of the form '%OPEN D:FILENAME.EXT'. Only one data file may be opened at one time. Both the drive specification and extension are optional. If omitted, the drive spec defaults to the currently logged-in drive and the extension defaults to blank.

Similarly, when a user has finished with one data file and wishes to use another, the %CLOS C-spec is used to close an opened Data File, thereby enabling the user to open another Data File. The %CLOS C-spec is simply of the form '%CLOS'.

Data Files may reside on any Disk Drive, and the user must take care not to change disks while a data file is in use. TFS makes no checks to see if this

TFS -- The Text Formatting System

was done, and subsequent loads of the Data File buffers will come from the disk addresses of the data file on the original disk if this occurs. No prediction can be made as to the results.

READING TFS DATA FILES

Once a Data File has been opened, the user will want to read records from it and insert them into his output. This is done by using the `%READ` C-spec.

A record takes the form of a line in the data file. When a `%READ` is encountered, the internal TFS word pointer is saved, and it is then set to point at the first word of the next record in the data file. Words are then pulled from the data file until the end of the record (end of line) is reached. At this time, the original word pointer is restored and processing continues with the word following the `%READ`. An arbitrary number of words, including none at all, may exist in a record within the data file.

The records of a data file, then, are read and processed exactly as if they simply existed within the source file. This greatly extends the flexibility of TFS, since words within a record may be TFS commands. Only the `%READ`, `%CLOS`, `%ASIS`, `%LOOP`, `%COPY`, and `%MAC` and `%ENDM` commands (C-Specs) may not exist within a Data File, and TFS will generate a fatal error if one is encountered.

EXAMPLES of the USE of TFS DATA FILES

The applications of TFS Data Files, then, are enormous. With the ability to contain TFS commands, the power of a TFS Data File is greatly extended over that of conventional data files.

Two applications are of particular interest. The first is in creating a Data File which may be used to provide a common initialization of a TFS Source File. If several TFS Source Files require the same type of initialization, a TFS Data File may be created containing the proper statements, and it may be read by the Source Files in an infinite loop. The Data File, of course, would have `%LEX` as its last word.

Specifically, the following illustrates such an application:

TFS Source File

```
%LOOP %READ %ENDL
%REM continue source
```

TFS Data File

```
%LMAR 0 %LLEN 85 %PGON 65
%SETN 1 %LEX
```

In this example, the loop in the first line of the TFS Source File is executed, reading the words in the TFS Data File one word at a time. Each time the `%READ` is executed, a line from the data file is read and processed. This loop will execute twice, and, when the `%LEX` C-spec is encountered, the processing will continue in the source file on the second line.

TFS — The Text Formatting System

The second application is in processing mailing lists. The following illustrates such an application:

TFS Source File

```
%OPEN DATAL %REM Open Data File
%LOOP %COPY 4 %READ
%CR %ENDC Dear %READ %BS ,
%P Hello. How are you
today?
%ASIS
                Sincerely,
                Your Friend
%ASIS
%PAGE %ENDL
%P This line is extra.
%CLOS
```

TFS Data File

```
Mr. James Jones
43 Ocean Avenue
Apt. 4
Sea Bright, NJ
Jim
Mr. John Smith
29 Queens Blvd
Sea Bright, NJ
John
%LEX
```

In this example, the TFS Data File is a mailing list, consisting of four lines of address and a fifth line containing the person's first name. Note that the Data File ends with a %LEX C-spec, and control is transferred to the line after the line containing %ENDL of the Source File when this is encountered.

The first line of the Source File reads the four lines of the address and inserts them into the output. The %CR on the next line of the Source File terminates each line read, and the %ENDC terminates the copy.

After the address is read, the word "Dear" is output at the beginning of the fifth line (note the last %CR before the %ENDC) followed by the name of the person from the fifth, tenth, etc., lines of the Data File. The internal pointer is backed up by the %BS C-spec, and a comma is placed immediately after the inserted name.

The body of the letter is then output, followed by the page eject in the Source File.

Here, then, are just two examples of the use of Data Files in TFS. As the user can see, the possibilities are enormous.

TFS — The Text Formatting System

CHAPTER 5

Miscellaneous C-specs, including Macros

The following are other C-specs in TFS:

- 1) The environmental control C-specs, %SAV and %RES,
- 2) The pause C-spec, %PAUS,
- 3) The exit C-specs, %EXIT, %STOP, and %HALT,
- 4) The keyboard input C-spec, %KB,
- 5) The comment C-spec, %REM,
- 6) The append C-spec, %APND, and
- 7) The Macro C-specs, %MAC and %ENDM.

The %SAV and %RES C-specs are used to save and restore the TFS environment, respectively. The TFS environment consists of the following:

- 1) The number of lines to skip between paragraphs,
- 2) The centering flags,
- 3) The automatic paragraphing flag,
- 4) The right justification flag, which indicates if right justification is turned on,
- 5) The current page number,
- 6) The paging flag, which indicates if page numbering is currently turned on,
- 7) The indent and exdent counts,
- 8) The left margin setting,
- 9) The length of the output line,
- 10) The underline flag, which tells if underlining is engaged,
- 11) The heading flag, which tells if a heading is currently set,
- 12) The footer flag, which tells if a footer is currently set,
- 13) The line spacing, and
- 14) The current value of the number buffer.

Whenever %SAV is encountered, these values are saved in a reserve buffer. Their values remain unchanged. At this time, the user may change whichever values he wishes. When he wishes to restore the saved environment, he simply

TFS — The Text Formatting System

enters the %RES command.

The %PAUS C-spec is used to send a message to the user and suspend operation of TFS until the user responds by typing any key on the keyboard of the principal I/O device. It takes the form of %PAUS <text>, where <text> is terminated by the end of the current line. If the user types an <ESC> after the message is displayed, TFS will abort.

The %EXIT, %STOP, and %HALT C-specs are used to terminate interpretation of the TFS source immediately. They force a page eject and then return to the operating system. They may be used for many reasons, but a very good use of these C-specs is to terminate infinite loops and the corresponding output when reading a data file.

The %REM C-spec allows the user to enter a remark, or comment, into the file. The comment starts with the first character following the word '%REM' and continues to the end of the line in the file. The next line is then interpreted normally.

KEYBOARD INPUT

Keyboard Input is a very useful feature of TFS which is designed specifically for application with form letters or documents. Keyboard Input causes TFS to pause in its output processing and allow the user to type one line of text to be inserted into the output at the current position. All of the CP/M input line editor commands are in effect, and processing continues only after the user hits the Return key.

The %KB C-spec engages Keyboard Input and is of the form '%KB text'. Upon encountering this C-spec, TFS prints the text on the rest of the line after the C-spec (%KB) on the user's console, outputs a <CR> <LF>, types the word 'INPUT: ', and waits for the user to type something. The text typed by the user is then inserted into the document, and TFS continues processing.

The text typed by the user, as in all text processed by TFS, may contain embedded commands (C-specs) like %T, %C, etc. Only the %ASIS, %MAC, %ENDM, %COPY, and %LOOP C-specs may not be placed in the text typed by the user, and a TFS fatal error will result if this is done.

For example, a way to use %KB to input the address for a form letter is --

```
%LOOP
%KB Input the address and type %LEX when done
%CR %REM start a new line
%ENDL
```

TFS — The Text Formatting System

APPENDING FILES

In some cases, the user may wish to load another file and continue formatting in the same environment as the previous file (i.e., he may wish to have the same line length, the same paragraph settings, etc.). The %APND C-spec was created to permit the user to do this. This command gives the TFS user the additional flexibility of producing a report which spans over several files.

The %APND C-spec is of the form '%APND D:FILENAME.EXT'. Upon encountering this C-spec, TFS clears the local file space and loads the specified file. It then continues formatting at the first word of the loaded file. The drive specification and file extension are optional; if the drive spec is omitted, the current logged-in disk is default, and if the file extension is omitted, ".TFS" is the default.

All the environmental attributes of the previous file are preserved, but the macros need to be redefined. Hence, the %APND C-spec easily allows the user to chain several files into one formatted listing.

MACROS

Macros are essentially subroutines placed within the TFS formatting file. Each macro is of the form of the %MAC C-spec followed by the name of the macro; this name must be four characters or less -- any additional characters will be discarded. The first word following the macro name is the first word of the macro. The contents of the macro include all words following its name up to and including the %ENDM termination word.

Macros are not executed when they are defined. They are executed only when their names are referenced. For example, if the TFS file contained:

```
%MAC TFS %UL TFS %UL %BS , %ENDM
...
%TFS you see
```

the phrase 'TFS,' would only be printed when the last line was encountered.

All macros must be defined before they are first referenced. TFS is a one-pass formatter, so the table of macro names is only formed as the macros are defined.

The nesting of macros is permitted. Macro definitions, however, may not be nested, but one macro may call another macro which in turn calls another. This type of nesting is permitted up to ten levels deep. An indirect recursion is not checked for by TFS, and such a situation could be disastrous. Up to twenty macros may be defined.

The best way to discover exactly what macros can do is to try them. The macro facility of TFS is indeed very powerful, and it can be of enormous value in creating formatted text.

TFS — The Text Formatting System

CHAPTER 6

TFS User and Error Messages

There are two types of messages issued by TFS: (1) User Messages and (2) Error Messages. This chapter identifies and describes the meaning of all major TFS messages.

TFS USER MESSAGES

PLEASE INSERT NEXT PAGE

Insert the next page into the printer; this message appears when the '/P' (PAUSE after each page) option is used. When ready, the user types any character to continue or <ESC> or Ctrl-C to abort TFS and return to the operating system.

TYPE ANY CHAR WHEN READY, <ESC> OR CTRL-C TO ABORT

This appears when a %PAUS C-spec is used. Again, the user types any character other than <ESC> or Ctrl-C to continue and <ESC> or Ctrl-C to abort TFS and return to the operating system. This message appears when TFS is first engaged in order to give the user an opportunity to set the Top of Form on the printer.

NEW PAGE SETTING — PLEASE SET TOP OF FORM

A %LINE command was issued, resetting the page parameters. Please reset the top of form on the printer. Once top of form is set, type <ESC> or Ctrl-C to abort or any other key to continue.

++ TFS ++ LOAD:

TFS — The Text Formatting System

TFS is in the process of opening the specified file as a source text file. This file was specified either in the TFS command line or by an %APND command.

++ TFS ++ OPEN:

TFS is in the process of loading the specified file as a data file. This file was specified in an %OPEN C-spec.

++ TFS ++ DISK OUTPUT SELECTED

FILE NAME: filename.DOC

DISK OUTPUT DRIVE (A/B/C/D)?

TFS has recognized the /D option and has specified the output file. The user is to select the disk drive (which must NOT be removed during output processing) to which the output file information will be sent.

LST: DISPLAY SET BY DEFAULT

TFS output is to go to the CP/M LST: device.

CON: DISPLAY SELECTED

TFS output is to go to the CP/M CON: device.

PAUSE SELECTED — LST: DISPLAY SET

The /P (Pause after each page) option was given in the command line. TFS output is to go to the CP/M LST: device.

INPUT:

A %KB (Keyboard Input) C-spec has been encountered. TFS is waiting for the user to type something (ending in a <CR>). CP/M input line editing is in effect (<DEL>, Ctrl-X, etc.).

TFS ERROR MESSAGES

\$\$DATA FILE NOT FOUND

The Data File named in an %OPEN C-spec was not found on the Logged-In or

TFS — The Text Formatting System

specified diskette drive.

\$\$DATA FILE NOT OPEN

A %READ was attempted when a Data File had not been opened.

\$\$EOF OF DATA

An attempt was made to read past the end of the opened Data File.

\$\$INVLD CMND IN KEYBOARD INPUT LINE

One of the restricted commands for the %KB C-spec was encountered in the input line just typed. See Chapter 5 for a list of these commands.

\$\$INVLD CMND IN DATA FILE

One of the restricted commands was encountered in the opened Data File. See Chapter 4 for a list of these commands.

\$\$LOOP ERR

An %ENDL or %ENDC was encountered without a preceeding %LOOP or %COPY. This will also occur if the %LOOP or %COPY is in one Source File and the %ENDL or %ENDC is in an appended Source File.

\$\$MACRO RET ERR

An error has occurred in the user's MACRO call structure. An %ENDM was encountered without a corresponding %MAC.

\$\$FILE NOT FOUND

The file referenced in an %APND C-spec was not found on the Logged-In or specified diskette drive.

\$\$TAB ERR

An attempt was made to tab to a column before the current column pointer.

\$\$ERROR — INVALID DISK DRIVE SPECIFIED

An invalid diskette drive specification (e.g., X:) appears in an %OPEN or %APND C-spec.

TFS — The Text Formatting System

\$\$ERROR — TEXT BUFFER OVERFLOW — BREAK UP SOURCE FILE

The TFS source file is too large to completely fit in the TFS text file buffer. Break it into two parts and link these parts using the %APND C-spec.

\$\$INVLD NUM

An error exists in the TFS Source File after a command which expects a numeric constant. Such commands are %CH, %TP, etc.

\$\$XDENT ERR

A %PX was attempted, and the resulting starting location of the Xdented Paragraph was before the first column of the printer.

TFS — The Text Formatting System

CHAPTER 7

A Sample TFS Source File

This chapter is designed to show the user specifically what a TFS source file looks like and how the various features of TFS may be employed to produce a document. As the user can see, the source file under discussion is this chapter itself. This first part of this chapter is a copy of the document after it has been formatted, and the last part of this chapter is a copy of the TFS source file.

As the user may note, three macros have been defined. These are used for generating a numbered itemized list. The macro %LIST initializes the list, the macro %ELST ends the list mode, and the macro %LE allows the user to enter an element into the list.

Macros, as one can see, provide a very useful tool for abbreviating the amount of text typed by the user. They also display the following advantages --

1. They can be made quite mnemonic and logical to use.
2. They can be used to "remember" the structures of the environments to switch to and from.
3. They can greatly improve the legibility of a TFS source file.
4. They are basically very flexible and useful in nature.

This chapter concludes the TFS manual. As the user can see, TFS, the Text Formatting System, is indeed a very useful, powerful, and flexible tool for text formatting, or word processing, applications.

The source listing of this chapter follows --

TFS — The Text Formatting System

```
%llen 80 %par 5 1 %rem set line length to 80 cols and paragraphs to
%rem indent 5 columns and skip 1 line between each paragraph

%mac list %lmar 15 %llen 40 %setr 0 1 %endm
%rem define the macro LIST to set the left margin to col 15, the line
%rem length to 40 columns, and the value of register 0 to 1

%mac elst %lmar 0 %llen 80 %skip 2 %endm
%rem define the macro ELST to reset the left margin and line length
%rem and to skip 2 lines

%mac le %p %r 0 %bs . %endm
%rem define the macro LE to start a new paragraph, display the value
%rem in register 0, increment the value after displaying it, and
%rem appending a period immediately after the number

%ch 7 A Sample TFS Source File
%rem start a new chapter

%ap %rem engage automatic paragraphing
%skip 2 %rem skip down 2 lines
```

This chapter is designed to show the user specifically what a TFS source file looks like and how the various features of TFS may be employed to produce a document. As the user can see, the source file under discussion is this chapter itself. This first part of this chapter is a copy of the document after it has been formatted, and the last part of this chapter is a copy of the TFS source file.

As the user may note, three macros have been defined. These are used for generating a numbered itemized list. The macro %LIST initializes the list, the macro %ELST ends the list mode, and the macro %LE allows the user to enter an element into the list.

Macros, as one can see, provide a very useful tool for abbreviating the amount of text typed by the user. They also display the following advantages --

%list

%le They can be made quite mnemonic and logical to use.

%le They can be used to "remember" the structures of the environments to switch to and from.

%le They can greatly improve the legibility of a TFS source file.

%le They are basically very flexible and useful in nature.

%elst

%skip 2

This chapter concludes the TFS manual. As the user can see, TFS, the %ul Text Formatting System %ul %bs , is indeed a very useful, powerful, and flexible tool for text formatting, or word processing, applications.

The source listing of this chapter follows --

TFS — The Text Formatting System

CHAPTER 8

Summary of the TFS Commands

The TFS Command

The general form of the command to invoke the Text Formatting System (TFS) is --

TFS filename.ext /o

where '.ext' and '/o' are optional. If '.ext' is omitted, '.TFS' is assumed. Valid options are --

Option Meaning

/Sn	Skip the specified number of pages
/V	View the output on the user's console
/P	Pause at the end of each page
/D	Send output to a disk file

Alphabetical Listing of the TFS Commands (C-Specs)

AP	CLRR	HEAD	NORJ	PX	SKIP
APND	COPY	INCR	OPEN	R	STOP
ASIS	CR	KB	P	READ	T
BLK	DR	LEX	PAGE	REM	TP
BR	ENDC	LINE	PAR	RES	UL
BS	ENDL	LLEN	PARK	RJ	
C	ENDM	LMAR	PAUS	SAV	
CB	EXIT	LOOP	PGOF	SETN	
CH	FOOT	MAC	PGON	SETR	
CLOS	HALT	N	PNUM	SP	

TFS — The Text Formatting System

Functional Listing of TFS Commands (C-Specs)

Output Control

AP	DR r	P
ASIS	ENDC	PAGE
BR	ENDL	PX
BS	FOOT text	R r
C text	HEAD text	RJ
CB	LEX	SKIP n
CH n text	LOOP	T n
COPY n	N	TP n
CR	NORJ	UL

Parameter Set

BLK c	LMAR n	PNUM n
CLRR	PAR n m	SETN n
INCR r	PARX n m	SETR r n
LINE n m	PGOF	SP n
LLEN n	PGON n	

Data File Manipulation

CLOS	OPEN filename	READ
------	---------------	------

Miscellaneous

APND filename	KB text	RES
ENDM	MAC name	SAV
EXIT	PAUS text	STOP
HALT	REM text	

HELP File for TFS

Page Number 1 -- HELP File Listing: TFS -- HELP FILE (TFS.HELP) FOR THE TEXT FORMATTING SYSTEM

TFS In General

Invoking TFS

Alphabetical Listing of the TFS C-specs

TFS C-specs Grouped by Class

TFS User Information Messages

TFS Error Messages

:TFS In General

The Text Formatting System, referred to as TFS, is a program which produces a modified printer listing of the contents of a CP/M text file. This listing is formatted; that is, it is modified according to control specifications contained within the file itself. TFS is a program which interprets these control specifications and produces the formatted listing.

These control specifications, hereafter referred to as C-specs, are commands of the form '%NAME', where '%' designates that a command name follows and 'NAME', a character string of up to four characters, is the name of the C-spec. The name of the C-spec may contain any combination of upper- and lower-case characters; all characters in a C-spec name are translated to upper-case.

In order to display a string starting with a '%', the string should be started with the escape sequence '%%'.

Refer to the TFS Manual, "TFS — The Text Formatting System: A Text Formatter Designed to Run Under the CP/M Operating System," for further information.

Page Number 3 -- HELP File Listing: TFS -- HELP FILE (TFS.HELP) FOR THE TEXT FORMATTING SYSTEM

:Invoking TFS

TFS is invoked by a conventional CP/M command line of the general form --

TFS d:filename.typ /o

where

d: is optional and may specify a drive (A, B, C, or D)
.typ is optional; .TFS is the default
/o is also optional, and it specifies one or more of the following options --

/P -- pause at the end of each page
/Sn -- skip the specified number of pages
/N -- view the output on the user's console
/D -- send output to disk file 'filename.DOC'
/? -- ask for general help (option list)

:Alphabetical Listing of the TFS C-specs

The C-specs (embedded commands) recognized by TFS are --

AP	CLAR	HEAD	NORJ	PX	SKIP
APND	COPY	INCR	OPEN	R	STOP
ASIS	CR	KB	P	READ	T
BLK	DR	LEX	PAGE	REM	TP
BR	ENDC	LINE	PAR	RES	UL
BS	ENDL	LIEN	PARX	RJ	
C	ENDM	LMAR	PAUS	SAV	
CB	EXIT	LOOP	PGOF	SETN	
CH	FOOT	MAC	PGON	SETR	
CLOS	HALT	N	PNUM	SP	

Page Number 5 -- HELP File Listing: TFS -- HELP FILE (TFS.HELP) FOR THE TEXT FORMATTING SYSTEM

:TFS C-specs Grouped by Class

Paragraphing -- AP, P, PAR i l, PARX i l, PX
ASIS Mode -- ASIS
Line C-specs -- BR, CR, LINE text physical, LLEN n, LMAR n, SKIP n, SP n
Headers and Footers -- FOOT text, HEAD text
Centering and Chapters -- C text, CB, CH n text
Paging -- PAGE, PGOF, PGON n, PNUM n, TP n
Data File -- CLOS, OPEN d:filename.typ, READ
Appending Files -- APND d:filename.typ
Halting TFS -- EXIT, HALT, STOP
Environment Save/Restore -- RES, SAV
Right Justification Control -- NORJ, RJ
Looping -- COPY n, ENDC, ENDL, LEX, LOOP
Backspacing, Tabulation, Underlining, Bold Face -- BS, T n, UL,
Register -- CLRR, DR r, INCR r, N, R r, SETN n, SETR r n
Macro Definition -- ENDM, MAC name
Output Pause -- PAUS text
Define Significant Blank -- BLK c
Keyboard Input -- KB text
Remark -- REM text

Page Number 6 -- HELP File Listing: TFS -- HELP FILE (TFS.HELP) FOR THE TEXT FORMATTING SYSTEM

:TFS User Information Messages

The TFS User Information Messages are non-fatal information or request messages issued by TFS to the user. They serve to prompt the user for action or to inform the user as to what TFS is currently doing. Most of the User Information Messages are self-explanatory.

The following are the User Information Messages issued by TFS --

PLEASE INSERT NEXT PAGE
TYPE ANY CHAR WHEN READY, <ESC> OR CTRL-C TO ABORT
NEW PAGE SETTING -- PLEASE SET TOP OF FORM
LST: DISPLAY SET BY DEFAULT
CON: DISPLAY SELECTED
PAUSE SELECTED -- LST: DISPLAY SET
++ TFS ++ LOAD: filename.typ
++ TFS ++ OPEN: filename.typ
++ TFS ++ DISK OUTPUT SELECTED
FILE NAME: filename.typ
DISK OUTPUT DRIVE (A/B/C/D)?
INPUT:

Refer to Chapter 5 of the TFS Manual for a complete explanation of these messages.

:TFS Error Messages

All TFS Fatal Error Messages are preceded by \$\$\$. Most of them are self-explanatory. The following are the Fatal Error Messages issued by TFS --

DATA FILE NOT FOUND	LOOP ERR
DATA FILE NOT OPEN	MACRO RET ERR
EOF OF DATA	FILE NOT FOUND
INVLD CMD IN KEYBOARD INPUT LINE	TAB ERR
INVLD CMD IN DATA FILE	INVLD NUM
INVALID DISK DRIVE SPECIFIED	TEXT BUFFER OVERFLOW -- BREAK UP
XIDENT ERR	SOURCE FILE

Refer to Chapter 6 of the TFS Manual for a complete explanation of these messages.

Source Listing of TFS

* PROGRAM NAME: TFS
 * AUTHOR: RICHARD CONN
 * DATE: 14 JUL 80
 * VERSION: 3.0
 * PREVIOUS VERSION: 1.4 (1 SEP 79), 2.0 (22 JAN 80), 2.1 (22 APR 80)

 * TFS -- FORMATS A CP/M TEXT FILE CONTAINING EMBEDDED TFS
 * COMMANDS AND OUTPUTS THE FORMATTED DOCUMENT ON EITHER
 * CON: OR LST:

* THE FORM OF THE TFS COMMAND IS:
 * TFS X:FILENAME.EXT

/P
 /S <NUMBER>
 /V

/D
 * ONLY 'FILENAME' IS REQUIRED.

* THE OPTIONS FOR THE TFS COMMAND ARE:

- 1) /P -- PAUSE AFTER PRINTING EACH PAGE. LST: BECOMES THE OUTPUT DEVICE.
- 2) /S -- SKIP THE SPECIFIED NUMBER OF PAGES AND START PRINTING FROM THERE.
- 3) /V -- VIEW OUTPUT ON CON:.. IN THIS MODE, TFS PAUSES AFTER EACH LINE AND WAITS FOR ANY KEY FROM THE USER. IF THE USER STRIKES THE <ESC> KEY, TFS ABORTS; OTHERWISE, IT CONTINUES.
- 4) /D -- SEND OUTPUT TO DISK

* REFER TO THE TFS REFERENCE, USER'S, AND DESIGN MANUALS
 * FOR FURTHER DETAILS.

 *

```

ORG 100H ; TPA FOR CP/M
JMP TFS ; START

* USER-DEFAULT VALUES
DV1 DB 40 ; NUM OF LINES OF TEXT PER PAGE
DV2 DB 51-40 ; NUM OF REMAINING LINES PER PAGE
DV3 DB 5 ; DEFAULT PARAGRAPH INDENT VALUE
DV4 DB 75 ; DEFAULT LINE LENGTH

* START OF TFS
TFS LDA FCB+9 ; EXTENSION?
CPI .
JNZ TFS0

* SET UP DEFAULT EXTENSION
LXI H,DEXT+8 ; MAKE 'TFS' THE DEFAULT EXT
LXI D,FCB+9
MVI C,3 ; 3 CHARS
CALL LCIR

* SET UP I/O AND STATUS ADDRESSES
TFS0 LDA BOOT+2 ; GET HIGH BYTE OF BIOS VECTOR
MOV H,A ; ... IN H
MVI L,6 ; CONSOLE STATUS ADR
SHLD STADR
MVI L,9 ; CONSOLE INPUT ADR
SHLD INADR
MVI L,0CH ; CONSOLE OUTPUT ADR
SHLD OUTADR
MVI L,0FH ; PRINTER OUTPUT ADR
SHLD PRADR

* GET VIEW OPTION IN OFLAG AND PAGE SKIP COUNT IN SKFLG
XRA A ; A=0
STA DOUT ; SET DISK OUTPUT FLAG TO OFF
STA OFLAG ; SET OPTION TO NULL
LXI H,0 ; SET SKIP COUNT TO ZERO
SHLD SKFLG
LXI H,BUFF ; SCAN COMMAND LINE FOR OPTIONS
MOV A,M ; GET BYTE COUNT
INX H ; PT TO FIRST CHAR
PUSH H ; SAVE PTR
CALL ADR ; PT TO LAST CHAR
MVI M,0 ; SET EOL CHAR

```

```

POP      H      ; PT TO FIRST CHAR OF LINE
PUSH     H      ; SAVE PTR
* OBTAIN SPEC FOR CURRENT DRIVE
MVI      C,25   ; INTERROGATE DRIVE NUMBER
CALL     BDOS
ADI      'A'    ; CONVERT TO LETTER
STA      CUR$DRV ; CURRENT DRIVE SPEC
POP      H      ; GET PTR
PUSH     H      ; SAVE PTR
* SCAN FOR DRIVE SPECIFICATION
TFS$DRV MOV     A,M      ; GET BYTE
INX      H      ; PT TO NEXT
ORA      A      ; 0 MEANS EOL
JZ       TFS$DRO
CPI      ':'    ; COLON MEANS WE FOUND ONE
JNZ      TFS$DRV ; CONT IF NOT
DCX      H      ; PT TO DRIVE SPEC
DCX      H
MOV      A,M    ; GET DRIVE LETTER
STA      CUR$DRV ; LOG IN DRIVE SPECIFIED BY CHAR IN A
TFS$DRO POP     H      ; GET PTR TO 1ST CHAR
CALL     ZPRR
DB       'TFS V3.0',0DH,0AH
DB       '  AT ANY TIME, TYPE EITHER <ESC> OR CTRL-C TO ABORT',0
* CHECK FOR OPTION -- DELIMITED BY /
CALL     ZPRR
DB       'LST: DISPLAY SET BY DEFAULT',0
MOV      A,M    ; GET CHAR
ORA      A      ; 0 MEANS EOL
JZ       TFS$STRT ; DONE W/SCAN
INX      H      ; PT TO NEXT
CPI      '/'    ; OPTION FLAG
JNZ      TFS$OL
MOV      A,M    ; GET NEXT CHAR (OPTION MNEUMONIC)
CPI      'V'    ; VIEW?
JZ       TFS$OV
CPI      'P'    ; PAUSE?
JZ       TFS$OP
CPI      'S'    ; SKIP?
JZ       TFS$OS
CPI      'D'    ; DISK OUTPUT?
JZ       TFS$OD
* INVALID OPTION -- ERROR MESSAGE

```

```

CALL    ZCR
CALL    ZPRR
DB      0DH,0AH
' TEXT FORMATTING SYSTEM COMMAND LINE IS --',0DH,0AH
'   TFS [X:|FILENAME|.EXT] [/O1|/O2|/O3]',0DH,0AH
' NOTE THAT DRIVE SPECIFICATION IS OPTIONAL -- LOGGED IN '
' DRIVE IS DEFAULT',0DH,0AH
' EXTENSION IS OPTIONAL -- ".TFS." IS DEFAULT',0DH,0AH
' USER OPTIONS DEFAULT TO OUTPUT ON LST:, NO PAUSE, NO SKIP'
DB      0DH,0AH,0DH,0AH
' VALID TFS OPTIONS ARE --',0DH,0AH
' /P = PAUSE AFTER PRINTING EACH PAGE',0DH,0AH
' /S NUM = SKIP "NUM" PAGES AND START PRINTING',0DH,0AH
' /V = VIEW OUTPUT ON CON:; DELAY AFTER EACH LINE',0DH,0AH
' /D = SEND OUTPUT TO DISK',0DH,0AH
DB      0

```

```

JMP     ZBOR

```

* CHECK FOR VIEW OPTION

```

TFSOV STA OFLAG ; SET OPTION FOR VIEWING ON CRT
CALL   ZPRR ; PRINT INFO
DB     'CON: DISPLAY SELECTED',0
JMP     TFSOI

```

* CHECK FOR PAUSE OPTION

```

TFSOP STA OFLAG ; SET OPTION FOR PAUSING ON PRINTOUT
CALL   ZPRR
DB     'PAUSE SELECTED -- LST: DISPLAY SET',0
JMP     TFSOI

```

* EXTRACT PAGE NUMBER FOR SKIP OPTION

```

TFSOS LXI D,0 ; SET ACCUMULATED NUMBER TO ZERO
TFSOP2 INX H ; PT TO NEXT DIGIT
MOV     A,M ; GET DIGIT
SUI     '0' ; CONVERT TO BINARY, IF POSSIBLE
JC      TFSOP4 ; DONE IF ERROR
CPI     10 ; MUST BE LESS THAN 10
JNC     TFSOP4
MOV     B,A ; VALUE IN B
MVI     C,9 ; MULT DE BY 10
PUSH    H ; SAVE PTR TO CHAR

```

```

MOV     H,D      ; PUT DE INTO HL
MOV     L,E
DAD     D        ; HL=HL+DE
DCR     C
JNZ     TFSOP3
MOV     A,B      ; GET VALUE
ADD     L        ; ADD TO NEW NUMBER
MOV     L,A
MOV     A,H
ACT     0
MOV     H,A
XCHG
POP     H        ; DE HAS ACCUMULATED NUMBER
JMP     TFSOP2   ; RESTORE PTR TO CURRENT CHAR

TFSOP4 XCHG      ; HL HAS ACCUMULATED NUMBER
SHLD    SKFLG    ; SET SKIP FLAG
XCHG     ; HL PTS TO "ERROR" CHAR
JMP     TFSOL

* SET DISK OUTPUT
TFSOD  MVI     A,OFFH ; ENABLE DISK OUTPUT
STA     DOUT
LXI     H,FCB+1 ; PT TO FILE NAME
LXI     D,OUT$FILE+1 ; PT TO OUTPUT FILE SPEC
MVI     C,8
CALL    LCIR     ; COPY FILE NAME
CALL    ZPRR
DB      0DH,0AH,'++ TFS ++ DISK OUTPUT SELECTED'
DB      0DH,0AH,' FILE NAME: ',0
LXI     H,OUT$FILE+1 ; PRINT FILE NAME
CALL    PRFN     ; PRINT FILE NAME
CALL    ZPRR
DB      0DH,0AH,'DISK OUTPUT DRIVE (A/B/C/D)? ',0
CALL    ZIN      ; GET RESPONSE
CALL    CAPS     ; CAPITALIZE
STA     OUT$DRV  ; SET OUTPUT DRIVE
CALL    ZOUT     ; PRINT CHAR
SUI     'A'      ; OK?
CPI     4
JNC     TFSOD$ERR
JMP     TFSOL

TFSOD$ERR: CALL    ZPRR

```

PAGE 6 -- TFS.ASM -- SOURCE TO THE TEXT FORMATTING SYSTEM 25 JULY 1980

DB 0DH,0AH,'INVALID DRIVE SPECIFICATION -- REENTER',0
JMP TFSOD

```

*  INITIALIZE BUFFERS
*
TFS$STRT:
    CALL    ZCR      ; NEW LINE
    LDA     CURDRV   ; LOG IN CURRENT DRIVE
    CALL    SETDRV
    LDA     DOUT     ; GET OUTPUT INDICATOR
    ORA     A        ; 0=NO
    JZ      TFS$STRT1
    LXI     D,OUT$FILE ; ESTABLISH OUTPUT FILE
    CALL    F$OPEN

TFS$STRT1:
    CALL    CLOSE1   ; CONTINUE INITIALIZATION
    CALL    CLRR
    CALL    LOAD     ; LOAD FILE SPECIFIED IN PCB
    CALL    PAUSEM
    CALL    ZCR
    LXI     H,LBUF
    SHLD   CPOS
    CALL    CLERL
    LDA     DVA
    STA     BUFLN
    STA     LNLEN
    STA     RJFLG
    LXI     H,ZBOF
    MOV     A,M
    CPI     EOF
    RZ

    CALL    LPCLR
    XRA     A        ; A=0; SET FLAGS
    STA     R0FLG1
    STA     RPFLG
    STA     NCOPY
    STA     ULINE
    STA     LWAR
    STA     PGFLAG
    STA     ATAB
    STA     MTAB
    STA     HDPLG
    STA     FTPLG
    STA     PSKIP

```

25 JULY 1980

PAGE 8 -- TFS.ASM -- SOURCE TO THE TEXT FORMATTING SYSTEM

```
STA PSKIPX
STA CENFL
STA CENBFL
STA PGONT+1
STA A
INR BLK
STA SPONT
STA PGONT
STA NO
STA A
INR NULLS
STA DV3
LDA IDNT
STA XDNT
STA DV1
LDA NL1
STA NL1BF
STA DV2
LDA NL2
STA
```

; A=1; SET FLAGS

; A=2; SET FLAGS

; SET PARAGRAPH INDENTS

; NUMBER OF LINES OF TEXT PER PAGE

; NUMBER OF LINES LEFT ON PAGE

```

* START NEW OUTPUT LINE
TFS1  XRA  A      ; SET CHAR COUNT
      STA  NCHARS

* MAJOR WORD-ACCUMULATION LOOP
LLOOP LXI  SP,STACK ; SET STACK POINTER
      CALL ZINK
      CALL SWRD ; GET WRD
      LDA  NCHARS
      ORA  A
      JZ   PWRD ; FORCE CHAR IF BUF EMPTY
      ADD  C
      MOV  B,A ; SAVE COUNT IN B
      LDA  BUFLN ; CHECK TO SEE IF IT FITS
      CMP  B
      JNC  PWRD
      PUSH H
      CALL CTEST ; TEST FOR CENTERING AND CENTER IF SET
      CALL OUTLINE ; PRINT LINE IF CENTERING DIDN'T OCCUR
      CALL BREK1 ; RESET OUTPUT LINE
      POP  H ; RESTORE PTR
      * PLACE WORD IN OUTPUT LINE BUFFER; SET BIT 7 TO UNDERLINE
      PWRD  H
      PUSH D
      PUSH B
      CPOS ; H&L PT TO NEXT WRD SPACE
      ULIN ; GET UL
      MOV  C,A
      LXI  D,MBUF-1 ; GET CHAR COUNT
      LDAX D
      MOV  B,A
      INX  D
      NCHARS ; UPDATE LINE CHAR COUNT
      LDA  B
      ADD  A ; ADD 1 FOR <SP> AFTER
      INR  A
      STA  NCHARS
      LDAX D ; GET & STORE CHAR
      ORA  C ; UL
      MOV  M,A
      INX  D
      INX  H
      DCR  B
      JNZ  PWRD

```

```

MVI M, ' ' ; PUT <SP> IN BUFFER
INX H ; NEW CURRENT POSITION
* CHECK FOR SPECIAL CHAR FOLLOWING WORD
DCX D ; PT TO LAST CHAR OF PREVIOUS WORD
LDAX D ; CHECK FOR END
CPI ' '
JZ EXSP
CPI '?'
JZ EXSP
CPI '!'
JZ EXSP
CPI ':'
JZ EXSP
JNZ PARDLO
* PLACE 2ND SPACE AFTER SPECIAL CHAR
EXSP MVI M, ' ' ; EXTRA <SP>
INX H
LDA NCHARS ; INCR CHAR COUNT
INR A
STA NCHARS
PARDLO SHLD CPOS
POP B
POP D
POP H
JMP LLOOP

```

```

* CLEAR OUTPUT LINE BUFFER
CLERL  PUSH  H      ; CLEAR LBUF
      LXI   H,LRUF
      MVI  B,140
      MVI  M,' '
      INX  H
      DCR  B
      JNZ  CLERL1
      POP  H
      RET

* LOOK FOR NEXT WORD; IF EOL ENCOUNTERED, SKIP TO NEXT LINE;
* IF AUTOPARAGRAPHING ENGAGED, PARAGRAPH IF 1ST CHAR OF LINE IS <SP>
* ON EXIT, FWRD PTS TO NEXT WORD W/HL
FWRD  MOV  A,M      ; GET CHAR
      CPI  ODH      ; GET TO NEXT LINE IF <CR>
      JZ   FWRD0
      CPI  '+'      ; SKIP <SP> OR LESS
      RVC
      INX  H
      JMP  FWRD

* <CR> ENCOUNTERED --- CENTER IF CENTERING IN PROGRESS
FWRD0  PUSH  H      ; SAVE PTR TO <CR>
      CALL  CTEST   ; TEST FOR CENTERING AND CENTER IF SET
      POP  H        ; RESTORE PTR TO <CR>
      * <CR> ENCOUNTERED --- IF IN INPUT LINE, RETURN TO DOCUMENT BODY
      LDA  RDFLG1   ; READ FROM INPUT LINE?
      ORA  A
      JZ   FWRD01   ; CONT IF NOT
      XRA  A        ; RESET INPUT LINE READ FLAG
      STA  RDFLG1   ; RESTORE SOURCE FILE PTR
      LHLD RTEMP1   ; FIND NEXT WORD
      JMP  FWRD      ; IF IN DATA FILE, RETURN TO DOCUMENT BODY
      * <CR> ENCOUNTERED --- IF IN DATA FILE, RETURN TO DATA FILE?
FWRD01  LDA  RDFLG  ; READ FROM DATA FILE?
      ORA  A
      JZ   FWRD1    ; CONT IF NOT
      INX  H        ; PT TO 1ST CHAR OF NEXT LINE
      INX  H
      SHLD DPTR
      XRA  A        ; RESET RDFLAG
      STA  RDFLG
      LHLD RTEMP    ; RESTORE SOURCE FILE PTR

```

```

JMP FWRD ; FIND NEXT WORD
* SKIP OVER <CR> <LF>, CHECK FOR EOF, AND CHECK FOR AUTO PAR & IMPLEMENT
FWRD1 INX H ; SKIP <CR>
INX H ; SKIP <LF>
MOV A,M ; CHECK FOR EOF
CPI EOF
JZ DONE
* IF ANOTHER <CR>, SKIP ALSO
CPI ODH ; <CR>?
JZ FWRD1
* AUTOMATIC PARAGRAPHING -- SIMPLE, ISN'T IT?
LDA APFLG ; AUTOMATIC PAR ENGAGED?
ORA A ; 0=NO
JZ FWRD
MOV A,M ; GET 1ST CHAR OF LINE
CPI ' ' ; ENGAGE 8P IF LEADING <SP>
JZ FWRD2
CPI 9 ; ENGAGE 8P IF LEADING <TAB>
RNZ
* FORCE PARAGRAPH
FWRD2 CALL PAR ; 8P EQUIVALENT
JMP FWRD ; FIND PROPER WORD
* SCAN COMMAND TABLE PTED TO BY DE (CMD COUNT BYTE IS PTED TO) FOR
* CMD PTED TO BY HL; ON EXIT, DE PTS TO ADR & HL PTS TO NEXT WORD
* IF FOUND, RET W/NZ -- IF NOT FOUND, RET W/Z
SCAN LDAX D ; GET CMD CNT
MOV B,A
INX D
ORA A ; ABORT IF NONE
RZ
* START SCANNING
SCAN0 PUSH H ; SAVE PTR TO CMD
PUSH D ; SAVE PTR TO TABLE ENTRY
MVI C,CNT ; # CHARS IN CMD
XCHG
* EXTRACT COMMAND, CAPITALIZE, AND COMPARE
SCAN1 LDAX D ; COMPARE
CALL CAPS ; CAPITALIZE CMD
CMP M
JNZ SCAN3
MOV A,M
CPI ' ' ; DONE IF <SP>
JZ SCAN2

```

```

INX H
INX D
IDR C
JNZ SCAN1
* SCAN SUCCEEDED -- GET ADR OF COMMAND IN DE
SCAN2 XCHG
SCAN21 SHLD CNEXT ; SAVE PTR TO NEXT CHAR
POP H ; PT TO ADR
MVI A,CNT
CALL ADR ; COMPUTE ADR
MOV E,M ; DE PT TO ADR
INX H
MOV D,M
POP H
LHLD CNEXT ; CLEAR STACK
MVI A,1 ; HL PT TO NEXT CHAR
ORA A ; NZ MEANS FOUND
RET

```

```

* SCAN CHECK -- <SP> MATCHES <CR>
SCAN3 XCHG
LDAX D ; GET TABLE CHAR
CPI
JNZ SCAN3A
MOV A,M
CPI 0DH
JZ SCAN21 ; MATCH
* SCAN FAILED -- PT TO NEXT TABLE ENTRY
SCAN3A INX D ; PT TO ADR
DCR C
JNZ SCAN3A
INX D ; SKIP ADR
INX D
POP H ; CLEAR STACK
POP H ; PT TO SEARCH STR
DCR B
JNZ SCAN0
DCX H ; PT TO %
XRA A ; Z MEANS NOT FOUND
RET

```

```

* SET IF WORD STARTS W/% -- IF SO, IT MAY BE A COMMAND
CMFRD MOV A,M ; CMNDS START W/%
CPI CCHAR
RNZ

```

```

* '%%' IS ESCAPE SEQUENCE -- PRINT AS '%'
MOV A,M ; CHECK FOR ESC SEQ
CPI CCHAR ; '%%'=CCHAR TO PRINT
RZ

PUSH H ; SAVE PTR
* SCAN FOR MACRO NAME FIRST
LXI D,MTAB ; CHECK FOR MACRO
CALL SCAN
JZ CMD ; CHECK FOR CMD IF NOT FOUND
MOV B,H ; SAVE PTR TO NEXT CHAR
MOV C,L
POP H ; CLEAR STACK
LXI H,ATAB ; STORE IN ADR TABLE
MOV A,M ; GET NEXT LOC DISP IN A
INR M ; NEW CNT
ADD A ; DOUBLE IT
INX H ; PT TO TABLE
CALL ADR ; COMPUTE SAVE ADR
MOV M,C ; STORE RET ADR
INX H
MOV M,B
MOV H,D
MOV L,E
JMP CKRET

* CHECK FOR COMMAND
CMD LXI D,CTAB ; CHECK FOR CMD
POP H ; GET PTR
CALL SCAN ; CHECK TABLE
RZ ; RETURN IF NOT FOUND
LXI B,CKRET ; SET UP RET ADR
PUSH B
PUSH D ; RUN FROM STACK
RET ; HL PTS TO NEXT CHAR IN FILE
* RETURN HERE AFTER COMMAND/MACRO LOCATED
CKRET POP D ; CLEAR STACK
* SEARCH FOR NEXT WORD; IF COMMAND/MACRO, EXECUTE IT AND SEARCH AGAIN;
* IF NOT COMMAND/MACRO, LOAD IT INTO WBUF; ON EXIT, HL PTS TO WORD AFTER
* FOUND WORD
SMRD CALL FWRD ; HL PTS TO NEXT CHAR
CALL CKWRD ; CHECK FOR TFS COMMAND
SMRDO LXI D,WBUF
MVI C,0 ; INIT CHAR COUNT

```

```

SWORD MOV     A,M
STAX   D      ; PUT IN BUFFER
CPI    21H    ; STOP IF < 21H
JC     SWORD1
INX    D
INX    H
INX    C      ; INCR COUNT
JMP    SWORD1
MVI    A,' '  ; STORE <SP>
STAX   D
MOV    A,C    ; STORE CHAR COUNT
STA    WBUF-1
RET

* END OF PROCESSING
DONE   CALL   PAGE
JMP    ZBOR

```

```

* OUTPUT LINE IN LBUP; RIGHT JUSTIFY IF DESIRED
OUTLN LDA NCHARS ; GET CHAR CNT
ORA A ; NONE?
RZ
MOV B,A
* IF LINE IN LBUP BEYOND LIMITS SPECIFIED, PRINT IT W/O JUSTIFICATION
LDA BUFLN ; CHECK FOR WITHIN LIMITS
SUB B
CPI CLIM ; SEE IF MORE THAN CLIM CHARS
JNC PRINT ; PRINT IF SO
* OUTPUT LINE
OUTLN LDA NCHARS ; LINE PRESENT?
ORA A
RZ
* CHECK FOR RIGHT JUSTIFICATION, AND PRINT IF NOT
LDA RJFLG ; RJ?
ORA A ; ZERO MEANS NO
JZ PRINT
* RIGHT JUSTIFY LINE
LDA NCHARS ; GET CHAR CNT (ENTRY POINT)
MOV B,A
LXI H,LBUP
CALL ADR ; HL PT TO END OF STR
DCX H
MOV A,M ; FIND LAST NON-BLANK CHAR
CPI ,
JNZ ENDL1
DCX H
DCR B
JMP ENDL
MOV A,B ; NEW CHAR CNT
STA NCHARS
SHLD CPOS ; STORE CURR POS
LDA BUFLN ; CHECK FOR WITHIN BOUNDS
SUB B
JC PRINT
JZ PRINT
MOV B,A ; EXTRA <SP> CNT IN B
MTI C,0
LXI D,LBUP ; COUNT # <SP> IN LINE
LDAX D
CPI ,

```

```

ONTLO      JNZ      ONTLO
           INR      C
           INX      D
           MOV      A,D
           CMP      H
           JNZ      ONTLP
           MOV      A,E
           CMP      L
           JNZ      ONTLP
           MOV      A,C
           STA      VLOOP
           MOV      A,B
           MVI      D,0
           SUB      C
           JNC      SPLP1
           INR      D
           INR      C
           JMP      SPLP
           ADD      C
           MOV      E,A
           MOV      A,D
           ORA      E
           JZ      PRINT
           MOV      B,D
           MOV      C,E
           LHLD     CPDS
           LXI      D,LBUF
           LDA      BUFLN
           XCHG
           CALL     XCHG
           DCX      D
           MOV      A,M
           STAX     D
           DCX      H
           DCX      D
           CPI      ' '
           JNZ      FLOOP
           MOV      A,B
           ORA      A
           JZ      FLP1
           STA      TEMP
           MVI      A,' '
           STAX     D
           ; END OF LINE?
           ;
           ; COMPUTE # EXTRA <SP> BET EA CHAR
           ; # EXTRA <SP>'S
           ; INCR COUNT
           ; CONT
           ; D=# CHRS BET EA, E=# EXTRA CHRS
           ; CHECK FOR NO EXTRA <SP>
           ; # CHARS BET EACH WORD
           ; # EXTRA CHARS
           ; DE PT END LBUF, HL PT END CUR LN
           ; GET CHAR
           ; STORE CHAR
           ; CHECK FOR <SP>
           ; LOOP FOR B CHARS
           ; STORE <SP>
           ; STORE EXTRA <SP>

```

```

DCX      D
IER      B
JNZ      FLP0
LDA      TEMP
MOV      B,A
MOV      A,C
ORA      A
JZ       FLP2
MVI      A,' '
STAX     D
IEC      D
IER      C
LDA      VLOOP
DCR      A
STA      VLOOP
JNZ      FLOOP
LDA      BUFLN
STA      NCHARS
* PRINT LINE CONTAINED IN BUFLN ON PROPER OUTPUT DEV (CON: IF /N OPTION,
* LST: OTHERWISE)
PRINT    LDA      NCHARS
MOV      B,A
LXI      H,BUFL
XRA      A
STA      ULFLG
MOV      A,M
POUT     ; GET CHAR
CALL     POUT   ; PRINT
ANI      80H    ; CHECK FOR UL
JZ       PRINL1
MOV      A,M
CPI      0A0H
JZ       PRINL1
MVI      A,1
STA      ULFLG
PRINL1   INX     H
DCR      B
JNZ      PRINT
* CHECK FOR UNDERLINING
LDA      ULFLG ; CHECK FOR UL
ORA      A
JZ       CRLF
* UNDERLINING TO BE DONE -- DO IT
LDA      OFLAG ; CHECK FOR CRT VIEW -- PAUSE IF SO

```

```

CPI      'V'      ; VIEW?
JNZ      ULLP
CALL     SYCHECK  ; IF SKIPPING, CONTINUE
JNZ      ULLP
CALL     ZESC     ; ESCAPE
LXI      H,LEUP  ; DO UL
IDA      NCHARS  ; GET NO CHARS
PUSH     PSW     ; SAVE CHAR CNT
* BACKSPACE TO BEGINNING OF LINE
MOV      B,A
MVI      A,8     ; <BS>
CALL     POUT    ; PRINT <BS>
DCR      B       ; COUNT DOWN
JNZ      ULLPBS
POP      PSW     ; GET CHAR CNT
* ADVANCE FORWARD, UNDERLINING AS YOU GO
MOV      B,A
PUSH     H
MOV      A,M     ; GET CHAR
INX      H
ANI      80H
JZ       ULLPC
MOV      C,B     ; COUNT IN C
DCR      B
JNZ      ULLPO
LDA      NCHARS
SUB      C
INR      A
MOV      B,A     ; ACTUAL COUNT IN B
POP      H       ; RESTORE PTR
MOV      A,M     ; GET CHAR
INX      H
CPI      0A0H    ; CHECK OF UL <SP>
JZ       ULLP2
ANI      80H
JZ       ULLP2
MVI      A,5FH  ; CHECK FOR UL
JMP      UL1PT
MVI      A,' '  ; <SP>
CALL     POUT
DCR      B
JNZ      ULLP1
* <CR> <LF>, MULTIPLE SPACING, AND PAGE IF REQUIRED

```

```

CRLF CALL CR          ; SPACING
      LDA SPONT
      ORA A
      RZ

```

```

* SPACE DOWN APPROPRIATELY
      MOV C,A          ; CNT IN C
      CALL LF
      LDA NL1BF        ; GET LINE CNT
      DCR A
      STA NL1BF
      JZ CRLF1
      DCR C
      RZ
      JMP CRLF0

```

```

* PAGE
CRLF1 LDA NL2          ; SPACE OUT PAGE
      MOV B,A

```

```

* PRINT FOOTER IF REQUESTED
      LDA FTFLG        ; FOOTER?
      ORA A
      JZ SPGL
      DCR B
      CALL LF
      PUSH B
      PUSH H
      WBSAV
      CALL PFOOT
      CALL WRES
      CALL CR
      POP H
      POP B

```

```

* PRINT PAGE NUMBER IF REQUESTED
SPGL  LDA PJFLAG       ; PAGE?
      ORA A
      JZ SPGL0
      DCR B
      CALL LF
      PUSH B
      LDA LNLEN        ; PUT '##'
      SUI 2            ; TWO IN
      MOV B,A
      MVI A,' '
      CALL POUT

```

```

DCR      B
JNZ      PGSPL
CALL     PRPGCNT ; PRINT PAGE NUMBER (PGCNT BUFFER)
CALL     CR
POP      B
CALL     LP
DCR      B
JNZ      SPPGLO

```

```

SPPGLO   * CHECK FOR PAUSE ON OUTPUT OPTION
          LDA     OFLAG ; CHECK FOR PAGING
          CPI     'P'
          JNZ     SPPG1
          CALL    SKCHK  ; CHECK FOR SKIP; RET W/NZ IF YES
          JNZ     SPPGL1
          CALL    ZPRR
          DB      'PLEASE INSERT NEXT PAGE',0
          CALL    ZESC  ; ESCAPE
          CALL    ZCR
          LDA     NLI   ; RESET CNT
          STA     NL1BF
          PUSH    H
          LHL    PGCNT ; SAVE HL
          INX     H      ; INCR PAGE COUNT
          SHLD   PGCNT
          POP     H
          CALL    SKCHK  ; CHECK FOR SKIP; RET W/NZ IF YES
          JZ      PGSPL1
          PUSH    H
          LHL    SKFLG ; SAVE HL
          DCX     H      ; GET SKIP COUNT
          SHLD   SKFLG
          POP     H
          PUSH    H
          CALL    WBSAV

```

```

          * PRINT HEADING OF NEXT PAGE IF REQUESTED
          LDA     HDPLG ; HEADING?
          ORA     A
          CNZ     PHEAD
          CALL    WBRES
          JMP     PHEAD1
          * SKIP CHECK UTILITY; RET W/NZ IF SKIPPING IN PROGRESS
          SKCHK  PUSH    H ; SAVE HL
          LHL    SKFLG ; GET SKIP FLAG

```

```

          SPPGL1
          STA     NL1BF
          PUSH    H
          LHL    PGCNT ; SAVE HL
          INX     H      ; INCR PAGE COUNT
          SHLD   PGCNT
          POP     H
          CALL    SKCHK  ; CHECK FOR SKIP; RET W/NZ IF YES
          JZ      PGSPL1
          PUSH    H
          LHL    SKFLG ; SAVE HL
          DCX     H      ; GET SKIP COUNT
          SHLD   SKFLG
          POP     H
          PUSH    H
          CALL    WBSAV

```

```

          * PRINT HEADING OF NEXT PAGE IF REQUESTED
          LDA     HDPLG ; HEADING?
          ORA     A
          CNZ     PHEAD
          CALL    WBRES
          JMP     PHEAD1
          * SKIP CHECK UTILITY; RET W/NZ IF SKIPPING IN PROGRESS
          SKCHK  PUSH    H ; SAVE HL
          LHL    SKFLG ; GET SKIP FLAG

```

```

          SPPGL1
          STA     NL1BF
          PUSH    H
          LHL    PGCNT ; SAVE HL
          INX     H      ; INCR PAGE COUNT
          SHLD   PGCNT
          POP     H
          CALL    SKCHK  ; CHECK FOR SKIP; RET W/NZ IF YES
          JZ      PGSPL1
          PUSH    H
          LHL    SKFLG ; SAVE HL
          DCX     H      ; GET SKIP COUNT
          SHLD   SKFLG
          POP     H
          PUSH    H
          CALL    WBSAV

```

```

          * PRINT HEADING OF NEXT PAGE IF REQUESTED
          LDA     HDPLG ; HEADING?
          ORA     A
          CNZ     PHEAD
          CALL    WBRES
          JMP     PHEAD1
          * SKIP CHECK UTILITY; RET W/NZ IF SKIPPING IN PROGRESS
          SKCHK  PUSH    H ; SAVE HL
          LHL    SKFLG ; GET SKIP FLAG

```

```

MOV     A,H
ORA     L
POP     H          ; GET HL
RET

* PRINT FOOTER
PFOOT LXI H,FTBUF ; PRINT FOOTER
      JMP PHEADS

* PRINT HEADER
PHEAD LXI H,HBUF  ; PRINT HEADING IN BUFFER
PHEADS MVI A,1    ; SET TAB CNT
      STA TABC
PHEAD0 MOV A,M    ; GET CHAR
      CPI ' '
      RC
      CCHAR      ; COMMAND?
      JNZ PHEADP
      INX H      ; COMMAND NAME?
      MOV A,M
      CALL CAPS  ; CAPITALIZE
      INX H
      * PRINT PAGE NUMBER?
      CPI '1'    ; PAGE NUMBER?
      JNZ PH1
      CALL PRGCNT ; PRINT PAGE NUMBER
      JMP PHEAD0

* DISPLAY REGISTER?
PH1    CPI 'D'
      JNZ PH2
      CALL GETNUM
      MOV C,A
      CALL LOCR
      * PRINT NUMBER IN A REG AS DECIMAL
PHEAD0 PUSH H
      CALL DOT2A ; PRINT
      LDA TABC
      INR A
      INR A
      STA TABC
      POP H
      JMP PHEAD0

* DISPLAY AND INCREMENT REGISTER?
PH2    CPI 'R'
      JNZ PH3

```

```

CALL GETNUM
MOV C,A
CALL LOCKR
MOV B,A
CALL INCR2
MOV A,B
JMP PHEAD0
* TABULATE?
PH3 CPI 'T'
JNZ PHE
CALL GETNUM
MOV C,A
LDA TABC
MOV B,A
CMP C
JNC PHEAD0
MOV A,C
STA TABC
MVI A,' '
CALL POUT
INR B
MOV A,C
CMP B
JNZ PH3L
JMP PHEAD0
PHE DCX H
MOV A,M
CALL POUT
LDA TABC
INR A
STA TABC
INX H
JMP PHEAD0
PHEAD1 CALL CRLF
CALL CRLF
POP H
RET
* PRINT PAGE NUMBER AS 4 DECIMAL DIGITS
PRPGCNT PUSH H
LHLD PGCNT
XRA A
STA LSPFL
LXI D,-1000

```

; PT TO \$CHAR

; PRINT CHAR

; TAB

; SAVE HL

; GET PAGE NUMBER

; SET LEADING <SP> FLAG

; COMPUTE 1000'S

```

CALL PRPGC
LXI D,1000 ; ADD IN
DAD D
LXI D,-100 ; COMPUTE 100'S
CALL PRPGC
LXI D,100 ; ADD IN
DAD D
LXI D,-10 ; COMPUTE 10'S
CALL PRPGC
LXI D,10 ; ADD IN
DAD D
MOV A,L ; COMPUTE 1'S
ADI '0' ; ASCII BIAS
CALL POUT
POP H ; GET HL
RET

* ADD DE TO HL UNTIL HL BECOMES NEG OR ZERO; PRINT COUNT OR LEADING <SP>
PRPGC MVI B,0 ; COUNT
PRPGC1 DAD D ; ADD IN DE
MOV A,H ; NEG?
CPI OF8H
JNC PRPGC2
INR B ; INCR COUNT
JMP PRPGC1
PRPGC2 LDA LSPFL ; GET LEADING <SP> FLAG
ORA B ; CHECK FOR LEADING <SP> AND ZERO COUNT
STA LSPFL ; NEW LEADING <SP> FLAG
JNZ PRPGC3
MVI A,' ' ; PRINT LEADING <SP>
JMP POUT
PRPGC3 MOV A,B ; GET COUNT
ADI '0' ; ASCII BIAS
JMP POUT

* WORD BUFFER SAVE UTILITY
WBSAV LXI H,WBUF-1
LXI D,WBUF
MVI C,41
JMP LCTR

* WORD BUFFER RESTORE UTILITY
WGRES LXI H,WBUF
LXI D,WBUF-1
JMP WBS1

* PRINT STRING PTED TO BY HL AND ENDING IN 0 ON PROPER DEVICE

```

```

PHL    MOV    A,M      ; GET CHAR
ORA    A
RZ      ; 0 AT END=DONE
INX     H
CALL    POUT          ; PRINT CHAR
JMP     PHL

* DO JUST <LF>
LF      MVI     A,0AH
JMP     POUT

* DO JUST <CR>
CR      MVI     A,0DH
CALL    POUT
LDA     NULLS        ; <NULLS> OUTPUT
ORA     A
RZ

MOV     B,A
XRA     A
CALL    POUT
DCR     B
JNZ     NULLO
RET

*
* READLN — READS AN INPUT LINE FROM THE CONSOLE AND PREPARES
* IT FOR USAGE BY TFS
*
READLN  LXI     D,INLINE      ; PT TO BUFFER
MVI     A,INLEN      ; PLACE LENGTH OF LINE IN BUFFER
STAX    D
MVI     C,10         ; CP/M READ BUFFER ROUTINE
PUSH    D            ; STACK HOLDS PTR TO 1ST BYTE OF BUFFER
CALL    BDOS
POP     H
INX     H            ; HL PTS TO 1ST BYTE OF BUFFER
MOV     A,M          ; PT TO CURRENT BUFFER LENGTH
INX     H            ; ... IN A
ADD     L            ; PT TO 1ST BYTE OF TEXT
MOV     L,A
MOV     A,H
ACI     0
MOV     H,A
MVI     M,0DH        ; COMPUTE PTR TO 1 BYTE AFTER LAST BYTE OF TEXT
INX     H
MVI     M,0AH        ; PLACE <CR> <LF> AFTER THE LINE

```

```
CALL ZCR ; NEW LINE
RET
* TRAP UTILITY FOR INVALID INPUT LINE COMMANDS
RDFCK1 LDA RDFLG1 ; INPUT LINE READ CHECK
ORA A
RZ
CALL PERR
DB 'INVLD CMD IN KEYBOARD INPUT LINE',0
JMP ZERR
```

*
* TFS COMMAND SECTION
*

```
*%KB -- INPUT AND PROCESS LINE FROM KEYBOARD
KB  CALL ZCR ; NEW LINE
* PRINT TEXT OF MESSAGE TO CONSOLE
KB1 MOV A,M ; GET CHAR
CPI 0DH ; DONE?
JZ KB2
CALL ZOUT
INX H ; PT TO NEXT
JMP KB1
* SET SOURCE FILE PTR, READ LINE, AND PT TO 1ST CHAR OF LINE
KB2 SHLD RTEMP1 ; SET SOURCE FILE PTR
MVI A,OFFH ; TURN ON INPUT LINE READ FLAG
STA R0FLG1
CALL ZPRR ; PRINT INPUT PROMPT
DB 'INPUT: ',0
CALL READLN ; READ INPUT LINE FROM CONSOLE
LXI H,INLINE+2 ; PT TO FIRST CHAR TYPED
RET
```

```
*%SP -- SET INTER-LINE SPACING
SPSET CALL GETNUM
STA SPCNT
RET
```

```
*%CH -- START NEW CHAPTER
CHAP CALL GETNUM ; SAVE PTR
PUSH H ; SAVE #
PUSH PSW ; SAVE #
CALL PAGE
CALL CR ; <CR>
MVI B,10 ; DOWN 10
LDA NL1BF ; CORRECT CNT
SUB B
STA NL1BF
MVI A,0AH ; <LF>
CALL POUT
DCR B
```

```

JNZ      CHAPL
LDA      BUFLN      ; CENTER
SUI      10         ; COMPUTE OFFSET
CALL     CENCNT     ; COMPUTE <SP> CNT
LDA      LMAR       ; GET LEFT MARGIN
ADI      1          ; DIVIDE BY 2
RAR

ANI      7FH        ; MAKE SURE MSB NOT SET
ADD      B          ; ADD IN <SP> CNT
MOV      B,A        ; NEW <SP> CNT
MVI      A,' '
CALL     POUT
DCR      B
JNZ      CHAPC
CALL     PRVTP
DB       'CHAPTER ',0
POP      PSW        ; GET NUM
CALL     DOT2
CALL     CRLF
POP      H          ; GET PTR
JMP      CEN        ; CENTER CHAPTER TITLE

```

```

*UL -- UNDERLINE TOGGLE
UL      IDA         ; GET UNDERLINE FLAG
ORA      A          ; CHECK FOR ZERO
JZ       UL1
XRA      A          ; SET A ZERO
STA      UL1
RET

UL1     MVI      A,80H ; SET UL FLAG
JMP      UL0

```

```

*AP -- AUTOMATIC PARAGRAPH TOGGLE
APAR    LDA      APFLG ; TOGGLE AUTO PAR FLAG
CMA
STA      APFLG
RET

```

```

*ASIS -- ENTER ASIS MODE
ASIS     CALL     RDEFCK
CALL     RDEFCK1 ; ASIS MAY NOT BE IN DATA FILE OR INPUT LINE
CALL     BREX
ASIS00   MOV      A,M ; SKIP TO NEXT LINE

```

```

INX      H
CPI      0DH      ; <CR>
JNZ      ASIS00
INX      H
MOV      A,M
CPI      EOF
JZ       DONE
CPI      CCHAR
JZ       ; ESCAPE CHAR IS CCHAR
LXI      D,LBUF
MVI      B,0
MOV      A,M
CPI      0DH
JZ       ASIS2
INX      H
INR      B
CPI      9
JZ       ASIS1
STAX     D
INX      D
JMP      ASIS1
MVI      A,' '
STAX     D
INX      D
MOV      A,B
ANI      7
JZ       ASIS1
INR      B
JMP      ASIS1
MOV      A,B
INR      A
STA      NCHARS
MVI      A,' '
STAX     D
PUSH     H
CALL     PRINT
CALL     BREK1
CALL     ZINK
POP      H
JMP      ASIS00
MOV      A,M
INX      H
CPI      0DH

```

ASIS0 ; PT TO 1ST CHAR OF NEXT LINE
; END OF FILE?
; ESCAPE CHAR IS CCHAR
; SEND LINE TO BUFFER
; INIT CHAR CNT
; GET CHAR
; DONE?

ASIS1 ; INCR CHAR CNT
; <TAB>?
; STORE CHAR FOR OUTPUT

ASIS2 ; <TAB> ENCOUNTERED -- <SP> APPROPRIATELY
; CHAR CNT
; EVERY 8
; INCR CHAR CNT
; COUNT TRAILING <SP>
; CHAR COUNT
; PLACE A <SP> AT EOL

AIRET ; PRINT LINE
; RESET LINE
; INT CHECK
; FIND <CR>

```

JNZ  AIR$T
DCX   H      ; PT TO <CR>
XPA   A      ; RESET CHAR CNT
STA   NCHARS
JMP   FWRD

```

*BREM -- SKIP TO NEXT LINE OF SOURCE; ALSO A UTILITY

```

NXTLN  MOV   A,M      ; FIND <CR>
      INX   H
      CPI   00H
      JNZ   NXTLN
      DCX   H      ; PT TO <CR>
      JMP   FWRD    ; DO FIND WORD PROCESSING

```

* TEST OF CENTERING AND CENTER IF SET

```

CTEST  LDA   CENFL    ; %C?
      ORA   A
      JNZ   CENTER
      LDA   CENBFL    ; %CB?
      ORA   A
      RZ

```

* CENTER ROUTINE FOR %C (CENTER) COMMAND; ALSO A UTILITY

```

CENTER XRA   A      ; TURN OFF CENTERING
      STA   CENFL
      CALL  CENLO
      JMP   BREK1    ; CENTER LINE
      LDA   NCHARS   ; BREAK FOR NEW LINE
      ORA   A        ; GET NUM OF CHARS IN LINE
      RZ            ; NONE?

```

```

      MOV   C,A      ; ... IN C
      LDA   BUFLN    ; LENGTH OF PHYSICAL LINE
      SUB   C
      JC    PRINT
      JZ    PRINT

```

* COMPUTE CENTERING <SP> COUNT

```

      CALL  CENCNT
      MVI   A,' '
      CALL  POUT
      DCR   B
      JNZ   CENL1
      JMP   PRINT

```

* COMPUTE CENTERING <SP> COUNT

```

      RAR
      ANI   7FH
      MOV   B,A      ; DIVIDE BY 2
      MOV   B,A      ; MAKE SURE DIVISIBLE BY 2
      MOV   B,A      ; COUNT IN B

```

25 JULY 1980

RET

```
*EXIT, $STOP, $HALT -- RETURN TO OPERATING SYSTEM
EXIT  CALL  PAGE
      JMP  ZEOR
```

```
*C -- BREAK LINE AND CENTER ONE LINE
CEN  CALL  BREK
      MVI  A,1      ; SET FLAG
      STA  CENFL
      RET
```

```
*CB -- BREAK LINE AND TOGGLE BLOCK CENTERING
CB  CALL  BREK
    LDA  CENBFL    ; GET FLAG
    CMA           ; TOGGLE
    STA  CENBFL
    RET
```

```
*FOOT -- DEFINE PAGE FOOTER
FOOT INX  H      ; SET UP FOOTER
      LXI  D,FTBUF
      LDA  HDFLG  ; SAVE HDFLAG
      STA  TMP
      CALL HEADC
      LDA  TMP
      STA  HDFLG
      MVI  A,1    ; SET FOOT FLAG
      STA  FTFLG
      RET
```

```
*HEAD -- DEFINE PAGE HEADER
HEAD INX  H      ; PT TO 1ST CHAR
      LXI  D,HBUF
      MOV  A,M    ; STORE LINE
      STAX D
      INX  H
      INX  D
      CPI  0DH
      JNZ  HEADC
      DCX  H      ; PT TO <CR>
      DCX  D
      XRA  A      ; LAST CHAR=0
      RET
```

```

STAX D
MVI A,1 ; SET HEAD FLG
STA HDFLG
RET

```

```

*1CR -- CARRIAGE RETURN
CARET LDA NCHARS ; CHECK FOR ZERO CHARS
ORA A
JZ CRLF ; JUST <CR> IF NO LINE

```

```

*1BR -- BREAK LINE
BREAK LDA NCHARS ; CHECK FOR ZERO CHARS
ORA A
JZ BREAK1

```

```

PUSH H
CALL CTEST ; CENTER IF ENGAGED
CALL OUTLN ; PRINT CURRENT LINE IF NOT CENTERED
POP H
PUSH H ; SAVE PTR TO NEXT WORD
XRA A ; RESET NO CHARS

```

```

STA NCHARS
LXI H,LRUP
SHLD CPOS ; RESET CUR POS IN OUTPUT LINE
CALL CLERL ; CLEAR OUTPUT LINE
CALL LMARG ; SET LEFT MARG
POP H
RET

```

```

*1RJ -- SET RIGHT JUSTIFICATION ON
RJ MVI A,1 ; SET RJ FLAG
STA RJFLG
RET

```

```

*1WORJ -- SET RIGHT JUSTIFICATION OFF
NORJ XRA A ; CLEAR RJ FLAG
STA RJFLG
RET

```

```

*1PAUS -- PAUSE FOR OPERATOR ACTION
PAUSE MOV A,M ; GET CHAR
CPI ODH ; EOL?
JZ PAUS1
INX H ; PT TO NEXT

```

```

CALL      ZOUT      ; PRINT MSG
JMP       PAUSE
CALL      PAUSM
JMP       ZCR
CALL      ZPRR
DB        ' TYPE ANY CHAR WHEN READY, <ESC> OR CTRL-C TO ABORT',0
CALL      ZESC      ; ESCAPE
RET

```

```

*1ENDM -- END OF MACRO; RETURN TO NORMAL PROCESSING FLOW
M1ENDM
CALL      RDECK
CALL      RDECK1    ; ENDM MAY NOT BE IN DATA FILE OR INPUT LINE
LXI       H,ATAS    ; GET RET ADR
MOV       A,M        ; CNT IN A
ORA       A
JZ        M1ENDE
DCR       A          ; SET NEXT ADR
MOV       M,A
ADD       A          ; DOUBLE A FOR ADR
INX       H
CALL      ADR        ; HL PT TO RET ADR
MOV       E,M
INX       H
MOV       D,M        ; DE=NEW HL
XCHG      ; HL LOADED
RET
M1ENDE
CALL      PERR
DB        'MACRO RET ERR',0
JMP       ZBOR       ; RETURN TO OS

```

```

*1P -- START NEW PARAGRAPH; ALSO USED FOR AUTOMATIC PARAGRAPHING
PAR
CALL      BREX      ; BREAK CURRENT LINE
LDA       PSKIP     ; SKIP LINES BET PARS
ORA       A
JZ        PAR1
STA       TMP
CALL      CRLF
LDA       TMP
DCR       A
STA       TMP
JNZ       PARSL
LDA       IDNT      ; GET # CHARS IN INDENT
ORA       A

```

```

RZ      MOV     B,A      ; CNT IN B
LDA     NCHARS
ADD     B      ; NEW CHAR CNT
STA     NCHARS
H
LHLD    CPOS      ; GET CURRENT POSITION
MVI     M,0A0H    ; SIGNIFICANT <SP>
INX     H
DCR     B
JNZ     PARL
SHLD    CPOS
POP     H
RET

```

```

*PAGE -- START NEW PAGE
PAGE    CALL    BREK
LDA     NL1BF      ; SKIP DOWN PAGE
MOV     B,A
XRA     A
STA     NL1BF
MVI     A,0A0H    ; <LF>
CALL    POUT
DCR     B
JNZ     PAGEL
JMP     CRLF1

```

```

*TP -- TEST FOR SPECIFIED NUMBER OF LINES LEFT ON PAGE
TPG     CALL    BREK
CALL    GETNUM
MOV     B,A
LDA     NL1BF      ; COMPARE AGAINST LINES LEFT
SUB     B
RNC
JMP     PAGE

```

```

* LOOP FLAG CLEAR UTILITY
LPCLR   XRA     A      ; CLEAR LOOP FLAGS
STA     CLFLG
STA     LLFLG
RET

```

```

*COPY -- SET START OF COPY LOOP
COPY    CALL    ROPCK

```

```

CALL RORCK1 ; COPY MAY NOT BE IN DATA FILE OR INPUT LINE
CALL GETNUM ; DEF COPY BLOCK
STA NCOPY ; # COPIES
CALL FARD ; FIND NEXT WORD
SHLD CPYLP ; SAVE PTR
MVI A,1 ; SET COPY LOOP ON FLAG
STA CLFLG
RET

```

```

*%ENDC -- END OF COPY LOOP
COPYEND LDA NCOPY ; CHECK CNT
ORA A
RZ
DCR A
STA NCOPY ; NEW CNT
RZ
LDA CLFLG ; CHECK COPY LOOP ON FLAG
ORA A
JZ LPERR
LHLD CPYLP ; PT TO LOOP ADR
RET

```

```

*%LOOP -- SET START OF INFINITE (TERMINATED BY %LEX) LOOP
LOOPLP CALL RORCK
CALL RORCK1 ; LOOP MAY NOT BE IN DATA FILE OR INPUT LINE
SHLD LPLP ; SAVE PTR TO LOOP
MVI A,1 ; SET LOOP LOOP ENGAGED FLAG
STA LLFLG
RET

```

```

*%ENDL -- END OF INFINITE LOOP
LPEND SHLD LPEX ; SAVE PTR TO LOOP END
LDA LLFLG ; CHECK LOOP LOOP ON FLAG
ORA A
JZ LPERR
LHLD LPLP ; GET PTR TO LOOP START
RET

```

```

*%LEX -- EXIT INFINITE LOOP
LPEXIT LDA LLFLG ; CHECK LOOP LOOP ON FLAG
ORA A
JZ LPERR
LDA RDFLG1 ; IN INPUT LINE?

```

```

ORA      A
JZ       LPEX01
XRA      A
STA      RDFLG1 ; NOT IN INPUT LINE ANYMORE
JMP      LPEX1
LPEX01   LDA      RDFLG ; IN DATA FILE?
ORA      A
JZ       LPEX1
XRA      A
STA      RDFLG
MOV      A,M     ; NOT IN DATA FILE ANYMORE
INX      H       ; SKIP TO EOL OF DATA
CPI      ODH
JNZ      LPEX0
SHLD     DPTR    ; NEW DATA FILE PTR
LHLD     LPEX    ; GET PTR TO LOOP EXIT
RET
LPERR    CALL     PERR
DB       'LOOP ERR',0
JMP      ZERR

```

*SKIP -- SKIP SPECIFIED NUMBER OF LINES

```

CALL     BREK
CALL     GETNUM ; GET # LINES
MOV      B,A
LDA      NLBF   ; COMPUTE DIFF
SUB      B
JC       PAGE
JZ       PAGE
STA      NLBF   ; NEW CNT
MVI      A,04H ; A=<LF>, B=# LINES TO SKIP
CALL     POUT
DCR      B
JNZ      SKIPL
RET

```

*SAVE -- SAVE WORKING ENVIRONMENT

```

SAV      PUSH     H
          LXI      H,ENVIR1 ; MOVE ENVIRONMENT 1 TO 2
          LXI      D,ENVIR2
          MVI      C,ENLEN ; NUMBER OF BYTES IN ENVIRONMENT
          CALL     LCIR  ; COPY HL TO DE FOR C BYTES
          POP      H

```

RET

*RES -- RESTORE WORKING ENVIRONMENT FROM PREVIOUS SAVE
 RES PUSH H
 LXI H,ENVIR2 ; MOVE ENVIRON 2 TO 1
 LXI D,ENVIR1
 JMP SRMOV

*1PX -- START EXTENDED PARAGRAPH
 PARX CALL BREX ; BREAK CURRENT LINE
 LDA PSKIPX ; SKIP LINES BET PARS
 ORA A
 JZ PARX1

PARXSL STA TEMP
 CALL CRLF
 LDA TEMP
 DCR A
 STA TEMP
 JNZ PARXSL
 LDA XDNT
 ORA A
 RZ
 MOV B,A
 LDA NCHARS
 SUB B
 JC PARXE
 STA NCHARS
 PUSH H
 LHLD CPOS
 DCX H
 DCR B
 JNZ PARXL
 SHLD CPOS
 POP H
 RET

PARX1 ; GET # CHARS TO XDENT
 A
 B,A
 NCHARS
 ; NEW CHAR CNT FOR LINE
 B
 PARXE
 NCHARS
 ; GET CURRENT POSITION
 ; BACK UP POINTER

PARXL ; NEW CURRENT POSITION
 CPOS
 ; GET CURRENT POSITION
 ; BACK UP POINTER
 B
 PARXL
 CPOS
 ; NEW CURRENT POSITION
 H
 RET
 H
 PERR
 CALL DB 'XDENT ERR',0
 POP H
 RET

*1PAR -- SET NORMAL PARAGRAPH PARAMETERS (INDENT, NUM LINES BET PARS)
 STPAR CALL GETNUM ; GET NUMERIC VALUE

```

STA      IDNT      ; SET INDENT COUNT
CALL     GETN JM
STA      PSKIP     ; SET NO OF LNS BET PAR TO SKIP
RET

```

```

*8PARX -- SET EXTENDED PARAGRAPH PARAMETERS
STPARX  CALL      GETNUM
STA      XONT      ; SET XDENT COUNT
CALL     GETNUM
STA      PSKIPX    ; SET NUMBER OF LINES TO SKIP
RET

```

```

*8LMAR -- SET LEFT MARGIN
STLMAR  CALL      GETNUM
STA      LMAR      ; SET LEFT MARGIN
RET

```

```

*8LLEN -- SET LINE LENGTH
STLLEN  CALL      GETNUM
MOV      B,A        ; SAVE IN B
LDA      LMAR      ; ADD TO LEFT MARG
ADD      B
STA      BUFLN     ; SET LENGTH OF LINE
RET

```

```

*8PGON -- TURN ON PAGE NUMBERING
STPAGE  MVI        A,1 ; TURN ON PAGE NUMBER FLAG
STA      PGFLAG
PUSH     H
LXI      H,1        ; SET NEXT PAGE TO BE 2
SHLD     PGONT
POP      H
CALL     GETNUM     ; GET POS
STA      LLEN      ; LINE LENGTH
RET

```

```

*8PGOF -- TURN OFF PAGE NUMBERING
PAGEOF  XRA        A ; TURN OFF PAGE
STA      PGFLAG
RET

```

```

*8PNJM -- SET NUMBER OF NEXT PAGE
STPNJM  CALL      GETNUM

```

```

XCHG          ; GET PAGE NUM IN HL
SHLD          PGCNT
XCHG          ; RESTORE HL
RET

*3LINE -- SET NUMBER OF TEXT/PHYSICAL LINES ON A PAGE
STLINE CALL GETNUM
STA NL1      ; NUMBER OF TEXT LINES ON A PAGE
STA NL1BF   ; RESET PAGING
MOV B,A      ; CNT IN B
PUSH B
CALL GETNUM
POP B
SUB B        ; REMAINDER IN B
STA NL2      ; NUMBER OF EXTRA LINES ON PG
CALL ZPRR
DB 'NEW PAGE SETTING -- PLEASE SET TOP OF FORM',0
CALL ZESC   ; ESCAPE
CALL ZCR
RET

```

```

* DECIMAL PRINT UTILITY -- PRINT A REG AS 2 DECIMAL DIGITS
DOT2 CALL DOT2A
JMP CRLF
DOT2A MVI B,10
MVI C,0
DOT2L SUB B
JC DOT1A
JZ DOT1
INR C
JMP DOT2L
DOT1 INR C
JMP DOT1B
DOT1A ADD B
DOT1B MOV B,A
MOV A,C
ADI '0'
CPI '0'
JNZ DOT1C
MVI A,'.'
CALL POUT
MOV A,B
ADI '0'

```



```

MOV     E,L
POP     H      ; GET HL BACK
POP     PSW    ; GET VALUE TO ADD TO
ADD     E
MOV     E,A
MOV     A,D
ACI     0
MOV     D,A   ; DE=ACCUMULATED VALUE
DCR     B     ; COUNT DOWN
JNZ     DIG1
MVI     A,1   ; SET NOT ZERO (NO ERROR)
ORA     A
MOV     A,E   ; LOW-ORDER IN A
JMP     DIGER1
DIGERR  CALL   PERR
DB      'INVLD NUM',0
JMP     ZERR  ; EXIT TO CP/M
DIGER1  POP     H
POP     B
RET

* PRINT ERROR UTILITY -- PLACE ERROR MSG ON CON:; MSG PTED TO BY RET ADR
PERR    MVI     A,'S' ; PRINT ERROR
        CALL    ZOUT
        CALL    ZOUT
        MVI     A,' '
        CALL    ZOUT
        POP     H      ; GET CHAR
        MOV     A,M
        INX     H
        ORA     A      ; CHECK IF DONE
        JZ      PERRD
        CALL    ZOUT
        JMP     PERRL
        PERRD   PCHL

*APPND -- APPEND FILE TO OUTPUT; CURRENT FILE CLEARED
APPND   CALL    LPCLR  ; CLEAR LOOP FLAGS
        XRA     A
        STA     MTAB   ; CLEAR MACRO TABLE
        CALL    SETFCB ; GET FCB FROM NEXT WORD
        CALL    LOAD   ; GET FILE
        CALL    ZCR
        LXI     H,ZBOF ; PT TO 1ST WORD

```

```

* SET FCB UTILITY -- EXTRACT NEXT WORD AS FILE SPEC AND INIT FCB
SETFCB PUSH H ; SAVE PTR TO NEXT WORD
LXI D,FCB ; CLEAR FCB
MVI A,0
MVI B,33 ; 33 BYTES
CALL FILL ; FILL BUFFER
MVI C,11 ; 11 CHARS
LXI D,FCB+1 ; PT TO EXT IN FCB
LXI H,DEXT ; PT TO 'TFS'
CALL LCIR ; PLACE DEFAULT EXT
POP H ; GET PTR TO NEXT WORD
CALL SWRD ; GET FILE NAME
LXI H,WBUF+1 ; CHECK FOR DRIVE SPEC
MOV A,M ; COLON?
DCX H ; PT TO 1ST CHAR IN WBUF
CPI ':'
JNZ SFCB0
MOV A,M ; GET DRIVE LETTER
SETDRV ; LOG IN DRIVE
INX H ; SKIP DRIVE SPEC
INX H
MVI B,8 ; 8 CHARS IN FILE NAME
LXI D,FCB+1 ; STORE IN FCB
MOV A,M ; GET CHAR
CPI '+' ; <SP>?
RC ; EXTENSION?
CPI '.'
JZ SFCB2
CALL CAPS
STAX D ; STORE OTHERWISE
INX H
INX D
DER B
JNZ SFCB1
MOV A,M ; EXT?
CPI ':'
RNZ
INX H ; PT TO 1ST BYTE OF EXT
MVI B,3 ; 3 BYTES
LXI D,FCB+9
LXI A,M ; GET CHAR
MOV A,M ; GET CHAR
CPI '+'

```

25 JULY 1980

```

RC      CALL    CAPS      ; PUT CHAR
STAX    D      ; PT TO NEXT
INX     H
INX     D
INX     B
IIR     SFCB2E
JNZ     RET
* LOG IN DRIVE WHOSE LETTER IS IN A REG
SETDRV:
PUSH H ; PUSH D ; PUSH B
STA CURSDRV ; SET CURRENT DRIVE
SUI 'A' ; CONVERT A-D TO 0-3
CPI 4 ; ERROR?
JNC SDRERR
MOV E,A ; SET FOR CALL
MVI C,14 ; LOG IN AND SELECT DISK
CALL BDOS
POP B ; POP D ; POP H
RET
SDRERR CALL PERR
DB 'ERROR -- INVALID DISK DRIVE SPECIFIED',0
JMP ZEOR
* FILE LOAD UTILITY -- LOAD FILE SPEC BY FCB INTO TEXT BUFFER
LOAD LXI D,FCB ; PT TO FCB
PUSH D ; SAVE PTR
CALL ZPRR
DB '++ TFS ++ LOAD: ',0
POP H ; GET PTR TO FCB
PUSH H
INX H ; PT TO FN
CALL PRFN
POP D ; GET PTR
MVI C,15 ; OPEN FILE
CALL BDOS
LXI H,ZBOF ; PT TO BOF
CPI OFFH ; ERROR?
JNZ LOAD1
CALL PERR
DB 'FILE NOT FOUND',0
JMP ZEOR
LOAD1 LXI D,FCB ; PT TO FCB
MVI C,20 ; READ NEXT RECORD

```

```

LDA      BOOT+2      ; CHECK FOR FILE TOO LARGE
DCR      A           ; BACK UP ONE PAGE FOR GOOD MEASURE
CMP      H           ; H PT TO THIS PAGE? --- OVERFLOW IF SO
JNZ      LOAD1C      ; CONTINUE IF NOT
CALL     PERR        ; PRINT ERROR
DB       'ERROR -- TEXT BUFFER OVERFLOW -- BREAK UP SOURCE FILE',0
JMP      ZERR        ; SAVE PTR
LOAD1C   PUSH        ;
CALL     BDOS        ;
POP       H          ;
PUSH     PSW         ; SAVE ERROR STATUS
MVI      B,128       ; LOAD 128 BYTES
D,BUFF      ; PT TO BUFFER
D        D           ; GET BYTE
M,A        ; PUT BYTE
INX      H           ; PT TO NEXT
INX      D           ;
DER       B          ;
JNZ      LOAD2       ;
POP       PSW        ; GET ERROR STATUS
ORA      A           ; 0=CONTINUE
JZ       LOAD1       ;
MVI      M,EOP       ; ENSURE EOP
RET

```

```

*MAC -- DEFINE MACRO
M*BEG    CALL         ROPCK
        CALL         ROPCK1
        CALL         FWRD
        CALL         SWRDO
        PUSH         H
        LXI          H,MTAB
        MOV          A,M
        INR          M
        INX          H
        ORA          A
        JZ           MAC2
        MVI          B,CCNT+2
        INX          H
        DCR          B
        JNZ          MAC1
        DCR          A
        JNZ          MAC0
        ; MAC MAY NOT BE IN DATA FILE OR INPUT LINE
        ; GET MACRO NAME
        ; IN WBUF
        ; SAVE PTR TO NEXT WORD
        ; STORE IN TABLE
        ; CHECK FOR FIRST
        ; SKIP TO CORRECT ENTRY PT
MAC0
MAC1
MAC2

```

```

MAC2      LXI      D,MBUF      ; DE PT TO CMD, HL PT TO TABLE LOC
          MVI      B,CCNT
          PUSH     H
          LDAX     D
          CALL     CAPS
          MOV      M,A
          INX      H
          INX      D
          CPI      ' '
          JZ       MAC4
          DCR      B
          JNZ      MAC3
          POP      H
          MVI      A,CCNT
          CALL     ADR
          SHLD     TMP1
          POP      H
          CALL     FWRD
          XCHG
          LHLD     MOV
          INX      H
          MOV      M,D
          XCHG
          CALL     CALL
          CALL     SARD0
          PUSH     H
          LXI      H,MBUF
          MOV      A,M
          INX      H
          CPI      CCHAR
          JNZ      MAC8
          LXI      D,MENDS
          MVI      B,CCNT
          XCHG
          LDAX     CALL
          CALL     CMP
          JNZ      INX
          INX      D
          DCR      B
          JNZ      MAC7

          ; DE PT TO CMD, HL PT TO TABLE LOC
          ; SAVE PTR
          ; GET & STORE
          ; CAPITALIZE
          ; DONE IF <SP>
          ; PT TO START
          ; PT TO ADR
          ; HL PT TO ADR POS
          ; SAVE PTR
          ; GET PTR TO NEXT CHAR
          ; FIND NEXT WORD
          ; GET TABLE LOC
          ; ADR IN TABLE
          ; HL PT TO NEXT WORD
          ; FIND NEXT WORD
          ; PUT IN BUFFER
          ; CHECK FOR CCHAR
          ; END?
          ; DE PTS TO ENTRY, HL TO 'ENDM'
          ; CHECK
          ; CAPITALIZE ENTRY

```

```

MAC8      POP      H      ; HL PT TO LOC AFTER MACRO
          RET
          POP      H
          JMP      MAC6      ; CONTINUE
* NUMBER CONVERT UTILITY — CONVERT LOW BCD DIGIT TO ASCII
CONVRT    ANI      OFH
          ADI      '0'
          CPI      '0'
          RNZ
          MVI      A,0A0H
          RET
* PLACE NUMBER IN REG A INTO OUTPUT LINE
PNUM      PUSH     PSH
          LXI      D,WBUF
          MVI      B,10      ; COMPUTE 10'S
          MVI      C,0      ; COUNT = 0
          SUB      B      ; A=A-10
          JC       PNUM02
          INR      C      ; INCR 10'S
          JMP      PNUM01
          ADD      B      ; ADD 10 BACK IN
          MOV      B,A      ; SAVE 1'S IN B
          MOV      A,C      ; GET 10'S
          ORA      A      ; PLACE ' ' IF ZERO
          JNZ      PNUM03
          MVI      A,' '
          STAX     D      ; <SP>
          JMP      PNUM04
          ADI      '0'      ; CONVERT TO ASCII
          STAX     D      ; STORE
          INX      D      ; PT TO NEXT
          MOV      A,B      ; GET 1'S
          ADI      '0'      ; CONVERT TO ASCII
          STAX     D      ; STORE
          MVI      A,' '      ; STORE TRAILING <SP>
          INX      D
          STAX     D
          MVI      A,2
          STA      WBUF-1
          MOV      C,A
          POP      PSH
          RET

```

```
*1SETN -- INITIALIZE COUNTER N
NEQ  CALL  GETNUM ; GET VALUE TO SET N TO
      STA  NO
      RET
```

```
*1N -- DISPLAY AND INCREMENT COUNTER N
NO    LDA  NO
      CALL FNUM
      INR  A
      STA  NO
      POP  D
      POP  D ; CLEAR ADR
      RET
```

```
*1BS -- BACKSPACE
BS    LDA  NCHARS
      DCR  A
      RC
      STA  NCHARS
      PUSH H
      LHD  CPOS
      DCX  H
      SHLD CPOS
      POP  H
      RET
```

```
* TRAP UTILITY FOR INVALID DATA FILE COMMANDS
RDFCK LDA  RDFLG ; READ DATA FILE CHECK
      ORA  A ; ABORT IF IN A DATA FILE
      RZ
      CALL PERR
      DB   'INVLD CMD IN DATA FILE',0
      JMP  ZBOR
```

```
*1OPEN -- OPEN DATA FILE AND INITIALIZE DATA FILE BUFFER
OPEN  PUSH  H ; SAVE PTR TO NEXT WORD
      CALL SETFCB ; SET UP FCB
      CALL ZPRR
      DB   '++ TFS ++ OPEN: ',0
      LXI  H,FCB+1 ; PT TO FCB NAME
      PUSH H ; SAVE PTR
      CALL PRFN ; PRINT FILE NAME
      POP  D ; PTR TO FCB
      DCX  D ; PT TO 1ST BYTE
```

```

MVI C,15 ; OPEN FILE
CALL BDOS
LXI H,DATA ; PT TO BUFFER
SHLD DPTR ; SET PTR TO NEXT WORD
CPI OFFH ; ERROR?
JNZ OPEN1
CALL PERR
DB 'DATA FILE NOT FOUND',0
JMP ZEOR
XRA A ; A=0
STA EOFLG ; SET NOT EOF
INR A ; A=1
STA DFLG ; SET DF LOAD
CALL DCOM ; LOAD 512 BYTES
CALL DCOM
POP H ; RESTORE PTR TO NEXT WORD (FILE NAME)
CALL FWRD ; FIND NEXT WORD
JMP SWRDO ; DISCARD IT
* PRINT FILE NAME UTILITY; FILE NAME IN FCB FORMAT PTED TO BY HL
PRFN MVI B,8 ; PRINT FILE NAME
CALL PRFN1
MVI A,'.'
CALL ZOUT
MVI B,3 ; PRINT EXTENSION
CALL PRFN1
CALL ZCR
RET
PRFN1 MOV A,M ; GET CHAR
CALL ZOUT
INX H ; PT TO NEXT CHAR
DCR B
JNZ PRFN1
RET
* LOAD 256 BYTES FROM DATA FILE
DCOM CALL DCOM0 ; LOAD 256 BYTES
DCOM0 LDA EOFLG ; EOF FROM LAST LOAD?
ORA A
RNZ
PUSH H ; SAVE PTR TO LOAD AREA
MVI C,20 ; READ NEXT RECORD
LXI D,FCB ; PT TO FCB
CALL BDOS
CPI OFFH ; EOF?

```

```

DCOM1      JNZ      DCOM1      ; SET EOF ENCOUNTERED FLAG
            STA      EOFLG      ; MOVE TO ADR IN DE
            POP      D           ; PT TO DATA
            LXI      H,BUFF     ; 1/2 BLOCK
            MVI      C,128      ;
            CALL     LCIR        ; HL PTS TO NEXT LOAD ADR
            XCHG              ;
            RET

* INITIALIZE DATA FILE FLAGS
CLOSE1 XRA A
        STA RDLG

*NCLOS -- CLOSE DATA FILE
CLOSE  CALL RDLG
        XRA A
        STA DFLG ; RESET FLAGS
        STA RDLG
        RET

*3READ -- READ NEXT LINE FROM DATA FILE
READ   CALL RDLG
        SHLD RTEMP ; SAVE POINTER TO NEXT WORD
        LDA DFLG   ; DF OPEN?
        ORA A
        JNZ READ1
        CALL PERR
        DB 'DATA FILE NOT OPEN',0
        JMP ZBOR
        LHL DPTR   ; SET TFS NEXT WORD POINTER
        MVI A,1    ; SET RDLG
        STA RDLG
        MOV A,M
        CPI EOF    ; EOF?
        JNZ READ2
        CALL PERR
        DB 'EOF OF DATA',0
        JMP ZBOR
        MOV A,H    ; CHECK FOR 2ND BLOCK
        CPI B2HI
        RNZ
        DCR H      ; 256 BYTES LESS
        PUSH H     ; SAVE PTR
        MVI H,B2HI ; PT TO HIGH BLOCK

```

```

MVI D,R2HI-1
MVI L,0
MOV E,L
LXI B,256
PUSH H
CALL LIR
POP H
CALL DCOM
POP H
RET
; HL = RAM ADR TO LOAD INTO
; PT TO NEXT DATA RECORD TO READ
; PT TO 1ST CHAR IN RECORD (LINE)

```

```

*BT -- TAB TO NEXT TAB STOP
TAB CALL GETNUM ; GET COLUMN NUMBER
MOV C,A ; STORE IN C
LDA NCHARS ; CHECK LINE CHAR CNT
INR A
MOV D,A
MOV E,A
CPE C
RNC
MOV A,C
SUB D
MOV C,A
MOV B,A
PUSH H
LXI H,LBUF ; MAKE <SP> SIG
MOV A,M
CPI ;
JNZ TAB2
ORI 80H ; SET MSB
MOV M,A
INX H ; PT TO NEXT
DCR D
JNZ TAB1
DCX H
MVI M,' '+90H
INX H
DCR C
JNZ TAB3
SHLD CPOS
MOV A,E
ADD B
DCR A

```

TAB1 H,LBUF ; MAKE <SP> SIG

TAB2 ; SET MSB

TAB3 M,' '+90H

```

STA  NCHARS
POP  H
RET
* LOAD REG VALUE SPECIFIED IN C INTO A
LOCR PUSH H ; LOCATE R VAL (R=C)
LXI  H,RV TAB
MOV  A,C
CALL ADR
MOV  A,M ; GET VAL
XCHG ; PTR IN DE
POP  H
RET

*SETR -- SET SPECIFIED REGISTER TO SPECIFIED VALUE
SETR CALL GETNUM ; GET REGISTER NUMBER
MOV  C,A
CALL LOCR ; LOCATE REGISTER NUMBER IN C
PUSH D ; SAVE REG LOC
CALL GETNUM ; GET VALUE
POP  D ; GET REG LOC
STAX D ; STORE VALUE
RET

*INCR -- INCREMENT SPECIFIED REGISTER
INCR CALL GETNUM ; GET REGISTER NUMBER
INCR1 MOV C,A ; REG NUM IN C
INCR2 CALL LOCR ; LOCATE REGISTER
INR  A
STAX D ; PLACE IT BACK
RET

*DR -- DISPLAY CURRENT VALUE OF SPECIFIED REGISTER
DR CALL DRI
POP  D
POP  D
RET
DRI CALL GETNUM
MOV  C,A
STA  TMP
CALL LOCR
CALL PNJM
RET

```

*R -- DISPLAY CURRENT VALUE OF SPECIFIED REGISTER AND INCREMENT AFTERWARDS

```
RINC
CALL DRI
LDA TMP
CALL INCR1
POP D
POP D
RET
```

*CLRR -- CLEAR ALL REGISTERS

```
CLRR
PUSH H
LXI H,RVTAB
MVI B,100
MVI M,1
INX H
DCR B
JNZ CLRR1
POP H
RET
```

*BLK -- DEFINE SIGNIFICANT BLANK CHARACTER

```
BLANK
CALL SBRD ; GET CHAR
LDA WBUF
STA BLK
RET
```

* DISPLAY OUTPUT CHAR UTILITY — IF /V OPTION, PRINT ON CON.; OTHERWISE, LST:
POUT:

PUSH PSW
ANI 7FH ; MASK
PUSH 9

MOV B,A
LDA BLK
CMP B
JZ POUTB

MOV A,B
POP B
CALL POUT0
POP PSW

POUTE

RET
MVI A,' ' ; PRINT BLANK
JMP POUTE

PUSH PSW
CALL SKCHECK
JZ POUT1
POP PSW ; CHECK FOR SKIPPING; RET W/NZ IF YES
RET ; DON'T PRINT

LDA DOUT ; CHECK FOR DISK OUTPUT
ORA A ; 0=NO
JNZ POUT3 ; DISK OUTPUT
LDA FLAG ; CHECK FOR VIEW OPTION
CPI 'V'

JNZ POUT2
POP PSW ; CHECK FOR <CR>
CPI 0DH
JNZ ZOUT

CALL ZESC ; ESCAPE
MVI A,0DH ; <CR>
JMP ZOUT

POUT2 POP PSW ; GET CHAR

JMP ZPOUT
POP PSW ; GET CHAR
JMP FSPUT ; PUT TO DISK

*
* GENERAL SUPPORT SECTION
*

NO-A000 217

ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/G 9/2
TFS--THE TEXT FORMATTING SYSTEM. A TEXT FORMATTER DESIGNED TO R--ETC(U)
JUL 80 R L CONN

UNCLASSIFIED

ML

2 14 2

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

END

DATE

FILMED

9-80

DTIC



~~NATIONAL BUREAU OF CRIMINAL INVESTIGATION~~

```

PUSH D
PUSH B
PCHL
POP H ; GET RET ADR
POP B
POP D
XTHL
RET

* CHAR INPUT FROM CON:
ZIN CALL PUT ; SAVE REGS
LXI H,ZIN1 ; SET UP RET ADR
PUSH H
LHLD INADR ; GET ADR OF INPUT ROUTINE
PCHL
CALL GET ; RESTORE REGS
RET

* CHAR OUTPUT ON CON:
ZOUT PUSH PSW ; SAVE A
CALL PUT ; SAVE REGS
MOV C,A ; CHAR IN C
LXI H,ZOUT1 ; SET UP RET ADR
PUSH H
LHLD OUTADR ; GET ADR OF OUTPUT ROUTINE
PCHL
CALL GET ; RESTORE REGS
POP PSW ; GET A
RET

* GET STATUS OF CON:
ZSTAT CALL PUT ; SAVE REGS
LXI H,ZSTAT1 ; SETUP RET ADR
PUSH H
LHLD STADR ; ADR OF STATUS ROUTINE
PCHL
CMA ; FLIP FLAGS
ANI 1 ; SET FLAGS -- 0 MEANS CHAR READY
CALL GET ; RESTORE REGS
RET

* CHAR OUTPUT TO LST:
ZFOUT PUSH PSW ; SAVE A
CALL PUT ; SAVE REGS
MOV C,A ; CHAR IN C
CALL SKCHX ; CHECK FOR SKIPPING; RET W/NZ IF YES
JNZ ZOUT1 ; DON'T PRINT IF SO

```

25 JULY 1980

```

LXI   H,ZCUT1 ; SET UP RET ADR
PUSH  H
LJLD  PRADR    ; GET ADR OF ROUTINE
PCHL

```

```

* CHECK FOR ABORT
ZINK  CALL  ZSTAT    ; CHAR READY?
ORA   A              ; 0 MEANS YES
RNZ

```

```

* WAIT FOR ANY KEY AND ABORT IF <ESC> OR CTRL-C
ZESC  CALL  ZIN       ; GET CHAR
CPI   ESC            ; <ESC>?
JZ    ZBOR
CPI   CTRLC           ; CTRL-C?
JZ    ZBOR

```

```

* RETURN TO CP/M
ZBOR  LDA   DOUT      ; DISK OUTPUT?
ORA   A              ; 0=NO
JZ    BOOT

```

```

CALL  F$CLOS    ; CLOSE OUTPUT FILE
JMP   BOOT

```

```

* EXCHANGE NYBBLES OF A REG
ZEN   RLC
      RLC
      RLC
      RLC
      RET

```

```

* PRINT LINE PTED TO BY RET ADR UNTIL 0 ENCOUNTERED ON LST: OR CON:
PRNTP XTHL
PRNTP1 MOV  A,M
      ORA  A          ; 0=DONE
      JZ   ZPRR2

```

```

CALL  POUT
INX   H
      JMP  PRNTP1

```

```

* PRINT LINE PTED TO BY RET ADR UNTIL 0 ENCOUNTERED ON CON:
ZPRR  XTHL
CALL  ZCR           ; NEW LINE
      MOV  A,M
      ORA  A          ; 0 MEANS DONE
      JZ   ZPRR2
CALL  ZOUT
INX   H

```

```

ZPRR2      JMP      ZPRR1      ; RESTORE ADR & HL
XTHL
RET
* PRINT <CR> <LF> ON CON:
ZCR      MVI      A,0DH      ; <CR>
CALL     ZOUT
MVI      A,0AH      ; <LF>
JMP      ZOUT
* MOVE W/COUNT IN C
LCIR      MVI      B,0      ; B=0
* LDIR SUBSTITUTE ROUTINE
LIR      MOV      A,M      ; GET BYTE
STAX     D
INX      D      ; PT TO NEXT
INX      H
DCX      B      ; COUNT DOWN
MOV      A,B
ORA      C
JNZ      LIR
RET
* FILL MEMORY PTED TO BE DE WITH BYTE IN A; B=BYTE COUNT
FILL      STAX     D      ; FILL IT
INX      D      ; PT TO NEXT
DCR      B
JNZ      FILL
RET
* (HL)=(HL)+(A)
ADR      ADD      L
MOV      L,A
RNC
INR      H
RET
* CAPITALIZE CHAR IN A IF LOWER-CASE ALPHABETIC
CAPS      CPI      61H      ; SMALL A
RC
CPI      7BH      ; SMALL Z + 1
RNC
SUI      020H      ; MASK FOR CAPITAL
RET

```

```
*****
*** BYTE-ORIENTED FILE OUTPUT ROUTINES -- FILEO ****
*****
```

```
*
* FILEO.ASM -- BYTE-ORIENTED FILE OUTPUT
* INCLUDED ROUTINES ARE -- FILE OUTPUT OPEN, FILE OUTPUT CLOSE, PUT
* F$OPEN -- OPEN OUTPUT FILE
* F$CLOS -- CLOSE OUTPUT FILE
* F$PUT -- PUT BYTE INTO OUTPUT FILE
*
```

```
*
* SUPPORTING ROUTINES
*
```

```
* F$ABORT -- ABORT PROCESSING
*
```

```
F$ABORT:
CALL ZPRR
DB 0DH,0AH,'FILED -- FILE ERROR -- ABORTING',0
JMP BOOT
```

```
* OERR -- PUT ATTEMPTED WITHOUT OPENING FILE
*
```

```
F$OERR:
CALL ZPRR
DB 0DH,0AH,'FILED -- PUT ATTEMPTED WITHOUT OPENING FILE',0
JMP BOOT
```

```
* WRIT$BLOCK -- WRITE BLOCK TO OUTPUT FILE
*
```

```
WRIT$BLOCK:
CALL F$LOGIN ; LOGIN OUTPUT DRIVE
LXI H,OSBUF ; PT TO OUTPUT BUFFER
LXI D,BUFF ; PT TO TEMP BUFFER
MVI C,128 ; 128 BYTES
CALL LCIR ; COPY FROM HL TO DE FOR C BYTES
LXI D,OSFCB ; PT TO DEFAULT FCB
MVI C,B$WR ; WRITE BLOCK
CALL B$DYS
ORA A ; OK?
```

```

JNZ    F$ABORT
CALL   F$LOGOUT      ; LOGOUT OUTPUT DRIVE
RET

```

```

*
* F$OPEN -- OPEN FILE WHOSE FCB IS PTED TO BY DE FOR OUTPUT
*

```

F\$OPEN:

```

PUSH H ; PUSH D ; PUSH B
CALL   F$LOGIN ; LOGIN OUTPUT DRIVE
LXI    H,0$FCB ; PT TO DEFAULT OPEN FCB
PUSH   H      ; SAVE PTR TO FCB
XCHG    ; HL PTS TO FCB, DE PTS TO BUFFER AREA
MVI     C,33  ; 33 BYTES TO MOVE
CALL    LCIR  ; MOVE IT
POP     D      ; GET PTR TO FCB
PUSH    D      ; SAVE PTR
MVI     C,B$SF ; SEARCH FOR FILE
CALL    BDOS
CPI     255    ; NO MATCH?
JZ      F$OPEN1 ; PROCEED TO OPEN IT IF NO MATCH
CALL    ZPRR
ODH,0AH, 'OUTPUT FILE EXISTS -- DELETE IT (Y/N<CR>=Y)? ',0
ZIN     ; GET RESPONSE
CALL    CAPS   ; CAPITALIZE
CPI     'N'    ; NO?

```

```

JZ      F$ABORT
CALL    ZCR
POP     D      ; GET PTR TO FCB
PUSH    D      ; SAVE PTR TO FCB
MVI     C,B$DF ; DELETE FILE
CALL    BDOS

```

F\$OPEN1:

```

POP     D      ; GET PTR TO FCB
MVI     C,B$WF ; MAKE FILE
CALL    BDOS
CPI     255    ; 255 MEANS NOT OK
JZ      F$ABORT
LXI     H,0$BUF ; PT TO BUFFER
SHLD    A,128  ; SAVE PTR
MVI     A,128  ; SET COUNT
STA     0$CNT
MVI     A,0FFH ; SET FILE OPENED FLAG
STA     0$OTLG

```

```
CALL F$C$COUT      ; LOGOUT OUTPUT DRIVE
POP B ; POP D ; POP H
RET
```

```
* F$PUT -- PUT BYTE IN REG A INTO OUTPUT FILE
```

```
F$PUT:
```

```
PUSH H ; PUSH D ; PUSH B
PUSH PSM ; SAVE PSM
LDA OSOFLG ; GET FILE OPENED FLAG
ORA A ; ZERO MEANS NO
JZ F$OERR
POP PSM ; GET PSM
PUSH PSM ; SAVE IT AGAIN
LHLD OSPTR ; GET PTR TO NEXT BYTE
MOV M,A ; STORE BYTE
INX H ; PT TO NEXT
SHLD OSPTR
LDA OSCNT ; GET COUNT
DCR A ; DECREMENT
STA OSCNT
JNZ F$PUT1 ; RETURN IF OK
```

```
* BUFFER FULL -- WRITE IT TO DISK AND RESET POINTER AND COUNT
```

```
LXI H,OSBUF ; RESET POINTER
SHLD OSPTR
MVI A,128 ; RESET COUNT
STA OSCNT
CALL WRIT$BLOCK
```

```
* NORMAL EXIT
```

```
F$PUT1:
```

```
POP PSM ; GET PSM
POP B ; POP D ; POP H
RET
```

```
* F$CLOS -- CLOSE FILE OPENED FOR OUTPUT
```

```
F$CLOS:
```

```
PUSH PSM ; SAVE A
PUSH H ; PUSH D ; PUSH B
CALL F$LOGIN ; LOGIN OUTPUT DRIVE
MVI A,CTRLZ ; PUT CTRL-Z
CALL F$PUT
```

```
F$CLOSE1:
```

25 JULY 1980

```

LDA    O$CNT ; EVEN BLOCK JUST WRITTEN?
CPI    128 ; JUST CLOSE IF SO
JZ     F$CLOSE2
XRA    A ; PUT ZERO
CALL   F$PUT
JMP     F$CLOSE1

F$CLOSE2:
XRA    A ; A=0
STA    O$OFLG ; SET OUTPUT OPENED FLAG
LXI    D,O$FCB ; PT TO OUTPUT FCB
MVI    C,B$CCL ; CLOSE FILE
CALL   B$DOS
CPI    OFFH ; ERROR?
JNZ     F$CLOSE3
CALL   Z$PRR
DB     0DH,0AH,'FILED -- ERROR IN CLOSING FILE',0
JMP     BOOT ; WARM BOOT TO CP/M

F$CLOSE3:
CALL   F$LOGOUT ; LOGOUT OUTPUT DRIVE
POP B ; POP D I POP H
POP    PSW ; RESTORE A
RET

* LOG IN OUTPUT DRIVE
*
F$LOGIN:
LDA    CUR$DRV ; GET CURRENT DRIVE
STA    TEMP$DRV ; SAVE
LDA    OUT$DRV ; GET OUTPUT DRIVE
JMP     SETDRV ; LOG IN

* LOG OUT OUTPUT DRIVE
*
F$LOGOUT:
LDA    TEMP$DRV ; GET CURRENT DRIVE
JMP     SETDRV ; LOG OUT

* BUFFERS
*
O$OFLG: DB 0 ; OUTPUT FILE OPENED FLAG (0=NO)
O$FCB:  DB 33 ; OUTPUT FILE FCB
O$BUF:  DB 128 ; OUTPUT BUFFER

```

PAGE 61 -- TFS.ASM -- SOURCE TO THE TEXT FORMATTING SYSTEM 25 JULY 1980

OSPTR: DS 2 ; OUTPUT CHAR PTR
OSCNT: DS 1 ; OUTPUT CHAR COUNT

* CP/M EQUATES AND ASCII CONSTANTS
*

BSCL EQU 16 ; CLOSE FILE
BSSF EQU 17 ; SEARCH FOR FILE
BSDF EQU 19 ; DELETE FILE
BSWF EQU 22 ; MAKE FILE
BSWR EQU 21 ; WRITE NEXT RECORD

*** END OF FILE

*

* TFS RESIDENT COMMAND TABLE!

CCNT	EQU	4	; # CHARS IN A CMD
CTAG	D8	S5	
	D8	'RB	; #RB
	DW	RB	
	D8	'AP	; #AP — TOGGLE
	DW	APAR	
	D8	'STOP	; #STOP
	DW	EXIT	
	D8	'HALT	; #HALT
	DW	EXIT	
	D8	'EXIT	; #EXIT
	DW	EXIT	
	D8	'LOOP	; #LOOP
	DW	LOOPLP	
	D8	'ENDL	; #ENDL
	DW	LPEND	
	D8	'LEX	; #LEX
	DW	LPEXIT	
	D8	'RJ	; #RJ
	DW	RJ	
	D8	'NORJ	; #NORJ
	DW	NORJ	
	D8	'PAUS	; #PAUS TEXT
	DW	PAUSE	
	D8	'UL	; #UL — TOGGLE
	DW	UL	
	D8	'ASIS	; #ASIS
	DW	ASIS	
	D8	'BR	; #BR
	DW	BREX	
	D8	'CR	; #CR
	DW	CARET	
	D8	'P	; #P
	DW	PAR	
	D8	'PX	; #PX
	DW	PAXX	
	D8	'PAR	; #PAR N N (INDENT, SPACING BET PARS)
	DW	STPAR	

```

DB      'PARX' ; $PARX N N
DW      STPARX
DB      'LMAR' ; $LMAR N (COLUMN NUMBER)
DW      STLMAR
DB      'LLEN' ; $LLEN N (COLUMN COUNT)
DW      STLLEN
DB      'LINE' ; $LINE N N (# TEXT LINES, PHYSICAL LINES/PAGE)
DW      STLINE
DB      'PGON' ; $PGON N (LOC OF PAGE #)
DW      STPAGE
DB      'PGOF' ;
DW      PAGEOF
DB      'PAGE' ; $PAGE
DW      PAGE
DB      'TP' ; $TP N (NUMBER OF LINES)
DW      TPG
DB      'SKIP' ; $SKIP N (NUMBER OF LINES)
DW      SKIP
DB      'PNUM' ; $PNUM N (NUMBER OF NEXT PAGE)
DW      STPNUM
DB      'HEAD' ; $HEAD TEXT
DW      HEAD
DB      'FOOT' ; $FOOT TEXT
DW      FOOT
DB      'C' ; $C TEXT
DW      CEN
DB      'CB' ; $CB -- TOGGLE
DW      CENB
DB      'CH' ; $CH N TEXT (CHAPTER NUM, TEXT OF HEADING)
DW      CHAP
DB      'SAV' ; $SAV
DW      SAV
DB      'RES' ; $RES
DW      RES
DB      'REM' ; $REM TEXT
DW      NEXTLN
DB      'COPY' ; $COPY N (NUM OF COPIES)
DW      COPY
DB      'ENDC' ; $ENDC
DW      CPYEND
DB      'SP' ; $SP N (NUM OF SPACES SET LINES)
DW      SPSET
DB      'SETN' ; $SETN N (N=VALUE TO SET REG N TO)

```

```

DW      NEQ      'BS'      ; %BS
DB      'BS'
DW      BS
DB      'N'      ; %N
DW      ND
DB      'APND'    ; %APND A:FILENAME.EXT ('.EXT'=.TFS' IF OMITTED)
DW      APND
DB      'OPEN'    ; %OPEN A:DATAFILE.EXT
DW      OPEN
DB      'CLOS'    ; %CLOS
DW      CLOSE
DB      'READ'    ; %READ
DW      READ
DB      'T'      ; %T N (COL TO TAB TO)
DW      TAB
DB      'SETR'    ; %SETR N N (REG NUM, VALUE)
DW      SETR
DB      'INCR'    ; %INCR N (REG NUM)
DW      INCR
DB      'R'      ; %R N (REG NUM)
DW      RINC
DB      'DR'      ; %DR N (REG NUM)
DW      DR
DB      'CLRR'    ; %CLRR
DW      CLRR
DB      'BLK'     ; %BLK CHORD (1ST CHAR BECOMES SIG BLANK)
DW      BLANK
DB      'ENDM'    ; %ENDM
DW      MMEND
DB      'MAC'     ; %MAC MACNAME TEXT
DW      MMIEG

```

END OF TFS RESIDENT COMMAND TABLE

*
 *
 *
 *


```

INLEN EQU 128
INLINE DS INLEN ; INPUT LINE FROM CONSOLE
DS 255 ; STACK AREA
STACK EQU $
ZBOP EQU $
ENDALL EQU $
B2HI1 EQU DATA/255
B2HI EQU B2HI1+1

BOOT EQU 0000H ; WARM BOOT ADR
BDOS EQU 0005H ; BDOS ENTRY POINT
BUFF EQU 80H ; TEMPORARY CP/M BUFFER
FCB EQU 5CH ; FILE CONTROL BLOCK
CTRLC EQU 'C'-40H ; ABORT CHAR
CTRLZ EQU 'Z'-40H ; CTRL-Z
EOP EQU CTRLZ ; EOP CHAR
ESC EQU 1BH ; <ESC>
CCHAR EQU '9' ; COMMAND PREFIX CHARACTER

END

```