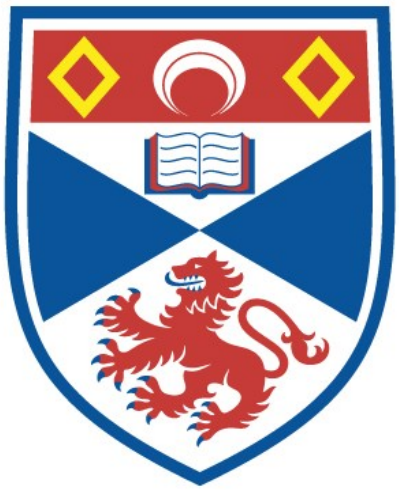
University of St Andrews



Full metadata for this thesis is available in
St Andrews Research Repository
at:
http://research-repository.st-andrews.ac.uk/

This thesis is protected by original copyright

TED - A PORTABLE CONTEXT EDITOR

MSc. THESIS

BY

EDITH HOWSON

This thesis describes the design and programming involved in implementing a portable context editor on various computers.

The research for the thesis was carried out at St.Andrews university by the undersigned under the supervision of Mr R. Morrison. The computer used was an IBM 360/44 sited at the university.

C O N T E N T S

1 INTRODUCTION

1.1 Context Editors

An essential part of any computer environment is a means of altering and revising text permanently stored within the computer system on magnetic storage devices.

In past years text was handled using punched cards or paper tape which could be manually altered. The fact that cards in a deck could be easily changed, inserted or deleted made the card storage system especially popular though somewhat bulky. Now, with the increasing use of magnetic storage it has become necessary to devise a method of updating files stored on these devices. Since the user has no access to such files except via the computer, a program is required to modify them. Such a program is called an editor.

The computer environment dictates the method of execution of an editor. It can be executed either in an on-line or a batch mode. An on-line editor is a very powerful tool. Edit commands are input via a remote access device eg teletype or visual display unit, and error output and verification listings are output to the same device. Using this method of execution the user can correct errors as they occur and can keep track of his position in the edited file. An on-line editor provides the user with a quick, convenient method of altering information stored on magnetic storage devices.

With a batch editor, on the other hand, the user has no control over his files during an edit run. Although the input commands are similar to these of the on-line editor, the error handling facilities must be more sophisticated. When errors occur the editor must decide whether it is safe to continue editing or not. Before a batch edit run is constructed it is advisable to have a printed copy of the file being edited to ensure the text being altered is uniquely identified. This precaution is not necessary with an on-line editor.

As part of the process of designing a new editor a study of some existing editors was made to establish what useful facilities and techniques could be incorporated into the new editor. The three editors examined in detail were a machine independent editor MITEM[4], the KDF9 COTAN ammender[5] and the UNIVAC 1108 editor ED[6].

MITEM comes in 6 versions graded from simple basic edit commands in the first version to powerful text manipulation facilities such as file merging and complex pattern matching in the sixth. Each version is a subset of the next highest level. The command language for MITEM is, I found, rather complex. The input/output line buffer concept for keeping the current line and the edited line was adopted in the new context editor. This allows for maximum manipulation of the current line and makes the backspace and start facilities simpler to implement (Section 2).

The editor used on the KDF9 computer is the COTAN ammender. This editor is not machine independent. The command language of the ammender is very much simpler than that of MITEM and consists of simple mnemonics with the use of non-alphameric characters only to delimit line or character commands. This command language was adopted as the base of the new editor's command language. The COTAN ammender runs in a very suitable on-line environment which gives it a lot of power. Commands are input to a file which can be altered in any way before being input to the ammender itself.

The UNIVAC 1108 editor has a very simple mnemonic command language but has rather limited context editing facilities. It tends to edit the file by the line numbers rather than the context, although it does have some context editing facilities. The tab facility and the facility for altering the field size are useful ideas and are adapted in the new editor. The facility for changing one string of characters for another is a good idea and although incorporated into the new editor its method of execution is entirely different (see exchange facility - Section 2).

A fuller discussion of current methods of on-line text editing is given in the paper, on-line Text Editing - A Survey[11].

From a user's point of view a context editor is only effective if it is easy to use and understand. The main factor in making an editor easy to use is the design of the command language. This should consist of simple and concise mnemonics and have some uniformity with as few restrictions as possible. The more powerful the commands become the more complicated the language structure. It is necessary, therefore, to establish a balance between powerful commands and simple command language.

A good error diagnostic system is also helpful to the user. Errors should be stated clearly and, in batch mode, appropriate action taken either to recover or to terminate the editor depending on the error type. In on-line mode the user determines the action to be taken when an error occurs.

1.2 Portable Software

As many aspects of software are standard on most computers, it seems rational to develop techniques whereby software written for one computer can be used on other computers with minimal changes. This would greatly reduce duplication of programming effort.

High level languages eg Fortran and Algol, go some way towards achieving portability. In theory, any computer supplied with the required compiler can execute programs written in the specified high level language. This is not always the case as there are many differences in high level language compilers and standards alter from one computer manufacturer to another. The idea in "Portable Software" is to produce a method of taking software written for one type of computer and, with minimum effort use it on another computer.

The technique used in this project is abstract machine modelling[2] and

realising the abstract machine on a real computer using a macro processor. This is the technique used in MITEM to achieve its machine independence. An abstract machine is one designed to carry out a specific task - in this case context editing. The operations and requirements of such an abstract machine must be clearly defined. Then the abstract machine must be realised on a real computer. The Stage 2 macro processor[4] is used for this task. Each operation of the abstract machine is described as a macro written in the assembler language of the real computer. When the macros have been processed the final output is the edit machine ready to be executed on the real computer.

To execute the editor on another computer the macros will have to be rewritten in the required assembly language. The portability of the editor depends on the simplicity of this task. Section 3 describes the technique of using a hierarchical structure of abstract machines requiring only the lowest level of machine to be written in the real computer's assembly language.

1.3 Aim of the Project

The aim of the project is to produce a fast, efficient context editor which can be easily adapted to run on any computer. The main work is in the design, coding and implementation of the context editor.

The following sections describe the design, and implementation of the context editor (TED). Section 6 contains the conclusions, extensions and improvements to the editor and its portability. For completeness included in the appendix is a users manual and flow charts.

2 COMMAND LANGUAGE DESCRIPTION

The user views the editor as a machine on which he can solve his edit problems. To do this he is aware of certain functions of the editor. They can be described as:

a I/O streams

b an edit work area

c a command language.

2.1 Input/Output Streams

The editor, as seen by the user, has four input/output streams. The file to be edited is read from the input stream and the final edited version of the file is written to the output stream. These two streams can be attached to any permanent storage device eg mag tapes disc or cards.

To communicate with the user the editor requires two streams - a command stream and a print stream. The user inputs his commands via the command stream which, in batch mode, is usually attached to a card reader and in on-line mode to a teletype or visual display unit. The print stream is used for error messages and verification listings. In on-line mode it is attached to the same device as the command stream and in batch mode it is normally attached to a line printer.

2.2 Buffers

The editor reads lines of text from the input file and holds them in the input line buffer (ILB) to be processed. As a line is edited it is written to the output line buffer (OLB). When the editing on that line is completed it is written to the output stream. The input line buffer provides access to the current line and the output line buffer is temporary storage for the edited line. Each of these buffers has a pointer to keep track of character editing within the line. The movement of these pointers is explained in the description of each command.

The editor can also be used to store information either from the input stream or the command stream.

2.3 The Command Language

The command language is the interface between the user and the editor. Commands consist of simple mnemonics followed by information required during execution of the command. It seems obvious to say that the command language should be kept simple while retaining the power of the editor. In order to achieve sophisticated editing it is necessary to establish the facilities of the editor and to construct the command language around them keeping the format as uniform as possible.

The following is a description of the command language and the action taken by the editor on executing the commands. There are basically two groups of commands:

a commands operating on lines or characters and

b special purpose commands for extra facilities in the editor eg looping, exchanging etc.

2.3.1 Line and character commands

The group of commands is used when individual operations are required on the input stream. Line commands read directly from the input stream and write directly to the output stream. Character commands operate within the line buffers. To distinguish between a line command and a character command a special character delimiter is used after the command word. The default delimiters are, a slash indicating a line command and a dot a character command. If no delimiter is present a character directive is assumed.

Each command is input on a new line and a line on the input stream is determined by the size of the input line buffer.

In the language description the following mnemonics are used:

n = any positive integer (if omitted 1 is assumed)

string = any string of characters that can be contained in the command line. Continuation is not permitted

del = a line or character delimeter

ILB = input line buffer

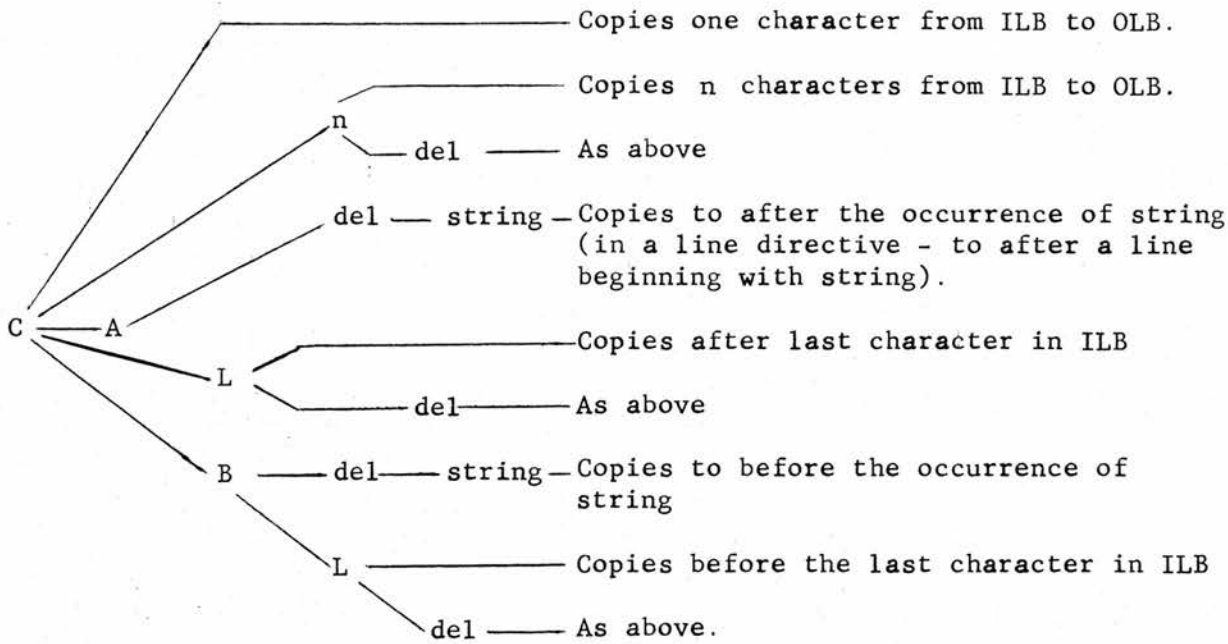
OLB = output line buffer.

2.3.1.1 Facility: copy

Function: The copy facility causes the pointers to be moved to a new position as specified by the information on the command stream. In processing a copy command lines may be copied from the input stream to the output stream, or characters copied from the input line buffer to the output line buffer. A specified number of lines or characters can be copied, a specified string of characters can end the copy (either before or after the string) or the last line in the file or last character in the line can terminate the execution of the copy.

Format: The mnemonic for copy is C. The information following the mnemonic C is interpreted by the command analyser and the appropriate copy action taken.

The following tree shows the build up of the copy command for character delimiters and the meaning of each. The character delimiters can be replaced by line delimiters. In this case lines are copied from the input stream to the output stream.



The find command, mnemonic F, is a special type of copy command. F must only be followed by a line delimiter and a string.

The mnemonic N is another special case of the copy command

Action: The action of pointers and buffers for each command is shown below in a series of examples. If during any copy command an end-of-file condition arises on the input stream copying stops and an error is output.

The mnemonics used in the example are as follows:

IPB Input stream pointer before the action has occurred, pointing to the current line

IPA Input stream pointer after the action, pointing to the new current line.

OPB Output stream pointer before action.

OPA Output stream pointer after action.

IPPB Input line buffer pointer before action.

IPPA Input line buffer pointer after action.

OPPB Output line buffer pointer before action

OPPA Output line buffer pointer after action.

Examples of line commands:

a) copy 5 lines

command stream - C5/

Input stream	Output stream	Comments
IPB⟶THIS LINE AND THE FOLLOWING LINES ARE USED AS EXAMPLES FOR COPY INSTRUCTIONS IPA⟶TO THE EDITOR	OPB⟶ THIS LINE AND THE FOLLOWING LINES ARE USED AS EXAMPLES FOR COPY INSTRUCTIONS OPA⟶	lines are copied directly from the input stream to the output stream

If the current line held in the input line buffer has been edited it is written to the output stream before the copy action occurs. Otherwise the current line is output as one of the lines of the copy instruction.

b) Copy after line beginning "LINES"

command stream - CA/LINE$

Input stream	Output stream
IPB⟶THE LINE AND THE FOLLOWING LINES ARE USED IPA⟶AS EXAMPLES FOR	OPB⟶ THE LINE AND THE FOLLOWING LINES ARE USED OPA⟶

Copy before line beginning "LINES" makes the line beginning "LINES" the current line and the command stream contains CB/LINES.

c) Copy after last line

command stream - CAL/

This instruction causes the editor to read from the input stream and write to the output until and end of file (EOF) conditions exists. There is no current line after this command.

Copy before last line makes the last line of the file the current line.

Examples of character commands

a) Copy 7 characters

Command stream - C7 or C7.

```
        ILB                                OLB

THIS  IS THE  CURRENT          THIS LINE  IS THE
   ↑      ↑                          ↑       ↑
 IPPB    OPPA                      OPPB    OPPA
```

b) Copy after ENT

Command stream - CA.ENT

```
        ILB                                OLB

THIS IS THE CURRENT            THIS LINE IS THE CURRENT
          ↑      ↑                          ↑       ↑
        IPPB    IPPA                      OPPB    OPPA
```

For copy before a string the pointers in both buffers are placed before the occurrence of the string. If the string does not occur in the current line it is written to the output stream. The next line becomes the current line and the search for the string continues.

For the find command the action is the same as above except that the input line buffer pointer is placed at the start of the line containing the specified string.

N causes the current line to be written to the output stream and a new current line read in regardless of the positions of the input line buffer pointer.

c) Copy after last character in line

Command stream - CAL.

```
       ILB                              OLB

A NEW CURRENT LINE          A NEW SECOND CURRENT LINE
    ↑       ↑                           ↑       ↑
  IPPB     IPPA                       OPPB    OPPA
```

d)

Copy before last character places the pointers before the last character on the current line.

2.3.1.2 Facility: delete

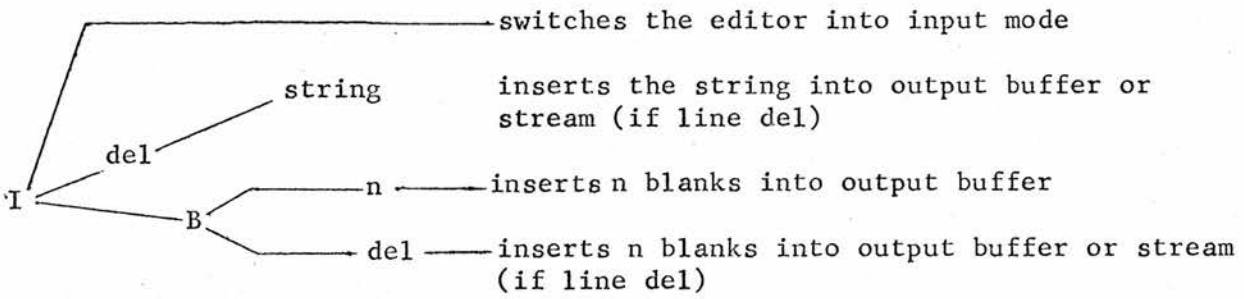
For every copy facility there is a corresponding delete facility.

The initial mnemonic for this facility is D. The action on the input stream or input line buffer is the same as for the corresponding copy command. The pointers to the output stream or output line buffer remains unchanged.

2.3.1.3 Facility: insert

Function: This facility inserts character strings into either the output stream or the output buffer. Another use is to copy lines from the command stream to the output stream, thus inserting new lines into the edit file. Blank characters and blank lines can also be inserted using a form of this command.

Format: The mnemonic for insert is I. The tree diagram shows the different forms of the insert command.



Action: If the command stream contains I and nothing else the editor goes into input mode. All further input on the command stream is written directly to the output stream until the occurrence of the command FIN in the command stream. FIN switches the editor back to edit mode.

Examples of line inserts

a) Insert the line - THIS IS NEW

Command stream - I/THIS IS NEW

Output stream	Comments
OPB——→ THIS IS NEW OPA——→ -	No action on input stream

If the current line has been edited it is output before the new insert line and a new current line read in from the input stream otherwise the current line remains unchanged.

b) Insert 4 blank lines

Command stream - IB4/

Output stream

OPPB——→LAST OUTPUT LINE
OPPA——→ } 4 blank lines

Examples of character inserts

a) Insert the string NEW

Command stream - I.NEW

ILB

```
THIS IS A LINE
        ↑
      IPPA
    & IPPB
```

OLB

```
THIS IS A NEW↑
        ↑   ↑
     OPPB  OPPA
```

b) Insert 5 blank characters

Command stream - IB5.

ILB

```
A BIT OF FUN
       ↑
     IPPB
   & IPPA
```

OLB 5 blanks

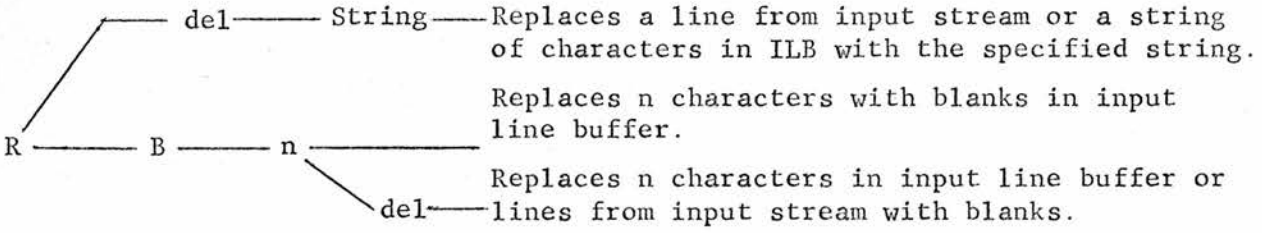
```
A BIT OF↑    ↑
        ↑    ↑
     OPPB   OPPA
```

When inserting characters into the output line buffer a check is made on the remaining characters in the input line buffer. If they cannot be read into the output buffer without overflowing, a warning message is output and the trailing characters truncated.

2.3.1.4 Facility: replace

Function: This command replaces a character string or the current line by a specified string. It can also be used to replace lines or characters with blanks. Replace is equivalent to a delete followed by an insert.

Format: The format of the replace instruction is similar to the Insert instruction as the tree diagram shows. Its mnemonic is R.

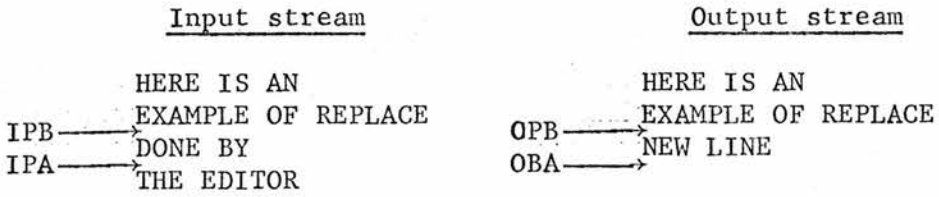


Action: A flag in the editor is set to indicate a replace command and the code for delete and insert is executed.

Examples of replacing lines

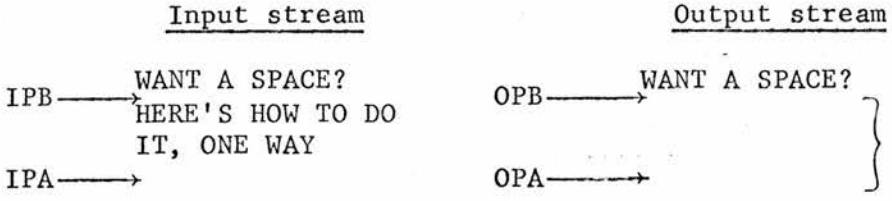
a) Replace the current line with NEW LINE

Command stream - R/NEW LINE



b) Replace 2 lines with blanks

Command stream - RB2/



Examples of replacing characters

a) Replace a string in the current line by APPLE PIE

Command stream - R.APPLE PIE

ILB

```
DO HAVE PEACH PURIE
     IPPB     OPPA
```

OLB

```
DO HAVE APPLE PIE
     OPPB      OPPA
```

b) Replace the next 5 characters with blanks

Command stream - RB5.

ILB

```
DO HAVE PEACH PURIE
     IPPB  IPPA
```

OLB

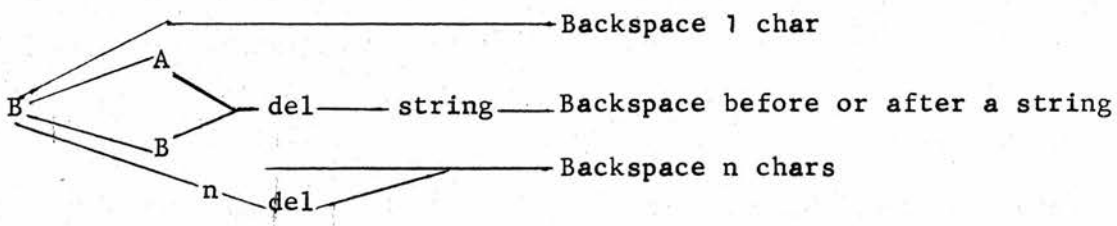
```
          5 blanks
DO HAVE  ⌒
     OPPB      OPPA
```

2.3.1.5 Facility: backspace

Function: This facility can only be used on the current line. It can place the pointers back a specified number of characters or back to a given string of characters within the input line buffer.

Backspacing of lines with the editor would cause a great amount of input/output operations. It can be done by issuing a START/Command (q.v.) and then positioning the pointer to the required line in the input stream. The editor is most efficient when editing is done sequentially through the file.

Format: The mnemonic for backspace is B, and the tree shows the format of meaning of the different backspace commands.



Action: The following examples show the action on the input and output line buffers and the position of the pointers on the execution of a backspace command.

Examples of backspace

a) Backspace 5 characters

Command stream - B5.

	ILB	OLB
Before execution	THIS LINE IS TO BE IPPB	THIS NEW OPPB
After execution	THIS NEW LINE IS TO BE IPPA	THIS OPPA

b) Backspace to before the string IS

Command stream - BB.IS

	ILB	OLB
Before execution	THIS LINE IS DAFT IPPB	THIS NEW LINE OPPB
After execution	THIS NEW LINE IS DAFT IPPA	TH OPPA

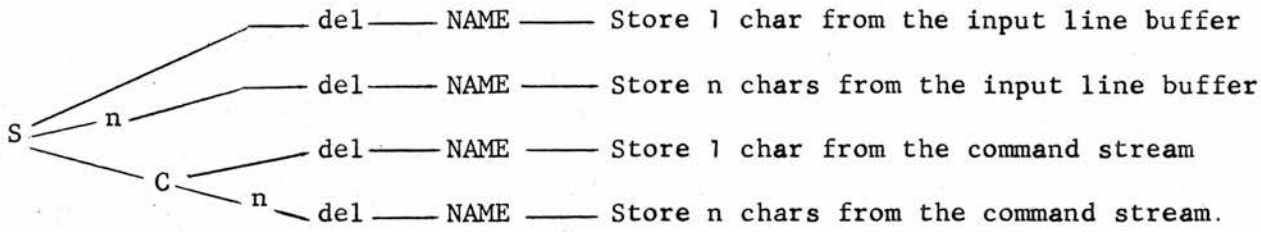
Backspacing is done on the edited line.

If the string specified does not exist on the output line buffer an error message is output and the pointers and buffers remain unchanged.

2.3.1.6 Facility: storing

Function: Strings of characters or lines from either the command stream or the input stream can be stored away in a buffer to be referrenced at a later stage in the edit run.

Format: The mnemonic for the store command is S. The name of the store area in the store command (NAME) must not contain more than 4 letters. The tree shows the character commands of the store instruction. For each character command there is a corresponding line command and lines are read and stored from the input stream instead of input line buffer.



Action: When storing a string of characters the dictionary of store names is scanned. If a duplicate name is found and the new string is smaller or equal to the old one a warning diagnostic is output and the new string is stored under the duplicate name. If the new string is greater than the old string a fatal error message is output.

Examples of character storing

a) Storing 5 chars from the input line buffer

Command stream - S5.A1

ILB

```
WE WISH TO STORE SOME RUBBISH
         IPPB
```

Store Area

```
.........↑STORE↑
      Pointer  Pointer
      before   after
       (X)
```

Dictionary

```
                    name size pointer in store area
..... ↑A1, 5, X↑
      Pointer    Pointer
      before     after
```

The input line buffer pointer is moved to point after the stored string

b) Storing 11 chars from the command stream

Command stream - SC11.A2 - line 1

THIS LOT OF - line 2

Store area

```
..... STORE THIS LOT OF↑
          Pointer     Pointer
          before      after
          (Y)
```

Dictionary

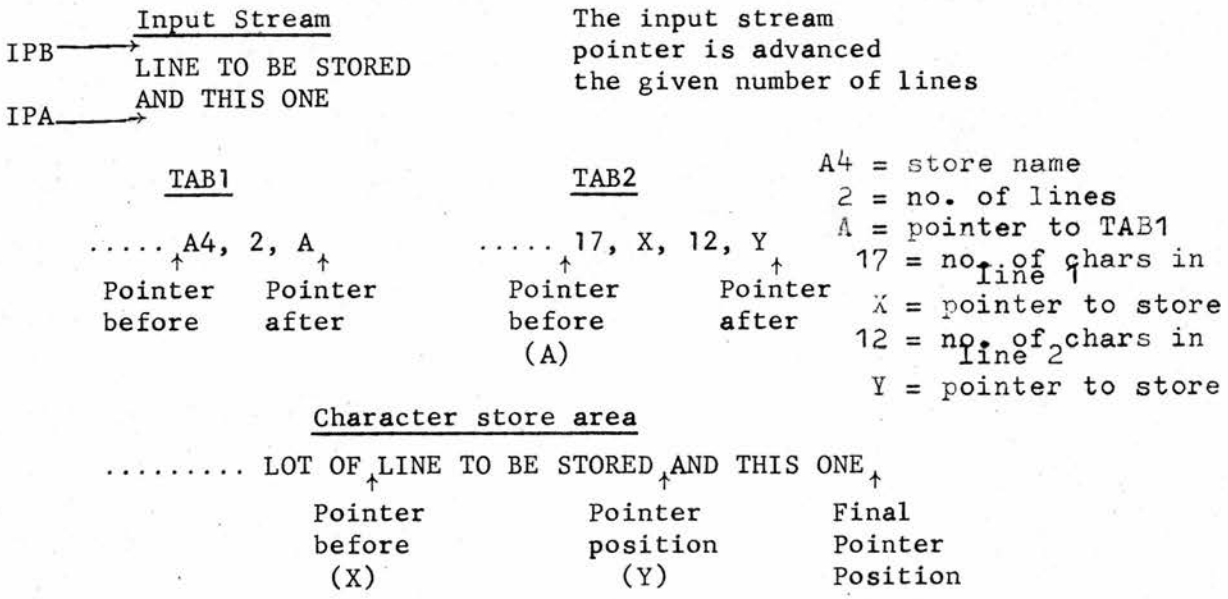
```
                          name size pointer in store area
..... A1, 5, X ↑A2, 11, Y ↑
      Pointer     Pointer
      before      after
```

When storing lines from the input stream or the command stream two tables are kept and the actual data is stored in the character store area. The first table contains the names of the line store areas, the number of lines and a pointer to the second table which contains, for each line, the number of characters in the line and the position of the first character in the character store area.

Examples of storing lines

a) Store 2 lines from the input stream

Command stream - S2/A4



b) Store 2 lines from the command stream

Command stream SC2/A3 - line 1

CA/XYZ - line 2

D5. - line 3.

These lines are stored in the same way as the ones read from the input stream. Line store names cannot be duplicated; an error message is output and no action taken if a duplicate name is found.

2.3.1.7 Facility: loading

Function: Loading into either the command stream or the output stream strings of characters or lines which have been previously stored with the commands described in 2.3.1.6.

Format: The format of the load instruction is L NAME, where NAME identifies an area of stored characters or lines.

Action: If a load command is used in place of a string then the character store directory is searched for the name and the required string replaces the load command in the command stream. If the name is not found in the character store directory a warning diagnostic is output and the command stream remains unchanged.

Examples of loading strings of characters

Command stream	before	CA/L A1
	after	CA/STORE
	before	CA/L XY
	after	CA/L XY error output.

Only the character store directory is scanned when a load command is issued in place of a string. The line store directory is scanned when the load instruction is in the command stream in place of a command. The lines from the load area are inserted into the command stream at the point where the load instruction occurs. As in the character store if the name is not found an error message is output and the load instruction ignored unless the editor is in input mode. In this case the instruction is copied to the output stream.

Examples of loading from the line store area

a) Command stream

```
I
L A4
AGAIN A LINE
FIN
```

```
        after   I
                LINE TO BE STORED
                AND THIS ONE
                AGAIN A LINE
                FIN
```

b) Command stream L A3

```
        after   CA/XYZ      These commands are
                D5.         now executed in order.
```

2.3.1.7 Facility: start

Function: This command moves the pointer to the start of the input stream or to the start of the input line buffer retaining all previous editing.

Format: This command has no single letter mnemonic. It consists of the word START followed by a line or character delimiter.

Action: When START is followed by a line delimiter the rest of the input stream is copied to the output stream which now becomes the new input stream.

START followed by a character delimiter causes the rest of the input line buffer to be copied to the output line buffer and the buffers are swapped. The input buffer is now the output buffer and vice versa. The output line buffer is set empty and the buffer pointers set to the start of the defined field in the new input line buffer.

2.3.2 The special purpose commands

There are several special features incorporated into the editor and the command language to operate them is described below. The mnemonics for these commands are different from the line and character commands. They tend to be full words and are not in any standard format. Only the command word is scanned by the command analyser. The rest of the information in the command is checked during execution of the command.

2.3.2.1 Facility: stop

Function: To terminate the editing run either normally or abnormally.

Format: EXIT is the normal end and CANCEL the abnormal halt.

Action: The abnormal stop is used only in an on-line environment. It exits from the editor scrapping all new files. The input stream remains unchanged.

The normal exit from an edit run copies the rest of the input stream to the output stream, closes and rewinds the files, giving a printed copy of the new file, if required.

2.3.2.2 Facility: loop

Function: This facility allows a group of commands to be executed a specified number of times or until an EOF condition exists.

Format: The sequence of edit commands is enclosed by the words LOOP and LOOPEND. LOOP can be followed by an integer specifying the number of times the loop has to be executed. If omitted, or zero, the loop is executed until an EOF condition exists on the input stream.

Action: All the statements from the first LOOP to the final LOOPEND are processed through the command analyser and stored in an area of core (LOOPCOM) in their coded state. Nested loops are permitted. Each time a LOOP statement is encountered the LOOP counter is increased by one and decreased on the occurrence of a LOOPEND. When it has returned to zero the loop is ready for execution.

On executing the loop commands are read sequentially from LOOPCOM and executed. Each occurrence of the statement LOOP causes the address of the next command to be placed on a stack together with the number of times the loop has to be executed, and on encountering a LOOPEND the top value of

the stack is interogated. If the number is non-zero it is decreased by 1 and returned to the stack and the address of the next command retrieved from the stack. If the number on top of the stack is zero, it is discarded and the next value investigated. The address of the next command to be executed is found on the stack. When the stack is empty the loop is exhausted and further editing commands are read from the command stream.

2.3.2.3 Facility: exchange

Function: This powerful feature allows one string of characters to be exchanged for another throughout the file while other editing is in progress.

Format: The command for switching on the exchange is EX followed by two strings delimited by any non-alphameric symbol.
eg EX*FIRST*SECOND*. The strings can be of varying length. To switch off the exchange the mnemonic NEX followed by a delimiter and the first string is used eg NEX/FIRST stops the action initiated by the previous example.

Action: On the occurrence of an exchange command the two strings are stored in an area in core and a flag set to indicate exchange is in progress. Each line input to the input line buffer from the input stream is scanned for the occurrence of the first string and it is replaced with the second before any further editing is done. There can be up to six exchanges simultaneously in an edit run but it must be pointed out that the use of exchange slows the editor down considerably as each input line has to be scanned once for every exchange command.

Each time a NEX is issued the exchange area is checked. If there are no more "live" exchanges in progress the exchange flag is unset and lines from the input stream are no longer scanned.

2.3.2.4 Facility: sequence

Function: This feature sequentially numbers the input file in the last eight positions of the defined field. These eight positions are split into two fields of 4 characters each.

Format: The sequencing command is of the form SEQ No1, INC1, No2, INC2.

where No1 - is the starting number of the first 4 positions in the field

INC1 - is the increment for this field

No2 - is the starting number of the second field

INC2 - is the increment of the second field.

If either of the increments is zero the corresponding starting number can be a string of up to 4 alphameric characters right justified within the field. If NO1 and INC1 are both zero the 8 character field is treated as one field with starting number No2 and increment INC2.

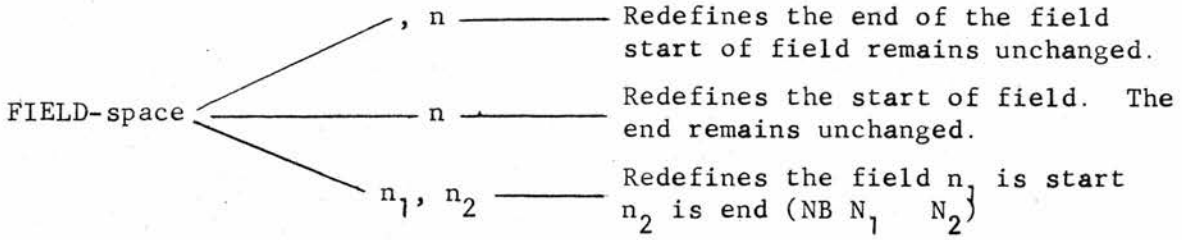
Action: All further input from the input stream is sequenced according to the information on the command stream.

If the defined field is less than 8 characters long the sequencing is truncated and a warning dianostic written out. The sequencing command is executed to completion before the command stream is read again, unlike the exchange facility.

2.3.2.5 Facility: field defining

Function: The field of the input line buffer to be edited can be defined using this instruction. By default it is from 1 to 80 but, to save time in scanning the complete field for specified strings it can be defined smaller. The field size can be changed any time during an edit run.

Format: The format of this command is shown below in the tree diagram.



Action: This facility sets pointers in the input line buffer to mark the beginning (SFP) and end (EFP) of the field which is to be edited. When a field defining command is given defining EFP only, the current value of SFP must be smaller than the new value of EFP. If not the command is ignored and the previous values of SFP and EFP are unchanged. Similarly a new value of SFP must be less than the current value of EFP, and if both are programmed the new SFP must be less than the new EFP.

2.3.2.6 Facility: tab

Function: Enables spaces to be inserted into an input string by defining a tab character and various tab stops.

Format: The mnemonic for this feature is T followed by a space, the tab character and then a string of tab stops in ascending order separated by commas. T followed by two question marks switches off the tab facility.

Action: This facility causes spaces to be inserted into the command stream within a defined string. Tab can be changed by a further tab command. Tab stops not being altered can be missed out in the new tab statement:

eg T $, 10, 20, 30, 40, 50, 60 sets up tabs in columns 10-60 in steps of 10

the statement T $, 15,,, 45,, 65 sets up tabs in columns 15, 20, 30, 45, 50, 65

Example in use of tabs

```
Command stream    T $, 10, 20, 30, 40      $ is the tab character.
                  I                        It can be any of the following:-
                  1$2$3$4$5                   * " / @ £ & ' ( )? + -
                  $$3$5
                  123$4
                  FIN
                  T??
```

```
                Col 1       Col 10      Col 20      Col 30      Col 40

is equivalent to I
                1           2           3           4           5
                                        3           5

                123         4
                FIN
```

After tab has been switched off by a T ?? command tab characters within a string are treated as part of the string.

2.3.2.7 Facility: delimiter changing

Function: This feature can be used to change the line delimiter (/) or character delimiter (.) to any other non-alphameric character.

Format: The line delimiter is changed by the command LINE followed immediately by the required character eg LINE* makes the new line delimiter an *. The mnemonic CHAR does the same for a character delimiter change.

Action: The new delimiter is set up in place of the old one and remains as the delimiter until another LINE or CHAR command is programmed or until the end of the edit run.

2.3.2.8 Facility: repeat last command

Function: Causes the previous command to be executed again.

Format: REP is the mnemonic for this command.

Action: The details of the previous command are retrieved and immediately re-executed without initialising any variable.

2.3.2.9 Facility: display

Function: This facility can only be used in an on-line mode with a visual display unit. It enables the user to manually alter 12 lines of the input stream on the screen and write them onto the output stream using a single command.

Format: The single command is the word DISPLAY.

Action: DISPLAY causes 12 lines from the input stream to be read and written to the display unit. These lines can then be altered in any way by the user and re-entered. This time they are read from the display unit and written out on to the output stream. If there are less than 12 lines in the input stream before an EOF marker occurs the number of lines output to the unit is counted and this number of lines read in again. In effect, DISPLAY deletes 12 lines from the input stream and inserts 12 new lines to the output stream.

3 THE ABSTRACT EDIT MACHINE

3.1 Abstract Machines

The technique employed to realise the editor on a real computer is that of a hierarchy of abstract machines[1]. At the top of the system of abstract machines is the edit machine designed solely to solve the problem of context editing. The basic operations of this machine are expressed in a high level language which is mapped on to the next level of abstract machine and so on down the line of abstract machines. The lowest level of abstract machine is the base for mapping on to a real computer. At this level the abstract machine is computer dependent, but since all higher level abstract machines are not dependent on the real computer only minimum effort is required to finally realise the top level of abstract machine on various real computers.

Efficiency of the system can be improved by mapping a higher level of abstract machine directly on to the real computer. This reduces the portability of the system. Therefore the easier it is to map the lowest level of abstract machine the more portable the software becomes. As efficiency is the primary consideration in this project there are only two levels of abstract machines in the hierarchy. The abstract edit machine functions are defined in a high level language in terms of the next level of abstract machine. This level is mapped directly on to the real computer. Thus the editor is not particularly portable but is very efficient.

3.2 Mapping Method

Macro processing is the method used to map one level of abstract machine on to the next. This technique allows the user to define the functions of the top level of abstract machine in a high level language. The final realisation of this machine on the real computer is in the machine language of this computer. Thus the user can tailor the assembly language

of the real machine to meet the requirements of the abstract machine at a higher level.

A macro is written for each basic operation of the highest level of abstract machine - in this case, the editor. When these macros are processed the final result is the assembly language instructions for the real computer to execute the required operations.

The macro processor used in this project was STAGE 2[4]. This is a powerful general purpose macro processor which is itself highly portable. The instructions for the editor form the top level of abstract machine and the expansions of each of these instructions in assembly code is the lowest level of abstract machine. Hence the realisation of the abstract edit machine on the real computer.

A specific advantage of using this method is that changes to the definitions relating the abstract machine to the real computer are very easily implemented. Thus it is easy to change the lowest level of abstract machine to suit various computers. The greater number of levels in the abstract machine hierarchy the more complicated the macros become. STAGE 2 is a very slow processor and requires a large core store, but the ease of changing the lowest level of the macro to suit various computers far outweighs this disadvantage.

3.3 The Design of the Edit Machine

The initial stage of the design of the abstract edit machine is to determine the requirements of the machine. To do this it is necessary to establish exactly what the functions of the machine are and how these functions should operate.

The initial requirement for any machine is a starting device. On the edit machine this initialises the editor and makes it ready to accept commands.

Similarly the stopping device switches everything off and closes the machine down.

The edit machine needs a memory to keep track of pointer positions, variables and save areas for saving data to be retrieved at a later stage in the editing. In addition to the memory the editor requires two buffers. One of these is used to hold the current input line of the file being edited and the other holds the updated output line. Pointers are used to keep track of the positions of characters in these buffers and a pointer moving mechanism is available on the edit machine. The editor requires a series of switches to indicate the existence of certain conditions eg is print on or off? is exchange in operation? etc.

A branching facility is another feature of the edit machine. This feature enables the machine to cope with conditional and unconditional branches. The machine determines the next operations if certain conditions arise eg it must be able to branch back and read in another command on the completion of the previous command or if the previous command was an exit command it must branch to the closing condition.

Another feature of the machine is a comparing facility. This enables it to compare two strings of characters and take appropriate action on whether a match is found or not. Associated with the compare facility is the searching mechanism. This is used to search for characters or numbers in the command stream and store them in memory for future use by the editor.

Input/output channels are an obvious requirement for any machine. The editor uses two input streams, one output stream and a general input/output work area. One of the input channels is for inputing commands to the editor. This is either a card reader or similar device for batch editing or a visual display unit for on-line editing. The other input channel is

for the edit file and is usually disc or magnetic tape. The output file is for the new edited file. The work file is a fast access file used to hold the partially edited file during the edit run.

The final requirement of the abstract edit machine is a set of instructions to operate it. Each basic editing operation is activated by a specific instruction and by executing all instructions the abstract edit machine is realised on the real computer.

3.4 Macro Descriptions

One macro is written for each instruction of the abstract machine and when all these macros are expanded the editor can be run on the real computer.

The macros used in the edit machine can be discussed in three catagories.

a Utility macros by the macro processor.

b Special purpose macros for the edit machine, and

c Instruction macros for the edit machine.

3.4.1 Utility macros for the macro processor

There are four macros used only by the macro processor in the expansion of the instruction macros.

The instruction macros are fairly general and if certain parameters are input to the macro which are non-standard then it may be necessary to skip some lines within the macro body. In order to do this macros for setting the skip counter are included into the program. One macro sets the skip counter to a given value if two parameters are equal, another sets it if they are not equal and the third sets it unconditionally.

The fourth utility macro, when called, indicates to the macro processor to halt all processing. It always comes at the end of a macro expansion run

Another utility macro is used to copy coding direct from the input to the macro processor without expanding it. It is necessary to have such a macro as portions of the editor have not been divided up into individual macros eg the command analyser.

3.4.2 Special purpose macros for the edit machine

As certain lines of coding are only used once during an edit run, they are included in several macros and only called up once during the macro expansion run. These are special purpose macros for the edit machine and each one is described in more detail.

a) START

This macro is the first one to be expanded. It causes the edit machine to be set ready to receive editing commands from the input stream. The first line of the file to be edited is read in and set up and the current input line.

Different computers require different start procedures. For the IBM 360/44 the start routine sets up external addresses, base registers etc.

b) STOP

This switches off the editor and returns the computer to the state it was in before the edit run began.

c) INPUT

This macro is required for every input operation. An input instruction (READ IN TO A) causes a branch to this area of code and it sets up the stream from which the data is to be read and the area in memory where the data is to be inserted. It must check for an end-of-file condition and set the appropriate flag. In the case of the IBM 360/44 implementation this macro causes a branch to a fortran subroutine to do the actual reading operation.

d) OUTPUT

Similarly this macro is used for all output operations. It has an extra flag to indicate whether the output has to be output to the print stream as well as written to the output stream. Once the output line buffer has been output this macro sets it to blank for the next editing operation. As in the case of INPUT a fortran routine is called in the IBM 360/44 implementation.

e) POINTERS

When a new line from the input stream is read in this macro is used to set the input line buffer and output line buffer pointers to their initial starting value. It narrows down the input line buffer to the required edit field. It also sets a pointer to point at the last significant character in the input line buffer.

f) MOVES

A move instruction causes the program to branch to this macro. It causes characters to be moved from one core location to another. The address of the core locations and the number of characters to be moved are set up by the move instruction before branching to this macro.

g) FETCH NAME

This macro fetches a name from the command area and places it in an area in memory. It is used when the command stream contains a load instruction.

h) FETCH NAME FROM STRING

This works in the same way as the previous macro except that it fetches the name of the load area from the memory instead of from the command stream. It is used when the load instruction is part of the input string of a command.

i) NUMBER

This macro is used when there are numbers to be decoded on the command stream. The command to find a number (FIND NUM) branches to this area of code and decodes the number on the command stream and stores it as a binary integer in memory for future use. If there is not an integer on the command stream an error is output except in the case of a sequencing command where either a number or a string of alphameric characters is expected.

j) NUM NOT FOUND

In a sequencing command when, instead of an integer, a character string is present in the command stream, this macro deciphers the string and stores it in core. It checks the number of characters in the string and outputs an error if it exceeds the maximum allowed, or space fills the string if there are less than the maximum number of characters.

k) SWAP UNITS

After a start command has been issued it is necessary to swap around the file so that the present input file becomes the work file and the present work file becomes the new input file, except after the first start command when the work file becomes the new input file. For the IBM 360/44 implementation a call to a fortran subroutine is activated to perform the required operations.

l) TABSTORE

If the tab character has been set, each string input from the command stream must be scanned for the occurrence of the tab character and the string space filled accordingly. This macro is called every time a string of characters has been input, to check for tabs and expand the strings.

m) RESET

When the command stream encounters an exit instruction this macro resets the

the files. It outputs the final edited file, scrubs the work file and closes all files. In the case of the IBM 360/44 implementation this macro calls a fortran subroutine to sort out the files.

n) SPACE TEST

This special purpose macro is used when copying before a line beginning with a specific string. It tests if the string contains significant spaces. If it does the compare test is done only at the start of the input line buffer. If no significant spaces are present on the input string significant spaces on the input line buffer are ignored.

o) FIND LAST CHAR

To determine whether an insert can be placed within the defined field limits it is necessary to find the length of the character string in the input buffer. This macro scans the input string from the end, looking for the first non-blank character, thus finding the length of the input string.

p) CONSTANTS

This macro sets up all the storage and constants required by the editor. It defines the buffers, save areas, pointers, initial read/write unit numbers, flags and tables.

q) DUMP

This is a debugging macro and is used only by the systems programmer It enables the memory of the edit machine to be dumped out at specific points during an edit run.

r) DUMPCHECK

This macro is used in conjunction with the DUMP macro described above It checks if a dump is required and if so causes a branch to the macro DUMP

s) DISPLAY

If visual display units are available to the editor this macro causes a

screenful of lines from the input stream to be displayed, manually edited and then read back in again from the display unit to the output stream. It firstly deletes the displayed lines from the input stream and then inserts the new lines from the display unit into the output stream. In the IBM/360 44 implementation this is done by a call to a fortran subroutine.

3.4.3 Instruction macros for the edit machine

Each basic instruction of the edit machine is defined by a macro. Some of these instructions use parameters to be input to the macro (eg in the addition macro A = B + C, A, B and C are the parameters). There are ten groups of instruction macro, each of which is described in fuller detail below.

a) Arithmetic macros

There are three arithmetic instruction macros. The first causes one area of core to be set equal to another area of core, a constant or a literal. The second is an addition macro and the third a subtraction macro.

i A = B (Parameters A and B). The contents of core location B are stored in location A. B can also be a constant or a literal.

ii A = B + C (parameters A, B and C). B and C are added together and placed in core location A. If any of the parameters is REG then the contents of register 1 are used instead of the variable.

iii A = B - C (parameters A, B and C). This subtracts C from B and puts the result in location A. Again, as above, the contents of register 1 are used if the parameter is REG.

b) Character moving macros

There are two macros for moving a string of characters from one core location to another. The first moves a specific number of characters from core location to another core location and contains a flag to indicate whether

the pointers to the specific core locations have to be retained. Its form is MOVE X CHARS FROM A TO B, FLAG. The parameters are X, A, B and FLAG This macro sets up the parameters X, A and B in preparation to branching to the special purpose macro MOVES which does the actual operation of moving the characters. On return from MOVES the flag is tested and if the pointers have to be kept it saves their values depending on A and B eg if A is the input line buffer the pointer value must be set to the input line buffer pointer etc.

The second move macro moves blanks into a specified core location. Its form is MOVE BLANKS to A, where A is the parameter.

c) Register instruction macros

The general registers are used as indices to the various store areas required by the editor. It is, therefore, necessary to have a set of macros to manipulate these registers.

i REG A = B (parameters A and B). This macro sets general register A to the contents of core location B or the value of B if B is a literal.

ii B = REG A (parameters B and A). The contents of general register A are placed in core location B.

iii GREG A + B. Adds the contents of B, or if B is a literal, the value of B to general register A.

iv GREG A - B. Subtracts the value of B from general register A.

v ADDRESS REG A = B. The address of core location B is placed in general register A.

d) Branching and associated macros

There is one unconditional branching macro and five conditional branching macros.

The unconditional macro has the form JUMP TO A. Where A is a label set up by the associated label marking macro LABEL A.

The conditional branching macros are as follows:

i If A rel B JUMP TO C (parameters are A, rel, B and C). If the relation (rel) between A and B is true then a branch to C is effected. C has been set up by the label macro described above.

ii If A JUMP TO B (parameters A and B). If the variable A has been set to the value true then a jump to label B is made, otherwise processing continues sequentially.

iii If NOT A JUMP TO B (parameters A and B). This is the opposite of the above macro. A branch to B is made if A is set to false.

iv If LOAD A ELSE B (parameters A and B). This macro tests the input string and if it is a load instruction branches to label A otherwise it branches to label B. If A has the value CONTINUE then processing continues with the next instruction instead of branching.

v BRANCH ON REG A TO B (parameters A and B). This is a conditional looping macro. It tests register A and if it is non-zero branches to label B at the same time reducing register A by one. Register A must have been previously set to the loop count.

e) Comparison macro

In order to compare two strings of characters in core and take action if a match is found, the macro COMPARE A WITH B IF FOUND C is used. This compares a string of characters in A with a buffer B and causes a branch to label C

if a match is found. The whole of buffer B is scanned for the occurrence of the string except when searching for a line beginning with a string of characters. In this case if no match is found on the first try the program continues sequentially.

f) Finding macros

There is a macro for retrieving integers from the command stream. It is called by the instruction FIND NUM. and causes the program to branch to the macro NUMBER to decode the integer from input format to binary.

The other finding macro retrieves a specific character from the command stream. It is used in the case of the tab instruction to check for the tab character and is called by FIND CHAR.

g) Input/output macros

The macro for reading data into a specified area from a specified unit is called by READ IN TO A, where A is the required input area. Before calling this macro it is necessary to establish in a variable in memory the unit number of the stream where the data is held. This macro calls up the special purpose macro INPUT.

Similarly to output a record in memory the macro WRITE OUT A, where A is the position in memory of the record to be output, is used. The unit number of the stream to which the data is to be written should have been previously established. This macro branches to the special purpose macro OUTPUT.

h) Checking for tabs.

When a string has been input from the command stream it must be checked for the occurrence of the tab character and expanded accordingly. The macro TAB CHECK causes a branch to the special purpose macro TABSTORE.

i) Setting initial pointers.

The macro SET POINTERS is used when a new record has been input from the edit file stream. It calls the special purpose macro POINTERS to set up the initial positions of the pointers to the input and output line buffers and also the pointer pointing to the last significant character in the line.

j) Swapping core areas.

Since buffers are used by the editor it is necessary to be able to swap them around and the macro SWAP A AND B (A and B are parameters) causes buffer A to become buffer B and vice versa.

4 THE COMMAND ANALYSER AND ERROR HANDLING ROUTINE

The command analyser and error handling routine are two separate entities in the editor. They are not mapped from an abstract machine but are substituted directly into the editor.

4.1 The Command Analyser

The command analyser accepts commands from the command stream, analyses them and if the commands are synactically correct causes a branch to the appropriate executing routines in the editor.

The analyser can be broken down into six logical stages. If an error occurs at any one stage control is passed to the error handler and analysing is terminated. The next command is then fetched from the command stream.

Fig. 1 is a general flow chart of the analyser, indicating the six stages described more fully later.

The command is input to the command string from the command stream. The string is scanned to find the first non-alphabetic character. The alphabetic characters are hashed to produce a unique number for each command. This number indexes the hash table which contains indexes to the command table (COMTAB). Using COMTAB the syntax of the command is checked and if correct an integer is allocated to the specific command. This integer is the command number.

Each command word is terminated by a specific delimeter. There is a set of tables which matches the allocated command number with the specified delimeter. Only those commands containing the correct delimeter are passed to the next stage.

In the final stage of analysing any further information that is required is extracted from the command string and stored for use during the command

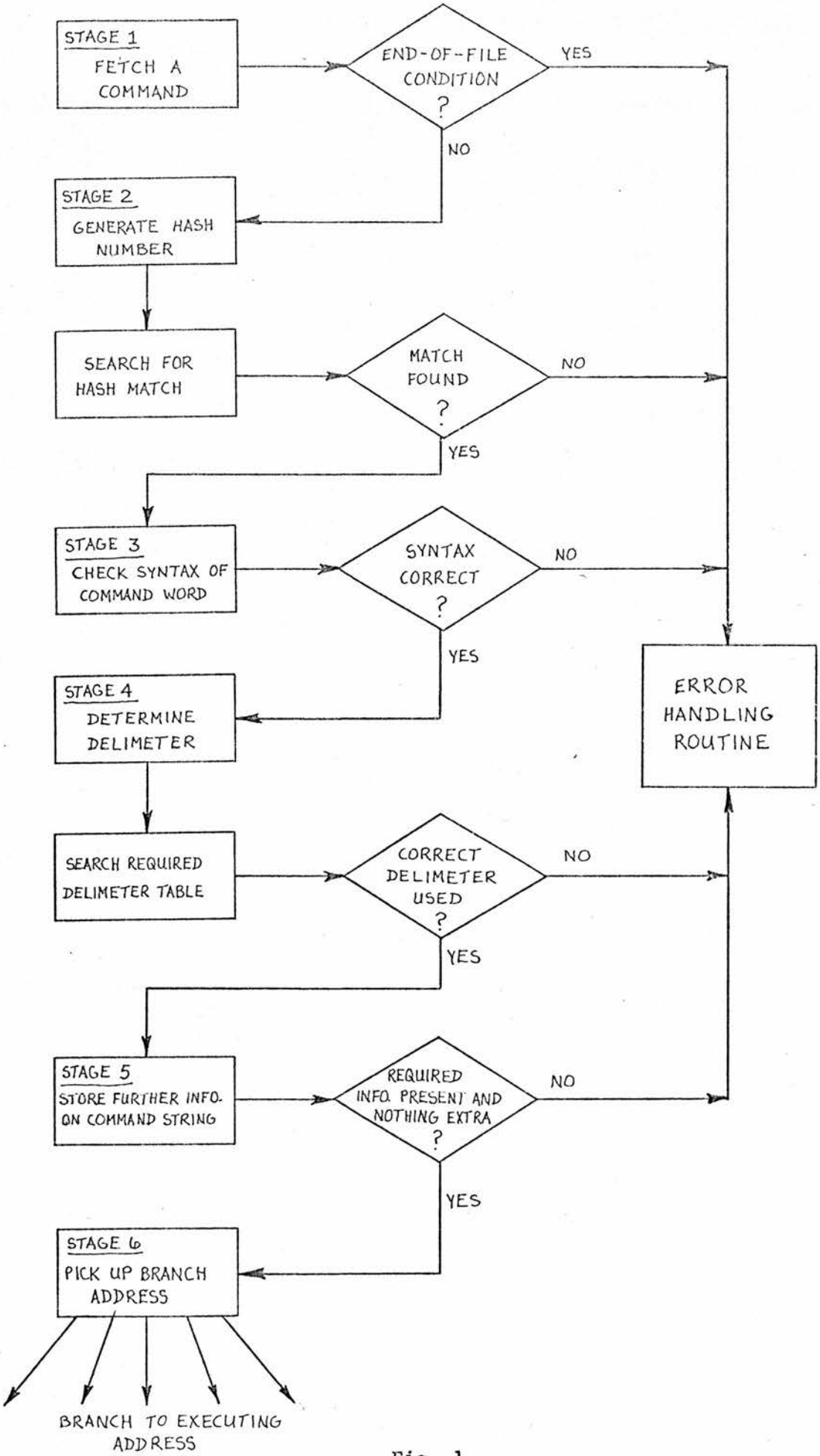

STAGE 1
FETCH A COMMAND
END-OF-FILE CONDITION ?
YES
NO
STAGE 2
GENERATE HASH NUMBER
SEARCH FOR HASH MATCH
MATCH FOUND ?
NO
YES
STAGE 3
CHECK SYNTAX OF COMMAND WORD
SYNTAX CORRECT ?
NO
YES
STAGE 4
DETERMINE DELIMETER
ERROR HANDLING ROUTINE
SEARCH REQUIRED DELIMETER TABLE
CORRECT DELIMETER USED ?
NO
YES
STAGE 5
STORE FURTHER INFO. ON COMMAND STRING
REQUIRED INFO. PRESENT AND NOTHING EXTRA ?
NO
YES
STAGE 6
PICK UP BRANCH ADDRESS
BRANCH TO EXECUTING ADDRESS

Fig. 1

execution eg a copy n lines instruction requires the integer n to be stored for future use. Similarly strings and names are stored from the command string. At this stage a flag is set to indicate whether the command is a line or a character directive.

If the required information does not exist in the command string or if there is an excess of information an error message is printed and further processing of the command terminated.

Finally the analyser causes a branch to the appropriate routine to execute the command. This is done by the selective branching technique. The addresses of the routines are set up in a table and the command number indexes this table, thus obtaining the correct branch address.

The command analyser is only re-entered when the previous command has been completed and more commands are present in the command stream.

The following is a detailed description of each stage in the command analyser.

4.1.1 Stage 1

The first step is to fetch a command from the command stream. The unit number of the command stream and the area of memory where the command will be stored are passed as parameters to the input routine. After a successful read the analyser continues into stage 2. If the read is unsuccessful ie an end-of-file condition exists on the command stream the editor is abandoned, and no files except the input stream are retained.

4.1.2 Stage 2

The command word is picked up from the command string and hashed according to a specified algorithm. If a valid hash number is generated stage 3 is entered otherwise the error handler outputs an error message and processing is discontinued.

The hashing technique causes a unique hash number to be generated for each command. The hashing method used is to "exclusive or" the binary bit pattern of each letter in the command word and then "exclusive or" the final accumulative result with the number of characters in the command word x 2^4. The final result is reduced to an integer less than 90 by subtracting 113 if it lies between 100 and 200 and 220 if it is greater than 200. Negative hash numbers and ones greater than 90 are invalid.

The hash number indexes the hash table (TABLE) which contains pointers to the command table (COMTAB). COMTAB contains information used to check the syntax of the command. If an entry in TABLE is negative there is no corresponding entry in COMTAB and the command is invalid.

4.1.3 Stage 3

The analyser now checks the syntax of the command by using COMTAB. This table contains the number of letters in the command, the command word and a number associated with the command type. If a syntax error occurs a message is written to the print stream via the error handler and the next command fetched. Otherwise the number allocated to the command type is picked up from COMTAB and stage 4 entered.

4.1.4 Stage 4

This stage examines the command delimeter and searches the delimeter tables to determine if the specified character is allowed. eg any copy statement must be followed by a line or character delimeter. If followed by any other character an error message is output. Similarly the command EXIT must be followed by a space. EXIT/ would cause a syntactical error.

By examining the delimeter the command type is established. If any character other than a line or character delimeter follows the command word it is one of the special purpose commands. If the command is a line or character directive then the appropriate marker flags are set up.

4.1.5 Stage 5

Any additional information in the command string is dealt with at this stage. The information is retrieved from the command string and stored in memory for further use during the command execution.

The type of information eg integers or strings, is determined. The command number is matched against the appropriate table to ensure that the information present is required by the specified command.

If the command is part of the body of a loop a table is scanned to check that the given command is valid within a loop.

The exchange command is treated as a special case. The two strings are joined together in one area of memory and two variables hold the lengths of the individual strings.

If the required information does not exist in the command string or if there is an excess of information the error handler outputs the appropriate error message and processing of the command is halted.

4.1.6 Stage 6

At the final stage of analysing the address of the coding to execute the command is retrieved from the table BRANCH by using the command number to index the table. A branch to the specific address is effected and the command executed.

Before branching strings of characters contained in the command are examined and expanded if TAB is in operation. If the string contains a load instruction the actual string is loaded from the store area.

Commands that are part of a loop body are not executed but stored in the loop command area to be executed when the loop is completed.

Flowcharts 1-6 in Appendix II give detailed descriptions of each stage in the command analyser.

By using this method of command analysing it is relatively simple to add more commands to the editor language. Any new command must have a unique hash number less than 90. The COMTAB indexing table must contain a positive entry when indexed by the hash number. This entry is the next available position in COMTAB. Details of the command must be inserted into COMTAB and into the appropriate diameter tables depending on the form of the new command.

The following diagram shows the link up of the tables.

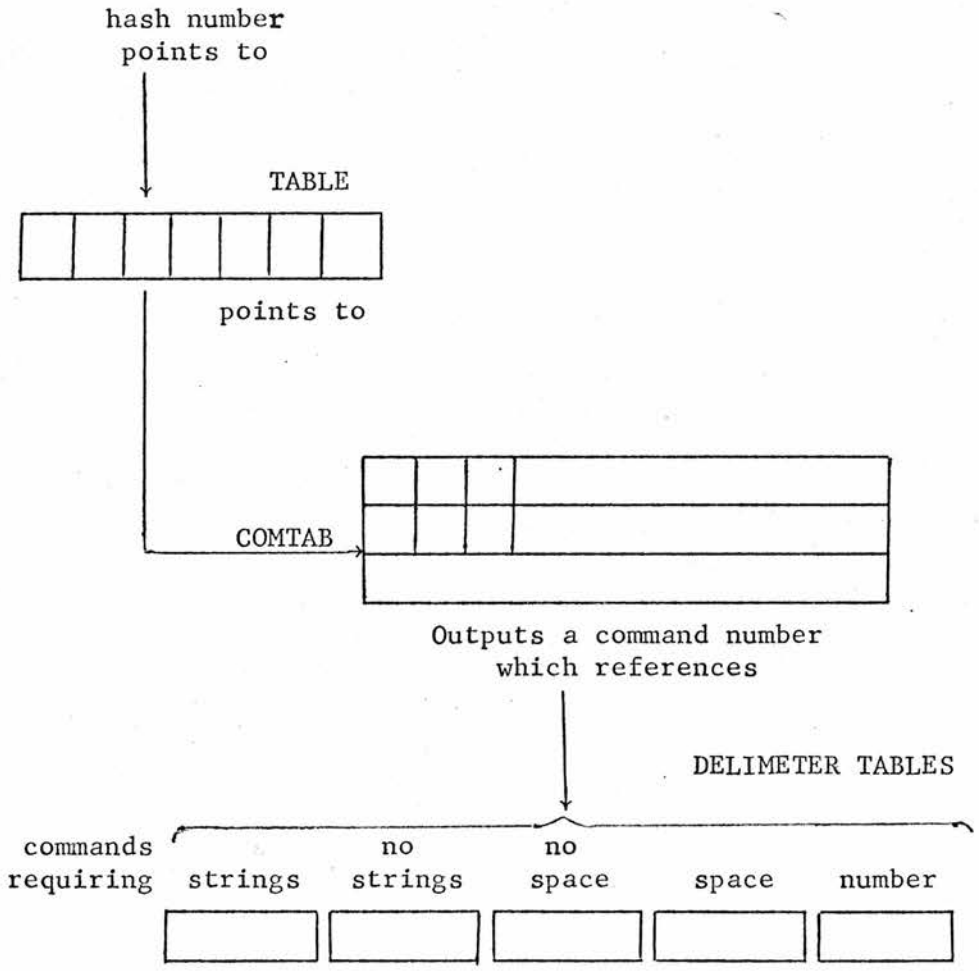


Fig. 2

4.2 Error Handling Routine

The error handler allocates an error number to the error condition and calls a Fortran subroutine. This routine writes an appropriate error message to the print stream and control returns to the error handler.

There are two types of errors - fatal and warning. The fatal error causes processing on the current command to be terminated. After a warning diagnostic, control is returned to the executing routine and processing continues.

4.2.1 Warning errors

There are only 5 warning diagnostics:

a) If a copy n command overflows the output line buffer, n is reduced to the maximum value allowed and processing continues.

b) If an insert command causes the output buffer to overflow characters at the end of the line are lost.

c) If a duplicate character store name is used the old name and store are overwritten, the new string of characters must be less than or equal to the old one. A fatal error message is output if the second string is greater than the first.

d) If the defined field is smaller than 8 positions and sequencing is required the sequence numbers are truncated.

e) If a string contains a load command and the name is not in the character or line store dictionaries a warning diagnostic is given and the load command treated as an input string.

4.2.2 Adding error messages to the system

It is simple to insert new error messages into the editor. A label is

allocated the new error message number. At this label the error message number is stored in memory. The handler then branches to the error routine which calls the Fortran subroutine. The new error message must also be inserted into the Fortran subroutine.

The following diagram shows the flow through the error routine.

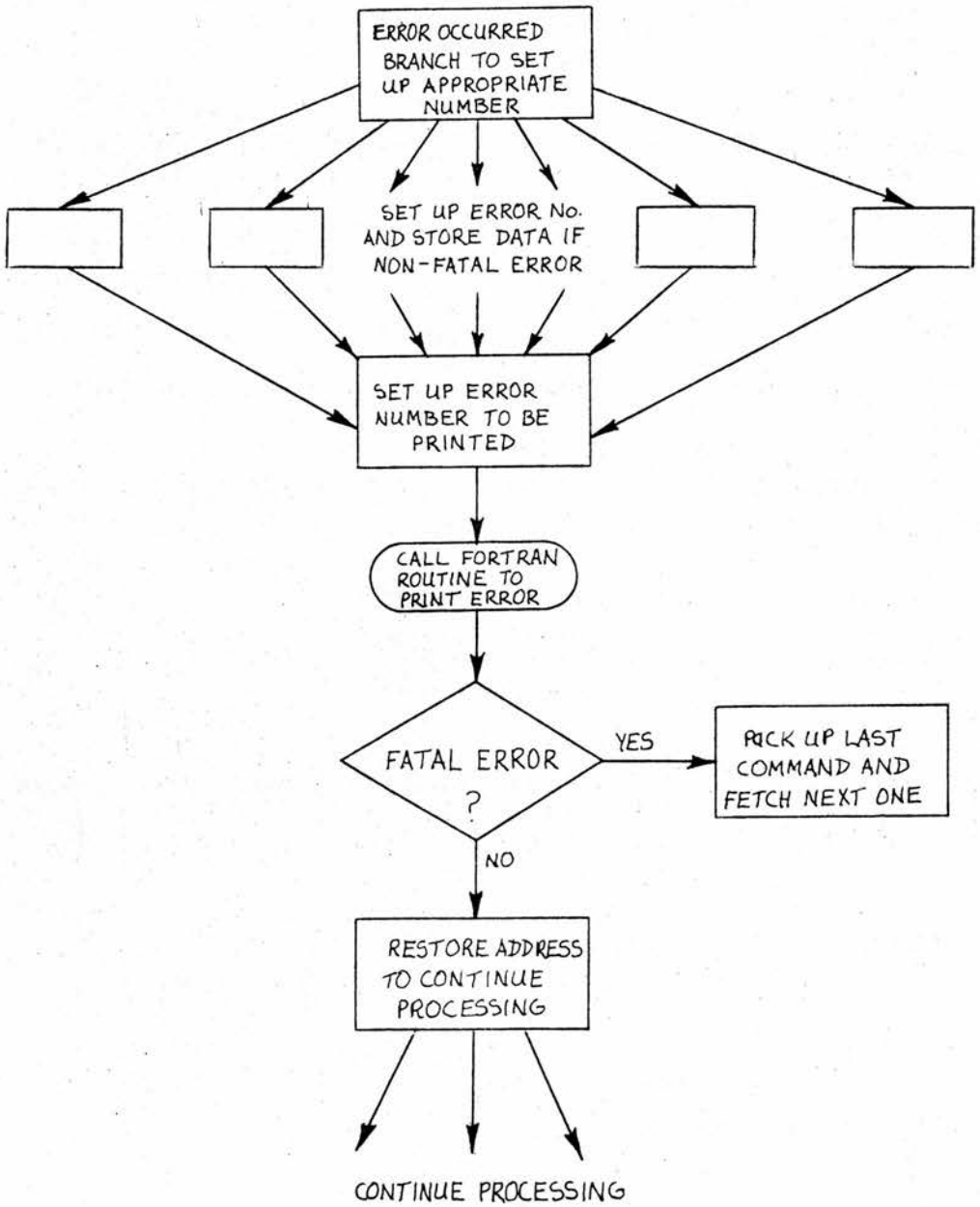


Fig. 3

5 IMPLEMENTATION OF CODE

This section describes the action taken during the execution of certain edit commands. The code generated by the macro processor is in the IBM 360 assembler language. To edit a file this code is executed on the IBM 360.

Chapter 2 describes the action of the editor for simple text editing commands. Two general registers are used to hold the relative addresses of the input line buffer and the output line buffer. Using this technique buffer swapping is achieved by swapping only the contents of the two general registers. Two pointers IPP and OPP are used to keep track of editing within the input line buffer and output line buffer respectively. The edit field is bounded by the start field pointer (SFP) and the end field pointer (EFP). A pointer (EP) is used to mark the last significant character on the input line.

This section describes the implementation of facilities in the editor which do not involve line and character editing.

5.1 Loading and Storing Facility

This facility allows strings of characters or a series of lines to be stored in the editor memory and retrieved at a later stage by the user. All stored characters, including these in line store commands, are held in the character store. A pointer (CHP) is used to mark the next available position in the character store. For storing character strings a character name dictionary is required. This dictionary contains the name of the store area, the number of characters in the stored string and a pointer to the position in the character store of the first character in the string. Similarly for storing lines a line name dictionary is used. This contains the name of the store area, the number of lines to be stored and a pointer to a line position table. This table contains, for each stored line, the number of characters within the line and a pointer to the character store.

5.1.1 Storing characters

Characters can be stored from either the input line buffer or the command stream. The command number indicates which input stream is used. The number of characters in the store string and the name of the store area are found from the command analyser and inserted into the character name dictionary along with a pointer to the next available position in the character store. The specified number of characters are then moved from the appropriate stream to the character store. If the characters are stored from the input stream they are copied to the output stream and the pointers advanced accordingly. Tab characters are not expanded at this stage. If a duplicate store name occurs, the old store is overwritten provided the new string can be contained in the space of the old string. Otherwise a fatal error message is output and the store command ignored. Fig. 4 is a diagram of the dictionary and store for the given commands.

Command stream	Input stream	Output stream
S7.A1	STORE X YZ	STORE X
SC5.A2		
XYZ12		

Pointers indicate positions after the command has been executed.

S Store from input stream

SC Store from command stream.

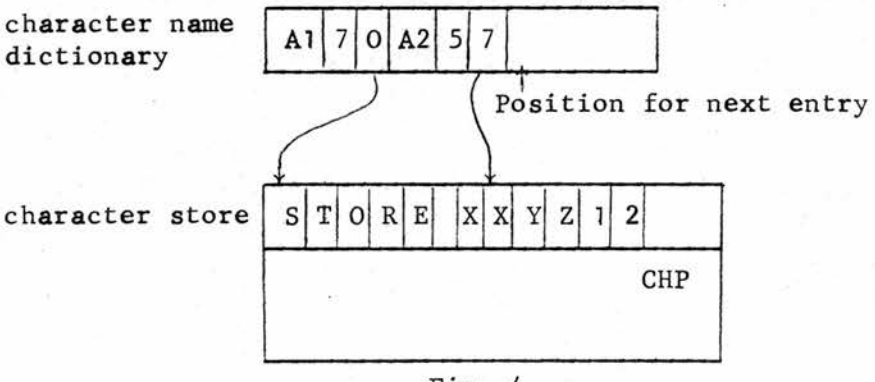


Fig. 4

The number of labelled store areas allowed in an edit run is restricted by the size of the dictionary and the total number of characters stored is

restricted by the size of the character store.

5.1.2 Storing lines

Lines for storing can be input from either the input stream or the command stream. As in character storing the command number determines the input stream. The store area name and the required number of lines are passed from the command analyser and inserted into the line names dictionary along with a pointer to the next available position in the line position table. Lines for storing are then read from the appropriate input stream Trailing blanks are ignored and the number of significant characters in each line is entered in the line position table with a pointer to the next available position in the character store. The significant characters are then moved to the character store.

Because of the complexity involved in differing line lengths duplicate line store names cause a fatal error and the command is ignored.

Fig. 5 shows the table structure of storing lines.

Command stream	Input stream	Output stream
S4/A3	FIRST LINE OF	FIRST LINE OF
SC2/A4	STORE.	STORE
CB.ABC	LOTS MORE	LOTS MORE
I.PQRST	LINES FOLLOW	LINES FOLLOW
	→ THIS ONE	→

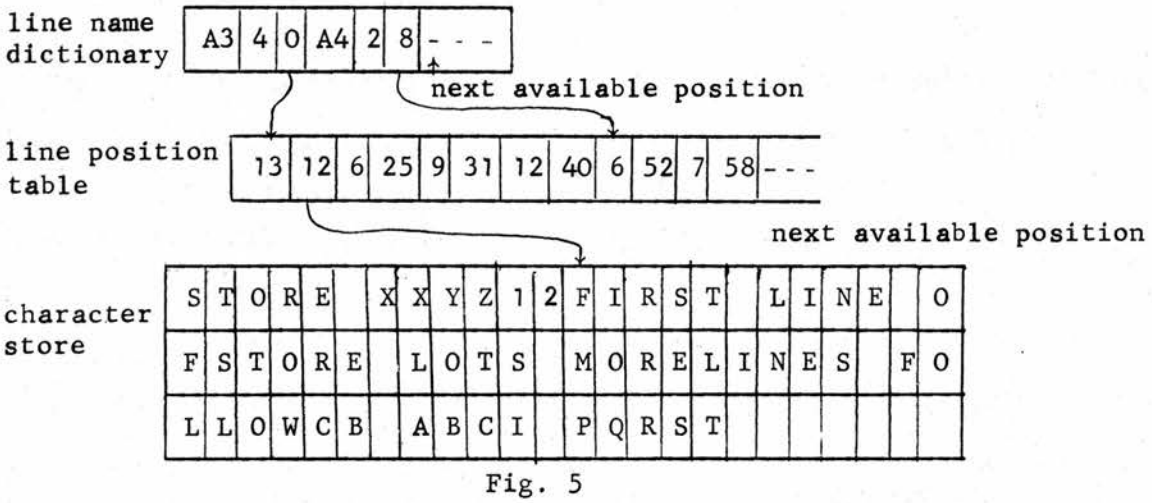


Fig. 5

The number of line store areas is restricted by the size of the dictionary and the total number of lines stored by the size of the line position table. The size of the character store restricts the total number of characters in both line and character store commands.

5.1.3 Loading character store areas

A character load occurs when a load instruction is encountered in the command stream in place of an input string eg CB/L A1. The character names dictionary is searched to find the specified name. The length of the stored string and its position in the character store are picked up from the dictionary and the characters moved from the character store to the command stream i.e.a command CB/L A1 now becomes CB/STORE X. If the tab facility is switched on the string is expanded accordingly except in the case of the exchange command (q.v.).

5.1.4 Loading line store areas

A load instruction in the command stream by itself eg L A3 causes the line name dictionary to be searched. The lines are then located in the character store via the line position table. When the editor is in input mode the required lines are read directly from the character store to the output stream. Otherwise a flag is set indicating to the editor that further commands are read from the store area instead of the command stream Commands are read and executed in this way until the named load area is exhausted.

In both line and character loading, if no match is found in the appropriate dictionary for the supplied name, a warning error message is output and the load instruction treated as an input string.

5.2 Looping

The looping facility is used to execute a sequence of commands a specified number of times or until an end-of-file (EOF) condition exists on the input

stream. All commands between the first loop start command the corresponding loop end command are stored in a loop command area (LOOPCOM) in the editor memory. This area is indexed by a pointer (LCP).

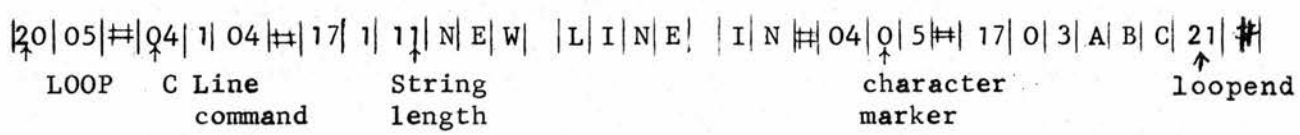
When executing nested loops two stacks are required. One holds the return address of each nested loop and the other holds the number of times each nested loop has to be executed. The function of these two stacks is described in Section 5.2.2.

5.2.1 Storing loop commands

When a loop command is encountered on the command stream a flag (LPSET) is set to indicate that further input on this stream has not to be executed but inserted into the loop command area. Each command within the loop is processed through the command analyser and, if syntactically correct, is stored in its coded form in the loop command area. This form consists of the command number, the command type (a line or character command) and any further information required when executing the command. A hash sign (#) is used to mark the end of each command in the loop command area eg the simple loop

```
LOOP 5
C4/
I/NEW LINE IN
C5.
I.ABC
```

LOOPEND is stored in the loop command area as follows:



For nested loops a counter LOOPCNT is used. This is increased each time a nested loop is encountered and decreased on the occurrence of a LOOPEND statement. When the count returns to zero the loop is "closed" and ready for execution

Load instructions and strings containing the tab character are not expanded

at this stage. They are copied directly to the loop command area and expanded during the execution of the loop.

5.2.2 Execution of the loop

The execution phase is entered immediately after the closing LOOPEND statement has been encountered. Commands are read from the loop command area and executed. The loop execution routines cause a branch to the command analyser where strings within the command are checked for tab stops or load instructions. These strings are expanded at this stage.

A flag is set to indicate that commands are to be input from the loop command area instead of from the command stream. A pointer (LPCOMP) marks the position of the next command in the loop command area Details of each command are read from this area and set up as they would have been from the command analyser. A branch to the selective branching section of the command analyser is effected. When the command has been successfully executed control is returned to the loop execution routine and the next command read from the loop command area.

For all looping, nested or not the address in the loop command area of the first instruction of the loop is placed on top of a loop address stack (LOOPSTK) and the number of times the loop has to be executed is placed on top of the loop number stack (LOOPNO). No branching to the command analyser is effected. This procedure occurs each time a loop n statement is encountered. When a LOOPEND statement occurs the top value of the loop number stack is examined. If this is non-zero it is decreased by one and replaced on the stack. The address in the loop command area of the first command within the current loop is picked up from the top of the loop address stack and commands executed from this address. When the top value of the loop number stack becomes zero it is discarded along with the corresponding entry in the loop address area. The next entry in the stack is examined

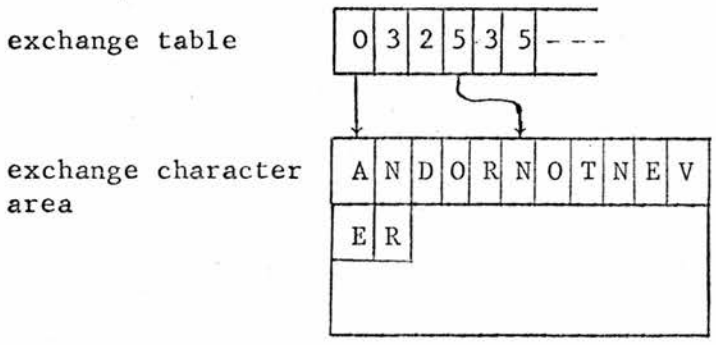
The procedure continues till the loop number stack is empty. This implies that the loop has been completed. The loop command area is then set empty and the flag LPEX unset indicating that subsequent commands are now to be read from the command stream.

The number of nests permitted within a looping structure is limited by the size of the loop address stack and the loop number stack. The total number of commands within the loop body is restricted by the size of the loop command area. However, it poses no problem to alter the size of these areas.

5.3 The Exchange Facility

This powerful feature of the editor allows one string of characters to be replaced by another while other editing proceeds.

Both strings on the exchange command are stored in the exchange character area. The exchange table contains, for each exchange instruction, a pointer to the start of the exchange string in the exchange character area, and the lengths of both strings. When the exchange facility is in operation every input line is scanned for the occurrance of the first string which when found is replaced by the second exchange string.
eg the commands EX/AND/OR/ and EX/NOT/NEVER/ cause the tables to be as follows:



Individual exchanges can be switched off during an edit run. A negative entry in the pointer position in the exchange table indicates that this

exchange is no longer operative.

eg the command NEX/NOT will produce in the exchange table:

0	3	2	-1	3	5	. . .

5.3 1 Storing exchange commands

An exchange command causes the flag EXCH to be set "on" This flag indicates to the editor that whenever a new line is read from the input stream it must be checked for the occurrence of exchange strings and altered accordingly.

When an exchange command has been successfully processed by the command analyser both strings are joined together in the area STRING, the variables STLEN and STLEN2 hold the respective string lengths The strings are then examined for the occurrence of load instructions and the flag XLD set as follows:

XLD = 0 if neither string is a load instruction
eg EX/AND/OR/

= 1 if the first string only is a load instruction
eg EX/L A1/XYZ/

= 2 if the second string only is a load instruction
eg EX/ABC/L A3/

= 3 if both strings are load instructions
eg EX/L A1/L A2/

= - ve when the exchange has to be switched off (see Section 5.3.3)
eg NEX/AND.

If XLD is 1, 2 or 3 the strings are expanded to contain the stored characters. The new values of STLEN and/or STLEN2 are saved

The next step is to set up in the exchange table the values of the pointers (XCP) to the next available position in the exchange character area (STLEN and STLEN2). The strings are stored in the exchange character area

and XCP updated accordingly.

The number of exchange commands in an edit run is restricted by the size of the exchange table, and the total number of exchange characters by the size of the exchange character area. Switching off exchange commands does not create extra space except when all exchanges have been switched off. When this occurs the exchange table and the exchange characters area are set empty.

5.3.2 Operating the exchange

In order to discontinue an exchange operation an NEX/string command is issued by the user. When such a command is encountered the flag XLD is set negative The exchange table is scanned and when a matching first string length is encountered the corresponding string in the exchange character area is compared with the string in the command stream. If a match is found the pointer position in the exchange table is set negative indicating that the exchange is no longer operative. When all entries in the exchange table are negative the exchange flag EXCH is unset and the exchange table and exchange character area are set empty.

An error message is output to the print stream if the exchange string cannot be matched in the exchange string area. Control is then returned to the command analyser to read in the next command.

It is advisable to switch off the exchange facility as soon as it is no longer required as the exchange operation is very time consuming

5 4 Sequencing

The sequence command is only scanned as far as the command word by the command analyser It then causes a branch to the sequencing section ignoring the remaining information in the command string This information is read and analysed in the sequencing section

The sequence numbers are inserted into the last eight positions of the defined field - by default in columns 73-80 of the current line. If the defined field is less than eight characters in length, the sequence numbers are truncated on the left and an error message written to the input stream. The eight character sequence area is divided into two fields of four characters each. The information on the sequence instruction is then analysed. This information consists of the starting value of the first field, its increment, the starting value of the second field and its increment. If the increment is zero the string is treated as an alphametric string. If the first two values on the sequence command is zero the two four-character fields are treated as one eight-character field This allows for sequence numbers greater than 9999 A flag ONEF is set when this condition exists.

The sequence number is built up and stored in an area in the editor memory called STRING. Alphameric constants are stored in the appropriate position in the area STRING ie. If the first increment is zero the first entry on the sequence command is stored in columns 1-4 of STRING and if the second increment is zero the third entry of the sequence command is stored in columns 5-8 of STRING. The incremented number is then stored in the empty field. If neither field is constant both numbers are incremented and stored in the appropriate fields. If the flag ONEF is set the incremented number is stored in columns 1-8 of the area STRING The first eight characters of STRING are copied to the last eight positions of the defined field in the input line buffer which is then written to the output stream The next line is read into the input line buffer from the input stream

This process continues until an end-of-file condition exists on the input stream The pointer remains at the end-of-file marker, thus if any further editing is required a START/command must be issued first

5.5 Field Defining

This facility enables the user to define the edit field within the input line buffer. By default the field is columns 1-80 The variable SFP contains the start of field pointer and EFP contains the end of field pointer.

As in the sequencing command only the command word is processed through the command analyser. The further information on the field command is analysed by the field command execution. The first integer on the field command is the start field pointer. If this integer does not exist the previous SFP remains in operation. The second integer in the command is the end field pointer and again the previous EFP is retained if no number exists in the field command. The final values of SFP and EFP must be in ascending order. SFP must be greater than zero and EFP must not be greater than 80. If any of these conditions are violated an error message is output and a new command read from the command stream.

The new position of the pointers are then set up in the current line before the routine branches back to the command analyser to fetch the next command

5.6 The Tab Facility

This useful facility allows the user to space out strings A non-alphameric character is to mark the tab stops. When this character occurs within an input string the string is space filled to the next tab stop

5.6.1 The tab command

As in the sequencing and field commands the tab command is processed by the command analyser only as far as the command word.

When a tab command occurs a flag TAB is set to indicate that the tab facility is in operation. The tab character is retrieved from the command string and stored in TABCHR. If this character is a question mark (?) the

next character is examined. If this also is a question mark the tab facility is switched off by setting the flag TAB to zero. Control is then returned to the command analyser to fetch the next command. Otherwise processing continues normally.

The tab stop positions are read from the command string and stored in a temporary array TABT. The final tab stops are stored in the array TABF. If no integer is present for any tab stop (ie the condition ",," occurs in the command stream) the corresponding tab stop in the array TABF is inserted into TABT provided it is greater than the last entry in TABT Tab stops must be in ascending order. If this is not the case an error message is output to the print stream and the tab instruction ignored. When an error occurs in the tab command the previous tab settings, if any exist, are retained.

A maximum of 9 tab stops is allowed. This is limited by the size of the arrays TABT and TABF. These arrays are easily altered If more than the maximum number of tab stops is significant an error message is sent to the print stream and the tab command ignored. If there are less than the maximum number of tab stops the remaining entries in TABT are sent to the maximum buffer size (ie 80). When all tab stops have been set up in TABT and no errors have occurred the array TABT is copied to TABF which always contains the final positions of the tab stops.

5.6.2 Expanding strings with tabs

A tab check is applied to all input strings on the command stream An area of core called STRING is used to hold the final expanded string This is initially set blank. If the tab flag is not set the characters are copied directly from the command stream to the area STRING Otherwise each character in the command stream is examined and matched with the tab character.

If a match occurs the next tab stop is retrieved from the array TABF and the area STRING is space filled up to this stop. Otherwise the character is copied to the next position in STRING. The new expanded string length is stored in STLEN and control returned to continue executing the command.

Errors occur when there are too many tab characters in the input string or when the area STRING has been filled beyond the current tab stop. If an error occurs the whole command is ignored.

6 CONCLUSIONS

6.1 The Editor

The context editor TED has been implemented satisfactorily on two operating systems available on the IBM/360 44 at St Andrews University It works in an on-line capacity under the RAX operating system and in batch mode under the 44MFT operating system. TED is efficient and very easy to use under both operating systems.

There are several improvements that can be made to the editor to make it more powerful. One major improvement would be a re-design of the sequencing facility. Sequence numbers should be applied to the edit file while other editing is in progress as in the exchange facility. The sequence command could be made more flexible to cope with varying field sizes and sequencing methods.

Restrictions on buffer and store area sizes are not desirable but it is difficult to get round this problem. One method would be for the user to specify his own restrictions which would be inserted into the editor during execution. This would make the editor more complicated to use A second method would be to use the dynamic storage technique but not all computer manufacturers provide the software to deal with this facility This method causes storage from a central pool to be allocated to the editor as required and linked together by a pointer network.

A backspacing line facility would be a useful addition to the editor. Limited backspacing of lines could be achieved by extending the input/output buffers. Several lines could be input at a time and backspacing within the buffers is a relatively simple operation. Unlimited line backspacing requires extensive input/output operations and file positioning This is very time and space consuming in the computer. Using the present version of the editor back-spacing can be achieved by a START/command followed by a command to locate

the required line.

6.2 On-line/Batch Ability

TED is a better batch editor than an on-line editor because the output listings are geared more to suit the batch user By altering the print facility TED could become an excellent on-line editor as it contains the powerful display feature which can only be used in an on-line environment. A good on-line editor should display the current line and all edited lines during processing. The user can then determine at a glance his position on the edit file.

6.3 Portability

TED is not very portable. It would take a fair amount of work to produce a working version of the editor on another computer although it would require much more effort to completely rewrite the editor. Portability can be improved by introducing more levels of abstract machines (see Section 3). By breaking the editor down into a system of abstract machines a set of simple assembler instructions for the required computer should be all that is necessary on the lowest level of abstract machines A problem arising from this method is that the efficiency of the editor is reduced by the introduction of lower levels of abstract machines A balance has therefore to be found between efficiency and portability

REFERENCES

1 POOLE, P. C. Hierarchical abstract machines. UKAEA, Culham Laboratory, near Abingdon, Berkshire.

2 NEWEY, M. C., POOLE, P. G. and WAITE, W. M. Abstract machine modelling to produce portable software - A review and evaluation. Software - practice and experience, Vol 2, 107-136, 1972.

3 WAITE, W. M. I/O Conventions for abstract machines. Department of Information Science, Monash University, Clayton, Victoria 3168, Australia.

4 POOLE, P. C and WAITE, W. M. The STAGE 2 macroprocessor users reference manual. Culham Laboratory, Berkshire, CLM-PDN 6/70.

5 POOLE, P. C. MITEM - A portable program for text manipulation.

6 CCTAN3 - A users guide to COTAN, amend command. Culham Laboratory.

7 A text editor, NEL memo CD70, November 1971. NEL, East Kilbride, Glasgow.

8 BOURNE, S. R. A design for a test editor. Computing Laboratory, University of Cambridge. Software - Practice and Experience Vol 1, 73-81, 1971.

9 DEUTSCH, L. P. and LAMPSON, B. W. An on-line editor University of California. Communications of the ACM Vol 10, No 12, December 1967.

10 FREEMAN, A. PDP-8 Context Editor (Mark 4), August 1968 Computer Science Department, Edinburgh University.

11 VAN DAM, A. and RICE, D. E. On-line text editing. Computing Surveys Vol 3, No 3, September 1971.

ACKNOWLEDGEMENTS

I would like to offer thanks to all who made this project possible.

The Civil Service department for awarding a bursary enabling me to attend University for 2 years.

Professor J Cole, Head of St Andrews University Computing Department and all his staff, especially Mr R Morrison for his invaluable assistance and supervision of the project.

The typing and tracing departments at National Engineering Laboratory, East Kilbride for greatly assisting in the production of this thesis.

A P P E N D I X I

TED

A CONTEXT EDITOR USER'S MANUAL

R. Morrison
E. Howson
J. Sommerville

Technical Report No. CL/73/3

Contents Page

ACKNOWLEDGEMENTS

During the course of development of TED a number of people have been involved. To them my thanks. The Editor manual has been produced using the FORMAT 44 documentation program. I would therefore like to thank Miss Glynis Fairlie and Mrs. Doris Simmonite for helping in preparing the manual and for their patience with us during our teething troubles with FORMAT 44

INTRODUCTION

TED is a sophisticated, powerful context editor which may be used in a time-sharing mode under RAX or in a batch mode under 44MFT.

This manual describes the editor, the editing commands, the method of use, and the JCL required to use the editor under RAX and 44MFT.

THE DESIGN OF THE EDITOR

The editor is designed to read line images from disc or tape files, perform the necessary editing, and to write a new edited file back on to disc/tape.

The editing method may best be understood by visualising a position pointer which may be moved through the text to where an amendment is to be made. The pointer is initially set before the first line of the file and may be positioned anywhere within the file by the use of line directives, which move the pointer through the file line by line, and character directives which position the pointer within a line.

When editing, a line is read from the input file to an internal 80 byte buffer called the input line buffer (ILB). The edited form of the line is held in a similiar buffer, the output line buffer (OLB). On the input of a new line to the ILB, the OLB is written directly to the new output file.

The file pointer, when within the ILB, is termed the input position pointer (IPP). It indicates the current character position within the line being edited. A corresponding pointer in the OLB, the output position pointer (OPP), indicates the current character position in the edited line. These pointers do NOT move in conjunction but move independently of each other.

THE INPUT-OUTPUT SYSTEM

The I/O system for the editor consists of 4 separate streams.

1) The Input Stream.

This is stream 2 under MFT or stream 5 under RAX. The file to be edited is read from disc or tape. Note that, under RAX, only files on the SYSLIB discs (ie those files saved by a /SAVE command) may be edited.

2) The Output Stream.

This is stream 3 under MFT and stream 10 under RAX . The new edited file is written on to disc or tape.

3) The Command Stream.

This is stream 5 under MFT or stream 9 under RAX. The commands are input either from cards or from the 2260 screen.

4) The Print Stream.

This is stream 6 under both systems. A listing of the commands and the edited file is produced on the line printer or on the 2260 screen.

At present both input and output files must be on tape or disc. There is no provision for connecting the card reader to the Read stream or the card punch to the Write stream.

THE EDIT COMMANDS.

A command to the editor consists of one or more characters which form a mnemonic for the operation required. The commands can be classified into 2 groups:

(a) those which make textual amendments and/or move the position pointer.

(b) auxiliary commands which make no amendments.

** Pointer-moving Commands

This group of commands is split into line and character directives.

A line command causes the pointer to be set at the beginning of a line and moved sequentially through the file, line by line. A line command is active from the current pointer position if this is at the beginning of a line otherwise the current line is copied to the output file and the line command becomes active from the following line.

A character command moves the pointer character by character through the file.

By default the delimiting symbol of a line command is a slash (/) and for a character command a dot (.). These may be changed by using the LINE (qv) and CHAR (qv) commands.

** Auxiliary Commands

These commands add additional facilities (eg provision for executing a sequence of commands in a loop) to the basic amendment commands and they considerably increase the power of the editor.

A full description of all the available commands is now given along with examples of each in use. In each example the position pointers are represented by a *. Unless otherwise stated, the furthest left (or upper)* in a buffer represents the pointer position before execution of the command, and the rightmost (or lower)* the position after execution.

Note that each edit command MUST be input on a separate line or card starting in column 1.

1. COPY COMMANDS.

These commands set the position pointer and copy the input file to the output file up to the pointer position. There are a total of ten copy directives, comprising five line directives and five character directives.

1.1. Copy A Given Number Of Characters Or Lines -- Cn. Or Cn/

This command moves the pointer along n characters or down n lines. If a number is omitted, n is assumed to equal 1. If the command is a character command then n must be less than the length of the line.

Examples.

(a) Copy 8 characters -- C8.

ILB	OLB
SU*BROUTINE* INPUT	SUBROUTINE*

(b) Copy 2 lines -- C2/

INPUT	OUTPUT
*SUBROUTINE INPUT	SUBROUTINE INPUT
INTEGER A,B	INTEGER A,B
*LOGICAL C	*

1.2 Copy File To Before A Given String -- CB.<string> Or CB/<string>

This command moves the pointer through the file and places it before the first occurrence of the given string.

If used as a line command, the string should be the first string on that line, as for an n character string only the first n characters of each line are checked. If there are preceding blanks at the start of a line (eg in a FORTRAN program) these are ignored unless they are all specified as part of the string.

Examples.

(a) Copy to before BUFFER -- CB.BUFFER

ILB	OLB
*C SEARCH INPUT*BUFFER	C SEARCH INPUT*

(b) Copy to before IF -- CB/IF

INPUT	OUTPUT
KL=PNTR	KL=PNTR
BYTE=NUMB	BYTE=NUMB
*IF(NUMB.EQ.64)GOTO 6	*

1.3 Copy File To After A Given String -- CA.<string> Or CA/<string>

This works in the same way as a CB command except that the pointer is placed after the given string.

Examples.

(a) Copy to after BUFFER -- CA.BUFFER

```
ILB                              OLB
*C SEARCH INPUT BUFFER*          C SEARCH INPUT BUFFER*
```

(b) Copy to after IF -- CA/IF

```
INPUT                            OUTPUT
*KL=PNTR                         KL=PNTR
BYTE=NUMB                        BYTE=NUMB
IF(NUMB.EQ.64)GOTO 6             IF(NUMB.EQ.64)GOTO 6
* GOTO 10                        *
```

1.4 Copy To Before Last Character Or Line -- CBL. Or CBL/

This command, if a character directive, copies all the current input line up to the last non-blank character and places the pointer before it.

If it is a line command the whole input file up to but excluding the last line is copied to the output file.

Examples.

(a) Copy to before last character -- CBL.

```
ILB                              OLB
*INTEGER A,B,C,*D                INTEGER A,B,C,*
```

(b) Copy to before last line -- CBL/

```
INPUT                            OUTPUT
*---                             ---
 ---                             ---
 ---                             ---
 CALL EXIT                       CALL EXIT
*END                             *
 eof
```

1.5 Copy To After Last Character Or Line -- CAL. Or CAL/

This command works as CBL except that the pointer is placed after the last character or line.

Example.

(a) Copy to after last character -- CAL.

```
ILB                              OLB
```

```
*INTEGER A,B,C,D*                    INTEGER A,B,C,D*
```

(b) Copy to after last line -- CAL/

```
 INPUT                              OUTPUT
*---                                ---
 ---                                ---
 ---                                ---
 CALL EXIT                          CALL EXIT
 END                                END
*eof                                *
```

2. DELETE COMMANDS

The delete commands are similar to the copy commands except that the pointer position in the OLB is unchanged ie nothing is output. Each command is illustrated by an example.

2.1 Delete A Given Number Of Characters Or Lines --Dn. Or Dn/

Examples.

(a) Delete 6 characters -- D6.

```
 ILB                                OLB
SUBROUTINE *OUTPUT*                 SUBROUTINE *
```

(b) Delete 2 lines -- D2/

```
 INPUT                              OUTPUT
DO 10 I=1,J                         DO 10 I=1,J
* A=B/C                             *
IF(A.GT.N) GOTO 20
*10 N=N+1
```

2.2 Delete To Before A String --DB.<string> Or DB/<string>

Examples.

(a) Delete before IN -- DB.IN

```
 ILB                                OLB
C *VARIABLES *INITIALISED           C *
```

(b) Delete to before RETURN -- DB/RETURN

```
 INPUT                              OUTPUT
GOTO 5                              GOTO 5
* STRP=1                           *
PQRP=5
* RETURN
END
```

2.3 Delete To After A String -- DA.<string> Or DA/<string>

Examples.

(a) Delete to after IN -- DA.IN

```
 ILB                              OLB
C *VARIABLES IN*ITIALISED        C *
```

(b) Delete to after RETURN -- DA/RETURN

```
 INPUT                            OUTPUT
GCTO 5                           GOTO 5
I=7                              I=7
* STRP=1                         *
PQRP=5
RETURN
*END
```

2.4 Delete To Before Last Character Or Line -- DBL. Or DBL/

Examples.

(a) Delete to before last character -- DBL.

```
 ILB                              OLB
DIMENSION *A(20),B(3*)           DIMENSION *
```

(b) Delete to before last line-- DBL/

```
 INPUT                            OUTPUT
*---                             *
---
---
CALL EXIT
*END
eof
```

2.5 Delete To After Last Line Or Character -- DAL/ Or DAL.

Examples.

(a) Delete to after last character -- DAL.

```
 ILB                              OLB
DIMENSION *A(20),B(3)*           DIMENSION *
```

(b) Delete to after last line -- DAL/

```
 INPUT                            OUTPUT
*---                             *
---
---
CALL EXIT
END
```

* eof

3. Insert commands

These commands insert a string of characters or lines into the file, setting the position pointer after the insert. There are five insert commands.

3.1 Insert A String Of Characters Into A Line -- I.<string>

This command inserts the given string immediately after the output pointer position.

If the insertion is such that the OLB will overflow the last characters in the ILB will be lost and a warning message output.

Example.

Insert the character string EDIT -- I.EDIT

ILB	OLB
//* EXEC FORTRAN	//EDIT*

3.2 Insert A Line Into The File -- I/<string>

This command inserts a line into the input file. If the position pointer is set at the beginning of a line the insert will be made before that line otherwise it will be made after the line containing the position pointer.

Example.

Insert the line C INITIALISE VARIABLES--I/C INITIALISE VARIABLES

INPUT	OUTPUT
INTEGER IN,OUT	INTEGER IN,OUT
*DATA IN,OUT/5,7/	C INITIALISE VARIABLES
	*

3.3 Insert Several Lines Into The Input File I --- FIN

To use this command, I is input on a line by itself followed by the required insertions. The insertions MUST be terminated by a FIN on a line by itself

Example.

Insert into input file the 2 lines

```
SUBROUTINE WRITE(A,B,C)
INTEGER A,B
```

The command stream for this would be

```
I
SUBROUTINE WRITE(A,B,C)
INTEGER A,B
FIN
```

```
INPUT                          OUTPUT
END*                           END
                               SUBROUTINE WRITE(A,B,C)
DIMENSION C                    INTEGER A,B
                               *
```

3.4 Insert Blanks -- IBn. Or IBn/

This command inserts n blank characters or lines into the input file.

Examples

(a) Insert 4 blanks into a line -- IB4.

```
 ILB                           OLB
C *SET N=0                     C    *
```

(b) Insert 2 blank lines -- IB2/

```
 INPUT                         OUTPUT
EN*D                           END
*
SUBROUTINE ARR(X)
                               *
```

4. REPLACE COMMANDS

These commands replace a given string or line in the input file with a new string or line. A replace command is equivalent to a delete followed by an insert.

4.1 Replace Characters Or Lines R.<string> Or R/<string>

This command is best illustrated by example.

Examples.

(a) Replace the next 6 characters with OUTPUT -- R.OUTPUT

```
 ILB                           OLB
*INPUT *FILE                   OUTPUT*
```

Note that as OUTPUT is one character longer than INPUT the space between INPUT and FILE will be lost if no blanks are inserted.

(b) Replace INTEGER A,B with REAL I,J,K

```
R/      REAL I,J,K

 INPUT                         OUTPUT
SUBROUTINE *ANT(A,B)           SUBROUTINE ANT(A,B)
INTEGER A,B                    REAL I,J,K
*DIMENSION C                   *
```

Note that in editing a FORTRAN program, the preceding blanks must be included in the R/ statement otherwise the replacement will be made from column 1 of the defined field.

4.2 Replace With Blank Characters Or Lines--RBn.or RBn/

This command replaces a string or line in the input file with blanks.

Examples

(a) Replace 2 characters with blanks.. RB2.

ILB	OLB
*20*WRITE(6,10) A	*

(b) Replace next 2 lines in file after pointer with blanks

INPUT	OUTPUT
---	---
---	---
* WRITE(6,10) A	
20 CONTINUE	
* A=B/C	*

5. BACKSPACE COMMANDS

There are three backspace instructions which may only operate on the current line ie backspace is a character command only. Backspacing takes place on the edited version of the line.

When a backspace command is executed, the ILB is copied from the IPP to the OLB. The buffers are then exchanged ie the OLB becomes the new ILB. The new IPP is then computed according to the instructions given in the backspace command and the ILB is copied to the OLB up to the new IPP.

5.1 Backspace To After A String -- BA.<string>

This command places the pointers after the given string.

Example.

Backspace to after SET -- BA. SET

Before execution of the commandthe buffers are

ILB	OLB
C SET POINTERS* ABC	C SET BUFFER POINTERS*

The execution of the command swaps the ILB and OLB and resets the pointers

ILB	OLB
C SET* BUFFER POINTERS ABC	C SET*

5.2 Backspace To Before A String -- BB.<string>

This works as BA. Except that the pointers are now set before the given string.

Example.

Backspace before SET -- BB.SET

Before execution

ILB	CLB
C SET POINTERS* ABC	C SET BUFFER POINTERS*

After execution

ILB	OLB
C *SET BUFFER POINTERS ABC	C *

5.3 Backspace A Given Number Of Characters -- Bn.

This command causes the pointers to be moved back by n characters.

Example.

Backspace 4 characters -- B4.

Before execution

ILB	OLB
IF(IN.EP.28)*IN=3	IF(IN.EP.2)*

After execution

ILB	OLB
IF(IN.E*P.2)IN=3	IF(IN.E*

6. START COMMANDS

The START instructions START. And START/ cause the pointer to be set at the beginning of the current line or at the beginning of the file.

In the case of the START instruction, the ILB, after the pointer, is copied to the CLB and the OLB becomes the new ILB with the pointer set at column 1 of the defined field.

Similarly for a START/ instruction: the input file is copied from the pointer position to the output file and this then becomes the new input file with the pointers at the beginning of the file.

7. THE FIND COMMAND -- F/<string>

This is a line directive only and sets the pointer at the beginning of the line containing the given string. It is exactly equivalent to a CA.<string> followed by a START.

8. THE NEWLINE COMMAND -- N

Input of an N on a line by itself outputs the current line whether it has been edited or not and reads in the next line of the ILB, setting the pointer at the beginning of the line.

9. THE EXIT COMMAND -- EXIT

This is the normal terminator to a sequence of edit commands. The remainder of the file that is being edited is copied to the new file. If this is not included as the last command, no editing is saved and no listing produced.

10. THE FIELD DEFINITION COMMAND -- FIELD m,n

This command effectively redefines the line size on which editing can take place. Characters outwith the defined field are copied from the input to the output file and cannot be acted upon by edit commands.

The FIELD m, n instruction defines the start of the field to be at column m and the end of the field at column n. If no field is specified the editor acts on columns one to eighty in each input line.

11. TAB SETTING COMMAND T £, k, l, m

The tab setting command allows strings to be input with no spacing between them and yet to be laid out correctly on the output file eg. It may be set so that FORTRAN statements always begin in column 7.

The tab statement contains the tab character (set above to £) and a list of between 1 and 9 tab stop points. When the tab stops have been set, input strings are scanned and space filled accordingly.

Tabs may be changed by inputting another T statement. Stops not requiring alteration may be left blank. The tab character may be any printable character and tab may be switched off by inputting T ??

Example: COMMAND SEQUENCE

```
T £,10,20,30
I/£AB£CD£EF
I/12£34£56
T %,,,35
I/A1%B2%C3%D4
T ??
I/MNOPQR£STU
```

Will output lines

```
COL 1          COL 10          COL20          COL30          CCL35
                AB              CD             EF
12              34              56
A1              B2              C3                            D4
MNOPQR£STU
```

Note that the omission of a tab character before the first string will set the first tab stop to column 1 of the defined field.

12. THE SEQUENCING COMMAND -- SEQ a,b,c,d,e

This command enablesthe user to sequence his file. At present this facility is not available for release.

13. STORE AND LOAD COMMANDS -- Sn., SCn., Sn/, SCn/, L

These instructions allow an often used character string or instruction sequence to be stored from either the command stream or from the input file and called when required by a simple load instruction. There are two store areas in the editor. Characters are held in the character store and lines in the line store. The string is stored in the appropriate area under a given name which may have up to four characters.

13.1 Store Characters From The Input Stream -- Sn.<name>

This command stores n characters from the input stream and labels them with the given name.

Example.

Store 4 characters and label them S1 -- S4.S1

```
ILB
SUBROUTINE *PTIN(A,B)
```

The characters PTIN will be stored in the character store under the name S1. The position of the pointer remains unchanged. The stored characters may be retrieved by using the load command as follows.
J/L S1 inserts PTIN as a line
I.L S1 inserts PTIN as a character string
CB.L S1 copies to before PTIN

13.2 Store Characters From The Command Stream -- Scn.<name>

This command stores n characters from the command stream and labels them with the given name

Example.

Store 10 characters from the command stream and label them A1 -- SC10.A1. The instruction sequence should be:

```
SC10.A1
SUBROUTINE - characters to be stored.
next edit command.
```

This may be loaded in exactly the same way as an Sn instruction.

13.3 Store Lines From The Input File -- Sn/<name>

This instruction stores n lines from the input file and labels them with the given name.

Example.

Store 3 lines from the input file and label them C2--S3/C2

INPUT FILE

```
*
SUBROUTINE PTIN(A,B)
INTEGER A
LOGICAL B
* REAL M,N
```

The first three lines will be stored under the name C2 but with the pointer position changed. Lines stored in the line store MUST be loaded with a load instruction on a line by itself eg the commands

```
I
L C2
FIN
```

will insert the 3 stored lines into the output file.

13.4 Store Lines From The Command Stream -- SCn/D3

This command stores n lines from the command stream and labels them D3. The lines stored may be edit commands or additions to the input file.

Example.

(a) Store 3 lines from the command stream and label them D3 -- SC3/D3. The command seguence is:

```
SC3/D3
SUBROUTINE PTOUT(C,D)
INTEGER C
LOGICAL D
next edit command.
```

Again a load instruction cn a line by itself MUST be used when loading lines stored by a Sn/ instruction.

Example.

(b) Store 4 commands and label them ISC4 -- SC4/ISC4. The command sequence is

```
SC4/ISC4
F/FORMAT
CB. (
I. '
N
next edit command
```

This stores the 4 commands following the store command - these commands are not executed at this time but are called by a load instruction in the command stream. Eg

```
CA. FORMAT
```

```
L ISC4
CB. 88
L ISC4
```

will cause the 4 commands to be loaded into the command stream after each copy instruction.

13.5 Notes On Store And Load Instructions

In any one run of the editor the maximum number of named character stores is 10 as is the maximum number of named line stores.

Up to 50 lines may be stored in the line store assuming that the total number of characters in these lines does not exceed 500.

If tab stops are set the tab characters must be included in the correct positions in the stored string and NOT input with the load instruction.

14. LINE AND CHARACTER DELIMITER COMMANDS

These instructions are available to change the delimiters for line and character commands.

To change the character command delimiter from a dot the command CHAR immediately(no space) followed by the new delimiter is input. Similiarly to change the line delimiter from a slash, LINE immediately followed by the new delimiter is input

Example

CHAR£ - character delimiter now a £ sign
LINE¢ - line delimiter now a ¢ sign

15. LOOPING COMMANDS -- LOOP N, LOOPEND

These instructions enable all statements between LOOP and LOOPEND to be executed n times. If n is omitted, looping continues until the end of file marker is reached on the input file.

The area reserved for looping statements allows 10 statements per loop but loops may be nested up to 16 deep and there is no limit to the number of unnested loops in any one run.

There are several commands forbidden within a loop. These are

```
FIELD
EXIT
LINE
CANCEL
CHAR
T
EX
L
```

If an error occurs during the execution of a loop the loop is exited and execution of the commands within the loop ceases.

Examples

```
LOOP 5
CA.INTEGER
I.A,B
LOOPEND
```

This is an example of a simple loop which will repeat the commands within it 5 times.

```
LOOP 4
F/INTEGER
LOOP 3
CB.A
R.B,C,D
IB3.
LOOPEND
CB.INPUT
LOOPEND
```

This is an example of a nested loop. The inner statements will be repeated 12 times in all, the outer ones, 4 times.

16. THE EXCHANGE COMMAND --- EX/<STRING1>/<STRING2>/

This command exchanges each occurrence of string 1 with string 2 throughout the file.

When an EX command is input, the editor stores the given strings and as editing of the file progresses, string 1 is searched for. When found a DA.<string> followed by a I.<string 2> is carried out and the strings are thus exchanged.

As exchange executes while editing is progressing several exchanges may be in operation at once - up to six exchange commands can be active at any one time.

An EX command can be switched off at any point in the file by the input of NEX/<string1>. This will switch off the exchange of string 1 and string 2 but will leave on any other exchanges.

Example

```
EX/REAL/INTEGER/
CB.OUTPUT
NEX/REAL
```

This would exchange REAL and INTEGER through the file until the string OUTPUT was found. The exchange would then be switched off.

Note that any non alphanumeric character (not necessarily a slash) may be used as a command delimiter in the EX and NEX commands.

COMMANDS AVAILABLE UNDER RAX ONLY.

17. THE DISPLAY COMMAND - DISPLAY

Input of the word DISPLAY will display 12 lines of the file on a 2260 screen, starting from the line containing the position pointer.

These lines may be overtyped on the screen and on pressing "shift and enter" the edited lines will be written to the output file.

This command is not available for use on the CDC terminals.

18. THE CANCEL EDITING COMMAND - CANCEL

Input of the word CANCEL will terminate editing immediately,no editing will be saved and no listing of the output file produced.

19. LISTING OPTION - NOLIST

If no listing of the edited file is required, input of the word NOLIST as a response to the prompt 'IS A FILE LISTING REQUIRED' will suppress the listing of the final output file on the 2260 screen.

EXAMPLES OF COMPLETE EDIT JOBS.

Two examples are given of jobs using the editor. The first uses relatively simple edit commands, the second some of the more sophisticated features of the editor.

INPUT FILE

```
      SUBRCUTINE RANDO(A,N,MR,C,M,Q,K)
      INTEGER A,C,M,N,Q
      DIMENSION MR(Q)
      DO 10 I=1,Q
      NN=A*N+C
      NN=MOD(NN,M)
   10 MR(I)=MOD(N,K)
      RETURN
      END
```

EDIT COMMANDS	COMMENTS
CB.(	copies to before 1st opening bracket
I.M	inserts M ie RANDO->RANDOM
CA.Q,	copies that line until after Q,
R.L	replaces K with L
N	inputs next line
CAL.	Copies that line till after last character
I.,MR(Q)	inserts MR(Q)
N	takes next line
D1/	deletes line & inserts line shown below
I/C SUBROUTINE TO GENERATE RANDOM NUMBERS	
C3/	copies 3 lines
CB.K	copies to before K
R.L	replaces by L
EXIT	writes new file and terminates editing.

OUTPUT FILE

```
      SUBROUTINE RANDOM(A,N,MR,C,M,Q,L)
      INTEGER A,C,M,N,Q,MR(Q)
C  SUBROUTINE TO GENERATE RANDOM NUMBERS
      DO 10 I=1,Q
      NN=A*N+C
      N=MOD(NN,M)
   10 MR(I)=MOD(N,L)
      RETURN
      END
```

This is an example of an actual edit job that was carried out on one of the editor I/O routines. For simplicity the routine has been slightly truncated.

INPUT FILE

```
      SUBROUTINE ERRPRN(CARD,NO)
      INTEGER*2 NO
      DIMENSION CARD(20),STRING(10)
      WRITE(6,98) CARD
   98 FORMAT(' ',20A4)
```

```
      GO TO (1,2,3,4,5)NO
    1 WRITE(6,101)
      GO TO 99
    2 WRITE(6,102)
      GO TO  99
    3 WRITE(6,103)
      GO TO 99
    4 WRITE(6,104)
      GO TO 99
    5 WRITE(6,105)
   99 RETURN
  101 FORMAT(' TOO MANY CHARACTERS')
  102 FORMAT(' COMMAND NOT VALID')
  103 FORMAT(' COMMAND SHOULD START WITH A LETTER')
  104 FORMAT(' INCORRECT SPELLING')
  105 FORMAT(' WRONG TERMINAL CHARACTER')
      END
```

```
EDIT COMMANDS          COMMENTS
CA.  20)               copies to after 20)
I.,STR1(10,7)          inserts given string
CA.  A4)               copies to after A4
DB/99 RET              deletes to before 99 RET- 10 lines of file.
                       Note the OPP is still set after A4
T £,7                  set tab stop to col 7
I
£DO 10 M=1,10          inserts three lines into file
10 STRING(M) = STR1(M,NO)
£WRITE( 6,STRING)
FIN
CB.101                 copies to before 101
LOOP                   loops till end-of-file
DA.FORMAT              deletes until after FORMAT
C2.                    Copies 2 characters and inserts a '.
I.'
CB.')                  copies rest of line to before ')
D2.                    Deletes ')
START.                 Returns pointer to start of line
C37.                   Copies 37 characters
I.")                   Inserts ")
N                      takes next line
LOOPEND
START/                 returns to beginning of the file
SC3/A1                 stores next 3 lines of command
                       stream as A1
£DATA STR1/            note tab character
(" EOF REACHED                    ")
(" FIND MUST BE A LINE COMMAND    ")
CB/99 RETURN           copies file to before 99 RETURN
D1.                    Deletes that line
I
L A1                   inserts stored lines
FIN
LOOP 7
CB.(                   inserts a ' before an
I.'                    opening bracket and after
CA.)                   A closing bracket, 7
I.'                    times
```

```
LOOPEND
I./                 inserts a / after last statement
START/
CA.  DATA STR1/     copies till after DATA STR1/
LOOP 7
IB5.                Inserts continuation
I.1                 character in col 6 and
CAL.                Blanks in first 5 columns
N
LOOPEND
CBL/                copies to before last line
I/$RETURN           inserts RETURN
EXIT                writes edited file
```

OUTPUT_FILE

```
      SUBROUTINE ERRPRN (CARD,NO)
      INTEGER*2 NC
      DIMENSION CARD(20),STR1(10,7),STRING(10)
      WRITE(6,98)CARD
   98 FORMAT(11,20A4)
      DO 10 M=1,10
   10 STRING(M)=STR1(M,N/)
      WRITE(L,STRING)
      DATA STR1/
     1'(" ECF REACHED                              ")'
     1'(" FIND MUST BE A LINE COMMAND              ")'
     1'(" TOO MANY CHARACTERS                      ")'
     1'(" COMMAND SHOULD START WITH A LETTER       ")'
     1'(" COMMAND NOT VALID                        ")'
     1'(" INCORRECT SPELLING                       ")'
     1'(" WRONG TERMINAL CHARACTER                 ")/
      RETURN
      END
```

Note that the file was edited in 3 stages with a return to the beginning of the file (START/) before each. This is inefficient in time and if a large file is being edited not particularly desirable but for small files breaking the editing into stages reduces the probability of user error.

USING THE EDITOR UNDER 44NFT

The editor can be used to edit disc or tape files under 44NFT.

The edit commands are input, one per card, using the following JCL.

```
//jobname  JOB  ,account-info name/dept.
//SYS002   ACCESS  dsname1,volid
//SYS003   ACCESS  dsname2,volid
//            EXEC   CLEDITOR
                    .
                    .
   editor commands
                    .
                    .
/*
/&
```

where dsname1 is the input file and dsname2 is the output file. Where volid is either on

TAPE='tape-serial'
DISK='disk-name' if not catalogued.

If a file is catalogued then ,volid need not be specified:
dsname 1 is the input file
dsname 2 is the output file

The input and output files must have been previously defined using an ALLOC statement, if they are direct access files. No tape or disc number is necessary if files are catalogued.

If the file to be edited is larger than 1000 cards a further card must be added to access a large workfile on disc SA46V3. This is:

```
//SYS000 ACCESS CLBGWRKA,DISK='SA46V3'
```

This will allow the editing of files of up to 5000 cards. For larger files a personal workfile should be defined using SYS000. This should be deleted on completion of the edit job.

USING THE EDITOR UNDER RAX

To use the editor under RAX requires the setting up of temporary workfiles on the SYSFIL disc.

The JCL statements are as follows

```
/INPUT
/FILE DISK=(1,name1),VOL=SYSFIL,DISP=(NEW,DELETE)
/FILE DISK=(4,name2),VOL=SYSFIL,DISP=(NEW,DELETE)
/JOB GO
/INCLUDE ISCIO
/INCLUDE ISEDIT
/INCLUDE file to be edited
/END RUN
```

a prompt 'PLEASE ENTER YOUR EDIT COMMANDS' followed by 'ENTER DATA' will appear on the 2260 screen. The edit commands should then be entered.

The edited file may be saved after completion of editing by the system command /SAVE name(lcck),SV. No other system commands (eg /PURGE) should precede this or the edited file will be lost.

The workfiles as defined have a size equivalent to 384 cards. They may be redefined to suit the file size (see RAX guide)

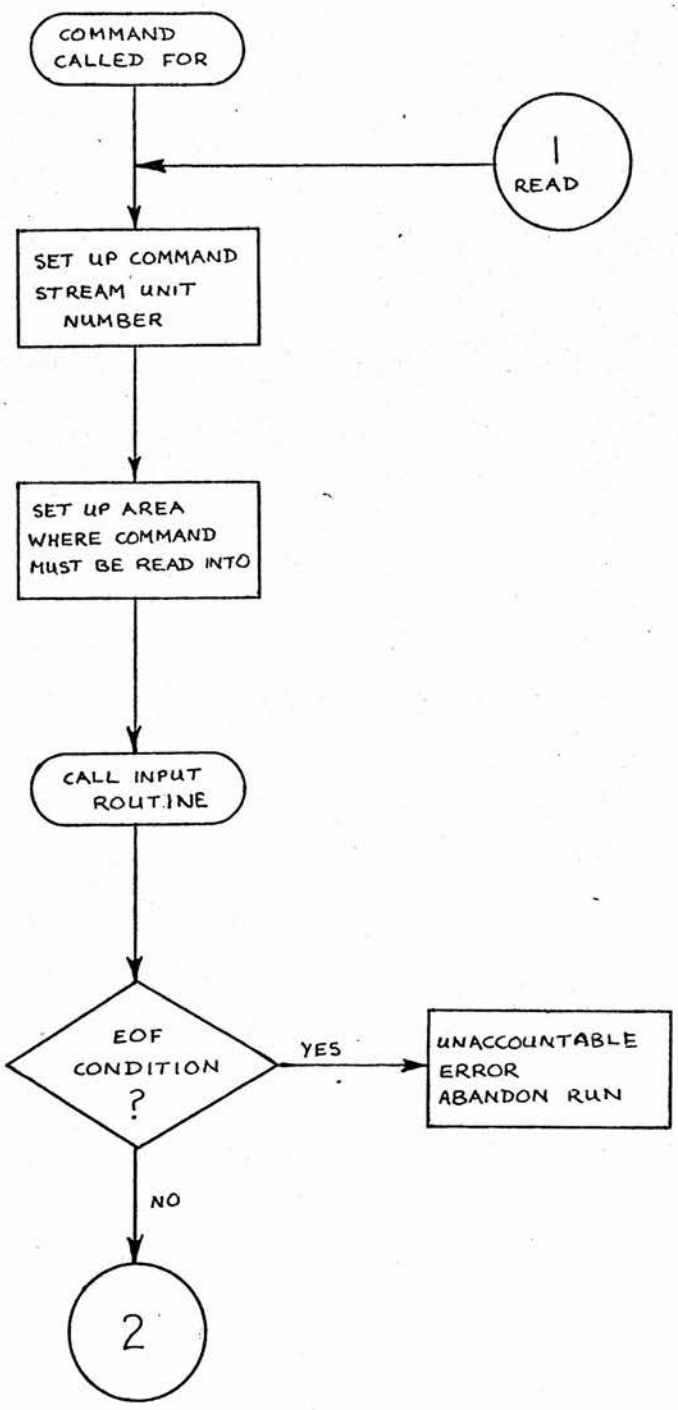
The editor is not available on the RAX background stream.

INDEX OF EDIT COMMANDS

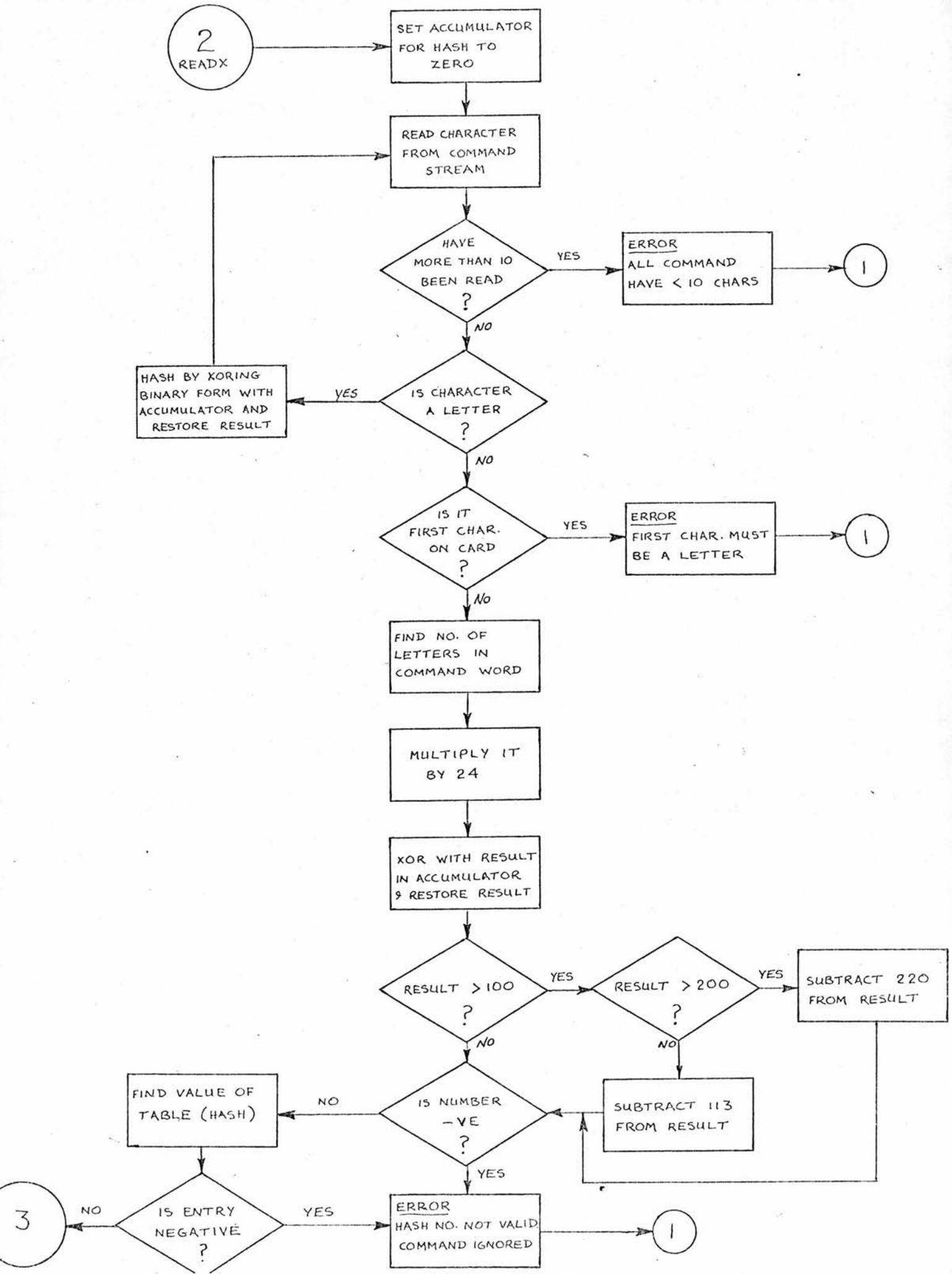
APPENDIX II

FLOWCHARTS FOR COMMAND ANALYSER

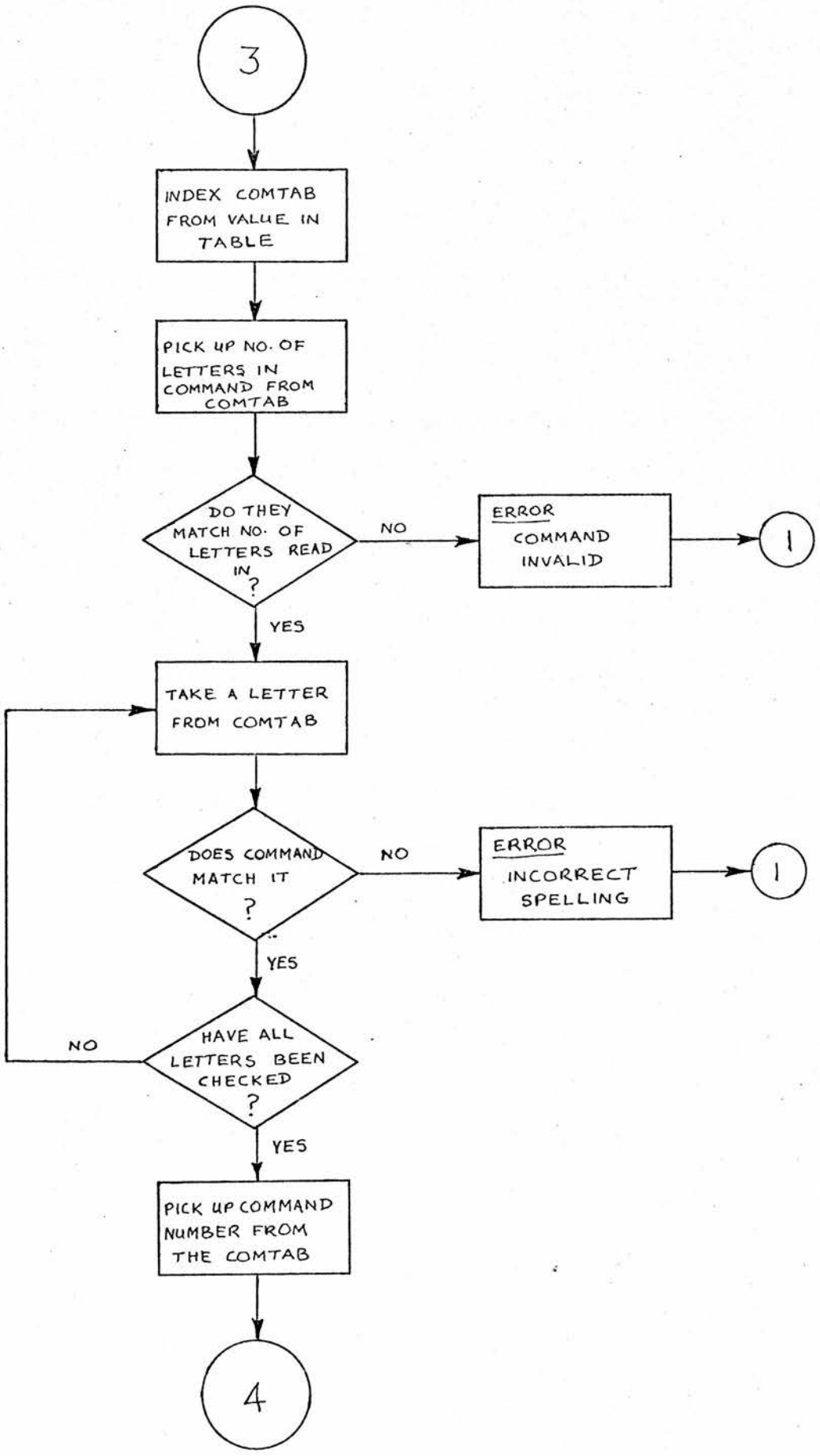
FLOWCHART 1



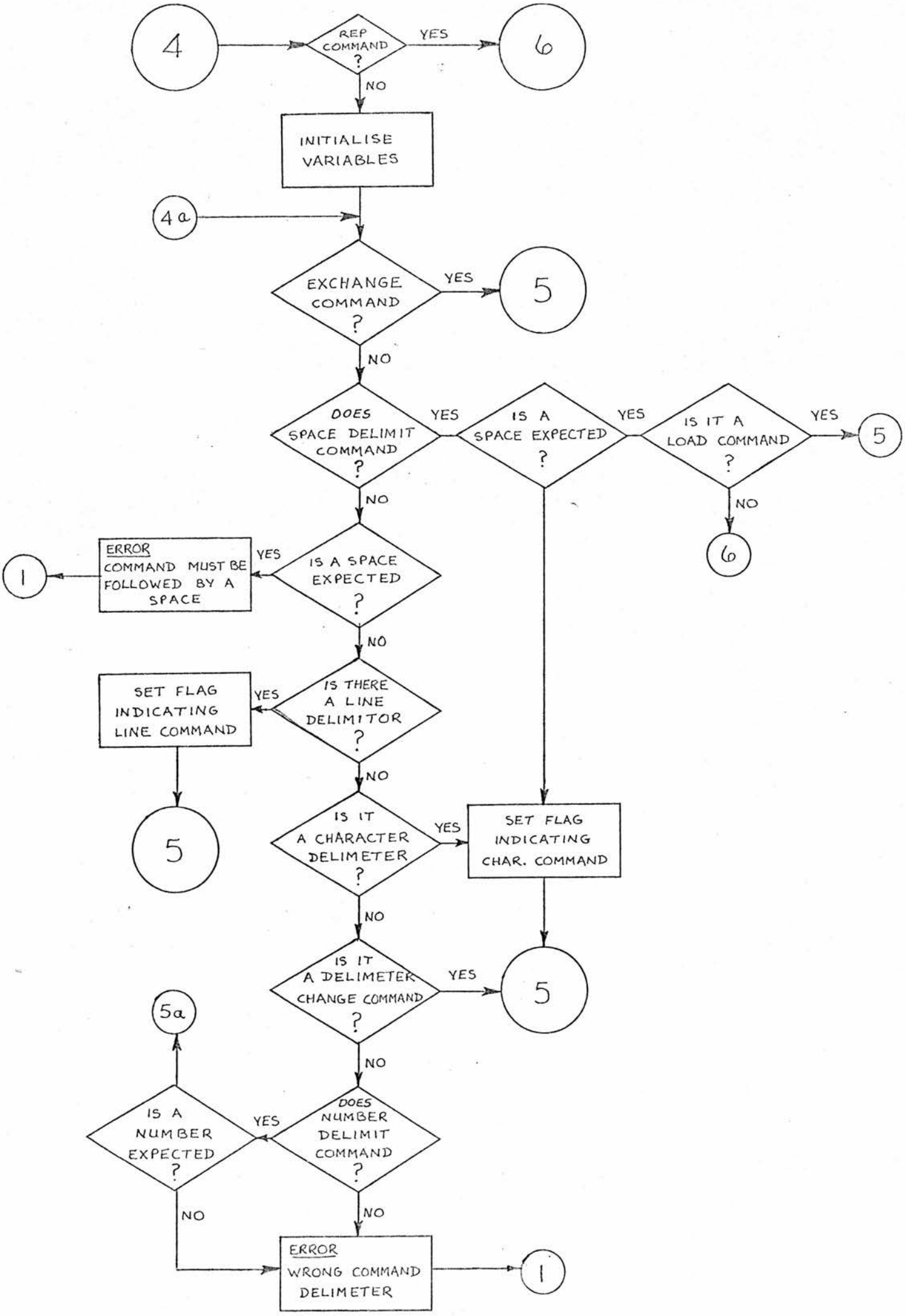
FLOWCHART 2


2
READX
SET ACCUMULATOR FOR HASH TO ZERO
READ CHARACTER FROM COMMAND STREAM
HAVE MORE THAN 10 BEEN READ ?
YES
ERROR
ALL COMMAND HAVE < 10 CHARS
1
NO
IS CHARACTER A LETTER ?
YES
HASH BY XORING BINARY FORM WITH ACCUMULATOR AND RESTORE RESULT
NO
IS IT FIRST CHAR. ON CARD ?
YES
ERROR
FIRST CHAR. MUST BE A LETTER
1
No
FIND NO. OF LETTERS IN COMMAND WORD
MULTIPLY IT BY 24
XOR WITH RESULT IN ACCUMULATOR & RESTORE RESULT
RESULT > 100 ?
YES
RESULT > 200 ?
YES
SUBTRACT 220 FROM RESULT
NO
NO
SUBTRACT 113 FROM RESULT
IS NUMBER -VE ?
NO
FIND VALUE OF TABLE (HASH)
YES
IS ENTRY NEGATIVE ?
NO
3
YES
ERROR
HASH NO. NOT VALID COMMAND IGNORED
1

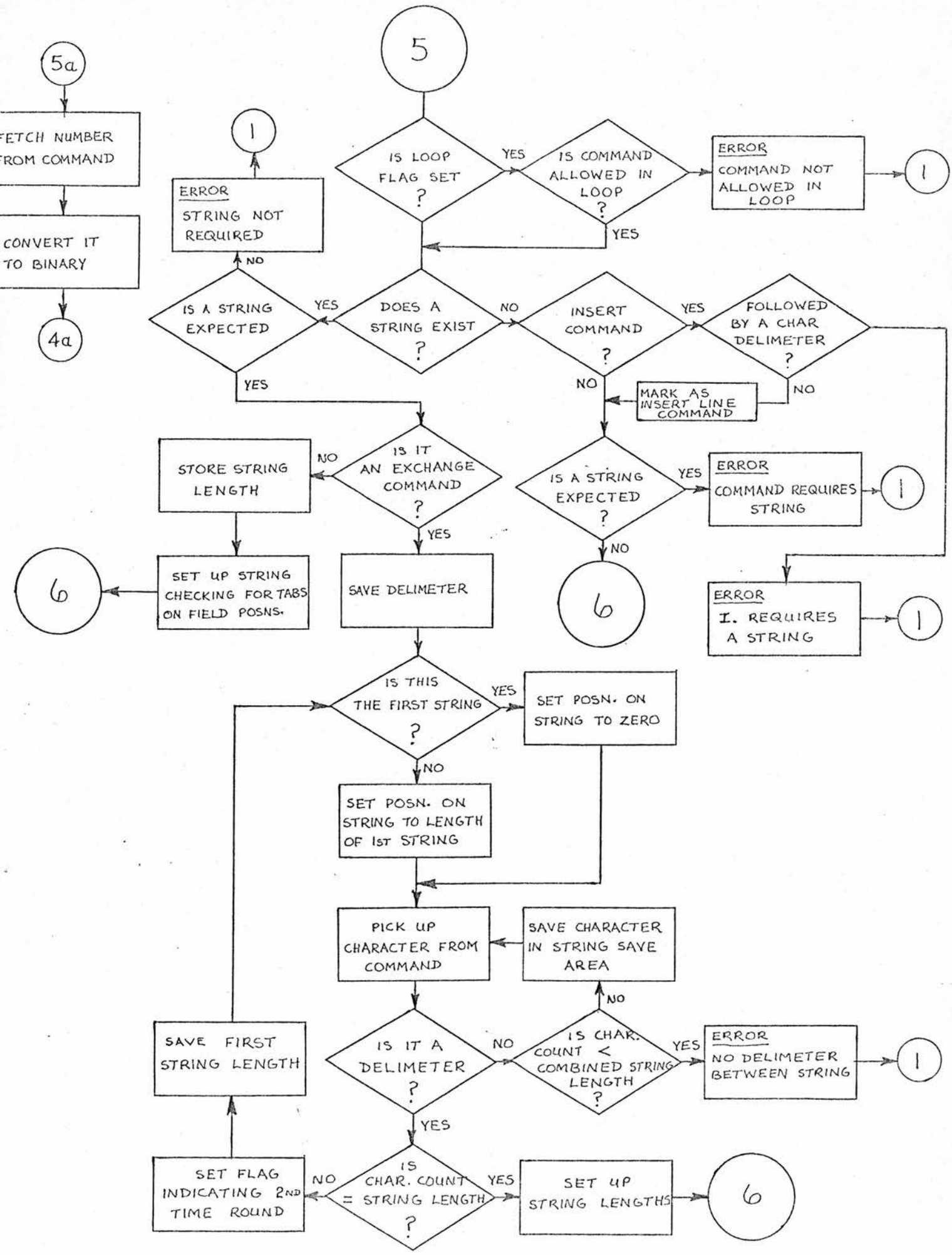
FLOWCHART 3


3
INDEX COMTAB FROM VALUE IN TABLE
PICK UP NO. OF LETTERS IN COMMAND FROM COMTAB
DO THEY MATCH NO. OF LETTERS READ IN ?
NO
ERROR COMMAND INVALID
1
YES
TAKE A LETTER FROM COMTAB
DOES COMMAND MATCH IT ?
NO
ERROR INCORRECT SPELLING
1
YES
HAVE ALL LETTERS BEEN CHECKED ?
NO
YES
PICK UP COMMAND NUMBER FROM THE COMTAB
4

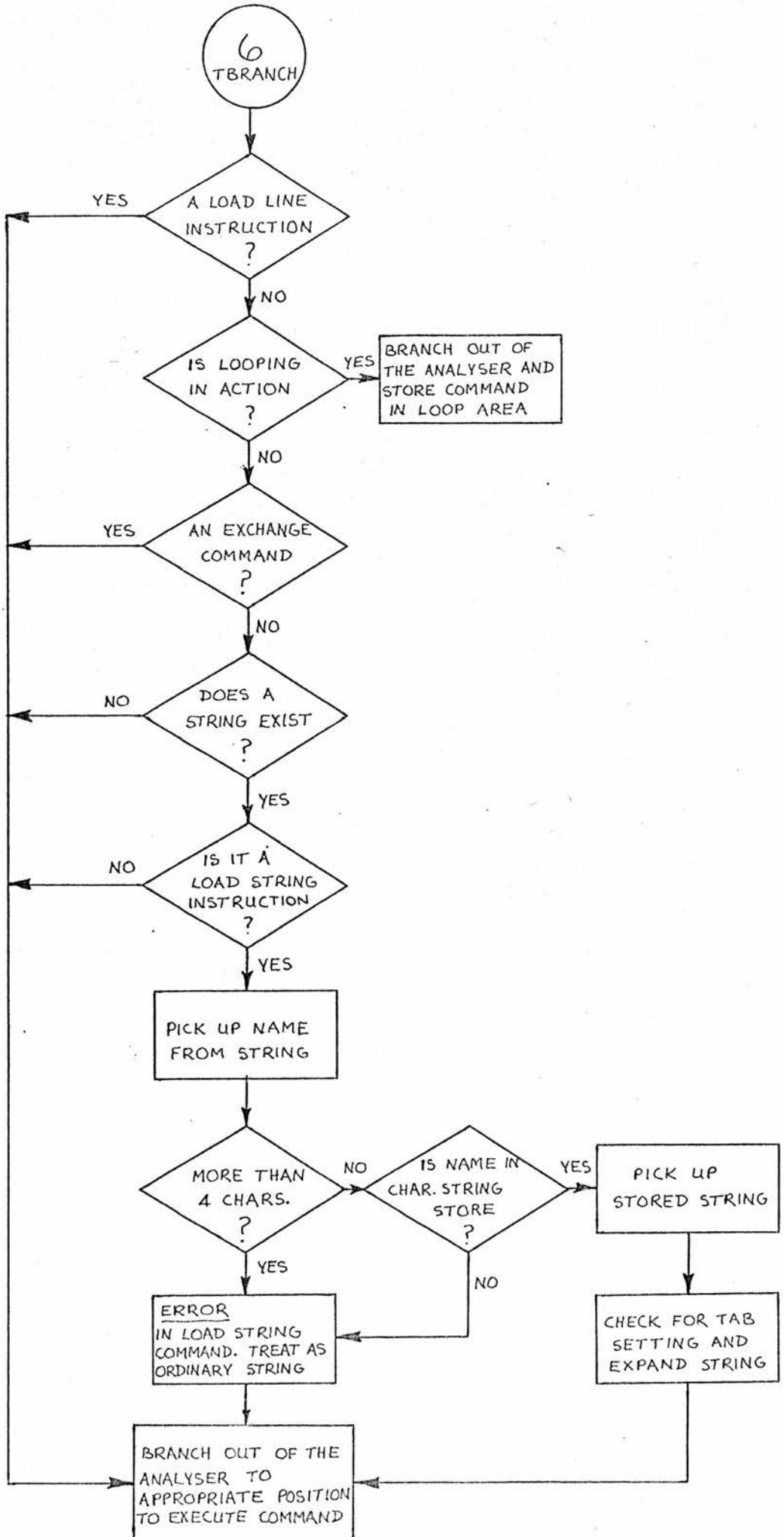
FLOWCHART 4


4
REP COMMAND ?
YES
6
NO
INITIALISE VARIABLES
4a
EXCHANGE COMMAND ?
YES
5
NO
DOES SPACE DELIMIT COMMAND ?
YES
IS A SPACE EXPECTED ?
YES
IS IT A LOAD COMMAND ?
YES
5
NO
6
NO
ERROR COMMAND MUST BE FOLLOWED BY A SPACE
YES
1
IS A SPACE EXPECTED ?
NO
SET FLAG INDICATING LINE COMMAND
YES
IS THERE A LINE DELIMITOR ?
NO
5
IS IT A CHARACTER DELIMETER ?
YES
SET FLAG INDICATING CHAR. COMMAND
NO
IS IT A DELIMETER CHANGE COMMAND ?
YES
5
NO
5a
IS A NUMBER EXPECTED ?
YES
DOES NUMBER DELIMIT COMMAND ?
NO
NO
ERROR WRONG COMMAND DELIMETER
1

FLOWCHART 5

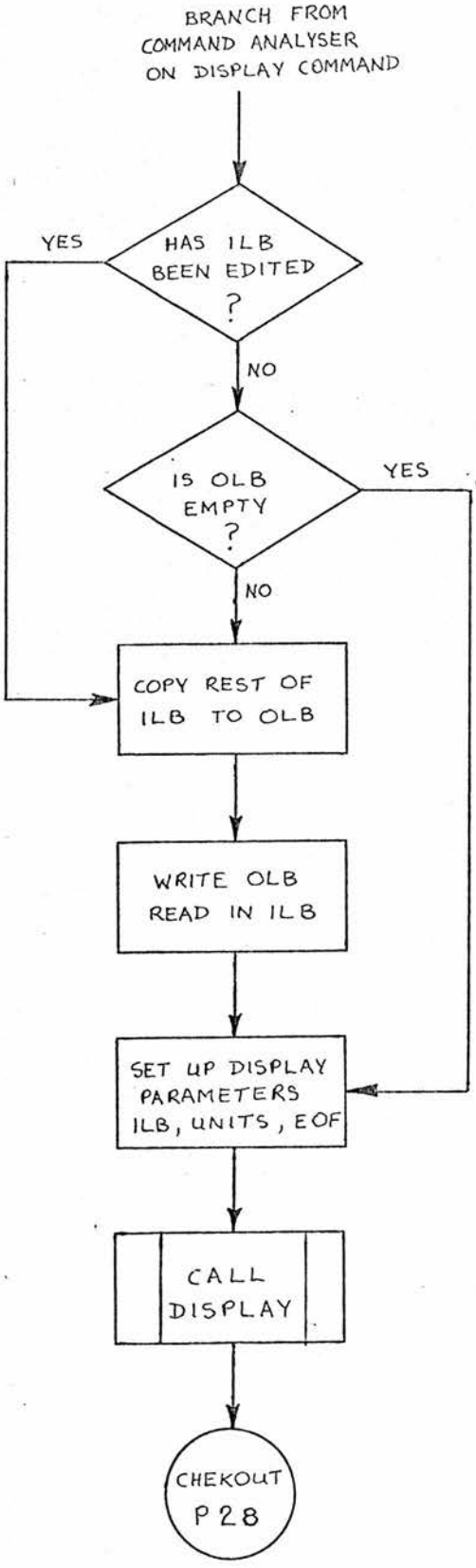

5a
FETCH NUMBER FROM COMMAND
CONVERT IT TO BINARY
4a
5
1
IS LOOP FLAG SET ?
YES
IS COMMAND ALLOWED IN LOOP ?
YES
ERROR
COMMAND NOT ALLOWED IN LOOP
1
ERROR
STRING NOT REQUIRED
NO
IS A STRING EXPECTED
YES
DOES A STRING EXIST ?
NO
INSERT COMMAND ?
YES
FOLLOWED BY A CHAR DELIMETER ?
NO
MARK AS INSERT LINE COMMAND
YES
NO
STORE STRING LENGTH
NO
IS IT AN EXCHANGE COMMAND ?
YES
IS A STRING EXPECTED ?
YES
ERROR
COMMAND REQUIRES STRING
1
NO
6
6
SET UP STRING CHECKING FOR TABS ON FIELD POSNS.
SAVE DELIMETER
ERROR
I. REQUIRES A STRING
1
IS THIS THE FIRST STRING ?
YES
SET POSN. ON STRING TO ZERO
NO
SET POSN. ON STRING TO LENGTH OF 1ST STRING
PICK UP CHARACTER FROM COMMAND
SAVE CHARACTER IN STRING SAVE AREA
NO
SAVE FIRST STRING LENGTH
IS IT A DELIMETER ?
NO
IS CHAR. COUNT < COMBINED STRING LENGTH ?
YES
ERROR
NO DELIMETER BETWEEN STRING
1
YES
SET FLAG INDICATING 2ND TIME ROUND
NO
IS CHAR. COUNT = STRING LENGTH ?
YES
SET UP STRING LENGTHS
6

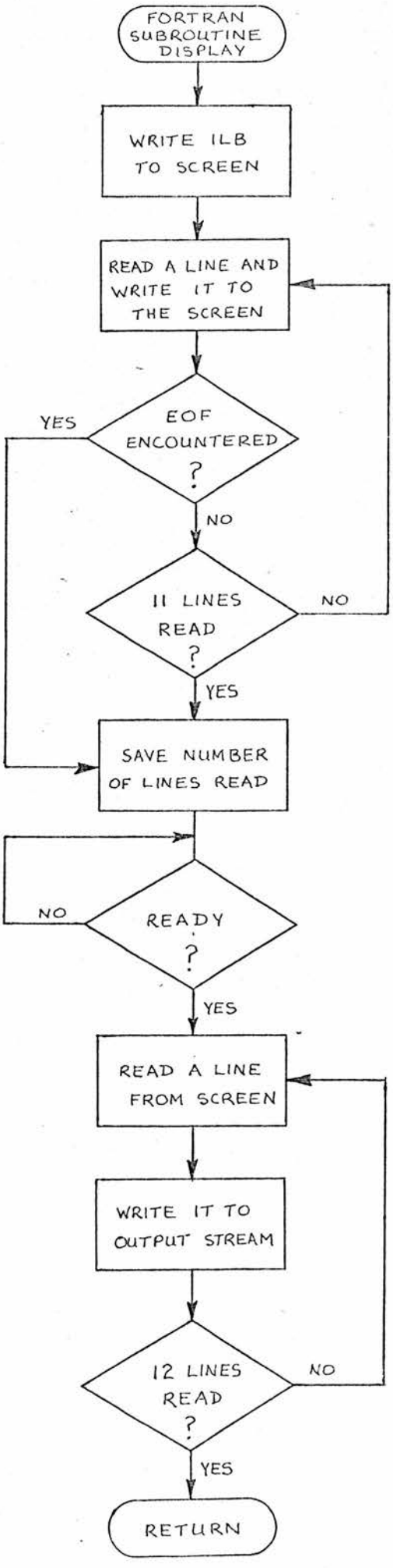
FLOWCHART 6


6
TBRANCH
A LOAD LINE INSTRUCTION ?
YES
NO
IS LOOPING IN ACTION ?
YES
BRANCH OUT OF THE ANALYSER AND STORE COMMAND IN LOOP AREA
NO
AN EXCHANGE COMMAND ?
YES
NO
DOES A STRING EXIST ?
NO
YES
IS IT A LOAD STRING INSTRUCTION ?
NO
YES
PICK UP NAME FROM STRING
MORE THAN 4 CHARS. ?
NO
IS NAME IN CHAR. STRING STORE ?
YES
NO
YES
PICK UP STORED STRING
ERROR
IN LOAD STRING COMMAND. TREAT AS ORDINARY STRING
CHECK FOR TAB SETTING AND EXPAND STRING
BRANCH OUT OF THE ANALYSER TO APPROPRIATE POSITION TO EXECUTE COMMAND

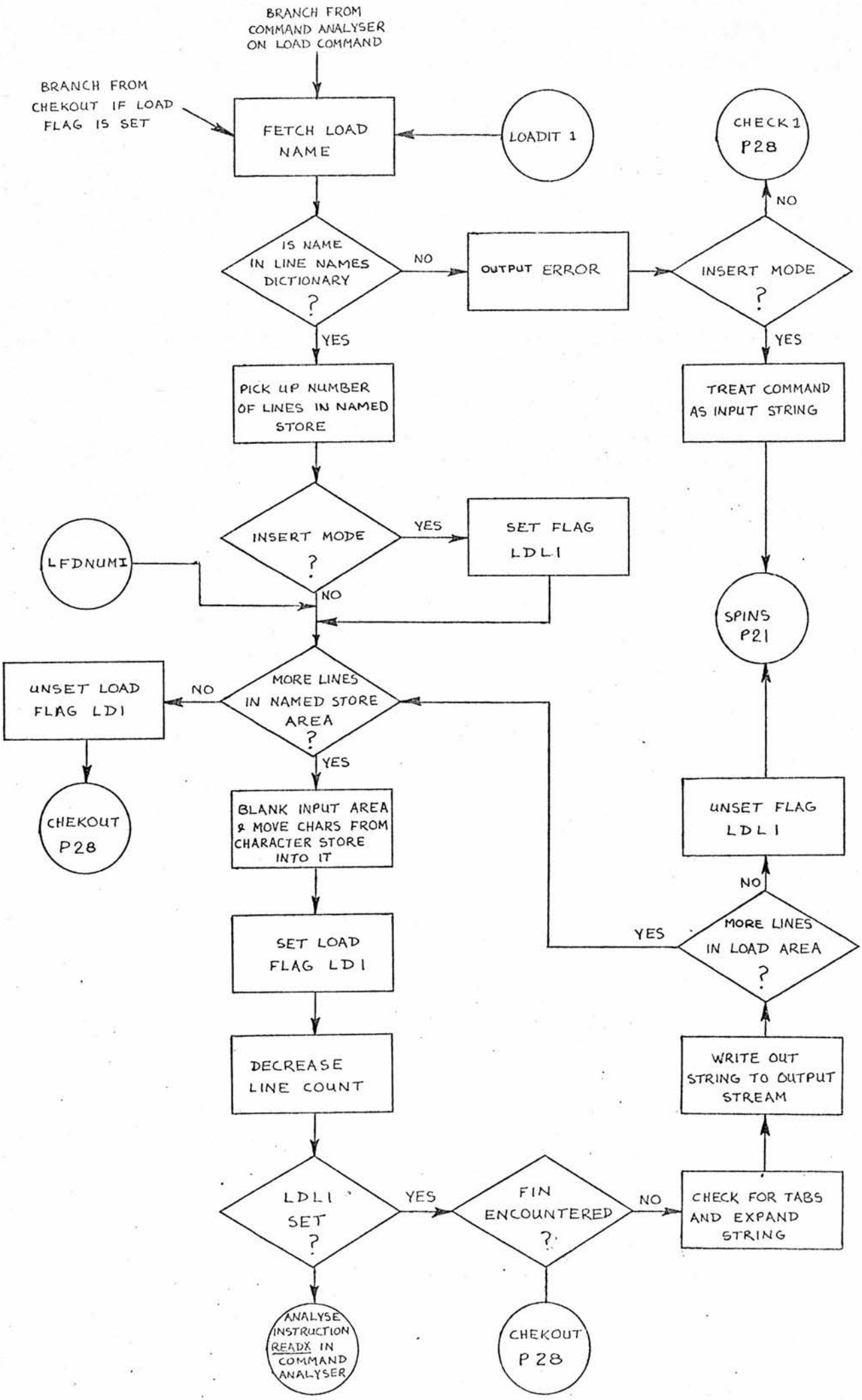
APPENDIX III

FLOWCHARTS FOR CODING OF EDITOR DISPLAY FACILITY

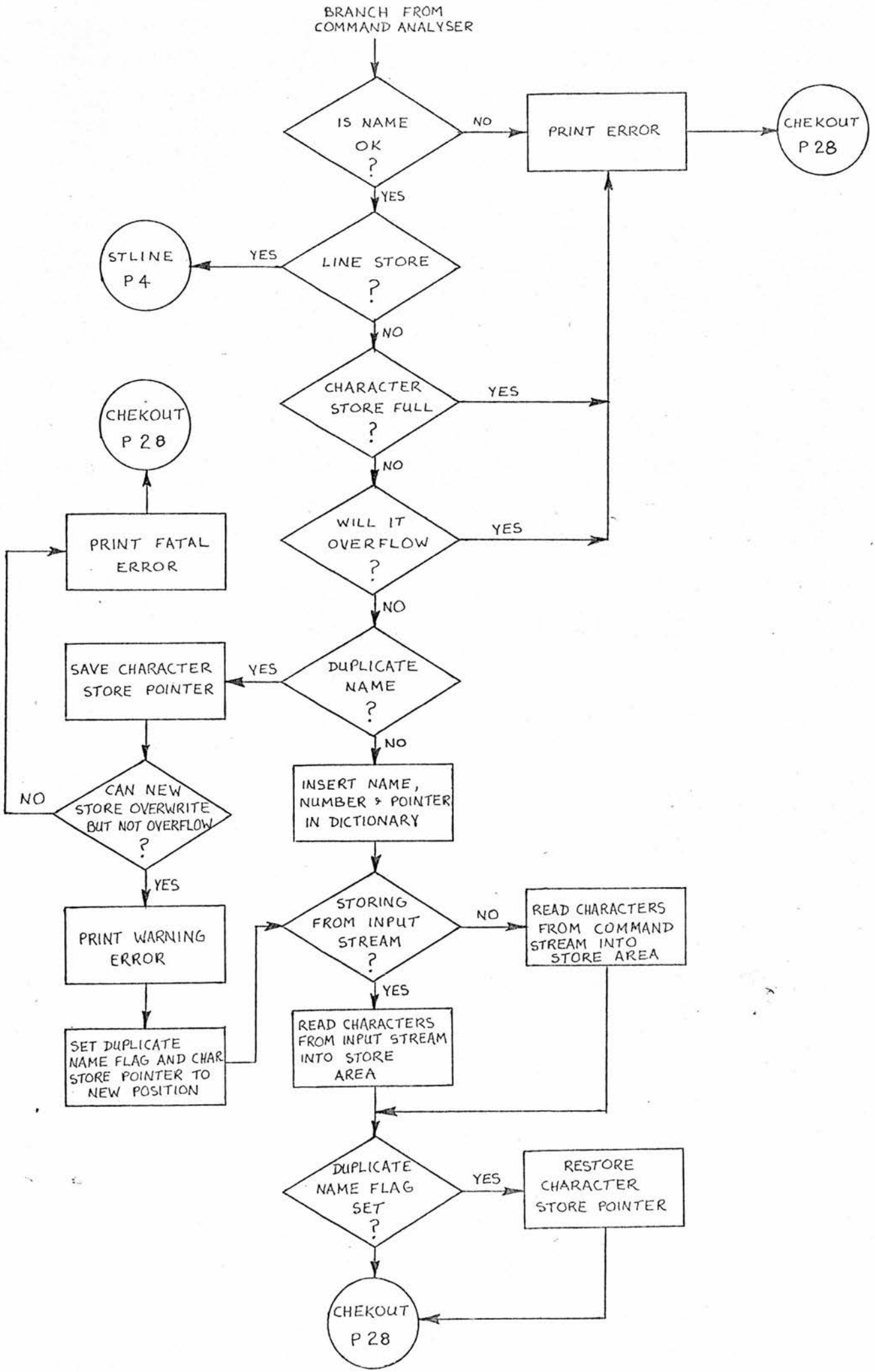




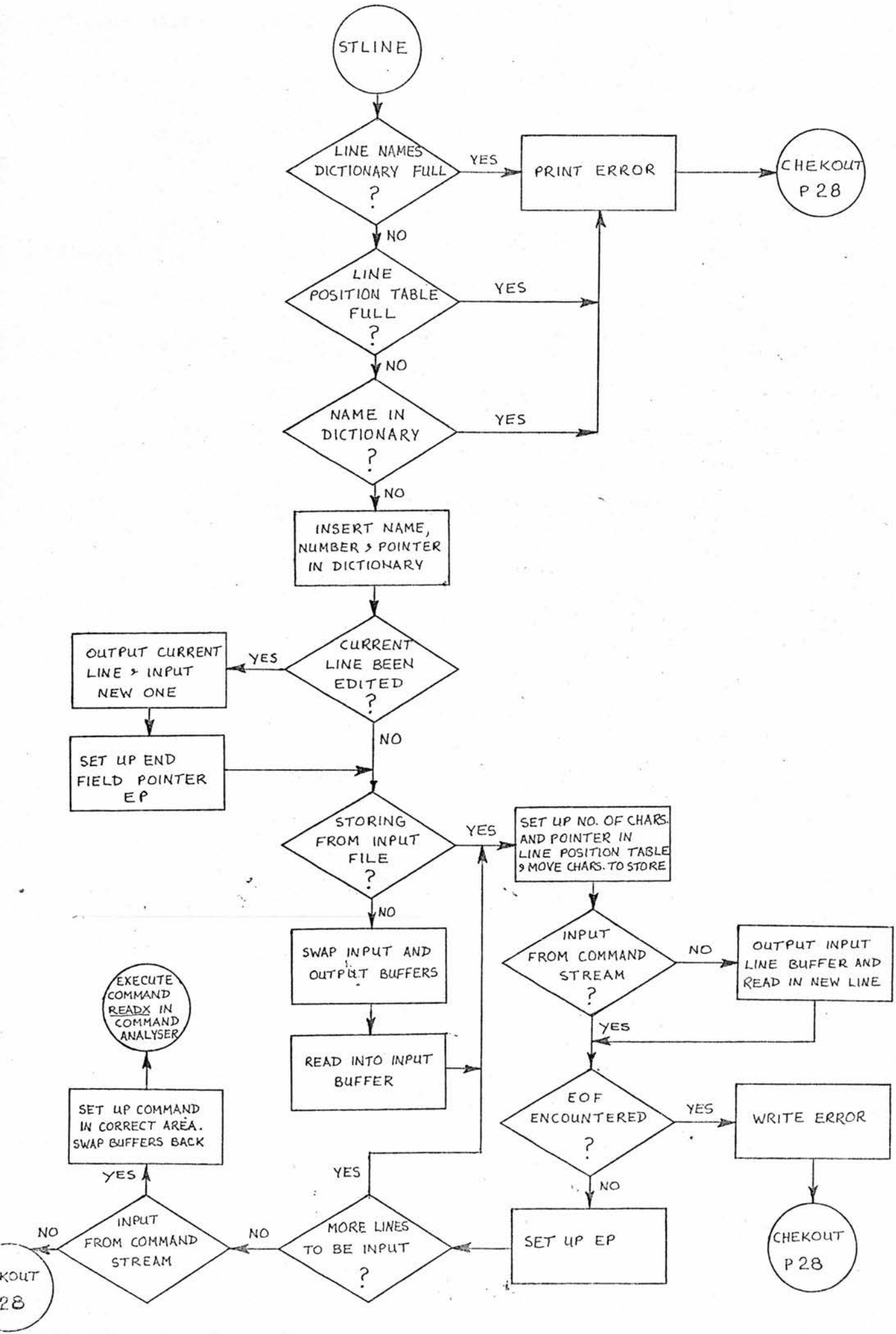
LOADING LINES



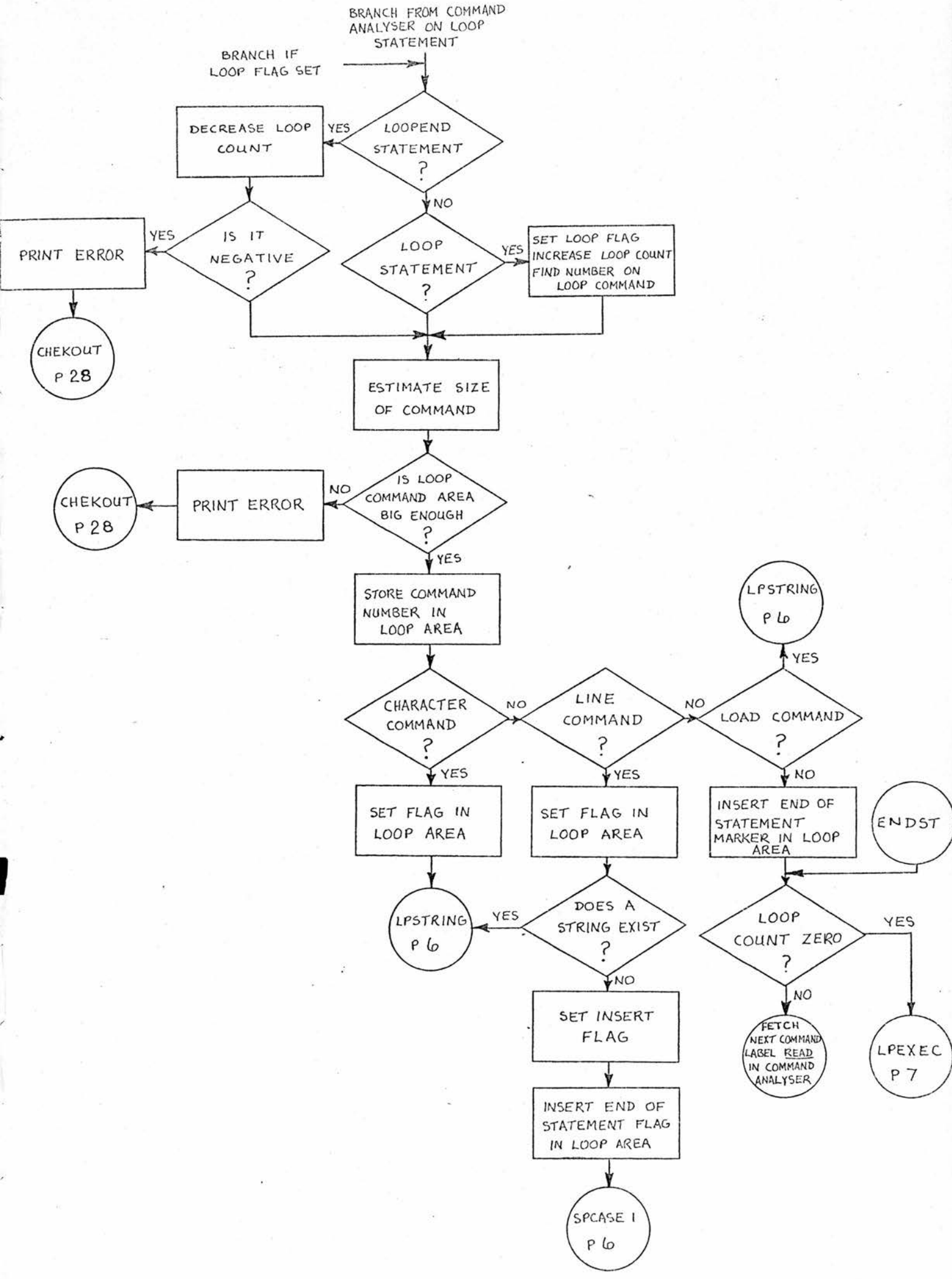
STORING LINES OR CHARACTERS



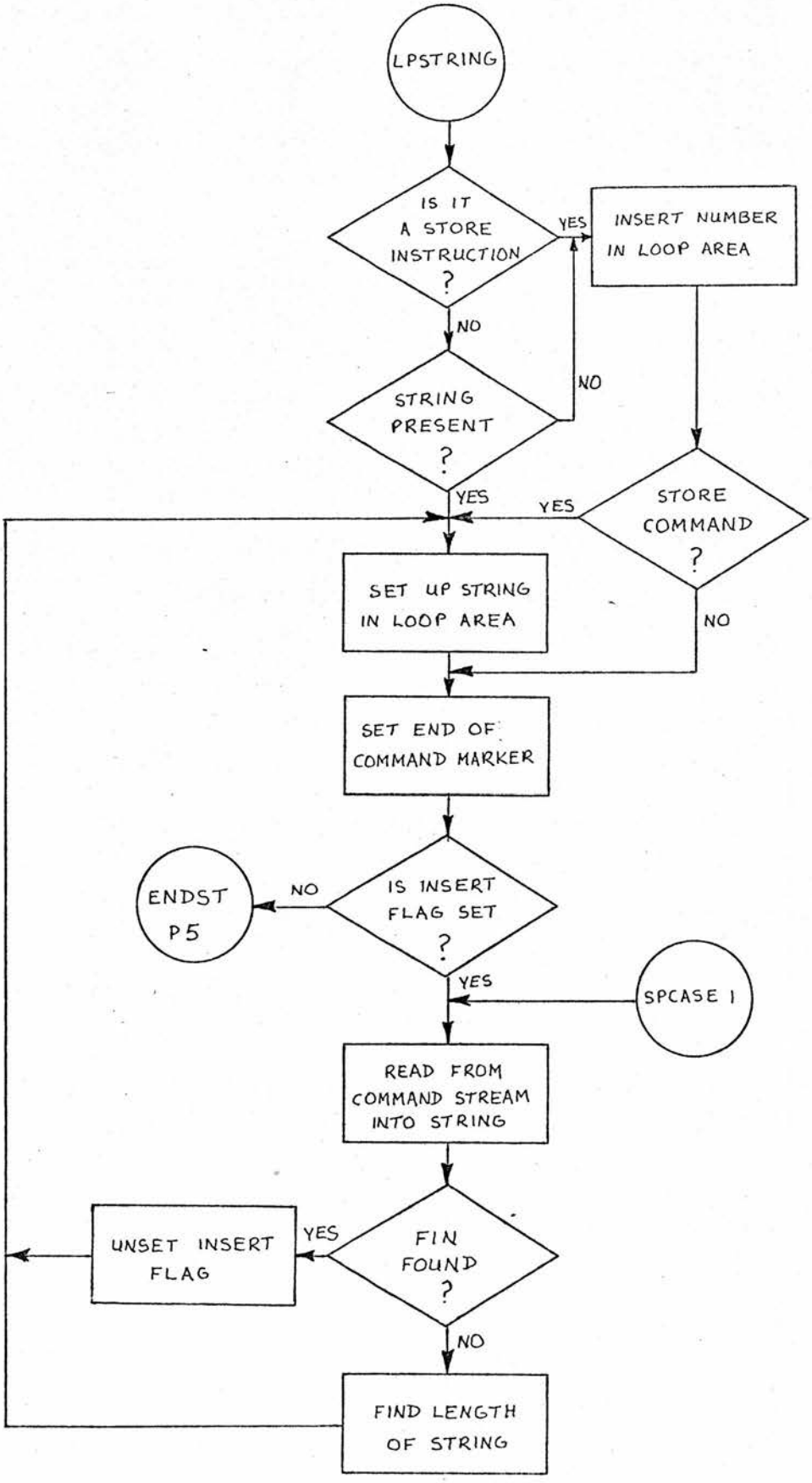
STORING LINES OR CHARACTERS (contd.)

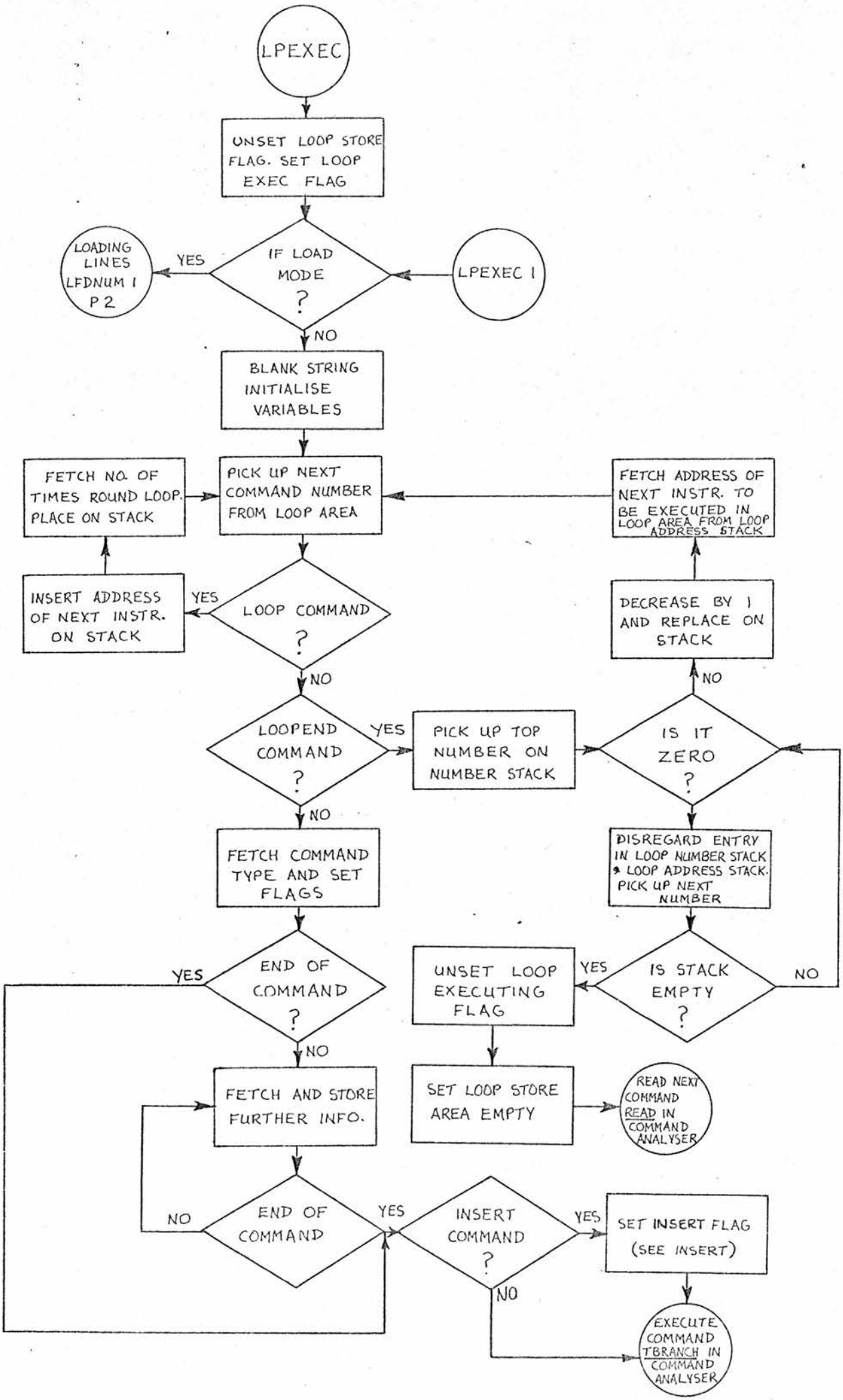


LOOPING FACILITY

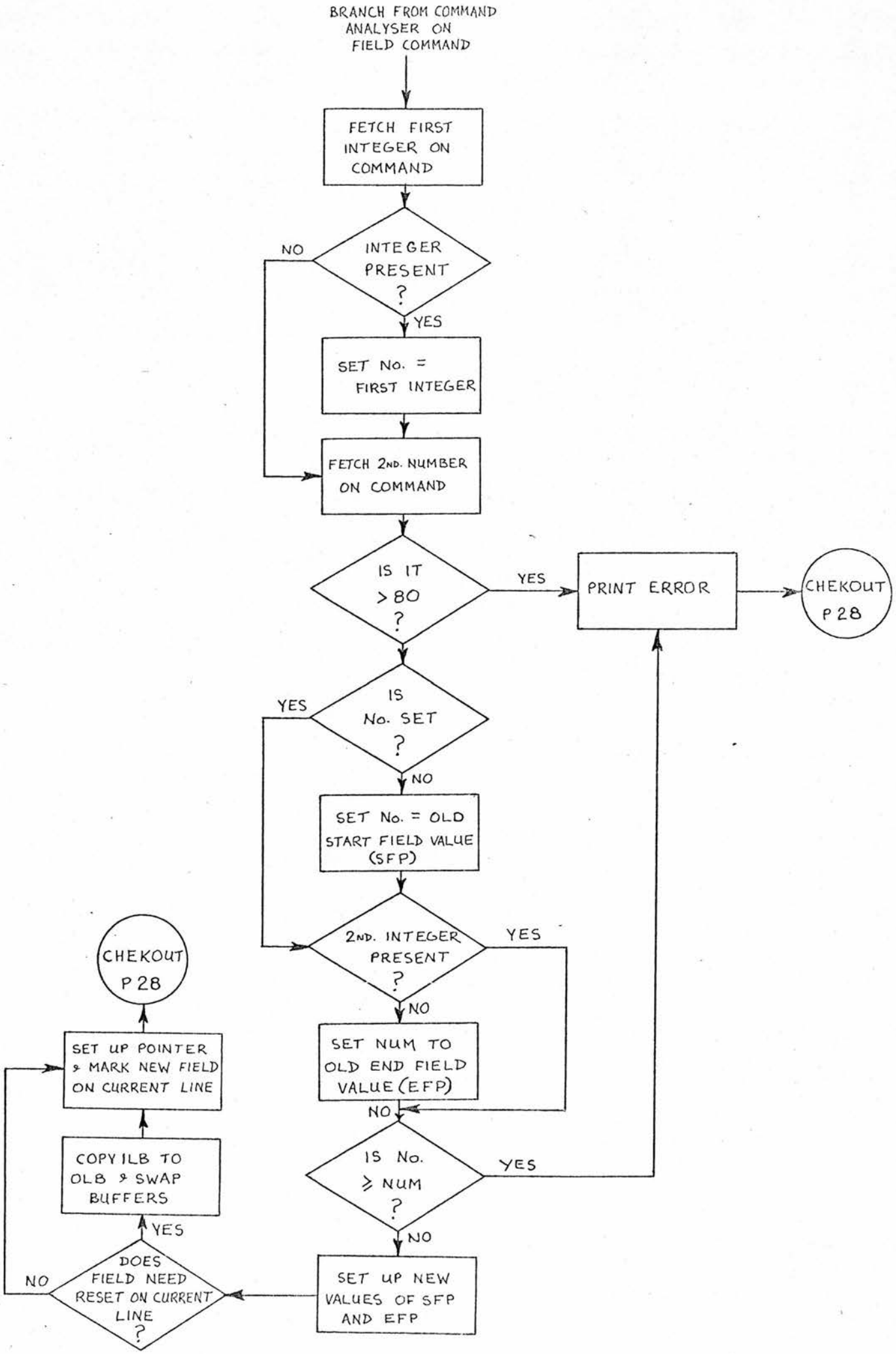


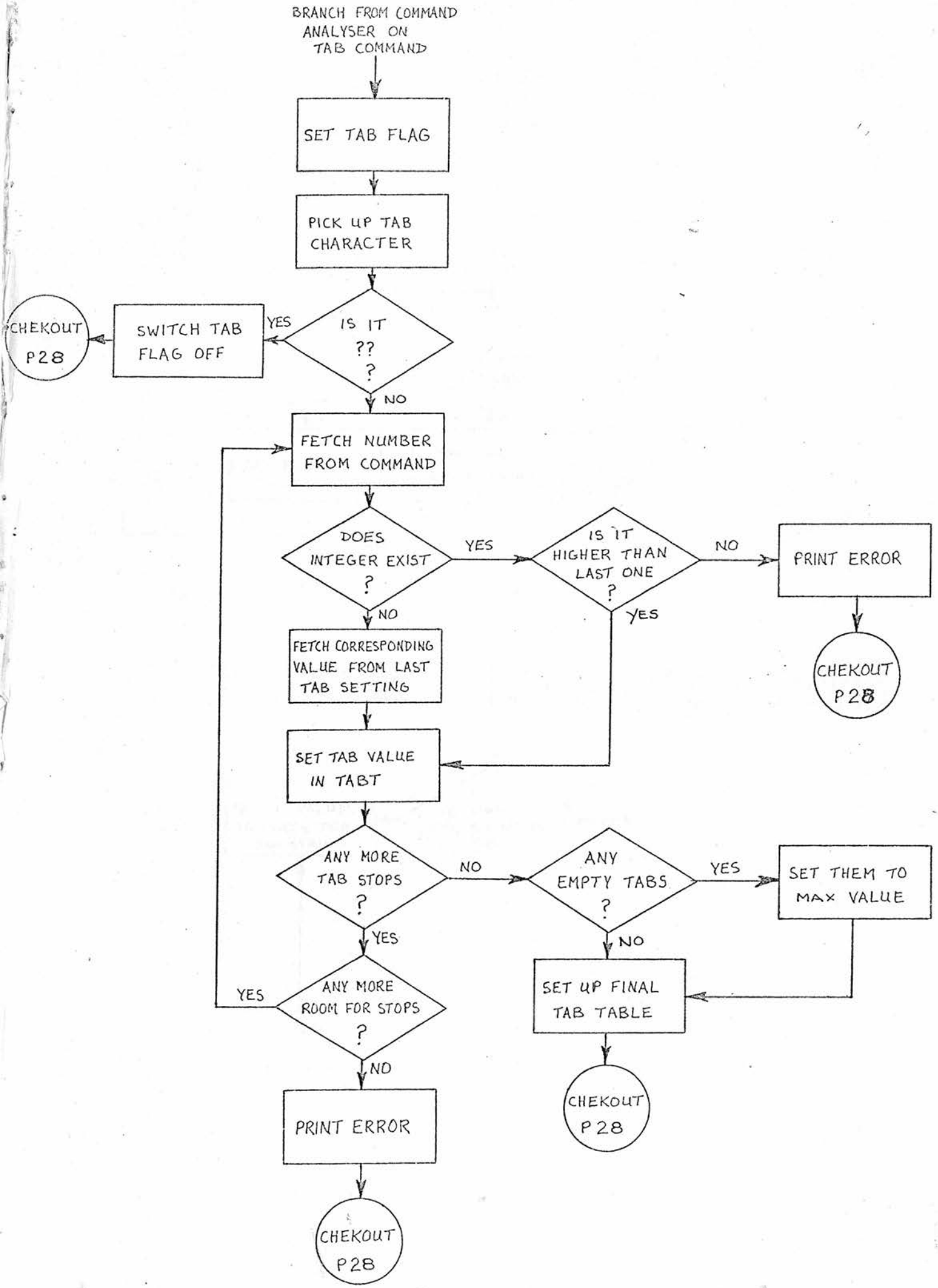
LOOPING FACILITY (contd.)



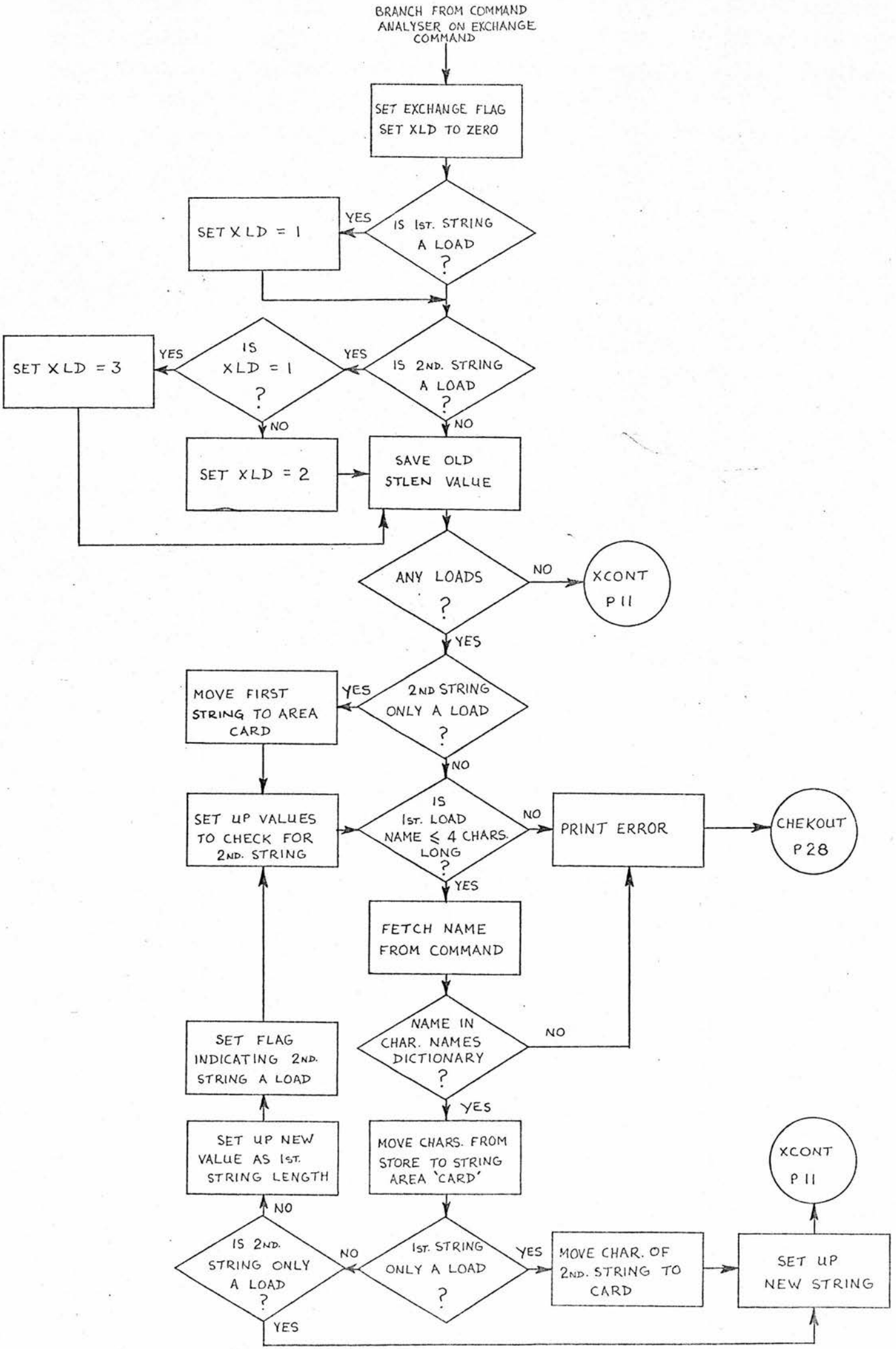

LPEXEC
UNSET LOOP STORE FLAG. SET LOOP EXEC FLAG
IF LOAD MODE ?
YES
LOADING LINES LFDNUM 1 P2
LPEXEC 1
NO
BLANK STRING INITIALISE VARIABLES
PICK UP NEXT COMMAND NUMBER FROM LOOP AREA
FETCH NO. OF TIMES ROUND LOOP. PLACE ON STACK
FETCH ADDRESS OF NEXT INSTR. TO BE EXECUTED IN LOOP AREA FROM LOOP ADDRESS STACK
LOOP COMMAND ?
YES
INSERT ADDRESS OF NEXT INSTR. ON STACK
DECREASE BY 1 AND REPLACE ON STACK
NO
LOOPEND COMMAND ?
YES
PICK UP TOP NUMBER ON NUMBER STACK
IS IT ZERO ?
NO
FETCH COMMAND TYPE AND SET FLAGS
DISREGARD ENTRY IN LOOP NUMBER STACK & LOOP ADDRESS STACK. PICK UP NEXT NUMBER
END OF COMMAND ?
YES
IS STACK EMPTY ?
YES
NO
UNSET LOOP EXECUTING FLAG
NO
FETCH AND STORE FURTHER INFO.
SET LOOP STORE AREA EMPTY
READ NEXT COMMAND READ IN COMMAND ANALYSER
END OF COMMAND
NO
YES
INSERT COMMAND ?
YES
SET INSERT FLAG (SEE INSERT)
NO
EXECUTE COMMAND TBRANCH IN COMMAND ANALYSER

FIELD DEFINING COMMAND

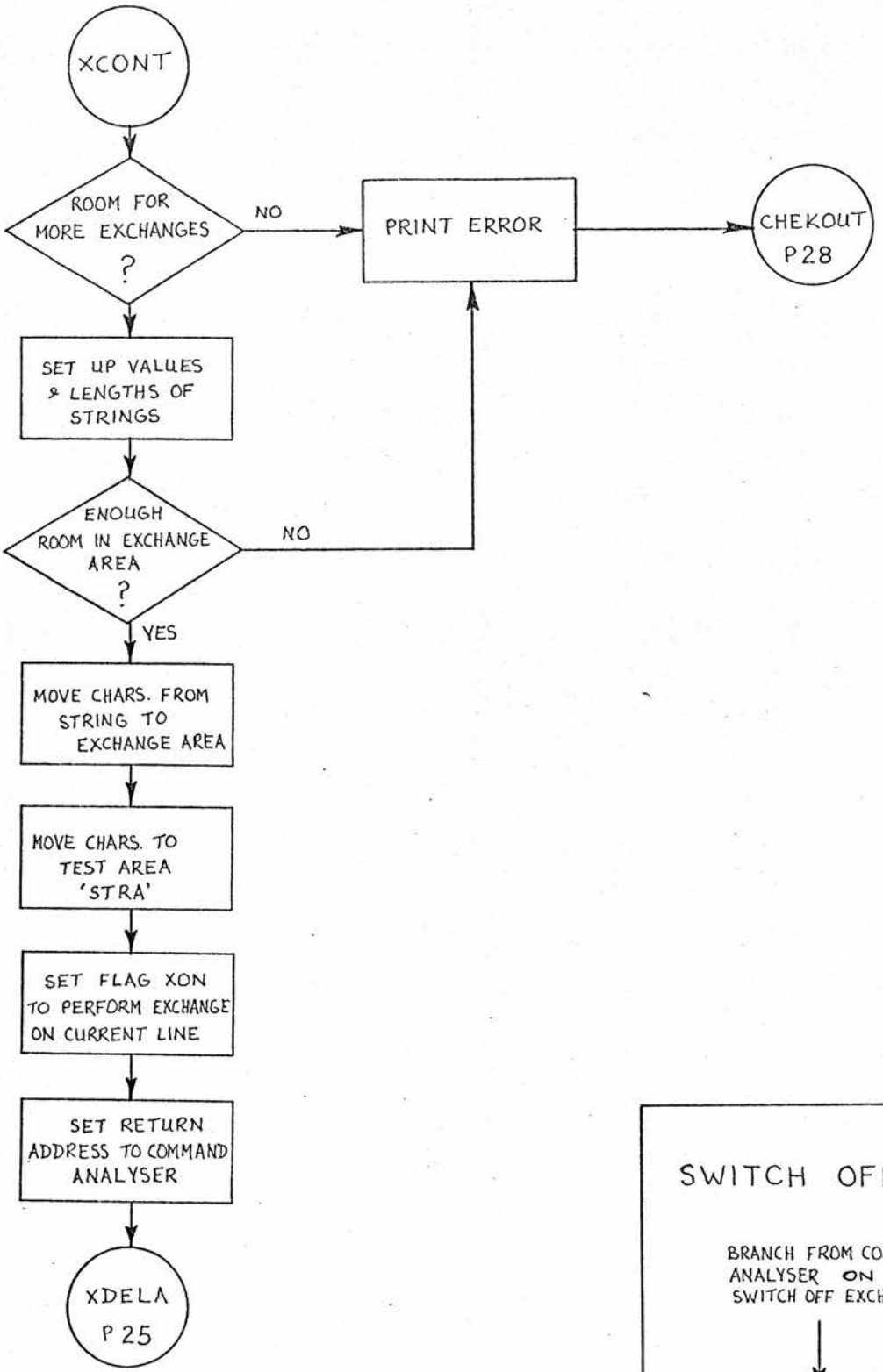



BRANCH FROM COMMAND ANALYSER ON TAB COMMAND
SET TAB FLAG
PICK UP TAB CHARACTER
IS IT ?? ?
YES
SWITCH TAB FLAG OFF
CHEKOUT P28
NO
FETCH NUMBER FROM COMMAND
DOES INTEGER EXIST ?
YES
IS IT HIGHER THAN LAST ONE ?
NO
PRINT ERROR
CHEKOUT P28
NO
YES
FETCH CORRESPONDING VALUE FROM LAST TAB SETTING
SET TAB VALUE IN TABT
ANY MORE TAB STOPS ?
NO
ANY EMPTY TABS ?
YES
SET THEM TO MAX VALUE
YES
NO
ANY MORE ROOM FOR STOPS ?
YES
NO
SET UP FINAL TAB TABLE
CHEKOUT P28
PRINT ERROR
CHEKOUT P28

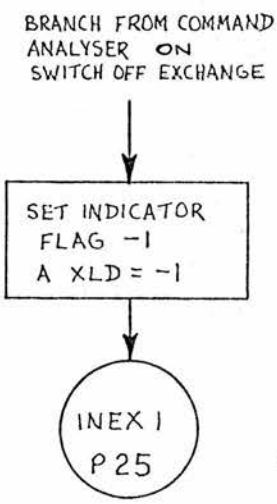
THE EXCHANGE COMMAND



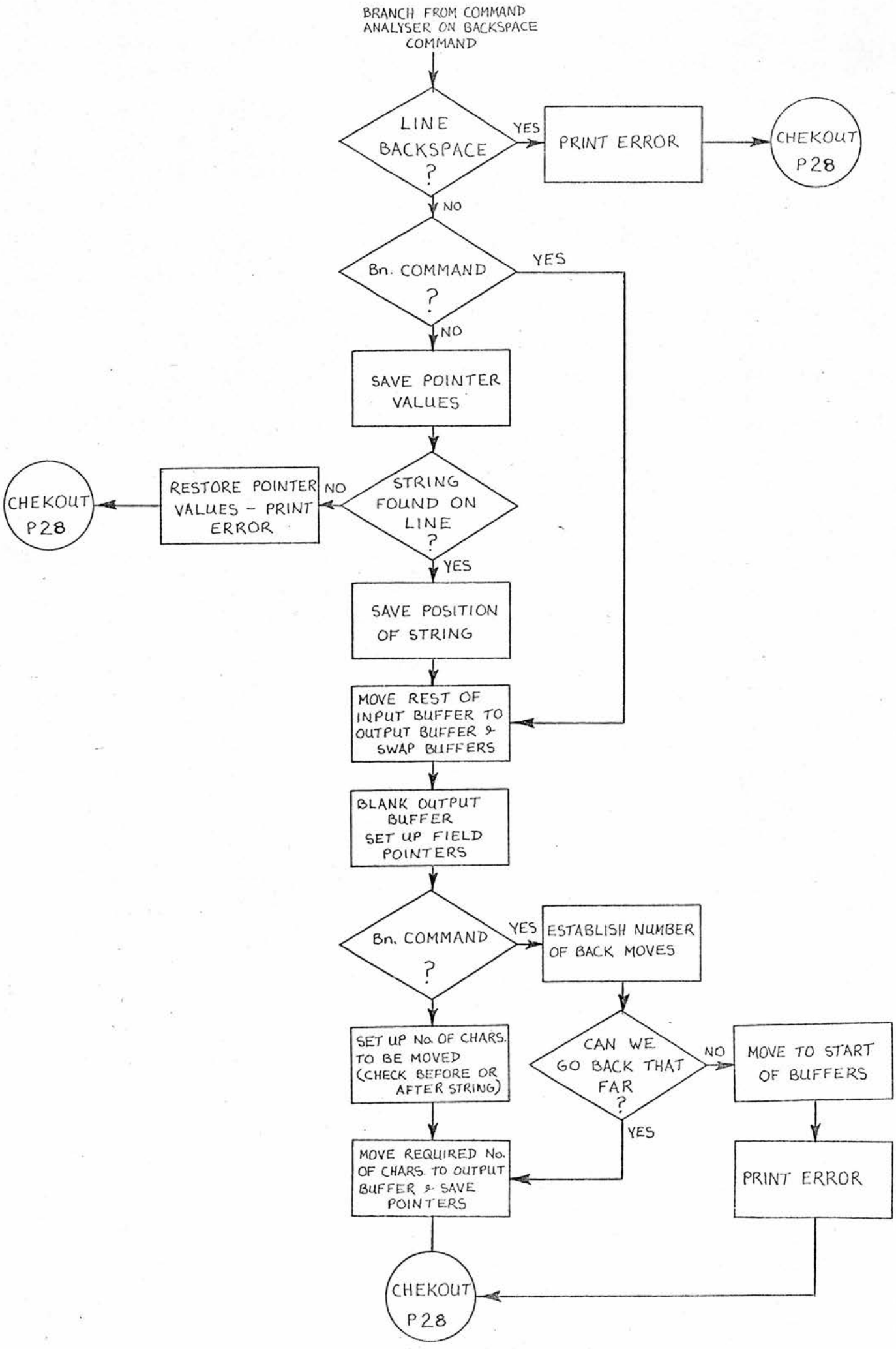
THE EXCHANGE COMMAND (contd.)



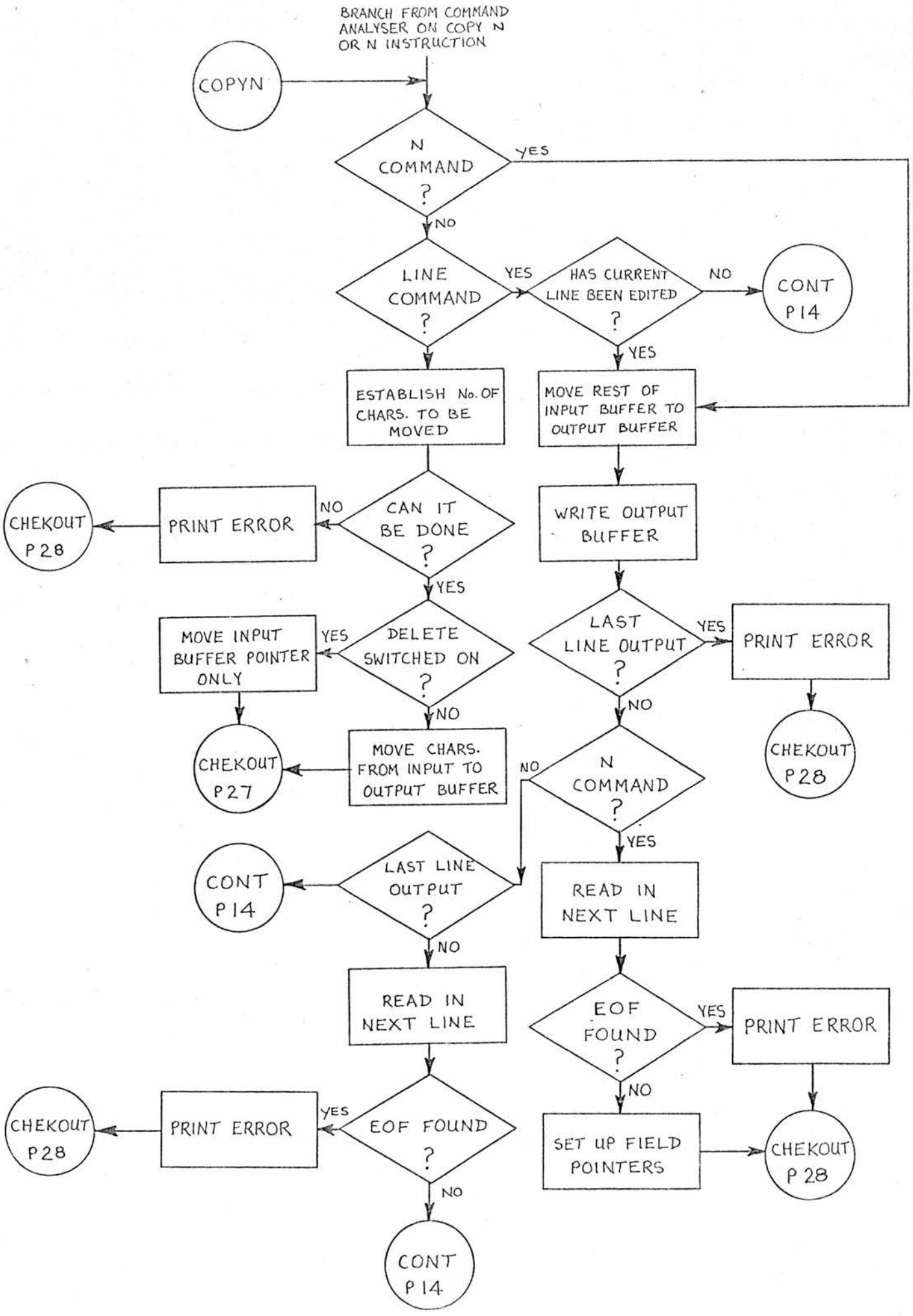
SWITCH OFF EXCHANGE



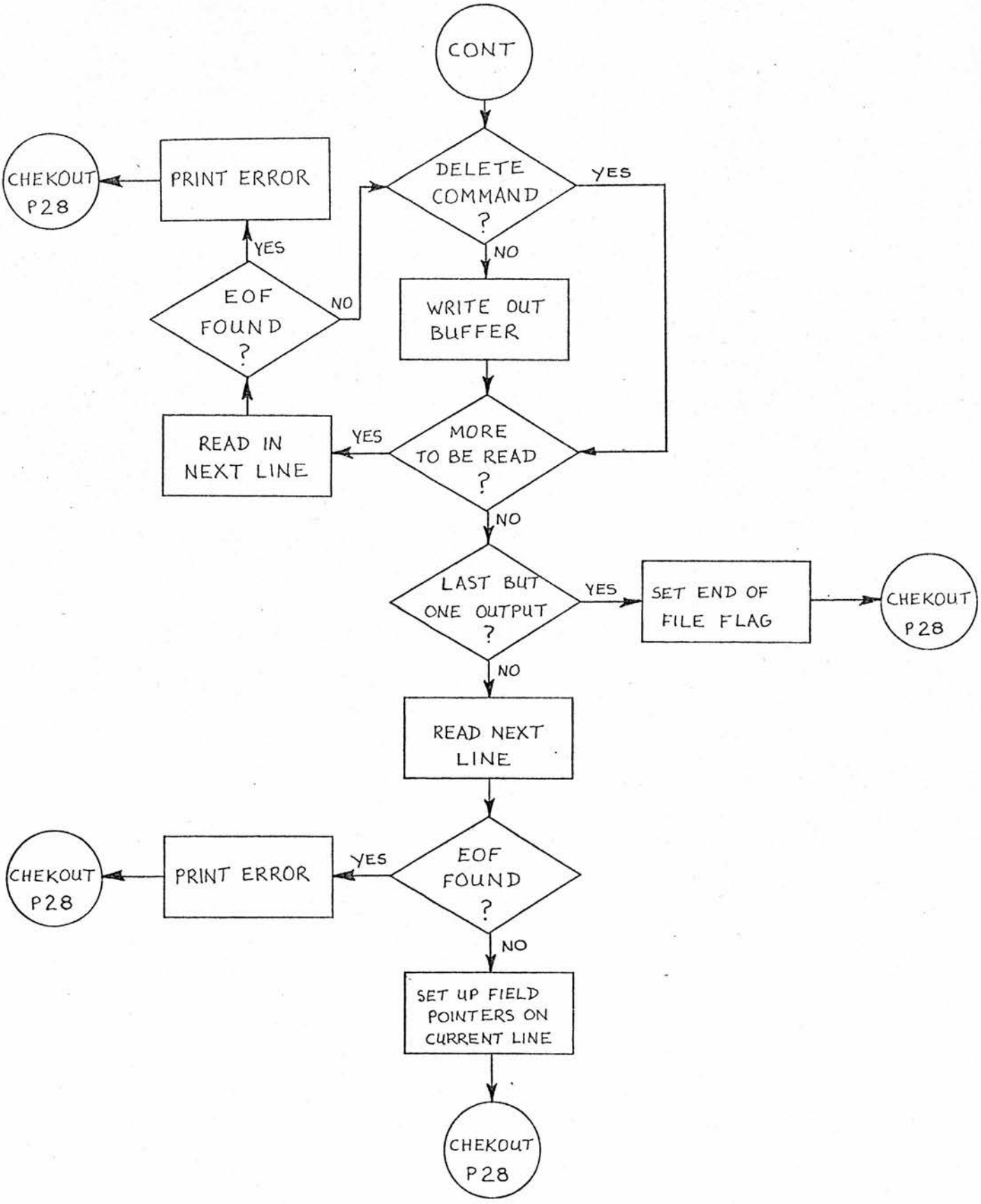
BACKSPACE FACILITY

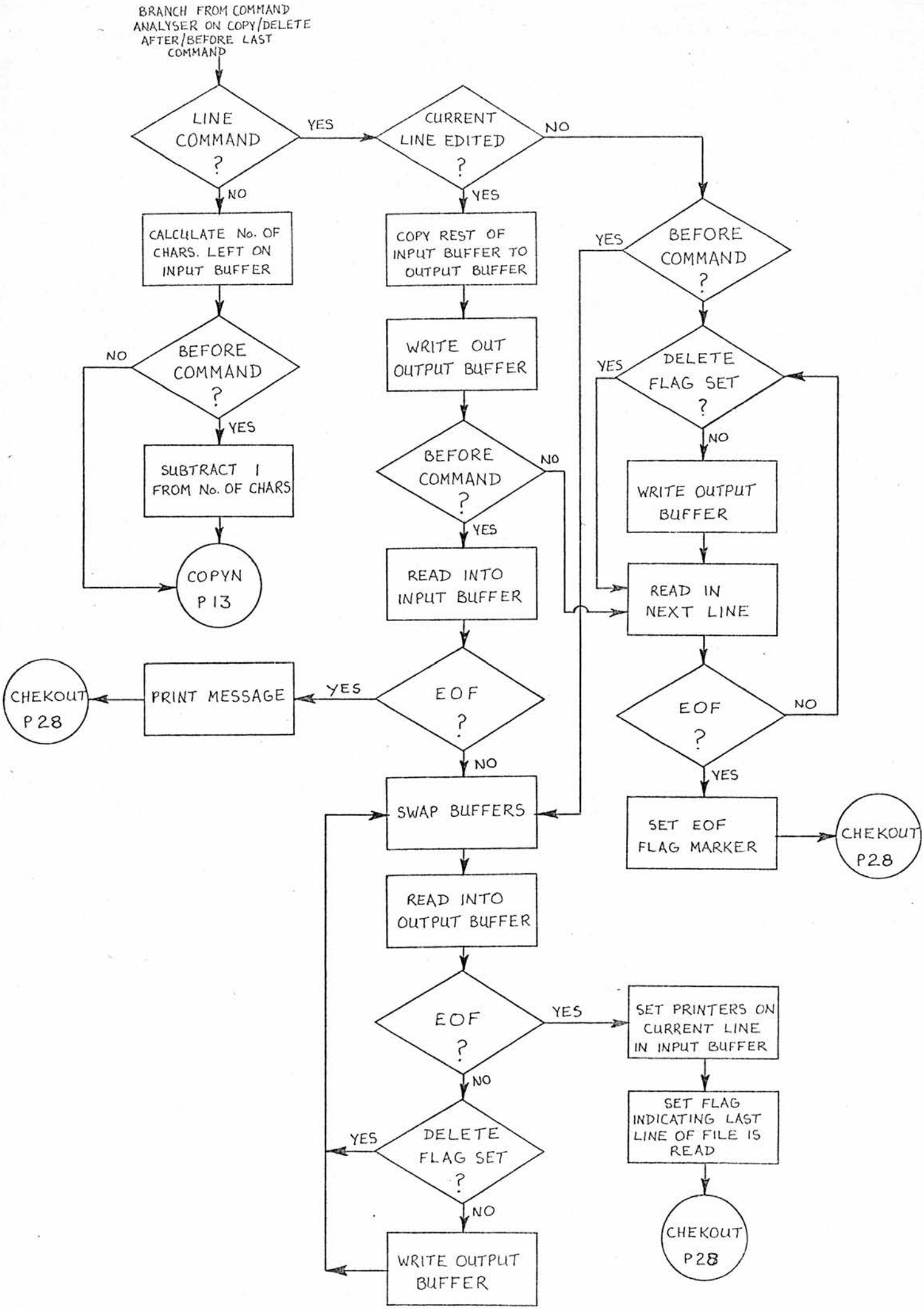


COPY / DELETE FACILITY

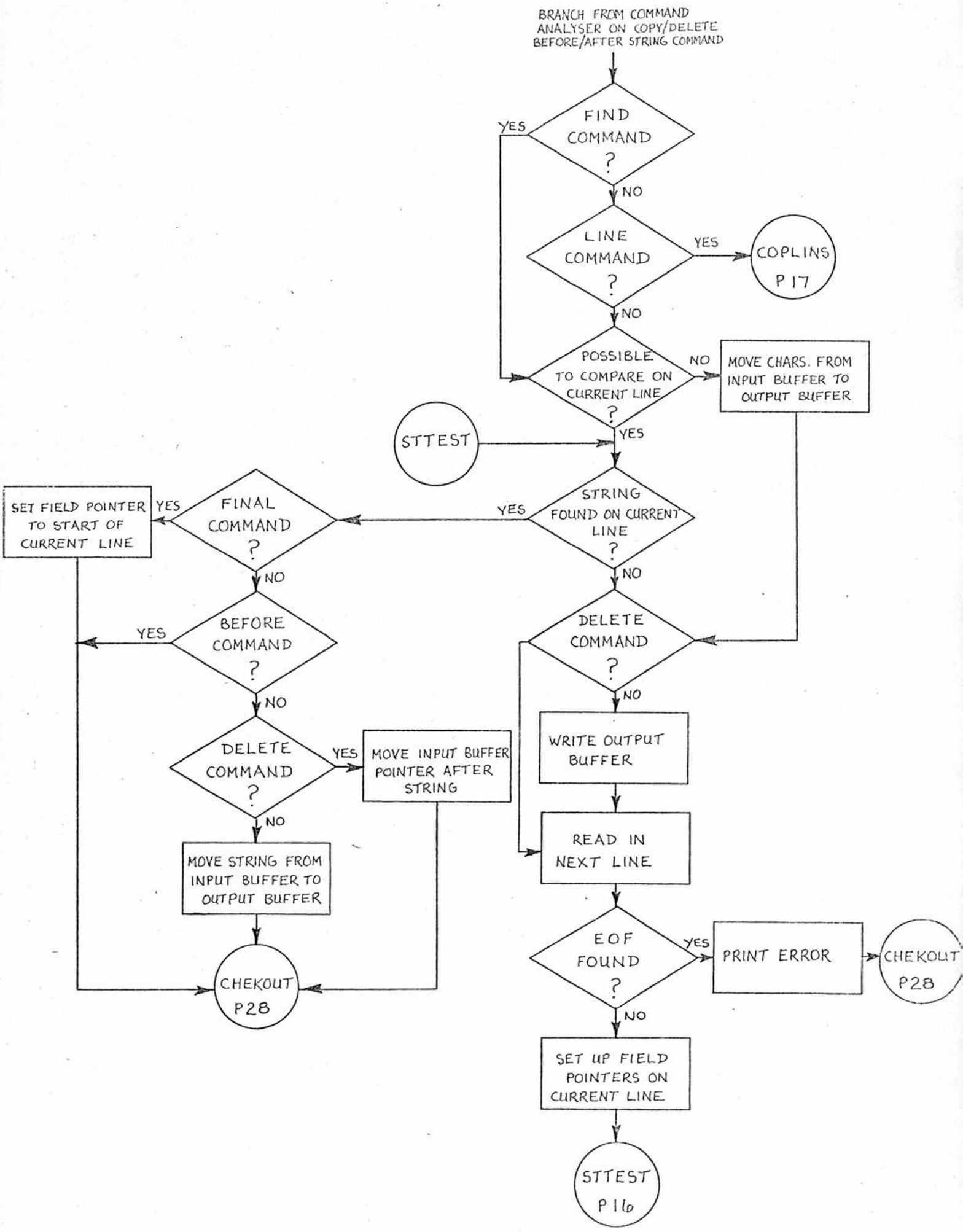


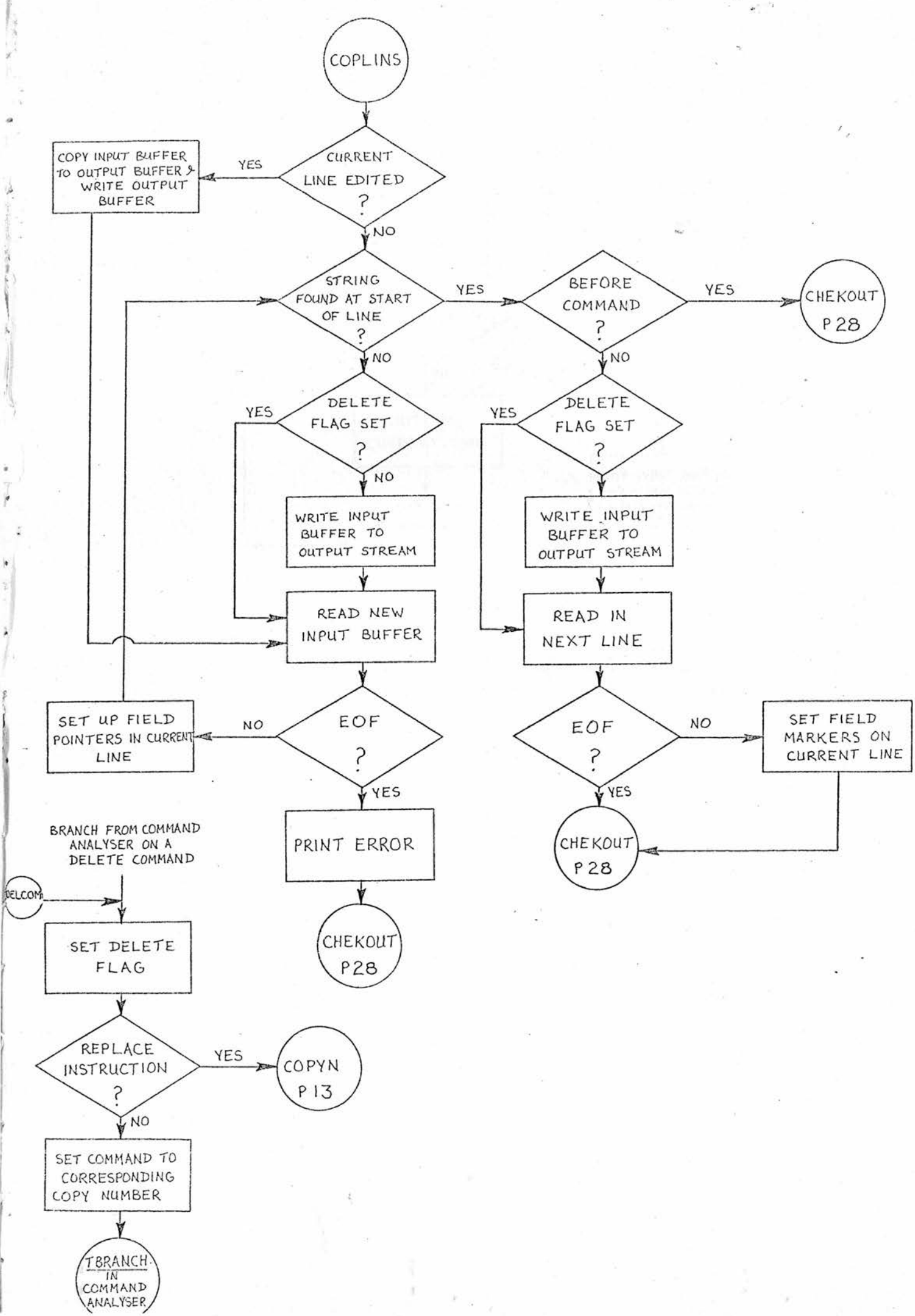
COPY / DELETE FACILITY (contd.)

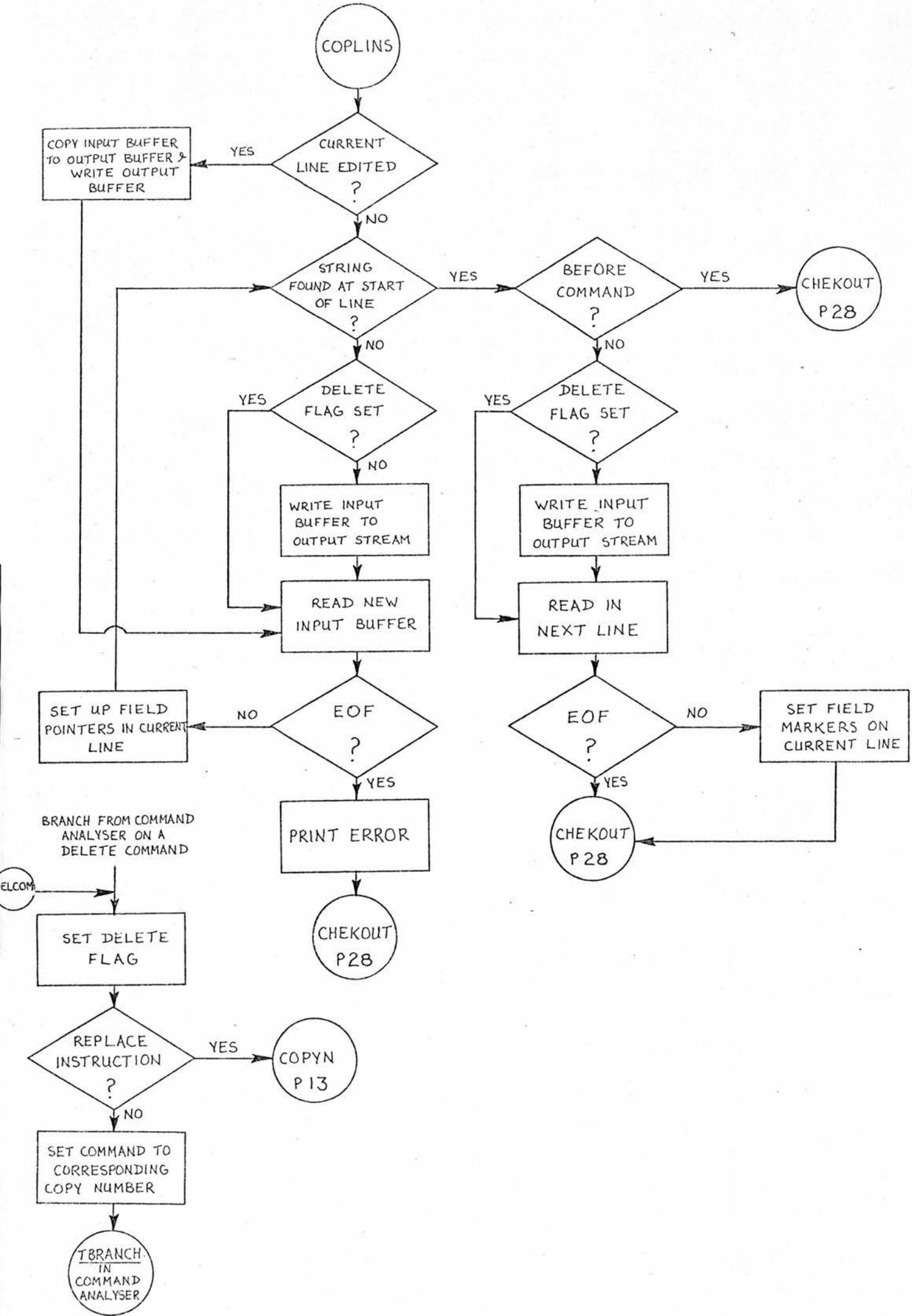



BRANCH FROM COMMAND ANALYSER ON COPY/DELETE AFTER/BEFORE LAST COMMAND
LINE COMMAND ?
YES
CURRENT LINE EDITED ?
NO
NO
YES
CALCULATE No. OF CHARS. LEFT ON INPUT BUFFER
COPY REST OF INPUT BUFFER TO OUTPUT BUFFER
BEFORE COMMAND ?
YES
BEFORE COMMAND ?
NO
YES
SUBTRACT 1 FROM No. OF CHARS
WRITE OUT OUTPUT BUFFER
COPYN P13
BEFORE COMMAND ?
NO
YES
READ INTO INPUT BUFFER
CHEKOUT P28
PRINT MESSAGE
YES
EOF ?
NO
SWAP BUFFERS
READ INTO OUTPUT BUFFER
EOF ?
YES
SET PRINTERS ON CURRENT LINE IN INPUT BUFFER
SET FLAG INDICATING LAST LINE OF FILE IS READ
CHEKOUT P28
DELETE FLAG SET ?
YES
NO
WRITE OUTPUT BUFFER
DELETE FLAG SET ?
YES
NO
WRITE OUTPUT BUFFER
READ IN NEXT LINE
EOF ?
NO
YES
SET EOF FLAG MARKER
CHEKOUT P28

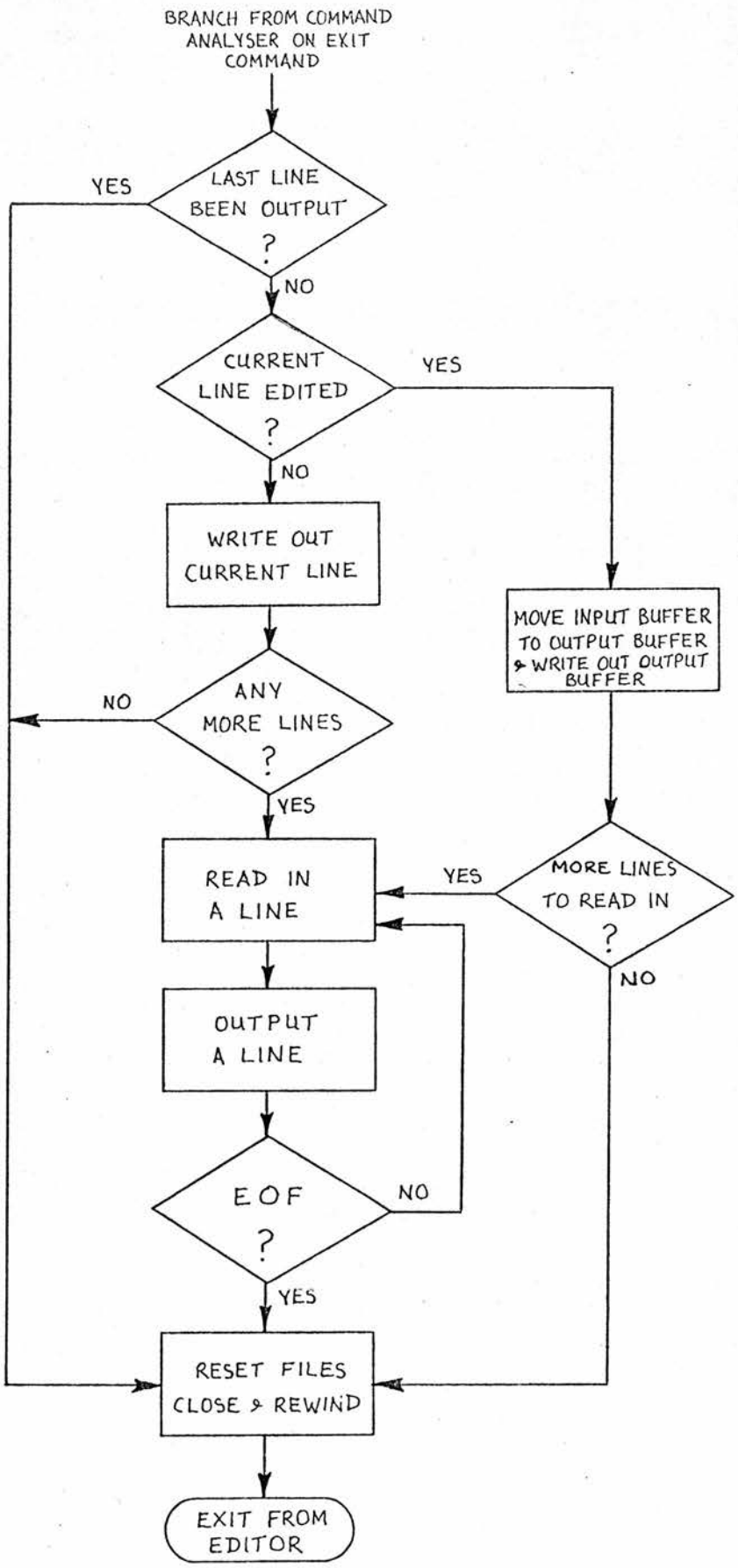
COPY/ DELETE FACILITY (contd.)

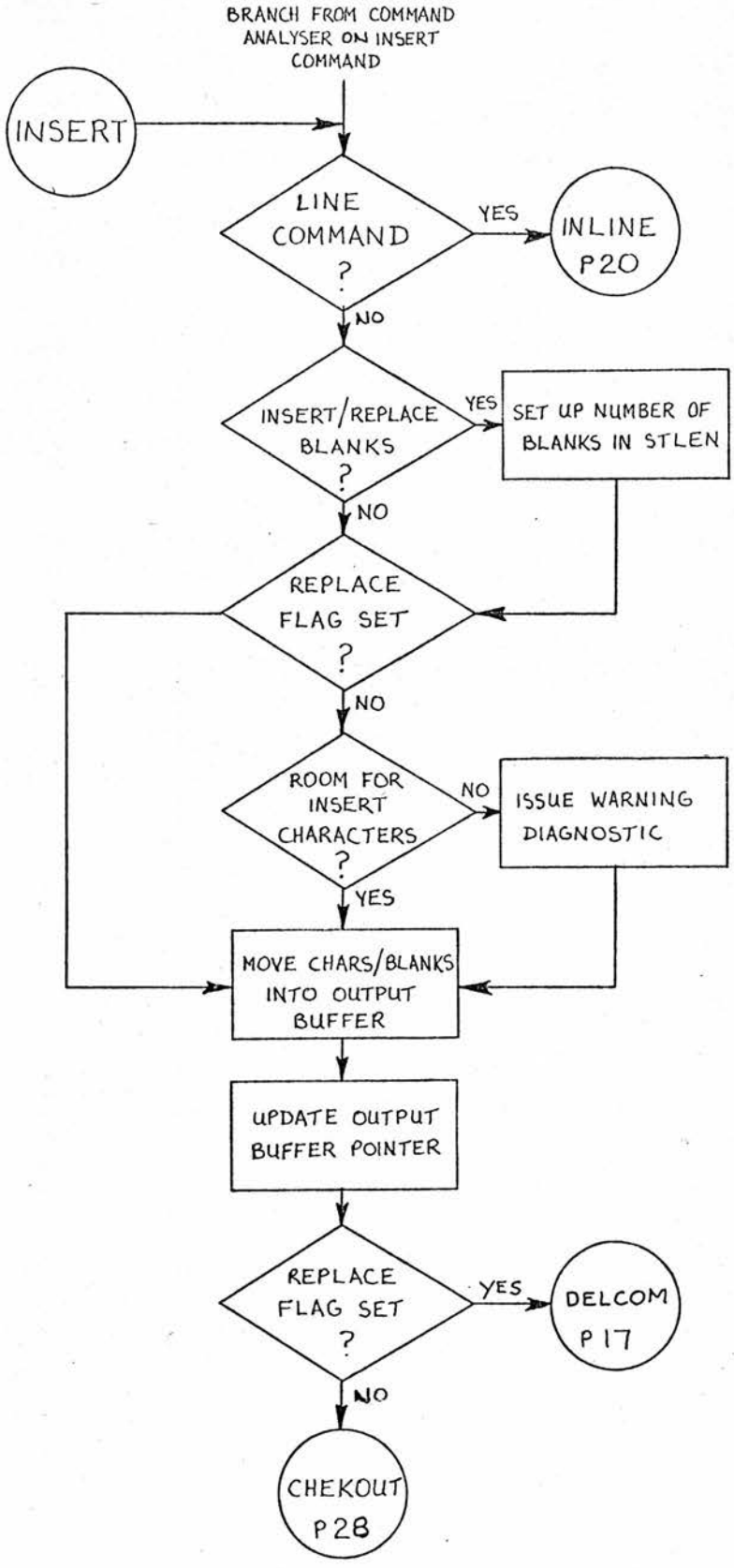



COPLINS
CURRENT LINE EDITED ?
YES
COPY INPUT BUFFER TO OUTPUT BUFFER & WRITE OUTPUT BUFFER
NO
STRING FOUND AT START OF LINE ?
YES
BEFORE COMMAND ?
YES
CHEKOUT P28
NO
NO
DELETE FLAG SET ?
YES
NO
DELETE FLAG SET ?
YES
NO
WRITE INPUT BUFFER TO OUTPUT STREAM
WRITE INPUT BUFFER TO OUTPUT STREAM
READ NEW INPUT BUFFER
READ IN NEXT LINE
EOF ?
NO
SET UP FIELD POINTERS IN CURRENT LINE
EOF ?
NO
SET FIELD MARKERS ON CURRENT LINE
YES
YES
PRINT ERROR
CHEKOUT P28
CHEKOUT P28
BRANCH FROM COMMAND ANALYSER ON A DELETE COMMAND
DELCOM
SET DELETE FLAG
REPLACE INSTRUCTION ?
YES
COPYN P13
NO
SET COMMAND TO CORRESPONDING COPY NUMBER
TBRANCH IN COMMAND ANALYSER

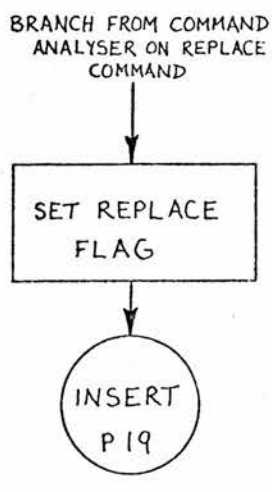

COPLINS
CURRENT LINE EDITED ?
YES
COPY INPUT BUFFER TO OUTPUT BUFFER & WRITE OUTPUT BUFFER
NO
STRING FOUND AT START OF LINE ?
YES
BEFORE COMMAND ?
YES
CHEKOUT P28
NO
NO
DELETE FLAG SET ?
YES
NO
DELETE FLAG SET ?
YES
WRITE INPUT BUFFER TO OUTPUT STREAM
WRITE INPUT BUFFER TO OUTPUT STREAM
READ NEW INPUT BUFFER
READ IN NEXT LINE
SET UP FIELD POINTERS IN CURRENT LINE
NO
EOF ?
YES
EOF ?
NO
SET FIELD MARKERS ON CURRENT LINE
YES
PRINT ERROR
CHEKOUT P28
CHEKOUT P28
BRANCH FROM COMMAND ANALYSER ON A DELETE COMMAND
ELCOM
SET DELETE FLAG
REPLACE INSTRUCTION ?
YES
COPYN P13
NO
SET COMMAND TO CORRESPONDING COPY NUMBER
T BRANCH IN COMMAND ANALYSER

EXIT FACILITY

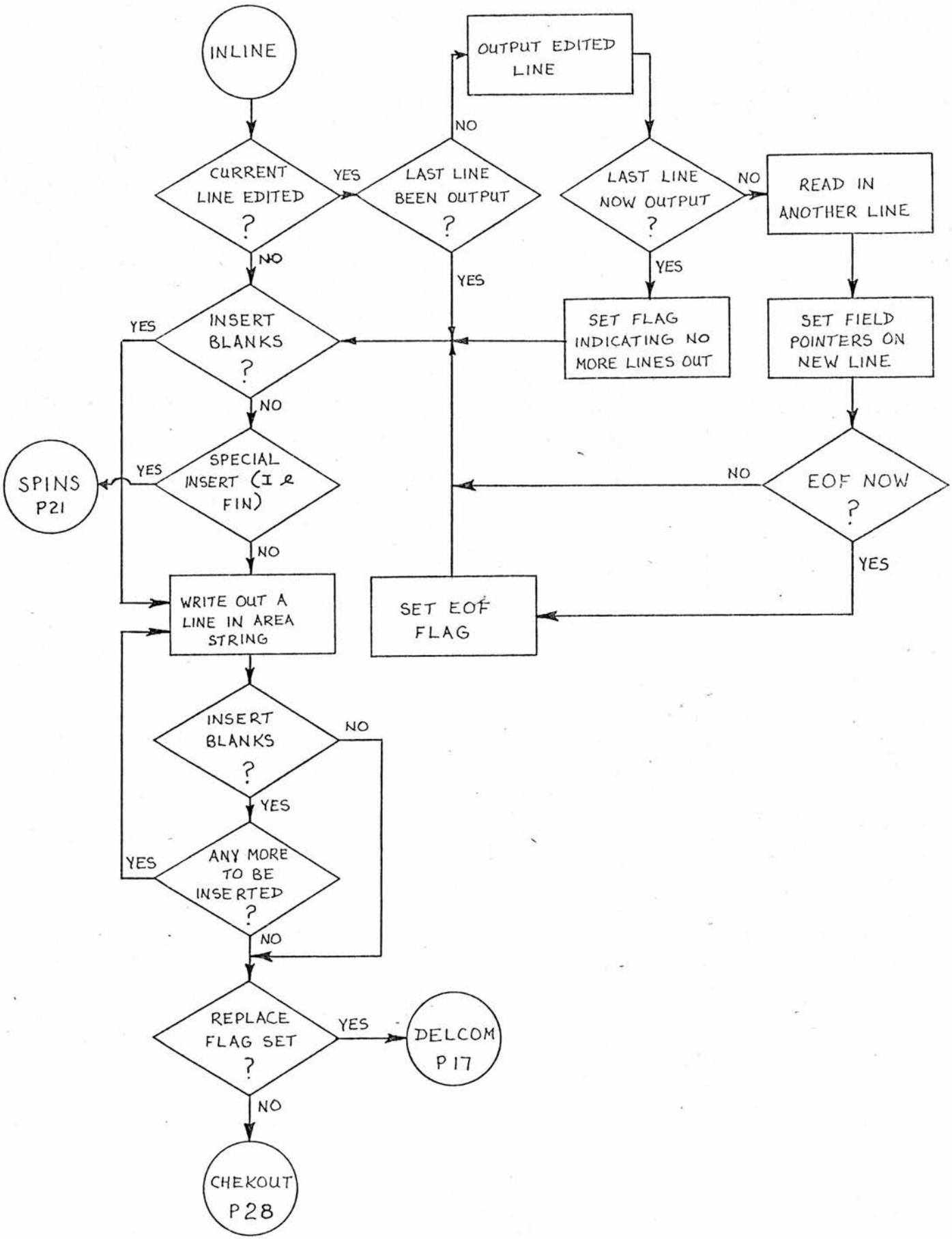




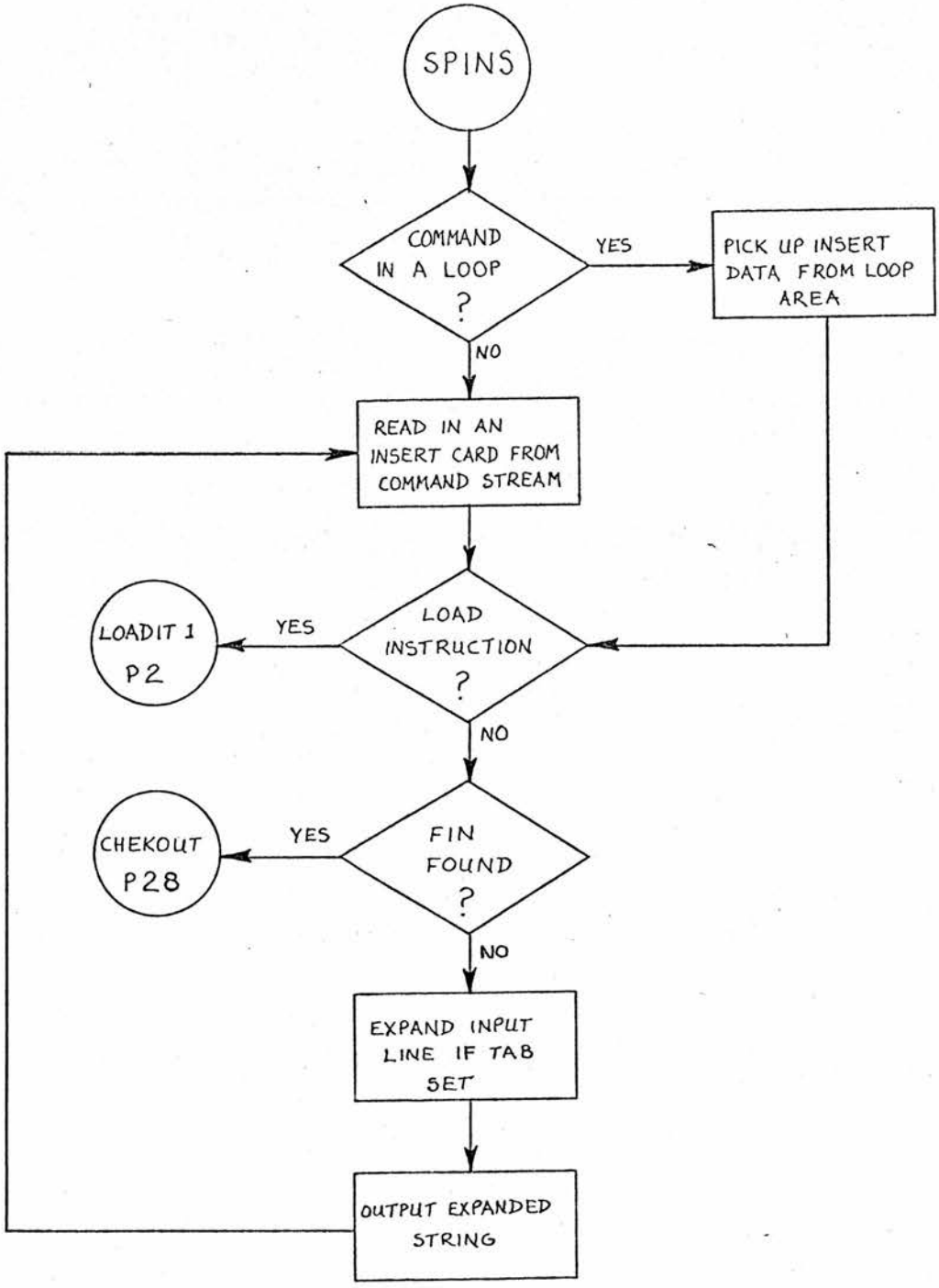
REPLACE ACTION

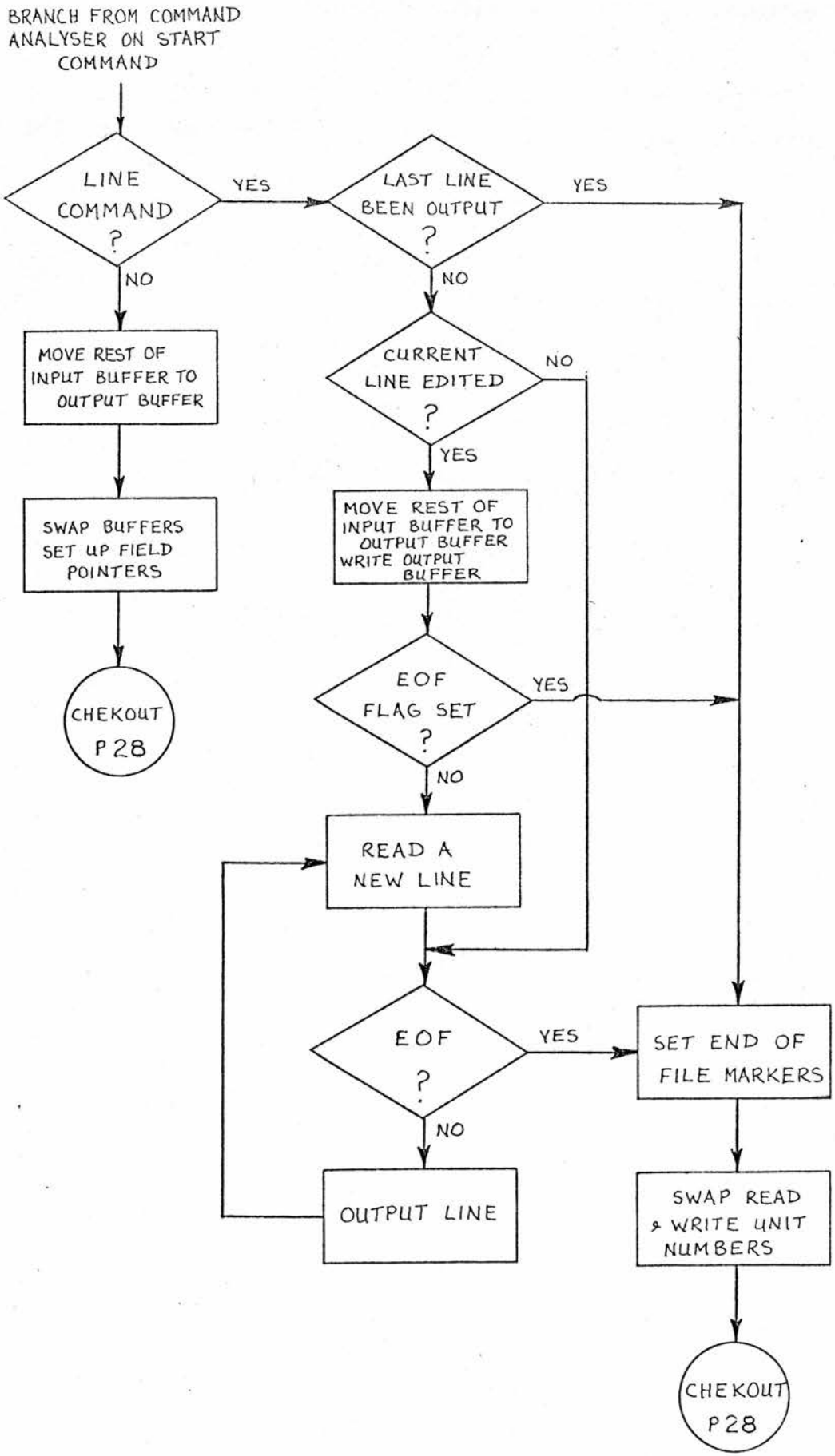


INSERT FACILITY (contd.)

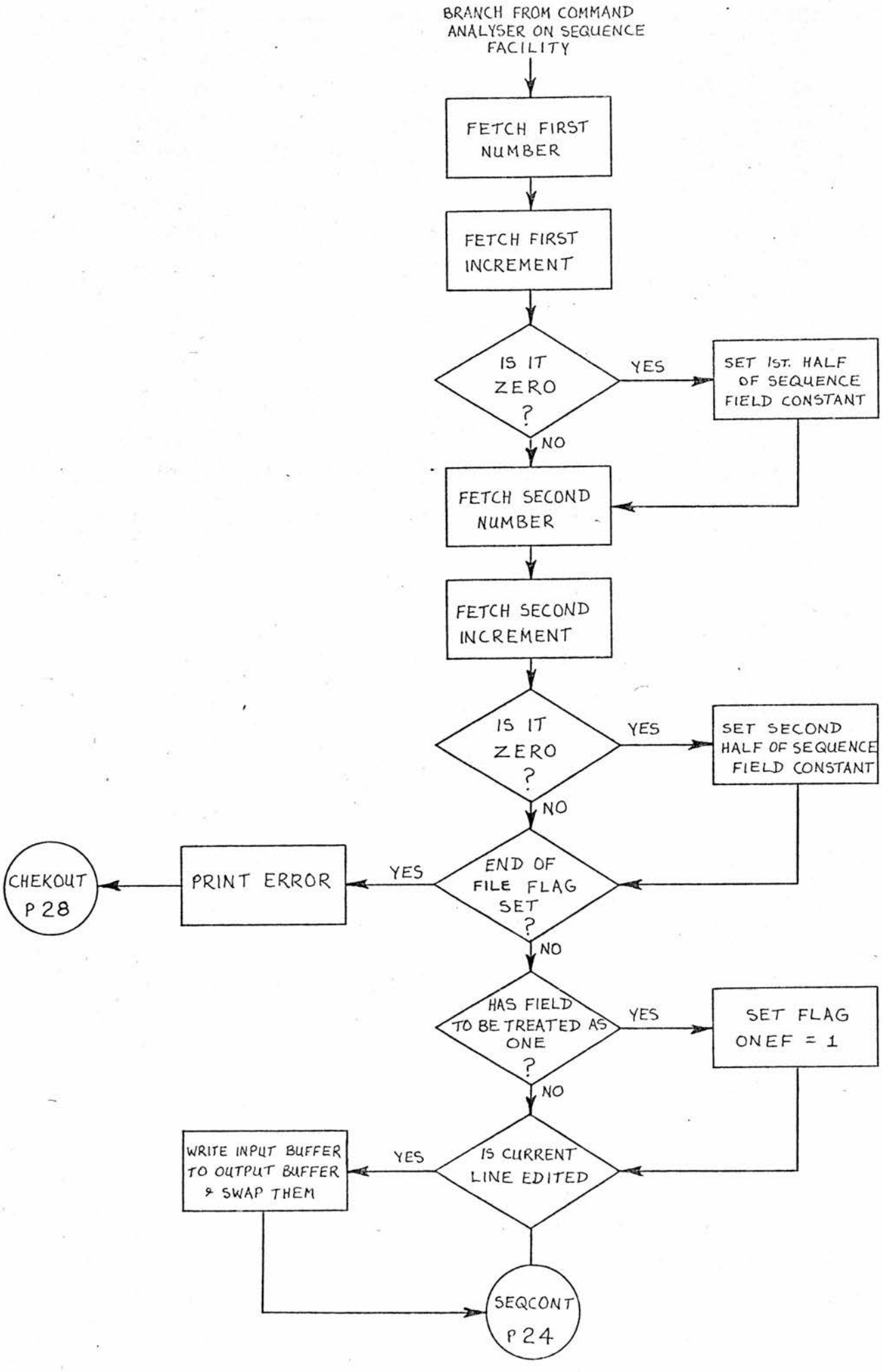


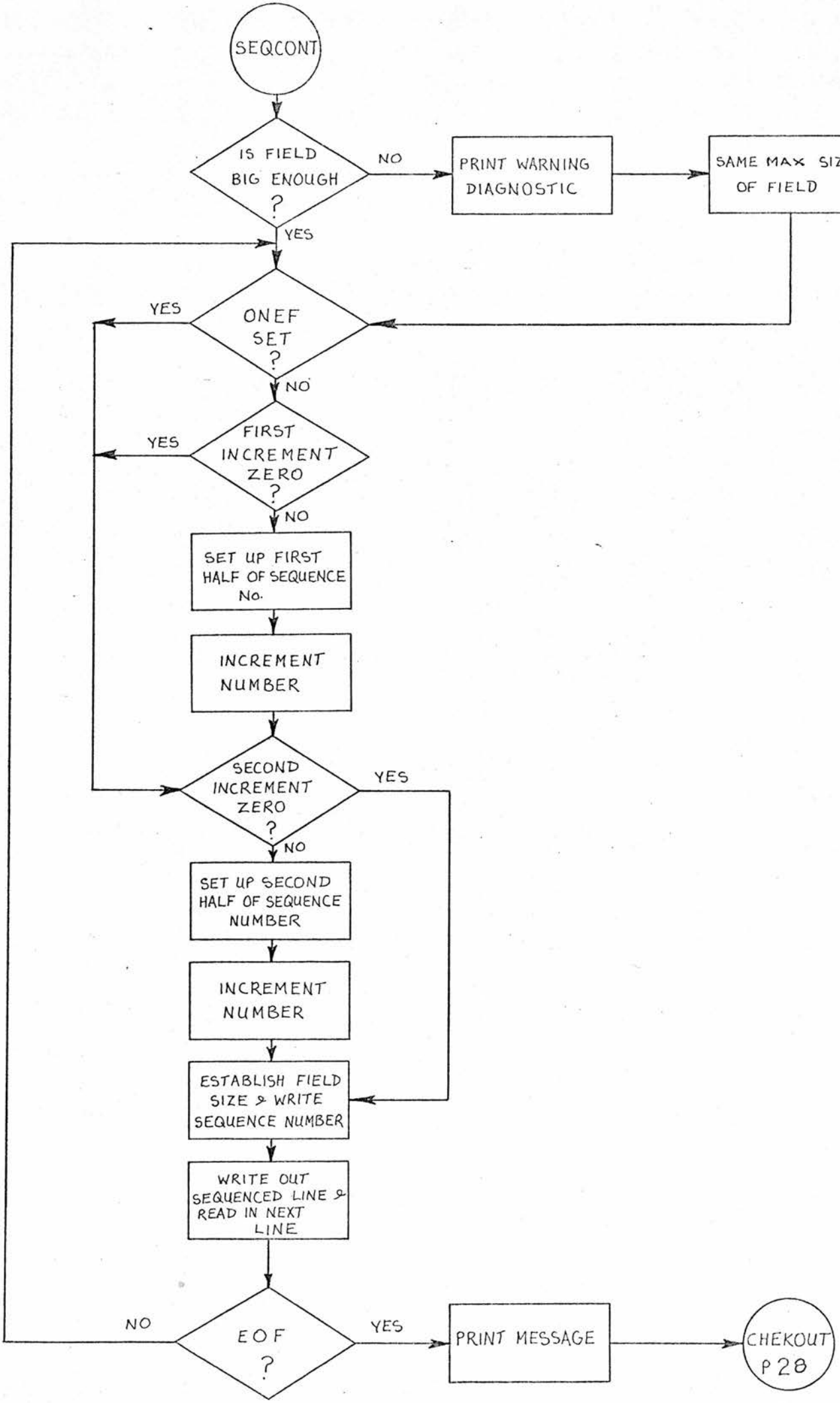
INSERT FACILITY (contd.)



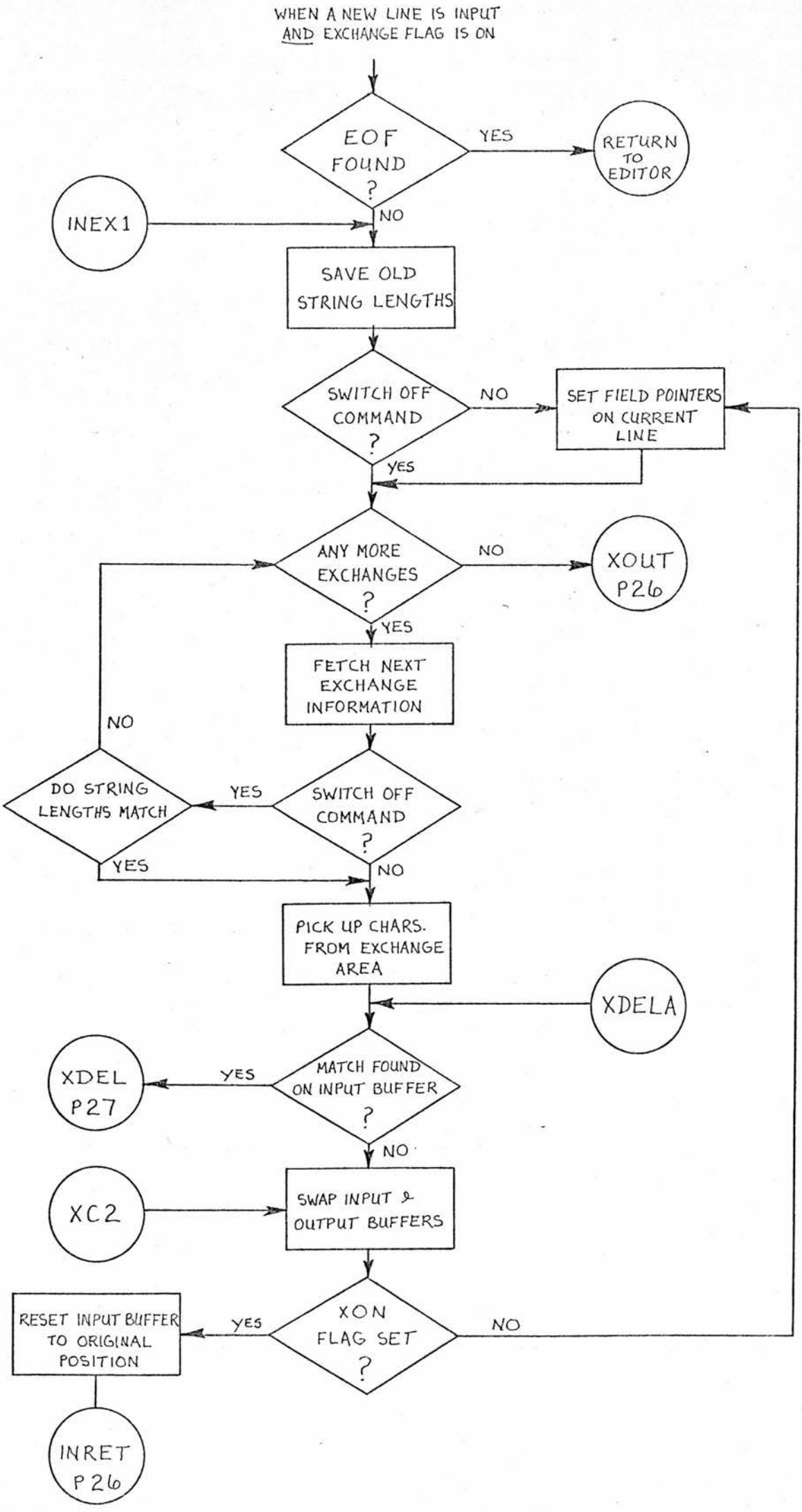

BRANCH FROM COMMAND ANALYSER ON START COMMAND
LINE COMMAND ?
YES
NO
LAST LINE BEEN OUTPUT ?
YES
NO
MOVE REST OF INPUT BUFFER TO OUTPUT BUFFER
SWAP BUFFERS SET UP FIELD POINTERS
CHEKOUT P28
CURRENT LINE EDITED ?
NO
YES
MOVE REST OF INPUT BUFFER TO OUTPUT BUFFER WRITE OUTPUT BUFFER
EOF FLAG SET ?
YES
NO
READ A NEW LINE
EOF ?
YES
NO
OUTPUT LINE
SET END OF FILE MARKERS
SWAP READ & WRITE UNIT NUMBERS
CHEKOUT P28

SEQUENCE FACILITY

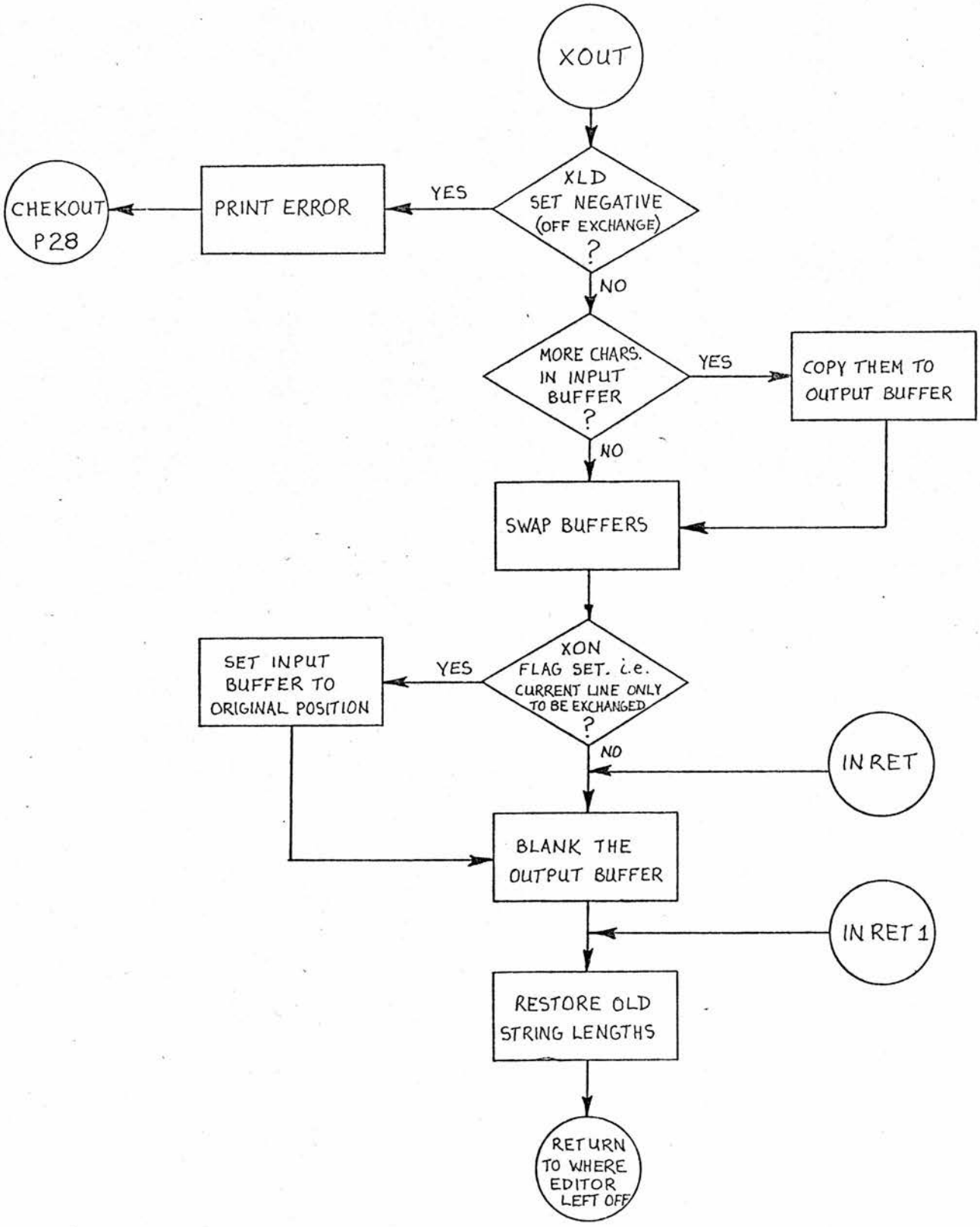



SEQCONT
IS FIELD BIG ENOUGH ?
NO
PRINT WARNING DIAGNOSTIC
SAME MAX SIZE OF FIELD
YES
ONEF SET ?
YES
NO
FIRST INCREMENT ZERO ?
YES
NO
SET UP FIRST HALF OF SEQUENCE No.
INCREMENT NUMBER
SECOND INCREMENT ZERO ?
YES
NO
SET UP SECOND HALF OF SEQUENCE NUMBER
INCREMENT NUMBER
ESTABLISH FIELD SIZE & WRITE SEQUENCE NUMBER
WRITE OUT SEQUENCED LINE & READ IN NEXT LINE
EOF ?
NO
YES
PRINT MESSAGE
CHEKOUT P28

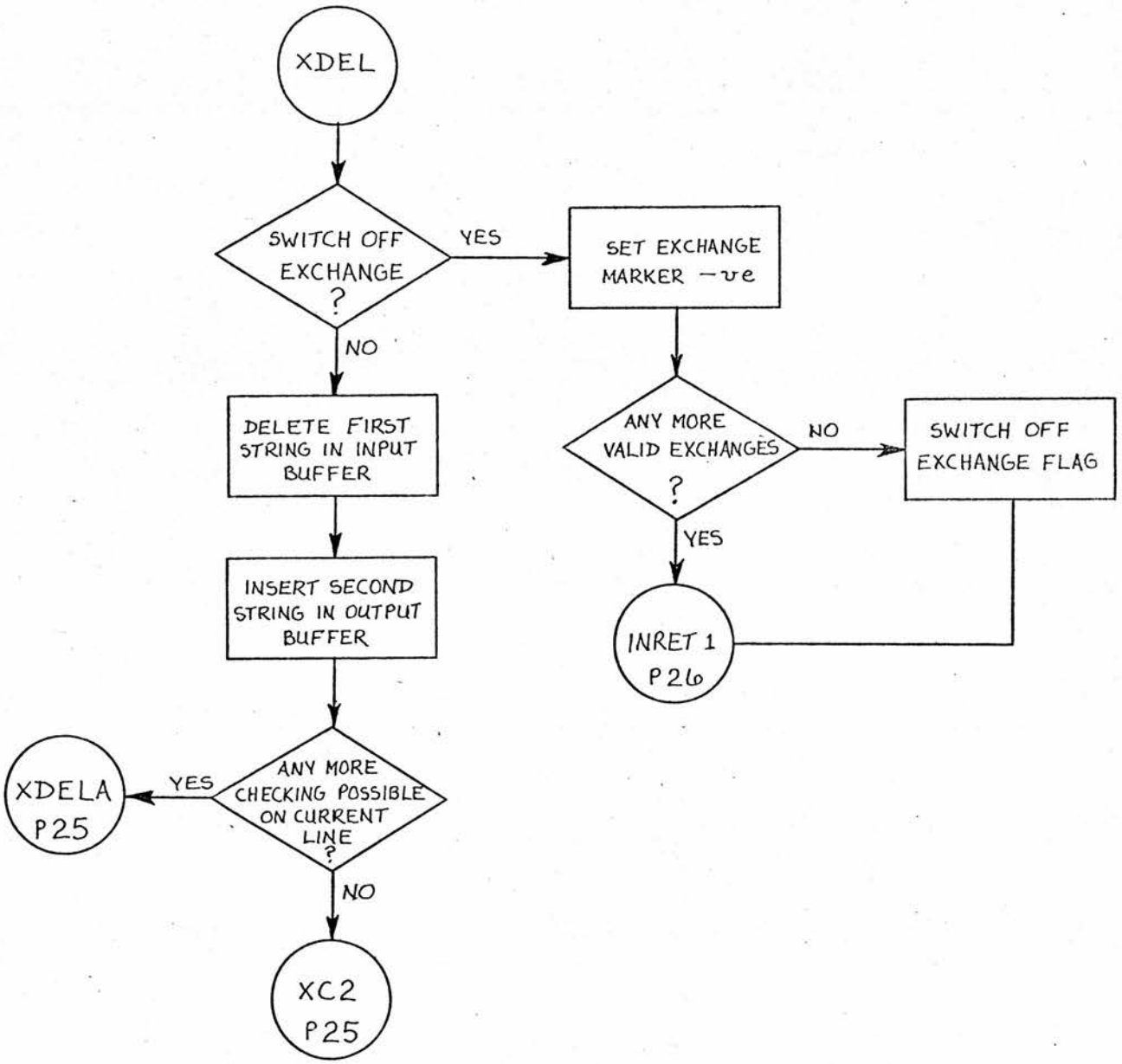
EXECUTING EXCHANGE



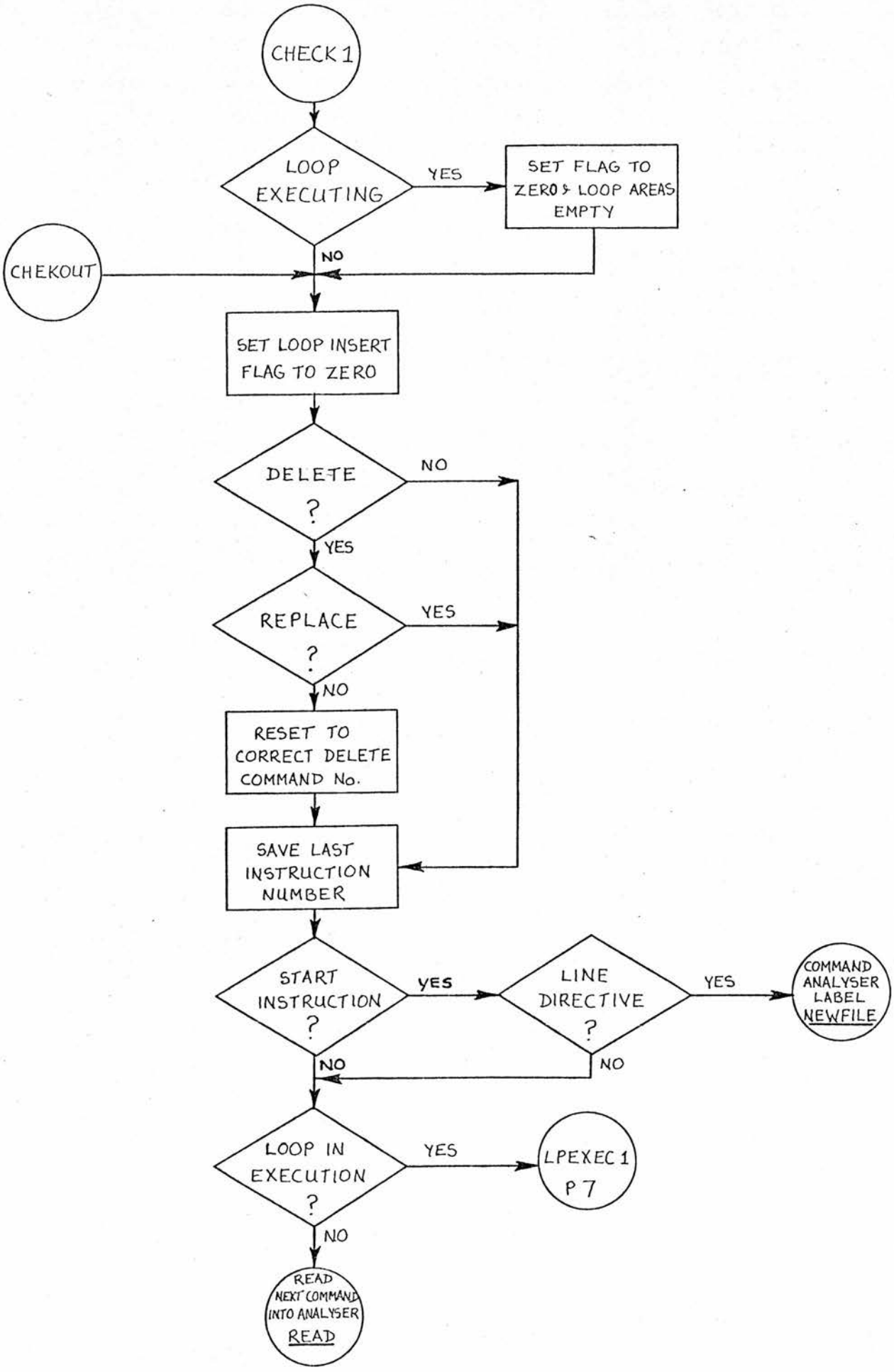
EXECUTING EXCHANGE (contd.)



EXECUTING EXCHANGE (contd.)



END OF COMMAND PROCEDURE



APPENDIX III

LISTING OF MACROS

```
.&_0 (+-*/)
COPY TO _.
_F2£
£


SKIP _.
_F4£
£


IF _ = _ SKIP _.
_F50£
£


IF _ ¬= _ SKIP _.
_F51£
£


LABEL _.
_10      EQU   *_F1£
£


JUMP TO _.
         B          _10_F1£
£


START.
         EXTRN     SWAPIT          *  EXTERNAL ROUTINES._F1£
         EXTRN     DUMPIT_F1£
         EXTRN     FILSRT_F1£
         EXTRN     ERRPRN_F1£
         EXTRN     INPUT_F1£
         EXTRN     OUTPUT_F1£
         EXTRN     DISPLY_F1£
TED2     CSECT                     *  CONTROL SECTION._F1£
         USING     *,15            *   SET UP BASES._F1£
```

```
          USING     *+4096,12_F1£
          USING     *+8192,13_F1£
          USING     *+12288,9_F1£
BEGIN     STM       14,12,12(13)        *  STORE REGISTERS._F1£
          ST        13,SAV13_F1£
          LM        12,13,BASER_F1£
          L         9,BASER+8_F1£
          LA        8,OLB               *  SAVE ADDRESS OF OLB._F1£
          LA        7,ILB               *  SAVE ADDRESS OF ILB._F1£
NEWFILE   EQU       *                   *  READ IN FIRST LINE_F1£
EOF = =H'0'£
UNIT = RUNIT£
READ IN TO ILB£
SET POINTERS£
£


STOP.
ENDP      L         13,SAV13_F1£
          LM        14,12,12(13)        *  END_F1£
          MVI       12(13),X'FF'_F1£
          BR        14                  *  RETURN._F1£
£


MOVE BLANKS TO _.
BL_10     DS        H                   *  SPACE FILL _10_F1£
          ST        0,BL_F1£
          ORG       BL_10_F1£
          ST        0,_10_F1£
          ORG       BL_10_F1£
          DC        X'D24F'_F1£
          ORG       BL_10+6_F1£
£


_ = _.
IF _10 ¬= STRING(3) SKIP 10£
REG 10 = 0£
          L         11,_20_F1£
          D         10,=F'10'_F1£
          AH        10,=H'240'_F1£
          STC       10,STRING(4)_F1£
          LA        10,0_F1£
          SH        4,=H'1'_F1£
```

```
            CH          4,FILLA_F1£
            BNL         *-24_F1£
_F9£
IF _10 ¬= UNIT SKIP 3£
            LH          1,_20_F1£
            ST          1,_10_F1£
_F9£
IF _20 ¬= UNIT SKIP 3£
            L           1,_20_F1£
            STH         1,_10_F1£
_F9£
IF _20 = STRING SKIP 10£
IF _20 = ST2 SKIP 9£
IF _20 = ST1 SKIP 8£
IF _20 = FULL SKIP 7£
IF _10 = FULL SKIP 6£
IF _10 = ST2 SKIP 5£
IF _10 = ST1 SKIP 4£
IF _20 = SAME SKIP 1£
            LH          1,_20_F1£
            STH         1,_10_F1£
_F9£
            L           1,_20_F1£
            ST          1,_10_F1£
£


_ = _ + _.
IF _10 = ST2 SKIP 7£
IF _10 = ST1 SKIP 6£
IF _20 = GREG SKIP 1£
            LH          1,_20_F1£
            AH          1,_30_F1£
IF _10 = REG SKIP 1£
            STH         1,_10_F1£
_F9£
            L           1,_20_F1£
            AH          1,_30_F1£
            ST          1,_10_F1£
£


_ = _ - _.
IF _20 = GREG SKIP 1£
            LH          1,_20_F1£
```

```
            SH          1,_30_F1£
IF _10 = REG SKIP 1£
            STH         1,_10_F1£.
£


_ = REG _.
IF _10 = FULL SKIP 2£
            STH         _20,_10_F1£
_F9£
            ST          _20,_10_F1£
£


REG _ = _.
IF _20 = FULL SKIP 5£
IF _20 = 0 SKIP 2£
            LH          _10,_20_F1£
_F9£
            LA          _10,0_F1£
_F9£
            L           _10,_20_F1£
£


ADDRESS REG _ = _.
            LA          _10,_20_F1£
£


GREG _ + _.
            AH          _10,_20_F1£
£


GREG _ - _.
IF _20 = =A(STRING) SKIP 2£
            SH          _10,_20_F1£
_F9£
            S           _10,_20_F1£
£


IF _ JUMP TO _.
IF _10 = SPACE SKIP 4£
```

```
                LH          1,_10_F1£
                CH          1,=H'1'             *  IF _10 BRANCH TO _20._F1
                BE          _20_F1£
_F9£
                LA          1,CARD_F1£
                AR          1,2_F1£
                CLI         0(1),C' '_F1£
                BE          _20_F1£
£


IF NOT _ JUMP TO _.
IF _10 ¬= SPACE SKIP 3£
                CLI         0(4),C' '_F1£
                BNE         _20_F1£
_F9£
                LH          1,_10_F1£
                CH          1,=H'0'_F1£
                BE          _20_F1£
£


IF _ _ _ JUMP TO _.
IF _10 = REG6 SKIP 11£
IF _10 = REG5 SKIP 8£
IF _10 = REG SKIP 5£
IF _10 = UNIT SKIP 10£
IF _30 = FULL SKIP 9£
IF _30 = FIN SKIP 8£
IF _30 = STRING SKIP 7£
                LH          1,_10_F1£
                CH          1,_30_F1£
SKIP 1£
                CH          5,_30_F1£
SKIP 1£
                CH          6,_30_F1£
SKIP 2£
                L           1,_10_F1£
                C           1,_30_F1£
_F1£
                B222        4444444444£
£


SWAP _ AND _.
```

```
IF _10 = ILB SKIP 6£
IF _10 = OLB SKIP 5£
          LH        1,_10           *  SWAP _10 AND _20_F1£
           LH       2,_20_F1£
          STH       2,_10_F1£
          STH       1,_20_F1£
_F9£
          LR        1,7             *  SWAP _10 AND _20._F1£
          LR        7,8_F1£
          LR        8,1_F1£
£

MOVE _ CHARS FROM _ TO _,_.
IF _20 = ILB SKIP 8£
          LR        10,7_F1£
IF _20 = OLB SKIP 2£
          LA        7,_20_F1£
SKIP 1£
          LR        7,8_F1£
IF _20 = OLB SKIP 4£
          LH        1,=H'0'_F1£
SKIP 3£
          LH        1.IPP_F1£
SKIP 1£
          LH        1,OPP_F1£
IF _30 = OLB SKIP 11£
          LR        11,8_F1£
IF _30 = ILB SKIP 2£
          LA        8,_30_F1£
SKIP 1£
          LR        8,10_F1£
IF _30 = CHTAB SKIP 3£
IF _30 = ILB SKIP 6£
          LH        2,=H'0'_F1£
SKIP 1£
          LH        2,CTP_F1£
SKIP 1£
          LH        2,OPP_F1£
SKIP 1£
          LH        2,IPP_F1£
          LH        3,_10_F1£
          BAL       14,MCVES_F1£
IF _20 = ILB SKIP 1£
          LR        7,10_F1£
```

```
IF _30 = CLB SKIP 1£
          LR        8,11_F1£
IF _40 = KEEP SKIP 1£
_F9£
IF _20 = ILB SKIP 2£
IF _20 = CLB SKIP 3£
SKIP 3£
          STH       1,IPP_F1£
SKIP 1£
          STH       1,OPP_F1£
IF _30 = ILB SKIP 3£
IF _30 = CLB SKIP 4£
IF _30 = CHTAB SKIP 5£
_F9£
          STH.      2,IPP_F1£
_F9£
          STH       2,CPP_F1£
_F9£
          STH       2,CTP_F1£
£


COMPARE _ WITH _ IF FOUND _.
IF _10 = CLB SKIP 3£
_30A      LR        1,7_F1£
          AH        1,IPP_F1£
SKIP 2£
_30A      LR        1,8_F1£
          AH        1,OPP_F1£
          LA        2,_20_F1£
          LH        3,STLEN_F1£
          SH        3,=H'1'_F1£
          EX        3,CLC_F1£
        BE          _30_F1£
IF _30 ¬= BEFLCK SKIP 1£
_F9£
IF _30 ¬= XDEL SKIP 1£
IF XLD L =F'0' JUMP TO XCHK£
IF _10 = CLB SKIP 5£
IF DELETE JUMP TO *+32£
MOVE =H'1' CHARS FROM ILB TO CLB,KEEP£
          B         *+16_F1£
IPP = IPP + =H'1'£
SKIP 1£
OPP = OPP + =H'1'£
```

```
          AH        1,STLEN_F1£
IF _10 = CLB SKIP 4£
IF REG NH EP JUMP TO _30A£
NO = =H'80' - IPP£
MOVE NO CHARS FROM ILB TO CLB,NCKEEP£
SKIP 1£
IF REG NH EP JUMP TO *-46£
£


FIND LAST CHAR.
          LA        1,CARD_F1£
          AH        1,=H'80'_F1£
          SH        1,=H'1'_F1£
          CLI       0(1),C' '_F1£
          BE        *-8_F1£
          S         1,=A(CARD)_F1£
          AH        1,=H'1'_F1£
          STH       1,STLEN_F1£
£


FIND CHAR.
          SR        1,1_F1£
          AH        2,=H'1'_F1£
          IC        1,CARD(2)_F1£
          CH        1,=H'64'          *  IF SPACE  ERROR._F1£
          BE        ERR3_F1£
          CH        1,=H'111'_F1£
          BE        TABQU_F1£
          STC       1,TABCHAR+1_F1£
          AH        2,=H'1'_F1£
          IC        1,CARD(2)_F1£
TABCOM    CH        1,=H'107'         *  COMMA EXPECTED._F1£
          BNE       ERR3_F1£
          B         TABCONT_F1£
TABQU     AH        2,=H'1'_F1£
          IC        1,CARD(2)_F1£
          CH        1,=H'111'_F1£
          BE        TABCFF_F1£
          B         TABCOM_F1£
TABCONT   EQU       *_F1£
£
```

```
FIND NUM.
          BAL       14,NUMF_F1£
£

TABCHECK.
          BAL       14,TABST_F1£
£

SET POINTERS.
          BAL       14,SETP          *  GO AND SET POINTERS._F1£
£

READ IN TO _.
IF _10 = ILB SKIP 3£
IF _10 = OLB SKIP 4£
          LA        1,_10_F1£
SKIP 3£
          LR        1,7_F1£
SKIP 1£
          LR        1,8_F1£
          BAL       10,IN            *  READ A RECORD FROM _10._F
£

WRITE OUT _.
IF _10 = CARD SKIP 6£
IF _10 = STRING SKIP 5£
IF _10 = OLB SKIP 2£
          LR        1,7_F1£
SKIP 1£
          LR        1,8_F1£
SKIP 1£
          LA        1,_10_F1£
          BAL       10,CUT1_F1£
£

DISPLAY.
          LA        2,0_F1£
          LR        1,7_F1£
          ST        1,PARAM(2)_F1£
          A         2,=F'4'_F1£
```

```
                LA          1,RUNIT_F1£
                ST          1,PARAM(2)_F1£
                AH          2,=H'4'_F1£
                LA          1,WUNIT_F1£
                ST          1,PARAM(2)_F1£
                AH          2,=H'4'_F1£
                LA          1,ECF_F1£
                ST          1,PARAM(2)_F1£
                LA          1,PARAM_F1£
                LR          4,13_F1£
                LA          13,SAVE_F1£
                LR          6,15_F1£
                L           15,=A(DISPLY)_F1£
                BALR        14,15_F1£
                LR          15,6_F1£
                LR          13,4_F1£
                LH          1,EOF_F1£
                CH          1,=H'1'_F1£
                 BE          ERR15_F1£
                B           CHEKOUT_F1£
£


INPUT.
IN              SR          2,2                 *  PREPARE PARAMETERS AND_F1
                ST          1,PARAM(2)          *  CALL INPUT._F1£
IF UNIT E =F'5' JUMP TC *+16£
IF EOF JUMP TO ERR15£
                AH          2,=H'4'_F1£
                LA          1,UNIT_F1£
                ST          1,PARAM(2)_F1£
                AH          2,=H'4'_F1£
                LA          1,EOF_F1£
                ST          1,PARAM(2)_F1£
                AH          2,=H'4'_F1£
                LA          1,PRINT_F1£
                ST          1,PARAM(2)_F1£
                LA          1,PARAM_F1£
                LR          4,13_F1£
                LA          13,SAVE_F1£
                LR          2,15_F1£
                L           15,=A(INPUT)_F1£
                BALR        14,15_F1£
                LR          15,2_F1£
                LR          13,4_F1£
```

```
IF UNIT E =F'5' JUMP TC *+16£
IF EXCH JUMP TC INEX£
        BR      10_F1£
£


OUTPUT.
        ST      10,FULL_F1£
        SR      2,2                 *  PREPARE PARAMETERS FOR_F
        ST      1,PARAM(2)          *  CUTPUT AND CALL IT_F1£
IF REG5 E =H'17' JUMP TC CUTP£
IF REG5 E =H'18' JUMP TC OUTP£
IF ENDF JUMP TO ENDOUT£
OUTP    AH      2,=H'4'_F1£
        LA      1,UNIT_F1£
        ST      1,PARAM(2)_F1£
        AH      2,=H'4'_F1£
        LA      1,PRINT_F1£
        ST      1,PARAM(2)_F1£
        LA      1,PARAM_F1£
        LR      4,13_F1£
        LA      13,SAVE_F1£
        LR      2,15_F1£
        L       15,=A(CUTPUT)_F1£
        BALR    14,15_F1£
        LR      15,2_F1£
        LR      13,4_F1£
        L       1,PARAM_F1£
        CR      1,8_F1£
        BNE     ENDOUT_F1£
OPP = =H'0'£
MOVE =H'80' CHARS FROM BL TO CLB,NOKEEP£
ENDOUT  L       10,FULL_F1£
        BR      10_F1£
£


POINTERS.
SETP    LR      10,14_F1£
        LR      1,7                 *  SET UP VALUE OF_F1£
        AH      1,=H'80'            *  EP BY SEARCHING_F1£
        SH      1,=H'1'             *  FROM END OF CARD_F1£
        CLI     0(1),C' '           *  TO FIRST NON-BLANK_F1£
        BE      *-8                 *  CHARACTER._F1£
        LR      2,7_F1£
```

```
            SR          1,2_F1£
            AH          1,=H'1'_F1£
            STH         1,EP_F1£
            CH          5,=H'4'_F1£
            BL          RETC                *  IF BACKSPACE MISS ALL THIS
IPP = =H'0'£
OPP = SAME£
MOVE SFP CHARS FROM ILB TO OLB,KEEP£
IF EFP NL EP JUMP TO RETC£
EP = EFP£
RETC        BR          10                  *  RETURN_F1£
£

MOVES.
MOVES       CH          3,=H'0'_F1£
            BNH         GOBK_F1£
MOVES1      IC          4,0(1,7)_F1£
            STC         4,0(2,8)_F1£
            AH          1,=H'1'_F1£
            AH          2,=H'1'_F1£
            C           8,=A(CHTAB)_F1£
            BE          GOBK1_F1£
            C            8,=A(LCOPCOM)_F1£
            BE          GOBK1_F1£
            C           8,=A(XCHRS)_F1£
            BE          GOBK1_F1£
            CH          2,=H'79'            *  OUTPUT BUFFER FULL._F1£
            BH          *+8_F1£
GOBK1       BCT         3,MOVES1_F1£
GOBK        BR          14_F1£
£

DUMP.
DUMPS       LA          1,LASTINS_F1£
            ST          1,PARAM_F1£
            LA          1,PARAM_F1£
            LR          4,13_F1£
            LA          13,SAVE_F1£
            LR          2,15_F1£
            L          15,=A(DUMPIT)_F1£
             BALR       14,15_F1£
            LR          15,2_F1£
            LR          13,4_F1£
```

```
            B           READ+16
£


NUMBER.
NUMF        LA          1,0_F1£
            STH         1,NUM_F1£
            ST          1,FULL_F1£
            LA          10,0_F1£
            LA          11,0_F1£
            LR          4,2                 *  GET POSITION ON CARD._F1£
            AH          4,=H'1'_F1£
            AH          2,=H'1'_F1£
            IC          1,CARD(2)_F1£
            CH          1,=H'240'_F1£
            BL          *+26_F1£
            CH          1,=H'249'_F1£
            BH          *+18_F1£
            SH          1,=H'240'_F1£
            M           10,=F'10'_F1£
            AR          11,1_F1£
            B           *-34_F1£
            CH          1,=H'107'_F1£
            BE          *+12_F1£
           CH          1,=H'64'            *  SPACE PRESENT._F1£
            BNE         ERR3_F1£
            CR          4,2_F1£
            BNE         *+8_F1£
            LH          11,=H'-1'_F1£
            ST          11,FULL_F1£
            STH         11,NUM_F1£
            BR          14_F1£
£


TABSTORE.
TABST       EQU         *_F1£
FULL = REG 14£
FILLB = REG 2£
REG 4 = 0£
IF REG5 E =H'22' JUMP TO *+12£
IF REG5 NE =H'17' JUMP TO TBS1£
IF NOT LINE JUMP TO TBS1£
NO = EFP - SFP£
IF REG L STLEN JUMP TO ERR18£
```

```
JUMP TO LOOPR1+4£
LABEL LOOPR1£
REG 5 = FILLA£
MOVE =H'80' CHARS FROM BL TO STRING,NOKEEP£
REG 4 = SFP£
TBS1      EQU       *_F1£
REG 2 = FILLB£
REG 1 = STLEN£
TABST1    LA        10,0_F1£
          LH        11,=H'64'          *  LOAD A SPACE_F1£
AGAIN     IC        3,CARD(2)          *  MOVE STRING ON CARD_F1£
          AH        2,=H'1'            *  INTO THE AREA STRING._F1£
          LH        6,TAB              *  CHECK IF TAB IS IN OPERAT
          CH        6,=H'1'_F1£
          BNE       STRCHR_F1£
          CH        3,TABCHAR          *  IF TAB CHAR GO AND SPACE F
          BNE       STRCHR_F1£
          LH        6,TABF(10)_F1£
          CH        6,=H'80'_F1£
          BE        ERR24_F1£
          AH        10,=H'2'_F1£
          CH        10,=H'16'_F1£
          BH        ERR24_F1£
          CR        4,6_F1£
          BNL       ERR25_F1£
          SR        6,4_F1£
          SH        6,=H'1' _F1£
          CH        6,=H'0'_F1£
          BE        LOOPR _F1£
          STC       11,STRING(4)_F1£
          AH        4,=H'1'_F1£
          BCT       6,*-8_F1£
          B         LOOPR_F1£
STRCHR    STC       3,STRING(4)        *  IF NO TABS THEN _F1£
          AH        4,=H'1'            *  STORE DATA IN STRING._F1£
LOOPR     BCT       1,AGAIN            *  LOOP COUNTER._F1£
STLEN = REG 4£
REG 14 = FULL£
          BR        14_F1£
£


CONSTANTS.
LASTINS   DC        H'0'_F1£
FILL      DC        H'0'               *  STORAGE_F1£
```

```
FILLA     DC      H'0'_F1£
IPPS      DC       H'0'_F1£
XON       DC       H'0'_F1£
FILLB     DC       H'0'_F1£
LCP       DC       H'0'_F1£
LNP       DC       H'0'_F1£
STNO      DC       H'0'_F1£
LDLI      DC       H'0'_F1£
LDL       DC       H'0'_F1£
LCP1      DC       H'0'_F1£
FLD       DC       H'0'_F1£
CHP       DC       H'0'_F1£
CTP       DC       H'0'_F1£
DUPN      DC       H'0'_F1£
LOOPCNT   DC       H'0'_F1£
LPEX      DC       H'0'_F1£
LPSET     DC       H'0'_F1£
LPCOMP    DC       H'0'_F1£
LSTP      DC       H'0'_F1£
LPMAX     DC       H'540'_F1£
TABCHAR   DC       H'0'_F1£
TAB       DC       H'0'_F1£
EOF       DC       H'0'_F1£
ENDF      DC       H'0'_F1£
ENDNL     DC       H'0'_F1£
DELETE    DC       H'0'_F1£
STLEN     DC       H'0'_F1£
STLEN2    DC       H'0'_F1£
EXCH      DC       H'0'_F1£
CHAR      DC       H'0'_F1£
LINE      DC       H'0'_F1£
REPL      DC       H'0'_F1£
PRINT     DC       H'0'_F1£
XPTR      DC       H'0'_F1£
XPT1      DC       H'0'_F1£
XLD       DC       H'0'_F1£
OLDST     DC       H'0'_F1£
XTP       DC       H'0'_F1£
XCP       DC       H'0'_F1£
LINSRT    DC       H'0'_F1£
ONEF      DC       H'0'_F1£
STL1      DC       H'0'_F1£
STL2      DC       H'0'_F1£
INC1      DC       H'0'_F1£
INC2      DC       H'0'_F1£
```

```
NUM       DC      H'1'_F1£
NO        DC      H'0'_F1£
IPP       DC      H'0'_F1£
OPP       DC      H'0'_F1£
SFP       DC      H'0'_F1£
EFP       DC      H'80'_F1£
EP        DC      H'80'_F1£
RUNIT     DC      H'2'_F1£
WUNIT     DC      H'1'_F1£
UNIT      DC      F'0'_F1£
PARAM     DC      4F'0'_F1£
STRING    DC      CL80' '_F1£
CARD      DC      CL80' '_F1£
ILB       DC      CL80' '_F1£
OLB       DC      CL80' '_F1£
LNAMES    DC      20F'0'_F1£
LCPOS     DC      100F'0'_F1£
CNAMES    DS      20F'0'_F1£
CHTAB     DC      CL180' '_F1£
          DC      CL180' '_F1£
          DC      CL180' '_F1£
LOOPCOM   DC      CL180' '_F1£
          DC      CL180' '_F1£
          DC      CL180' '_F1£
LOOPSTK   DC      16H'0'_F1£
LOOPNO    DC      16H'0'_F1£
XTAB      DC      20H'0'_F1£
STRA      DC      CL80' '_F1£
XCHRS     DC      CL180' '
          DC      CL180' '
          DC      CL40' '
FULL      DC      F'0'_F1£
ST1       DC      F'0'_F1£
ST2       DC      F'0'_F1£
FIN       DC      X'C6C9D540'_F1£
DMP       DC      X'C4E4D4D7'_F1£
BL        DC      CL80' '
CLC       DS      H                    *  SIMULATE A CLC INSTRUCTION
          ST      0,0(0,2)_F1£
          ORG     CLC_F1£
          ST      0,0(0,1)_F1£
          ORG     CLC_F1£
          DC      X'D500'_F1£
          ORG     CLC+6_F1£
TABLE     DC      H'40,-1,-1,-1,-1,0,14,49,-1,166'_F1£
```

```
            CH          6,=H'4'_F1£
            BL          LOADIT1+2_F1£
£

FETCH NAME FROM STRING.
            IC          3,STRING(10)_F1£
            AH          10,=H'1'_F1£
            STC         3,FULL(4)_F1£
            AH          4,=H'1'_F1£
            BCT         6,XL123_F1£
£

IF LOAD _ ELSE _.
            CLI         0(4),C'L'_F1£
            BNE         _20_F1£
GREG 4 + =H'1'£
            CLI         0(4),C' '_F1£
IF _10 = CONTINUE SKIP 2£
            BE          _10_F1£
SKIP 1£
            BNE         _20_F1£
£

BRANCH ON REG _ TO _.
            BCT         _10,_20_F1£
£

SWAP UNITS.
            SR          2,2_F1£
            LA          1,RUNIT_F1£
            ST          1,PARAM(2)_F1£
            AH          2,=H'4'_F1£
            LA          1,WUNIT_F1£
            ST          1,PARAM(2)_F1£
            LA          1,PARAM_F1£
            LR          4,13_F1£
            LA          13,SAVE_F1£
            LR          2,15_F1£
            L           15,=A(SWAPIT)_F1£
            BALR        14,15_F1£
            LR          15,2_F1£
```

```
          LR      13,4_F1£
          B       CHEKOUT          *  COMMAND EXECUTED._F1£
£

NUM NOT FOUND.
          A       3,=X'40000000'_F1£
NONUM     LH      4,=H'4'          *  ALPHA CHARS ON SEQ CARD._F
          L       1,BL_F1£
          ST      1,FULL_F1£
          LA      3,0_F1£
          LA      1,0_F1£
N2        IC      3,CARD(2)        *  READ AND STORE ALPHA CHARS
          STC     3,FULL(1)_F1£
          CH      3,=H'107'_F1£
          BE      NNAME_F1£
          AH      1,=H'1'_F1£
          AH      2,=H'1'_F1£
          SH      4,=H'1'_F1£
          CH      4,=H'0'_F1£
          BL      ERR35_F1£
          B       N2_F1£
NNAME     CH      4,=H'0'_F1£
          BE      RET_F1£
          L       3,FULL_F1£
          SRL     3,8_F1£
          BCT     4,NNAME+12_F1£
          ST      3,FULL_F1£
RET       BR      14_F1£
£

RESET.
RESET     SR      2,2_F1£
          LH      1,WUNIT_F1£
          ST      1,UNIT_F1£
          LA      1,UNIT_F1£
          ST      1,PARAM(2)_F1£
          LA      1,PARAM_F1£
          LR      4,13_F1£
          LA      13,SAVE _F1£
          LR      2,15_F1£
          L       15,=A(FILSRT)_F1£
          BALR    14,15_F1£
          LR      15,2_F1£
```

```
£


DUMPCHECK.
          L       1,CARD_F1£
          C       1,DMP_F1£
          BE      DUMPS              *  FOR PROGRAMMER USE ONLY_F1
£


ENDALL.
_F0£
££
                  END OF MACROS
                  ************
```

LISTING OF MACRO GENERATING STATEMENTS

```
START.
LABEL READ.
DUMPCHECK.
IF LDL JUMP TO LFDNUM1.
UNIT = =H'5'.
READ IN TO CARD.
COPY TO COPYEND.

    (INSERTED HERE IS THE CODING FOR THE COMMAND ANALYSER
     AND ERROR HANDLING ROUTINE)

COPYEND
LABEL DISPLAY.
IF IPP H SFP JUMP TO DISP1.
IF OPP NH SFP JUMP TO DISP2.
LABEL DISP1.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
UNIT = RUNIT.
READ IN TO ILB.
LABEL DISP2.
DISPLAY.
LABEL LOADIT1.
FETCH NAME,
LABEL LOADIT.
REG 6 = 0.
LABEL LCHECK.
IF LNAMES(6) E STRING JUMP TO LFDNUM.
GREG 6 + =H'8'.
IF REG6 H LNP JUMP TO ERR34.
JUMP TO LCHECK.
LABEL LFDNUM.
GREG 6 + =H'4'.
STNO = LNAMES(6).
GREG 6 + =H'2'.
LCP1 = LNAMES(6).
IF REG5 NE =H'17' JUMP TC LFDNUM1.
```

```
LDLI = =H'1'.
LABEL LFDNUM1.
IF STNO E =H'0' JUMP TO LDEND.
MOVE BLANKS TO CARD.
REG 6 = 0.
REG 6 = LCP1.
NUM = LCPOS(6).
GREG 6 + =H'2'.
NO = LCPOS(6).
GREG 6 + =H'2'.
LCP1 = REG 6.
REG 6 = NO.
MOVE NUM CHARS FROM CHTAB(6) TO CARD,NOKEEP.
LDL = =H'1'.
STNO = STNO - =H'1'.
IF LDLI E =H'1' JUMP TO SPINS1.
JUMP TO READX.
LABEL LDEND.
LDL = =H'0'.
JUMP TO CHEKOUT.
LABEL SPINS1.
IF CARD E FIN JUMP TO CHEKOUT.
FIND LAST CHAR.
REG 2 = 0.
TABCHECK.
UNIT = WUNIT.
WRITE OUT STRING.
IF STNO H =H'0' JUMP TO LFDNUM1.
LDLI = =H'0'.
JUMP TO SPINS.
LABEL STORE.
IF STLEN H =H'4' JUMP TO ERR35.
IF LINE JUMP TO STLINE.
IF CHP H =H'159' JUMP TO ERR30.
REG = CTP + NUM.
IF REG H =H'499' JUMP TO ERR30.
REG 6 = 0.
LABEL NMCHK.
IF CNAMES(6) E STRING JUMP TO DUPNAM.
GREG 6 + =H'8'.
IF REG6 NH CHP JUMP TO NMCHK.
REG 4 = CHP.
CNAMES(4) = STRING.
REG 6 = CHP.
GREG 6 + =H'4'.
```

```
CNAMES(6) = NUM.
GREG 6 + =H'2'.
CNAMES(6) = CTP.
GREG 6 + =H'2'.
CHP = REG 6.
LABEL SETUP.
IF REG5 E =H'26' JUMP TO STRCOM.
MOVE NUM CHARS FROM ILB TO CHTAB,NOKEEP.
CTP = CTP + NUM.
JUMP TO STREAD.
LABEL STRCOM.
UNIT = =H'5'.
READ IN TO CARD.
MOVE NUM CHARS FROM CARD TO CHTAB,KEEP.
LABEL STREAD.
IF NOT DUPN JUMP TO CHEKCUT.
CTP = FILLA.
JUMP TO CHEKOUT.
LABEL STLINE.
IF LNP H =H'159' JUMP TO ERR30.
REG = LCP + NUM.
REG = GREG + NUM.
IF REG H =H'199' JUMP TO ERR30.
REG 6 = 0.
LABEL LCHK.
IF LNAMES(6) E STRING JUMP TO ERR32.
GREG 6 + =H'8'.
IF REG6 L LNP JUMP TO LCHK.
REG 4 = C.
REG 4 = LNP.
LNAMES(4) = STRING.
REG 6 = LNP.
GREG 6 + =H'4'.
LNAMES(6) = NUM.
GREG 6 + =H'2'.
LNAMES(6) = LCP.
GREG 6 + =H'2'.
LNP = REG 6.
UNIT = RUNIT.
IF IPP NH SFP JUMP TO STRLIN.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TC OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
UNIT = RUNIT.
```

```
READ IN TO ILB.
SET POINTERS.
LABEL STRLIN.
IF REG5 NE =H'26' JUMP TO SLINE1.
SWAP ILB AND OLB.
UNIT = =H'5'.
READ IN TO ILB.
SET POINTERS.
LABEL SLINE1.
NO = =H'0'.
REG 6 = LCP.
LABEL SLINE.
LCPOS(6) = EP.
GREG 6 + =H'2'.
LCPOS(6) = CTP.
GREG 6 + =H'2'.
MOVE EP CHARS FROM ILB TO CHTAB,NOKEEP.
CTP = CTP + EP.
IF REG5 E =H'26' JUMP TO NOWRITE.
FILLA = UNIT.
UNIT = WUNIT.
WRITE OUT ILB.
UNIT = FILLA.
LABEL NOWRITE.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
SET POINTERS.
NO = NO + =H'1'.
IF REG L NUM JUMP TO SLINE.
LCP = REG 6.
IF REG5 NE =H'26' JUMP TO CHEKOUT.
MOVE =H'80' CHARS FROM ILB TO CARD,NOKEEP.
SWAP ILB AND OLB.
JUMP TO READX.
LABEL DUPNAM.
GREG 6 + =H'4'.
FILL = REG 5.
FILLA = CTP.
IF CNAMES(6) H NUM JUMP TO ERR31.
JUMP TO ERR32.
LABEL STRESET.
DUPN = =H'1'.
REG 5 = FILL.
CNAMES(6) = NUM.
GREG 6 + =H'2'.
```

```
CTP = CNAMES(6).
JUMP TO SETUP.
LABEL LOOPSTR.
IF REG5 E =H'21' JUMP TO LCCPND.
IF REG5 NE =H'20' JUMP TO LPCOMD.
LPSET = =H'1'.
LOOPCNT = LOOPCNT + =H'1'.
FIND NUM.
LABEL LPCOMD.
REG = LPCOMP + STLEN.
REG = GREG + =H'10'.
IF REG H LPMAX JUMP TO ERR29.
REG 10 = LPCOMP.
LOOPCOM(10) = REG 5.
GREG 10 + =H'2'.
IF NOT CHAR JUMP TO LINETST.
LOOPCOM(10) = =H'1'.
JUMP TO LPSTRING.
LABEL LINETST.
IF LINE JUMP TO LINEST.
IF REG5 NE =H'20' JUMP TO SINGST.
LOOPCOM(10) = NUM.
GREG 10 + =H'2'.
LABEL SINGST.
IF REG5 E =H'19' JUMP TO LPSTRING.
LOOPCOM(10) = =H'123'.
GREG 10 + =H'2'.
LABEL ENDST.
LPCOMP = REG 10.
IF LOOPCNT E =H'0' JUMP TO LPEXEC.
JUMP TO READ.
LABEL LINEST.
LOOPCOM(10) = =H'0'.
IF STLEN NE =H'0' JUMP TO LPSTRING.
IF REG5 E =H'17' JUMP TO SPCASE.
LABEL LPSTRING.
GREG 10 + =H'2'.
IF REG5 E =H'25' JUMP TO NUMSET.
IF REG5 E =H'26' JUMP TO NUMSET.
IF STLEN E =H'0' JUMP TO NUMSET.
LABEL KEEPST.
LOOPCOM(10) = STLEN.
REG 6 = 0.
LABEL LPCOM.
GREG 10 + =H'2'.
```

```
LOOPCOM(10) = STRING(6).
GREG 6 + =H'2'.
IF REG6 L STLEN JUMP TO LPCOM.
LABEL LPCOMF.
GREG 10 + =H'2'.
LOOPCOM(10) = =H'123'.
GREG 10 + =H'2'.
IF LINSRT JUMP TO SPCASE1.
JUMP TO ENDST.
LABEL SPCASE.
LINSRT = =H'1'.
GREG 10 + =H'2'.
LOOPCOM(10) = =H'123'.
GREG 10 + =H'2'.
UNIT = =H'5'.
LABEL SPCASE1.
LPCOMP = REG 10.
READ IN TO STRING.
REG 10 = LPCOMP.
IF STRING E FIN JUMP TO SPEND.
ADDRESS REG 4 = STRING.
GREG 4 + =H'79'.
LABEL SPCOMP.
IF NOT SPACE JUMP TO KEEPST2.
GREG 4 - =H'1'.
JUMP TO SPCOMP.
LABEL KEEPST2.
GREG 4 - =A(STRING).
GREG 4 + =H'1'.
STLEN = REG 4.
JUMP TO KEEPST.
LABEL SPEND.
STLEN = =H'3'.
LINSRT = =H'0'.
JUMP TO KEEPST.
LABEL NUMSET.
LOOPCOM(10) = NUM.
IF REG5 E =H'25' JUMP TO KEEPST1.
IF REG5 E =H'26' JUMP TO KEEPST1.
JUMP TO LPCOMF.
LABEL KEEPST1.
GREG 10 + =H'2'.
JUMP TO KEEPST.
LABEL LOOPND.
LOOPCNT = LOOPCNT - =H'1'.
```

```
IF REG L =H'0' JUMP TO ERR19.
JUMP TO LPCOMD.
LABEL LPEXEC.
LPEX = =H'1'.
LPSET = =H'0'.
LSTP = SAME.
LPCOMP = SAME.
LABEL LPEX1.
IF LDL JUMP TO LFDNUM1.
REG 10 = LPCOMP.
REPL = =H'0'.
DELETE = SAME.
NUM = =H'1'.
MOVE BLANKS TO STRING.
LABEL LPEX2.
REG 5 = LCOPCOM(10).
GREG 10 + =H'2'.
IF REG5 E =H'20' JUMP TO UPSTK.
IF REG5 E =H'21' JUMP TO DNSTK.
REG 6 = LOOPCOM(10).
GREG 10 + =H'2'.
IF REG6 E =H'123' JUMP TO ENDINS.
IF REG6 E =H'1' JUMP TO STCHAR.
LINE = =H'1'.
CHAR = =H'0'.
JUMP TO NXTVAL.
LABEL STCHAR.
CHAR = =H'1'.
LINE = =H'0'.
LABEL NXTVAL.
STLEN = LCOPCOM(10).
NUM = SAME.
GREG 10 + =H'2'.
IF REG E =H'123' JUMP TO ENDINS.
REG 6 = 0.
IF REG5 E =H'25' JUMP TO *+12.
IF REG5 NE =H'26' JUMP TO STINS.
STLEN = LOOPCOM(10).
GREG 10 + =H'2'.
LABEL STINS.
REG 1 = LCOPCOM(10).
GREG 10 + =H'2'.
IF REG E =H'123' JUMP TO ENDINS.
STRING(6) = REG 1.
GREG 6 + =H'2'.
```

```
JUMP TO STINS.
LABEL ENDINS.
LPCOMP = REG 10.
IF STLEN NE =H'123' JUMP TO EXINS.
IF REG5 NE =H'17' JUMP TO EXINS.
LINSRT = =H'1'.
STLEN = =H'0'.
LABEL EXINS.
JUMP TO TBRANCH.
LABEL UPSTK.
REG 1 = LOOPCOM(10).
REG 6 = LSTP.
GREG 10 + =H'4'.
LOOPSTK(6) = REG 10.
LOOPNO(6) = GREG - =H'1'.
GREG 6 + =H'2'.
LSTP = REG 6.
JUMP TO LPEX2.
LABEL DNSTK.
REG 6 = LSTP.
GREG 6 - =H'2'.
IF LOOPNO(6) E =H'0' JUMP TO NUMZ.
LOOPNO(6) = GREG - =H'1'.
REG 10 = LOOPSTK(6).
JUMP TO LPEX2.
LABEL NUMZ.
LSTP = REG 6.
IF REG6 E =H'0' JUMP TO LPABS.
GREG 10 + =H'2'.
JUMP TO LPEX2.
LABEL LPABS.
LPEX = =H'0'.
LPCOMP = SAME.
JUMP TO READ.
LABEL FIELDINS.
FIND NUM.
IF NUM NH =H'0' JUMP TO *+12.
NUM = GREG - =H'1'.
NO = NUM.
FIND NUM.
IF NUM H =H'80' JUMP TO ERR3.
IF NO NL =H'0' JUMP TO *+12.
NO = SFP.
IF NUM NL =H'0' JUMP TO *+12.
NUM = EFP.
```

```
IF NO NL NUM JUMP TO ERR23.
SFP = NO.
EFP = NUM.
IF OPP H =H'0' JUMP TO FMV.
IF IPP NH =H'0' JUMP TO FSET.
LABEL FMV.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
SWAP ILB AND OLB.
LABEL FSET.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL TABS.
TAB = =H'1'.
FIND CHAR.
REG 6 = C.
REG 4 = C.
LABEL NUMGET.
FIND NUM.
IF NUM NL =H'0' JUMP TO TESTOR.
NUM = TABF(6).
JUMP TO LOADNO.
LABEL TABOFF.
TAB = =H'0'.
JUMP TO CHEKOUT.
LABEL TESTOR.
IF NUM NH NO JUMP TO ERR23.
LABEL LOADNO.
TABT(6) = SAME.
GREG 6 + =H'2'.
IF SPACE JUMP TO LASTNO.
IF REG6 H =H'16' JUMP TO ERR24.
NO = NUM.
JUMP TO NUMGET.
LABEL LASTNO.
IF REG6 H =H'16' JUMP TO TABFS.
LABEL SET80.
TABT(6) = =H'80'.
GREG 6 + =H'2'.
IF REG6 NH =H'16' JUMP TO SET80.
LABEL TABFS.
REG 6 = C.
TABF(6) = TABT(6).
GREG 6 + =H'2'.
IF REG6 NH =H'16' JUMP TO TABFS+4.
```

```
JUMP TO CHEKOUT.
LABEL EXCHNG.
EXCH = =H'1'.
XLD = =H'0'.
IF STLEN NH =H'2' JUMP TO X1.
ADDRESS REG 4 = STRING.
IF LOAD CONTINUE ELSE X1.
XLD = =H'1'.
LABEL X1.
IF STLEN2 NH =H'2' JUMP TO XLCAD.
ADDRESS REG 4 = STRING.
GREG 4 + STLEN.
IF LOAD CONTINUE ELSE XLCAD.
IF XLD E =H'1' JUMP TO *+16.
XLD = =H'2'.
JUMP TO XLOAD.
XLD = =H'3'.
LABEL XLOAD.
OLDST = STLEN.
IF XLD E =H'0' JUMP TO XCONT.
IF XLD E =H'2' JUMP TO XL2.
REG 14 = 0.
REG 6 = STLEN.
REG 10 = =H'2'.
LABEL XLOAD1.
GREG 6 - =H'2'.
IF REG6 H =H'4' JUMP TO ERR35.
FULL = BL.
REG 4 = 0.
LABEL XL123.
FETCH NAME FROM STRING.
REG 6 = 0.
LABEL XSEARCH.
IF CNAMES(6) E FULL JUMP TO XFND.
GREG 6 + =H'8'.
IF REG6 H CHP JUMP TO ERR33.
JUMP TO XSEARCH.
LABEL XFND.
GREG 6 + =H'4'.
NO = CNAMES(6).
GREG 6 + =H'2'.
REG 6 = CNAMES(6).
MOVE NO CHARS FROM CHTAB(6) TO CARD(14),NOKEEP.
IF XLD E =H'1' JUMP TO XL1
IF XLD E =H'2' JUMP TO XSTRING.
```

```
STLEN = NO.
XLD = =H'2'.
LABEL XL3,
REG 14 = STLEN.
REG 6 = STLEN2.
REG 10 = OLDST.
GREG 10 + =H'2'.
JUMP TO XLOAD1.
LABEL XL2.
MOVE STLEN CHARS FROM STRING TO CARD,NOKEEP.
JUMP TO XL3.
LABEL XL1.
STLEN = NC.
REG 14 = STLEN.
REG 6 = OLDST.
MOVE STLEN2 CHARS FROM STRING(6) TO CARD(14),NOKEEP.
JUMP TO XSTRING+8.
LABEL XSTRING.
STLEN2 = NO.
NO = STLEN + STLEN2.
MOVE NO CHARS FROM CARD TO STRING,NOKEEP.
LABEL XCONT.
REG 6 = 0.
REG 6 = XTP.
IF REG6 NL =H'60' JUMP TO ERR36.
REG 14 = 0.
REG 1 = 0.
XTAB(6) = XCP.
GREG 6 + =H'2'.
XTAB(6) = STLEN.
GREG 6 + =H'2'.
XTAB(6) = STLEN2.
GREG 6 + =H'2'.
XTP = REG 6.
NO = STLEN + STLEN2.
REG 14 = XCP.
XCP = XCP + NO.
IF REG H =H'400' JUMP TO ERR36.
MOVE NO CHARS FROM STRING TO XCHRS(14),NOKEEP.
NO = STLEN + STLEN2.
MOVE NO CHARS FROM STRING TO STRA,NOKEEP.
XON = =H'1'.
IPPS = OPP.
ADDRESS REG 10 = READ.
FULL = REG 10.
```

```
JUMP TO XDELA.
LABEL BACKSP.
IF LINE JUMP TO ERR20.
IF REG5 E =H'1' JUMP TO BACKBAN.
FILLA = EP.
FILL = OPP.
EP = GREG + STLEN.
OPP = =H'0'.
COMPARE OLB WITH STRING IF FOUND BACKBA.
OPP = FILL.
EP = FILLA.
JUMP TO ERR21.
LABEL BACKBA.
FILLA = OPP.
OPP = FILL.
LABEL BACKBAN.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
IPP = OPP.
SWAP ILB AND OLB.
OPP = =H'0'.
MOVE =H'80' CHARS FROM BL TO OLB,NOKEEP.
SET POINTERS.
IF REG5 E =H'1' JUMP TO BACKNO.
NO = FILLA.
IF REG5 E =H'3' JUMP TO *+8.
NO = GREG + STLEN.
OPP = =H'0'.
IPP = SAME.
MOVE NO CHARS FROM ILB TO OLB,KEEP.
JUMP TO CHEKOUT.
LABEL BACKNO.
NO = IPP - NUM.
IPP = =H'0'.
OPP = SAME.
IF IPP L SFP JUMP TO ERR22A.
MOVE NO CHARS FROM ILB TO OLB,KEEP.
JUMP TO CHEKOUT.
LABEL ERR22A.
NO = SFP.
IPP = =H'0'.
OPP = SAME.
MOVE NO CHARS FROM ILB TO OLB,KEEP.
JUMP TO ERR22.
LABEL COPYN.
```

```
IF REG5 E =H'34' JUMP TO COPLIN2.
IF LINE JUMP TO COPLIN.
REG = IPP + NUM.
IF REG H EFP JUMP TO ERR11.
LABEL COPYN1.
IF DELETE JUMP TO DEL1.
MOVE NUM CHARS FROM ILB TO CLB,KEEP.
JUMP TO CHEKOUT.
LABEL DEL1.
IPP = IPP + NUM.
JUMP TO CHEKOUT.
LABEL COPLIN.
REG 6 = NUM.
IF IPP H SFP JUMP TO COPLIN2.
IF OPP NH SFP JUMP TO CONT.
LABEL COPLIN2.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
IF ENDNL JUMP TO ERR15.
IF REG5 NE =H'34' JUMP TO CONT1.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL CONT1.
IF ENDNL JUMP TO CONT.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
LABEL CONT.
IF DELETE JUMP TO DEL2.
UNIT = WUNIT.
WRITE OUT ILB.
LABEL DEL2.
BRANCH ON REG 6 TO CONT1.
IF ENDNL JUMP TO SETEND.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL SETEND.
```

```
ENDF = =H'1'.
JUMP TO CHEKOUT.
LABEL COPABL.
IF LINE JUMP TO ABLINE.
NUM = EP - IPP.
IF REG5 E =H'8' JUMP TO BEFORE.
IF NUM NH =H'0' JUMP TO CHEKOUT.
JUMP TO COPYN.
LABEL BEFORE.
NUM = NUM - =H'1'.
IF REG NH =H'0' JUMP TO ERR13.
JUMP TO COPYN.
LABEL ABLINE.
IF OPP H SFP JUMP TO ABL1.
IF IPP NH SFP JUMP TO READL.
LABEL ABL1.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
IF REG5 E =H'8' JUMP TO BEFOL.
JUMP TO READ1.
LABEL READL.
IF REG5 E =H'8' JUMP TO OLBP.
IF DELETE JUMP TO READ1.
UNIT = WUNIT.
WRITE OUT ILB.
LABEL READ1.
UNIT = RUNIT.
READ IN TO ILB.
IF NOT EOF JUMP TO READL+8.
ENDF = =H'1'.
JUMP TO  CHEKOUT.
LABEL BEFOL.
UNIT = RUNIT.
READ IN TO OLB.
IF EOF JUMP TO ERR14.
JUMP TO SWAP.
LABEL OLBP.
UNIT = RUNIT.
READ IN TO OLB.
IF EOF JUMP TO POINTSET.
IF DELETE JUMP TO SWAP.
UNIT = WUNIT.
WRITE OUT ILB.
```

```
LABEL SWAP.
SWAP ILB AND OLB.
JUMP TO OLBP.
LABEL POINTSET.
ENDNL = =H'1'.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL COPABS.
IF REG5 E =H'30' JUMP TO *+16.
IF LINE JUMP TO COPLINS.
REG = IPP + STLEN.
IF REG NH EFP JUMP TO STLENCK.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
JUMP TO CARRYON.
LABEL STLENCK.
COMPARE ILB WITH STRING IF FOUND BEFCHK.
LABEL CARRYON.
IF DELETE JUMP TO *+18.
UNIT = WUNIT.
WRITE OUT OLB.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
SET POINTERS.
JUMP TO STLENCK.
LABEL BEFCHK.
IF REG5 E =H'30' JUMP TO FINDS.
IF REG5 E =H'6' JUMP TO CHEKOUT.
IF DELETE JUMP TO DEL3.
MOVE STLEN CHARS FROM ILB TC CLB,KEEP.
JUMP TO CHEKOUT.
LABEL FINDS.
IPP = =H'0'.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL DEL3.
IPP = IPP + STLEN.
JUMP TO CHEKOUT.
LABEL COPLINS.
IF OPP H SFP JUMP TO CPL1.
IF IPP NH SFP JUMP TO GCCN.
LABEL CPL1.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TC OLB,NOKEEP.
```

```
UNIT = WUNIT.
WRITE OUT OLB.
JUMP TO GETLIN.
LABEL GOON.
COMPARE ILB WITH STRING IF FOUND BEFLCK.
SPACE TEST.
IF DELETE JUMP TO GETLIN.
UNIT = WUNIT.
WRITE OUT ILB.
LABEL GETLIN.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
SET POINTERS.
JUMP TO GOON.
LABEL BEFLCK.
IPP = SFP.
IF REG5 E =H'6' JUMP TO CHEKOUT.
IF DELETE JUMP TO *+18.
UNIT = WUNIT.
WRITE OUT ILB.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO CHEKOUT.
SET POINTERS.
JUMP TO CHEKOUT.
LABEL DELCOM.
DELETE = =H'1'.
IF REPL JUMP TO COPYN.
GREG 5 - =H'6'.
JUMP TO TBRANCH.
LABEL EXIT.
UNIT = WUNIT.
IF ENDF JUMP TO RESET.
IF OPP H SFP JUMP TO EXIT1.
IF IPP NH SFP JUMP TO ILBCUT.
LABEL EXIT1.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO CLB,NOKEEP.
WRITE OUT OLB.
IF EOF JUMP TO RESET.
JUMP TO SSTOP.
LABEL ILBOUT.
WRITE OUT ILB.
IF ENDNL JUMP TO RESET.
```

```
LABEL SSTOP.
UNIT = RUNIT.
READ IN TO ILB.
UNIT = WUNIT.
IF EOF JUMP TO RESET.
WRITE OUT ILB.
JUMP TO SSTOP.
LABEL INSERT.
IF LINE JUMP TO INLINE.
IF REG5 E =H'17' JUMP TO INON.
IF REG5 E =H'22' JUMP TO INON.
STLEN = NUM.
LABEL INON.
IF REPL JUMP TO INSREPL.
NUM = EP - IPP.
NO = EFP - OPP.
REG = GREG - NUM.
IF REG NH STLEN JUMP TC ERR18.
LABEL INSREPL.
IF LINE JUMP TO TBS1.
MOVE STLEN CHARS FROM STRING TC CLB,NOKEEP.
OPP = OPP + STLEN.
NUM = STLEN.
IF REPL JUMP TO DELCOM.
JUMP TO CHEKOUT.
LABEL INLINE.
IF OPP H SFP JUMP TO INL1.
IF IPP NH SFP JUMP TO INSL.
LABEL INL1.
IF ENDF JUMP TO INSL.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
IPP = =H'0'.
OPP = SAME.
IF EOF JUMP TO INSLX.
UNIT = RUNIT.
READ IN TO ILB.
SET POINTERS.
IF NOT EOF JUMP TO INSL.
LABEL INSLX.
ENDF = =H'1'.
LABEL INSL.
REG 6 = NUM.
```

```
IF REG5 E =H'18' JUMP TO INOUT.
IF REG5 E =H'23' JUMP TO INOUT.
IF STLEN E =H'0' JUMP TO SPINS.
REG 6 = =H'1'.
LABEL INOUT.
UNIT = WUNIT.
WRITE OUT STRING.
BRANCH ON REG 6 TO INOUT.
IF REPL JUMP TO DELCOM.
JUMP TO CHEKOUT.
LABEL SPINS.
IF LINSRT JUMP TO LPSPNS.
UNIT = =H'5'.
READ IN TO CARD.
LABEL LPSP.
ADDRESS REG 4 = CARD.
IF LOAD LOADIT1 ELSE SPINSA.
LABEL SPINSA.
IF CARD E FIN JUMP TO CHEKOUT.
FIND LAST CHAR.
REG 2 = 0.
TABCHECK.
UNIT = WUNIT.
WRITE OUT STRING.
JUMP TO SPINS.
LABEL LPSPNS.
MOVE =H'80' CHARS FROM BL TO CARD,NOKEEP.
REG 10 = 0.
REG 10 = LPCOMP.
STLEN = LOOPCOM(10).
GREG 10 + =H'2'.
REG 6 = 0.
LABEL LPSP1.
CARD(6) = LOOPCOM(10).
GREG 6 + =H'2'.
GREG 10 + =H'2'.
IF REG6 L STLEN JUMP TO LPSP1.
GREG 10 + =H'2'.
LPCOMP = REG 10.
JUMP TO LPSP.
LABEL REPLACE.
REPL = =H'1'.
JUMP TO INSERT.
LABEL STARTER.
IF LINE JUMP TO STARTL.
```

```
NO = EP - IPP.
MOVE NO CHARS FROM ILB TC OLB,NOKEEP.
SWAP ILB AND OLB.
IPP = SFP.
OPP = SAME.
JUMP TO CHEKOUT.
LABEL STARTL.
IF ENDF JUMP TO SWAPUNS.
IF OPP H SFP JUMP TO STL1N.
IF IPP NH SFP JUMP TO TRYIT.
LABEL STL1N.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TC OLB,NOKEEP.
UNIT = WUNIT.
WRITE OUT OLB.
IF EOF JUMP TO SWAPUNS.
LABEL STR2.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO SWAPUNS.
LABEL GETON.
UNIT = WUNIT.
WRITE OUT ILB.
JUMP TO STR2.
LABEL TRYIT.
IF EOF JUMP TO *+8.
JUMP TO GETON.
UNIT = WUNIT.
WRITE OUT ILB.
LABEL SWAPUNS.
ENDF = =H'0'.
ENDNL = SAME.
SWAP UNITS.
LABEL PRNSET.
IF PRINT JUMP TO *+16.
PRINT = =H'1'.
JUMP TO READ.
PRINT = =H'0'.
JUMP TO READ.
LABEL NEXCH.
XLD = =H'-1'.
ADDRESS REG 10 = CHEKOUT.
JUMP TO INEX1.
NUM NOT FOUND.
LABEL SEQNCE.
```

```
FIND NUM.
ST1 = FULL.
FIND NUM.
INC1 = NUM.
IF REG NE =H'0' JUMP TO N1.
STRING = ST1.
LABEL N1.
FIND NUM.
ST2 = FULL.
FIND NUM.
INC2 = NUM.
IF REG NE =H'0' JUMP TO SEQALL.
REG 4 = =H'4'.
STRING(4) = ST2.
LABEL SEQALL.
IF EOF JUMP TO ERR15.
ONEF = =H'0'.
IF ST1 NE =H'0' JUMP TO CONTSEQ.
IF INC1 NE =H'0' JUMP TO CONTSEQ.
ONEF = =H'1'.
LABEL CONTSEQ.
IF IPP H SFP JUMP TO LINCUT.
IF OPP NH SFP JUMP TO SEQCONT.
LABEL LINOUT.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
SWAP ILB AND OLB.
LABEL SEQCONT.
REG = EFP - =H'8'.
IF REG L SFP JUMP TO ERR16.
LABEL SEQCON1.
IF ONEF JUMP TO ONLY2.
IF INC1 E =H'0' JUMP TO ONLY2.
REG 4 = =H'3'.
FILLA = =H'0'.
STRING(3) = ST1.
ST1 = ST1 + INC1.
IF INC2 E =H'0' JUMP TO ONLY1.
LABEL ONLY2.
IF INC2 E =H'0' JUMP TO ONLY1.
REG 4 = =H'7'.
FILLA = =H'0'.
IF ONEF JUMP TO *+12.
FILLA = =H'4'.
STRING(3) = ST2.
```

```
ST2 = ST2 + INC2.
LABEL ONLY1.
IPP = EFP - =H'8'.
IF REG L SFP JUMP TO SEQ1.
NO = =H'8'.
REG 6 = C.
JUMP TO SEQ2.
LABEL SEQ1.
IPP = SFP.
NO = EFP - SFP.
REG 6 = =H'8'.
GREG 6 - NO.
LABEL SEQ2.
MOVE NO CHARS FROM STRING(6) TO ILB,NOKEEP.
UNIT = WUNIT.
WRITE OUT ILB.
UNIT = RUNIT.
READ IN TO ILB.
IF EOF JUMP TO ERR15.
JUMP TO SEQCON1.
LABEL OUT1.
OUTPUT.
INPUT.
LABEL INEX.
IF EOF JUMP TO RETC.
LABEL INEX1.
FILLB = REG 6.
FULL = REG 10.
STL1 = STLEN.
STL2 = STLEN2.
IF DELETE JUMP TO INRET.
XPTR = =H'0'.
IF XLD L =H'0' JUMP TO XCHK.
SET POINTERS.
LABEL XCHK.
REG 6 = XPTR.
IF REG6 NL XTP JUMP TO XOUT.
LABEL XCHK1.
XPT1 = XTAB(6).
IF REG L =H'0' JUMP TO XNP.
GREG 6 + =H'2'.
STLEN = XTAB(6).
GREG 6 + =H'2'.
STLEN2 = XTAB(6).
GREG 6 + =H'2'.
```

```
XPTR = REG 6.
IF XLD NL =H'0' JUMP TO XCOMPR.
IF STLEN NE STL1 JUMP TO XCHK.
LABEL XCOMPR.
REG 4 = XPT1.
NO = STLEN + STLEN2.
MOVE NO CHARS FROM XCHRS(4) TO STRA,NOKEEP.
LABEL XSTCHK.
IF XLD NL =H'0' JUMP TO XDELA.
ADDRESS REG 1 = STRING.
GREG 1 + SFP.
JUMP TO XDELA+6.
COMPARE XCHRS(4) WITH STRA IF FOUND XDEL.
LABEL XC2.
SWAP ILB AND OLB.
IF XON JUMP TO XON1.
SET POINTERS.
JUMP TO XCHK.
LABEL XNP.
GREG 6 + =H'6'.
IF REG6 NL XTP JUMP TO XOUT.
JUMP TO XCHK1.
LABEL XDEL.
IF XLD NL =H'0' JUMP TO XFD.
REG 6 = XPTR.
GREG 6 - =H'6'.
XTAB(6) = XLD.
XLD = =H'0'.
REG 6 = 0.
LABEL XINR.
REG 1 = XTAB(6).
IF REG NL =H'0' JUMP TO INRET1.
GREG 6 + =H'6'.
IF REG6 L XTP JUMP TO XINR.
EXCH = =H'0'.
XTP = SAME.
XCP = SAME.
JUMP TO INRET1.
LABEL INRET.
MOVE =H'80' CHARS FROM BL TO OLB,NOKEEP.
LABEL INRET1.
STLEN = STL1.
STLEN2 = STL2.
REG 10 = FULL.
REG 6 = FILLB.
```

```
JUMP TO RETC.
LABEL XOUT.
IF XLD L =H'0' JUMP TO ERR12A,
IF IPP NH SFP JUMP TC INRET.
NO = =H'80' - IPP.
MOVE NO CHARS FROM ILB TO OLB,NOKEEP.
SWAP ILB AND OLB.
IF XON JUMP TO XON1.
SET POINTERS.
JUMP TO INRET,
LABEL XON1.
IPP = =H'0'.
OPP = SAME.
MOVE IPPS CHARS FROM ILB TO CLB,KEEP.
XON = =H'0'.
JUMP TO INRET.
LABEL XFD.
IPP = IPP + STLEN.
REG 4 = STLEN.
MOVE STLEN2 CHARS FROM STRA(4) TC OLB,KEEP.
IF IPP NL EP JUMP TO XC2-28.
JUMP TO XSTCHK.
LABEL ERR12A.
XLD = =H'0'.
JUMP TO ERR12.
MOVES.
POINTERS.
NUMBER.
DUMP.
TABSTORE.
LABEL CHECK1.
IF NOT LPEX JUMP TC CHEKOUT.
LPEX = =H'0'.
LPCOMP = SAME.
LOOPCNT = SAME.
LABEL CHEKOUT.
LINSRT = =H'0'.
IF NOT DELETE JUMP TO STLAST.
IF REPL JUMP TO STLAST.
GREG 5 + =H'6'.
LABEL STLAST.
LASTINS = REG 5.
IF REG5 NE =H'29' JUMP TC LCOPCH.
IF LINE JUMP TO NEWFILE.
LABEL LOOPCH.
```

```
IF LPEX JUMP TO LPEX1.
JUMP TO READ.
RESET.
STOP.
CONSTANTS.
ENDALL.
```

END OF M.G.S.LISTING
****