A SIMULATOR OF MULTIPLE INTERACTIVE USERS TO DRIVE A TIME-SHARED COMPUTER SYSTEM

by

HOWARD JACQUES GREENBAUM

S.B., Massachusetts Institute of Technology
(1967)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

October, 1968

Signature of Author _____
Deparment of Electrical Engineering, October 1968

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Graduate Students

A SIMULATOR OF MULTIPLE INTERACTIVE USERS TO DRIVE A TIME-SHARED COMPUTER SYSTEM

by

HOWARD JACQUES GREENBAUM

Submitted to the Department of Electrical Engineering
on October 1968 in partial fulfillment of the
requirements for the Degree of Master of Science.

ABSTRACT

In the construction and maintenance of a Time-Shared Computer System the need arises for a tool which can provide a controlled, repeatable environment for the purpose of making performance measurements.

This thesis describes the use of a small second computer to simulate the actions of multiple interactive users over individual communication lines. Each simulated user exhibits responses similar to those of a "normal" interactive user. Accordingly, the "Simulator" recognizes and verifies responses transmitted to it by the system being tested. The Simulator also emulates a "think time" corresponding to a normal user's think time between typing lines on the console. Text corresponding to a user's console input, as well as control information regarding think time simulation and verification of responses from the system being tested, are retrieved from prepared scripts which have been pre-stored on the small computer's magnetic disc unit.

The programming package for the Simulator System has the capability of simulating up to 12 users. For the purpose of this thesis, however, only four users are simulated. The Simulator System is intended to be used to test the M.I.T. CTSS and MULTICS (Time-Shared Computer Systems). However, it is designed to be adaptable for testing most time shared computer systems having serial character oriented input/output over communications lines interfacing with 103A compatible data sets.

THESIS SUPERVISOR: Jerome Howard Saltzer
TITLE: Assistant Professor of Electrical Engineering

ACKNOWLEDGEMENT

I wish to thank my advisor, J. H. Saltzer for his guidance and for his interest in me. His numerous suggestions were particularly beneficial.

I would also like to thank T. Skinner, J. Grochow, N. Morris, S. Dunten, and C. Garmon of Project MAC for their help with the many technical problems which plagued me during the implementation of this thesis.

Thanks are also due to F. Hennie for his editorial comments regarding several segments of this document, and to Marsha Baker for the careful job of typing this manuscript.

Special thanks are due to my wife, Maureen, for the exceptional patience, understanding, unselfish devotion, and fortitude she displayed throughout the long and arduous course of this thesis. I would also like to thank her for the help she extended to me in solving many of the programming impasses reached during the implementation of the Simulator System.

This empty page was substituted for a blank page in the original document.

TABLE OF CONTENTS

A SIMULATOR OF MULTIPLE INTERACTIVE USERS TO DRIVE A TIME-SHARED COMPUTER SYSTEM

by

HOWARD JACQUES GREENBAUM

S.B., Massachusetts Institute of Technology
(1967)

This empty page was substituted for a blank page in the original document.

I. INTRODUCTION

A. Summary

The ability to evaluate the performance of a Time-Shared Computer System (TSS) plays an important role in the implementation and maintenance of such a system. This thesis describes a mechanism to simulate multiple interactive users typing from prepared scripts to provide a controlled, repeatable environment for the TSS being tested in order that meaningful performance measurements may be made. Simulation of multiple, interactive users is performed by a "Simulator System" implemented on a small, inexpensive second computer, the PDP-8, produced by the Digital Equipment Corporation.

In order to simulate normal, human, interactive users accurately, the Simulator recognizes and verifies responses transmitted to it by the Time-Shared Computer System being tested. In addition, the Simulator System emulates the human user's think time between typing lines on his console. The "Scripts", controlling the actions of the simulated users, contain the information necessary to perform the checking and verification of the responses from the TSS being tested, as well as the think time emulation.

The scripts for the Simulator are prepared in advance by the experimenter. They are encoded into a special script "language", input into the PDP-8 and converted to magnetic tape files using the PDP-8 symbolic text editor STECO*. In this manner an experimenter may produce a library of scripts on magnetic tape.

*A version of the TECO symbolic text editor, designed by Dan Edwards.

Subsequently, the experimenter uses the SCRIPT LOADER program of the Simulator System to move designated script files on magnetic tape to the PDP-8 disc unit. The SCRIPT LOADER program also reformats the scripts and performs some error detection on input script formatting.

Once the chosen scripts have been placed on the disc, simulation of users commences, controlled by the scripts on the disc. The simulation progresses until the scripts of all simulated users have been exhausted, or until a "fatal" error occurs. In either case, a message is posted at the PDP-8 on-line console giving the reason for termination.

During the simulation, a copy of all text transmitted to and from the Simulator is kept on magnetic tape. This tape copy may be listed or analyzed by one of several other programs provided as part of the Simulator System.

The initial implementation of the Simulator System, as described in this thesis, can simulate up to twelve (12) users. However, each simulated user requires an individual telephone communications line, and the PDP-8 requires a special hardware interface for each additional communications line. The initial implementation of the Simulator System will therefore support a maximum of only four users. Additional users may be simulated by increasing the number of communications lines and interfaces, as described in Appendix B.

The System is designed to simulate users on either the M.I.T. IBM 7094 Compatible Time-Shared System (CTSS)[1]; or the M.I.T. GE 645 MULTiplexed Information and Computing Service (MULTICS)[2]. It is intended that the

System be flexible enough to simulate users on most Time-Shared Computer systems that use Bell Telephone type 103A compatible data-sets to transmit and receive console input and output.

At present, research into the problem of making performance measurements on Time-Shared Computer Systems by simulating user loads is being carried on at at least two other installations. Workers at the IBM Corporation are using an IBM System/360 Model 40 computer to simulate user loads on their System/360 Model 67 Time-Shared Computer System. Another effort is currently going on at the Carnegie Mellon University to test their IBM System/360 Model 67 Time-Shared Computer System by simulating user loads. At this time, no further information is available to the author regarding other efforts to simulate user loads to test Time-Shared Computer Systems.

B. Description of the Contents of this Document

The first half of Chapter II is divided into three sections.

1. An historical solution to the problems of making meaningful performance measurements on a Time-Shared Computer System.
2. A new approach to making performance measurements on Time-Shared Computer Systems.
3. The mechanization of this new method using a remote computer.

The first portion deals with the overall problem of making performance measurements on a Time-Shared Computer System and discusses an approach previously used to make such measurements on M.I.T.'s CTSS. This method had the disadvantage that long periods of time were required to obtain meaningful formation, since the results were statistical in nature.

The second part describes an alternate approach to solving the problem of making meaningful performance measurements on a Time-Shared Computer System. Performance measurements can be made in a brief period of time if one controls the environment of the system being tested. In order to do this, one restricts the user community of the TSS being tested to a selected group of users, and provides each user with a prepared script to guide his actions at the TSS console. This procedure provides a controlled--and more important, repeatable--environment for

the TSS being tested, so that meaningful performance measurements can be made. The measurements may, then, be made in the relatively short period of time necessary for the restricted users to run their scripts.

The third section discusses the mechanization of this procedure using a PDP-8 computer to simultaneously simulate this controlled user community. Mechanization provides the advantages of the procedure just outlined, with the added advantages that the performance measurement experiment may be conducted by a single individual at the PDP-8, rather than many, and the procedure is made more precisely repeatable through the use of the second computer.

The third section also deals with the requirements of the scripts used to drive such a multiple user simulator. In addition to the text lines that a human user would normally type at his console, the script also includes a "verifier line" used by the Simulator to check and verify responses transmitted to it by the TSS being tested. The script also includes a "verifier time limit" and a "think time". The former allows the Simulator to check for normal operation of the TSS during simulation; the latter allows the Simulator to mimic the time spent in "head scratching" by a human user.

The second half of Chapter II is a brief description of the implementation of the "Simulator System". This section outlines the steps necessary to use the Simulator System.

Chapters III and IV deal with the use and implementation of the Simulator System. Chapter III describes the content and exact format of the scripts used to drive the Simulator. Chapter IV describes the implementation of the Simulator System. Included in this chapter are a description of the PDP-8, a description of the hardware interfaces added to the PDP-8 to accommodate the extra communications lines necessary for the Simulator, and a description of the programs constituting the Simulator System.

Chapter V describes two of the problems which still remain. The first is the problem of coping with transmission errors due to noise and dropout on the telephone communications lines. The second is the problem of proper design of the scripts used to drive the Simulator. These problems must be considered by an experimenter using the Simulator System.

One appendix, Appendix A, deserves mention in the body of this summary. This appendix is the user's manual for operation of the Simulator System, and provides a step by step guide to the operation of the System.

II. OVERVIEW OF THE USER SIMULATOR

A. Problems of Performance Measurement on Time-Shared Computer Systems

A Time-Shared Computer System is a computer system in which many users share a computer facility in seeming simultaneity. When implementing, testing or modifying such a system, it is valuable to be able to evaluate system performance. However, since such a system has a varied load of users simultaneously using the resources of the facility, and the number and work load of these users constantly changes, it is difficult to draw meaningful conclusions about system performance by looking at measurements made at any given instant.

1. An Historical Method of Performance Measurement

One method of ascertaining system performance is to take an average of relevant system parameters and meterings over a lengthy period of time. This approach is taken by Scherr[3] and Hastings[4] to amass data about the MIT CTSS. Their effort involved making minor modifications to the supervisor to meter various system functions. In taking their measurements they found that it was necessary to sample over long periods of time (on the order of weeks) to average out the effects of abnormal or distorted loads on the system, which occurred sporadically.

There are two disadvantages inherent in making such a measurement. First is the necessity for sampling the relevant meters over lengthy periods of time to achieve the desired averaging effect. Second, the measurements

are at the mercy of the user community currently loading the system. There is no real assurance that two such trials of the experiment will yield precisely the same results, even with a lengthy averaging period.

Clearly, if a system modification were made, it would take a lengthy averaging period again to obtain a new measure of the system performance. Therefore, one concludes, that this method does not lend itself readily to making measurements in a reasonably short period of time.

2. A New Approach to Performance Measurement

Another approach to the performance measurement problem is to completely control the actions of the user community. By restricting the computer facility to a special group of users for the duration of the experiment, and providing each of the group of users with a prepared script specifying exactly the console input to type, we would have, in effect, a controlled, repeatable experiment. If the system were to remain the same, any subsequent run using the same scripts would yield substantially the same results. If a system modification were made, the control group could run using the same prepared scripts both before and after the modification. By monitoring the relevant system parameters and meters during both runs, a meaningful measure of relative performance could be obtained in the short periods of time required for the runs.

Comparing the two strategies, one observes the following disadvantage of the latter scheme. The first scheme, though it takes a long period of time, guarantees that measurements taken on the Time-Shared Computer represent the parameters of the normally loaded system. The second scheme's accuracy depends heavily on the types of scripts used by the restricted group. If the scripts used by the special group of users do not impose a "normal" load on the system being tested, any measurements taken on the system being tested will not necessarily reflect normal system performance measurements. On the other hand, the second scheme allows an experimenter specifying the scripts to load the system being tested abnormally so as to make measurements not normally possible in the first scheme. In either case, whether the scripts represent a "normal" or "not-normal" load on the system being tested, the experimenter has a controlled, repeatable experiment.

3. Mechanization of the New Method of Performance Measurement

As described previously we wish a controlled, repeatable experimental environment to provide meaningful performance measurements of a Time-Shared Computer System (TSS) in a relatively short period of time. However, two problems still remain in designing this controlled experiment. The first is the synchronization of all the users so that they start in a coordinated fashion. The second is the elimination of the individual idiosyncrasies of the users as they type at their consoles. If, instead of having many human users taking part in the experiment, we use another computer to simulate the actions of these many users, we can then at least

standardize, if not eliminate these problems. The use of another computer allows precise timing and repeatability, as well as the convenience of a fully automated procedure.

We would, therefore, like to use a computer to simulate as closely as possible the actions of many users sitting at conventional consoles, typing from prepared scripts. The problem of actually specifying and performing the measurements on the TSS being tested will be considered in this thesis only insofar as it directly influences the basic Simulator System design. The mechanism to implement the above scheme, will be a general purpose tool adaptable to operate on most time-shared computer systems using Bell Telephone 103A (or 103A compatible) data sets to receive and transmit console input and output. The Simulator, as it will henceforth be called, is designed to test both the IBM 7094 CTSS and the General Electric 645 MULTICS configurations at M.I.T. Project MAC.

The two underlying objectives in the design of this Simulator are:

1. The simulator must accurately mimic the actions of "normal" interactive users as closely as possible.
2. The simulator must perturb the system being tested as little as possible.

In meeting objective #1 above, we would expect that the scripts that an automated simulator uses would contain more information than the script which would be given to a human participating in such an experiment as described in the introduction. In addition to the output text a human

user would normally type at his console, the Simulator needs the following types of information to accurately simulate the actions of its human counterpart:

1. A verifier line
2. A verifier time limit
3. A "think time" datum

These three items, explained in detail in the following paragraphs, plus the output text line, form the body of the script for the simulator. The exact formating of the scripts is described in a later chapter of this thesis.

The following example illustrates the functions of the three additional data.

This example assumes operation on M.I.T.'s CTSS. Assume also that an operator has dialed up the data set for a "user" on the Simulator, and the Simulator is to log the "user" onto CTSS. The Simulator should not commence "typing" its login command until it has received its "READY" message on the input telephone line for the user. The Simulator must have some way of determining that the "READY" line has been received. The "verification line" provides the information necessary to make this determination. This "verification line" is compared against every incoming line of text from the communications line for the particular user, until a "match" is found The verification line for this example might appear as "READY" to signify

that the simulator is not to proceed outputting its text line, the login command, until a line consisting of "READY" is received by the Simulator. Thus, if after the simulator is "dialed up" for a "user", CTSS were to transmit the following lines to the Simulator:

MAC7A1

USERS=22

SYSTEM WILL BE DOWN TONIGHT FROM 1800 TO 2100 HOURS

READY

the simulator would not commence transmitting its login command until the "READY" message were received.

However, what if the system response after dialing up were:

CTSS NOT IN OPERATION

and nothing more following? We would wish some sort of indication of this error condition. We therefore add another piece of information to the simulator script, namely a maximum amount of time we wish to allow to find such a match as the "READY" match in our example. When this time has been exceeded before finding a match, the Simulator would signal an error condition to the operator of the Simulator, since something has gone wrong. This setting of a maximum time limit before finding a match is called the "Verification time limit", item 2 above.

Finally, we make the following observation. A human interactive user sitting at his console, rarely types a response to the computer

immediately and at full console speed after his ready light comes on. In general, there is some time spent in "head scratching" while the user is sitting at his console.* A human user might do at least some of this "head scratching" in the middle of a long line, however, the Simulator does all its "head scratching" before transmitting a line. This time delay between verifying a response line and transmission of the next simulated user's output text line is called "think time" in this document. For example, if we wished the simulator to delay sending its login command for 20 seconds after verifying that the "READY" message has been received, we would need another piece of information, the "think time", in the script to signify this 20 second delay.

By implementing the Simulator on a remote computer, we can best attempt to realize the second objective, namely not perturbing the TSS being tested. In the present implementation, each simululated user is to be assigned his own communications line. Another scheme using one, two or several multiplexed communications lines was contemplated as a possible method of implementing the Simulator. This implementation would require no additional hardware, or at worst, considerably less than providing an individual communications line for each simulated user. However, this scheme would require modification of the system being tested in order that the information could be de-multiplexed. Upon further investigation, it was noted that such de-multiplexing could distort the performance of the system being tested. In the case of MULTICS, the system for which the

*Scherr[3] observed that think times of CTSS users in a certain period were exponentially distributed, with mean of 30 seconds.

Simulator is orginally intended, the distortion might make the resulting performance measurements of questionable worth. By providing each simulated user with his own dataphone line, the system being tested is required to act as it would with normal human users at teletypes. All manipulation relating to simulation is performed externally to the system being tested. It is expected that the system being tested will run as if human users were loading it.

B. Brief Description of the Implementation of the Simulator

The user Simulator is implemented on M.I.T.'s PDP-8 computer at Project MAC. The PDP-8 was chosen as the remote computer for the Simulator, since it is physically close to the two systems upon which it will initially be used (MULTICS and CTSS). Also it lends itself well to the hardware modifications necessary to accommodate the datasets required for each simulated user's telephone line.

The operation of the user Simulator consists of five distinct phases. The first phase is the design and encoding of the scripts to be used in each simulation. The experimenter desiring to use the Simulator must design his scripts so as to exercise the object computer in such a manner as to make performance measurements meaningful. The "caveat" mentioned before, namely that a simulation made with scripts not representing the load of normal users will produce distorted performance measurements, should be observed. Of course, this seeming problem could be looked at as a means to produce unusual loadings on a system for the purpose of special performance tests.

Once the scripts are designed, they must be encoded into a special format. This format is described in detail in Chapter III of this thesis. (See figure II-1).

The second phase of the Simulator consists of the routine inputting and editing of the encoded scripts using a special version of the PDP-8 symbolic text editor, TECO, to produce symbolic magnetic tape files. TECO is similar in its capabilities to CTSS's "EDL" symbolic text editor. It has the distinct advantage, however, that the text being edited is displayed on the

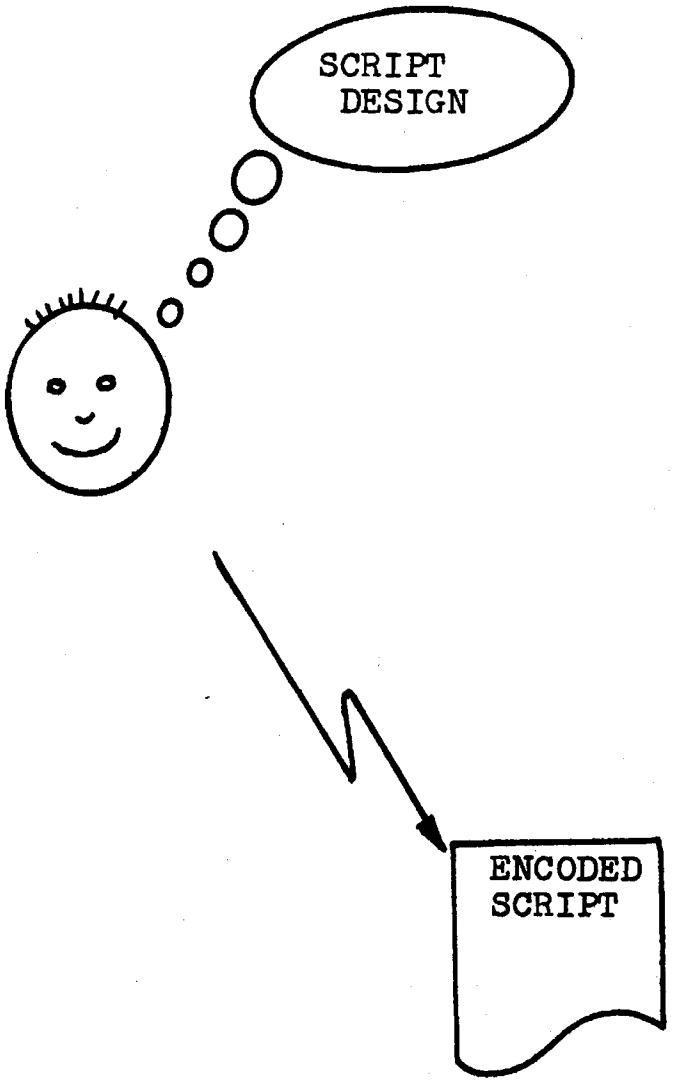


FIGURE II-1: Script design and encoding, as performed by experimenter.

PDP-8 CRT display unit, and hence is fully visible to the operator. (See figure II-2) The use of TECO is discussed in Appendix A of this thesis. A summary of the commands and their functions appear in Appendix C.

Once an experimenter has established a "library" of symbolic files on magnetic tape, he must choose those scripts he wishes to use for a particular simulation. During the third phase the experimenter uses the "SCRIPT LOADER" program to transfer the chosen script tape files onto the PDP-8's magnetic disc for subsequent use by the Simulator. (See figure II-3) The SCRIPT LOADER program performs four functions. The first is that it allows the experimenter to specify beforehand exactly which scripts he wishes to use for a given simulation, and make only these scripts available to the Simulator during simulation. The second function it has is its reformatting function. Reformatting eliminates much of the redundant and unnecessary control characters which are present in a script file, and thus makes the script more compact. This compactness speeds up the operation of the simulator program since these unnecessary characters are eliminated and less time is spent in reading scripts. The third function of the SCRIPT LOADER is placing the reformatted scripts on the disc. It is necessary to place the scripts on the disc for speed of Simulator execution. Disc access time for random access is on the order of 30-40 milliseconds. In order to randomly access a script file on magnetic tape, roughly 1000 times more time would be necessary. The Script Loader program also performs diagnostic and error-detecting functions on its

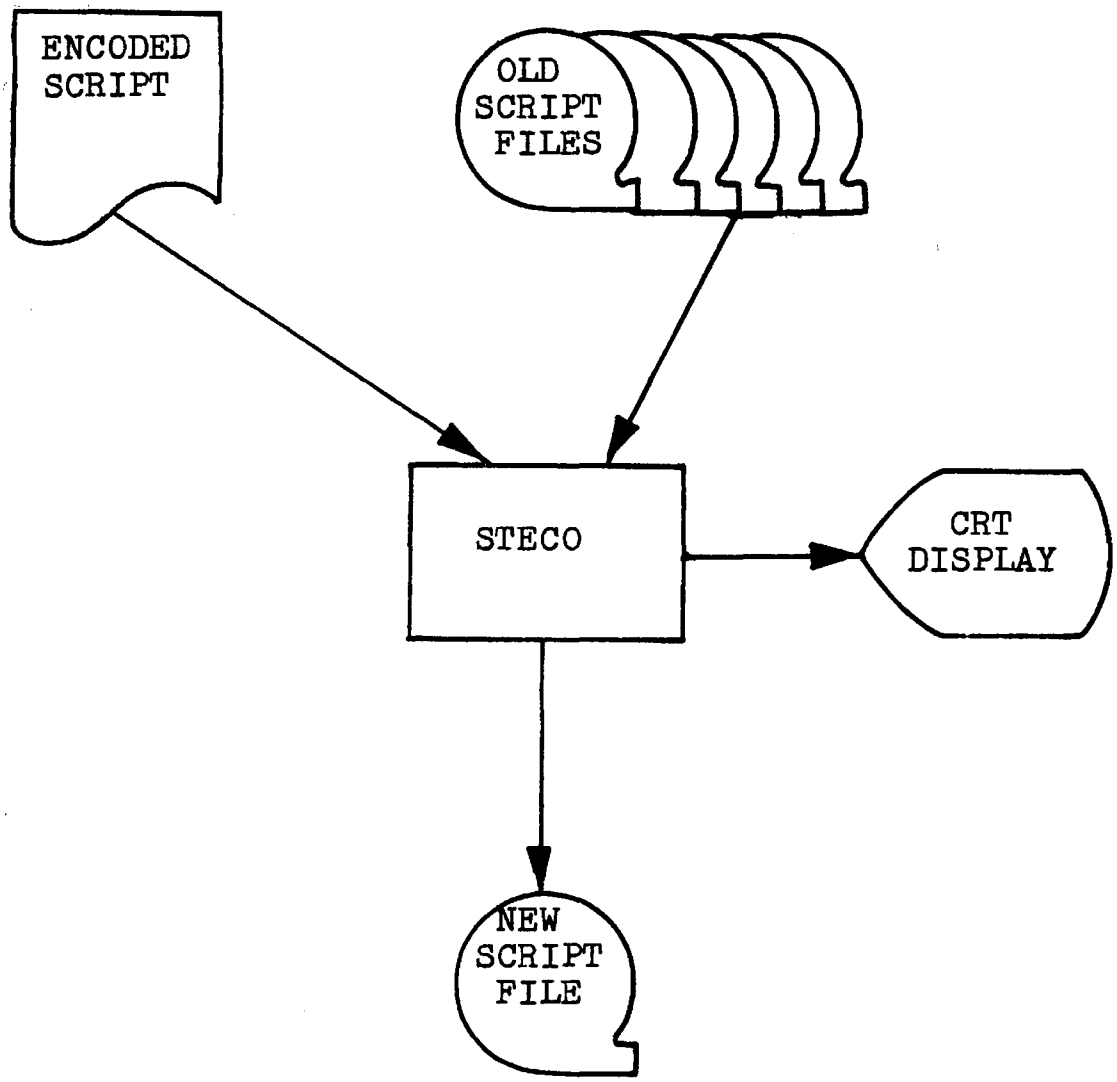


FIGURE II-2 : Creation of New Script file by STECO from encoded script and/or old script files.

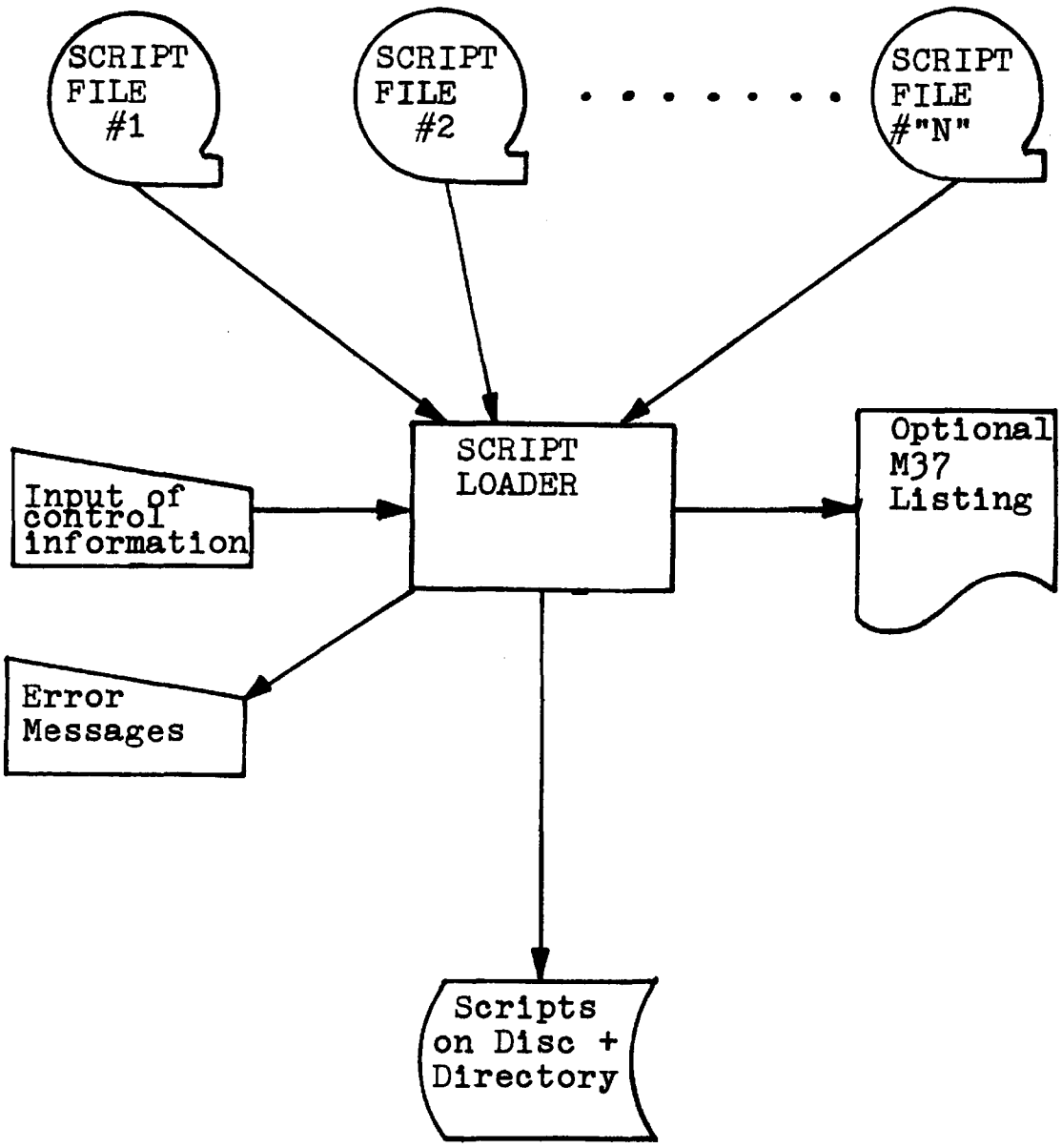


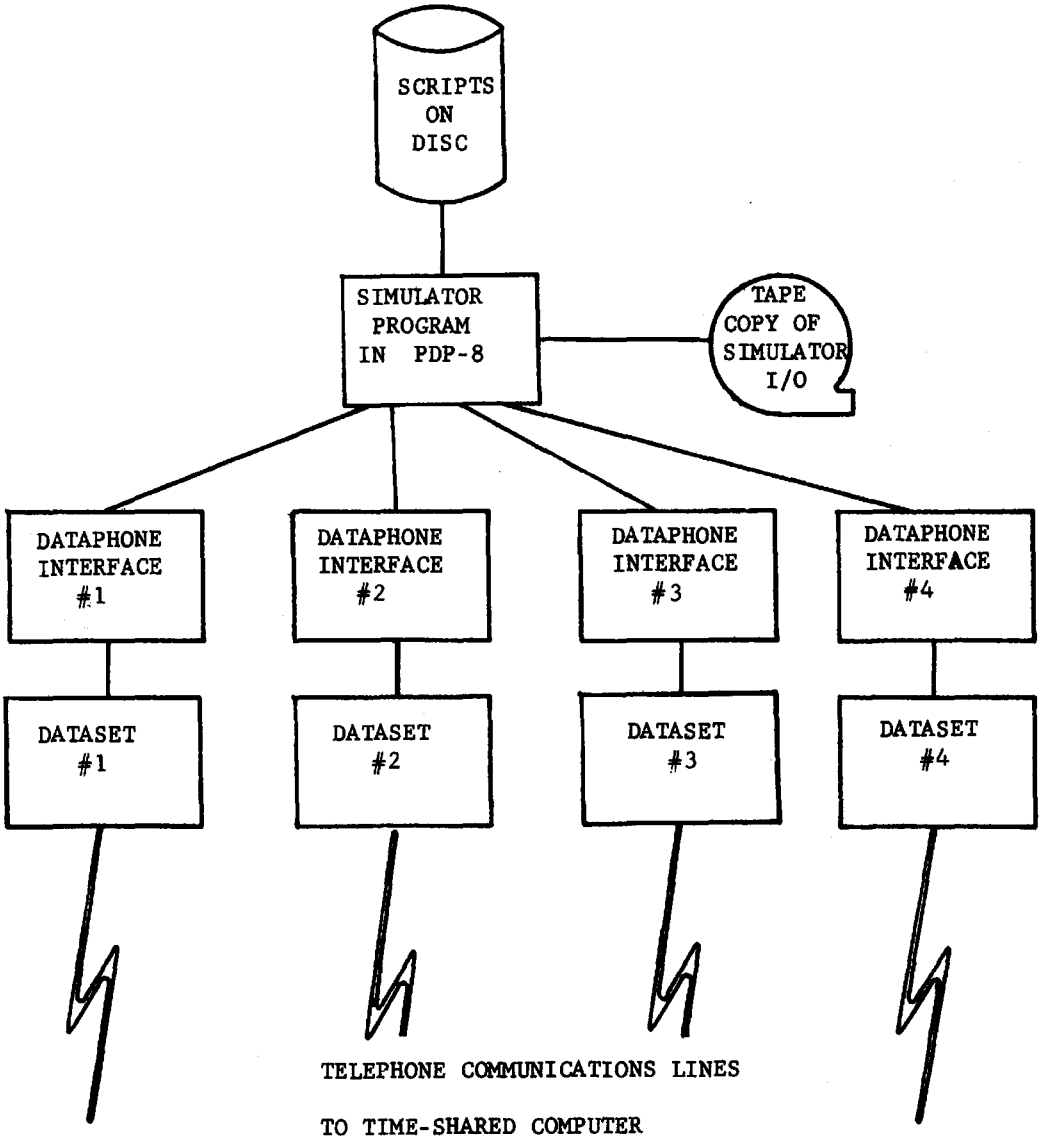
FIGURE II-3: Loading of the disc with the script files, using the SCRIPT LOADER program.

script files. Checking is performed for format errors in the script, while transferring the script information onto the disc for the later use of the Simulator. Thus erroneously formatted scripts are detected before simulation time. In view of this checking feature the Script Loader program may be used independently of the Simulator to pre-check script files for errors.

The fourth phase consists of using the PDP-8 to test a Time Shared System. Since the scripts have previously been loaded onto the PDP-8 disc, this phase consists of loading the Simulator program, dialing up each individual dataphone line for each user to be simulated, and then issuing a command on the PDP-8 teletype console to start the Simulator operation. During this phase the Simulator derives its operating information from the scripts on the disc, performs its verification on all input coming into it over the dataphone line, performs "think time" simulation, and outputting of text on the dataphone lines. (See figure II-4) In case of error conditions due to parity errors on the dataphone lines, time-outs on verification, etc., appropriate messages are printed on the PDP-8 teletype console and simulation may be terminated.

During the entire simulation, a record is kept of all input and output received and transmitted. (See figure II-4) This record is kept in the form of a magnetic tape file, which may be listed and/or manipulated after simulation. The use of this magnetic tape file forms the last and optional phase of the simulation. Thus a complete picture of the simulation may be

FIGURE II-4. Simulator Program and PDP-8 Configuration



retrieved by examination of this tape file. The tape file is in standard PDP-8 OS8 file format and may be operated upon by routines using the available OS8 file system. The initial version of the Simulator will provide two routines, one for straightforward listing of the tape file on a per user basis, and the other to provide a picture of the response time between "WAIT" messages on "MULTICS" or execution time parameters of the MULTICS "READY" messages.

III. THE SCRIPT

A. Introduction

The script is the external data base for the Simulator. It controls the actions of the Simulator during the simulation runs. Before making a simulation run, one must first design script(s) to exercise the system being tested. Once the script(s) are designed, the operator must create magnetic tape files of the symbolic scripts using the PDP-8 symbolic text editor STECO. The use of STECO is described in Appendix A. The operator then uses these tape files as input for the Script Loader. The Script Loader is a program which translates the script files into a special format and deposits the scripts in this special format onto the disc. It also deposits accumulated control information for each script onto the disc. During this translation, errors in the formatting of scripts are detected and the operator is notified through the means of error messages on the PDP-8 on-line console. An optional feature of this translation process allows a copy of the scripts to appear on the Model 37 Teletype, which may be dialed up to the PDP-8 during the script loading phase.

The scripts contain the following information:

1. Output text - those characters which would normally be typed by a human user at his console.
2. Verification lines - information in a special code, representing awaited response(s) from the system being tested.

3. Verifier time limit - a number representing the maximum number of minutes the Simulator is to allow to elapse without having recognized the proper response line from the TSS being tested as represented by the verification time. At simulation time, if this time is exceeded without the required recognition first having taken place, an error condition is made known to the operator by an error message appearing at the PDP-8 on-line console.
4. Think time - a number representing a delay of seconds between achieving a positive verification and before sending the next line of text to the system being tested.
5. Format control information - used to delimit the above four items.

The remainder of this chapter describes the content and format of each of the above items and finally the format of the entire script as a unit.

B. Description of the Individual Script Components

1. The Output Text Line(s)

The output text line is the string of characters which a normal human interactive user would type on his teletype console as input to a time-sharing computer system with which he is communicating. While a human user would have certain restrictions imposed upon him by the system he is using, in reference to content and syntax of the strings he would be typing as input, the scripts used by the Simulator place several additional restrictions. The first restriction is that input text lines may be no longer than 120 characters in total. Second, the Simulator will be emulating Teletype Model 37's (either high or low speed, depending upon the setting of internal parameters). However, not all the Model 37's character set will be acceptable as a character of the text line string. The exceptions are: "#", "@", "FORM FEED", "CARRIAGE RETURN", and "DELETE". These restrictions stem not from any intrinsic limitations of the simulator, but from the use of these characters as special function characters for the PDP-8 symbolic text editor TECO. An added restriction is that there be no embedded "new line" characters within the text line since the New Line character (henceforth abbreviated "<NL>") serves as a delimiter for text lines in the Simulator. These delimiting <NL> characters are included in the text being transmitted.

In summary, an output text line may be a character string up to 120 characters in length consisting of the characters in Appendix D as legal characters and terminating with a NL character.

2. The Verifier Line

In Chapter II the rationale for a verification line was presented. In summary, it was found that some method of testing the content of lines sent by the time-shared system being tested, was necessary to determine when the TSS has sent its last line before expecting a response from a user. The Simulator accomplishes these decisions by examining each "line" sent to it from the TSS being tested (on a per user basis) and comparing it with a "verifier" line included in the simulated user's script. If the verifier line "matches", the next text line is sent or preparations are made for further verification on subsequent lines of text. If no "match" is found, subsequent lines transmitted by the TSS being tested to the Simulator are examined and a match is attempted, until such time as either a "match" is found, or a preset "time limit", also included in the script (described in the next section), is exceeded.

If the exact character by character representation of a response line, sent by the TSS being tested, is known, it is trivial to provide a "verifier" script line to match it, on a character by character basis. However, in general, the exact format of a response line from the TSS being tested is not known. The following two examples point this out;

1. Consider the CTSS standard "READY" message. It is of the format "R 1.020 + 0.052", where the numbers represent execution and swap time (CTSS parameters) respectively. Of course these numbers differ, for even the same program executed twice, being dependent on system loading. In addition,

each of the numbers may have zero, one, two or three or more digits to the left of the decimal place, depending on the length of execution time and swap time needed. We are faced with having to verify such a line with these two independently variable fields, and determine if the line is indeed a "ready" message.

2. Several more problems arise from the fact that both CTSS and MULTICS perform optimization on their input and output lines. For example, a user may type a line beginning with 10 space characters. However, CTSS or MULTICS may choose to convert these ten spaces into a TAB character to conserve time and space. How then can the simulator solve the problem of "matching" these lines?

It is necessary that the script's verifier lines possess the capabilities of finding a "match" for lines such as above unambiguously. Therefore a script language for the verifier lines was developed with these and other similar problems in mind.

It was first necessary to choose two characters which have a special interpretation when encountered in a Simulator script verifier line. These are "!" and "?". Consequently it is forbidden to use these characters in a verifier line to obtain a direct match. These two special characters have the following interpretation:

1. The "!" character

A "!" found in the verifier line means that the simulator should not attempt to match against the next character of the input line. In effect it skips the processing of the "next" character in an input text line, but takes into account the fact that there is one character there. The occurrence of n "!"'s in a row signify that no direct match is to be attempted upon the next "n" characters in the input. For example, the following input text line:

"abcdefghi.<NL>"

can be matched using the following verifier line, if all one were interested in was the period terminating the line and the number of characters preceding it:

"!!!!!!!!!.<NL>"

The string "abcdefghi" would not be matched against, however the number of characters, the period and the NL characters would, and a positive match therefore found.

2. The "?" character

A "?" encountered in a verifier line signals the Simulator to eliminate the matching of an unknown number of characters until it encounters the first occurrence of a match on the character immediately following the "?" in the verifier line, and then to continue verification from there. For example, if our verifier line were "?.<NL>" and the input

line to the simulator we wish to attempt to match, as in the last example were:

"abcdefghi.<NL>"

We would find that a "match found" condition is signalled since the "?" character would eliminate the attempting of matching against the characters "a, b, c, d, e, f, g, h, and i" however, the period and the <NL> would be found and a match signalled.

The advantage of the "?" character over the "!" is that one would not have to know precisely how many arbitrary characters occur before the "." in this case to achieve a match. For instance, input text lines of "abcde.<NL>" or "XyZPq.<NL>" would also produce a match with the verifier line "? .<NL>".

If a "?" is encountered at the end of a verification line, this is a signal to the Simulator that the rest of the line is not to be verified. For example, one could verify the aforementioned CTSS "READY" line with the verifier line "R <NL>". The Simulator would find a match for the "R "(R<space>) characters of the ready line by comparing against the "R " characters of the verifier line, and not attempt to verify the rest of the line, due to the "?", therefore signalling a match.

Therefore the "?" is the logical extension of the "!" character where one does not know exactly how many characters to skip matching. However, great care should be exercised in the use of the "?" since erroneous lines may be verified positively in many instances. If the exact number of unknown characters to be skipped is known, it is advisable to use the "!" convention.

Finally, the simulator design only retains the first 31 characters of any input line for matching purposes (See Chapter 5). Keeping this in mind, one should design the verifier line so as to be able to find a match based on the first 31 characters of any input text line. This also implies that any single verifier line may not be longer than 31 characters, not including the terminating <NL> character.

3. Verification Time Limit

In Chapter two the rationalization for the need for a verification time limit was presented. In summary, we wish to place an upper limit upon the amount of time the Simulator is to wait to verify a particular line from the script before the Simulator signals an error condition. It was found that this was necessary to inform the experimenter of machine downages, and other anomalies in operation.

This time limit is placed in the script on the line immediately before the verifier line and consists of one, two or three digits. It must be terminated by a <NL> character. This number represents the maximum number of minutes we wish to allow to elapse while waiting for a match of the verification line.

4. Think Time

In Chapter 2 the rationalization for the think time before sending a line of output text was presented. In summary, it was noted that a user, in general, does not immediately type a response to the TSS being tested once he is given the "go-ahead". There is a certain amount of time spent in "head-scratching" and thinking, before and during the

typing of his next line of text. Although a user may spend this time during typing of the line, the Simulator will wait, to simulate this "think time", before transmitting the next text line.

Therefore, a number is placed in the script representing the number of seconds we wish to have the Simulator wait after having determined that it is to transmit its next line, by having found a match against to its verifier line. This number is represented in the script as a one, two or three digit (decimal) number representing the number of seconds of "think time" we wish to elapse before the transmission of the text.

C. Script Format

The individual components of a script have now been defined and their internal format explained. Composing a complete script from these components is the topic of the following discussion.

A complete script consists of subsections in one of three modes: verifier mode, text mode, and comment mode. Ignoring momentarily the comment mode subsections, the script is composed of alternating subsections of verifier and text modes. Each script must begin with a verifier mode subsection, and each verifier subsection must be followed by a text mode subsection, and this in turn must again be followed by a verifier mode subsection. A script may terminate at the conclusion of any of the three modes. Comment mode subsections may be inserted between subsections of either modes, and are ignored by the script loader. Comment mode subsections may not be inserted in the middle of any particular mode other than another comment mode subsection.

A verifier mode subsection consists of "n" couplets of lines where "n" must be greater than or equal to 1. There is no upper limit on the number of couplets which may comprise a verifier mode subsection, the only restriction being that there is sufficient room on the PDP-8 magnetic disc to accomodate all the information. The first line of a couplet is the Verifier Time limit, discussed previously, which limits the amount of time allowed to elapse before a match is found on the following verifier line. The second line

of the couplet is the actual verifier line used. Each of these lines must be delimited on the right by the <NL> character.

The start of a verifier mode subsection is signalled by the character string:

":V<NL>"

Thus a verifier mode subsection to verify the line

"XXXXXXXpqr<NL>"

in a maximum of three minutes might appear as:

:V<NL>

003<NL>

?pqr<NL>

(where <NL> signifies the New Line Character).

The <NL> character delimiting the verifier line on the right is included as a character in the verification procedure. However, occasionally one wishes to attempt to match a line not terminated by a <NL> character. This exception occurs, for instance, on CTSS during the use of QED, when a command is issued to write a file using a name for which a file already exists. QED requests permission to delete the old file name before writing the new file and expects an answer on the same line as its query. The line appears as:

Do you wish to delete this file?

There is no <NL> character following this line, and hence the verifier line to match this line may not contain a <NL> character. However, the syntax of the verifier line is such that <u>every</u> verifier line

must be delimited on the right by a <NL> character. To accomodate this exceptional case and others of the same ilk, the following convention is established:

> If a verifier line is terminated on the right by the string"\<NL>" (<backslash> <NL>), instead of just "<NL>", the verifier line will not include the <NL> character as part of its matching procedure.

This convention, of course, precludes directly matching a line which ends with the string "\<NL>". This line however, can be matched by using the "!" or "?" conventions. This convention only applies to the occurence of the string "\<NL>" which terminates the verifier line. The "\" (backslash or escape) character may be used in the normal manner within the verifier line.

Text mode subsections are similarly constructed. Each text mode subsection consists of "n" couplets, where n is greater than or equal to 1. Each couplet consists of a "think time" of one, two or three decimal digits terminated by a <NL> character, followed by the text line to be transmitted, also terminated by a <NL> character. The <NL> character terminating the text line is included in the text to be transmitted.

The start of a text mode subsection is designated by the character string ":T<NL>". A text mode subsection to transmit the character string for a simulated user of the following characters:

"HOW ARE YOU?<NL>"

after waiting for 23 seconds to simulate think time would appear in the script as:

:T<NL>

23<NL>

HOW ARE YOU?<NL>

where the characters "<NL>" again signify the presence of a New Line character.

The comment mode subsections are available for documentation purposes in the script, but do not influence the actions of the Simulator. A comment mode subsection may be inserted in between subsections of the other two modes, but may NOT be embedded within the body of one of the other two modes.

The start of a comment mode subsection is signified by a ":C<NL>" string, and a comment mode subsection continues until the occurrence of either the ":V<NL>" delimiter string or the ":T<NL>" delimiter string. These delimiters must occur on new lines, i.e. nothing else may appear with these delimiter strings on a line.

A comment mode subsection may be of any length and contain any characters with the exception of the verifier and text mode subsection delimiters, ":V<NL>" and ":T<NL>".

Table 1 contains a BNF description of the syntax of the Script Language introduced in this chapter.

TABLE 1: BNF DESCRIPTION OF THE SCRIPT LANGUAGE

<script> ::= <script unit> | <script> <script unit>

<script unit> ::= <comment> :VNL <ver sect> <comment> :TNL <text sect>

<comment> ::= :CNL <any string> NL <comment> | <NULL>

<NULL> ::= the empty string

<any string> ::= any string of legal characters on the M37
listed in Appendix except those containing
the folowing substrings:
"NL:TNL" or "NL:VNL"

<ver sect> ::= <verifier couplet> <ver sect> | <verifier couplet>

<verifier couplet> ::= <max time> NL <ver sect> NL

<max time> ::= $<\text{digit}>_{1}^{3}$

<digit> ::= 0|1|2|3|4|5|6|7|8|9

$$<\text{ver string}> ::= <\text{string of characters listed in Appendix E}>_{1}^{32}$$

<text sect> ::= <text couplet> <text sect> | <text couplet>

<text couplet> ::= <think time> NL <text string> NL

$$<\text{think time}> ::= <\text{digit}>_{1}^{3}$$

$$<\text{text string}> ::= <\text{string of characters listed in Appendix E}>_{0}^{120}$$

D. An Example of a Script

1. Introduction

This section provides the reader with an example of a script. This example is presented as an effort to tie together the ideas of the previous sections of this chapter in a simple yet comprehensive example. This example is taken from a short console session on the MIT CTSS. The console listing as a human user would normally see it appears in Figure III-1. Note, however, that the left hand margin contains line numbers which are not part of the CTSS console printing. These line numbers have been added to facilitate references to lines in this example.

2. The Example

The example session chosen is very simple. It consists of:

a. "Logging in" on CTSS, to gain access to the system.

b. Some simple editing using the CTSS editor QED.

c. Logging off CTSS, thus disconnecting the console from CTSS.

In the explanation of this example, only superficial knowledge of the operation of CTSS and QED will be assumed.

a) Logging in on CTSS

Lines 1-11 of the example of Figure III-1 represent the "logging in" function onto CTSS. After the human user has dialed up his console to CTSS, lines 1, 2, and 3 are typed by CTSS. Line 1 is an indication of system capacity and loading.

FIGURE III-1. The Sample Console Session

```
 1   MAC7A3: USERS= 8, MAX=  30.
 2   READY
 3
 4   login t234 greenb
 5   W 1201.9
 6   Password
 7   T0234 5134 LOGGED IN 08/10/68 1202.0 FROM 20000.
 8   LAST LOGOUT WAS 08/10/68 1201.3 FROM 20000.
 9   CTSS BEING USED IS MAC7A3
10   R 4.083+.783
11
12   qed
13   W 1203.0
14   QED
15   i
16   This is dummy line 1
17   This is dummy line 2
18    f
19   1,$p
20   This is dummy line 1
21   This is dummy line 2
22   q
23
24   R .833+2.083
25
26   logout
27   W 1205.2
28   T0234 5134 LOGGED OUT 08/10/68 1205.2 FROM 20000.
29   TOTAL TIME USED =   .1 MIN.
30
```

Line 2 signifies that following line 3 the user should type in this "login command. Line 3 is a single <NL> character typed by CTSS, and merely spaces the "READY." message of line 2 apart from the user's response on line 4.

Line 4 is the user's response, after having seen the "READY." message of line 2 and the <NL> of line 3. The user types:

login t234 greenb

followed by a <NL> character to log himself in. The first word, "login" is the command, the latter two, the user's identification, in this case the author's. After typing line 4, the user waits for the "W" or "wait" message, telling him the system is evaluating the line he has typed. After CTSS types the wait message, line 5, it types line 6, requesting the user's "secret" password. This password is not printed on the console when it is typed by the user to preserve its secrecy. If printed it would appear between lines 6 and 7 of the listing. In this example we will represent the password as:

******

The user terminates his password with the <NL> character, and CTSS evaluates this. If the password is accepted, as it has been in the listing of Figure III-1, CTSS prints out some information pertaining to the user and the current message of the day, if any.

Lines 7, 8, and 9 are these lines. After printing this information CTSS signals that it is ready to accept console input by typing its ready message, line 10, followed by a blank line consisting of a single <NL> character. The ready message is a different type from the first ready message. It consists of the characters "R" (second character is a blank) followed by two "real" numbers. The first of these is the execution time of the preceding step, in this case the login command. The second number is the "swap" time, a system parameter of interest since the user is billed according to both these numbers. These two numbers are separated by a "+" (plus sign). Each of these numbers has the following format:

i) None, one, two, or more digits representing seconds

ii) a decimal point

iii) three digits representing tenths, hundreths and thousandths of seconds respectively

We will now construct a script segment to perform the manipulations just explained for logging in. In the previous part of the chapter it was stated that all script lines must be terminated with a <NL> character. In our example, however, these terminating <NL> characters will not appear as the string "<NL>" as in previous sections. They will be represented as they would be in a normal script listing as a new line. The single exception to this is when a script line consists only

of a <NL> character. In this case the string "<NL>" will be typed to signify this type of line, as well as moving to the next line. In the following script, "think time" numbers will be arbitrary, and verifier time limits only a guess. In most cases, verifier lines are not unique, but only exemplary.

As described in a previous section of this chapter dealing with script format, all scripts must begin with a verifier mode subsection. Since CTSS first responds with lines 1,2, and 3 which of these should one verify. Line one, for this example, will not be verified although it could have been. In this example, lines 2 and 3 will be verified. The following verifier subsection accomplishes the matching for lines 2 and 3.

```
:V
2
READY.
1
<NL>
```

The example script segment allows a maximum time for verification of line 1 of 2 minutes. On each line transmitted to it by CTSS it attempts a character by character match on the characters:

READY.<NL>

where the string "<NL>" represents a new line character. Since line 1 doesn't match this verifier line, line 2 is tried. A match is found, hopefully within the 2 minutes maximum allotted for this match to take place. Once this match is found, the

next couplet, consisting of a 1 minute maximum time and a single <NL> character is used for matching against subsequent lines. Since the next line on the console is a single <NL> character, it too is matched.

The next step would be taken by the user; he must type the "login" command. This is represented by the text mode subsection:

```
:T
30
login t234 greenb
```

We have arbitrarily placed a 30 second wait time before the "typing" of this line.

The CTSS response to the typing of this line is lines 5 and 6. For this example line 5 will not be verified. Line 6 will be verified by the following verifier mode subsection:

```
:V
2
?password?
```

if the response from CTSS, line 6, arrives within 2 minutes. Note the presence of the two "?"'s. These are necessary since line 6 is typed in red. They take into account the control characters necessary to perform the red and subsequent black shifts for this line.

Once the match on the "password" line is found, the user must type his password. This will be represented here as"******". The following text mode subsection accomplishes the typing of the password after a 10 second "think time":

```
:T
10
******
```

Once this line is sent, CTSS responds with lines 7, 8, 9, 10, and 11. Since we have already noted that we cannot depend on the placement nor content of lines 7, 8, and 9, we will only attempt to verify lines 10 and 11. This verification is accomplished by the following verifier mode subsection:

```
:V
5
R ?.!!!+?.!!!
1
<NL>
```

Again the time portion of each couplet is arbitrary. Lines 7, 8, and 9 will not "match" against the first verifier line. Line 10, however, will. Referring back to the previous discussion of this ready message, we find that it always begins with the two characters "R" and a space. Following this is a real number which may have none, one, two or more digits to the left of the decimal point. The "?" will eliminate direct matching on these and forces searching only on the decimal point which immediately follows it in the verifier line. Once the decimal point is found, three more characters are skipped, due to the verifier line substring "!!!". Immediately following these three characters, a "+" must occur. This does, indeed, occur in line 10. The "?" following the "+" in this case causes no action since no characters occur before finding the decimal point. The decimal point, is matched, three characters are again skipped since the verifier line contains "!!!". Then the concluding NL character is matched.

b) The QED text editing section

The second portion of the CTSS session spans lines 12 through 25, inclusive. After having logged in successfully in the last section, we first issue the CTSS command to call the text editor QED. The human user would type "qed<NL>" as on line 12. CTSS responds with the "wait" message on line 13. Once CTSS loads and transfers control to QED, QED responds by typing line 14 on the console. For this example, we will simply enter a two line text into one of QED's buffers and request that QED print it. This is done by typing lines 15 through 19, inclusive, on the console. QED will then list the buffer. This listing is lines 20 and 21. When QED has finished typing these lines we wish to terminate operation of QED. This is done by typing a "q" on the console, line 22 of Fig. III-1. This returns control to the CTSS monitor, which then responds with lines 24 and 25, the ready message, as before.

The script segment to accomplish these manipulations follows:

```
:T
15
qed
:V
2
QED
:T
5
i
10
This is dummy line 1
100
This is dummy line 2
33
\f
29
1,$p
:V
```

```
4
This ? line 1
1
This is dumm! line 2
:T
30
q
:V
3
R ?.!!!+?.!!!
1
<NL>
```

As before, both think and verifier times are arbitrary. The first text mode subsection waits 15 seconds, then causes the simulator to type "qed". The following verifier mode subsection awaits the response from QED of "QED<NL>", and ignores the wait message line. The next text mode subsection causes lines 15 through 19 of Fig. III-1 to be typed. The following verifier mode subsection is just one of countless ways to verify the response from QED, i.e. lines 20 and 21 of Fig. III-1. After this, the text mode subsection causes the issuing of the quit character "q" after a 30 second wait. The final verifier mode subsection, verifies the reception of the ready message from CTSS and the following <NL> character as in the previous section.

c) Logging out

The script constructed so far accomplishes the login procedure, and some simple text input using the symbolic text editor QED. To conclude the session, a user must log out. After having recognized the ready message in the last section, a user at the

console then types "logout<NL>". This terminates his interaction with CTSS. The system responds with lines 27 through 30, giving the user some indication of his status on CTSS. Then, CTSS automatically disconnects (hangs up) the console dataphone.

The script segment for the Simulation to accomplish this logout procedure must first contain a text mode subsection to transmit the logout line to CTSS. It may also contain a verifier mode subsection to test the subsequent lines to test if the logout was successful. In this example, only line 29 will be tested for. The script segment for logout is:

```
:T
10
logout
:V
4
TOTAL TIME USED =?
```

This completes the script.

3. The Complete Script

Figure III-2 is the complete script constructed in the last section. It contains, in addition to the text and verifier mode subsections constructed previously, several comment mode subsections. These comment mode subsections are included to demonstrate their proper placement and format in a script.

FIGURE III-2. The Complete Script

```
:C
This script is an example of a "typical" script for use with
the simulator.
:V
2
READY.
1
<NL>
:C
The previous verifier mode subsection waits for the CTSS
preliminary "ready" message.  It allows a maximum of 2
minutes to find the ready message, and 1 minute for the
second following NL (new line) character.  The following
text mode subsection issues the login command.
:T
30
login t234 greenb
:C
Having issued the login command we must wait for CTSS to
request the simulator to type its password.  The next
verifier mode subsection will wait 2 minutes for CTSS
to request the password before signalling an error condition.
:V
2
?password?
:C
Since the password request has been by now issued, the
simulator must reply with the proper password, represented
here as "******".
:T
10
******
:C
The Simulator will now wait for the standard CTSS ready
message for 5 minutes maximum.
:V
5
R ?.!!!+?.!!!
1
<NL>
:C
The following section requests CTSS to load and pass control
to QED, the text editor.
:T
15
qed
:C
The simulator must now wait for QED to print out its own acknowledge-
ment "QED".
:V
2
QED
```

```
:C
The simulator now enters two lines of text into QED's
buffers and requests that these be printed out.
:T
5
i
10
This is dummy line 1
100
This is dummy line 2
33
\f
29
1,$p
:C
Having requested the printout of these two lines, the
Simulator  now waits for them to be typed out by QED.
Note that this verifier subsection is different from
the one previously presented in the example.  This
version requires an exact match on every character.
:V
4
This is dummy line 1
1
This is dummy line 2
:C
Next, the simulator is to transmit a quit for QED.
:T
30
q
:C
We must now wait again for the CTSS ready message
:V
3
R ?.!!!+?.!!!
1
<NL>
:C
The simulator is now to log off from CTSS.
:T
10
logout
:C
After issuing the "logout" command, the simulator is
to wait for the "TOTAL TIME USED =" line.
:V
4
TOTAL TIME USED =?
:C
The script ends here.  No special termination is required.
```

IV. SIMULATOR SYSTEM IMPLEMENTATION

A. Introduction

This chapter contains a description of the implementation of the Simulator System. The Simulator System is implemented upon the PDP-8 computer at Project MAC, M.I.T. The first part of the discussion in this chapter concerns itself with the organization of the PDP-8 computer, in order to put the rest of the chapter into proper perspective. Following the discussion of the PDP-8 organization, a description of the hardware modifications made to the PDP-8 computer to accomodate the Simulator System is presented. The remainder of the chapter is devoted to providing an overview of the operation of the programs comprising the Simulator System. It is impractical to present a completely detailed description of the operation of the programs comprising the Simulator System. Therefore, the detailed description of all programs is relegated to the appropriate appendices of this thesis. In these appendices the reader will find flowcharts and descriptions depicting the step-by-step operation of each of the programs of the Simulator System.

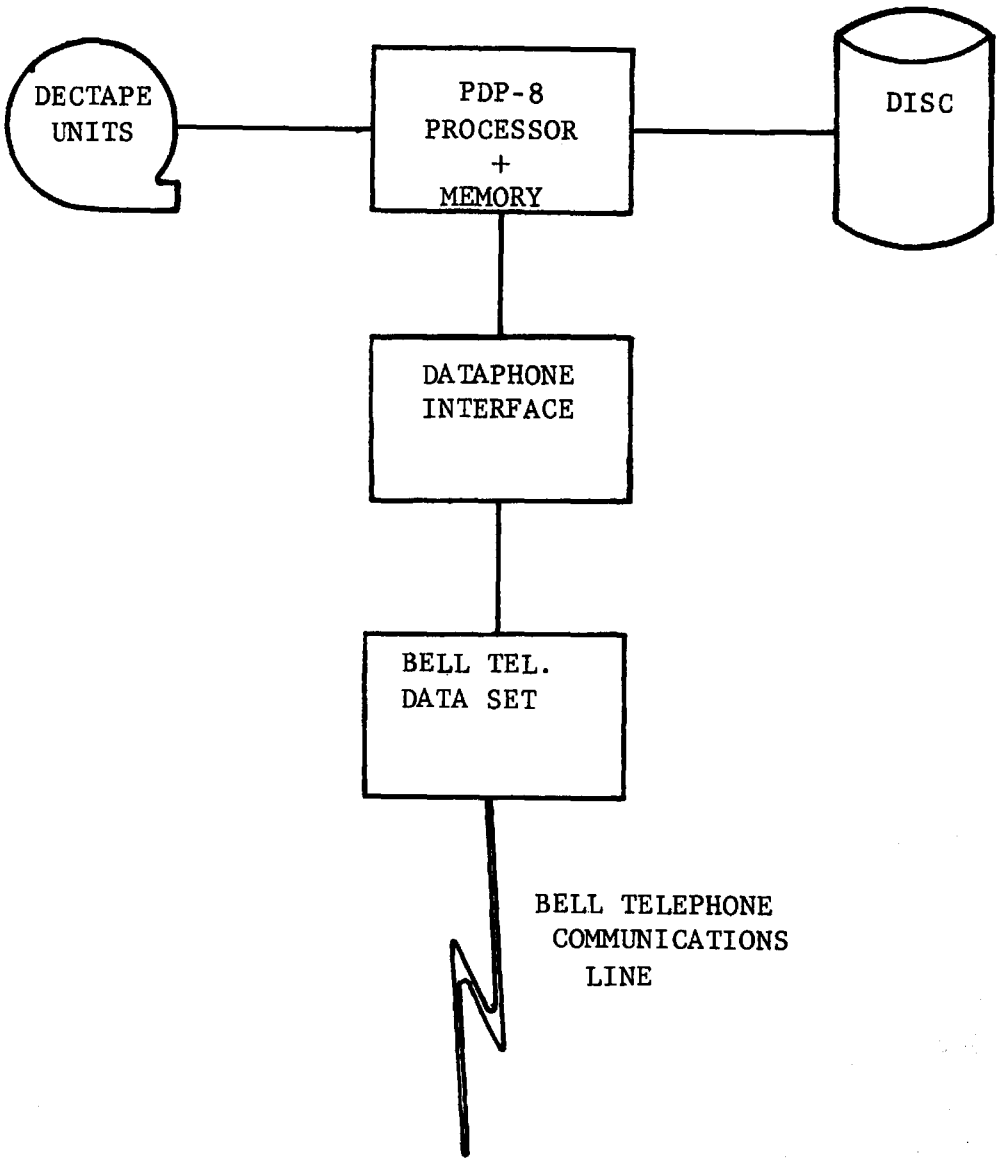
B. The Computer Hardware and Modifications

The Simulator System is implemented upon the DEC PDP-8 computer situated at M.I.T.'s Project MAC. The M.I.T. PDP-8 is a small, general purpose computer with 8,000 twelve bit words of primary core storage, 32,000 twelve bit words of secondary disc storage, two magnetic tape drives, a CRT display unit and a ASR33 teletype console. The PDP-8 was chosen for the implementation of the Simulator since it allows easy hardware modification. The original PDP-8 hardware configuration is pictured in figure IV-1.

A stated objective in chapter III was to allow each simulated user to have his own dataphone line. The original PDP-8 configuration had one hardware interface for a dataphone. Since the aim of the initial implementation of this Simulator is to support a simulation of up to four users, three additional hardware interfaces for dataphones were constructed and integrated into the PDP-8. This allows one dataphone line for each of up to four users to be simulated.* A complete description of the modifications necessary for these interfaces is included in Appendix B. It should again be noted that the Simulator System programs are capable of supporting the simulation of up to twelve users. In order to accommodate this number, additional hardware interfaces must be constructed. The initial implementation of the Simulator accommodates only four simulated

*The original design of the dataphone interfaces was done by Fred Luconi as part of his Master's Thesis.[5]

FIGURE IV-1. Original PDP-8 Configuration Less CRT Display



users, since this number will be sufficient to exhibit proper operation of the Simulator. Details necessary to add more dataphone interfaces to the existing configuration may be found in Appendix B. The present hardware configuration for the PDP-8 is depicted in Figure II-4.

The remainder of this chapter consists of an overview of the programs which comprise the Simulator System.

C. STECO- The Simulator System Symbolic Script Editor

The M.I.T. PDP-8 Operating System has available to it a versatile symbolic text editor, called "TECO". However, TECO requires the user to type on the Teletype ASR33 which is attached to the PDP-8. The ASR33 is an early model teletype and has available only upper case alphabetics. Use of only upper case alphabetics is too severe a restriction to be placed on Script files for the Simulator. Therefore modifications were made to the original TECO program to accomodate an external TELETYPE Model 37 over a dataphone line, so that both upper and lower case alphabetics could be used. The initial conversion was accomplished through the efforts of Tom Skinner of Project MAC at M.I.T. Subsequent modifications were made by the author of this thesis to better adapt this new TECO for use in writing and editing scripts, such as the addition of parity bits in characters sent to the M37, and the manipulations necessary to properly display text on the PDP-8's CRT Display unit.

Details of operation of this new version, called STECO, are given in Appendix A of this thesis. MAC-MEMO-191 describing a similar version of TECO for the PDP-6 should also be useful for a beginning user.

D. The Script Loader Program

The function of the script loader program is to take as input those script files chosen by the operator, reformat them, and write the reformatted information out on the PDP-8's magnetic disc unit. During the reformatting process, checking is done to ensure proper input format of the scripts. In the case of errors in the format, appropriate error messages are printed out on the PDP-8 on-line console.

An optional feature of the Script Loader program is the ability to list the input script files on the off-line M37 Teletype which must first be dialed up on the PDP-8 dataphone channel Ø.

A detailed description of the implementation of the SCRIPT LOADER program complete with flowcharts may be found in Appendix E.

E. The Simulator Program

The heart of the Simulator System is the Simulator Program. The Simulator program has the responsibility of simulating multiple users. The remainder of the programs comprising the System, are merely peripheral support to the Simulator program. While these peripheral programs of the System are fairly straightforward in operation, the Simulator program is relatively complex. The Simulator program is the only program in the System which is multi-programmed. It has three levels of interrupt, two of which are software implemented, since the PDP-8 has only two levels of interrupt.

The routines comprising the Simulator program may best be categorized by the level of interrupt at which they execute. The three levels of interrupt, and the major programs which operator at these levels are:

1. High priority interrupt level
    a) The OS8 file system used to write physical tape records
    b) The interrupt handling routine
2. Clock level interrupt
    a) The controller "ENTRY"
    b) The dataphone read-in routine "READIN"
    c) The dataphone output routine "XMIT"
3. Fully interruptable routines; the service routines
    a) The disc reading routine "DREAD"
    b) The verification routine "VERIFY"
    c) The tape writing interface with OS8 file system "TWRITE"
    d) The tape buffer controller "TMOVE"

The interrupt scheme operates in the following manner. Routines in group 3 are fully interruptable, that is interrupts of any kind are given priority during their operation. Routines at the group 2 level are interruptable only by tape and disc interrupts. Routines in group 1 are non-interruptable.

The remainder of this section is a brief outline of the operation of the Simulator program. For a complete, step-by-step description of the operation of the routines comprising the simulator program consult the flowcharts of Appendix F.

The most important and most complex routines of the Simulator program are those routines at the level 2 in the interrupt scheme. These routines are the ENTRY, the READIN, and the XMIT routines. These routines are interruptable only by DEC tape and disc interrupts. Upon detection of a clock interrupt by the interrupt handling routine, the ENTRY routine receives control. Clock interrupts occur at three times the bit rate. This rate is necessary to detect the "start pulse" on the dataphone input lines. However, the transmission of new information onto the dataphone output lines, must only occur once per bit time. The first function of the ENTRY routine is to determine if this third time has occurred. If it is the third clock interrupt time, the XMIT routine is called; if not the READIN routine is called. If the XMIT routine is given control, it first checks if the "one second" software clock for think time emulation has reached a count of one second yet. If the one second interval has elapsed, it updates the threaded list for think times and checks if any simulated users' think time have lapsed. If so these users are made "active" for transmission.

If the software clock has not reached one second yet, or after the think time checking has been done, XMIT prepares the next set of bits for transmission over the dataphone output lines. Users which are active are assigned the proper bits on their lines, since characters are transmitted in serial-bit fashion. Non-active users have binary ones on their lines. Lines corresponding to "no-users for this simulation" have zeros on them. XMIT then passes control to the READIN routine.

READIN is responsible for reading the dataphone input lines. The function of READIN is to recompose the serial-bit characters on all used lines. Once READIN has assembled an entire 8 bit character for a simulated user, it checks the parity of this character, and then calls the Work Area Manager, one of its subroutines. The Work Area Manager (WAM) has the responsibility of keeping the first 31 characters of each input line in its work areas for later verification by the VERIFIER. Upon each call from READIN, the WAM appends the character passed to it to the proper work area as determined by the user's ID number, also passed to the WAM as a parameter. Once a line has been completed, as indicated by the presence of an ACK, New Line, or WRU character, the WAM calls a routine to place an entry in the verifier's queue so that this work area may be verified. The status of this work area is now changed to "verify status" so that no further characters will be appended to it. This work area is returned to the pool of free work areas, after the verifier routine has performed its verification upon its contents. When the WAM returns control to READIN, control is in turn passed to the interrupt handler which returns control to the routine interrupted by the clock interrupt, one of the routines of group 3.

The routines of group 3 are interruptable by interrupts of any kind. They are constructed so as to fit into the multi-programmed environment necessary to the operation of the Simulator program. After Simulator initialization, control is passed to a "control loop" aptly named "CTLOOP". The function of CTLOOP is to repetitively call the three routines, DREAD, VERIFY, and TWRITE. These routines are continually called by CTLOOP and control usurped only by means of interrupts from the disc, tapes, or clock.

The first routine in the loop is the DREAD routine. DREAD has the responsibility of reading script segments from the disc. It is queue driven. Entries are put into its queue by calls from the other routines of the Simulator program. Upon receiving control DREAD first checks if the previously issued read, if any, has not yet completed. If it hasn't, DREAD returns control to CTLOOP. It there was no previous read or if the current read has completed, some bookkeeping is done to process the new script segment just read and reset pointers for the next read for this user. Then DREAD checks if there is another request for a disc read in its work queue. If not, the routine is exited. If there is, the disc read is initiated, and DREAD exits. The check for disc read completion is checked the next and successive times DREAD is given control.

The next routine in the CTLOOP is VERIFY. This routine is also queue driven. Upon entry, VERIFY checks its queue if there are any entries to process. If so the first of these is taken. Calls are made

to the disc subroutines to determine if there is a verifier line available for this user. If the verifier line is not yet available, the queue entry is placed at the bottom of the queue, and the next entry is tried. If all entries have been checked once, VERIFY exits. If an entry is found for which there is a verification line available, the verifier routine attempts to "match" the characters in the work area against the script verification line as described in chapter III. If a match is found, a request is put into the disc queue so that the next text or verification line may be retrieved for further processing. If no match is found, or if a match has been found and the disc request has been consequently made, the Verifier calls TMOVE to move the work area contents to the tape writier's buffers for the tape copy of the simulation transactions. VERIFY then exits to the control loop, CTLOOP.

The last routine called by the control loop is the tape writing routine TWRITE. This routine is not queue driven. The tape writing routine has the responsibility of issuing calls to the OS8 file system write routines to write the tape copy. When control is passed to TWRITE, TWRITE checks its buffer usage table, BTABLE, to see if there are enough buffers in it to write out a complete block (4 buffers) of tape. These control table entries are updated by the TMOVE routine when TMOVE uses a buffer. If there is a complete block to be written, TWRITE calls the file system to initiate the tape write. If not, TWRITE exits. As with the DREAD routine, checking for completion of the tape write is done the next time the routine is called.

The final routine of the group 3 routines is the TMOVE subroutine. The subroutine is called from many other routines of the Simulator program. It is called with the starting address of a word area or script segment as a parameter. The function of the TMOVE routine is to move the 32 characters starting at the address passed to it into a tape buffer, and perform some bookkeeping for TWRITE. These 32 character buffers will later be written by TWRITE onto the magnetic tape.

F. Some Observations

There are still several questions about the Simulator System which remain unanswered. In the discussion of these questions and their answers, it is hoped that some of the problems involved with designing and implementing such a user simulator may come to light for the benefit of those interested in implementing a similar tool, as well as for the curious reader.

The first question which may come to mind is "why are there five distinct phases in the operation of the Simulator System?" One portion of the answer lies in the fact that the PDP-8 is a small computer and there is insufficient core memory to accommodate simultaneously the four of the five phases of the Simulator System which require the use of the PDP-8. Phase one, the design and encoding of the desired scripts by the experimenter does not require the use of the PDP-8. Phase two, the input and editing of the encoded scripts to produce magnetic tape files requires all of available core memory, and for this reason, if for no other, must constitute a separate phase. Phase three, consisting of choosing, reformatting and placing of the chosen scripts on the PDP-8 magnetic disc unit for later simulation requires roughly three-quarters of avialable core memory. Simulation, forming the fourth phase requires all of available core memory. Finally, the fifth phase consisting of the listing and/or analysis of the tape of all Simulator transactions has no fixed amount of core memory requirements, since this phase, in general, consists of programs written by the experimenter to suit his own requirements.

However pressing the core memory requirements may be, this is not the only reason for separating the Simulator System into several distinct phases. Even if all programs necessary to support the operation of the Simulator System could be accommodated in a large core memory for the PDP-8, it would be advantageous to separate the Simulator System into its five phases. The reason behind such a bold statement lies in the fact that separation of the Simulator System into its five phases represents the most flexible and easiest to use configuration for the experimenter. Phase one, designing and encoding of the scripts is a function independent of the PDP-8 and would, of necessity, be a distinct phase, much as writing a computer program down beforehand for any other assembler or compiler would be.

Separating phase two, consisting of inputting and editing of the scripts, allows the experimenter to produce a library of scripts on magnetic tape at any time, independent of the use of the rest of the Simulator System. Keeping phase three separate from the rest of the Simulator System allows the experimenter to use the SCRIPT LOADER program to check his script files for errors without having to go on with the simulation. In addition, since the scripts are stored on tape files for input to the SCRIPT LOADER program, the experimenter may choose to load any of the script files available. Thus the subsequent simulation may be run with any of the scripts on the library. It is, therefore, advantageous to divide the Simulator System into the above distinct phases.

The final question to be considered concerns itself with the scripts. This question may be neatly divided into two pieces. First "does the script contain enough information to allow accurate simulation of an interactive console user?". Second, "why is the script formatted as it is?".

The answer to the first question is both "yes" and "no". It is "yes" in that all of the activities which a normal interactive console user carries on at the keyboard of a TELETYPE Model 37 may be accurately simulated or approximated. One non-keyboard function, that is the QUIT activity has not been included in the script. However, there is no intrinsic prohibition to the addition of this feature to the Simulator System.

The script is formatted as it is for the following reasons. First, the format was chosen to make the script relatively readable and easy to code. Second, it contains enough redundant information, in the form of the control strings ":T" and ":V" and the use of the New Line characters to make checking and detection of format errors possible by the SCRIPT LOADER program.

This empty page was substituted for a blank page in the original document.

V. PROBLEMS WHICH REMAIN

The Simulator system described in this thesis is a tool. And like most tools it is limited in its scope. In addition there are several problems intrinsic and extrinsic to the use and operation of the Simulator System. The purpose of this chapter is to point out some of the Simulator's deficits and other problems which still remain and are intrinsic to the operation of such a tool.

A. Handling of Errors

The Simulator provides for one dataphone line for each simulated user. To the TSS being tested, it appears as if there is a "normal" human user at the other end of the dataphone line. However, a human user is equipped with many "features" which have not been built into the Simulator. The most important is the ability to deal with abnormal situations which may occur at any time during the human's interaction with the TSS. There are many such abnormal situations which may occur during operation. Some of these are:

1. Transmission errors on the dataphone lines, due to noise or dropout.
2. System crashes of the TSS the human is conversing with.
3. Abnormal system (TSS) operation leading to "strange" or unexpected responses.

The Simulator can detect these problems. In the case of parity errors on the dataphone lines, the Simulator issues an error message. If the TSS being tested crashes, or an expected response line is never received

because an erroneous one or none was sent, then eventually the verifier time limit is exceeded, and a message to this effect is issued. But detection is only half the problem. The present implementation of the Simulator only detects such problems. What could be done to correct these errors, if anything?

If the system crashes, obviously, the experiment should be repeated when the TSS to be tested is back in proper operation. Errors due to transmission errors and abnormal responses though, may in part be correctable. In these cases when errors are detected, we could salvage our experiment by having the Simulator issue a "QUIT" over the faulty communications line. This would cause the TSS being tested to quit processing the current program for the user and return to command level. Then the Simulator could either backtrack or move forward to some pre-established point in its script and begin simulation for this user anew. Implementation of this feature is not prohibitively difficult. However its desirability is questionable, for the following reason.

The Simulator is a tool to be used to provide an experimenter with a means of performing a controlled, repeatable experiment on a TSS whereby he can make performance measurements on the TSS. Implementation of the feature above puts the repeatability of such a simulation in question. If the above mentioned "feature" is indeed implemented, and such an error is encountered during the simulation, the simulation made under these conditions is most likely not repeatable. Hence one of the basic purposes of the Simulator, repeatability, is defeated.

B. Design of the Scripts

The single largest problem facing the experimenter about to use the Simulator System, is the design of his scripts. The discussion of this problem is not intended to point out any deficit or defect in the Simulator. Instead, the discussion will attempt to expose some of the problems, the solutions of which are experimenter's responsibility. Each of these component problems contribute to the problem of designing the "proper" scripts for a given simulation experiment.

There are several tightly enmeshed problems which the experimenter must consider before designing a script for the Simulator. These are:

1. How to make performance measurements relating to the TSS being tested?
2. Does the simulation performed represent a "normal" user load?
3. For the purpose of an individual simulation, should the scripts represent a normal load?

How one makes performance measurements on a TSS being tested, is a problem well beyond the scope of this thesis. The way in which the experimenter makes the performance measurements on a particular TSS is a very important problem, and may, in its own way, affect the results of the measurements. The experimenter must make very sure that the metering procedure he is using does not affect the functioning of the TSS being tested. But if it unavoidable does, he must be able to take this perturbance into account in analyzing his performance measurements.

Questions 2 and 3 above represent another side of the overall problem. The two questions may be rephrased a slightly different way:

"How does the user wish to exercise the TSS being tested?"

The Simulator provides the experimenter with complete control over the environment of the TSS being tested. Unlike measurements made by averaging parameters of a normally operating and normally loaded system, the Simulator may be used to put "abnormal" or unusual loads on the TSS being tested. Thus an experimenter may design his scripts so that they exercise one portion of the TSS being tested heavily, or another portion not at all.

Conversely, the experimenter must be certain, that, if no unusual loading is desired, the scripts he designs and provides to the Simulator system do indeed represent what one might call a "normal" load for the system.

VI. CONCLUSIONS

The purpose of this section is to discuss the usefulness, flexibility and adaptability of the Simulator System described. In the previous chapters of this thesis, an implementation of one solution to making performance measurements on time-shared computer systems was developed. The point was made that previous attempts at making such performance measurements involved taking statistical averages over long periods of time of relevant meters of the time-shared computer system to be tested. One goal of the implementation presented in this thesis is to provide performance measurements of a time-shared computer system in a relatively short period of time, on the order of minutes to hours, rather than weeks as in the previous method. Another goal of this implementation was to produce a repeatable environment for the system being tested, in order that different versions of the time-shared computer system could be tested and a measure of relative performance ascertained.

The method used to provide the performance measurements in a short period of time, as well as to introduce repeatability of these measurements, is to use a second, small computer to simulate the actions of multiple, interactive console users typing at their consoles from prepared scripts. In this way, performance measurements may be made in the short period of time necessary to perform the user simulation. In addition, since each simulated user is controlled by a script, the environment provided by the user simulator is exactly repeatable.

The Simulator System is also very flexible, being capable of simulating almost all the actions a human user may perform at his console. The notable exceptions are the lack of the "QUIT" feature and the restriction of the Simulator scripts to the character set of Appendix D. The Simulator System may, however, be changed rather straightforwardly to include the "QUIT" feature if it is so desired. The limitation of the scripts to the character set of Appendix D is not an intrinsic limitation of the Simulator itself. Instead, it stems from the use of the prohibited characters as special function characters by the symbolic text editor STECO. Modifications to STECO to provide for different special function characters instead of the current ones would eliminate this shortcoming.

In conclusion, the Simulator System promises to be a valuable tool in the immediate future as an aid to the implementers of MULTICS at Project MAC, M.I.T. It is to be used initially for two purposes. First, it will be used to provide a controlled, repeatable load for the purpose of making performance measurements on MULTICS, to determine weak points and bottlenecks in the MULTICS system. In addition it will be used as a "standard" load in the following way. Several "standard" simulations exercising as much of the MULTICS system as possible will be devised. Following each major system modification these "standard" experiments will be run to determine if either system performance has improved as a result of these changes, or if any new "bugs" have been introduced as a result of these changes. Thus the Simulator System will provide a valuable tool to help in the implementation and improvement of the MULTICS system.

LIST OF APPENDICES

This empty page was substituted for a blank page in the original document.

APPENDIX A - Users Handbook to Simulator System Operation

Introduction

There are five phases to the operation of the Simulator System:

A. Script creation.

B. Script input and editing to produce symbolic magnetic tape files.

C. Transferring the symbolic tape files from tape onto the PDP-8 magnetic disc. During this phase, error detecting is also done.

D. User Simulation - this phase consists of using the Simulator Program to simulate the requested number of users in order to test a particular time-shared computer system.

E. Post Simulation use of the magnetic tape file of all transactions which occurred during the Simulation.

This document provides a step by step guide to the operation of the Simulator System. All references in this document refer to the Master's Thesis by Howard J. Greenbaum (M.I.T.) October, 1968. This guide assumes a working knowledge of the contents of the above thesis.

A. Script Creation

The first, and by far the most important, step in the use of the Simulator System is the design of the scripts. Several considerations enter into the design of the scripts to be used by the Simulator. The first consideration is how to design a script so as to exercise the TSS so that meaningful performance measurements may be made during operation. The solution to this problem is beyond the scope of this handbook and depends mainly on the aims of the experimenter using the Simulator System. The second consideration in the design of a script, lies in the construction of the verifier lines contained in the script. Since the verifier lines are used by the Simulator to determine when an awaited response line from the TSS being tested has been received, they must be rather exact in their construction. A verifier line that is faultily constructed may cause the Simulator to decide a line is indeed the expected line when, in fact, this line was not the expected response line, but one which was close enough in format to the expected line. Conversely, a faultily constructed verifier line may prevent a match from occurring. When the verifier time limit is consequently exceeded, an error condition is signalled due to this lack of a match. In most cases, this would cause the entire simulation to be scrapped since the simulated user would not behave as planned. It is therefore up to the script designer to plan his scripts precisely, much as he would design a piece of assembly coding for a program. The best

procedure for checking a prospective script is to actually run this prospective script by hand on the TSS to be tested. Then, the experimenter can determine if the script does indeed represent the console interactions.

Once a prospective script has been designed and hand checked, it must be coded into the format described in chapter III of the companion Masters Thesis report by Howard Greenbaum, for later processing by the Script Loader Program. Since comment mode subsections may be placed between modes of any other type, it is advised that the experimenter make liberal use of this feature to improve readability of his scripts for documentation and debugging purposes.

B. Magnetic Tape Symbolic Text File Creation on the PDP-8

1. Introduction

Once the experimenter has designed and encoded his scripts, he must create a symbolic text file on the PDP-8 magnetic tape facility. This is most easily done using a version of the PDP-8 symbolic text editor TECO, called STECO. The special version of TECO, STECO, is available on the PDP-8. STECO accepts most of the ASCII character set. A list of the acceptable input characters appears in Appendix D. STECO accepts input from a Teletype Model 37 console only. Certain characters are reserved for control functions in STECO and hence may not be typed in as script characters. These are:

#

@

New Page Character (Form Feed)

Carriage Return

Delete

STECO is a versatile symbolic text editor. It accepts input from a Teletype Model 37 console or from previously constructed magnetic tape files. Text being inputted or edited is displayed on the PDP-8's Model 338 CRT Display unit. Since large blocks of the file are immediately visible at once on the Display, editing is greatly facilitated. STECO is similar in its capabilities to the CTSS symbolic editor EDL (or EDA), having the added advantage of the Display feature.

2. Use of STECO to Produce Symbolic Script Files on Magnetic Tape

A description of the use of STECO follows. The experimenter mounts the special tape labelled "SIMULATOR SYSTEM" on a PDP-8 tape drive and dials this tape drive to #1. He then mounts his library tape, perhaps initially blank, on the other tape drive and dials this drive to #2. By this time, the PDP-8 should be powered up. The bootstrap should be in core. If the bootstrap is not in core, the experimenter should consult MAC Memo M-341 for details of starting up from scratch.

If the bootstrap is in core, the experimenter should set the PDP-8 CPU console toggle switches to "007600" (octal). He should then press the "LOAD ADDRESS" key on the CPU console and then the "START" key. The tape on drive #1 should begin back and forth motion, indicating that the operating system is being brought into the PDP-8 core memory. When the operating system has completed its initialization, it will type an "R" on the PDP-8 on-line ASR33 console. The experimenter should then type "STECO" followed by a Carriage Return character. This will cause the operating system to load STECO into memory and transfer control to STECO.

It is very important to make sure at this point, that the PDP-8 clock is set to its FAST clock rate. This is accomplished by removing the decorative blue cover panel from the upper right hand portion of the PDP-8 mainframe. In approximately the center of the wiring matrix, there is a switch controlling the clock rate. This switch must be set to its extreme right hand position. If it is not already set to its extreme

right hand position, set it so. This switches the clock to its high speed rate which is necessary to operate the Teletype Model 37 console correctly.

After the operating system has loaded STECO into core memory, the PDP-8 ASR33 console is no longer used by STECO. All transactions with STECO are performed on the Model 37 Teletype console located next to the PDP-8 until the QUIT command is issued. For this reason it is necessary to dial up the PDP-8 dataphone channel 0 (zero) from the Model 37. This may be done by putting the PDP-8 channel 0 dataphone handset into "AUTOANSWER" mode by pressing the right hand-most button on the receiver of channel 0, and dialing this dataphone up from the Model 37 console. At present this extension is 368. Once the Model 37 console is connected, STECO is ready to be put into operation.

The complete operation of STECO will not be described in this thesis. However, MAC Memo M-191 does give a complete description of TECO as available on the PDP-6. The PDP-8 STECO lacks many of the features of the PDP-6 TECO, however, this memo will provide the experimenter with the fundamentals for using STECO. A summary of the commands available on the PDP-8 version of STECO appears in Appendix C of this thesis. Note that these commands are a subset of the commands available on the PDP-6 version of TECO.

There are however, three noteworthy differences between the PDP-6 TECO and STECO. First, a single character in the command buffer may be deleted by the use of "#" convention. The deleted character is then

"echoed" on the printer. Second the "@" character deletes the entire command buffer. Thus the use of these two characters corresponds to the "#" and "@" conventions used on CTSS and MULTICS The third difference is that instead of the "ALT MODE" character described in Memo M-191, STECO uses the "DELETE" key on the Model 37 console. Note that the "DELETE" character is echoed as ~ on the Model 37 typewriter.

An added feature of STECO is that command mnemonics may be entered in either upper or lower case letters. However, great care should be exercised when entering file names for script files, whether input or output. These file names must be given in UPPER case (with the exception of numerals, which must, of course, be in lower case shift). This warning applies also to the script formation in that upper case and lower case alphabetics are decidedly different characters. Careful note should be made of this fact when constructing verifier lines.

3. Terminating Operation of STECO

When an experimenter has finished using STECO, he issues the quit command, which consists of typing a "Q" character followed by two "DELETE" characters. Control is then returned to the Operating System and the teletype may be disconnected by "hanging up" in the normal CTSS manner. Disconnection frees that channel 0 line for the later use by the Simulator. However, if the experimenter subsequently intends to perform the Script Loading phase, described in the next section, he should leave the Model 37 teletype connected, since it may also be used by the Script Loader program.

C. Script Loading

1. Introduction

Once the user has established a "library" of symbolic script files his next step is to make these scripts available to the Simulator. A separate step is necessary to accomplish this. This step consists of selecting the scripts to be used in a given simulation, and using the Script Loader program to transfer these script files from magnetic tape onto the PDP-8 magnetic disc for subsequent use by the Simulator.

The function of the Script Loader program is to read the tape script files, perform error checking on the format of the script files, notifying the operator of errors, reformat the script for use by the Simulator, and finally place the specially formatted script on the magnetic disc. It also keeps a record of where each script starts and places this "directory" on the first "page" of the disc (a page is 128 words long). This directory is later used by the Simulator to determine where to begin reading the disc for each script.

Since the Script Loader does perform error checking on the script files, it may also be used to pre-pass scripts independently of the Simulation to check for errors. Operation in either case is the same.

An optional feature of the Script Loader program is the ability to list the script files on the Model 37 teletype as they are loaded. If this option is specified, the Script Loader program prints a line-numbered listing of the script file. If the experimenter suspects that there might be bugs in the script file, it is advisible to use this option, since all error messages printed by the Script Loader program refer to the line numbers in

the script file, which the experimenter must otherwise count manually. In addition, if an error occurs, and the listing feature is used, the character at which the error is first detected is underlined in red.

The drawback to using this listing feature is that the loading progresses at the character rate of the Model 37 teletype. If the feature is not specified, the loading occurs at a considerably faster rate. If one has already checked his files before using the Script Loader, it is advantageous to eliminate this listing feature.

It should be noted here that during operation of the Script Loader, whenever the operator (also refered to as the experimenter) types at the ASR33 PDP-8 on-line console, the standard CTSS "#" and "@" conventions are in effect. That is, typing a "#" deletes the last single character typed; typing a "@" erases the entire last line typed at the console.

2. Operation of the Script Loader Program

As in the STECO phase, the tape labelled "SIMULATOR SYSTEM" must be on drive # 1. After bootstrapping the operating system as described in section B.2 of this chapter, the operating system will type an "R" on the ASR33 on-line console of the PDP-8. The experimenter should then type:

LOADER

on the ASR33 console, terminated by a carriage return character. The operating system will then proceed to load the Script Loader program and transfer control to it. The Script Loader program, upon receiving control, will first type:

DO YOU DESIRE SCRIPT LISTING ON THE M37 TELETYPE (TYPE YES OR NO)

If the experimenter wishes the line-numbered listing of all the script files he is about to load, the response is "YES" terminated by a carriage return character. If no listing is desired, the response is "NO" terminated by a carriage return character.

If the experimenter's response was "YES", the Script Loader will type the line:

DIAL UP THE M37 TELETYPE ON DATAPHONE CHANNEL 0

If the M37 is not already dialed up to dataphone channel 0, the experimenter should connect the M37 teletype to the PDP-8. The procedure to do this is described in section B.2 of this chapter. It is very important for proper operation of the M37 to have the PDP-8 clock set to its fast rate. The procedure to do this is also described in section B.2 of this chapter.

If the experimenter's response to the Script Loader query on the listing option was "NO" (no listing desired), no special action pertaining to the clock or M37 need be taken.

At this point, the "SIMULATOR SYSTEM" tape may be dismounted from its drive, and the experimenter's script library tape(s) mounted on the drive(s). Be sure that the drives each are dialed to a different number. Having the two drives dialed to the same number has catastrophic effects.

The next line typed by the Script Loader is:

NUMBER OF USERS TO BE SIMULATED:

This piece of information is requested for two reasons. First, the Script Loader program will only process as many scripts as the response given dictates. Second, this number is placed in a "control directory" written on page zero of the disc and used by the Simulator to determine how many users it is to simulate. The correct response is a single digit from 1 to 9,

which represents the number of scripts to be processed. If the Simulator is to be used immediately following the Script Loader, this number also represents the number of users to be simulated, and, hence, for the present implementation, must not exceed four. A carriage return must be typed immediately after this single digit. Any characters typed after the first character are ignored. As before, if errors are made, the standard CTSS "#" and "@" conventions apply for erasure of the characters and deleting of lines.

Once the number of scripts to be processed has been entered, the Script Loader requests the name and drive number for each of the script files to be processed, until the requested number have been processed. The Script Loader first asks for the script file name by typing the following, on the ASR33 console:

FILE NAME FOR SCRIPT N:

where N is the number of the current script to be processed. The experimenter should respond with a one to six alphanumeric character file name for the script which is to be processed next. The file name should be left justified, since spaces are significant. Immediately following the last character of the file name, the operator should type a carriage return character signifying the end of the line. No extra blanks should be inserted anywhere in this response. As before, the standard "#" and "@" conventions apply.

The Script Loader, then requests the drive number upon which this file is to be found, by typing:

TAPE DRIVE NO:

The proper response is a one digit number from one to eight representing the drive number upon which the file is to be found. This should be followed immediately by a carriage return character. Again, the "#" and "@" conventions apply.

The Script Loader then attempts to open the file, with the given name, on the given drive. If the file is found and opened properly, the loading proceeds. If an error in opening the file occurs, an appropriate error message is typed, and the Script Loader requests the file name and drive again; this time, hopefully, the correct information will be supplied.

The Script Loader will keep requesting file names and drive numbers to process script files until it reaches the number given in the number of users request. Upon completion, the Script Loader types:

SCRIPT CHECKING AND DISC LOADING COMPLETED!

and then returns control to the operating system.

If, in the course of processing a script, the Script Loader comes upon an error in script format, it will print a message describing the type of error which occured and the corresponding line number. If the Model 37 teletype print option was specified, the character causing the error is underlined in red in the listing. The Script Loader then attempts to process the rest of the script as if no error had occurred. This sometimes results in errors being indicated on lines following the original error, where really no error exists, since the first error threw the Script Loader program out of synchronization. However, the

Script Loader is designed so that it will eventually re-synchronize itself, usually within a few lines. A list of the error messages and their interpretation appears in Appendix G.

D. User Simulation

1. Introduction

Once the experimenter has loaded the PDP-8 disc unit with the scripts to be used for a simulation, he may proceed with the user simulation. It is during this phase of the operation of the Simulator System that the multiple user simulation takes place, under control of the scripts previously loaded onto the disc.

2. Operation of the Simulator

The user simulation phase of the Simulator System requires the least amount of human interaction of the phases constituting the System. The operator should make sure that the bootstrap loader is in the PDP-8 core memory and that the clock rate switch is in the fast rate as described in section B.2 of this appendix. In addition, a DECTAPE preformatted and containing only the operating system, OS8, in the first 74_8 blocks of the tape should be mounted on tape drive #2. The Simulator System master tape should be mounted on tape drive #1.

The operating system must then be bootstrapped into the PDP-8 core memory as described in section B.2 of this appendix. Once the ready message is printed on the ASR33 console of the PDP-8, the operator should type "SIMULAT" followed by a carriage return. This causes the PDP-8 operating system to load the Simulator program into core and then begin execution.

The first action of the Simulator program takes is to open the tape file on drive #2. This may be verified by the operator by noting that the

tape on drive #2 rocks back and forth several times during opening. Once the file is opened, user simulation may begin. At this point, the communications lines connecting the PDP-8 to the TSS to be tested must be dialed up. The operator should dial up only as many lines, as there are users to be simulated. The operator may dial the lines up in any order. However, only lines numbered zero (0) through n-1, where "n" is the number of users to be simulated, may be used. Simulation for each user commences at the time that the lines corresponding to that user is connected by means of pushing the "data" button on the dataphone connected to that line.

During the operation of the Simulator program, the panel lights of the PDP-8 should continually glow, and the copy tape on drive #2 should occasionally move. These are indications that the Simulator program is in operation. Error conditions are related to the operator by means of error messages typed on the PDP-8 console by the Simulator program.

When the Simulator program has exhausted the scripts for the users to be simulated, it signals termination of operation by typing:

"NORMAL END OF SIMULATION"

on the PDP-8 console. The tape file on drive #2 is then closed, and control is returned to the OS8 monitor.

If for any reason the Simulator should terminate abnormally or the operator wishes to terminate the simulation before the normal end the tape file on drive #2 will be lost, unless the operator forces closing of the file. This is accomplished by halting the PDP-8, if it is not

already halted, by pressing the stop button. Then execution should be re-started at location 602_8 in field Ø. This is the start of the tape closing routine. If this procedure is not followed, the file will not be properly closed, as it would be if a normal termination occurred. This renders the file unreadable by the OS8 file system.

E. Post - Simulation Phase

During the operation of the Simulator, a magnetic tape file is created. This magnetic tape file contains a copy of the first 31 characters of every line of the text transmitted by the Simulator and received by the Simulator. Thus after the simulation, the operator may list this file using a special program described later, to obtain a "hard copy" of all the transactions which occurred during the simulation. In the initial implementation of the Simulator, only the first 31 characters of each input and output lines are copied. This restriction arises due to the severe limitations on core memory space available in the PDP-8 for buffers. This limitation, however, is not binding, and may be changed in later implementations. This is the same restriction which applies to verification lines, that is, only the first 31 characters of an input line may be used for verification purposes.

The tape file is in a very simple format, and is easily manipulated through the use of the PDP-8 OS8 File System. The use of OS8 File System is described in MAC Memo M-339. The file name for this file to be used when employing the OS8 File System is

"SIMOUT"

The file consists of the PDP-8 "standard" 128 (12 bit) word blocks. Each block is divided logically into four 32 word sections. The first 12 bit word of each section contains the "USER ID" for the test that follows. This is a binary number starting at "0" (zero), for the first user, to "n-1", for the "n^{th}" user. The next 31 twelve

bit words contain 31 right justified ASCII (7 bits plus parity) characters comprising the text which was copied. Each block of 128 words contains four such sections of an ID followed by 31 ASCII characters. If the line which was copied was less than 31 characters long,and it was terminated by a NL character, there will be random characters after this Nl character. If the line was equal to or greater than 31 characters in length, no NL character will appear, and all characters should be taken as significant parts of the text. In general, there will be lines consisting only of an "ACK" character (MULTICS and CTSS control character = 006 octal), followed by garbage. These lines should be ignored.

As part of this thesis, one program will be included on the "SIMULATOR SYSTEM" tape to provide a listing of the SIMOUT tape file. This program is called LISTER. It will produce a listing on the Model 37 Teletype console of all the lines on the SIMOUT tape file on a per user basis. To use LISTER, the experimenter should mount the SIMULATOR SYSTEM tape on a drive dialed to #1. The tape containing the output file should be mounted on a drive dialed to #2. OS8 should be loaded, and the Model 37 Teletype should be dialed up to the PDP-8 on dataphone channel 0. The clock should be set to its fast (rightmost) position. (The above procedures are described in detail in section B.2 of this chapter). When OS8 responds with its ready message "R", the operator should type:

LISTER

followed by a carriage return. This will bring the LISTER program into core memory.

The LISTER program will then request the number of the user for which the file is to be listed. If the operator wished to list the transactions for user #3, he would respond with a "3" followed by a carriage return. When the LISTER is done listing the files for the simulated user requested, it will repeat its request for another user number. If more user files are to be listed, another number corresponding to a simulated user should be given. When the experimenter is finished using the LISTER, he should respond with a "Q" on the ASR33 typewriter in place of a user number. This will cause the LISTER program to terminate and return control to the operating system, OS8.

Programs similar to the LISTER may be constructed to analyze instead of simply list the SIMOUT file.

This empty page was substituted for a blank page in the original document.

APPENDIX B - Description of New Dataphone Interfaces For the PDP-8

The purpose of this appendix is two-fold. First it is designed to explain the operation of the dataphone interfaces on the PDP-8 in general. Second, it provides the reader with enough information to add more dataphone interfaces to the PDP-8, should the need arise. For the purpose of this thesis, only three additional dataphone interfaces were added to the PDP-8. This provides the Simulator with the facilities for simulating four users over four individual dataphone lines, since one dataphone interface already existed.

The design of the dataphone interfaces was completed by Fred Luconi as a part of his Master's Thesis entitled: "REAL-TIME BRAILLE TRANSLATION SYSTEM"; M.I.T., 1964. This design was modified slightly to eliminate an unneeded feature, the IO skip instruction. Otherwise, the design remains intact.

The dataphone interface is a group of DEC "Flip Chips" designed to interface the PDP-8 computer with a Bell Telephone 103A data set. It is necessary to use such an interface, since data-set signals are not directly compatible with PDP-8 logic levels. Exact circuit diagrams and point-to-point wiring charts for the additional three dataphone interfaces may be found in the maintenance files of the PDP-8 at Project MAC, MIT.

Program Operation Using the Dataphone Interface(s)

Characters are transmitted over the dataphone lines in serial-bit fashion. Character recomposition on input, and decomposition on output to achieve this bit-serial fashion must be carried on under program control. For output, the program places a bit of a character into a specified bit of the AC corresponding to the specific dataphone line to be used. The program then executes the IOT instruction for transmission, and the dataphone line is set to its proper sense, either 0 or 1 depending on the bit placed in the AC. For input another IOT instruction reads the dataphone line(s), and places the binary representation of their status into the proper bits of the AC. It is up to the program to time the reception and transmission of the bits so that the proper bit rate is achieved.

Background

The PDP-8 is a small computer with a 12 bit word. When an instruction is fetched from core storage, it is placed in a 12 bit Memory Buffer. There the first three bits are examined for the operation code. Of the eight instruction codes possible, the one representing an octal six(6) is the IOT instruction. Bits 3 through 8 (where the bits are numbered 0-11) of an IOT instruction are the device select code. Bus drivers in the processor make both the binary 0 and 1 output signals of the Memory Buffer Flip Flops available to all devices over the I/O bus. Bits 9, 10, and 11 of the instruction control the enabling of the IOP pulses which are also made available to the devices over the I/O bus. (See figure B-1)

The Dataphone Interface Components

The dataphone interface consists of three parts. The first is the device selector section. This section is a group of diode gates which produce a signal only when the proper device number is present in bits 3-8 of the instruction. Since the device code for the dataphone is 61 (octal), a signal will appear at point M, only when the Memory Buffer bits 3-8 contain a 61 (octal). These gates are followed by an inverter, which reverses the sense of the signal so that when the dataphone interface is selected, a ground signal is present at output "B". This signal is gated with the IOP4 pulse, amplified by a pulse amplifier and is available at output "A".

The second portion of the dataphone interface is the output subsection. Upon device selection, and having bit 9 of the instruction set to "1" (enabling IOP-4) this subsection takes the contents of AC bit "i", and converts it into the proper signal for the dataphone input for transmission over the line. This subsection is duplicated for each AC bit to be transmitted. It operates in the following manner: The device selection signal in combination with the IOP-4 pulse, amplified by a pulse amplifier are logically combined with the signal from the AC buffer Flip Flop "1" output, to set the output flip flop labelled Module B in figure B-2. The signal is inverted and changed to a level of either 0 of -15 volts by Module E of figure B-2. Module E's output is input to Module F which converts the signal to levels of +7 or -8 volts required by the data set. This signal is held by the Flip Flop until changed by subsequent IOT instructions.

FIGURE B-1: Device Selector Sub-section

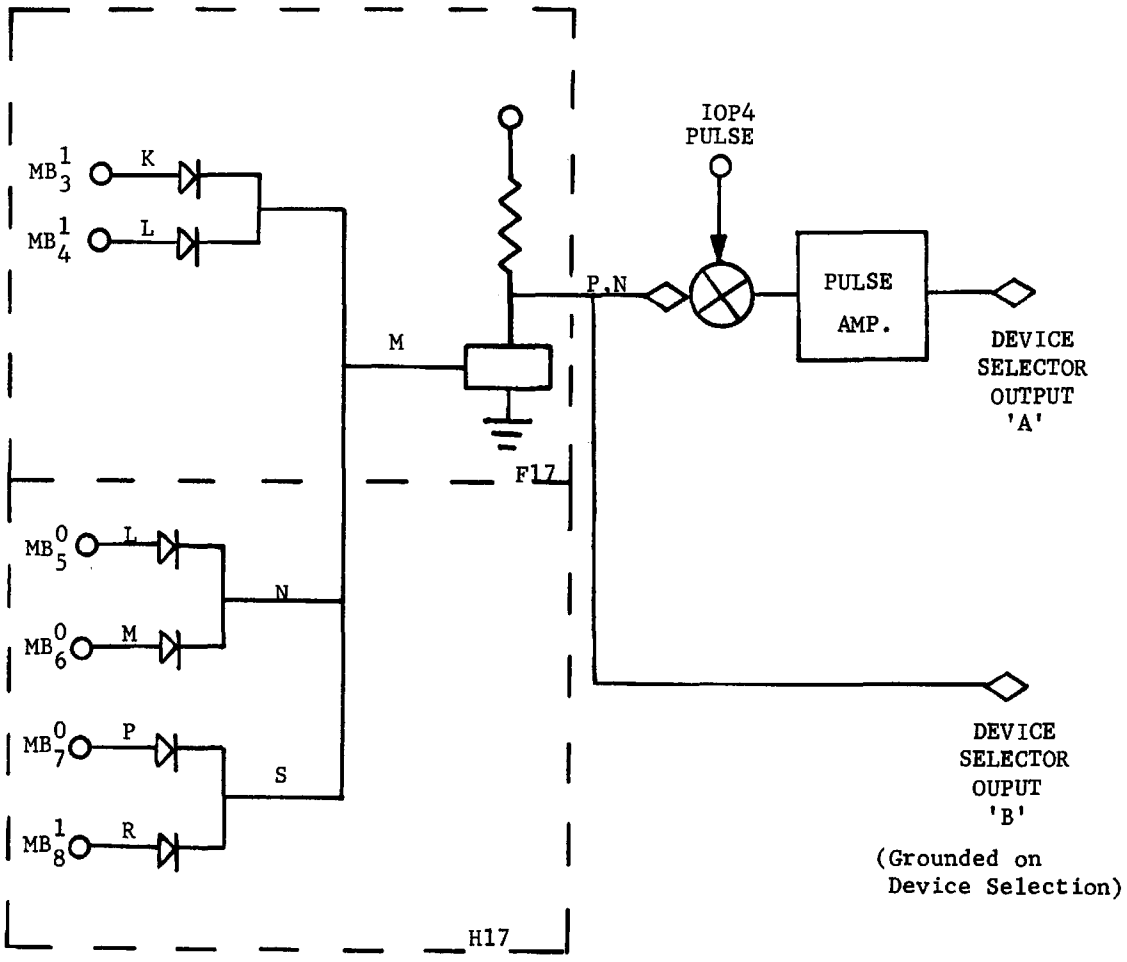
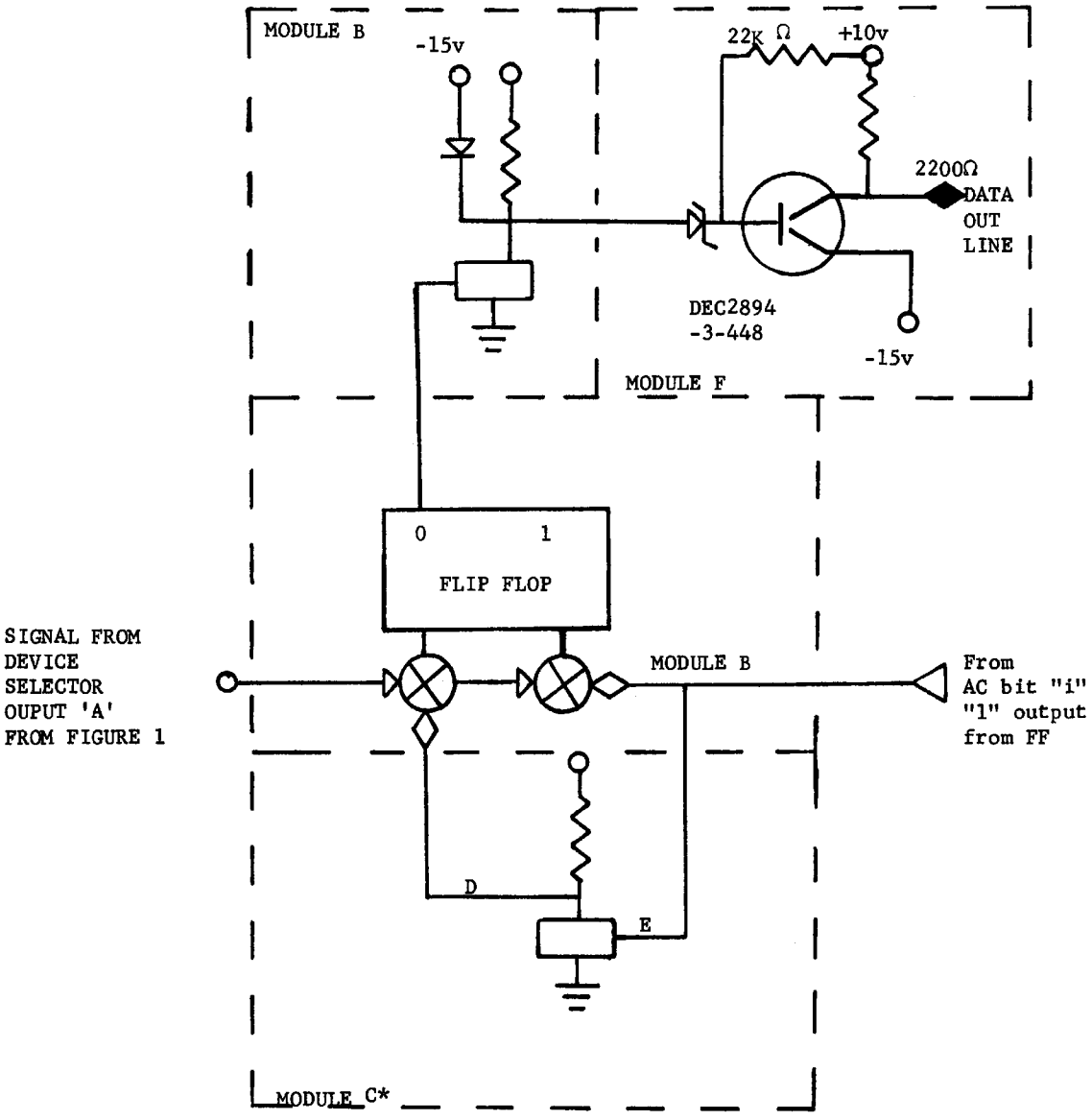


FIGURE B-2. Single Bit Output from AC to Data Line (bit "i" of AC)



The third portion of the dataphone interface, is the input subsection. Its function is to read the line from the dataphone, and strobe this signal, properly converted to PDP-8 logic levels, into AC bit "i". It operates in the following manner. The device selector signal logically combined with the signal from the dataphone and the IOP-2 pulse are strobed into the AC bit "i" by the pulse amplifier (see Figure B-3). This section, too, must be duplicated for each dataphone setup.

Since there will be many dataphones used in the experiment, there must be one input module and one output module for each dataphone. Each input module will be logically paired with an output module, through the use of the same AC bit. Thus a dataphone line will be associated with an AC bit. However, there are two different I/O instructions for input and for output, the first enabling only the IOP-2 pulse generator, and the second, for output enabling only the IOP-4 pulse generator. The overall logical placement for multiple dataphones is depicted in figure B-4.

Constructing Additional Dataphone Interfaces

The Device Selector Module:

The Device Selector Module is not duplicated when constructing additional interfaces. Its device selection signals are used to drive many interfaces. Output "A", the output from the pulse amplifier produces a 70 ma. pulse output. Each additional interface added requires (nominally) 6 ma. input driving current, thus allowing up to

FIGURE B-3. Single bit input from dataphone to AC bit "i"

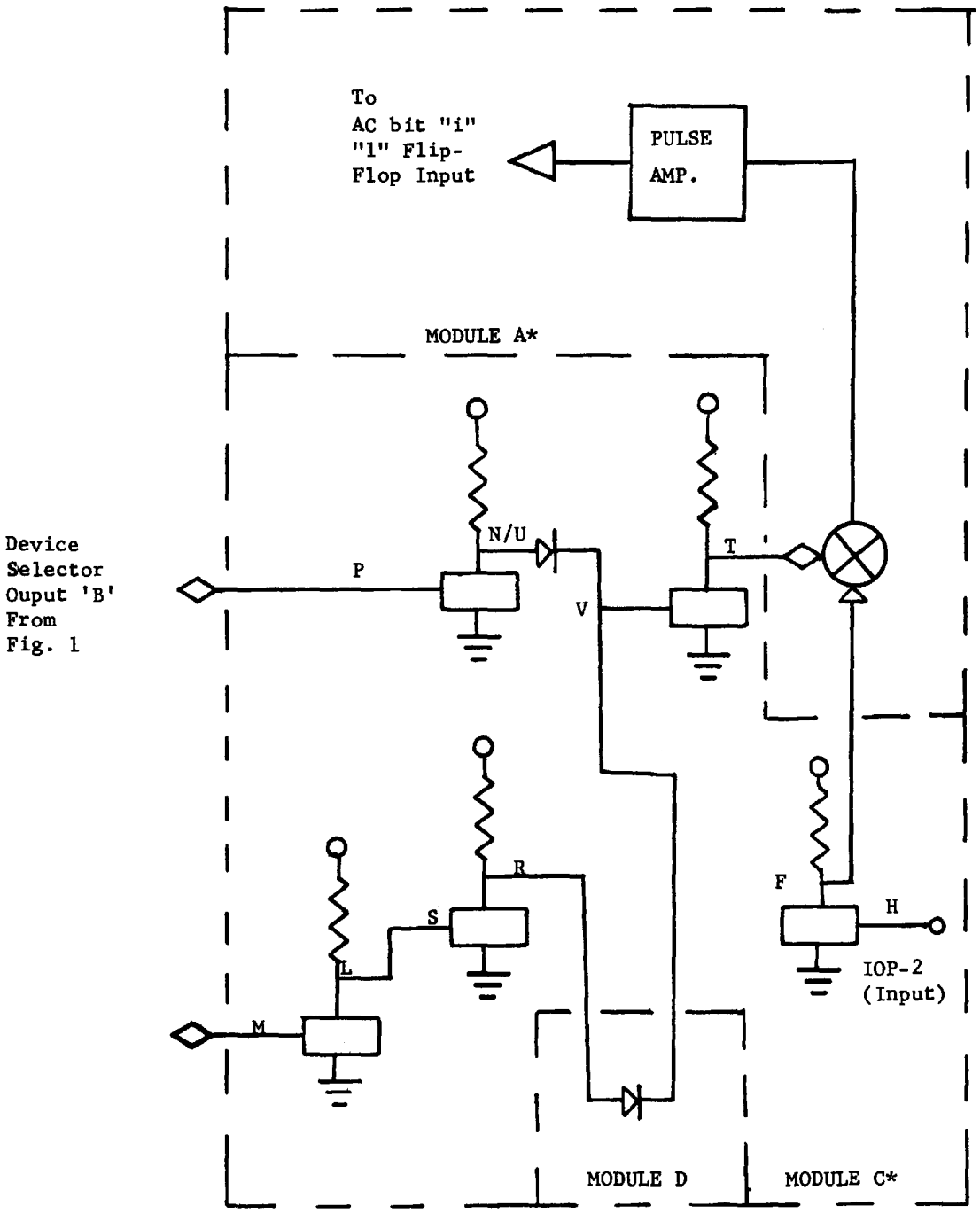
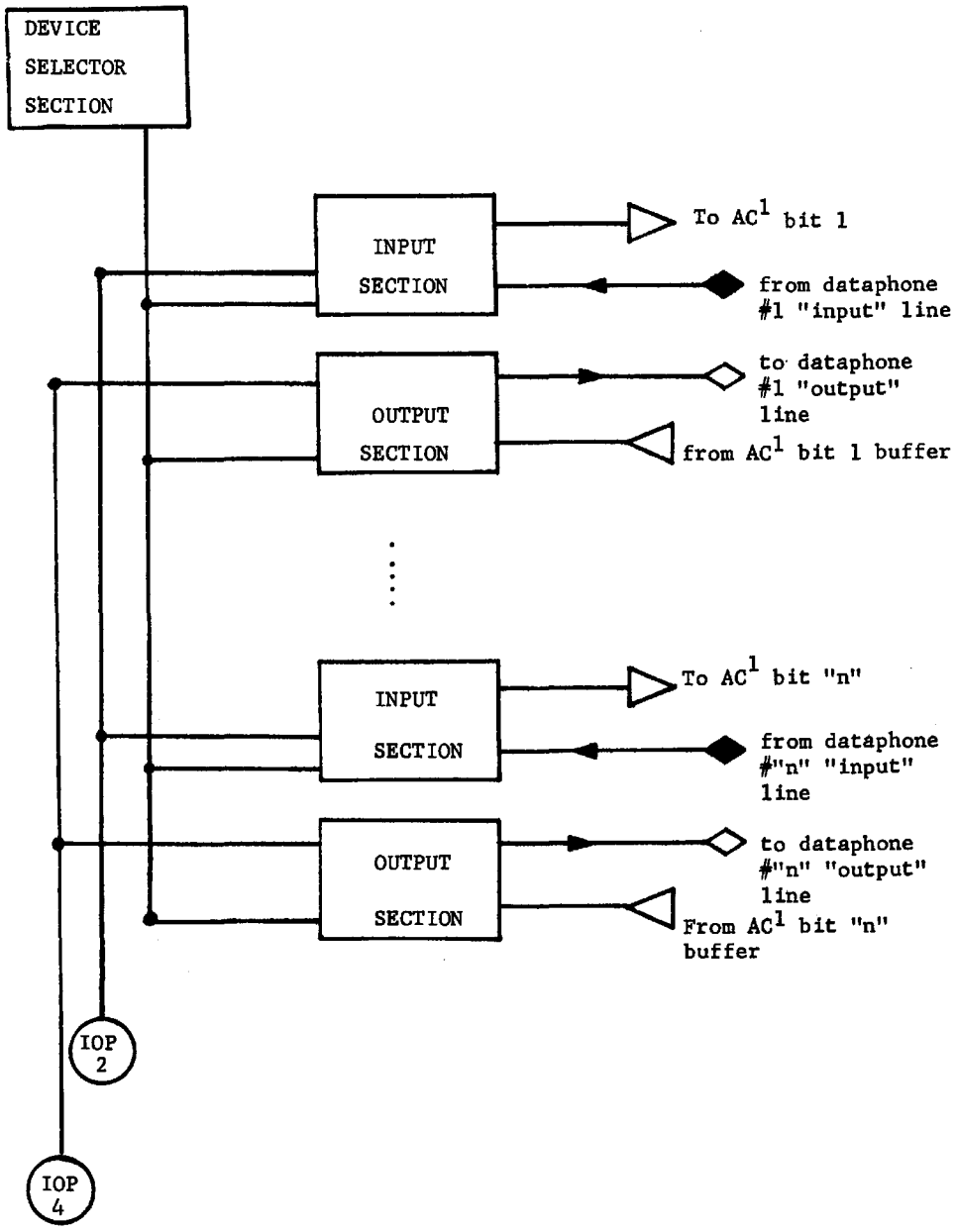


FIGURE B-4: Overall Interface Design for Many Dataphones



11 additional output subsections to be connected to the one pulse amplifier output. The output "B", output from module labelled F17 is capable of producing an 18 ma. output. The level input to the pulse amplifier is 2 ma. load. Each additional input interface subsection require a 1 ma. driving current. Thus 16 input subsections can be added with no modification to the device selection subsection.

Constructing Additional Input and Output Subsections

DEC supplies Flip Chips, which in general have several duplicated functional units per Flip Chip. For example, Flip Flops are supplied, most economically, two per Flip Chip, and Pulse Amplifiers, three per Chip. With this in mind, it is most economical to construct input and output subsections in groups of three's. That is, three input and three output subsections are to be constructed together to minimize cost.

The following parts list enumerates those Flip Chips necessary for constructing three input and output subsections for a dataphone interface.

DEC FLIP CHIP #	FLIP CHIP DESCRIPTION	QUANTITY NEEDED	MODULE DESIGNATION IN FIGS. 1 & 2
W600	3 Clamped Inverters	1	E
R001	7 Diodes (4 still unused)	1	D
W990	Blank Board	1	F
R202	2 Flip Flops	2	B
R603	3 Pulse Amplifiers	1	A
R107	7 Inverters	3	C

It should be noted that the R001 will have four unused diodes, so if still more interface subsections are constructed further economics could be realized by using these.

In addition to the required Flip Chips, other individual components are necessary to construct the module labelled F, which must be hand wired. These components are mounted on the W990 blank Flip Chip Board, and the parts for 3 interfaces will fit comfortably on one W990. The parts necessary for <u>each</u> additional interface are:

1 22,000 ohm resistor 1/2 watt

1 2,200 ohm resistor 1/2 watt

1 DEC 2894-3-448 computer transistor

1 8 volt 1 watt Zener Diode

APPENDIX C - STECO Command Summary

COMMAND	DEFINITION
A	Append page from DECtape
±nC	Character, move pointer by
±nD	Delete characters
nF	Form feed, insert at character n
F	(same as ZF)
I ... $	Insert character string at pointer
nJ	Jump pointer to character n
K	Kill entire buffer
±nK	Kill n lines
±nL	Line, move pointer by
N ... $	Next page, go to if string ... not on this one
nP	Punch (write on DECtape) n lines
m,nP	Punch characters m through n
P	(same as ZEHP)
nR ... $	Read DEC tape-file with name ... on unit n
S ... $	Search for string ...
TR	Teletype Read instead of DECtape
TW	Teletype Write intead of DECtape
UR	Unteletype Read (Read from DECtape)
UW	Unteletype Write (Punch on DECtape)
nW ... $	Write on DECtape file with name ... on unit n

COMMAND	DEFINITION
XR	Close Read file on DECtape
XW	Close Write file on DECtape
Y	Yank new page from DECtape (same as KA)
(...)	Evaluate ... and treat final pointer as number
<.. ; ..>	Iteration form
Q	quit

NOTE ON M37: "$" is DELETE key

Commands may be in upper or lower case

STECO Arguments

ARGUMENT	DEFINITION
digits	decimal numbers
+	arithmetic operator
-	arithmetic operator
B	Beginning of text buffer
Z	End of text buffer
,	separate numerical arguments
.	current pointer location
H	(same as B,Z)

This empty page was substituted for a blank page in the original document.

APPENDIX D - Character Set for the Scripts

The appendix contains the list of characters permissible for use as characters in the script. A supplementary list of characters is provided of those characters which are definitely not permissible in the script. A third list of characters used as special function characters by STECO is provided, as well. The characters in the third list are not permissible in scripts only by virtue of the fact that they are used by STECO in special ways. However, one may modify STECO so that these characters are useable in scripts, by substituting other characters in their place.

TABLE I: Legal Character Set for Scripts

	0	1	2	3	4	5	6	7
000		SOH	STX			WRU	ACK	BEL
010	BS	HT	NL	VT			RRS	BRS
020	DLE	DC1	HLF	DC3	HLR	NAK	SYN	ETB
030	CAN	EM	SUB	PFX	FS	GS	RS	US
040	Space	!	"		$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100		A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	^	_
140	`	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	|	}	~	

NOTE: Positions left blank are occupied by characters that are not legal for scripts.

TABLE II: Characters Not Permissible in Script File

OCTAL CODE	USE
003	Control character used as end of file on tape
004	Control character used to turn console off

TABLE III: Characters Used as Special Function Characters by STECO

CHARACTER	DEFINITION
#	used as single character erase character
@	used to erase entire command line
FORM FEED	used as page separator
DELETE	used as STECO escape character, and string delimiter
CARRIAGE RETURN	inserted by STECO after every NL character to provide nice looking display

This empty page was substituted for a blank page in the original document.

APPENDIX E - Implementation of the SCRIPT LOADER program

This appendix is divided into two sections. The first section describes the transformation made by the SCRIPT LOADER program as it transfers the scripts from tape to the disc. The second section is a brief description of the exact implementation of the SCRIPT LOADER program complete with flowcharts.

A. Script Format on the Disc

1. The Disc Directory Format

The first type of information written on the disc by the SCRIPT LOADER program is the Disc Directory. The Disc Directory contains the starting address of each script loaded onto the disc. This address requires two PDP-8 words, since a disc address can range from 0 to 32K, and a single 12 bit PDP-8 word can only address 4096 locations. The first of the two words contains a binary number from 0 to 7 representing the disc "field" in which the script starts. The second word is the displacement from address 0 of the field at which the script begins. The Disc Directory is used by the Simulator program during its initialization phase to determine where to begin reading of the disc for each simulated user's script. Another piece of information included in the directory is the number of scripts processed, and hence also the number of users later to be simulated.

Starting addresses on the disc for each script are kept track of during the operation of the script loader. The number of users to be simulated is input by the experimenter as a response to a Script Loader query during operation of the Script Loader. When the Script Loader has successfully completed processing its required number of scripts, the last step before returning control to the operating system consists of writing this directory out on page zero of field zero of the disc. When the Simulator later goes into operation, it initializes its data bases from this directory. The format of this control directory in page zero of field zero of the disc is:

LOCATION	
0	Number of users to be simulated
1	Field of starting address of script 1
2	Field of starting address of script 2
3	" 3
4	" 4
5	" 5
6	" 6
7	" 7
8	" 8
9	" 9
10	" 10
11	" 11
12	" 12
17	Starting address within field above of script 1
18	" 2

etc.

2. The Reformatted Scripts on Disc

The second type of information written on the disc by the SCRIPT LOADER program consists of the reformatted scripts. The reformatted scripts on the disc are very similar in format to the script files created by the user using STECO. Reformatting is done to conserve space on the disc unit, since the storage capacity of the disc is limited to 32K characters.

The format of a single character on the disc is as follows. Each character and binary integer occupies one 12 bit word on the disc. Binary integers are represented in their "true" form, that is without sign. Alphanumeric characters are stored in their 7 bit ASCII format, right justified. The bit to the left of the first of the seven bits is the parity bit, and also appears in the disc word for each character. The following discussion describes the format of each of the subsections of the script as they appear on the disc.

a) Comment Mode Subsections

One of the main function sof reformatting is to conserve space. Since comment mode subsections of the script perform no function in the Simulator phase, they are not included in the reformatted script on the disc.

b) Verifier Mode Subsections

There is no special character to signify the start of Verifier Mode Subsection (VMS). The presence of a VMS is denoted by its location within the script body. Hence, the ":VNL" string is only used by the Script Loader program, and does not appear on the disc.

Each VMS couplet is transformed in the following manner. The first line of the VMS couplet is the verifier time limit. This one, two or three decimal digit number is converted into a 12 bit binary integer, which is placed on the disc. The terminating NL character is not placed on the disc. The second line of the couplet, the verifier line, is placed on the disc character by character as it appears in the tape script file. The single exception to this is if the terminating NL character is immediately preceded by a backslash character. In this case the NL character is not placed on the disc.

Each couplet is terminated by one of two termination characters. These characters are the "Intermediate Terminator" character, and the "Final Terminator" character abbreviated "ϕT" and "ϕF" respectively. These terminator characters are necessary for the Simulator program. The "ϕT" terminator character is used to terminate a verifier couplet when more verifier couplets immediately follow it in the current VMS. The "ϕF" terminator is used to terminate the last verifier couplet of a VMS. The presence of the "ϕF" signals that the next character following it belongs to the subsequent text mode subsection.

In summary, each couplet of a VMS appears on the script as a binary integer, representing the maximum verification time,

immediately followed by the verification line characters. Intermediate couplets are terminated by a "¢T"; the final couplet is terminated by a "¢F". Characters following a "¢F" belong to the text mode subsection following the current VMS.

c) Text Mode Subsections

Text Mode Subsections (TMS) of a script tape file are transformed in a similar manner to VMS. Each couplet of a TMS is reformatted in the following manner. The first line of the couplet is the one, two, or three decimal digit number representing the "think time". As is the VMS couplet, this is converted to a 12 bit binary integer and placed on the disc. The second line of the couplet is placed on the disc exactly as it appears in the tape script file, character by character. Since all text lines must end with a NL character, the Simulator program, when later executed, senses the end of a text mode couplet when it reaches the NL character. If there are subsequent text mode couplets in the TMS, the Script Loader program places a special character, abbreviated here as "¢N", immediately after the NL character which terminates the text line. This signifies to the simulator that there is another text line to send. The last text mode couplet in a TMS ends simply with a NL without a "¢N" character following it. This is a signal to the Simulator that the next character on the disc belongs to the next VMS.

d) Script Termination

As explained in the discussion of a previous section of this chapter, there is no special character which signals the end of a script file. That is, the experimenter using STECO to produce and edit his scripts has no responsibility to type a special character to signify that the script ends at a particular point. The experimenter merely ends his file by typing the STECO command to close the currently open write file. STECO, upon receiving this command, the "XW" command, automatically inserts a special end of file character after the last character of the text before it writes out its last buffer. The Script Loader program checks for this special character. When it finds this special character, it terminates its tape file reading for the particular script it is processing. It also puts out its own special character to signify the end of a script on the disc. This character is called the ENDTXT and TXTEND character in this document. It is a 12 bit word of all ones.

B. The Script Loader Program

1. Initialization Phase

When called into execution, the first action of the Script Loader program is to request if the operator desires the "listing" option. If the operator responds affirmatively, the Script Loader continues in its initialization, since the program normally includes the listing option. If the response is "NO", the Script Loader program eliminates the calls to the teletype output routines.

Next the Script Loader program requests the operator to type the number of scripts to be processed. The operator responds with a single numeric character representing this number of scripts. This number is stored for later use by the script loader as well as being stored in the template for the disc directory. The flowchart depicting the initialization phase appears in Figure E-1.

2. READ-the Script File Opening, Reading and Unpacking

This section, depicted in the flowchart of Figure E-2, consists of requesting the operator to type the file name for the current script to be processed, and the tape drive on which this file is to be found. Once this has been done, and the responses provided, the script file is opened. If an error occurs during the open, the error message is printed out, and the requests for script name and drive repeated.

If the file is opened properly, the first three pages are read into the tape read buffer. Since the tape files are packed five characters to every three 12 bit words, the Script Loader next proceeds

to unpack the three page tape input buffer into an overlapping five page buffer. Following this some initialization of pointers and counters is done.

3. The GETNXT Routine

The GETNXT routine is used during the operation of the Script Loader Program to get the next character from the unpacked buffer. It also eliminates spurious characters from the scripts such as the Carriage Return character, and the Form Feed character which are inserted by STECO as format control characters in the tape files. If a call to the GETNXT routine finds the unpacked buffer depleted, GETNXT calls the READ routine of Figure E-2 to read and unpack a new three page block from the tape file. When control is returned from the READ routine, GETNXT gets the next valid character from the buffer and returns it to its original calling routine.

The secondary entry point to the GETNXT routine is called "PSEUD1". This entry point is used only once for each script file, after the first 3 page block has been read and unpacked. It eliminates some of the bookkeeping done by GETNXT, which must not occur for the first buffer of a script file.

If the next character in the unpacked buffer is the end of script (003) character GETNXT transfers control to the FINISH routine.

4. The GETHIS and GETOLD routines

The GETHIS routine is called from other routines in the Script Loader to retrieve a copy of the character currently being processed from the unpacked buffer.

The GETOLD routine is called to retrieve a copy of the character previous to the one being processed from the unpacked buffer. Flowcharts for these two routines are found in Figure E-4.

5. The FINISH Routine

The FINISH routine is called by GETNXT when GETNXT senses the end of the script character (0003). The FINISH routine puts the end of script character onto the disc after the last character processed, and calls the UNDISC routine to update the Disc Dictionary template. If at this time, the required number of scripts has been processed, the disc directory is written in page zero, field zero of the disc, the done message is printed, and control is returned to the OS8 operating system. If the required number has not yet been processed, the current input tape file is closed, some bookkeeping is done, and the entry "BETA" in the READ routine is transferred to, in order that the next script file may be processed.

6. The Script Loader Control Loop

Once the Script Loader has opened, read, and unpacked a script file, control is transferred to the entry point "RESTRT" of the Script Loader Control Loop. The control loop is responsible for controlling the processing of the script. The control loop examines control character strings, and calls appropriate subroutines to process subsections of the script. If errors in control string format and/or placement are found, error message printing routines are called to advise the operator of these error conditions. Then the control loop attempts to re-synchronize itself to permit further processing of the script. For a detailed diagram of the operation of the control loop, see Figure E-6.

7. The Verifier Mode Subsection Processor

The verifier mode subsection processor is called from the control loop after finding the string "NL:VNL". The function of this routine is to process all verifier mode couplets contained in the verifier mode subsection. In the case that errors are found appropriate error messages are posted on the PDP-8 on-line console. Two subroutines used in this routine are "VDISCI" and VDISC2". These serve as the interface between the VMS processor and the disc writing routines. In addition they keep count of the characters in a particular verifier line and warn the operator by means of an error message if a verifier line is too long. Operation of these routines is depicted in Figure E-7.

8. The Output Text Mode Subsection Processor

The Output Text Mode Subsection Processor routine is called from the control loop after finding the control string "NL:TNL". Its function is to process the text couplets comprising the TMS. After processing all the couplets of the current TMS, control is returned to the control loop. (See Figure E-8)

9. The Number Processing Routines

The Number Processing Routines consist of three subroutines. The major routine is NP1, with secondary entry point NP2. This routine processes the numeric portion of a verifier or text mode couplet. By calling the SAVDIG and CONVRT subroutines, the one, two or three digit number in the couplets are converted into a binary integer and put out onto the disc. Control is then returned to the calling routine, either OPTEXT, or VERLIN. In the event of an error, appropriate error messages are printed. (See Figure E-9).

10. The Comment Mode Processor

The Comment Mode Processor is called from the control loop upon occurrence of the string "NL:CNL", signifying a comment mode subsection. Since comment mode subsections are provided for documentation purposes only, and do not influence the operation of the Simulator, they are ignored. Therefore the function of the comment mode processor is to ignore characters up to the next "mode-determining" string. (See Figure E-10)

11. The Disc Buffer Routine

The disc buffer routine is called from the rest of the Script Loader routines when a character is to be appended to the disc buffer for subsequent writing on the disc. Entry point DISC1 converts the character passed to it into the same character with parity bit added. Entry point DISC2 retains the character in true form. In either case the character is appended to the disc buffer. If after appending, the buffer is full, the routine DWRITE is called to write out the buffer onto the disc, and re-initialize the appropriate pointers. Control is then returned to the calling routine. (See Figure E-11)

12. Disc Write Routine

This routine is called from the disc appending routine of figure E-11 when the disc buffer is full. The function of this routine is to write the current disc buffer onto the disc, detect disc hardware errors and check if the disc is full. In addition, the disc directory is updated, and re-initialization of the disc buffer pointers is done.

When done, control is returned to the disc appending routines of Figure E-11. If a disc error has occurred, an error message is printed on the on-line console. If the disc storage area has been exceeded, the last 50 characters of the script segment causing the overflow are printed out to facilitate re-writing of the scripts. (See Figure E-12)

13. The Disc Bookkeeping Routine

UNDISC, the disc bookkeeping routine is called upon reaching the end of a script. The function of this routine is to force the last page of buffer out onto the disc, so that the next user script begins on an even page boundary. In addition, the disc directory template is updated to reflect the starting addresses of the next script. (See Figure E-13)

FIGURE E-1. Script Loader Initialization

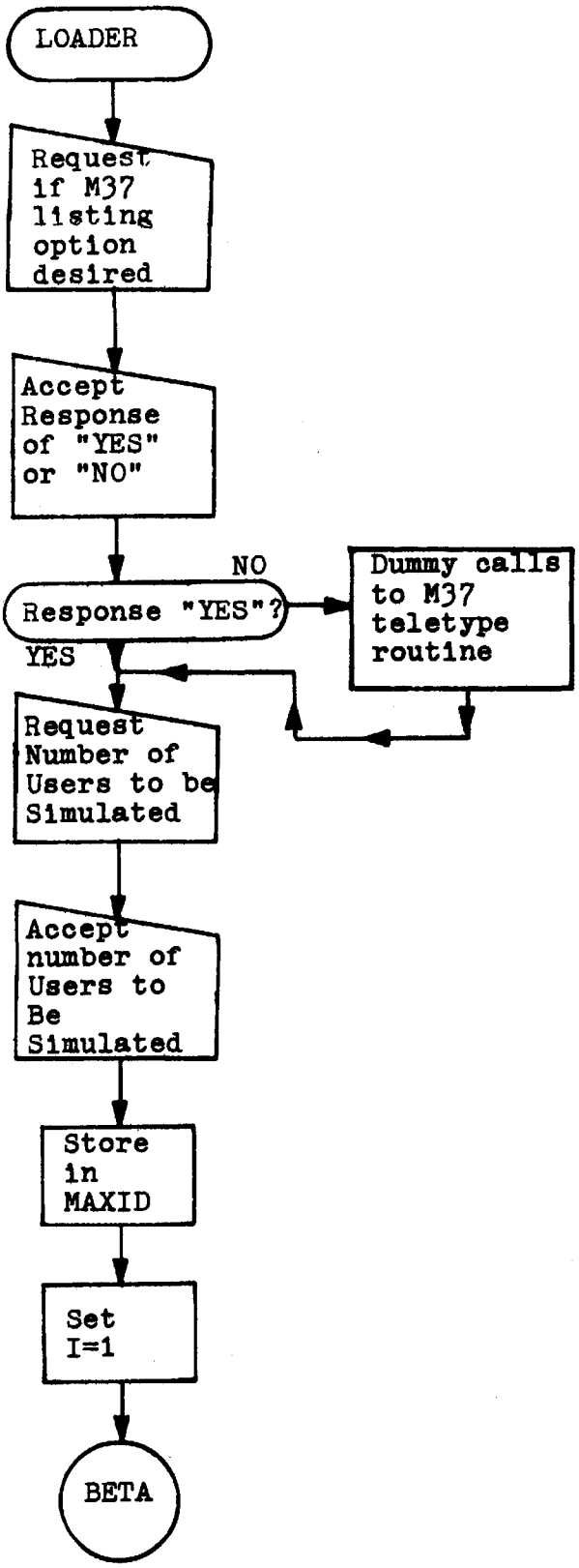


FIGURE E-2. Scipt File Opening, Reading and Unpacking

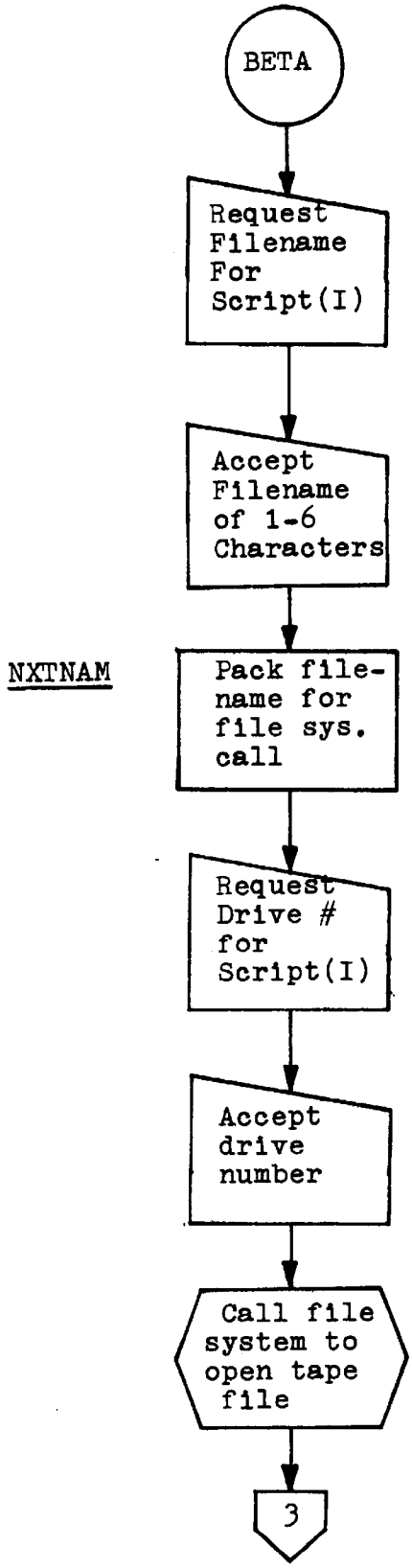


FIGURE E-2. con't

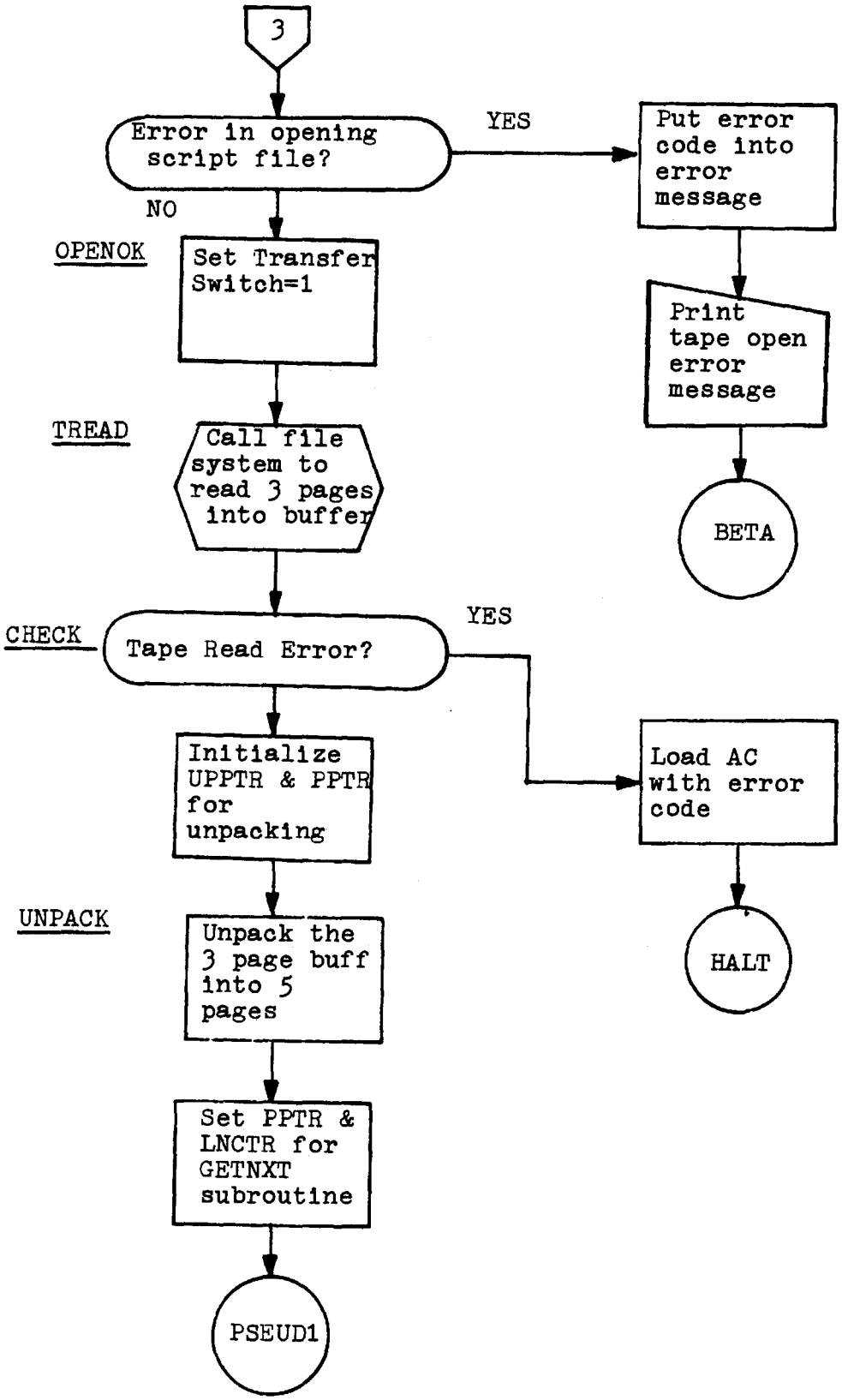


FIGURE E-3. The GETNXT Routine

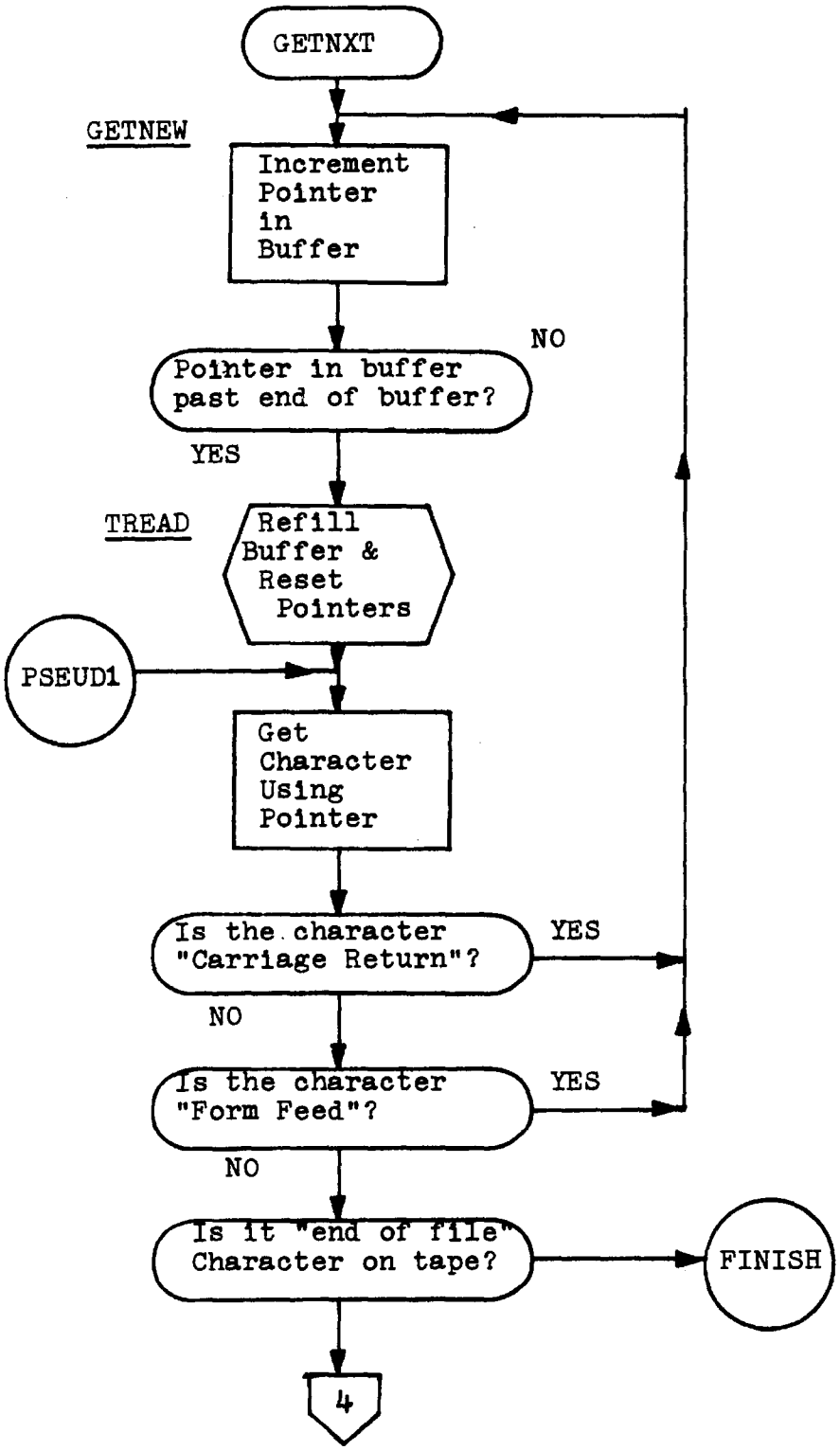


FIGURE E-3. con't

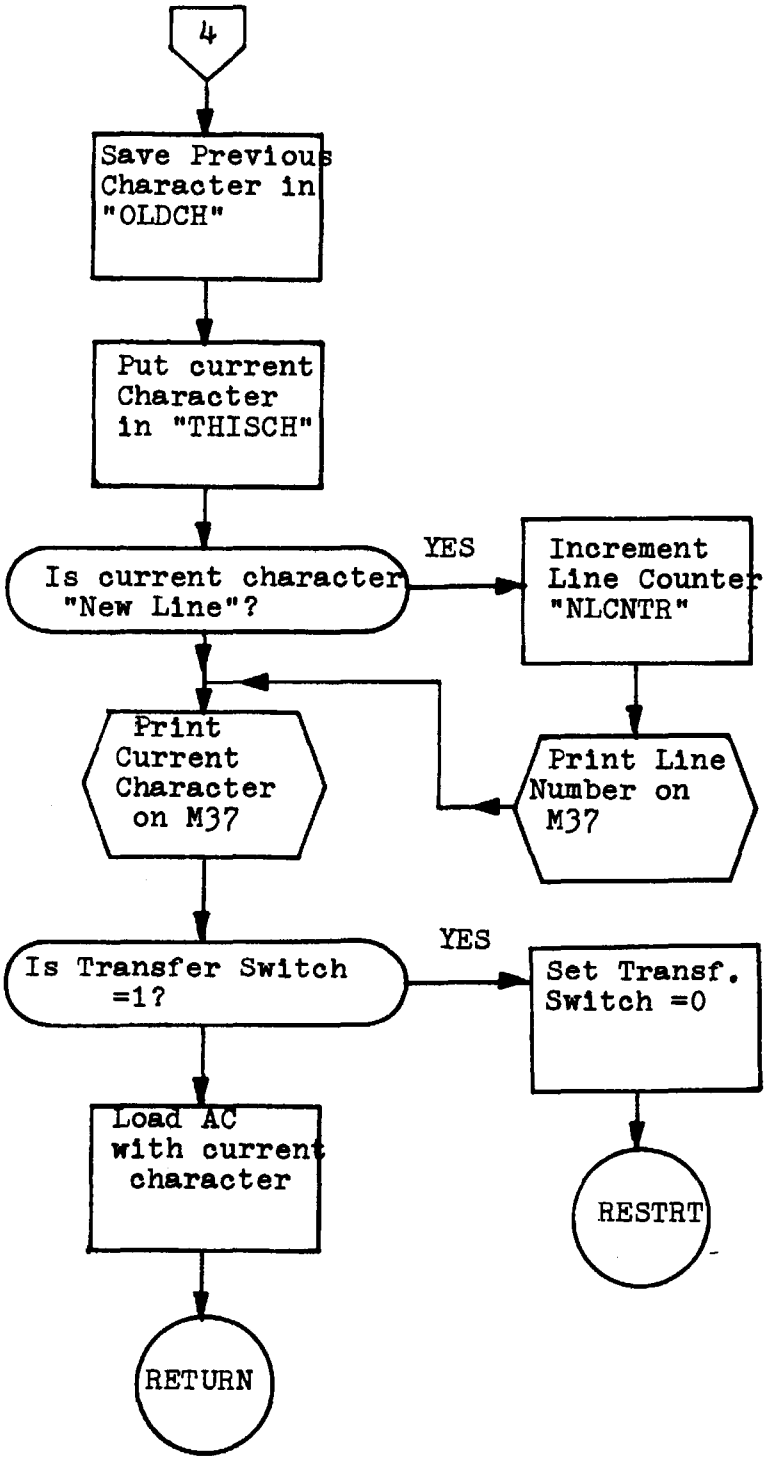
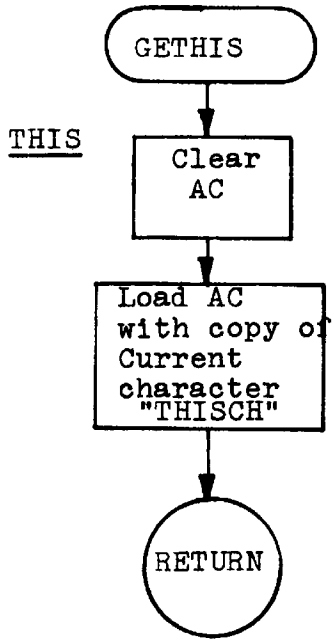


FIGURE E-4. The GETHIS and GETPST Routines



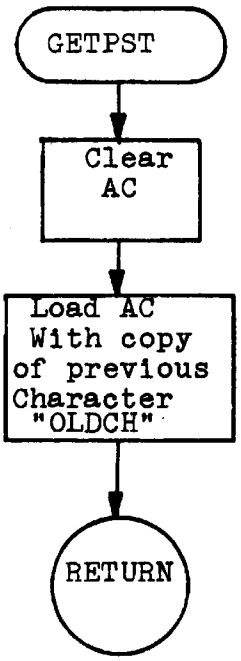


FIGURE E-5. The FINISH ROUTINE

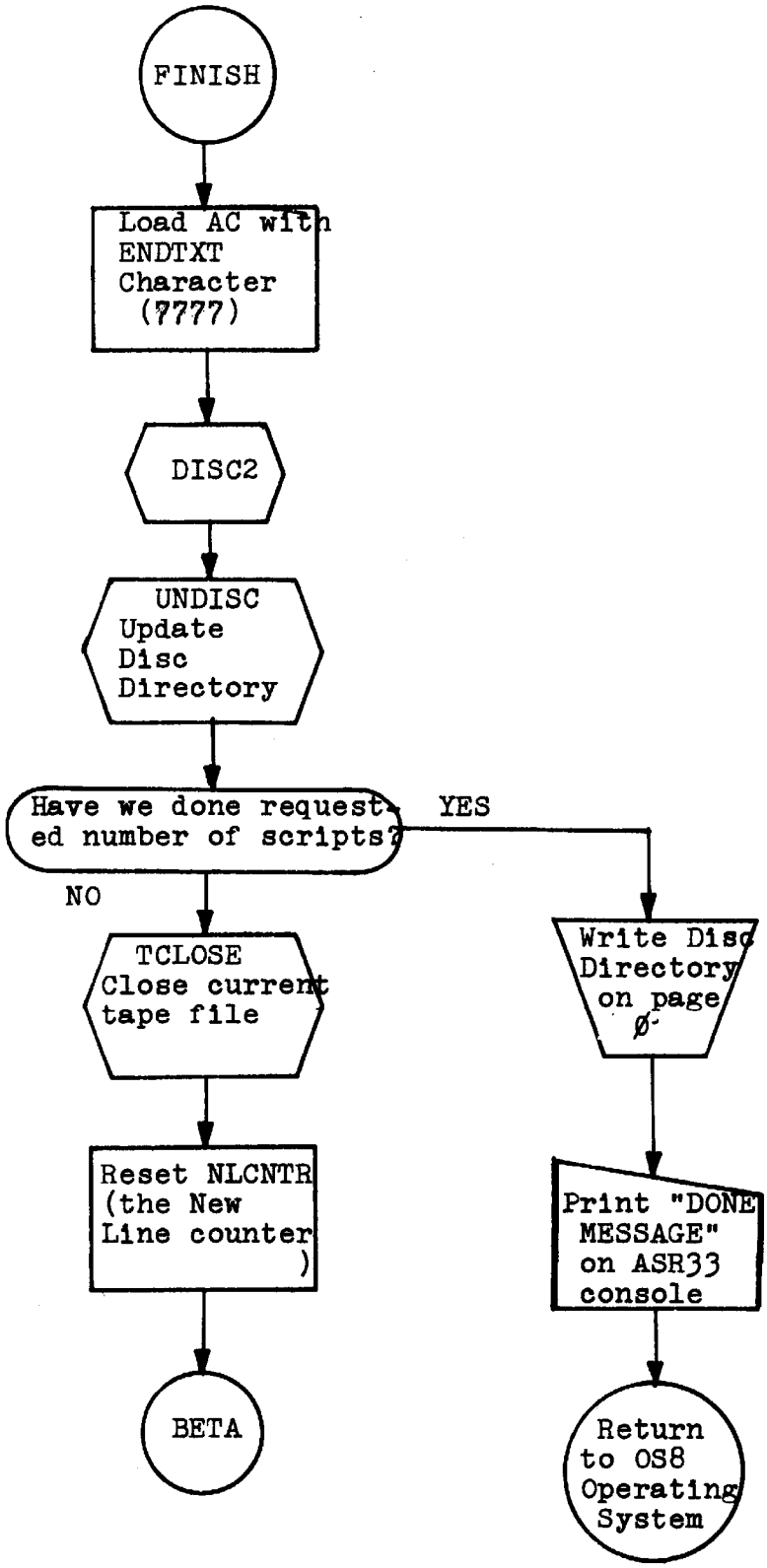


FIGURE E-6. The Script Loader Control Loop

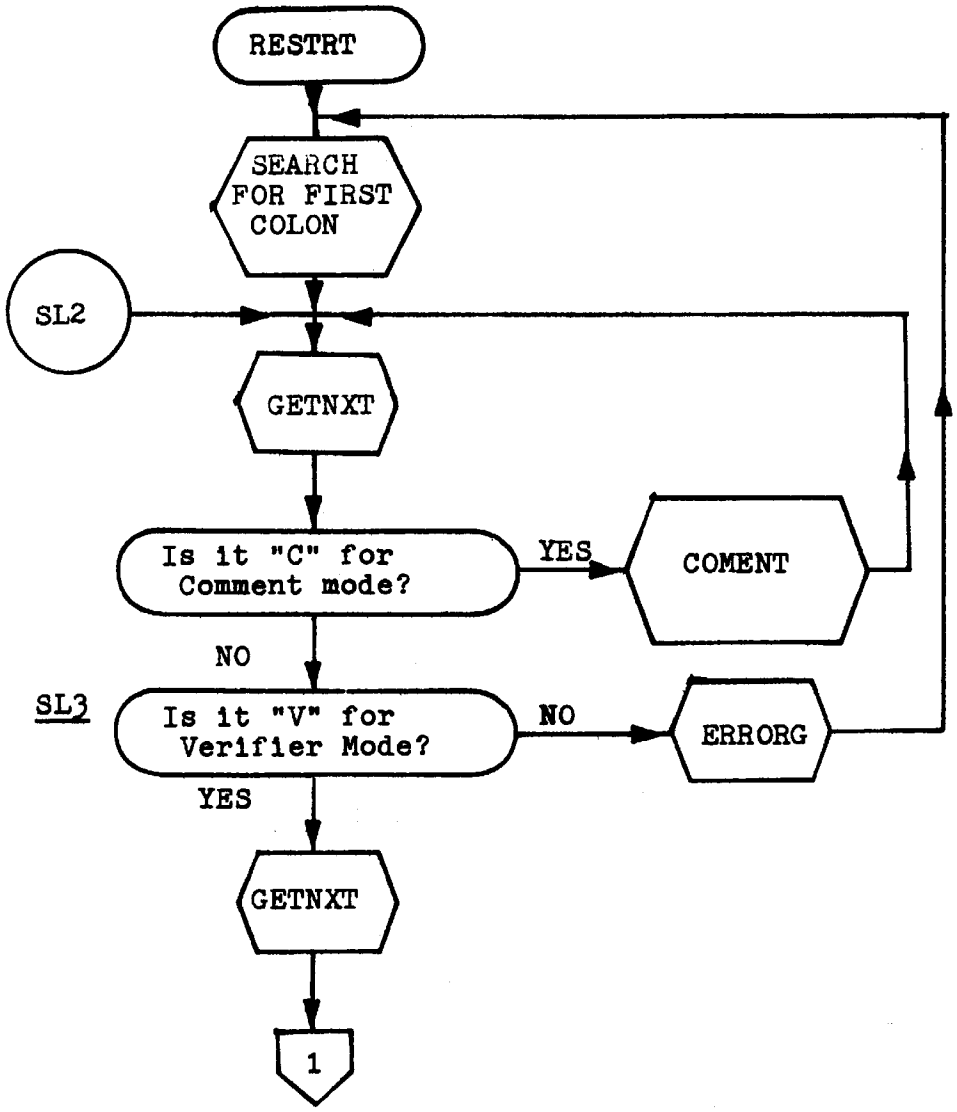


FIGURE E-6. con't

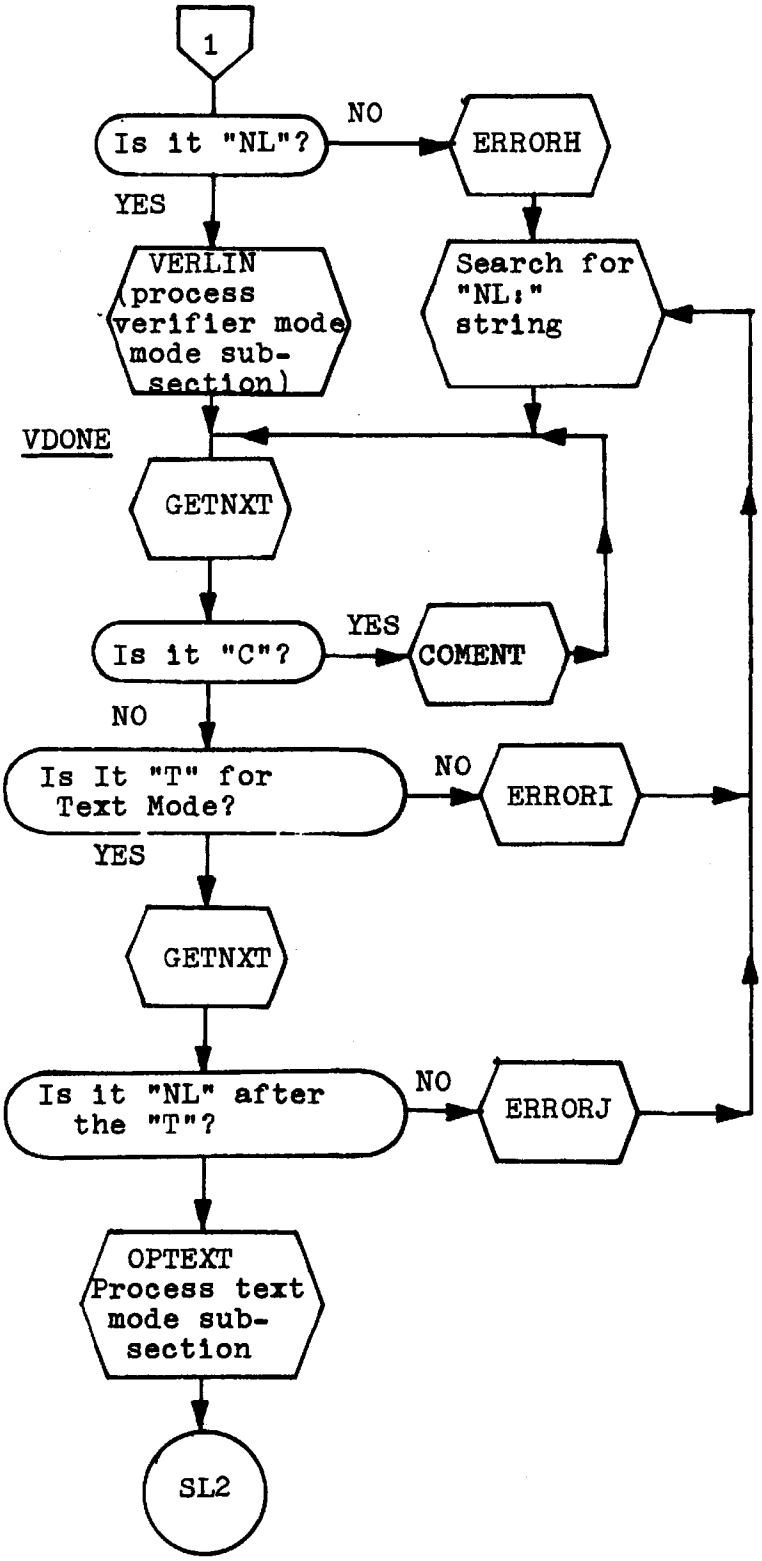


FIGURE E-7. The Verifier Mode Subsection Processor

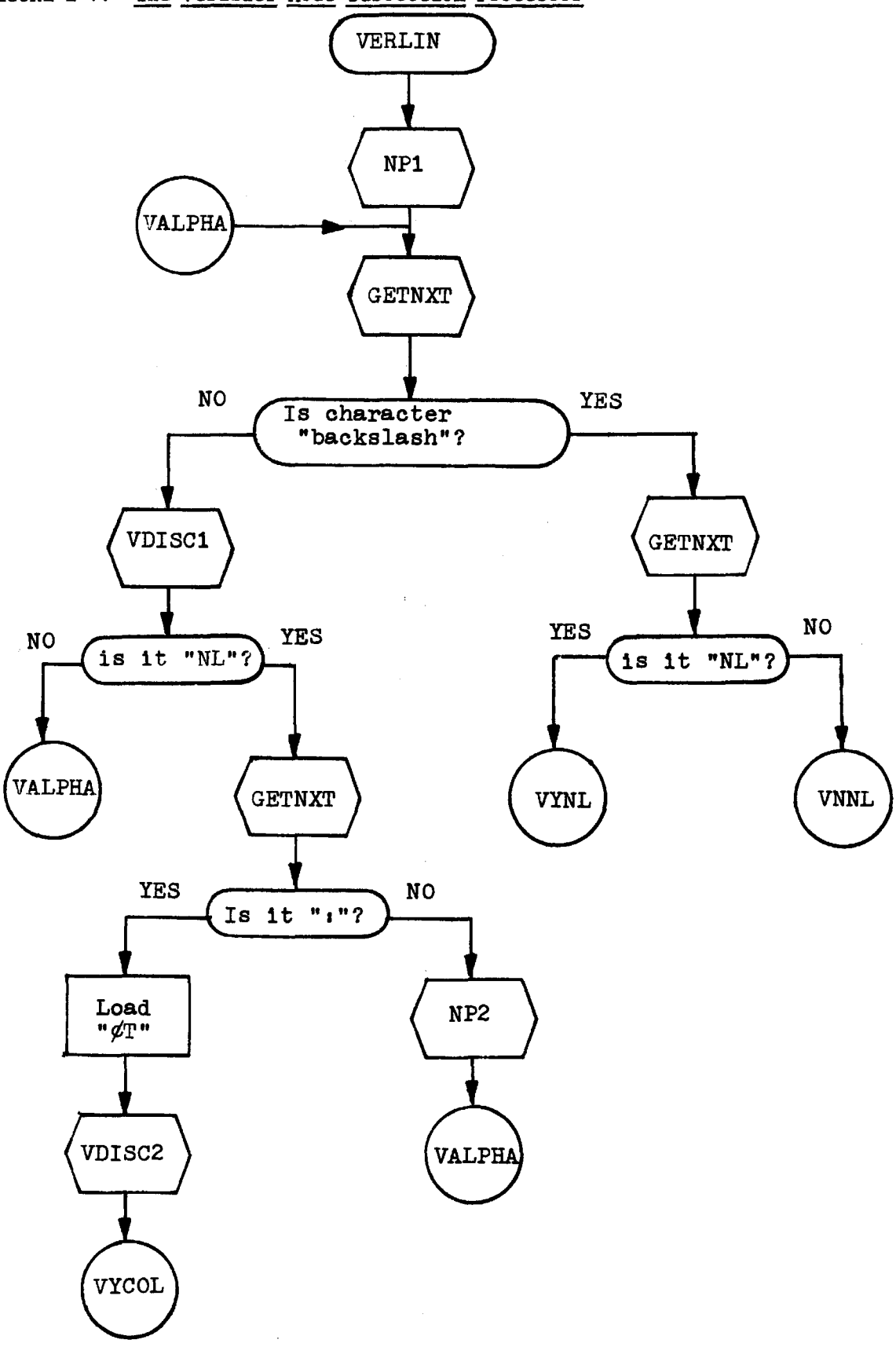


FIGURE E-7. con't

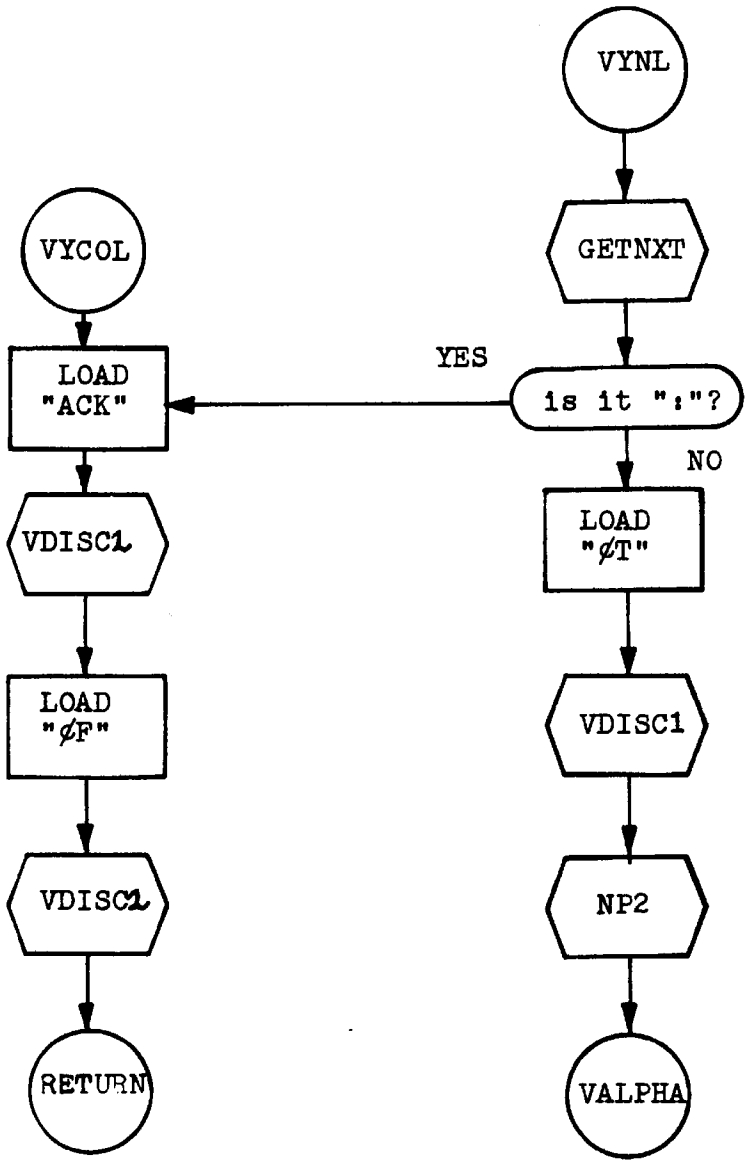


FIGURE E-7. con't

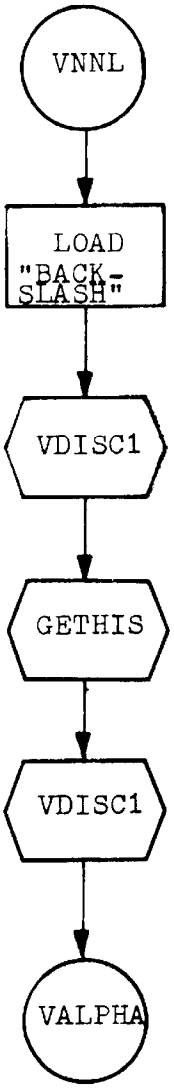


FIGURE E-7. con't

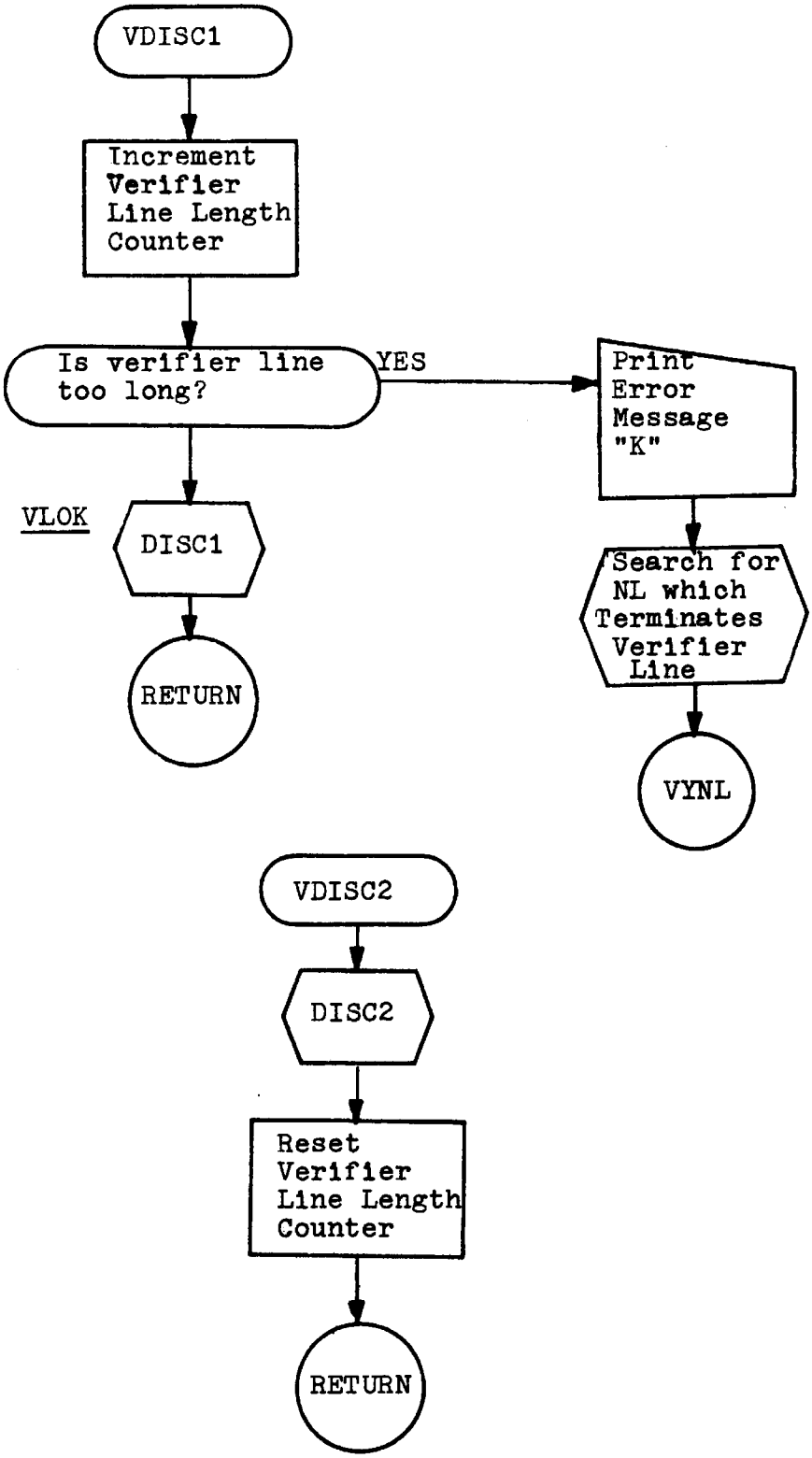


FIGURE E-8. The Output Text Mode Subsection Processor

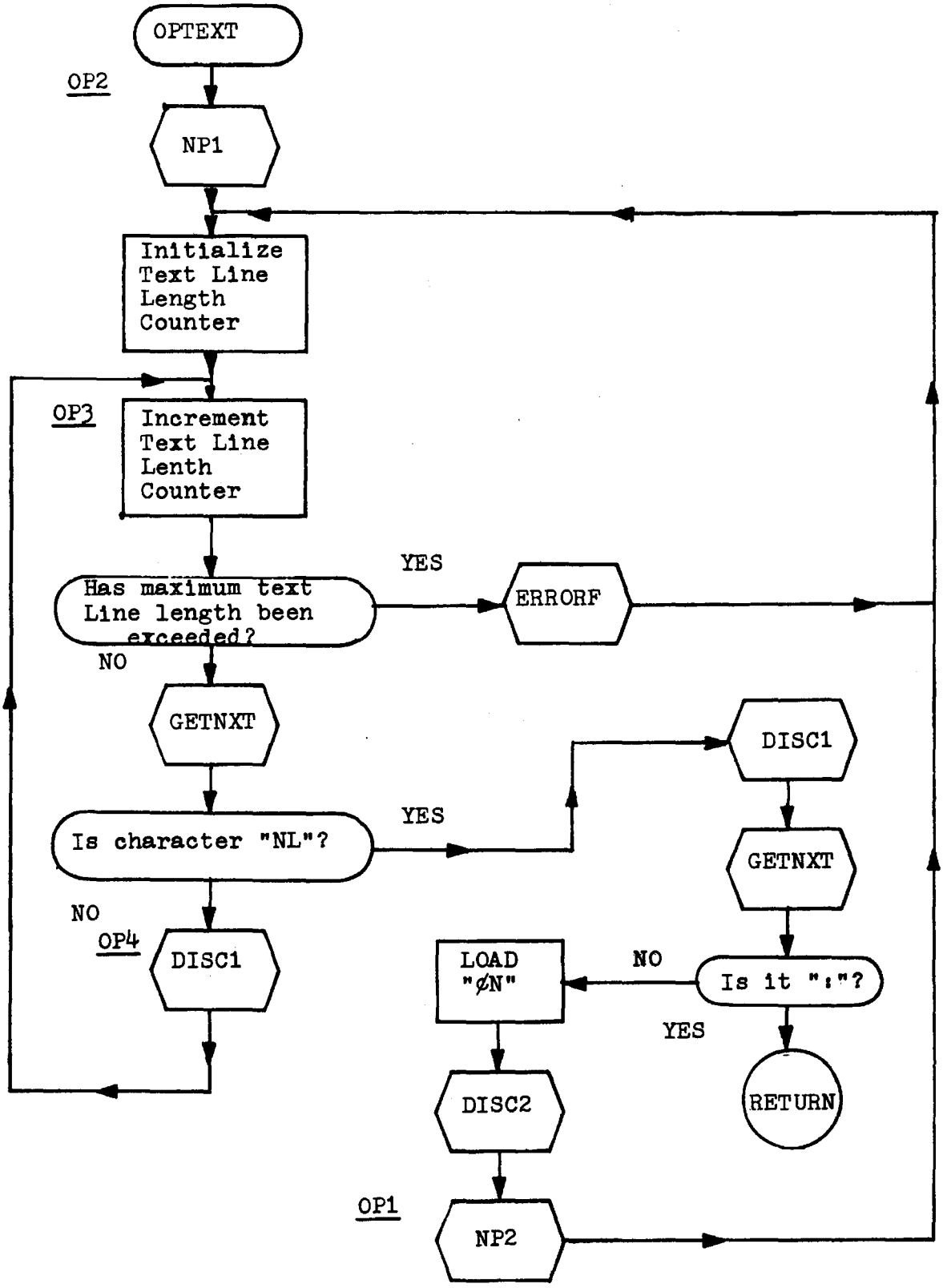


FIGURE E-9. The Number Processor Routines

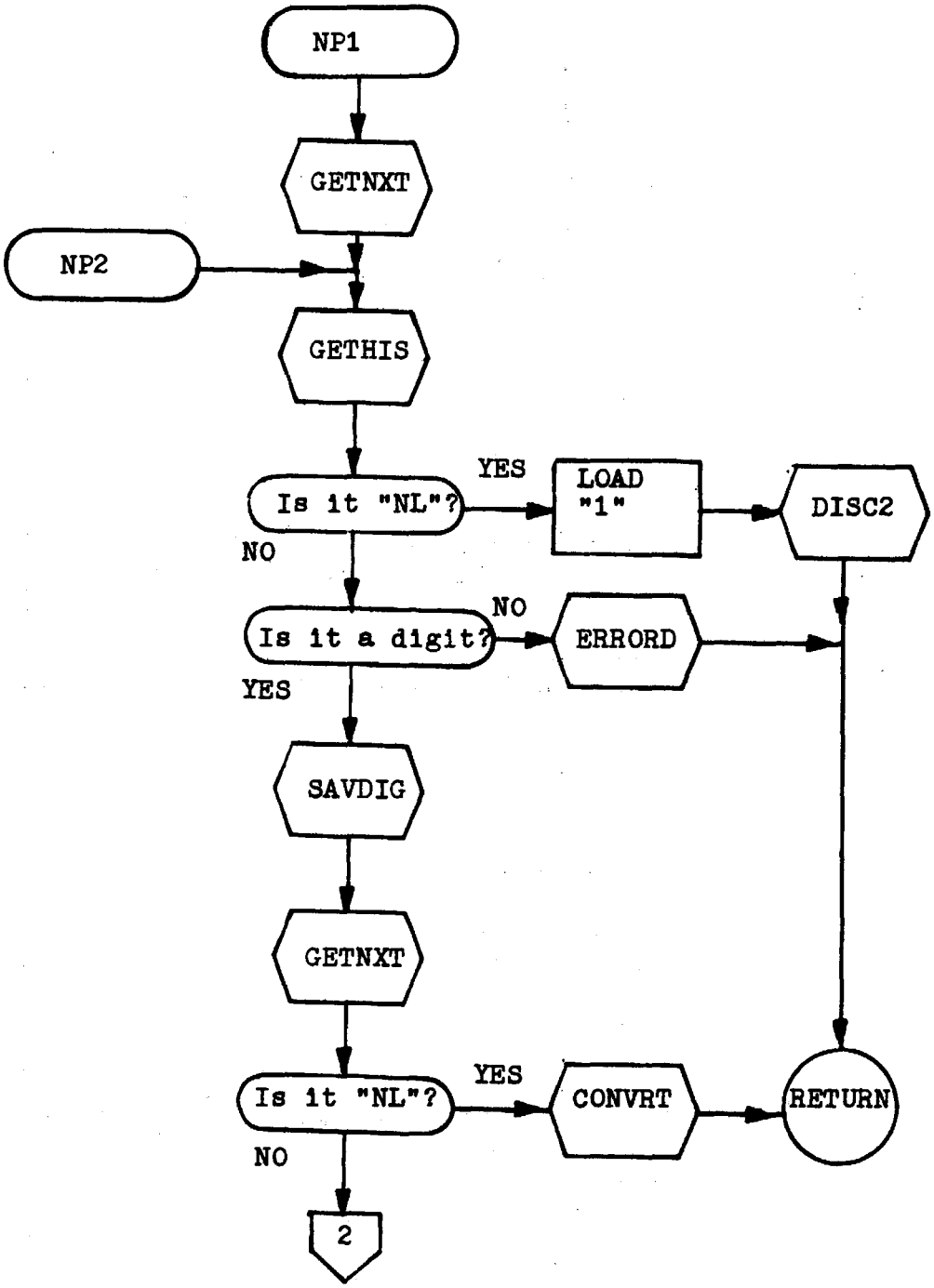


FIGURE E-9. con't

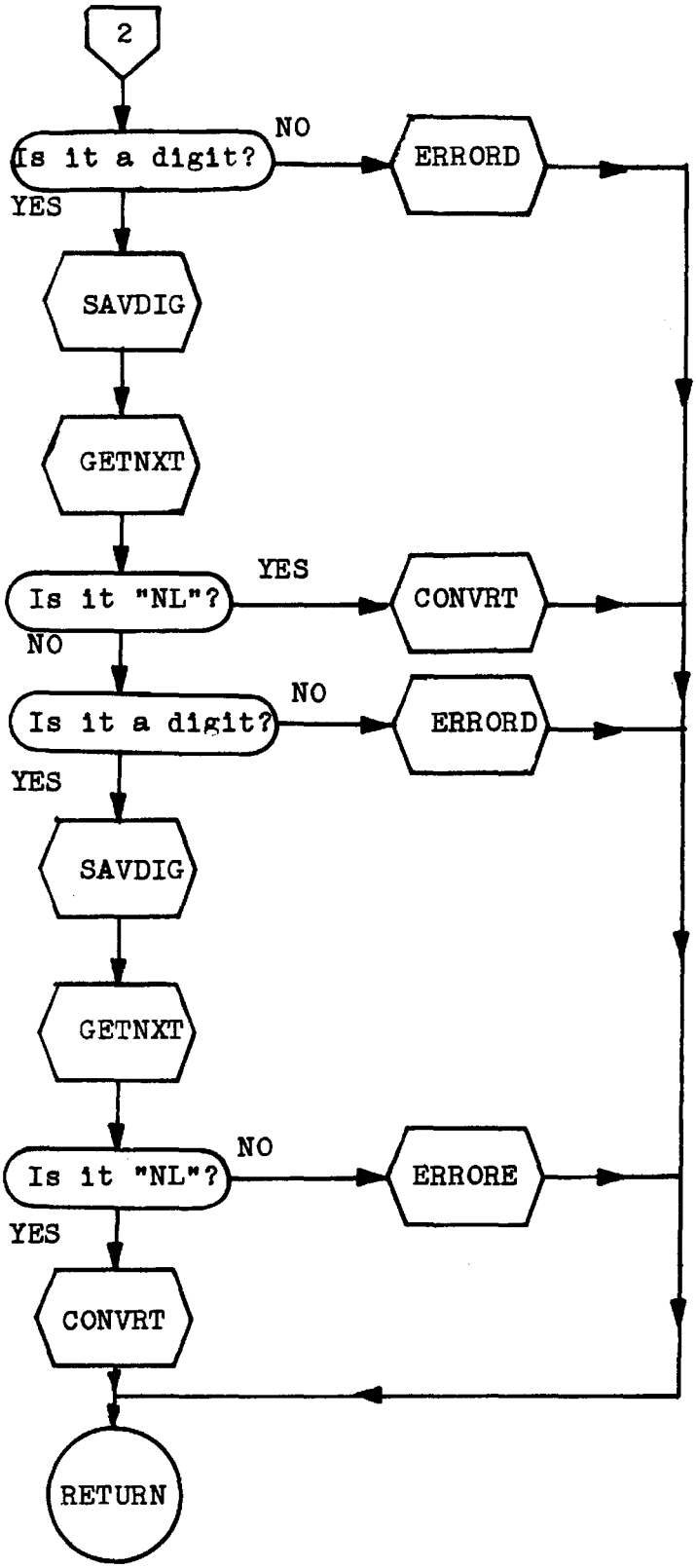


FIGURE E-9. con't

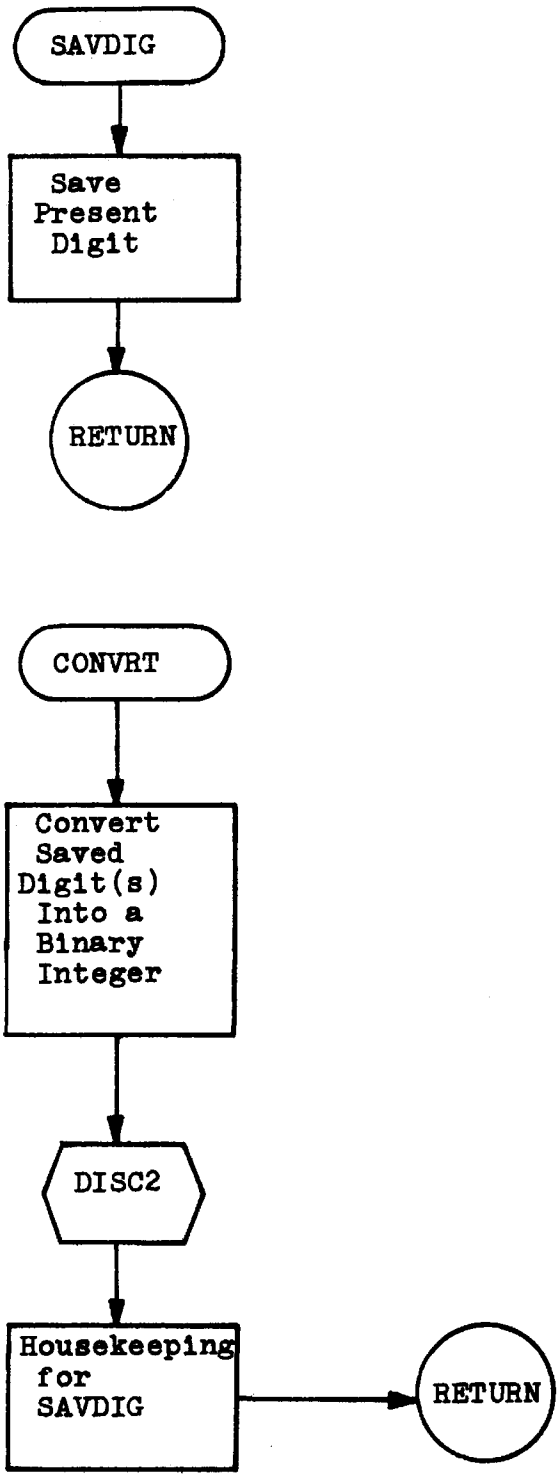


FIGURE E-10. The Comment Mode Subsection Processor

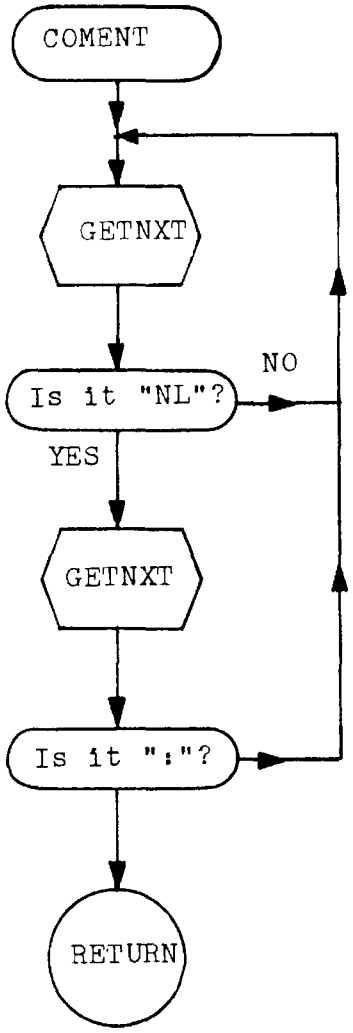


FIGURE E-11. Disc Buffer Routines

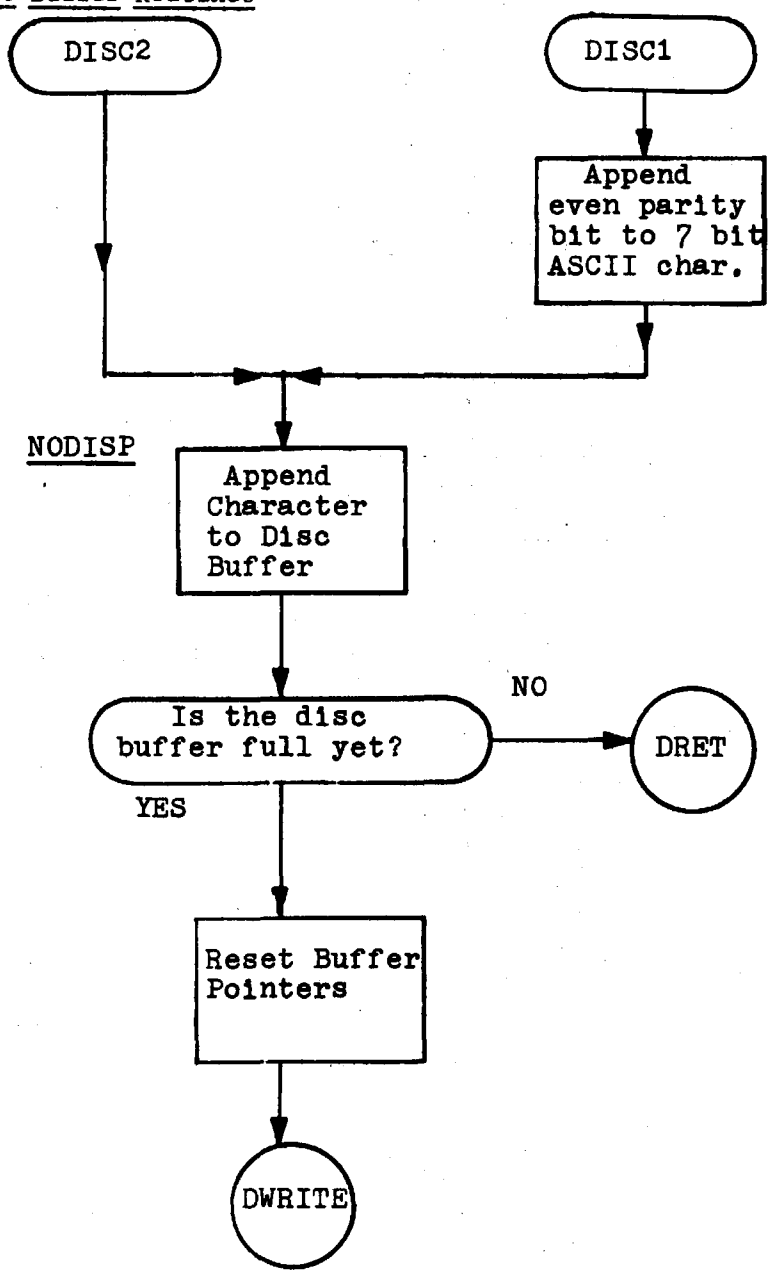


FIGURE E-12. Disc Write Routine

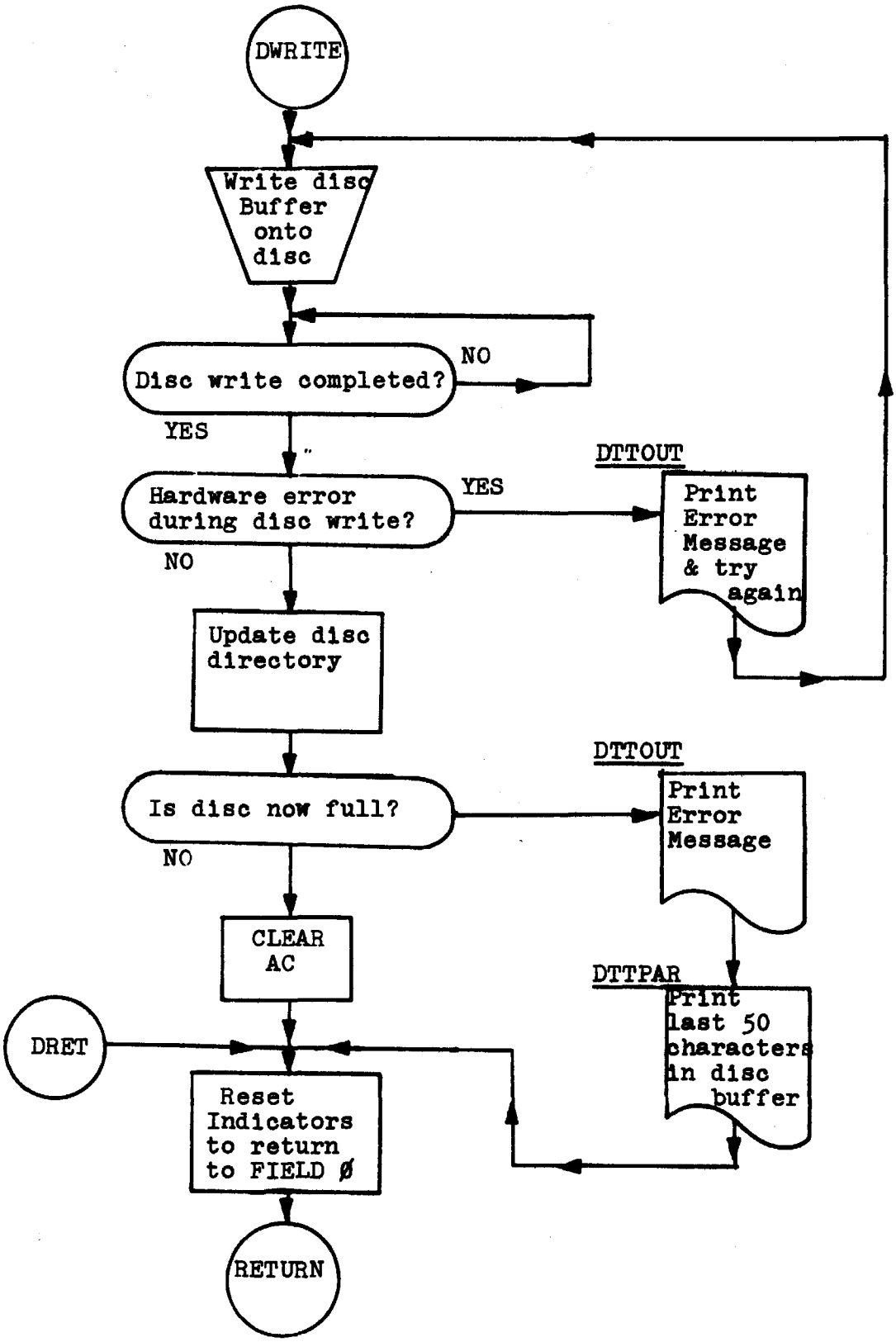
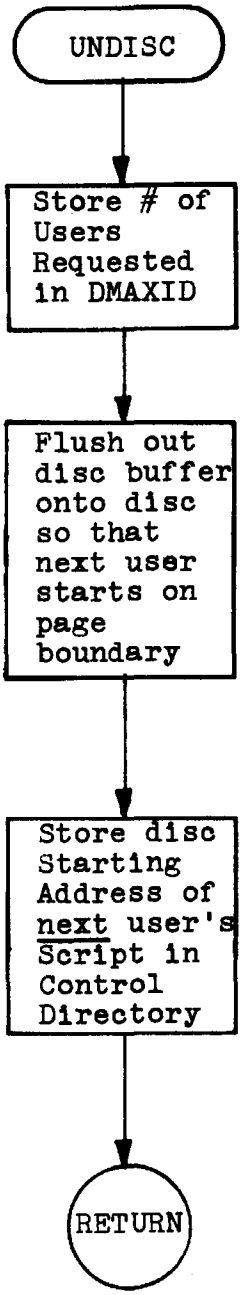


FIGURE E-13. Disc Bookkeeping Routine



This empty page was substituted for a blank page in the original document.

APPENDIX F - Implementation of the Simulator Program

This appendix contains flowcharts depicting the step-by-step operation of the Simulator program. For a less detailed description of the Simulator program's operation see Chapter IV Section E.

FIGURE F-1. Simulator Initializing Routine and Control Loop

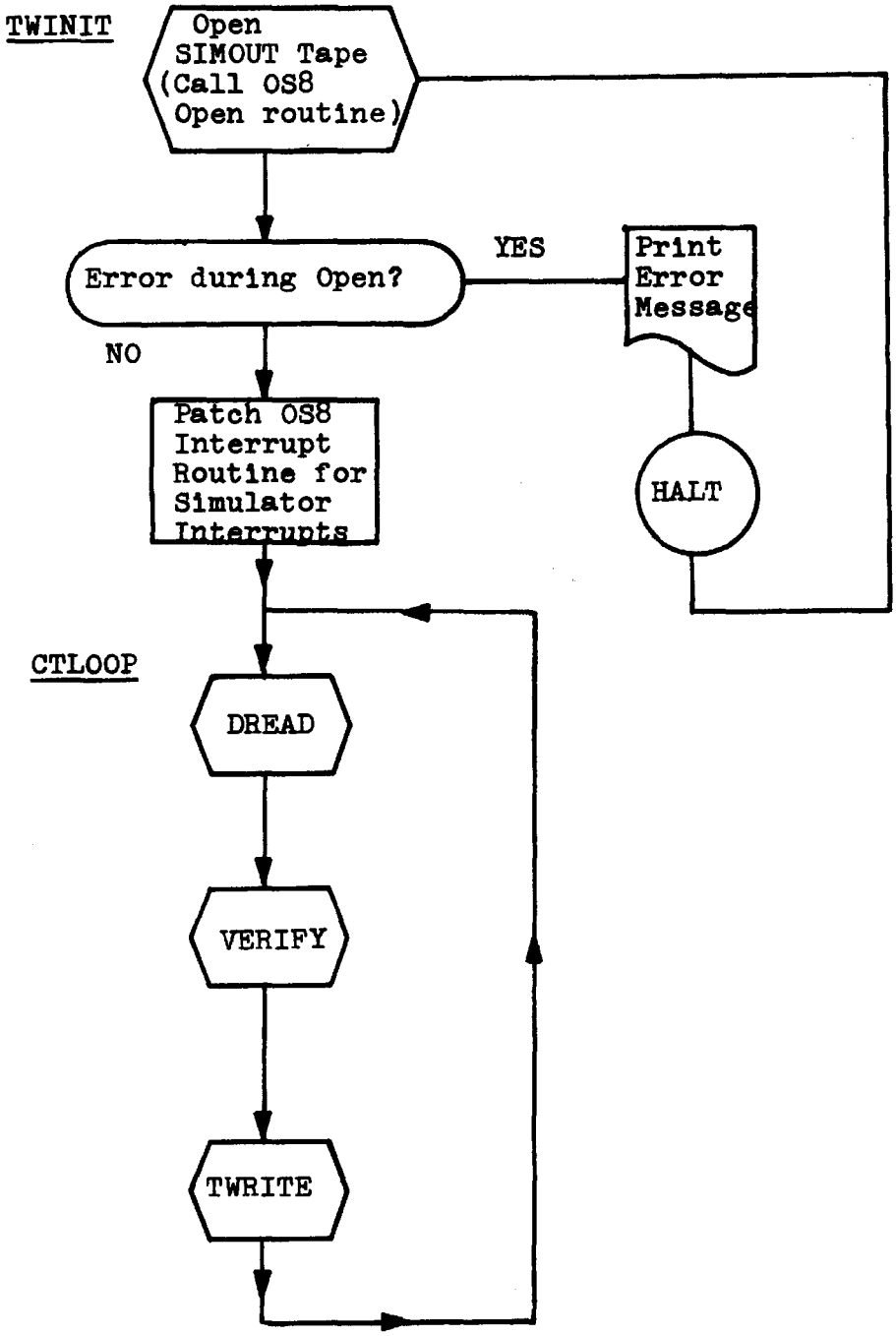


FIGURE F-2. Simulator Interrupt Handling Routine

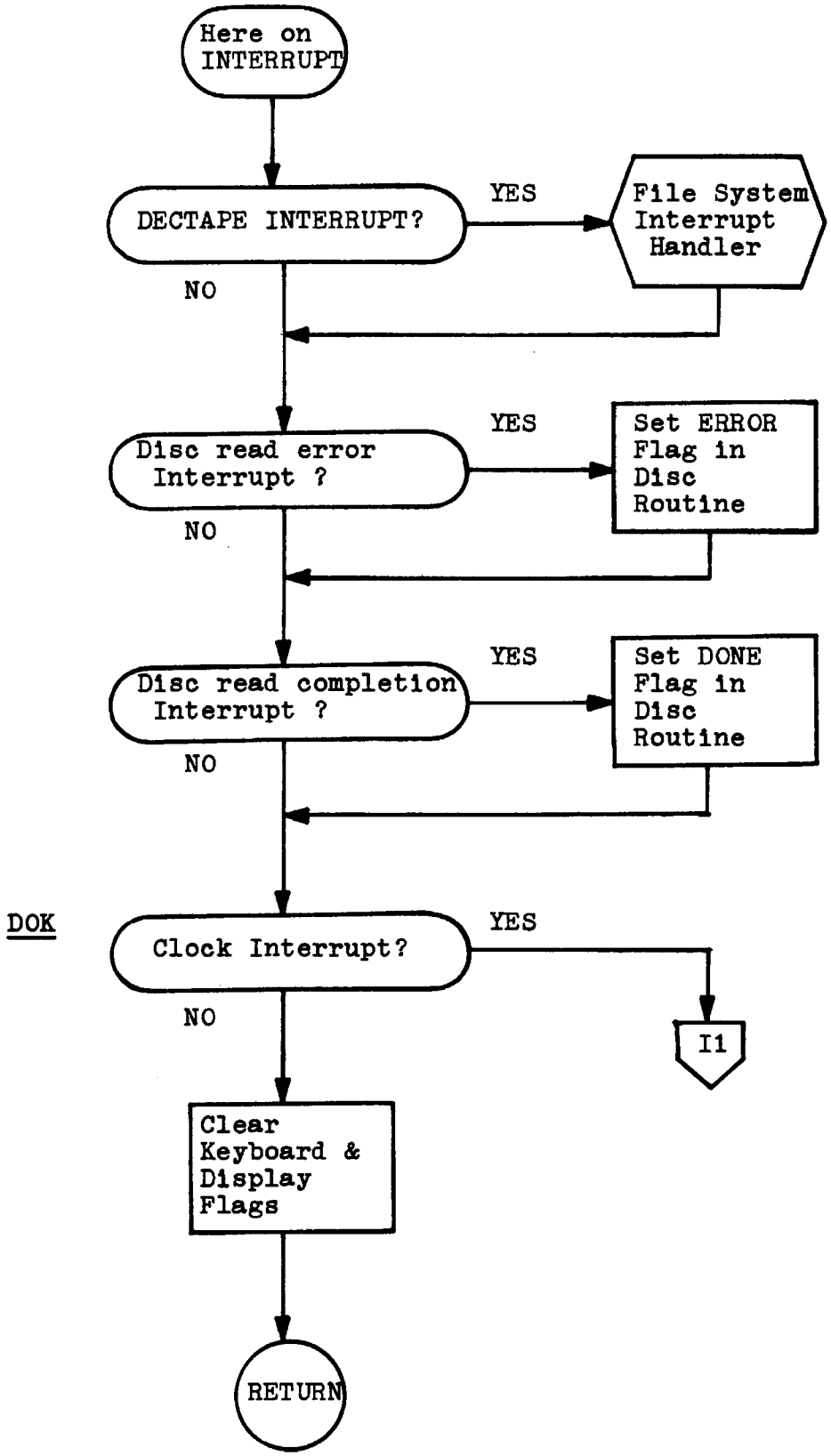


FIGURE F-2. con't

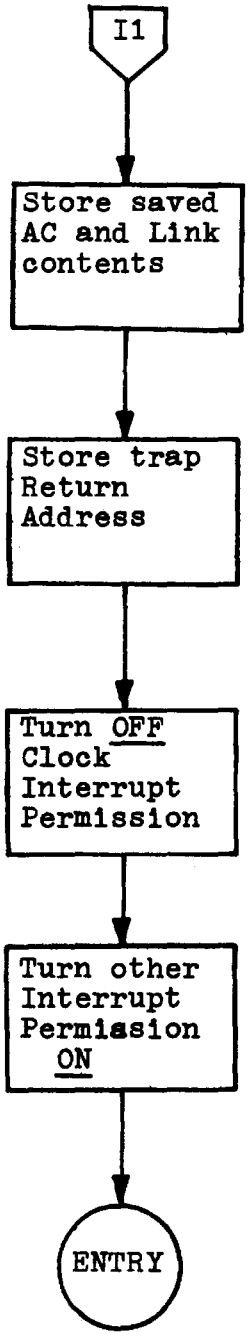


FIGURE F-3. ENTRY-Simulator Dataphone Routine Controller and Think Time Coordinator

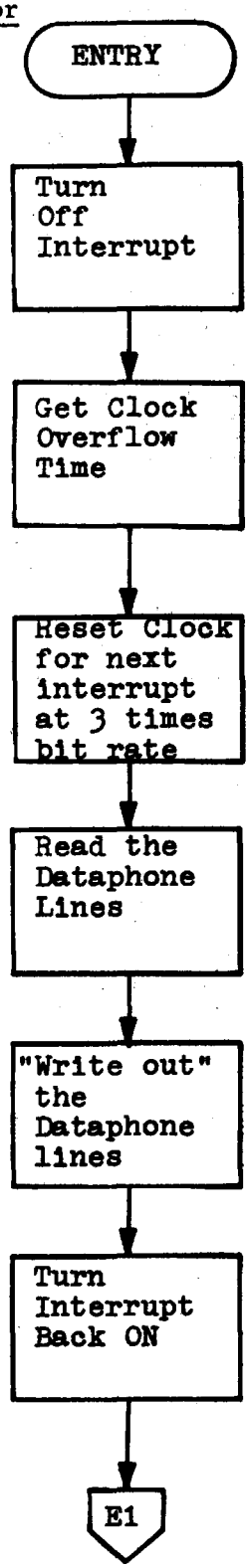


FIGURE F-3. con't

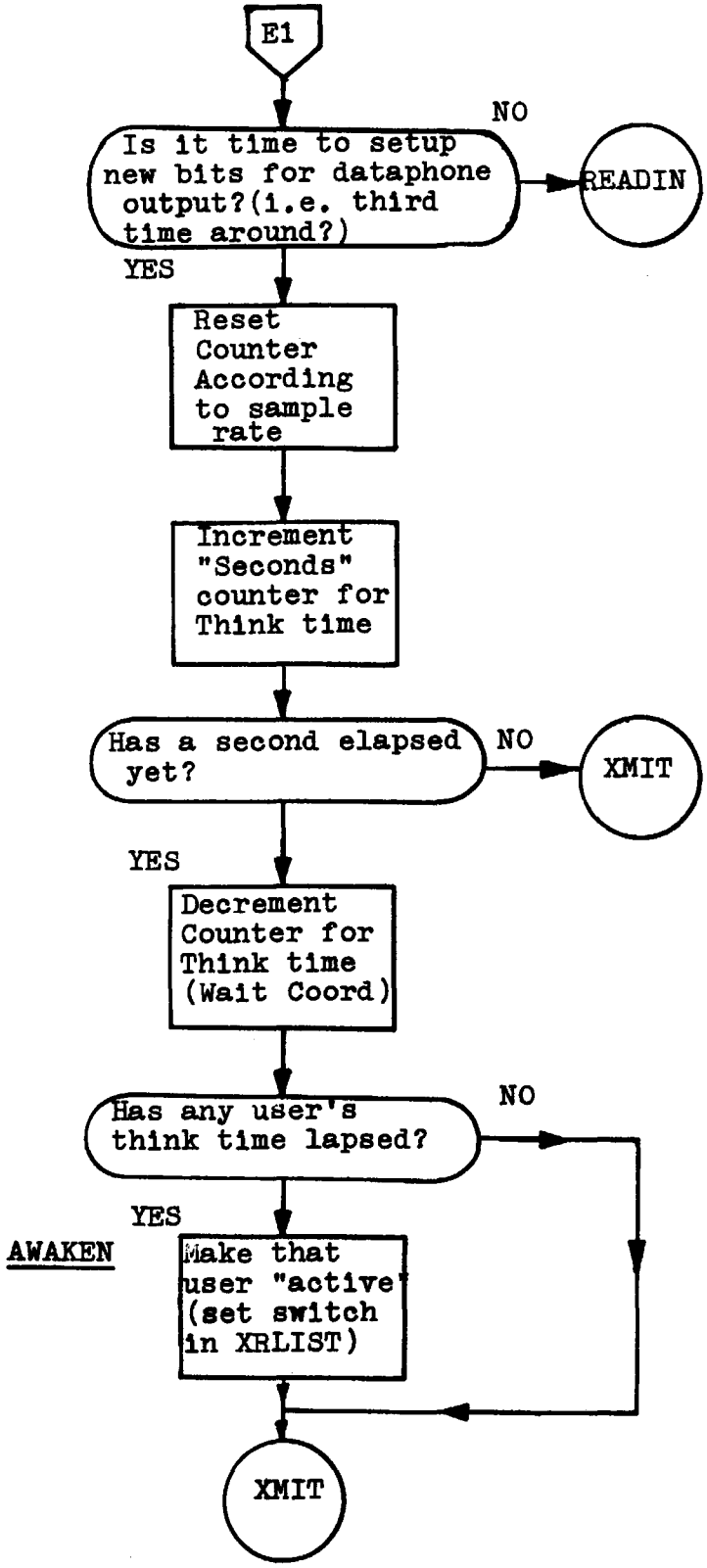


FIGURE F-4. XMIT-The Dataphone Lines Output Routine

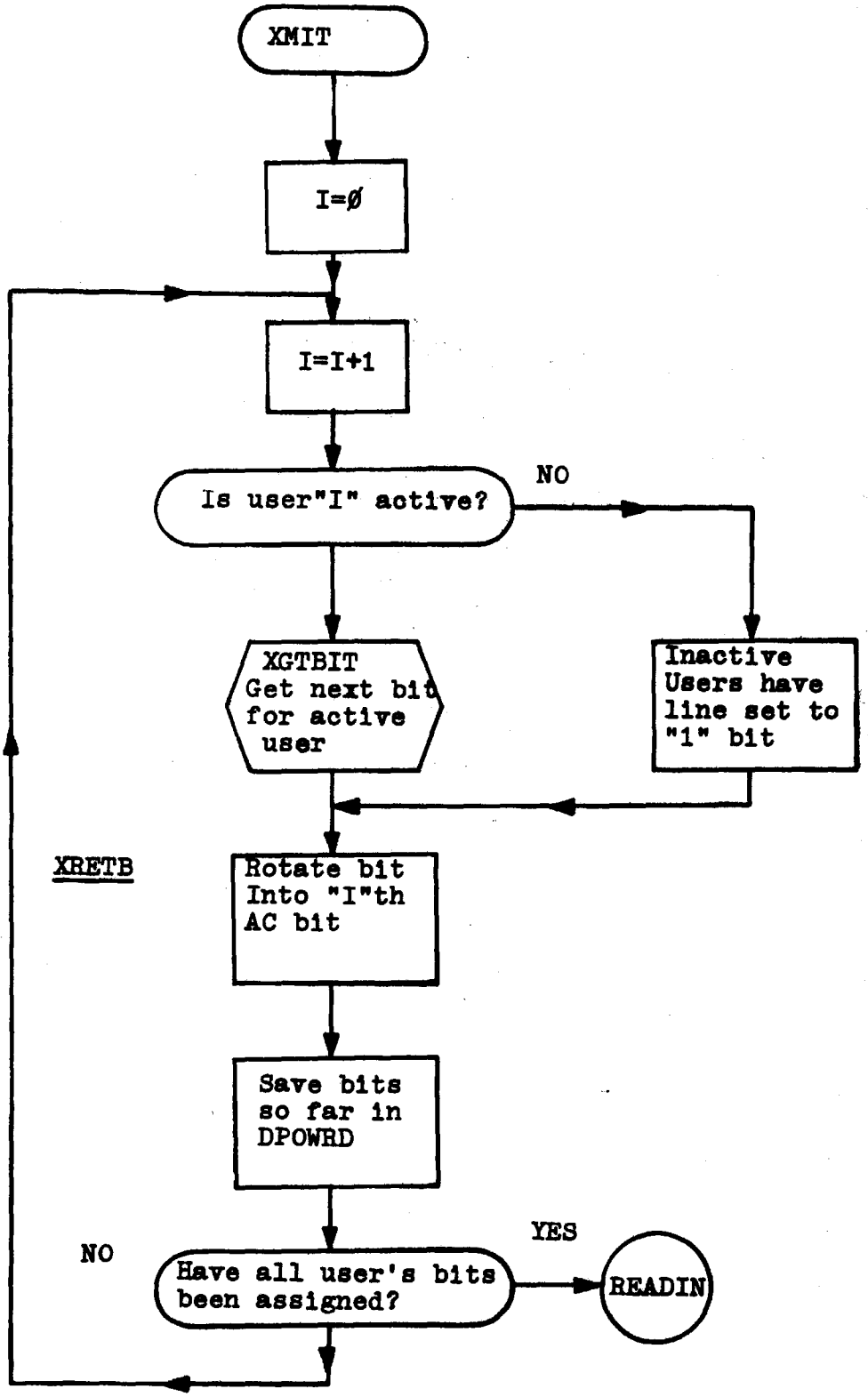


FIGURE F-4. con't

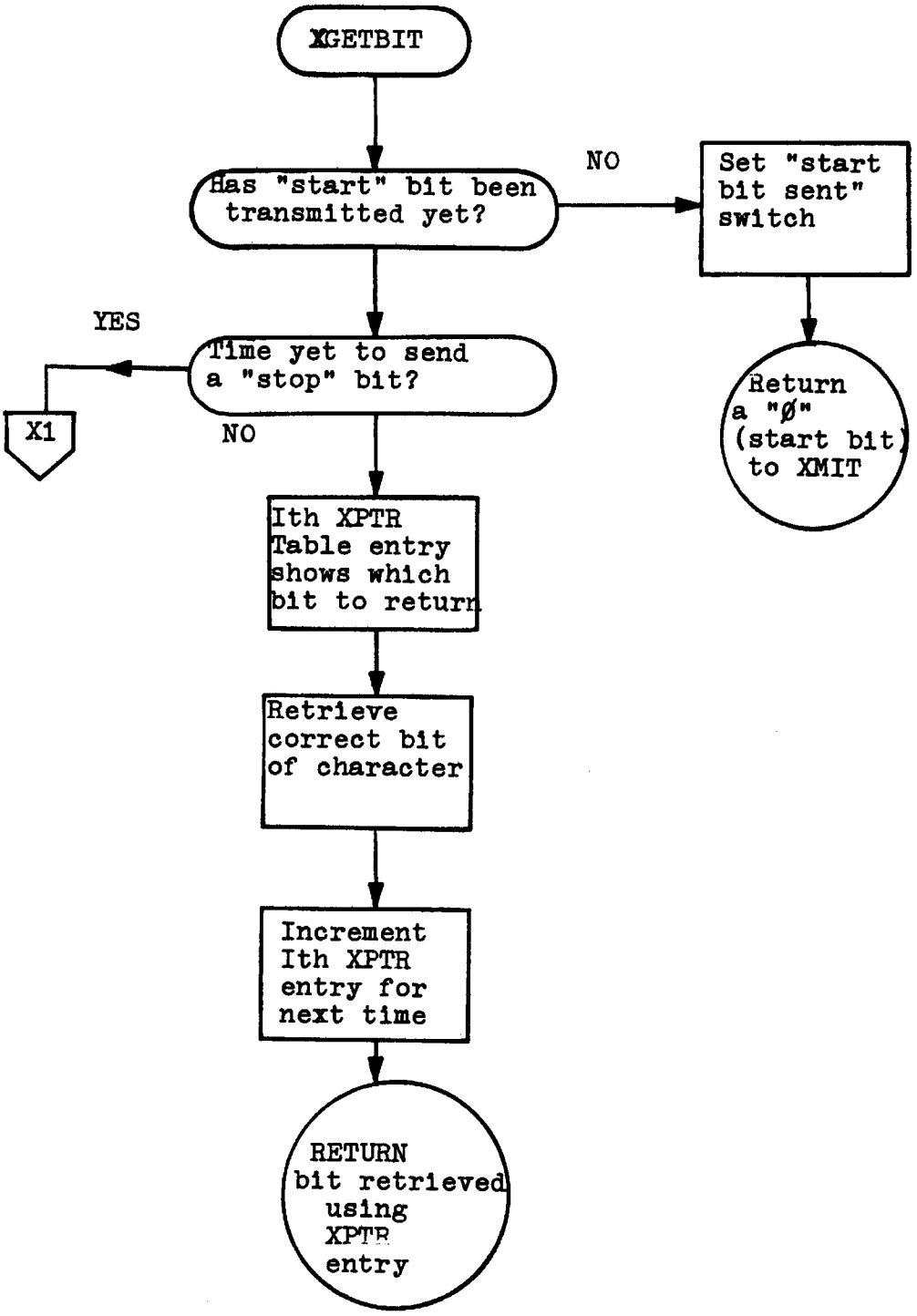


FIGURE F-4. con't

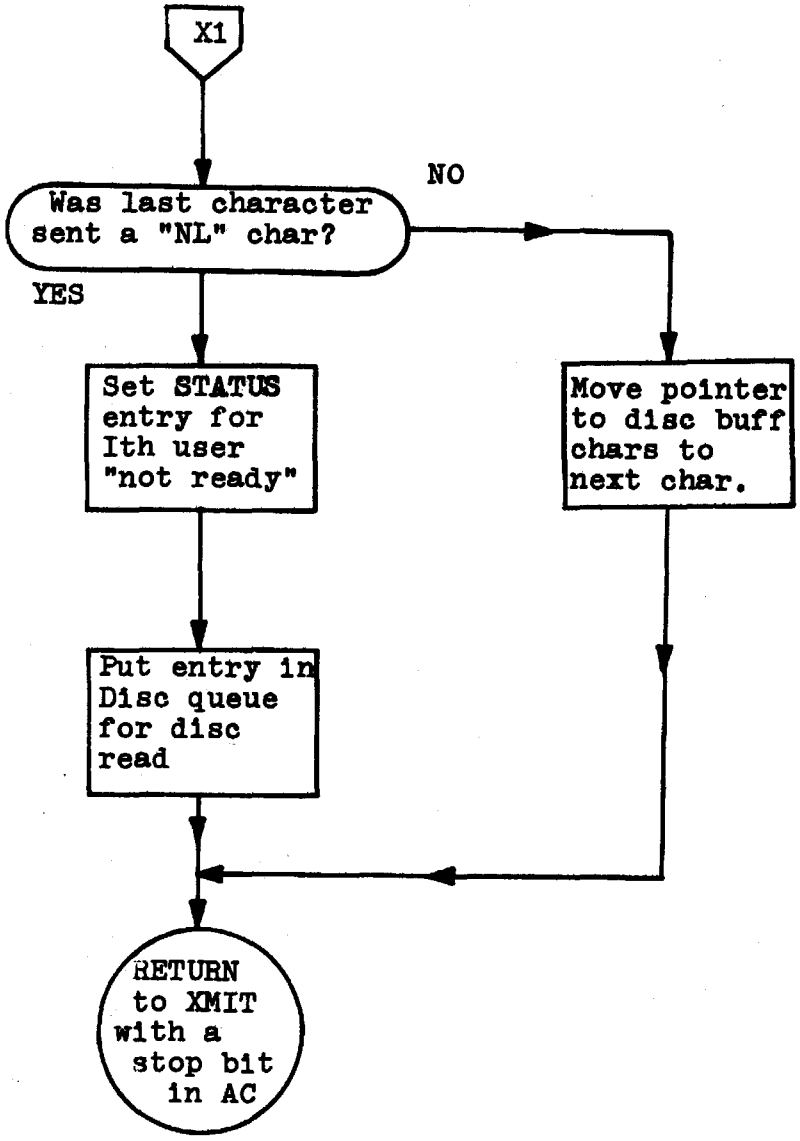


FIGURE F-5. READIN-The Dataphone Lines Input Routine

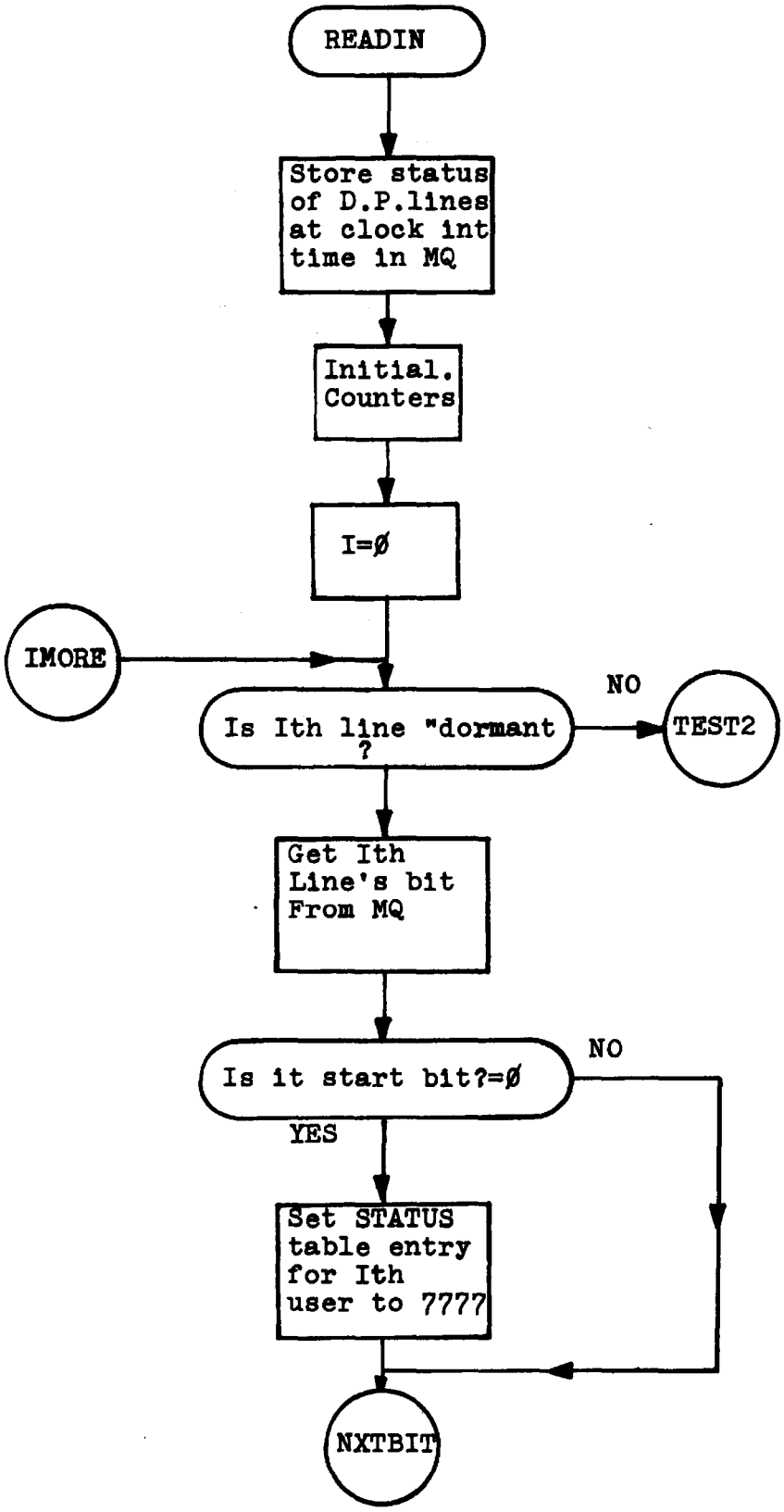


FIGURE F-5. con't

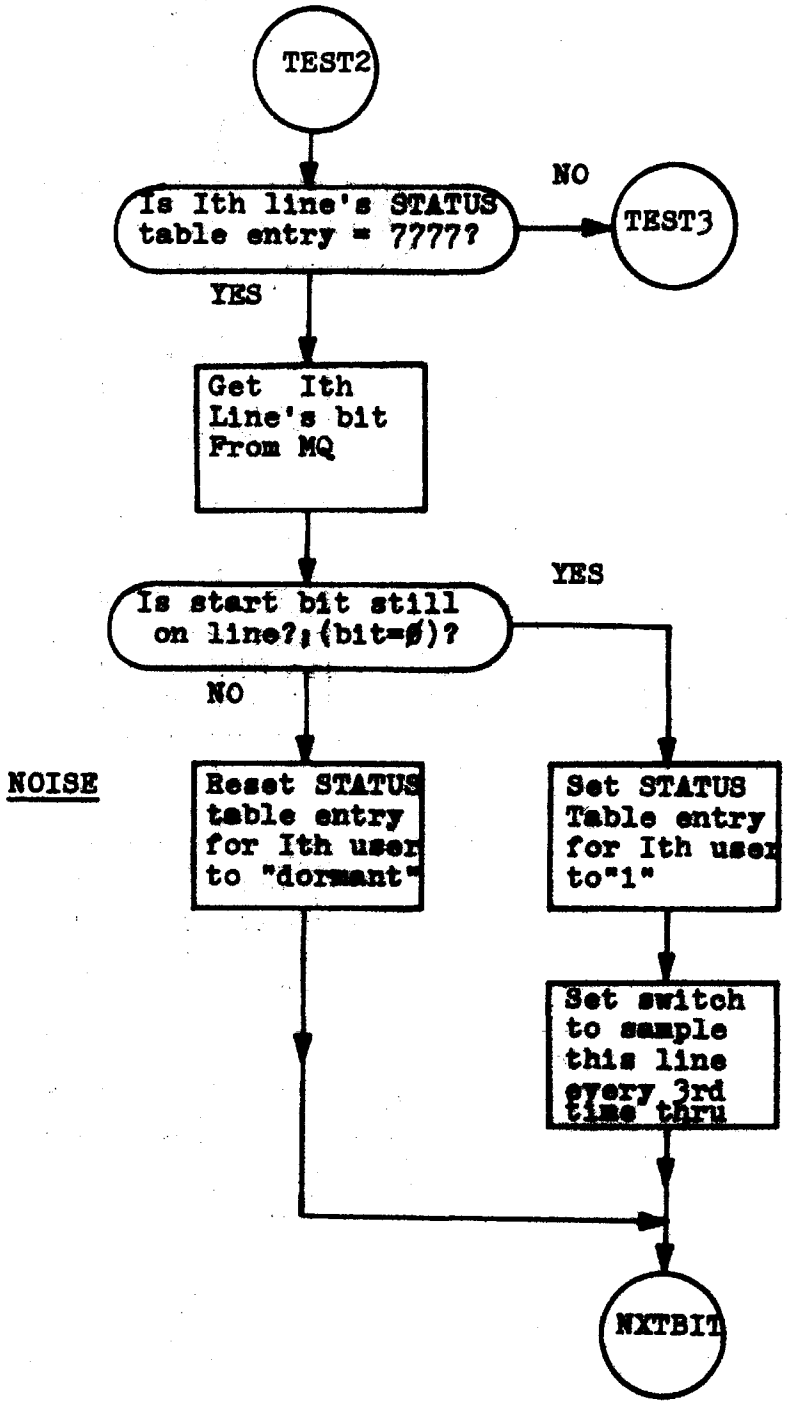


FIGURE F-5. con't

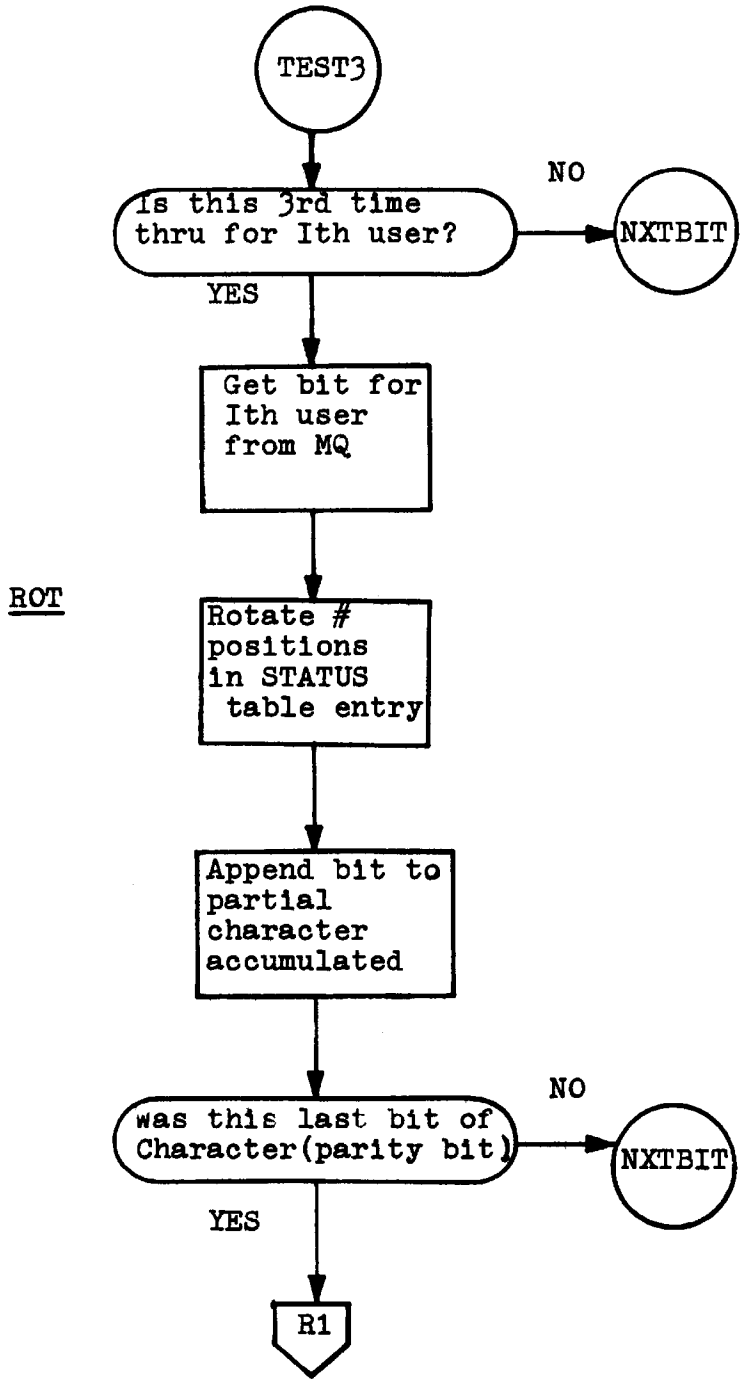


FIGURE F-5. con't

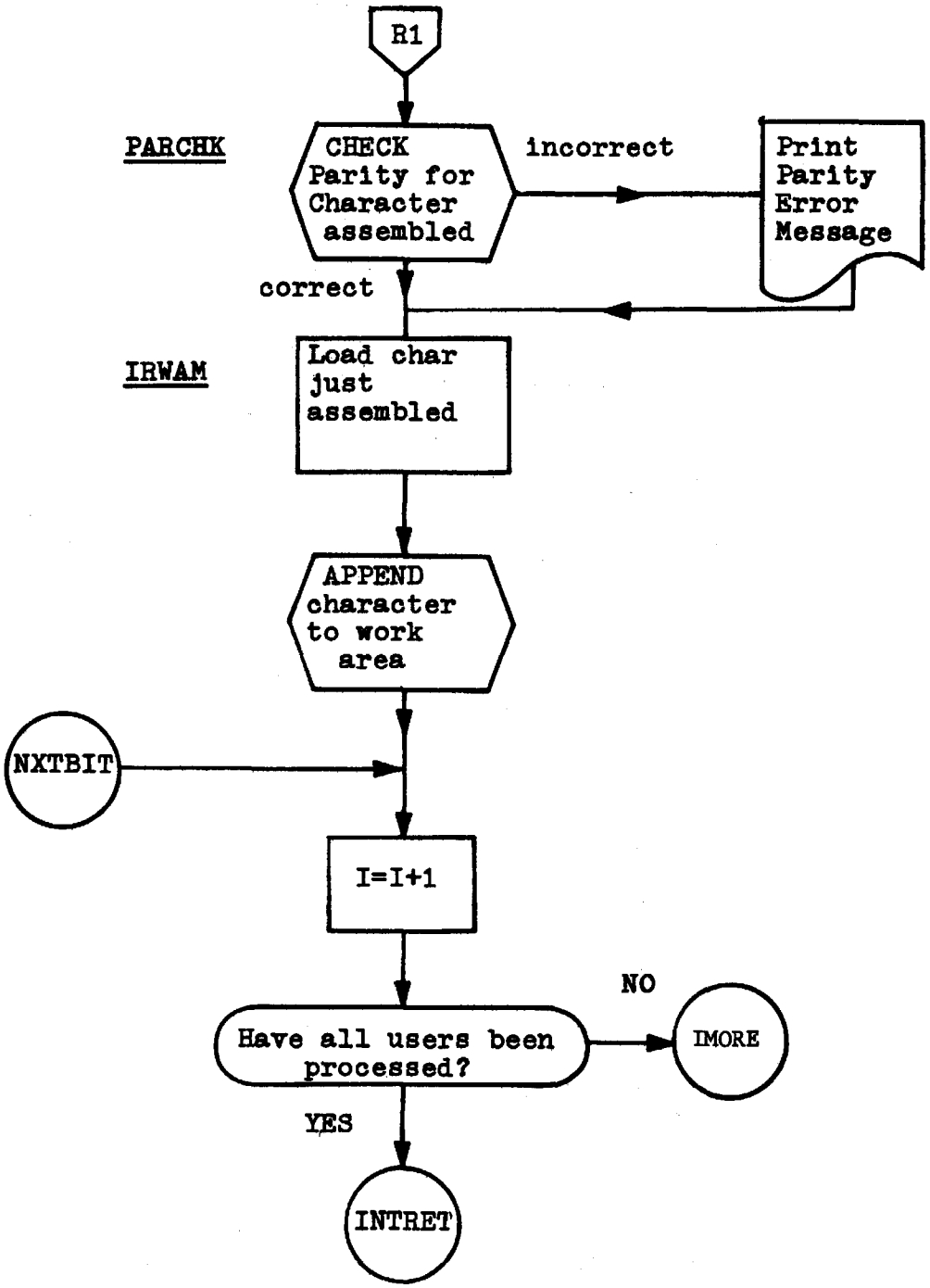


FIGURE F-6. APPEND-The Work Area Manager Routine

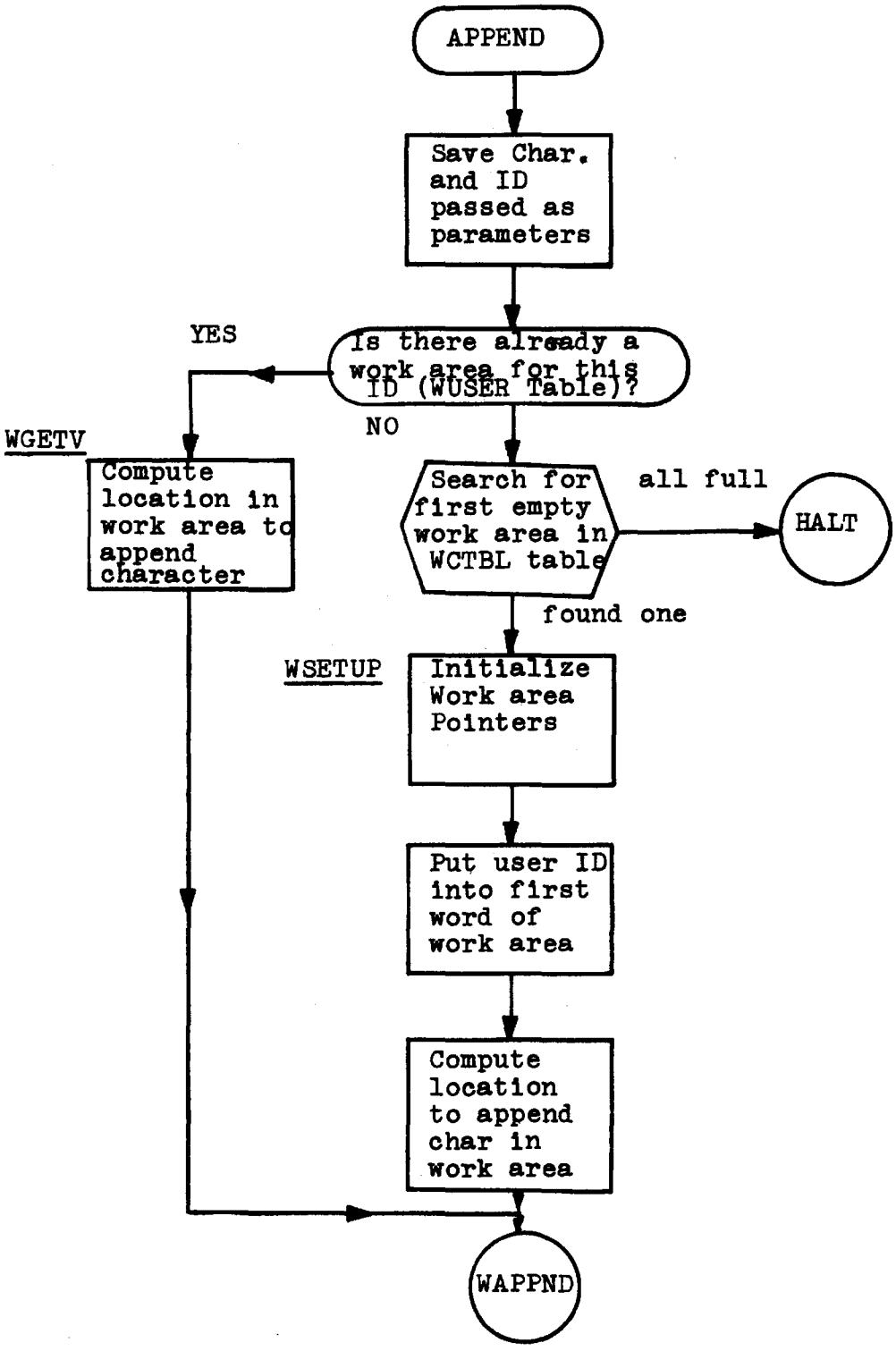


FIGURE F-6. con't

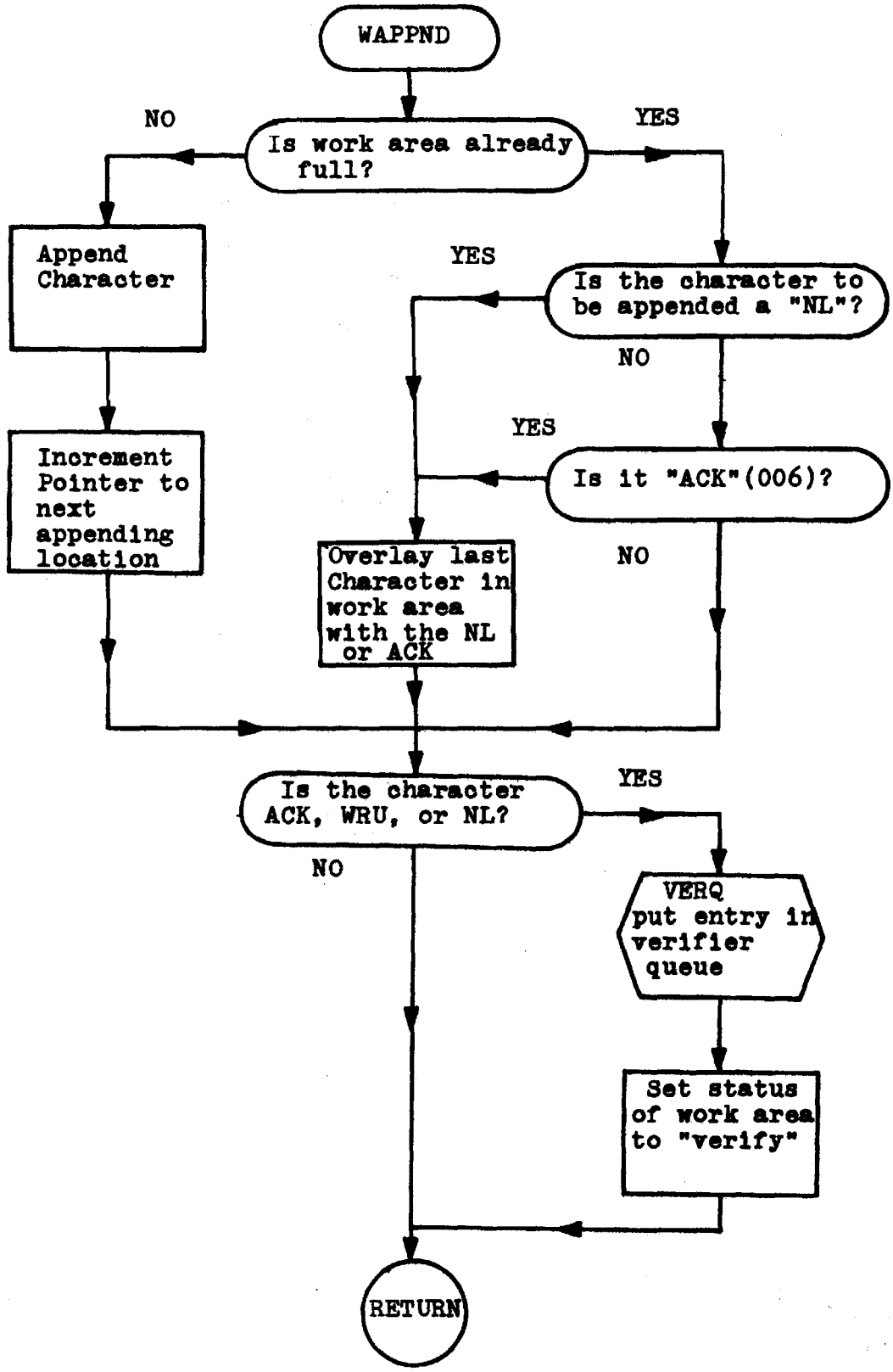


FIGURE F-7. INTRET-Clock Interrupt Return Routine

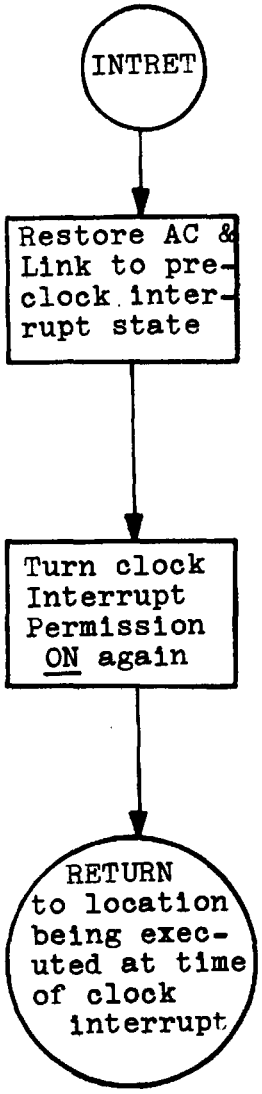
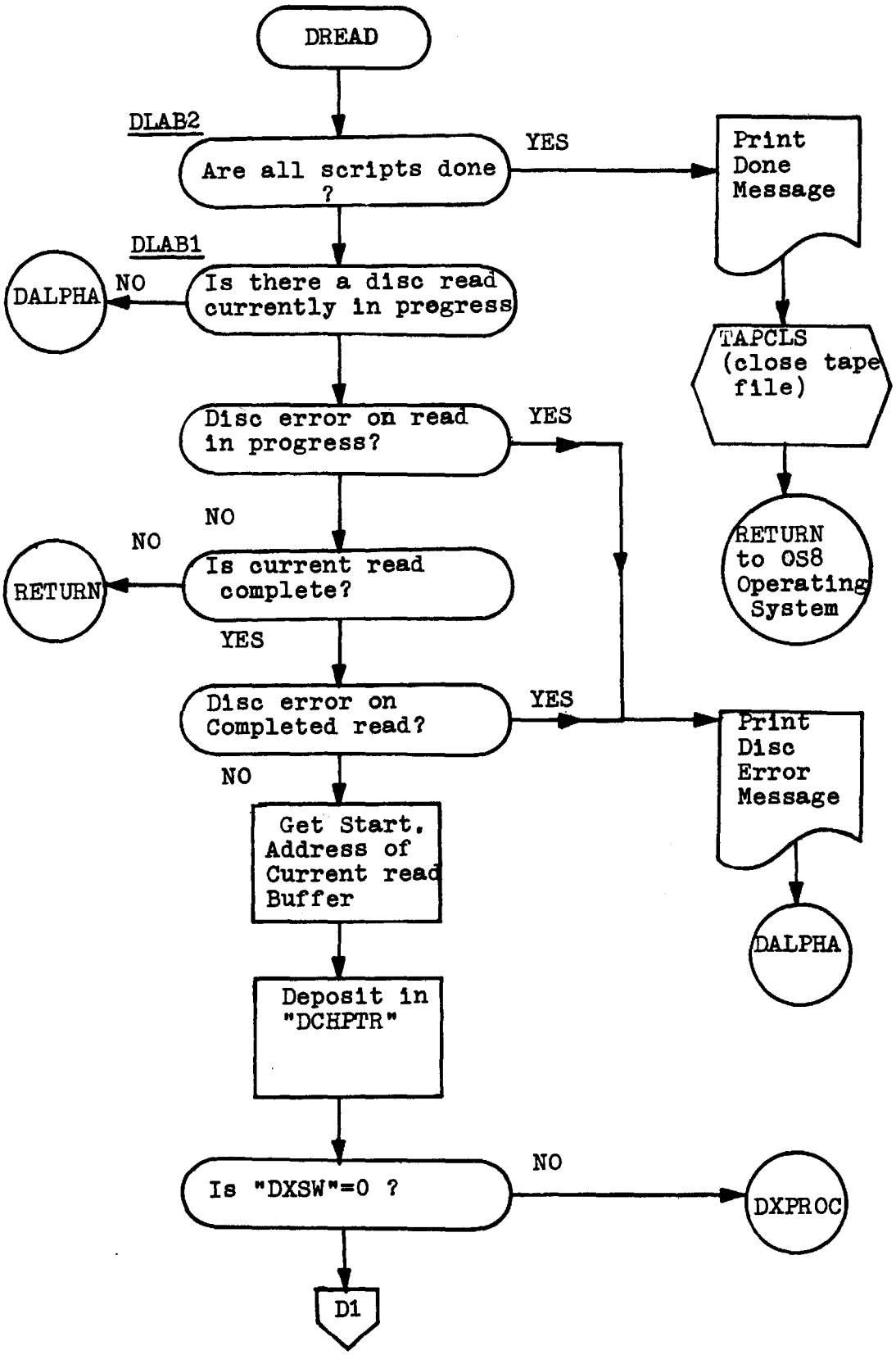


FIGURE F-8a. DREAD-Disc Read Routine



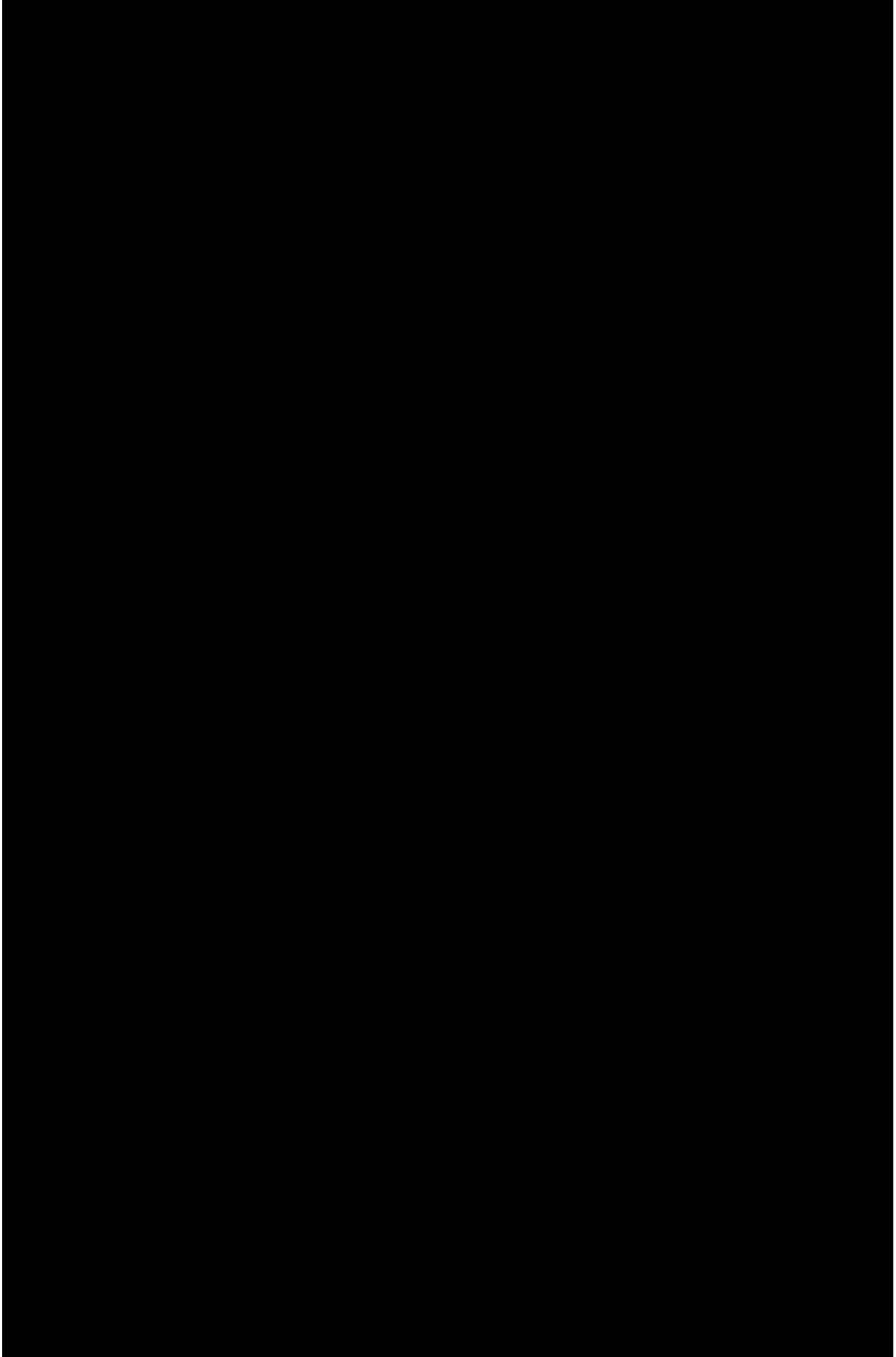


FIGURE F-8a. con't

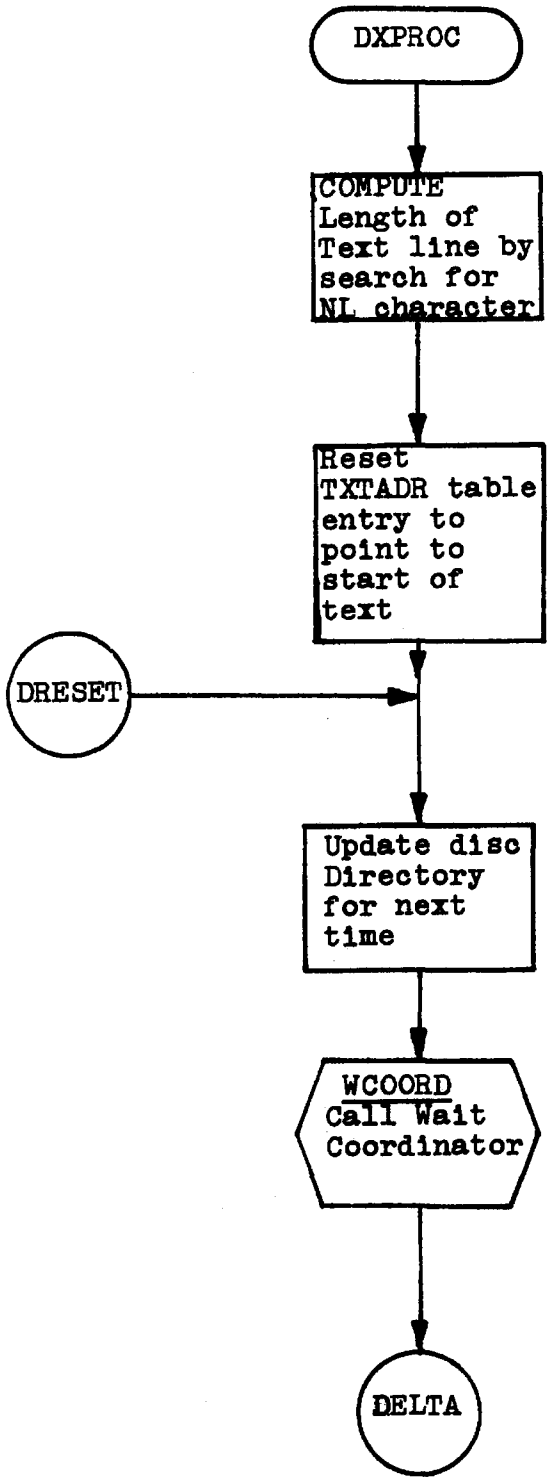


FIGURE F-8a. con't

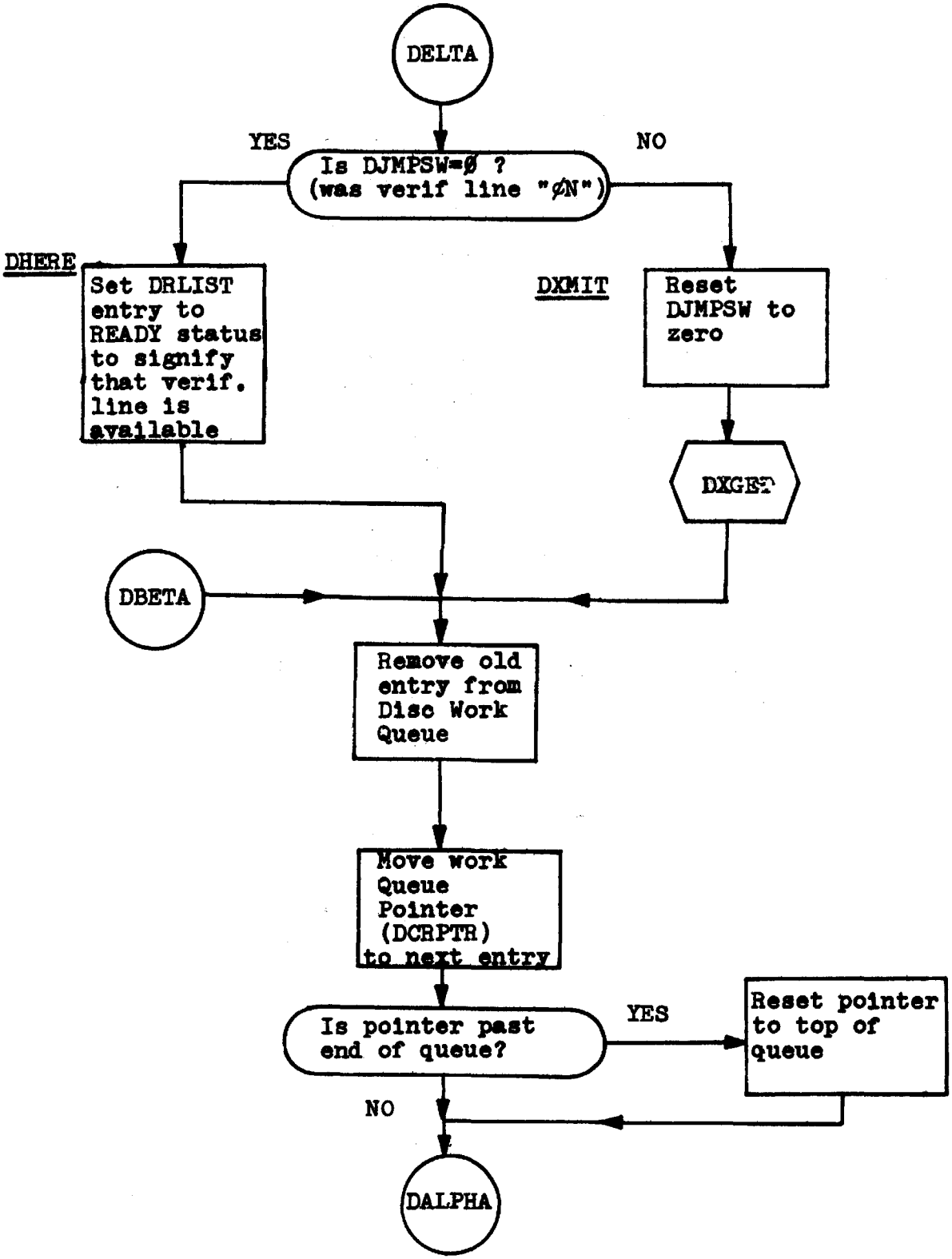


FIGURE F-8a. con't

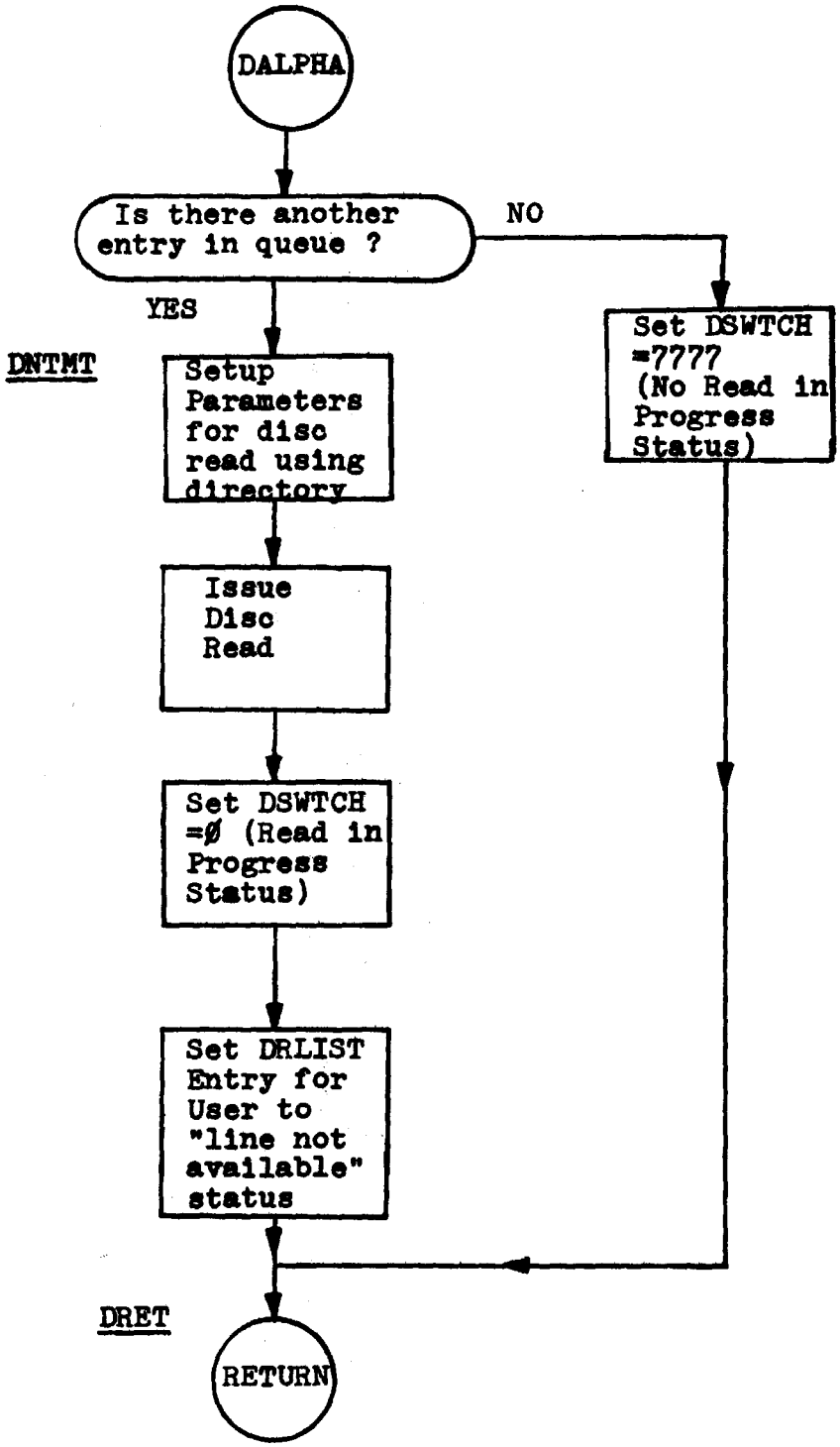


FIGURE F-8b. DQPUT-Put an entry in the Disc Work Queue for Next Verifier Line

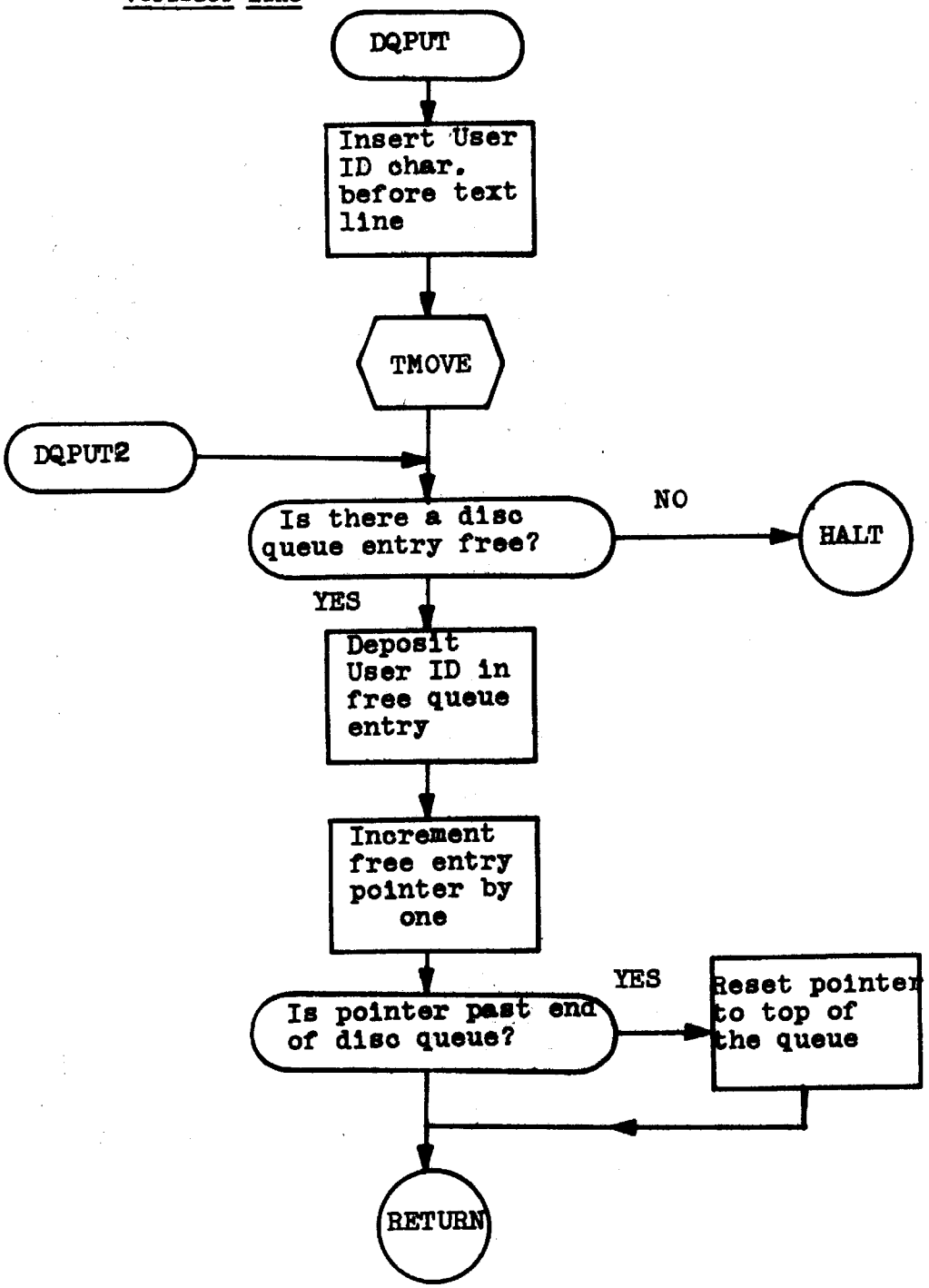


FIGURE F-8c. DTIME-Retrieve Think Time from Script Text Line

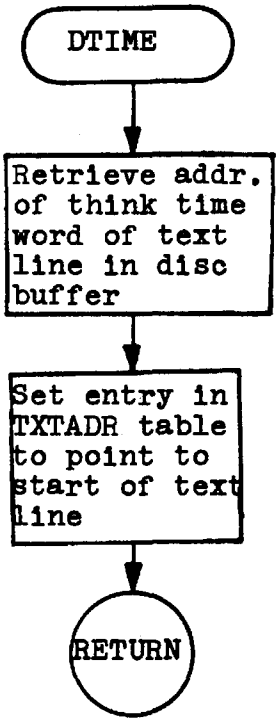


FIGURE F-8d. DXGET-Put an Entry in Disc Work Queue for Next Text Line

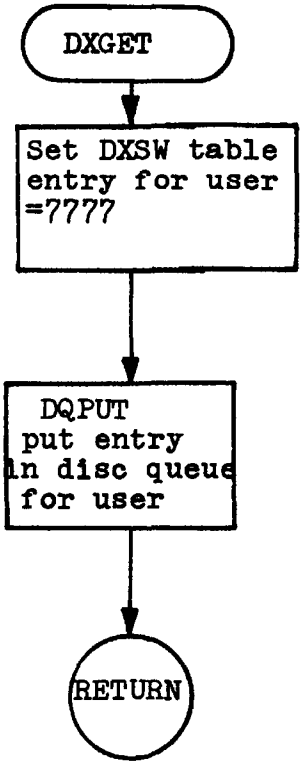


FIGURE F-8e. DSEARCH-Test if Verification Line for User is Available

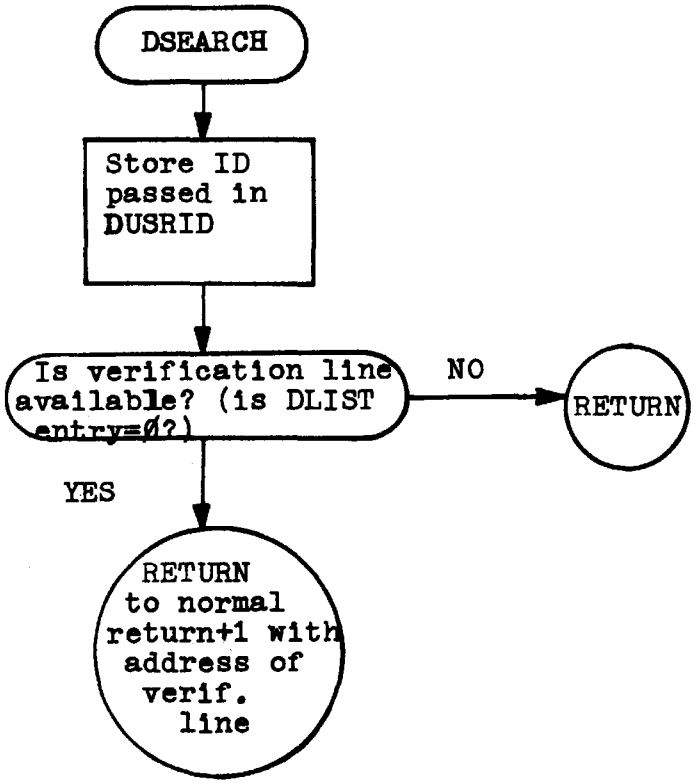


FIGURE F-9. VERIFY-The Verifier Routine

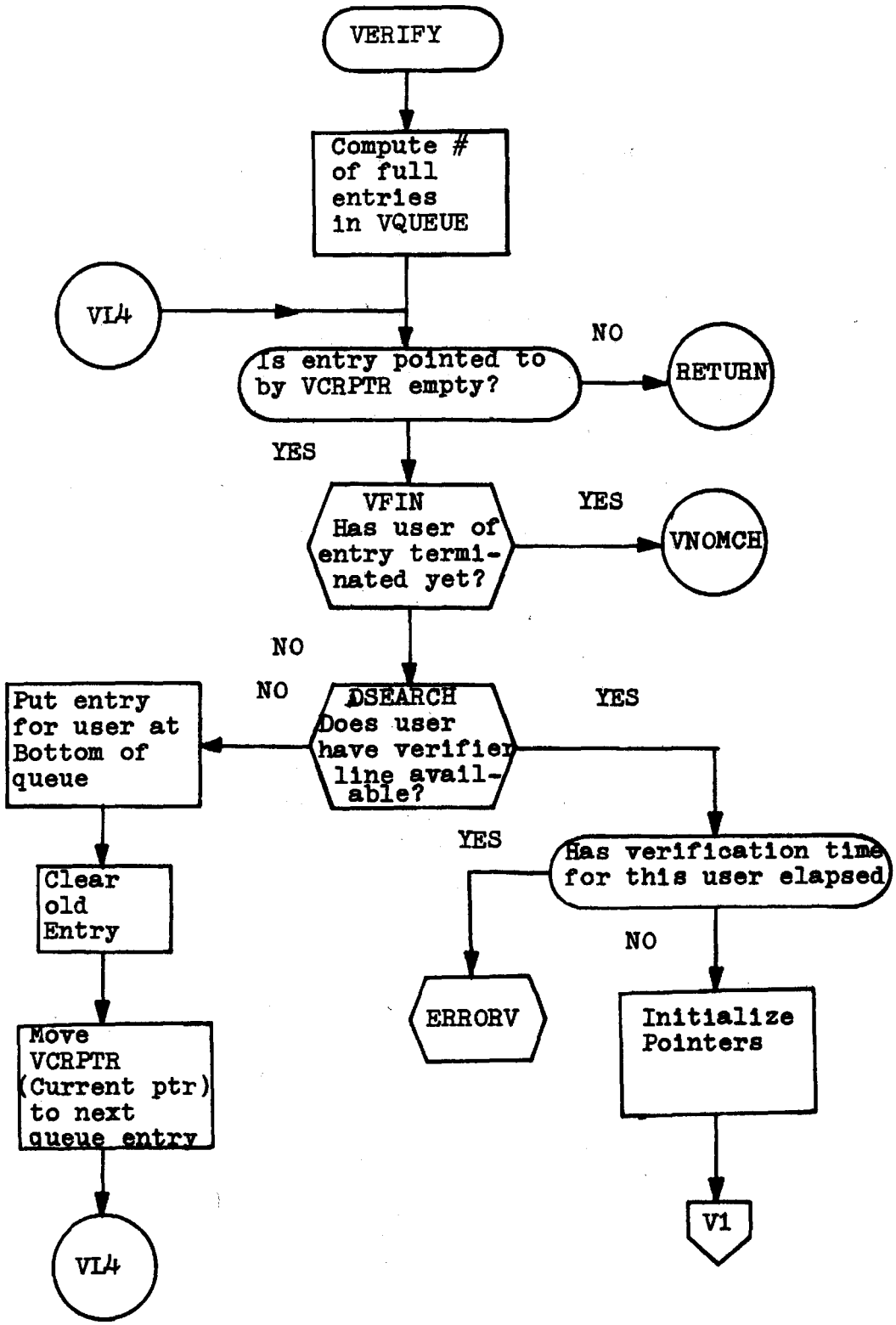


FIGURE F-9. con't

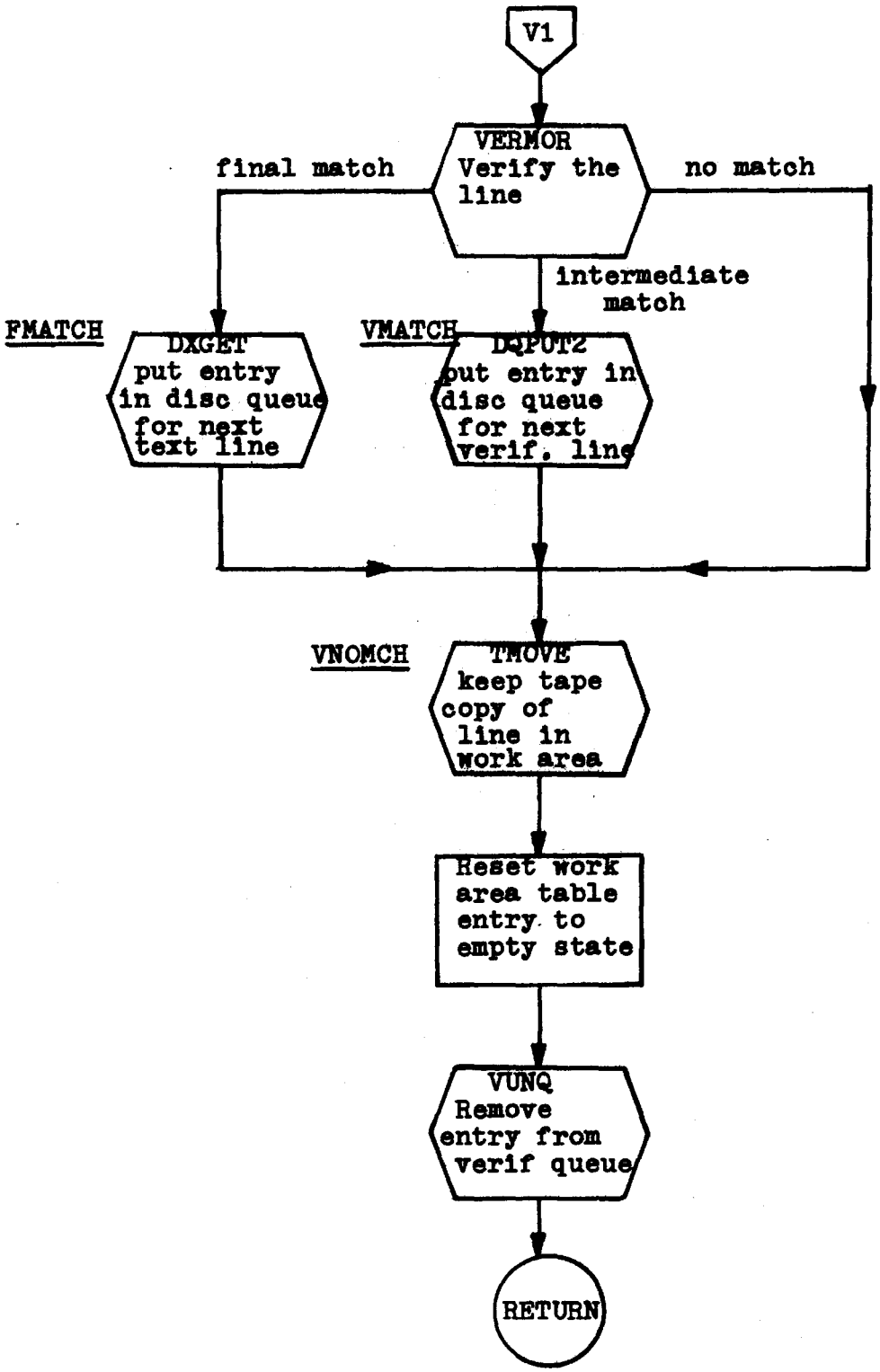


FIGURE F-9. con't

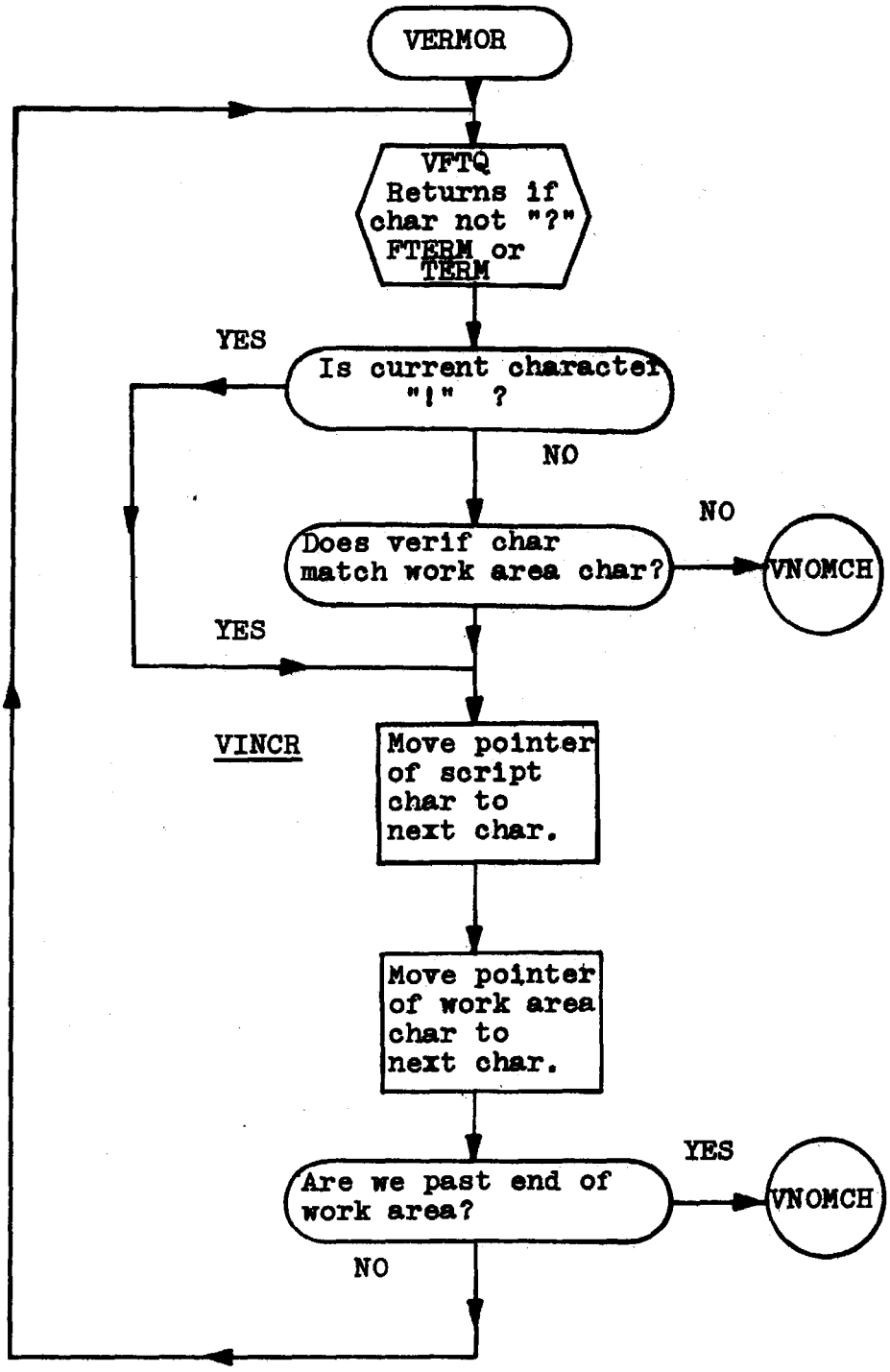


FIGURE F-9. con't

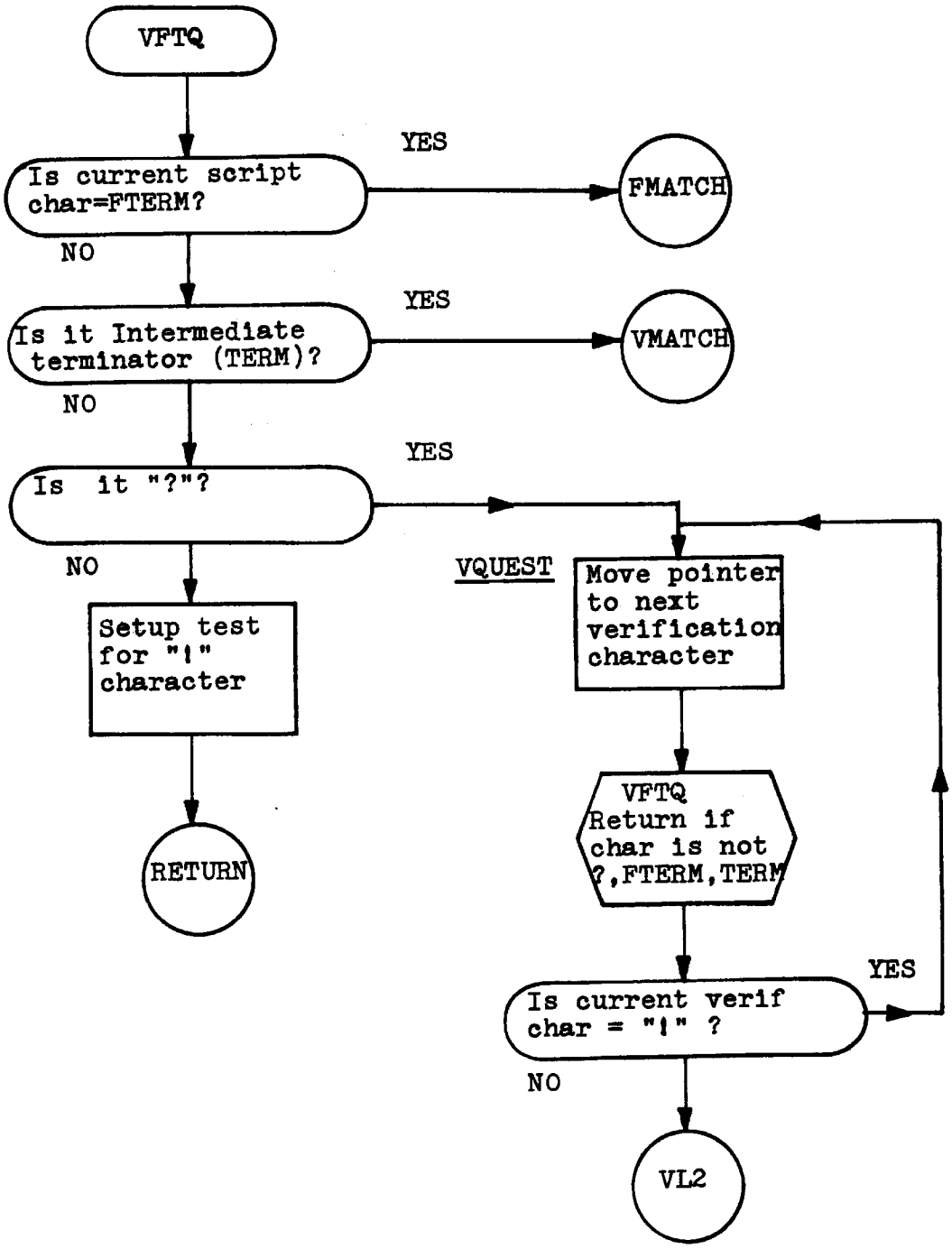


FIGURE F-9. con't

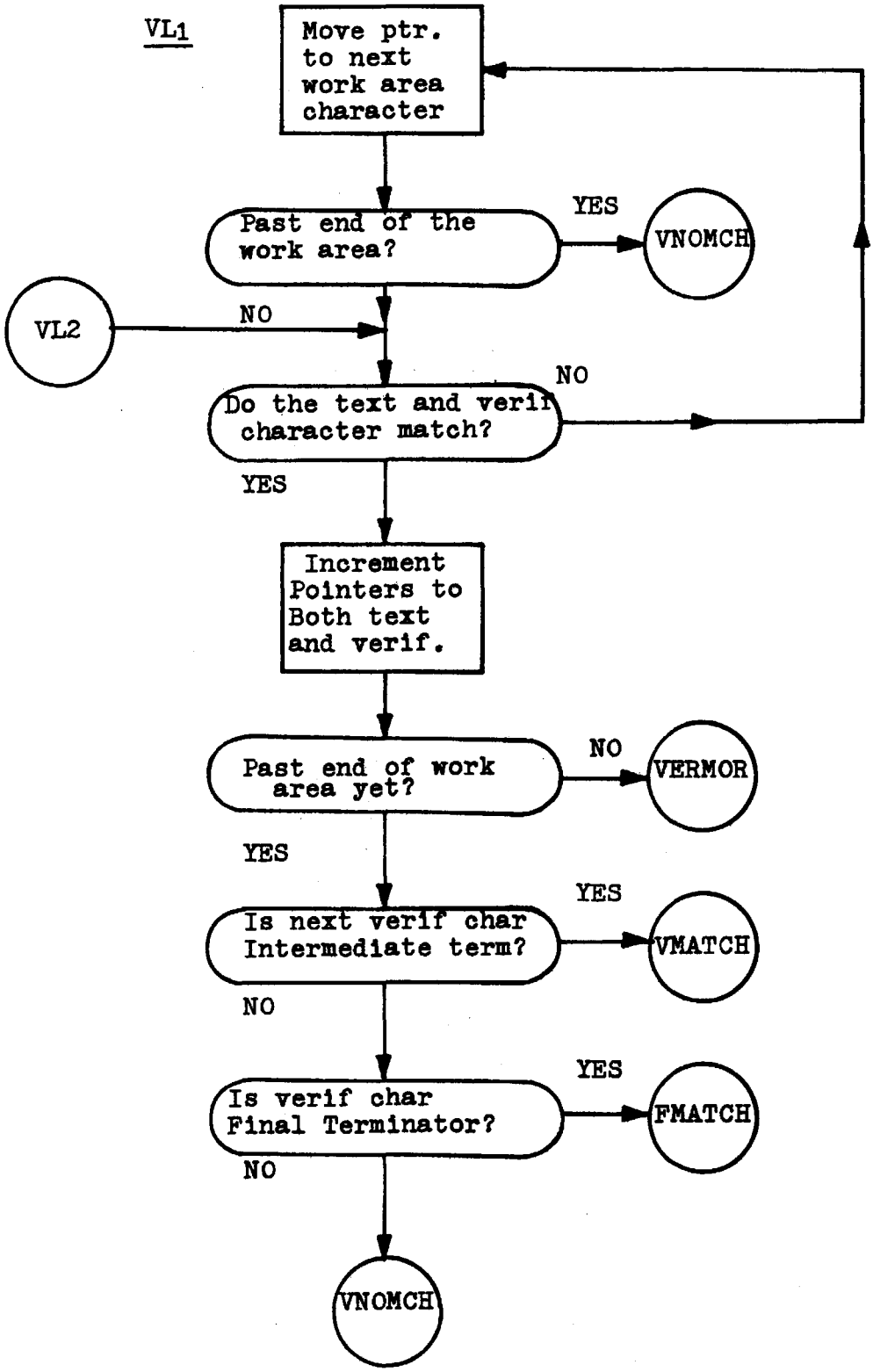


FIGURE F-10. TWRITE-The Tape Writing Routine

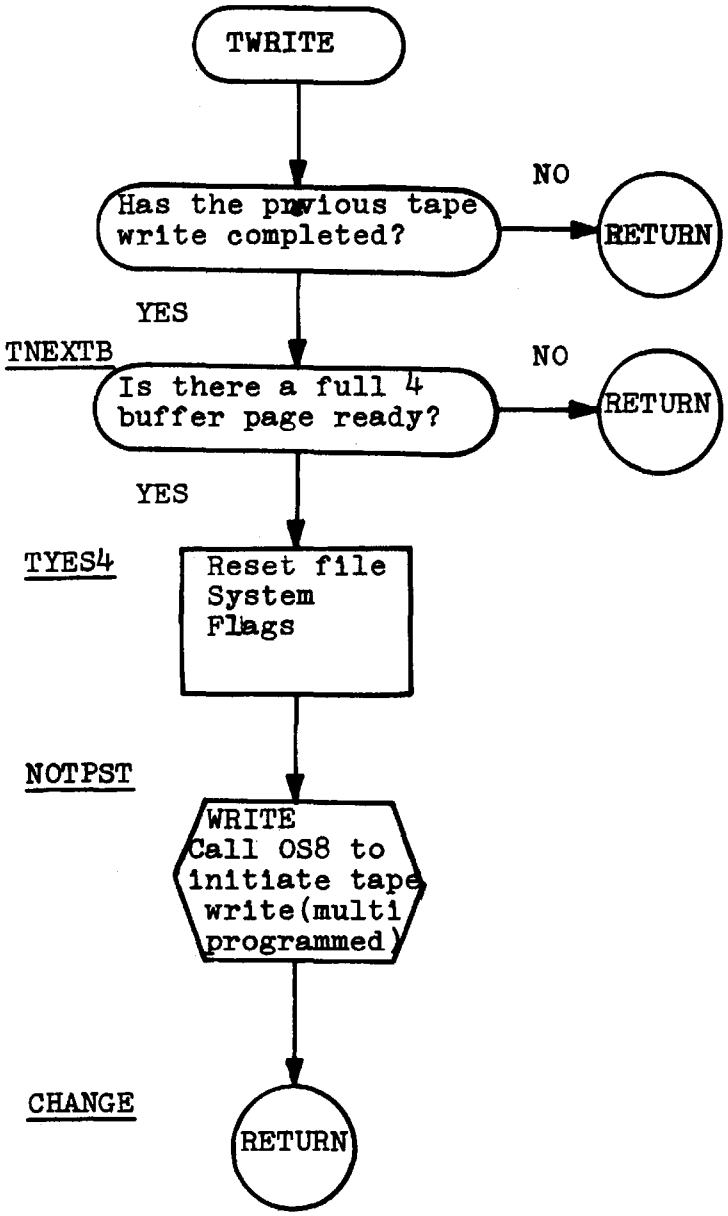


FIGURE F-11. TPMOVE-Routine to Transfer Text and Script Line to Copy Tape Buffer

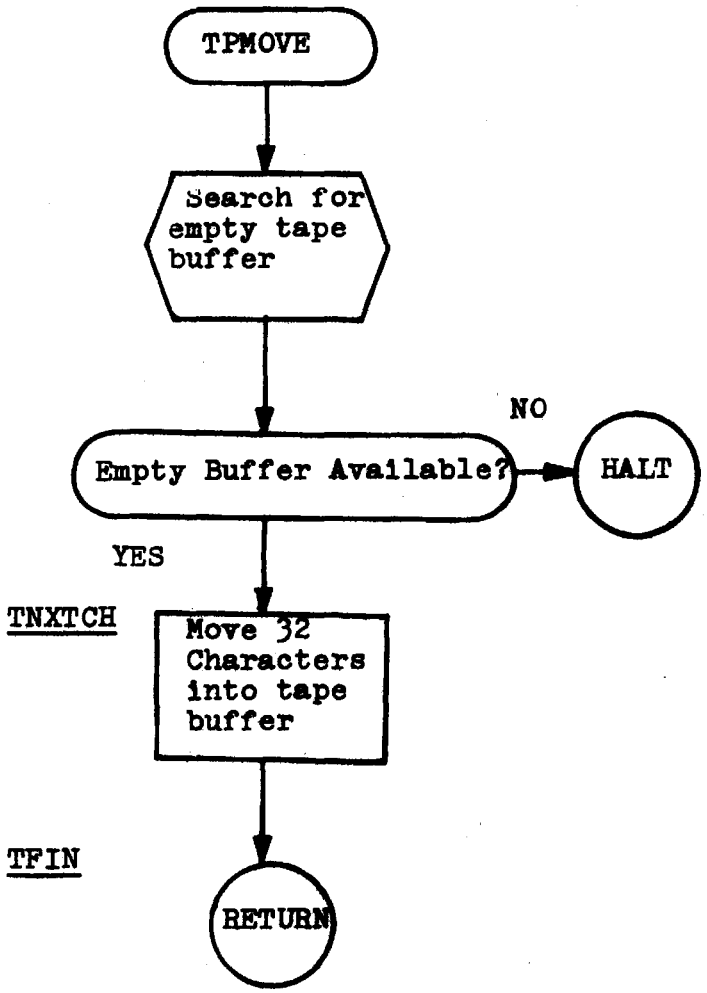
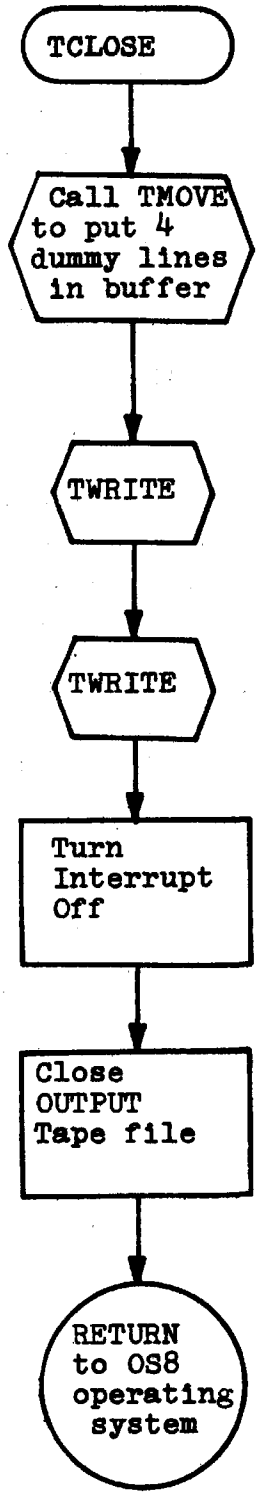


FIGURE F-12. TCLOSE-Tape Closing and Simulator Termination Routine



APPENDIX G - Script Loader Error Messages

The appendix provides a list of the Script Loader's error messages and their probable causes. The majority of the error messages issued by the Script Loader indicate a line number. In this appendix, this four digit line number will be represented by "****". It should be noted that in several instances an error on one line may cause error indications on subsequent lines, even though these subsequent lines may not have errors in them.

The following are the error messages issued by the Script Loader, and their probable causes:

1. "TAPE ERROR * TRY AGAIN"

This message is posted when a file cannot be opened by the file system. The "*" is replaced by the ASCII representation of the error code. For a complete list of error codes pertinent to this type of error see MAC Memo-191. In general this error message appears if the file name has been misspelled, or an incorrect drive number for the file has been specified. It is very rare that this error will be caused by a physical tape error. The user response is to retype the file name and drive number again, when the Script Loader requests them.

2. "NON-DIGIT ON LINE ****"

This error message is posted in cases when the Script Loader is expecting either a verifier time limit or think time line of a couplet. This type or error usually results from the omission of such a line when

composing a script with multiple verifier or text couplets. The user response is to correct script file in which the error occurred. This error also occurs when a letter is typed instead of a numeral in such a line.

3. "TOO MANY DIGITS ON LINE ****"

Both the verifier time limit and the think time lines allow a maximum of three digits to the line. If more than three appear on a line the above message is posted. The user response is to correct script file in which error occurred.

4. "OUTPUT TEXT LINE LENGTH EXCEEDED AT LINE ****"

Output text lines of text mode subsection couplets may only be of 120 characters in length. If this is exceeded the above message is posted, and subsequent characters are ignored. The user response is to shorten the output text line.

5. "ILLEGAL SEQUENCE OF MODES. 'V' MISSING AT LINE ****"

As described in Chapter III, the script must consist of altemating verifier and text mode subsections, with the exception that comment mode subsections may be interspersed between the above subsections. The above error message is posted when the Script Loader encounters two text mode subsections in a row. The user's response to this indication is to either consolidate the two text mode subsections, or insert a proper verifier mode subsection inbetween. The error could also have been caused if a character other than a "V" were found after the ":".

6. "IMPROPER VERIFICATION GROUP FORMAT AT LINE ****"

This error message is posted when the string ":VNL" is incorrect. In general it is caused by the absence of the "NL" character, due to putting something additional on this line. The user should respond by correcting the line by placing a NL character immediately after the ":V" string.

7. "FOUND N/L:T---TERMINATING N/L MISSING AT LINE ****"

This message is posted for the similar reason to #6 above. A ":T" was found, but its terminating NL character is missing. User response as in #6.

8. "IMPROPER TEXT LINE FORMAT AT LINE ****"

This error message is posted, as is the error message in part #5 of this appendix. This time, however, the Script Loader has come upon a character after a ":" where a "T" should have been. The user response is to correct letter after the ":" or otherwise revise his script.

9. "DISC OVERFLOW ERROR. SCRIPT TOO LONG
LAST FEW CHARACTERS WERE:"

This error message is posted when the disc capacity has been exceeded. The user response is to either shorten his scripts, or lessen the number of scripts and hence users to be simulated. After this message is posted, the last fifty (50) characters in the disc buffer are printed out, to determine where in the script the overflow occurred.

10. "DISC HARDWARE ERROR ********"

This error message is posted if a disc hardware error occurs. The PDP-8 magnetic disc is an exceptionally reliable device, and it is expected that this type of error will rarely occur. If this message is posted, disc diagnostics should be run immediately. However, this message will also be posted for every character processed after error message #9 above, so in this case no special action should be taken.

BIBLIOGRAPHY

1. CORBATO, F. J. et al., The Compatible Time-Sharing System: A Programmer's Guide, 1st ed., M.I.T. Press, Cambridge, Mass. 1963.

2. CORBATO, F. J. and V. A. VYSSOTSKY, Introduction and Overview of the Multics System, AFIPS Conference Proceedings, Vol. 27, Part 1, Spartan Books, Washington, D. C., F. J. C. C., 1965.

3. SCHEER, A. L. An Analysis of Time-Shared Computer Systems, Project MAC, M.I.T., Technical Report, MAC-TR-18, June 1965.

4. HASTINGS, T., Operating Statistics of the MAC Time Sharing System Project MAC, M.I.T., Memorandum MAC-M-280, December, 1965.

5. LUCONI, F. L., Real-Time Braille Translation System, Massachusetts Institute of Technology, Master's Thesis, May, 1965.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Master's Thesis, Department of Electrical Engineering, October 1968

5. AUTHOR(S) *(Last name, first name, initial)*
Greenbaum, Howard J.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
January 1969	192	5

8a. CONTRACT OR GRANT NO.
Office of Naval Research, Nonr-4102(01)
b. PROJECT NO.
NR 048-189
c.
RR 003-09-01
d.

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC-TR-58

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency
3D-200 Pentagon
Washington, D.C. 20301

13. ABSTRACT
Constructing and maintaining a Time-Shared Computer System requires a controlled, repeatable environment for making performance measurements. This thesis describes the use of a small second computer to simulate the actions of multiple interactive users over individual communication lines. Each simulated user exhibits responses similar to those of a "normal" interactive user; these are recognized and verified by the "Simulator". The Simulator also emulates a "think time" corresponding to a normal user's think time between typing lines on the console. Text corresponding to a user's console input, as well as control information regarding think time simulation and verification of responses from the system being tested, are retrieved from prepared scripts which have been pre-stored on the small computer's magnetic disc unit.

Although the programming package is capable of simulating up to 12 users, only four are simulated here. The Simulator System is intended to be used to test the M.I.T. CTSS and Multics time-shared computer systems. However, it is designed to be adaptable for testing most time-shared computer systems having serial character oriented input/output over communications lines interfacing with 103A compatible data sets.

14. KEY WORDS

Computers	Multiple interactive users	Simulators
Machine-aided cognition	On-line computer systems	Time-shared computers
Multiple-access computers	Real-time computers	Time-sharing

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification