LY20-0762-0

Installed User Program

SCRIPT/370 Text Processing Facility Under Virtual Machine Facility/370 (VM/370) Systems Guide

Program Number 5796-PAF

SCRIPT/370 is an IBM Installed User Program designed for use with Virtual Machine Facility/370. It provides text-processing facilities. It executes as a command of the Conversational Monitor System (CMS), a component of VM/370. This document describes the interface between the SCRIPT program and CMS, the program organization and structure of SCRIPT/370, and the algorithms for text-processing used by it. It is intended for use by programmers who will maintain or modify the system.

IBM

CONTENTS

SECTION 1: INTRODUCTION....................................3
Purpose of the SCRIPT Program..........................3
Environmental Characteristics..........................3
Physical Characteristics...............................4
Operational Considerations.............................4

SECTION 2: METHOD OF OPERATION...........................7
Initialization...7
Main Processing Sequence...............................7
List Link Element Form................................15
Right Margin Justification Algorithm..................17
Control Word Processing...............................19
Switch Setting.....................................21
Variable Setting...................................22
Special Action Routines............................22
Group 1 Processing -- Page Title...................25
Group 2 Processing -- Page Eject...................25
Group 3 Processing -- Switch Input File............27
Group 4 Processing -- Sectioning...................28
Group 5 Processing -- Status Handling..............29
Group 6 Processing -- Symbol Processing............30
Group 7 Processing -- Terminal I/O.................34
Group 8 Processing -- Termination..................34
Group 9 Processing -- Multiple Column..............36
Group 10 Processing -- Miscellaneous...............40
Termination...41

SECTION 3: PROGRAM ORGANIZATION AND DIRECTORY............42
Program Organization..................................42
Module Directory......................................42
Internal Subroutines of SCSPRT.....................42

SECTION 4: DIAGNOSTIC AIDS...............................47
Error Handling..47
Debugging Facilities..................................47
Register Use..48

SUPPORT PERIOD SERVICES

During a specified number of months immediately following initial availability of this licensed program, designated as the SUPPORT PERIOD, the customer may submit documentation to a designated IBM location when he encounters a problem which his diagnosis indicates is caused by an error in this licensed program. During this period only, IBM through the program author(s) will, without additional charge, respond to an error in the current unaltered release of the licensed program by issuing known error correction information to the customer reporting the problem and/or issuing corrected or notice of availability of corrected code. However, IBM does not guarantee service results or represent or warrant that all errors will be corrected. Any onsite programming services or assistance will be provided at a charge.

WARRANTY

EACH LICENSED PROGRAM IS DISTRIBUTED ON AN 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED.

First Edition (November 1972)

This edition corresponds to Release 1 of SCRIPT/370 and to all subsequent modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are periodically made to the specifications herein; before using the publication in connection with the operation of IBM systems, refer to the latest IBM System/360 and System/370 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office in your locality.

Address comments concerning the contents of this publication to IBM Corporation, VM/370 Publications, 24 New England Executive Park, Burlington, Massachusetts, 01803.

SECTION 1: INTRODUCTION

The SCRIPT program provides the facility to format and print a manuscript that has been stored as one or more files in the Conversational Monitor System (CMS). The SCRIPT program operates as a CMS command.

PURPOSE OF THE SCRIPT PROGRAM

The SCRIPT program reads unformatted manuscript text from one or more specified files. As the text is read, it is inspected for the occurrence of SCRIPT control words which may define formatting characteristics. Under the control of the default settings or user specified control words, the text is formatted and outputted.

ENVIRONMENTAL CHARACTERISTICS

The SCRIPT program is designed for use with CMS. CMS operates under the VM/370 system. SCRIPT is able to operate in the minimum configuration required by CMS, though additional configuration facilities may be required to support certain SCRIPT features such as upper and lower case high-speed printer output, which requires the TN print train.

The SCRIPT program operates in the CMS user's area just like typical user programs and other processing programs. The interfaces between this program and CMS are:

- The CMS command line parameter list.
- CMS file system functions (ERASE, RDBUF, WRBUF, FINIS and STATE)
- Printer I/O function (PRINTR)
- Console I/O functions (WAITRD and TYPLIN).
- CMS simulated OS macro instructions (SPIE, GETMAIN, FREEMAIN).

All files to be processed as input to the SCRIPT program must have a filetype of SCRIPT. By virtue of the SCRIPT filetype, these files consist of variable length records with a maximum line length of 132 characters (actually SCRIPT limits line length to 240 characters, but CMS and CP further limit user text lines to 132 characters).

PHYSICAL CHARACTERISTICS

The SCRIPT program consists of a single CMS load module (filetype MODULE). When the user requests the SCRIPT command, the CMS Command Processor causes the SCRIPT module to be loaded into the user's area, starting at location X'20000', and then control is transferred to SCRIPT. Upon completion or abnormal termination, SCRIPT returns control to the CMS Command Processor.

The SCRIPT module's instructions and static data areas require about 36K bytes of main storage. In addition, certain SCRIPT functions such as Save/Restore-Status, Multiple Column Processing, and Set-Symbols, may require dynamic main storage assignment via the GETMAIN macro instruction. If there is not enough dynamic main storage available to satisfy the requirements of such a SCRIPT function, an error message is printed and SCRIPT processing is terminated.

OPERATIONAL CONSIDERATIONS

The SCRIPT program is invoked via a Command Parameter List. When the user enters a command to CMS, the CMS Command Processor scans the request and converts it to the standard format of a CMS Command Parameter List. The parameter list consists of a sequence of eight-byte entries, one for each word entered as part of the request. See the _IBM Virtual Machine Facility/370 Assembler Programmer's Guide_, GC20-1802, for additional information on parameter list format and command invocation.

When the SCRIPT program is loaded and receives control from CMS, register 1 contains the address of the parameter list. The parameter list may contain the following information in consecutive eight-byte fields:

- The command name, SCRIPT (required).
- The filename of the master CMS file to be processed. A filetype of SCRIPT is assumed (required).
- The CENTER option which causes all output to be shifted right on the printed page.
- The CONTINUE option which causes processing to continue after detecting and printing any recoverable errors.

- The DEBUG option which causes the SPIE SVC macro instruction to be bypassed, thus allowing the correct processing of DEBUG breakpoints.

- The FILE option which causes the output to be directed to a file instead of the terminal or printer. The file is named $filename SCRIPT, where filename is the name of the input master file.

- The MARK option which causes the beginning of each line of the original input to be marked by underlining the first character on the output.

- The NOWAIT option which causes output to start immediately without waiting for the first page to be adjusted.

- The NUMBER option which causes the input filename and line number to be printed to the left of the formatted output line.

- The OFFLINE option which causes the output to be directed to the line printer instead of the terminal.

- The PAGExxx option which causes output to be suppressed until page xxx is reached.

- The QUIET option which causes the version identification information, normally printed on the terminal immediately after SCRIPT gets control, to be suppressed.

- The SINGLE option which causes processing to terminate after printing a single page.

- The STOP option which causes printing to pause at the bottom of each page to allow readjusting or changing of the paper.

- The TRANSLATE option which causes the character translate table to be initialized so that lower-case letters are printed as upper-case letters.

- The UNFORMATTED option which causes the master SCRIPT file to be printed without any formatting, ignoring all control words.

- The 2PASS option which causes two processing passes through the input, with actual output only during the second pass.

The options, if any specified, should be enclosed by left and right parentheses. The options, except for PAGExxx, may be abbreviated by truncation down to two characters. For example, CONTINUE may be abbreviated as CONT or CO.

Output consists of the processed text, diagnostic messages and information messages. Output is sent to the terminal by use of the CMS TYPLIN function, to the line printer by use of the CMS PRINTR function, or to a file by use of the CMS WRBUF function.

A temporary file named CMSUT1 SCRIPT may be created during SCRIPT processing. It is automatically erased on normal SCRIPT termination.

SECTION 2: METHOD OF OPERATION

This section describes the logic and operation of the SCRIPT program and emphasizes the flow of data and control information through buffers and tables (see Method of Operation Diagram 1).

INITIALIZATION

The CMS command processor uses the CMS SVC 202 linkage to invoke the SCRIPT program. Control is passed initially to the primary SCRIPT control section (SCSPRT). This routine starts by performing the following initialization operations: (see Method of Operation Diagrams 2 and 3).

1. Checks the filename specified in the Command Parameter List. If it was not specified, prints error message and terminates. If it was the single character ?, prints the list of SCRIPT options and control words, using entry point SPRCWORD in CSECT SCSFOR and then terminates.

2. Analyzes the options specified in the parameter list, if any, by means of the PARMROUT internal function. Appropriate indicators and variables are set for the options specified.

3. Types the version number identification (unless supressed by the QUIET option) and issues a SPIE SVC macro to regain control in case of a program interrupt (unless supressed by the DEBUG option).

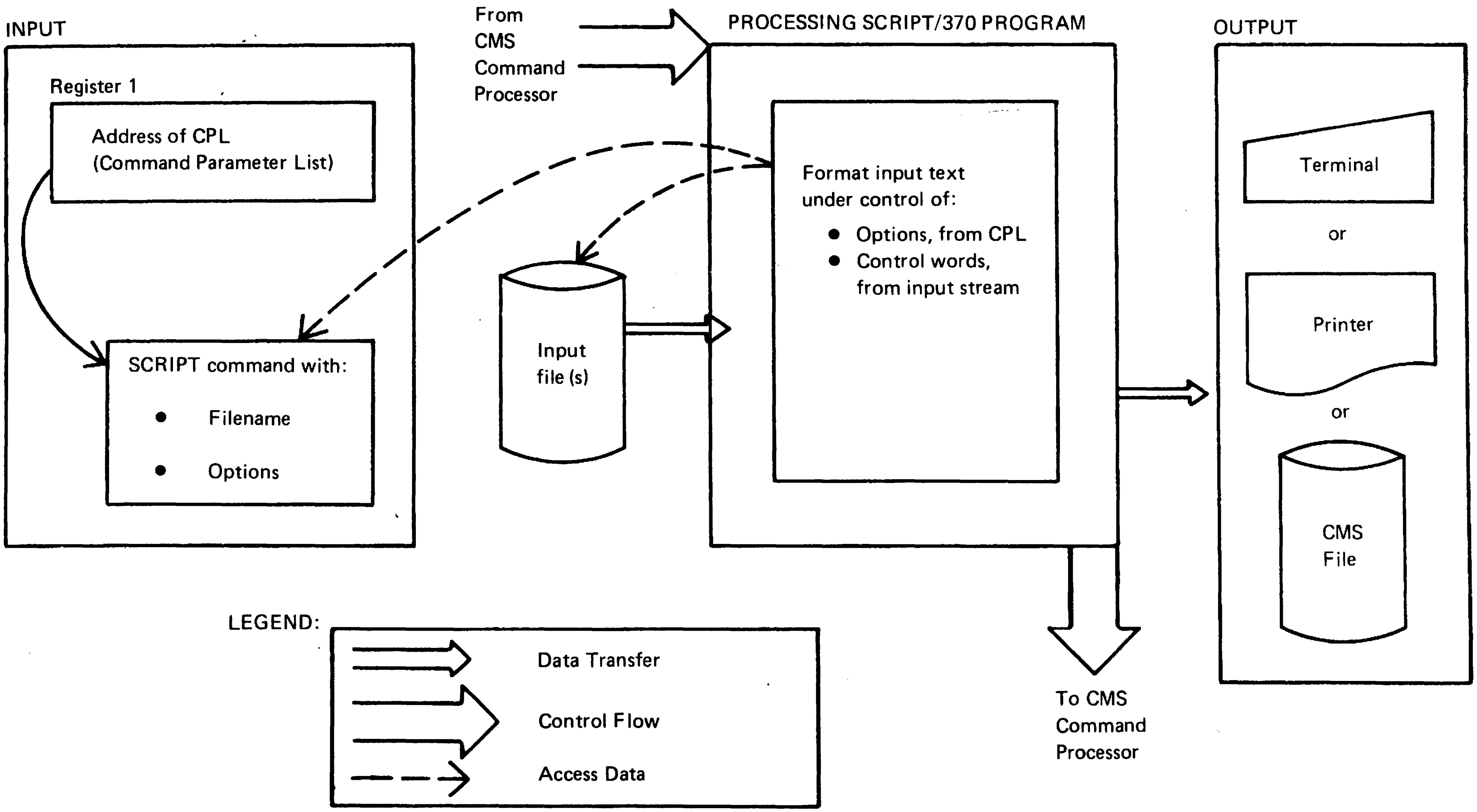
4. Sets the variables and counters (file name, line number, page number, etc.) to their appropriate initial values.

5. Transfers control to the main processing loop (label MAIN within CSECT SCSPRT).

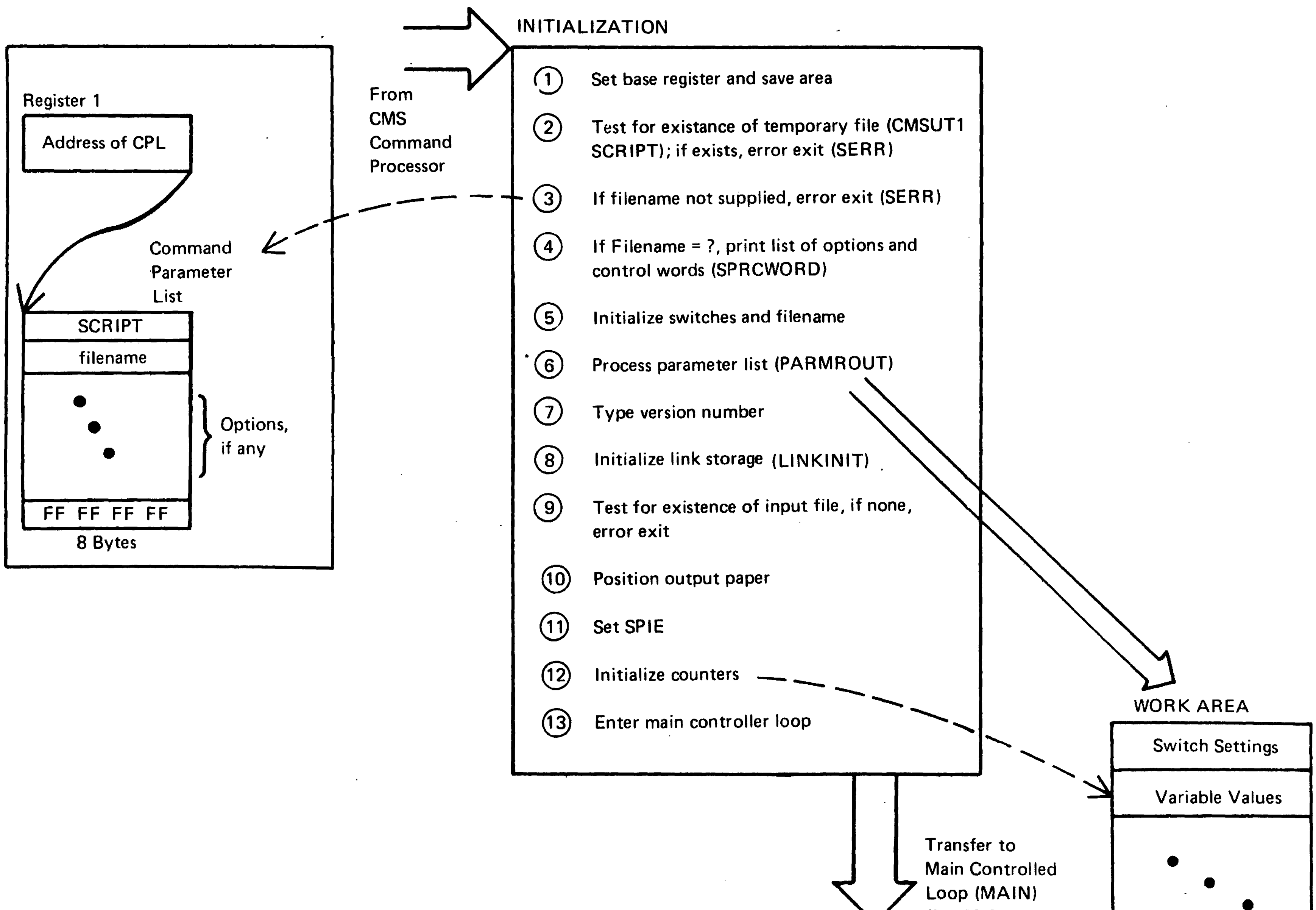
MAIN PROCESSING SEQUENCE

The main processing loop performs the following operations. (see Figure 1 and Method of Operation Diagrams 4a and 4b):

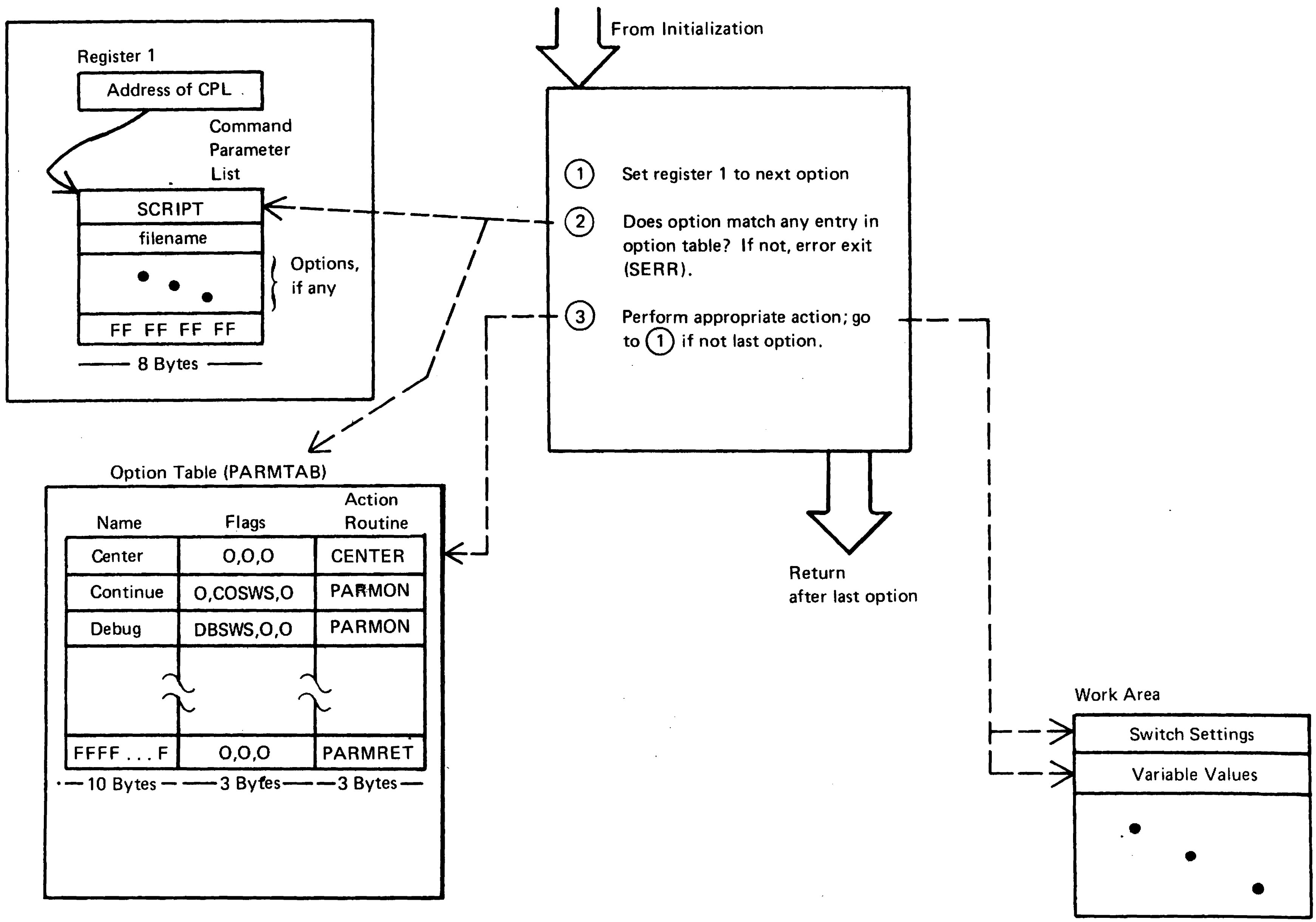
1. Reads the next data line from the current input file.



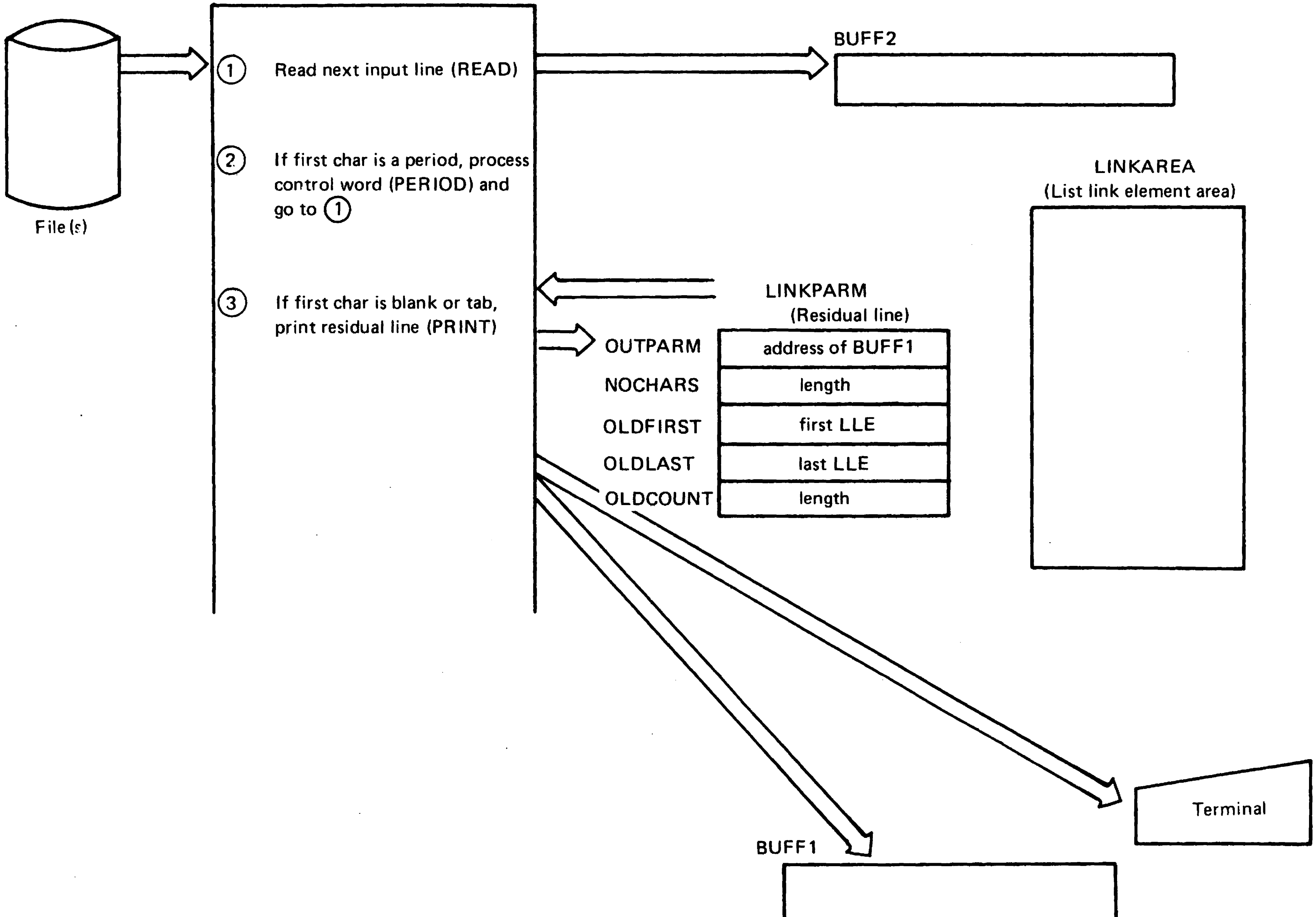
METHOD OF OPERATION DIAGRAM 1. PROCESSING OVERVIEW



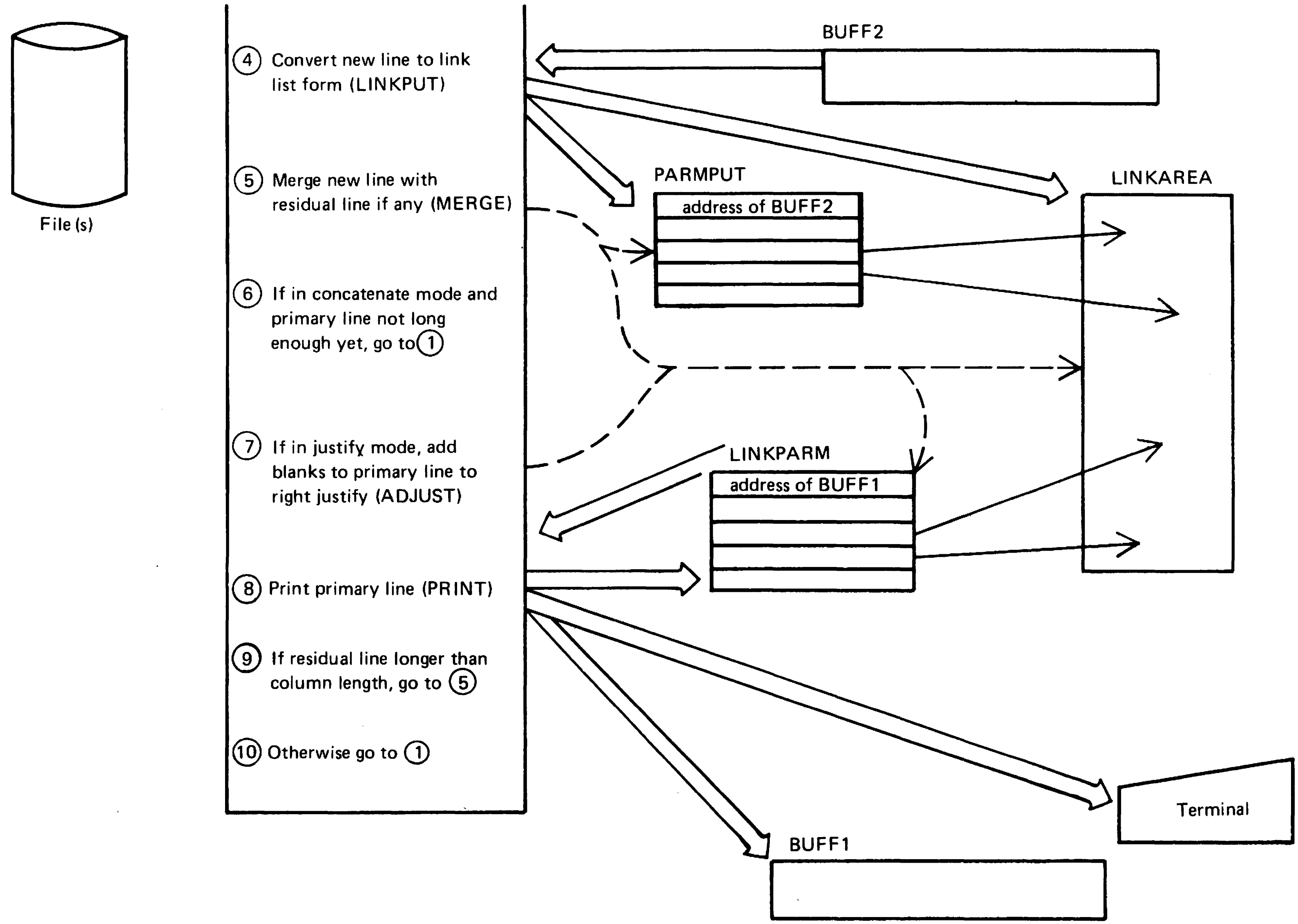
METHOD OF OPERATION DIAGRAM 2. INITIALIZATION



METHOD OF OPERATION DIAGRAM 3. OPTION PROCESSING



METHOD OF OPERATION DIAGRAM 4a. SCRIPT, MAIN CONTROLLER



METHOD OF OPERATION DIAGRAM 4b. SCRIPT MAIN CONTROLLER

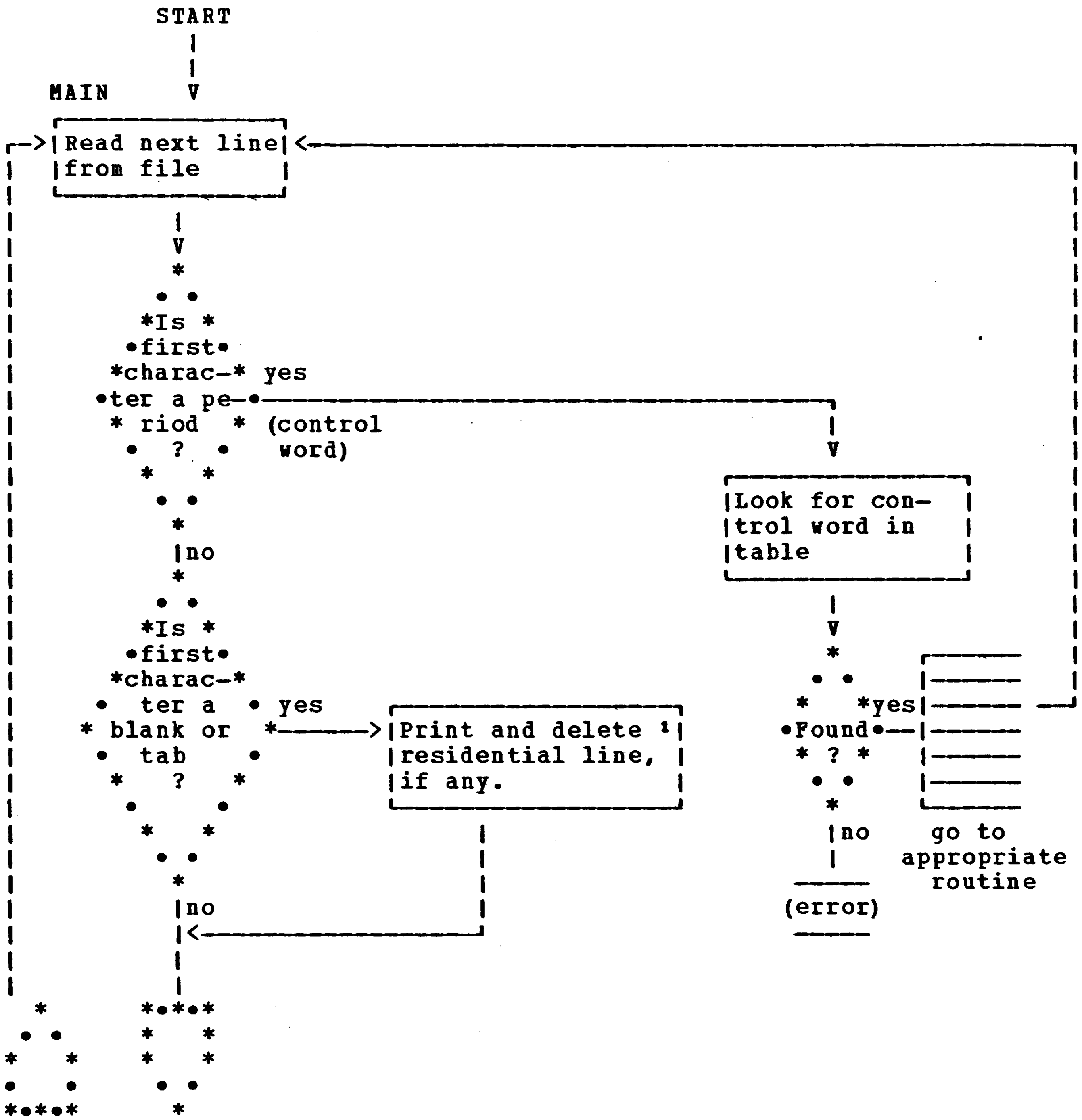
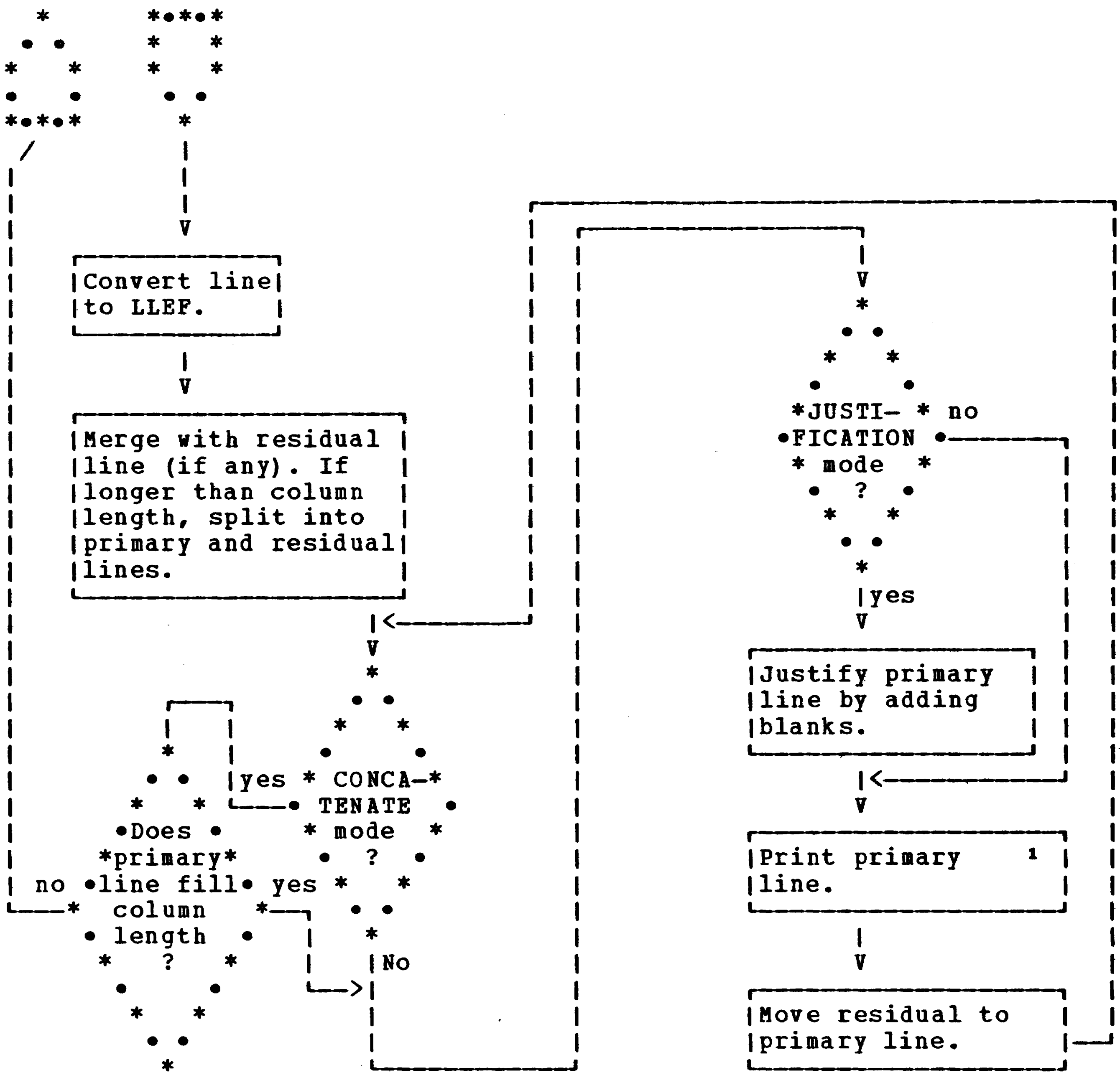


Figure 1. Main Processing Loop (Part 1 of 2)



[1] Printing line causes line to be deleted from linked list buffer.

Figure 1. Main Processing Loop (Part 2 of 2)

If the end-of-file is reached and the current file had been imbedded, resumes reading from next line of the file that invoked the current file. If the end-of-file condition is reached in the master file, processing terminates.

2. Examines the first character of the input line. If it is a period (.), control is transferred to the control word processor (label PERIOD in CSECT SCSPRT). If the first character of the line is a blank or a tab character, any residual line from previous input is sent to the output device. (This is the break function).

3. The new text line is converted into List Link Element Form (LLEF). The new line is then merged with any residual line from previous input. If the merged line's length exceeds the current column length setting, it is split into two LLEF lines - a primary and a residual line. The split occurs between words such that the primary line is equal to or less than the specified column length. The primary line is sent to the output device after being right-justified (if JUSTIFICATION-MODE is in effect). This process is repeated starting at step 1 above until the input files have all been processed.

LIST LINK ELEMENT FORM

As noted in the description of the main processing sequence, the most recent text input, while being manipulated, is stored in a list link element form (LLEF). Each character of text is physically stored in a separate link block. Pointers are used to indicate the order of the characters on the line and the occurrence of overprinted characters (e.g., underlined characters). All explicit backspace characters are removed since they are not necessary in the LLE form.

Figure 2 illustrates an example text line in three forms : (1) as a printed (graphic) line, (2) as a sequence of physical bytes as typed at the terminal, and (3) in the link list element form. Utility routines are incorporated into the SCRIPT program to convert text lines between physical byte strings and LLE form.

The list link element form is used for two different purposes in SCRIPT: (1) processing of overprinted characters on output, and (2) formatting of the line. The physical mechanism for producing overprinted characters is different for a terminal (i.e., uses character1 - backspace - character2 sequence) than for a line printer (i.e., print

Graphic Character String: A=BøC

Physical Character String: A<_=B<_o<ø<_C<_
(17 bytes, < represents backspace character)

Link List:

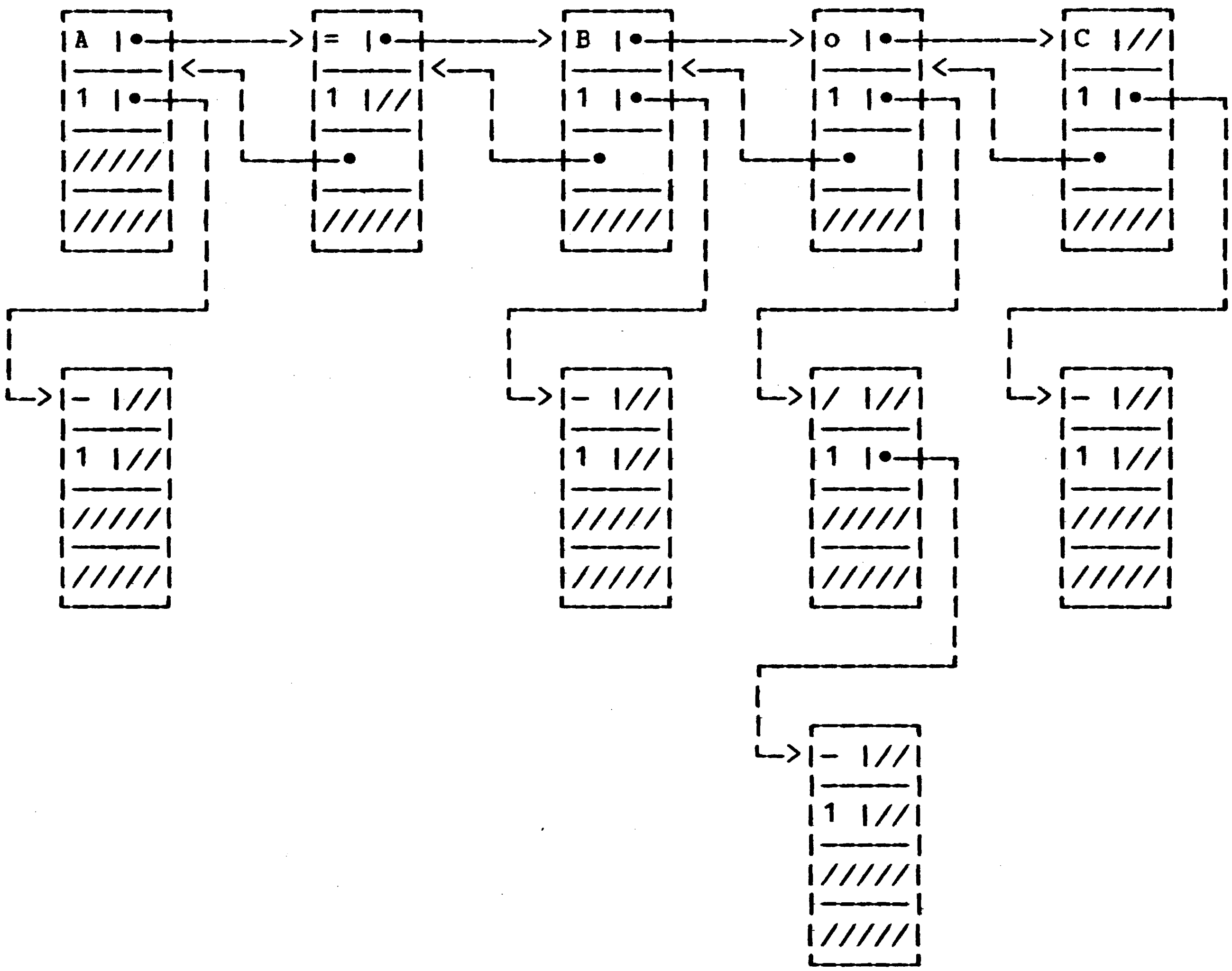


Figure 2. Example List Link Element Form.

entire line, and then, without advancing the paper, print characters). Straight-forward algorithms are provided in SCRIPT for converting a line in LLE form into the appropriate character string(s) needed for either terminal or line printer output format.

In the process of converting the input text into formatted output, especially for producing right margin justification, it is often necessary to split a line into two parts, merge two parts together, or convert a single blank into multiple blanks. These tasks are simplified by use of the LLE form for representing text lines internally.

The list link element format is further described and discussed in the IBM publication: "SCRIPT: An Online Manuscript Processing System" by S. E. Madnick and A. Moulton. This report has been published in the IEEE Transactions on Engineering Writing and Speech, Vol. EWS-4, No. 2, August 1968, and can be obtained by written request to IEEE at 345 East 47th Street, New York, New York 10017.

RIGHT MARGIN JUSTIFICATION ALGORITHM

Two passes over the primary line are needed to justify the left and right margins. During the first pass the primary line is scanned up to and including the last complete "word" (group of non-blank characters separated by blanks) contained within the length desired for justification. The number of words is determined along with the number of spaces remaining between the last word and the required line length. Dividing the number of spaces needed by one less than the number of words produces the number of extra spaces that should be inserted after each word for correct justification. Unfortunately, it is not possible to insert fractional spaces. Therefore the fractional components are accumulated until at least a half space is accumulated, a whole space is inserted into the line and subtracted from the accumulated sum of fractions.

The second pass is required to record the added spaces. Each link element contains a multiplier field initially set to one. During the second pass the multiplier for the appropriate blank link elements is increased without altering the data structure.

The LLE data structure is also used for a variety of other facilities, such as interpreting "tab" characters, and converting them to the appropriate number of blanks or user specified "pad" characters.

(The character "." represents a blank.)

```
|<---------------------Desired Length--------------------->|
 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 A  B  .  C  D  .  E  F  .   G  H  I  .  J  K  L  .  M  .  N  O  P  R  S
<---->   <---->   <---->    <------->   <------->   <->
  1        2        3           4           5        6

          Desired Line Length = 20
          Number of Complete Words - 1 = 5
          Number of Spaces Needed = 2
          Number of spaces to be inserted between every word = 0
          Number of Extra Spaces per Word = 2/5 or 4/10

|<---------------------Desired Length ------------------->|
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 A  B  .  C  D  .  .  E  F  .  G  H  I  .  .  J  K  L  .  M
     (4/10)    (8/10)     (2/10)       (6/10)          (0)
```

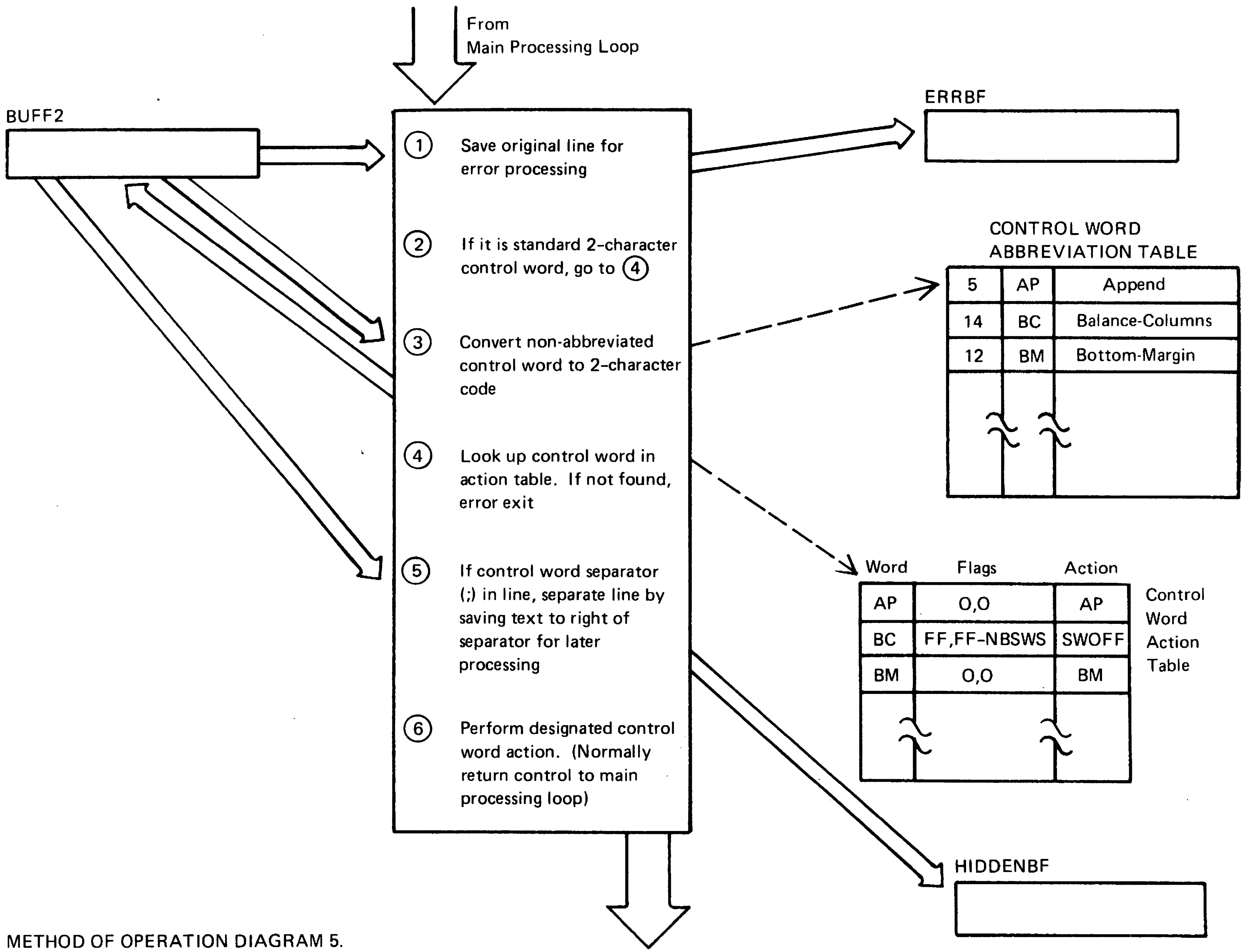
Figure 3. Justification Process

CONTROL WORD PROCESSING

When an input line starting with a period is read by the main processing loop, control is transferred to the control word processing routine (see Method of Operation Diagram 5).

The control word name, immediately following the period, may be in either of two forms: standard abbreviated form or unabbreviated (possibly truncated) form. For example, .BC is the standard abbreviated form for the Balance-Columns control word. On the other hand, .BALANCE-COLUMNS and .BALANCE are examples of the unabbreviated form of the Balance Columns control word, untruncated and truncated respectively. If the unabbreviated form is read, it is converted to the standard abbreviated form by the SABBREV routine.

The control word action table is searched, by binary look-up, for a match with the abbreviated control word. If a match is not found, an error exit is taken and an appropriate error diagnostic is printed. If a match is found, the action indicated in the table is performed.



METHOD OF OPERATION DIAGRAM 5.
CONTROL WORD PROCESSING

In general, one of three kinds of actions is performed:

1. A particular switch (binary variable) is set on or off.
2. A particular variable (or variables) is set to a specific value.
3. A special action routine is invoked.

In the first two cases the particular switch or variable set has its effect by being part of the normal computations of the main processing loop. For example, the JUSTIFICATION-MODE switch is tested and the COLUMN-LENGTH variable is used as part of the formatting illustrated in Method of Operation Diagram 4.

SWITCH SETTING:

Control words that merely cause switches to be set <u>on</u> or <u>off</u> are facilitated by means of the action routines SWON and SWOFF. All such switches are contained in the bytes SWITCH and/or AUGSW and the particular bits to be turned on or off are designated in the flag bytes of the control word action table.

The control words that fit into this category and their action are listed in Table 1.

Control Word	Abbreviated	Switches	ON/OFF
Balance-Columns	BC	NBCSWS	Off
Break	BR	None	-
Comment	CM	None	-
Concatenate-Mode	CO	NFSWS	Off
Format-Mode	FO	NFSWS,NJSWS	Off
Justification-Mode	JU	NJSWS	Off
No-Balance-Columns	NB	NBCSWS	On
No-Concatenate-Mode	NC	NFSWS	On
No-Format-Mode	NF	NFSWS,NJSWS	On
No-Justification-Mode	NJ	NJFSWS	On
Single-Space-Mode	SS	DSSWS	Off

Table 1.
Processing of Control Words that Set Switches.

VARIABLE SETTING:

Control words that primarily cause a variable to be set have separate short action routines. These action routines check that the value to be assigned to the variable is allowable. For example, the heading margin setting cannot exceed the top margin setting.

The control words that fit into this category and the corresponding variable(s) set are listed in Table 2.

Control Word	Abbreviated	Variables
Bottom-Margin	BM	BOTMRG
Center	CE	CECNT and CERISWS=off
Column-Length	CL	CECNT and CERISWS=off
Control-Word-Separator	CW	CSTABLE
Double-Space-Mode	DS	DSCNT=2 and DSSWS=on
Footing-Margin	FM	FTMRG
Heading-Marging	HM	HDMRG
Indent	IN	INDL, RMARGIN, OFFL, OFFLI
Literal	LI	LICNT
Line-Length	LL	LLZ (CLZ if CLSWS =off)
Line-Spacing	LS	DSCNT and DSSWS =on
Offset	OF	INDL, RMARGIN, OFFL, OFFLI
Page-Length	PL	PL, PLCT
Page-Number	PN	Switches: PAGOFF, PINCRNO, ROMANSW
Page-Number-Symbol	PS	PAGENSYM
Right-Adjust	RI	CECNT and CERISWS =on
Terminal-Input	TE	TECNT
Top Margin	TM	TOPMRG
Undent	UN	UNDL, RMARGIN

Table 2.
Processing of Control Words that
Set Variables.

SPECIAL ACTION ROUTINES:

The remaining 31 SCRIPT control words require special action routines to perform their functions. To simplify the explanations, these control words can be conveniently divided into ten groups:

1. Page titles
2. Page eject
3. Switch input file

4. Sectioning
5. Status handling
6. Symbol processing
7. Terminal I/O
8. Termination
9. Multiple Columns
10. Miscellaneous

The control words, their corresponding action routines name, and their division into these ten groups is illustrated in Table 3.

Control Word	Abbreviated	Name of Action Routn	Group
Append	AP	AP	3: Switch input file
Bottom-Title	BT	BT	1: Page titles
Column-Begin	CB	CB	9: Multiple column
Column-Definition	CD	CD	9: Multiple column
Conditional-Column-Begin	CC	CC	9: Multiple column
Conditional-Page-Eject	CP	CP	2: Page eject
Conditional-Section	CS	CS	4: Sections
Delay-Imbed	DI	DI	10: Miscellaneous
End-of-File	EF	EOFSET	8: Termination
Even-Page-Bottom-Title	EB	EB	1: Page Titles
Even-Page-Eject	EP	EP	2: Page Eject
Even-Page-Top-Title	ET	ET	1: Page Titles
Footing	FT	FT	1: Page Titles
Heading	HE	HE	1: Page Titles
Imbed	IM	IM	3: Switch input file
Odd-Page-Bottom-Title	OB	OB	1: Page Titles
Odd-Page-Eject	OP	OP	2: Page eject
Odd-Page-Top-Title	OT	OT	1: Page Titles
Page-Eject	PA	PA	2: Page Eject
Quit	QU	TRUEEND	8: Termination
Read-Terminal	RD	RD	7: Terminal I/O
Restore-Status	RE	RESTORE	5: Status
Revision-Code	RC	RC	4: Sections
Save-Status	SA	SAVE	5: Status
Set-Symbol	SE	SET	6: Symbols
Space-Line	SP	SP	10: Miscellaneous
Substitute-Symbol	SU	SUB	6: Symbols
Tab-Setting	TB	TB	10: Miscellaneous
Top-Title	TT	TT	1: Page Titles
Translate-Character	TR	TR	10: Miscellaneous
Type-on-Terminal	TY	TY	7: Terminal I/O

Table 3.
Processing of Control Words that
Require Special Action Routines.

GROUP 1 PROCESSING -- PAGE TITLE

Control Words:		Entry Point
.BT	Bottom-Title	(BRENTRY)
.EB	Even-Page-Bottom-Title	(EBENTRY)
.ET	Even-Page-Top-Title	(ETENTRY)
.FT	Footing	(BTENTRY)
.HE	Heading	(HEENTRY)
.OB	Odd-Page-Bottom-Title	(OBENTRY)
.OT	Odd-Page-Top-Title	(OTENTRY)
.TT	Top-Title	(TTENTRY)

The group 1 control words are processed by the STITLE CSECT within the SCSFOR module. There are multiple entry-points into this CSECT as indicated in the list above (e.g., BTENTRY, EBENTRY, etc.).

There are 12 buffer areas used by these routines. There is a set of three separate buffers kept for each odd-page top title, odd-page bottom title, even-page top title, and even-page bottom title (i.e. 3 buffers per title x 4 titles = 12 buffers). The three buffers per title are used to (1) hold the portion of the title to left-adjusted, (2) the portion of the title to be centered, and (3) the portion of the title to be right-adjusted. The four basic control words Even-Page Bottom Title, Even-Page Top Title, Odd-Page Bottom Title, and Odd-Page Top Title each directly fill in one of the four buffer sets. The control word Bottom Title fills in both the even-page and odd-page bottom title buffer sets, and similarly for the Top Title control word. The Heading and Footing control words operate similarly but only the left-adjusted portion of the title is retained from the control word. There is no centered positions and the right-adjusted portion is set to the characters PAGE &. The Heading and Footing control words are primarily provided for compatability with earlier versions of SCRIPT.

When the bottom of a page or top of a page are encountered during normal SCRIPT text formating, the FORMTITL entry-point into the STITLE CSECT is called. It selects the appropriate even/odd top/bottom title buffer set, formats it, and returns it for outputting. In order to perform this operation, FORMTITL is provided with the following information:

- current page number.
- current line length setting.
- whether top or bottom title requested.
- whether arabic or roman numerals are to be used for page numbering.
- whether page numbering is to be supressed for the Heading Control Word.

GROUP 2 PROCESSING -- PAGE EJECT

Control Words:

.CP	Conditional-Page-Eject
.EP	Even-Page-Eject
.OP	Odd-Page-Eject
.PA	Page-Eject

These control words are all variations on the basic Page-Eject control word. If the user specified number is less than the difference between PLCT (number of lines left on page) and BOTMRG (number of lines reserved for a bottom margin), a Page-Eject occurs. Otherwise, the control word is ignored.

The Even-Page Eject and Odd-Page Eject control words always cause a Page-Eject. They may cause one additional Page-Eject, if necessary, to make PAGEN (the current page number) even or odd, respectively.

The basic PAGE internal routine is quite important since it is also automatically invoked whenever the PLCT becomes less than or equal to BOTMRG during normal output formating. Each time an output line is produced, the PLCT is decremented and this check is made.

If the Page-Eject control word was used and the new page number is explicitly specified, the special entry PAGEZ into the PAGE routine is used. Otherwise, the PAGE entry-point is used and the new page number counter, NEWPAGEN, is set to PAGEN+1. PLCT - BOTMRG + FTMRG blank lines are generated to position the output forms to the bottom title location. A bottom title, if one had been specified earlier, or a blank line is generated. Then, BOTMRG - FTMRG + TOPMRG - HDMRG blank lines are generated to position the output forms to the top title location.

Before printing the top title on the new page, there are several checks made first. If the STOP option has been specified, a pause is generated to allow the user to manipulate the terminal paper. If the SINGLE page option had been specified, processing terminates. If the PAGEXXX option had been specified, the new page number is compared with the number specified by the user and the no print switch, NPSWS, is turned off if the page numbers match. Finally, the top title is printed and HDMRG blank lines are printed. This leaves the output form position for resuming text output on the new page. Control returns to PAGE's caller for processing of further control words or text.

GROUP 3 PROCESSING -- SWITCH INPUT FILE

Control Words:
.AP Append
.IM Imbed

These two control words involve very similar processing. The Append control word is somewhat simpler and will be described first. Before attempting to switch input files, a CMS STATE file system function is used to determine if the specified file exists. If it doesn't, an error exit is taken, otherwise processing continues. If any additional arguments, in addition to the file name, are specified, the special set symbols &0, &1, &2, etc. are defined via the SCSET routine (see Group 6 control word processing). The input file name is then changed and the file line number is reset. Then, if the Append control word was being processed, the old input file is closed and control returns to process the next control word or input text. All further input requests are directed to the new input file.

If the Imbed control word is encountered, there are a few differences. Imbed maintains a stack, up to 8 levels deep, that includes:

- file name
- file mode
- file line number
- hidden buffer (if additional control words or text were entered on the same line after the Imbed control word).

Before calling the common Append routine, Imbed sets aside all the necessary stack information. If the Append function returns correctly (i.e, there were no errors), the stack is updated.

The difference between Append and Imbed is much more obvious when the READ internal function is examined. It is responsible for reading the next line from the current input file. When it encounters an END-OF-FILE condition on a file, the input file is closed. It then examines the Imbed stack. If the stack is empty, processing is terminated - this is the normal termination condition. Otherwise, the top entry on the stack is removed and becomes the current input file and the read operation is retried. The Imbed stack operates in a last-in first-out (LIFO) mode. Thus, the reversion is to the input file that contained the Imbed control word that specified the current input file that has been processed.

GROUP 4 PROCESSING -- SECTIONING

Control Words:
.CS Conditional-Section
.RC Revision-Code

These two control words are used to delineate a section of the text. In the case of the Conditional-Section control word, the specified section may be included or excluded from the output text. In the case of the Revision-Code control word, the specified section may be marked by a designated revision code symbol in the left margin. The actual processing of these two control words involve numerous differences and will be explained separately.

The Conditional-Section processing requires the use of two byte arrays, each 9 bytes long - 1 byte for each of the 9 possible section codes. The arrays are called CSINCLUD and CSCURRON. CSINCLUD(n) is X'00' if section(n) is to be included or X'FF' if section(n) is to be ignored. CSCURRON(n) is X'FF' if processing is currently in section(n), otherwise it is X'00'. In addition, there is a single bit, called CSSWS, in the switch byte AUGSW2 that is set to B'1' if input text is currently being skipped (that is, CSCURRON(n) = X'FF' and CSINCLUD(n)= X'FF' for some value of i).

The .CS n INCLUDE or .CS n IGNORE control words merely set the CSINCLUDE array as described above. The .CS ON or OFF control words start by setting CSCURRON(n) = CSINCLUD(n) or CSCURRON(n) = X'00', respectively. Then, if any element of CSCURRON is X'FF', CSSWS is set to B'1', otherwise it is set to B'0'.

In the main SCRIPT processing cycle all input is ignored whenever the CSSWS is set to B'1', except for subsequent .CS n OFF control words.

The Revision-Code processing is similar but slightly more complex due to the need to handle revision codes. The primary data bases used are a revision code table, RCCHAR, and a stack, RCSTACK. When a non-blank revision code character is defined, it is placed into the RCCHAR byte array. Also, the output is shifted 3 spaces right to allow room for the revision code in the left margin. This shift is not necessary if there is already a shift in effect due to the CENTER or NUMBER options of SCRIPT.

When a .RC n ON control word is processed, any revision code currently in affect is saved in the RCSTACK. On the other hand, when a .RC n OFF is processed, the previous revision code, if any, is reinstated. At any time, RCCURR contains the current revision code number in affect or zero if none is in effect. Also, RCCHAR(0) is set to the corresponding revision code character. In the normal SCRIPT processing, whenever an output line is generated by the PRINT function, as opposed to lines that are actually skipped, the current revision code character, if any, is inserted into the left margin.

There is a special case that must be handled carefully. A very short line may be preceeded and followed by RC ON and RC OFF, respectively. In this case, the revision code may be turned on and then turned off before the output text line is filled and printed. In this case, there would be no revision code appearing on the output. To handle this case, a special RCSTALL variable is used. If, at the time that an RC OFF is processed, there is text stored in the internal residual input buffer, RCSTALL is set to the current revision code character. When the next line of output is generated, the RCSTALL character is placed in the left margin and automatically reset to be blank.

GROUP 5 PROCESSING -- STATUS HANDLING

Control Words:
- .RE Restore-Status
- .SA Save-Status

The Save and Restore Status control words use a stack to save/restore the current state of: (1) the binary switch bytes (e.g., SWITCH, AUGSW, etc.), (2) the control word variables (e.g., PL, LL, etc.) and (3) the output translate table.

When the first Save Status control word is encountered, sufficient space to have a 5 level stack is allocated via the GETMAIN macro. In the current version each stack level requires 475 bytes or a total of 2375 bytes for a 5 level stack. At the completion of processing, the stack area, if allocated, is released via the FREEMAIN macro. If the Save Status control word is not used in the SCRIPT input, the stack area will not be allocated and the GETMAIN/FREEMAIN will not be required.

GROUP 6 PROCESSING -- SYMBOL PROCESSING

Control Words:
- .SE Set-Symbol
- .SU Substitute-Symbol

The processing of symbols in SCRIPT is quite elaborate and is handled principally by the SCSYM CSECT. The Set Symbol control word is processed by the SCSET entry-point of SCSYM. The Substitute-Symbol control word does very little work directly, it merely sets or resets the SUBCNT variable. The actual substitution of symbols is performed by the SCSUB and SCSUB2 entry-point of SCSYM; these entries are invoked automatically by the internal READ function which provides the next input line to the basic SCRIPT processing cycle. After reading each input line, READ calls SCSUB if SUBCNT is non-zero. If substitution for a set symbol array reference, such as &REFERENCES(*), results in a line longer than 130 characters, substitution is only performed up to that point. After SCRIPT has processed this portion of the line, instead of reading a new input line, READ calls the SCSUB2 entry to get the next portion of the substitution. When the substitution has been completed, READ reverts to reading new input lines.

The set symbols are stored in a symbol table that is dynamically allocated via GETMAIN when the first set symbol is defined. If the symbol table is used, it occupies all of available GETMAIN space minus 24K which is reserved for other uses. Each individual set symbol requires 32 bytes in the symbol table. In a typical 320K CMS virtual machine, there is about 160K available for the symbol table which is sufficient to handle 5000 symbols.

The 32 byte symbol table entry is used as follows:

- TABSYM = 11 bytes for symbol's name, such as ALPHA.
- TABDATA = 14 bytes for the symbol's value, such as "HELLO".
- TABDLING = 1 byte to indicate actual length of symbol's value.
- TABIDX = 2 bytes for index subscript, if any, such as in &ALPHA(4).
- TABPTR = 4 bytes for pointer to next array element, if subscripted symbol.

Figure 4 illustrates the structure of the symbol table. Note that all subscripted set symbols, such as &BETA(1), &BETA(2), and &BETA(4), are chained together

Entry Number	Name	Value	Value Length	Index	Next Element
1	ALPHA	"14"	2	0	—
2	BETA	"2"	1	0	3
3	BETA	"John"	4	1	6
4	GAMMA	"14"	2	0	—
5	BETA	"Stu"	3	4	—
6	BETA	"Madnick"	7	2	5

(a) Symbol Table

```
.Substitute on
.SET ALPHA = 14
.SET BETA() = 'John'       —sets  BETA(1)
.SET GAMMA = &ALPHA
.SET BETA(4)  = 'Stu'
.SET BETA() = 'Madnick'    —sets  BETA(2)
```

(b) Set-Symbol Sequence that Produces Symbol Table

Figure 4. Set-Symbol Table

in order of their index. The chain starts at the "master" symbol for the array, which is called either &BETA or &BETA(0) for the example above. Recall that the single SCRIPT control word

```
.SET  BETA()='Madnick'
```

is identical to the sequence:

```
.SET  BETA  =&BETA+1
.SET  BETA(&BETA)='Madnick'
```

which accounts for &BETA(0) having a value of "2" and &BETA(1) and &BETA(2) being set as indicated in Figure 4. If &BETA (or, equivalently, &BETA(0)) had not been initialized in advance by the user, it is automatically set to zero by the first &BETA() reference.

When the symbol table is allocated, the special symbols &SYSYEAR, &SYSMONTH, &SYSDAYOFY, &SYSDAYOFM, &SYSDAYOFW, &SYSHOUR, &SYSMINUTE, and &SYSECOND are initialized and stored in the table.

The SCSET entry into the SCSYM CSECT handles the Set-Symbol control word. The SPARSE internal function is used to parse the free form control word into a list of tokens, each 15 bytes long. For example, the control word

```
.SET  GAMMA() = &ALPHA+1
```

would be converted into a list of 8 tokens as follows:

	Length	Value
1.	4	".SET"
2.	5	"GAMMA"
3.	X'FF'	"("
4.	X'FF'	")"
5.	X'FF'	"="
6.	6	"&ALPHA"
7.	X'FF'	"+"
8.	1	"1"

Special 1-byte break characters, such as =, (,), +, -, *, and /, have the number 255 (X'FF') stored in the length byte.

The processing is neatly divided into two stages, first the left side of equal sign is handled and then the right side. The left side must be one of three basic forms:

```
1.  .SET SYMBOL = ---
2.  .SET SYMBOL() = ---
3.  .SET SYMBOL(n)=---
```

Each case is handled somewhat differently. In all cases, the corresponding entry in the symbol table is found or created, if it did not previously exist in the symbol table.

The right side of the equal sign may be either a single token or an arithmetic expression. If it is a single token, it may be either quoted or unquoted, for example:

```
1.  .SET W = 'HELLO'
2.  .SET X = HELLO
3.  .SET Y = '*&X.*'
4.  .SET Z = *&X.*
```

Cases 1 and 2 are treated exactly the same, the quotes are only needed if there are imbedded blanks. Cases 3 and 4 are handled differently. Y will be assigned the value "*&X.*" since set substitution for &X is suppressed, whereas Z will be the value "*HELLO*" since set substitution for &X is requested. NOTE: Since the general Substitute-Symbol ON control word takes affect immediately after reading the input line, the symbol &X will be substituted in both cases 3 and 4 even before processing by the Set-Symbol control word <u>if</u> Substitute-Symbol ON mode is active. If case 3 is to set Y to the value "*&X.*", then Substitute-Symbol OFF must be in effect.

If the right side of the equal sign is an arithmetic expression, it is evaluated left to right. Constants, such as 14, are converted to binary for computation and symbols, such as &COUNT, are retrieved from the symbol table and converted to binary for computation.

After the right side has been processed, the resulting value is stored in the symbol table entry located during the first step.

The SCSUB entry of the SCSYM CSECT performs the symbol substitution function. It is automatically called by the READ routine to scan and process each input line if substitution mode is in affect. The line is scanned left to right for set symbols. After substitution for a symbol, the line is rescanned left to right. The substituion is complete when a scan is made that does not find a set symbol to be substituted. For example, the sequence:

```
.SET    X  = 'A'
.SET    Y  =  2
.SET    A1 =  'Monday'
.SET    A2 =  'Tuesday'
Today is &&X.&Y
```

will result in the following substitutions:

```
1.  Today is &&X.&Y        (scan and find &X.)
2.  Today is &A&Y          (substitute for &X.)
3.  Today is &A&Y          (scan and find &Y)
4.  Today is &A2           (substitute for &Y)
5.  Today is &A2           (scan and find &A2)
6.  Today is Tuesday       (substitute for &A2)
```

If a complete array substitution is requested, such as &BETA(*), all elements of &BETA, except for &BETA(0), are substituted. The elements are separated by a comma and a blank. If the sequence illustrated in Figure 4 had been processed, the text:

```
.SUBSTITUTE; The names are &BETA(*)..
```

would result in the line:

```
The names are John, Madnick, Stu.
```

If such an array substitution causes the line to exceed 130 bytes, further substitution is suspended and the substituted portion is provided to the READ routine for processing along with a return code indicating that the substitution is incomplete. The entry SCSUB2 may be used to resume substitution and return the next portion. SCSUB2 should be called repeatedly until it returns a code that indicates that substituion has been entirely completed.

GROUP 7 PROCESSING -- TERMINAL I/O

Control Words:
.RD Read-Terminal
.TY Type-on-Terminal

These control words are quite simple. For the Read-Terminal control word, a WAITRD console I/O function is invoked as many times as specified. The actual line entered is stored in the BUFF2 area and is ignored and/or overlayed by the next WAITRD. If output is not to the console, the corresponding number of blank lines are generated.

For the Type-On-Terminal control word, a TYPLIN console I/O function is invoked to print the message specified. This message is always printed on the terminal regardless of the output device specified for the formatted SCRIPT output.

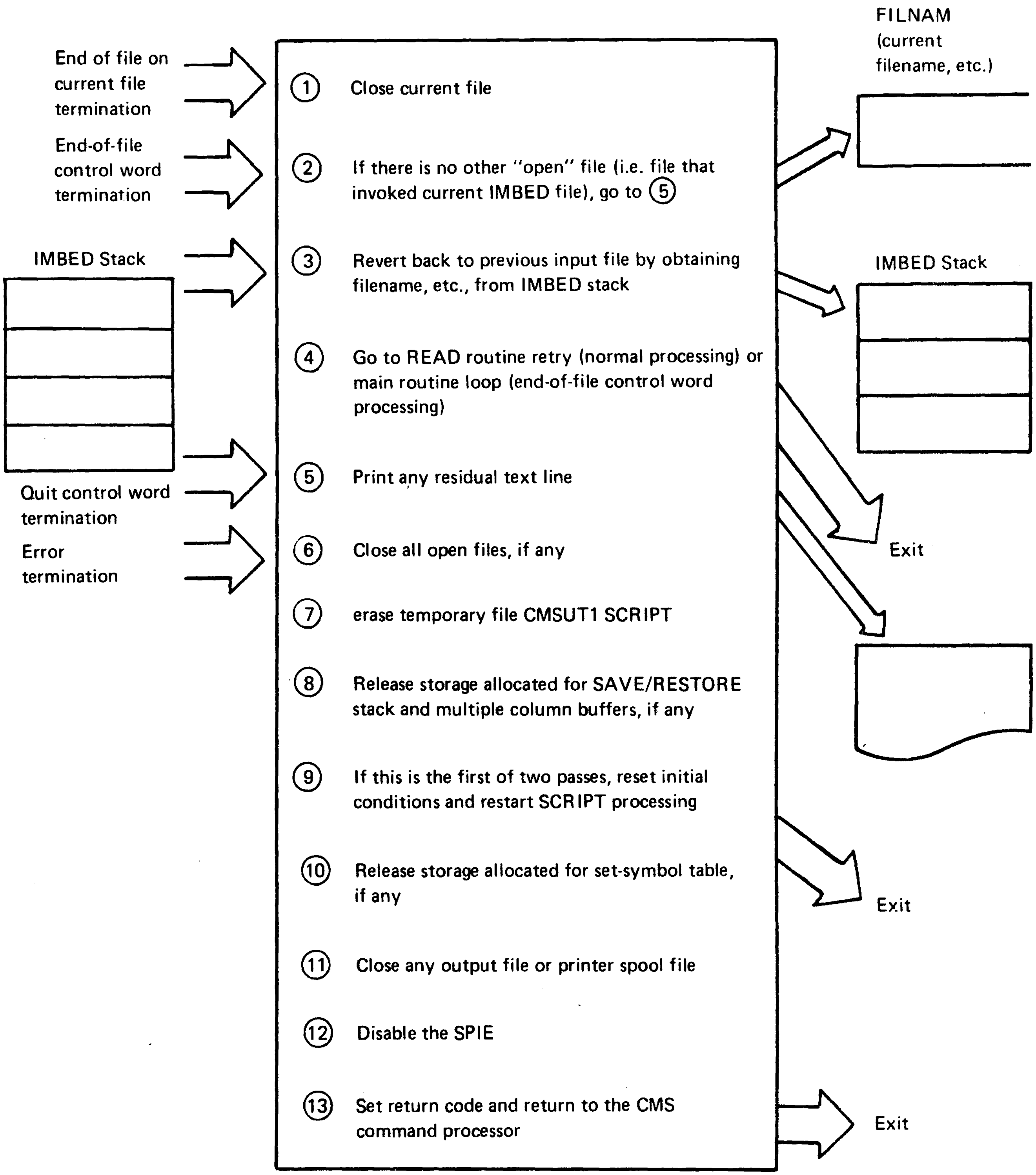
GROUP 8 PROCESSING -- TERMINATION

Control Words:
.EF End-of-File
.QU Quit

Both of these control words utilize the standard SCRIPT termination sequence (see Method of Operation Diagram 6), but at different entry stages. The End-of-File control word simulates the end-of-file error return for the current file and uses exactly the same processing sequence, except that the current file is <u>not</u> closed (i.e., the call to the CMS file system function FINIS is bypassed). This corresponds to step 2 in Method of Operation Diagram 6.

The End-of-File control word causes input processing to revert to the file that invoked the current imbed file. If the current file was not imbedded, then processing terminates. The Quit control word, on the other hand, results in an unconditional immediate termination. This corresponds to step 5 in Method of Operation Diagram 6.

Termination processing is explained further in the TERMINATION section.



METHOD OF OPERATION DIAGRAM 6. TERMINATION

GROUP 9 PROCESSING -- MULTIPLE COLUMN

Control Words:
- .CB Column-Begin
- .CC Conditional-Column-Begin
- .CD Column-Definition

The processing of these control words, as well as the Balance-Columns (.BC), No-Balance-Columns (.NB), and Column-Length (.CL) control words, are all highly related to the multiple column processing mechanism used in SCRIPT. Thus, the overall mechanism will be explained before elaborating upon the processing of the individual control words.

When operating under standard single-column format, SCRIPT directly outputs a text line as soon as it has been completely formatted. Thus, at any time, there is at most a single line of text, the residual line, buffered in main storage. This line is kept in linked list element form (LLEF). When operating under multiple-column format, the entire page of text must be formatted and saved internally before any output can be generated. After the page has been completed, the columns can be balanced, if necessary, and the composite multiple-column output lines can be produced.

Storage space to save the page of text is allocated via GETMAIN. A variable conditional GETMAIN request is issued for 21120 bytes (approximately 160 lines of 132 bytes each buffer capacity), although processing will proceed if as little as 4096 bytes of buffer space is available. If during processing, there is insufficient buffer space to hold a complete page of text, an error termination occurs. This error could occur if the page size is defined very large (for example, 400 lines per page) or if the user's virtual machine is so small that there is little buffer space available.

The text lines are stored as variable-length character strings within the allocated storage buffer area. Each data line starts on a half-word boundary and begins with a four-byte header followed by the actual text. The header consists of two half-word fields: a relative pointer to the next data line and the actual length of the current data line. Figure 5 presents an example of the text storage format. The four text lines illustrated are 15 bytes (X'000F'), 7 bytes (X'0007'), 1 byte (X'0001'), and 16 bytes (X'0010') long, respectively. Each data line specifies the relative address of the next data line. The absolute address is computed by adding the relative address to the base address of the storage area.

Text lines (4 lines)

1. This is line 1.
2. Line 2.
3.
4. Third data line.

Storage Buffer Area

Assume storage buffer starts at location X'28000'.

```
Location          Data Lines

X'28000'          X'0014',X'000F',C'This is line 1.'
X'28014'          X'0020'X0007',C'Line 2.'
X'28020'          X'0026',X'0001',C' '
X'28026'          X'0000',X'0010',C'Third data line.'
                    A        A        A
                    |        |        └─Data
                    |        |
                    |        └─Data length
                    |
                    └─Relative address of next data line
```

Figure 5. Example of Text Storage for Multiple Column Processing

For example, the relative address X'0014' specified on the first data line indicates that the second data line starts at X'28000' + X'0014' = X'28014'.

The data line with its associated header information is called a Line Control Block (LCB). The LCB's are grouped into "areas" (i.e., columns). There is a separate Area Control Block (ACB) for each group of LCB's. Each ACB specifies the address of the first LCB and last LCB in the area as well as a count of the number of LCB's in the area. All the LCB's in an area group are chained together via the "next LCB" relative pointer in the LCB header. Figure 6 illustrates the relationship between the ACB's and LCB's. There are 2 ACB's in use, each represents a logical sequence of print lines. The reader should examine the figure and decipher the messages represented by each ACB.

In order to manage the overall buffer storage space, there is a single Storage Control Block (SCB). Most of the SCB information is quite static, such as the location (base address) of the buffer, the size of the buffer, the ending address of the buffer and the minimum/maximum space to be requested via GETMAIN. The SCBNEXT field is the most active data element of the SCB. The buffer space is allocated on a "wrap-around" basis. LCB's are allocated space one after another starting at the beginning of the buffer area. When the end of the buffer is reached, allocation restarts at the beginning again. The SCBNEXT field specifies the address of the buffer space to be allocated next. Some of the more subtle details will be explained below. It should be noted that conventional "garbage collection" techniques are not used and the buffer space is sequentially allocated.

Now that the LCB, ACB, and SCB mechanisms have been described, the overall multiple column processing technique can be explained. SCRIPT currently uses nine ACB's, called COLACBS, one for each of up to 9 text columns. The input text lines are formatted exactly as if it were single column processing. The Column-Length or Line-Length settings control the length of each generated line if FORMAT mode is in effect. Instead of outputting each formatted line onto the terminal, printer, or file, it is stored in the buffer area via the LSTORE routine and associated with COLACB(1), the first ACB. When the first column has been completed, either by filling all the lines or by an explicit Column-Begin control word, the page line counter, PLCT, is reset to the top of the column and formatting continues -- this time using COLACB(2). This process continues for as many columns as the user designated on the Column-Definition control word or until an explicit Page-Eject control word is encountered. This procedure is only used if two or more columns have been specified or if a single column has been specified but it is not to start in the first print position of the line according to the user's Column-Definition (e.g.,

Location	LCB's (in buffer area)
X'28000'	X'0018', X'0007', C'This is'
X'28008'	X'0000', X'0006', C'wrong.'
X'2800E'	X'0014', X'0005', C'It is'
X'28014'	X'0008', X'0003', C'not'
X'28018'	X'0000', X'000B', C'an example.'

ACB1

ACBFIRST = X'28000'
ACBLAST = X'28018'
ACBLINES = X'00002'

ACB2

ACBFIRST = X'2800E'
ACBLAST = X'28008'
ACBLINES = X'00003'

Figure 6. Relationship Between Area Control Blocks (ACBs) and Line Control Blocks (LCBs)

.CD 1 10).
After all the required columns have been filled or there has been an explicit Page-Eject control word, preparation for actual output commences. If column balancing is in effect and there has not been any explicit Column-Begin control word, the columns are balanced. This is accomplished by evenly distributing the LCB's among the ACB's. This actually involves several steps. First, all the LCB's are combined into a single LCB chain. The number of LCB's is divided by the number of columns to determine the appropriate number of LCB's desired per column. The one long LCB chain is then subdivided into sequences of the appropriate length and assigned to the ACB's.

The output lines are formed by getting one line out of each ACB via the LFETCH routine. Each line is positioned in the output buffer space as specified in the user's Column-Definition control word. The line is then outputted to the terminal, printer, or file. This process continues for each line of the page until all the ACB's are empty. The LFETCH routine automatically deletes the LCB from the buffer area and sets the storage space to zero. If the LSTORE routine ever attempts to allocate an LCB in space that is non-zero, this would indicate that the buffer space was too small and an error exit would occur.

After the entire page of text has been extracted from the buffer space and outputted, normal sequencing continues. Footings and headings are generated and processing of the next page of text commences. Thus, in multiple column processing, an entire page of text time is kept in main storage.

<u>GROUP 10 PROCESSING</u> -- <u>MISCELLANEOUS</u>

Control words:
- .DI Delay-Imbed
- .SP Space-Lines
- .TB Tab-Setting
- .TR Translate-Character

These control words are each handled quite specially and will be explained separately. The Delay-Imbed control word copies the designated input text lines directly into a temporary file named CMSUT1 SCRIPT. The CMS file system functions, such as ERASE, WRBUF, and FINIS, are used for this purpose, in addition to the SCRIPT READ routine. While copying, each line of input must be examined for the possibility of a .DI OFF if the Delay-Imbed was invoked by a .DI ON. When the copying is completed either by the user specified line count or a .DI OFF, the DIPENDFG flag is set and control returns to the SCRIPT main processing loop. Normal SCRIPT processing resumes. After every page eject, whether automatic or explicit, the DICHK routine is always invoked. If the DIPENDFG flag is not set, DICHK does nothing. If the flag has been set, any residual text lines are appended to the CMSUT1 SCRIPT file and the DINEXTFG flag is set. Control returns to the normal SCRIPT processing. Eventually, the READ routine is used to get the next input. At that time the DINEXTFG flag is examined. If it is set, the line ".IM CMSUT1" is returned as the next line to be processed. From this point on, processing is handled by the Imbed control word routine.

The Space-Lines control word routine determines the number of lines to be spaced by multiplying the user specified count by the line spacing count as previously set by the Single-Space, Double-Space, or Line-Spacing control words. The SPACER utility routine is used to actually produce the appropriate number of blank output lines.

The Tab-Setting control word routine uses two tables named TABS and NEWTABS. Each table contains a list of tab settings, each entry is 2 bytes long. One byte specifies the column position, the other byte specifies the fill character to be used. The TABS table contains the default tab settings. When a Tab-Setting control word is encountered, the NEWTABS table is set as specified. The TABTAB pointer points to either TABS or NEWTABS. It is initially set to TABS and is reset to TABS whenever a Tab-Setting control word with no settings is encountered. TABTAB is set to NEWTABS whenever new tab settings are in effect. The SCRIPT line formatting routines use the tab settings to produce the desired result. While each input line is in the Linked List Element Form, the tab character is converted to the "fill" character, which is normally a blank, and the character multiplier is set to generate the appropriate number of "fill" characters. When the line is later linearized for output, the desired effect takes place.

The Translate-Character control word routine uses a 256 byte table named TRANTAB. This table is used by means of the 370 TRANSLATE (TR) machine instruction. If either the TRANSLATE option was specified in the SCRIPT command line or any Translate-Character control word was processed, the TRSWS flag is set. Subsequently, every output line is subjected to translation immediately prior to actual output. Each Translate-Character control word sets one byte in the TRANTAB table.

TERMINATION

The SCRIPT termination processing has already been covered in the section explaining the Group 8 control words, End-of-File and QUIT. Method of Operation Diagram 6 describes the overall process.

Termination processing may be invoked due to the following events:

1. Encountering a physical end-of-file or End-of-File control word while processing the primary input file.

2. A Quit control word.

3. An error condition (certain error conditions are recoverable and do not cause termination if the CONTINUE option had been specified).

SECTION 3: PROGRAM ORGANIZATION AND DIRECTORY

This section lists the SCRIPT program routines and describes their function. Each individual assembly module, control section (CESCT), and entry name is identified.

PROGRAM ORGANIZATION

The SCRIPT program consists of four separate assembly modules named: SCSPRT, SCSFOR, SCSLNK, and SCSLIN. The SCSPRT module, which contains the SCRIPT CSECT, is the primary routine of the program. It includes the initialization, main processing loop, and termination functions of the SCRIPT program. The other assembly modules, and their associated CSECTs and ENTRYs, serve as utility routines to the SCSPRT module. Thus, the SCRIPT program structure has only a two level module hierarchy.

MODULE DIRECTORY

As noted above, the SCRIPT program consists of four assembly modules. These assembly modules are further divided into 13 control sections (CSECTS). In addition to the CSECT names, there are 31 entry-points into these control sections. The module directory, depicted in Figure 7, indicates the hierarchical relationship between assembly modules, control sections, and entries. For each control section and entry there is a brief description of that entry's function.

INTERNAL SUBROUTINES OF SCSPRT

The SCSPRT assembly module (SCRIPT CSECT) contains an action for every SCRIPT control word. Many of these actions merely involve setting a binary switch and are performed by using a common routine with a parameter. The other actions require subroutines internal to the SCSPRT module. In some cases entries into the other assembly modules are used to perform part of the control word action. These control word

Assembly Module	Control Sections (CSECTs)	Entries	Description
SCSPRT			
	SCRIPT		Main module.
		SPRT	Alternate name.
		SCRIPT2	Alternate name.
		SCSPRT	Alternate name.
	TAB		Control word table.
SCSFOR			
	SCSFOR		Dummy CSECT.
	SFOR		Merge, truncate and adjust LLE lines.
		MERGE	Append new line to residual line.
		ADJUST	Insert fill characters.
		CENTER	Center or right-adjust line.
	SLNK		Process Link List Element (LLE) lines.
		LINKINIT	Initialize link list storage.
		LINKPUT	Convert string to LLEF.
		LINKGETT	Convert LLEF to typewriter format.
		LINKGETP	Convert LLEF to printer format.
		LINKSTAR	Pointer to start of free list.
	LINKAREA		Link List Element storage area.
	STITLE		Process title control words.
		ETENTRY	Process even-top title.
		EBENTRY	Process even-bottom title.
		BTENTRY	Process bottom title.
		TTENTRY	Process top title.
		OTENTRY	Process odd-top title.
		OBENTRY	Process odd-bottom title.
		HEENTRY	Process heading title.
		FORMTITL	Format and return appropriate title.
		PSENTRY	Set page-number-symbol character.
	SABBREV		Convert unabbreviated control word.
		SPRCWORD	Print list of legal control words.

Figure 7. Module Directory (Part 1 of 2)

Assembly Module	Control Sections (CSECTs)	Entries	Description
SCSLNK			
	SCSLNK		Dummy CSECT.
	SERR		Print error messages.
	SERRM		Error message table.
	SCSYM		Process set-symbol and substitution.
		SCSET	Process set-symbol control word.
		SCSUB	Scan line and substitute symbols.
		SCSUB2	Resume substitution if SCSUB overflow.
		TABCLOSE	Deallocate symbol table space.
SCSLIN			
	SCSLIN		Process multiple column format.
		LOPEN	Initialize buffer space.
		LCLOSE	Deallocate buffer space.
		LSTORE	Store a line in buffer.
		LPAGE	Form a multiple-column line.
		LFETCH	Fetch a single line from buffer.
		LBALANCE	Rearrange buffer lines to balance column

Figure 7. Module Directory (Part 2 of 2)

subroutines are identified in the assembly listing for SCSPRT. The mapping from control word to action routine is specified by the TAB CSECT which is also part of the SCSPRT assembly module.

The following utility subroutines are internal to the SCSPRT assembly module:

Entry	Function
COLUMN	Start new column (invoked by Column-Begin or bottom of page).
COLDUMP	Output a page of multiple column text that has been previously stored in the multiple column buffer area.
CVB	Convert EBCDIC character string to a binary number.
GETARG	Scan input line in BUFF2 for location of first argument.
GETNUM	Scan input line in BUFF2 and convert argument to binary number.
GPARSE	Scan input line in BUFF2 and convert in to sequence of 8-byte tokens.
IOPRINT	Direct output line to the appropriate output device (terminal, printer, or file).
PARMROUT	Process the CMS Parameter List and set the appropriate binary switches and variables.
PAGE	Generate a page-eject including bottom and top titles required.
PRINT	Convert text line from link list element form (LLEF) to linear form and initiate output operations (uses the PRINT1, PRINT2, and PRINT3 utility routines).
PRINT1	Adjusts the character multiplier of a LLEF line to account for amount to be centered.
PRINT2	Convert LLEF line to linear form appropriate for output device.
PRINT3	Cause the linearized line to be outputed (via IOPRINT).
READ	Read next input line into BUFF2 area. Other related activities are initiated in this

routine, such as set-symbol substitution and end-of-file processing.

SPACER Generate the appropriate number of blank lines.

SWON/SWOFF OR or AND, respectively, the 2-byte argument with the SWITCH and AUGSW switch bytes.

SECTION 4: DIAGNOSTIC AIDS

This section describes the error handling procedure employed by SCRIPT. It also outlines facilities provided in SCRIPT that assist in the debugging process.

ERROR HANDLING

Whenever an error is detected during SCRIPT processing, the SERR routine is invoked for error handling. It is provided with the following information in its parameter list:

1. Index code (error number x 4).
2. Last control word line.
3. Cumulative input line counter.
4. Current file name.
5. Current file line number.
6. Number of active files.
7. Pointer to the Imbed stack.

Associated with each index code there is: error message text, an action code, and a return code. Depending upon the action codes, the count of lines read, the trace back of Imbeds, and/or the last control word line will be printed on the terminal. There are three possible return actions: (1) do not allow error retry, use standard termination sequence, (2) allow error retry if user had specified the CONTINUE option, or (3) terminate immediately without completing the normal termination sequence.

DEBUGGING FACILITIES

In addition to conventional debugging techniques, SCRIPT provides two addition diagnostic aids. By specifying DEBUG as a SCRIPT command option, the SPIE program interrupt processor will be inhibited. Under these circumstances, the CMS DEBUG program can be used for interactive examination of SCRIPT variables and flow of control. See the _IBM Virtual Machine Facility/370 Programmer's Guide to Debugging_, GC20-1807, for additional information.

Program errors are difficult to isolate if the error is allowed to propagate its effect so that the error condition is not detected until much later during processing. To help isolate errors, SCRIPT checks the legality of variables at various points during processing. If an internal variable is found to be invalid, processing terminates immediately with an appropriate error message. The SCRIPT user is advised to forward the error message printout to the appropriate programming personnel for error analysis. A similar procedure is used if the SCRIPT program interrupt handler is activated (established via the SPIE macro instruction unless the DEBUG option has been used).

REGISTER USE

The register usage for the SCSPRT module are listed below.

Register	Use
0	Work register.
1	Contains address of parameter lists, also used as work register.
2	work register.
3	Work register.
4	Work register.
5	Linkage resiter (return address) for internal subroutines, also used as work register.
6	Base register.
7	Work register.
8	Base register.
9	Work register.
10	Work register.
11	Work register.
12	Work register.
13	Contains address of save area, simultaneously serves as a base register.
14	Linkage register (return address) to the CMS command processor, also used as work register.
15	Linkage register (entry address) to SCRIPT external utility routines.

LY20-0762-0

SCRIPT/370 Text Processing Facility Under Virtual Machine Facility Systems Guide Printed in U.S.A. LY20-0762-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)