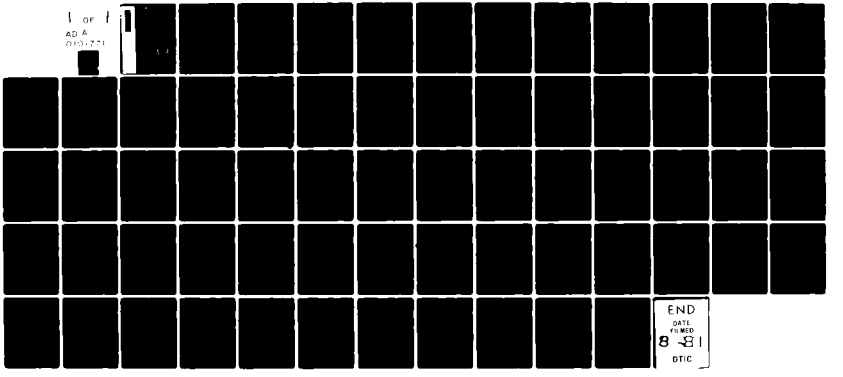
AD-A101 771 MISSION RESEARCH CORP ALEXANDRIA VA F/G 9/2
FREFORM: AN ALGORITHM FOR PROCESSING FREE-FORMAT DATA,(U)
FEB 81 J GREENFIELD, B GOPLEN N00173-80-C-0059
UNCLASSIFIED MRC/WDC-R-011 NL



END
DATE FILMED
8-81
DTIC

LEVEL II

(2)

MRC/WDC-R-011

FREFORM: AN ALGORITHM FOR PROCESSING FREE-FORMAT DATA

Jack Greenfield
Bruce Goplen

February 1981

Prepared for: Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, D.C. 20375

Contract No: N00173-80-C-0059

MISSION RESEARCH CORPORATION
5503 Cherokee Avenue, Suite 201
Alexandria, Virginia 22312

Copy No. 10

MRC/WDC-R-011

(6)

FREFORM: AN ALGORITHM FOR PROCESSING FREE-FORMAT DATA

(1) Jack Greenfield
Bruce Goplen

(9) February 1981

(12) 65

Prepared for: Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, D.C. 20375

(15) Contract No: N00173-80-C-0059

MISSION RESEARCH CORPORATION
5503 Cherokee Avenue, Suite 201
Alexandria, Virginia 22312

DTIC 345709

ABSTRACT

A library subroutine named FREFORM has been developed for use on the PDP-11/10 computer. This subroutine can be accessed by user programs to process data stored on disk or data entered at the computer terminal. The use of FREFORM eliminates the need for format statements in programs which process or use such data. More importantly, restrictions on the data format and syntax are greatly reduced.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1
INTRODUCTION

Subroutine FREFORM can process data which are relatively unrestricted in format and syntax. Thus, it can be used in place of the usual formatted I/O statements. The objective is to simplify user programs while at the same time enhancing the efficiency of terminal use. FREFORM differs from conventional processing algorithms in that it is highly flexible. (NAMELIST, for example, is intended primarily for batch mode, and requires user knowledge of variable names.) FREFORM is designed specifically for users of NRL's Data Acquisition System.

The usefulness of FREFORM can be illustrated with a simple example. Consider the input data sequence,

^SHOT^4395^SIO2^^1.000E+03^^4.000E-02^^3.142E+00^^235,

entered by a user at a terminal console. (The ^ symbol will be used to explicitly represent a blank, or space.) The conventional program will contain format statements contructed specifically to read such data strings. To read the data sequence above, the user's program might contain the following statements:

```
     READ(5,100)ALFA,NSHOT,BETA,(COEF(I),I=1,3),I519
100  FORMAT(1X,A4,1X,I4,1X,A4,3(2X,E10.3),I5)
```

Execution of this read statement will cause an ASCII representation of the word "SHOT" to be stored in the central memory location referenced by the word, ALFA. Similarly the integer value, 4395, will be stored in the word referenced by NSHOT, et cetera. Of course, if the data entry is made incorrectly (e.g., the wrong spacing, use of F- rather than E-format, et cetera) the result can be catastrophic if not recognized and corrected.

As an alternative, this same user program might contain the statement:

CALL FREFORM(VALU,NVAL).

In this case, execution will result in the representation of "SHOT" stored in VALU(1), the integer, 4395, stored in VALU(2), et cetera. This alternative is clearly simpler for the programmer. However, the real virtue of this approach is that the user at the terminal console is no longer restricted to any exact format. Thus, equivalent results will be obtained from the data sequence,

SHOT^^^4395,SIO2,1E3^.04,^^3.142^235.

Note the (arbitrary) insertion of commas and/or blanks as well as the varied syntactical forms in this second data sequence. Acceptable variations are limitless in number. Thus, FREFORM frees the user from the necessity of remembering and exactly reproducing program formats.

The remainder of this report is organized as follows. Section 2 explains how FREFORM functions in normal and test modes, the method in which data is processed, and the simple rules for data syntax. Section 3 discusses implementation on the DEC-10 and the PDP-11/10 computers and presents results obtained from an actual run using the test and diagnostic algorithms. Flow diagrams for the test algorithm (TESTFF) and the processing algorithm (FREFORM) are presented and explained in Section 4. The complete FORTRAN listing of both algorithms as implemented on the DEC-10 is reproduced in Section 5.

SECTION 2
HOW FREFORM FUNCTIONS

2.1 THE NORMAL, TEST, AND DIAGNOSTIC MODES

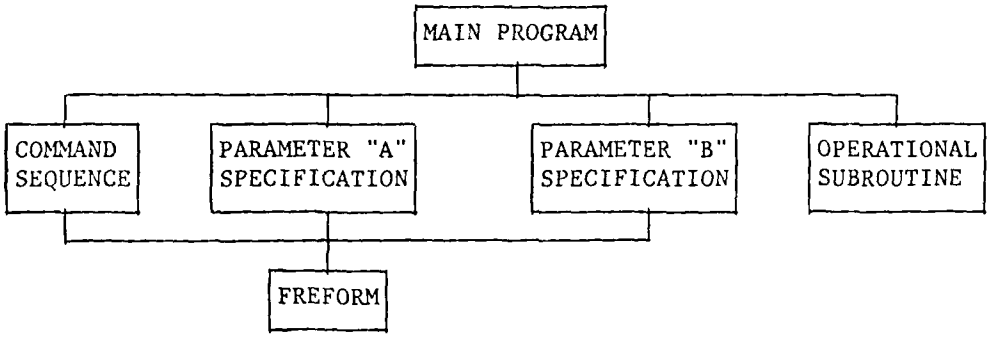
2.1.1 Normal Mode

The subroutine, FREFORM, exists as a library subroutine on the PDP-11/10. As such, it can be accessed by any user program which requires data. Such data might reside on disk, but more commonly would be entered by a user from the terminal console (VT55). Thus, FREFORM provides a means of entering input data to the computer.

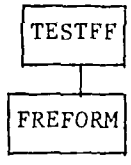
A typical application is illustrated schematically in Figure 1(a). This example consists of a main program and four subroutines, three of which require input data to be entered from the terminal. The command sequence subroutine would be used to enter key words specifying operations to be performed by the program. Additional data (either "A" parameters or "B" parameters) might be required depending upon the specified operations. In all cases, the commands (alphanumeric words), parameters (integer, fixed point, and floating point numbers) and comments (alphanumeric text) can be entered simply from the terminal in response to queries from the input subroutines. Data so entered will automatically be processed by FREFORM.

2.1.2 Test Mode

A main program named TESTFF is also available on the PDP-11/10. This program is designed to validate and demonstrate the performance of FREFORM. This mode of application is illustrated schematically in Figure 1(b).



(a) A Typical Program Application in the Normal Mode.



(b) Application in the Test Mode

Figure 1. Application Modes for FREFORM.

TESTFF allows a user to enter a string of data at the terminal in an arbitrary format. This data will be processed by FREFORM, and the interpreted values will be returned to the user (printed) at the terminal. One of the restrictions on use of TESTFF is that all of the values returned from each sequence string will be printed according to the same format specification. Thus, all of the values submitted in each sequence string must be of the same data type. Of course, this restriction does not apply to the normal use of FREFORM.

2.1.3 Diagnostic Mode

Subroutine FREFORM contains a system diagnostic mode. It is activated by altering a data statement in FREFORM so that the variable, ITST, is set to unity rather than zero. The diagnostic mode causes a listing of all system-dependent parameters and ASCII character variables to be printed on the first call. Subsequently, all record images and process strings will be printed prior to translation. This mode is extremely useful for implementing the algorithm on a new system, or for analyzing anomalous results.

Results from an actual run using the test and diagnostic modes are presented in Section 3.

2.2 DATA PROCESSING

In the normal mode, a sequence string of data will be supplied by the user in response to a query from the user program. Following a call to subroutine FREFORM, the processing of this data occurs in four major steps:

(1) A record image (usually 80 characters) is read into an integer array, JCIM. One character is placed in each array element. A new record image is obtained when all of the characters on the previous image have been processed.

(2) Each character in JCIM is tested against each of the four string delimiters (see below) and possibly stored in an integer array, JCST, which contains the process string. One character is stored in each array element. When a delimiter is detected, testing stops and processing is initiated.

(3) A test is made to determine whether the process string contained in JCST is a numerical value or an alphanumeric character string.

(4) The contents of JCST are interpreted and then stored in the array, VALU, by either a number decoding algorithm or a character string packing algorithm, according to the result of the test. The array, VALU, is returned to the calling program unit (user program) when the entire sequence string has been read, translated, and stored.

2.3 DATA SYNTAX

2.3.1 The Record String and Other Definitions

The total set of data to be processed by FREFORM may be regarded as one contiguous string of characters, referred to as the record string. Four special characters embedded in the record string - the slash, the comma, the asterisk and the blank - serve as delimiters. Except for the special case discussed in the next paragraph, all image boundaries in the data are ignored.

Each call to FREFORM will process a certain amount of data referred to as a sequence string. The length of a sequence string can be varied by two different methods. With the first method, the sequence string terminates whenever a slash delimiter is encountered in the data.

This method is useful when processing large, structured strings in the batch mode. The second method, which is more useful in the iteractive mode, terminates the sequence string when it contains a preset number of record images. (This preset number is contained in a data statement which initializes the variable, NIPS. For terminal applications, a value of unity will cause the scope line (one record image) and the sequence string to be equal. Then each line on the scope will be processed separately, without the need to type slash delimiters to terminate the sequence strings.)

Within the sequence string, individual data are delimited by blanks, commas, or asterisks, as indicated in the discussion of data types, which follows.

2.3.2 Allowable Data Types

FREFORM allows four types of data:

(1) Numerical Constants - Rational number constants are recognized in fixed, integer, and exponential formats (examples - 143, 3.14159, 1.E-07). Data must be separated by space or comma delimiters. Nonessential characters will be automatically supplied during interpretation (thus, for example, 1E-7 and 1.0E-07 will be interpreted identically).

(2) Alphanumeric constants - Alphanumeric constants are recognized by the first character, which must be alphabetical. Succeeding characters may be alphabetical, numerical, or may be any ASCII special character other than the slash or asterisk, or the system-dependent control characters (examples - A123, Z$45, START). Data must be separated by space or comma delimiters.

(3) Null Values - A null value is created by recognition of successive commas, with no intervening characters. (Blanks, or spaces, between adjacent commas will be ignored. For example, the character string (, ,,) will produce two null values.)

(4) Text - Any character string delimited by two asterisks (at beginning and end) will be recognized as text, and will be retained intact with embedded spaces, commas, and all ASCII characters (except the slash and asterisk which would terminate the string, and the system-dependent control characters). The text will be coded in as many words as are required to process the string.

The number of characters devoted to a single numerical or alphanumeric constant may not exceed the value of the variable, NCPS. If the number of characters in any delimited data value exceeds NCPS, a delimiter is assumed after every NCPS characters in the data, beginning with the leftmost character. (For example, with NCPS = 4, the string " 2345.67891E-23 " would be divided into four values: "2345", ".678", "91E-", and "23".) Thus, if one of the values represent an incorrectly formatted numerical constant, an error will occur, as described below.

2.3.3 Error Analysis

FREFORM is intentionally tolerant of varied syntactical forms. However, two types of errors which can occur in the formation of a numerical constant are recognized.

(1) Syntax error in the mantissa - Any data entry (excluding text delimited by asterisks) beginning with a nonalphabetical character is assumed to be a numerical constant. If this constant cannot be interpreted, a syntax error results. (Examples: 25.00F+02, ++15, &STRING)

(2) Syntax error in the exponent - The exponent of a floating point numerical constant must be an integer value. (It may include an algebraic sign). If an exponent cannot be interpreted, a syntax error results. (Examples: 2E2., 2EE3, 2E-I)

For these two cases, a blank value will be substituted for the data entry, and an error message will be printed. A typical error message is shown below:

```
******** ERROR DETECTED BY SUBROUTINE FREFORM IN RECORD IMAGE 5 **********
TEXT HERE IS AN ERROR DIAGNOSTIC   25F/
COLUMN:  36 PROCESS STRING:  25F
********************************************************************************
```

The message contains information helpful in locating and analyzing the error. The first line gives the number of the record image on which the error was encountered. Images are numbered sequentially beginning from the first call to FREFORM. The second line reproduced the first eighty (80) characters of the image. The third line identifies the column number of the entry and reproduces the numerical value of the process string.

SECTION 3
IMPLEMENTATION AND TESTING

3.1 SYSTEM-DEPENDENT PARAMETERS

Data is read from a file with logical unit number 5. Thus, input data from an I/O device (such as a card reader or console) will be processed if the logical unit number 5 is assigned to the input file by the calling program, or if the logical unit number 5 is associated with the I/O device by the system. Output is written to a file with logical unit number 6. Thus, messages will be sent to a printer if the logical unit number 6 is assigned to the output file by the calling program, or if this association is made by the system.

In addition, certain parameters and ASCII character variables are defined in data statements in FREFORM. Parameter values suitable for the DEC-10 and the PDP-11/10 computers are given in Table 1. In general, these parameters must be selected for system compatibility as well as the intended application. The parameters NCIW and NCRW (number of characters per integer word or real word, respectively) are needed to pack character data and to decode numerical data. The parameter NCPS (number of characters per process string) determines the length of the largest numerical constant which may be processed; NCPS should be small enough that numbers beyond the range of the machine will not be accepted. Since NCPS also determines the length of the largest alphanumeric constant which may be processed, it is often convenient to make NCPS some integral multiple of NCRW. (Note also that the length of the JCST array must always be greater than NCPS. This prevents overflow of the array and ensures the existence of at least one blank at the end of every process string.)

TABLE 1

SYSTEM-DEPENDENT PARAMETERS

Parameter	DEC-10	PDP-11/10
NCIW	5	2
NCRW	5	4
NCPS	20	20
NCPI	80	80
NIPS	1	1

The proper value of NCPI (number of characters per image) depends on the file or I/O device from which data is read. A large value may be useful when reading large sections of text since all of the text can be returned at one time. Setting NIPS to unity, however, makes each sequence string exactly one record image in length. This choice is more suitable for terminal applications, since it allows the operator to omit slashes from the data sequence.

THE ASCII character variables are used by FREFORM for comparison tests. JCSL, JCAS, JCSP and JCCO are used to identify delimiters. JCMI, JCPL, JCAE, JCDP, JCNI and JCNF are used to identify parts of a numerical constant. JCAI and JCAF are used to determine whether the first character in a process string is alphabetic. The bits containing the character in each of the character variables must occupy the same position within the machine word as they do in the storage arrays JCIM and JCST for the comparison tests to function correctly. The diagnostic mode in FREFORM can be enabled by changing the value of a flag (ITST is initialized in a data statement) from zero to unity. This will result in a listing of the system dependent parameters when FREFORM is first executed. In addition, all data processed in this mode will be listed at various stages of processing. This diagnostic capability is provided to aid in implementation of FREFORM on a different system. An example of its use follows.

3.2 TESTING WITH TESTFF

The main program, TESTFF, is intended to provide demonstration and validation capability. It allows a terminal operator to send data sequences to FREFORM, and to immediately inspect the processed results.

To use TESTFF, the type of data (see Section 2.3.2) must be specfied by a keyword at the beginning of each sequence string. (If the data type is not specified in the first sequence string, program execution

is terminated.) Once a data type has been specified, all sequence strings are assumed to be of that type until a new type is specified. TESTFF recognizes three data type keywords: "INTG", "FLTG", and "TEXT". Program execution is terminated by the keyword, "STOP", which must occur at the beginning of a sequence string. Any data following the word "STOP" will not be printed by program TESTFF.

Table 2 presents output from a sample run in the diagnostic mode made interactively from a terminal console. Note that the output originates both from TESTFF and from FREFORM (diagnostic mode). The source is clearly indicated by the headers at the beginning and end of each section of output.

The first execution of FREFORM (diagnostic mode is enabled) produces a listing of the system-dependent parameters and ASCII characters. In the first example, the data sequence begins with the keyword "TEXT". Diagnostic output from FREFORM shows the actual process strings, while the TESTFF output lists the representation of each word in the array, VALU. Thus, characters representing the word "TEXT^" are stored in VALU(1), and so forth.

The second example differs from the first in that alphanumeric text is used (asterisk delimiters). Note the embedded blanks, commas, et cetera in the processed output. The third example involves integer numbers. Note the representation of a null value, produced by two adjacent commas. Floating point numbers are processed in the fourth example, which includes a deliberate error (5F2 is syntactically unrecognizable). The resulting error message would have been produced even if the diagnostic mode is disabled.

The final example consists of three sequence strings of text. (The first two are delimited by slash characters.) Note that the integer

value, 2, in the second string is correctly interpreted by FREFORM, even though TESTFF attempts to print the result in a text (alphanumeric) format. The final sequence string contains the keyword "STOP", which terminates execution of TESTFF.

TABLE 2

OUTPUT FROM SAMPLE RUN OF FREFORM
IN THE DIAGNOSTIC MODE

```
********************** PROGRAM TESTFF  -  MARCH 1981 **********************

************************** SUBROUTINE FREFOR DIAGNOSTIC **************************

 ASCII CHARACTER VARIABLES

 JCHE    JCAF    JCAI    JCAS    JCCO    JCDP
   E       Z       A       *       ,       .

 JCNI    JCNF    JCNI    JCPL    JCSL    JCSP
   -       9       0       +       /

 SYSTEM-DEPENDENT PARAMETERS

 NCIU    NCRW    NCPI    NCPS    NIPS
   5       5      80      20       1

******************************** END DIAGNOSTIC ********************************

TEXT     ALPHANUMERIC CHARACTER STRINGS:  RUN5  Z123, AT@SYMBOL

************************** SUBROUTINE FREFOR DIAGNOSTIC **************************

 SEQUENCE STRING:   1  RECORD IMAGE:   1

TEXT     ALPHANUMERIC CHARACTER STRINGS:  RUN5  Z123, AT@SYMBOL

 COLUMN:  1  PROCESS STRING: TEXT
 COLUMN: 10  PROCESS STRING: ALPHANUMERIC
 COLUMN: 23  PROCESS STRING: CHARACTER
 COLUMN: 33  PROCESS STRING: STRINGS:
 COLUMN: 43  PROCESS STRING: RUN5
 COLUMN: 49  PROCESS STRING: Z123
 COLUMN: 55  PROCESS STRING: AT@SYMBOL

********************************** END DIAGNOSTIC **********************************

********** END OF FROM PROGRAM TESTFF **********
```

TABLE 2, Cont.

```
     TEXT
     ALPHA
     NUMER
     IC
     CHARA
     CTER
     STRIN
     GS:
     RUNS
     Z123
     ATOSY
     MBOL

********************** END OUTPUT FROM PROGRAM TESTFF **********************

*TEXT STRING WITH EMBEDDED BLANKS, COMMAS, SPECIAL CHARACTERS & NUMBERS !  700*

********************** SUBROUTINE FREFOR DIAGNOSTIC **********************

 SEQUENCE STRING:   2  RECORD IMAGE:   2

*TEXT STRING WITH EMBEDDED BLANKS, COMMAS, SPECIAL CHARACTERS & NUMBERS !  700*

 COLUMN:  1  PROCESS STRING: TEXT STRING WITH EMB
 COLUMN: 21  PROCESS STRING: EDDED BLANKS, COMMAS
 COLUMN: 41  PROCESS STRING: , SPECIAL CHARACTERS
 COLUMN: 62  PROCESS STRING:  & NUMBERS !  700

********************** END DIAGNOSTIC **********************



********************** OUTPUT FROM PROGRAM TESTFF **********************

     TEXT
     STRIN
     G WIT
     H EMB
     EDDED
      BLAN
     KS, C
     OMMAS
     , SPE
     CIAL
     CHARA
     CTERS
      & NU
```

TABLE 2, Cont.

```
    MBERS
     ! 7
    00

********************** END OUTPUT FROM PROGRAM TESTFF **********************

INTG     700  600, 500,    ,300

********************** SUBROUTINE FREFOR DIAGNOSTIC **********************

 SEQUENCE STRING:   3  RECORD IMAGE:   3

INTG     700  600, 500,    ,300

 COLUMN:  1  PROCESS STRING: INTG
 COLUMN: 10  PROCESS STRING: 700
 COLUMN: 15  PROCESS STRING: 600
 COLUMN: 20  PROCESS STRING: 500
 COLUMN: 27  PROCESS STRING:
 COLUMN: 29  PROCESS STRING: 300

********************** END DIAGNOSTIC **********************



********************** OUTPUT FROM PROGRAM TESTFF **********************

    INTG
      700
      600
      500

      300

********************** END OUTPUT FROM PROGRAM TESTFF **********************

FLTG     -7E+002,    6E2    5E2    3E2    300

********************** SUBROUTINE FREFOR DIAGNOSTIC **********************

 SEQUENCE STRING:   4  RECORD IMAGE:   4

FLTG     -7E+002,    6E2    5E2    3E2    300

 COLUMN:  1  PROCESS STRING: FLTG
 COLUMN: 10  PROCESS STRING: -7E+002
 COLUMN: 17  PROCESS STRING: 6E2
 COLUMN: 23  PROCESS STRING: 5E2
```

TABLE 2, Cont.

```
 COLUMN: 36  PROCESS STRING: 5F2


********** ERROR DETECTED BY SUBROUTINE FREFOR IN RECORD IMAGE    4 **********

FLTG     -7E+002,    6E2    5E2    5F2    300

 COLUMN: 36  PROCESS STRING: 5F2
********************************************************************************



 COLUMN: 43  PROCESS STRING: 300

********** END DIAGNOSTIC **********


********** OUTPUT FROM PROGRAM TESTFF **********

    FLTG
-0.700000E+03
 0.600000E+03
 0.500000E+03

 0.300000E+03

********** END OUTPUT FROM PROGRAM TESTFF **********

TEXT     THREE SEQUENCE STRINGS / HERE IS NO. 2 / STOP

********** SUBROUTINE FREFOR DIAGNOSTIC **********

 SEQUENCE STRING:   5  RECORD IMAGE:   5

TEXT     THREE SEQUENCE STRINGS / HERE IS NO. 2 / STOP

 COLUMN:  1  PROCESS STRING: TEXT
 COLUMN: 10  PROCESS STRING: THREE
 COLUMN: 16  PROCESS STRING: SEQUENCE
 COLUMN: 25  PROCESS STRING: STRINGS

********** END DIAGNOSTIC **********
```

TABLE 2, Cont.

```
********** OUTPUT FROM PROGRAM TESTFF **********

    TEXT
    THREE
    SEQUE
    NCE
    STRIN
    GS

********** END OUTPUT FROM PROGRAM TESTFF **********

 COLUMN: 35  PROCESS STRING: HERE
 COLUMN: 40  PROCESS STRING: IS
 COLUMN: 43  PROCESS STRING: NO.
 COLUMN: 47  PROCESS STRING: 2

********** END DIAGNOSTIC **********



********** OUTPUT FROM PROGRAM TESTFF **********

    HERE
    IS
    NO.
     P

********** END OUTPUT FROM PROGRAM TESTFF **********

 COLUMN: 51  PROCESS STRING: STOP

********** END DIAGNOSTIC **********



********** OUTPUT FROM PROGRAM TESTFF **********

    STOP

********** END OUTPUT FROM PROGRAM TESTFF **********


********** PROGRAM TESTFF - EXECUTION COMPLETED **********
```

SECTION 4

FLOW DIAGRAMS

This section describes the operation of program TESTFF and subroutine FREFORM. In general, the code breaks down into a number of relatively well-defined algorithms. Each such algorithm is presented on a separate flowchart, and is given a unique name, such as "INTEGER", for example. Then the use of this particular algorithm by other sections of the code would be indicated by the symbol, [INTEGER] , in the flowcharts.

The following discussion refers to the flowcharts of TESTFF, shown in Figures 2, 3, and 4. At the beginning of the program, a header is printed. The major loop is then executed until it is terminated by the flag ISTP. Inside the loop, FREFORM is called. The second loop (number 2) checks the data value in the first element of array VALU against each of the four keywords "INTG", "FLTG", "TEXT", and "STOP". (The keywords are contained in an array, VWRD.) If a match occurs, the flag JFLG is set to the current value of the loop index, INDX. Following the loop, a header is printed. A "computed GOTO" on JFLG is then executed causing a branch to one of four program blocks which corresponds to the keyword matched. Three of the blocks (corresponding to keywords "INTG", "FLTG", and "TEXT") cause the contents of VALU to be printed in a format described by the keyword. Use of the variable NFST in program blocks "INTG" and "FLTG" permits the keyword to be printed in character code format. The fourth block sets the flag ISTP and causes the word "STOP" to be printed. When the major loop is terminated, a footing is printed, and execution stops.

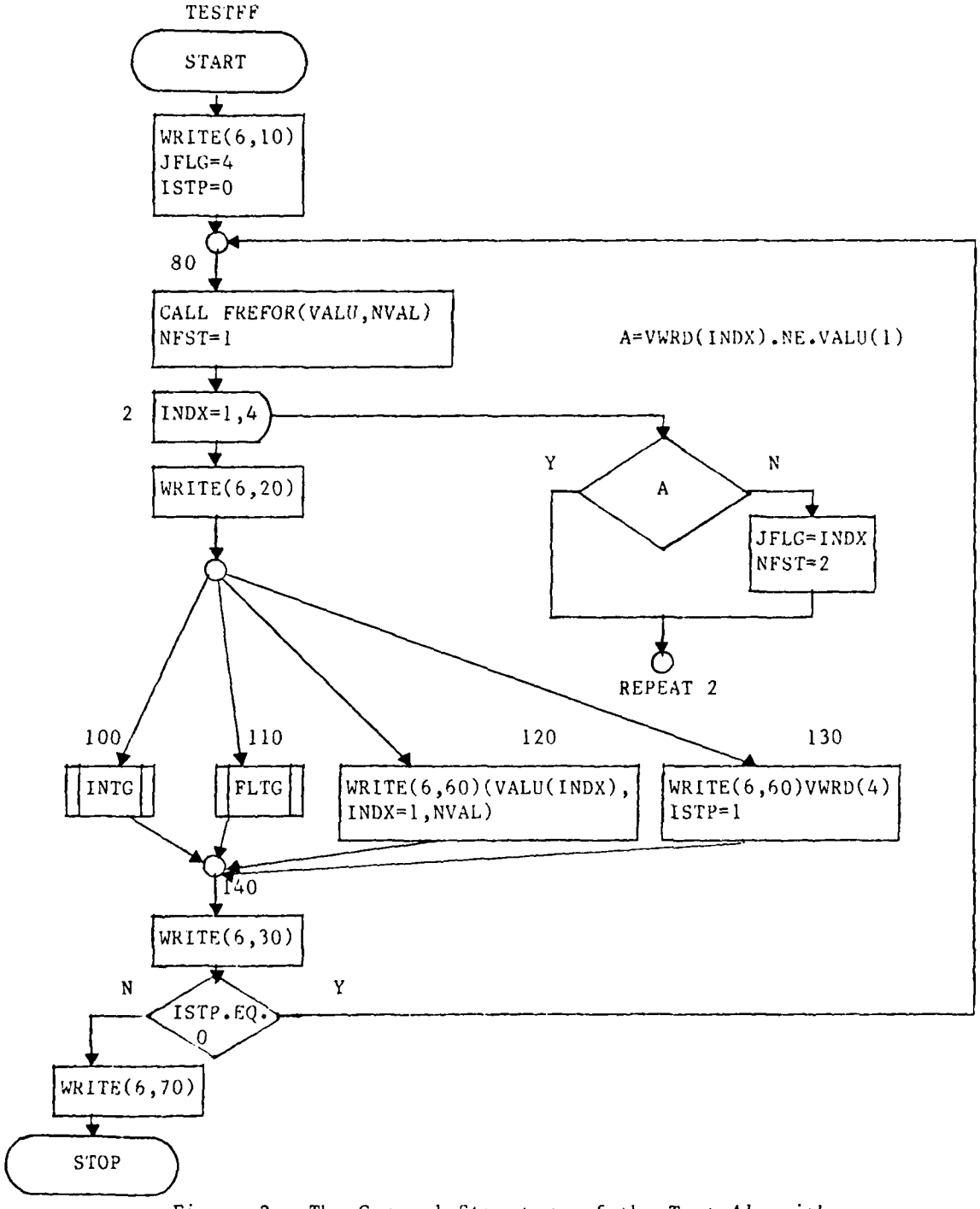


Figure 2. The General Structure of the Test Algorithm.

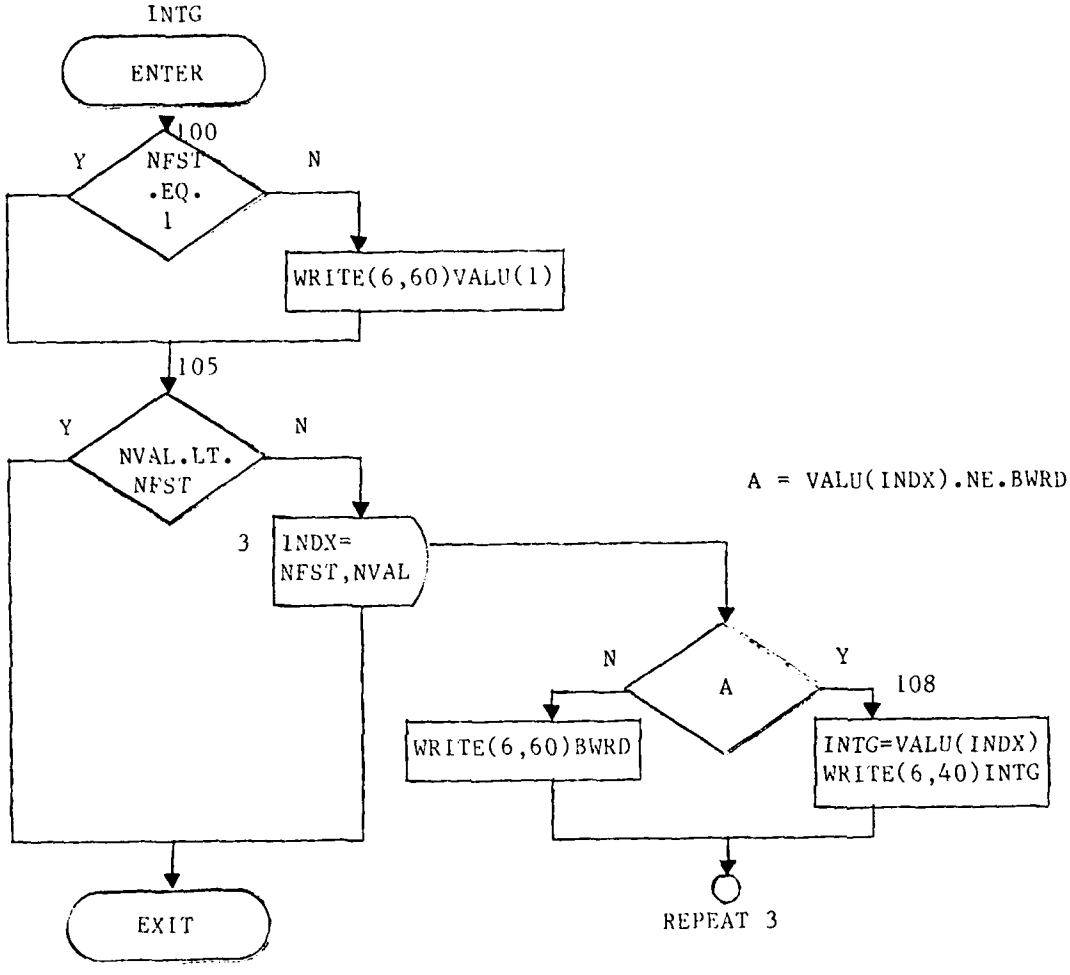


Figure 3. Processing Integers in the Test Algorithm.

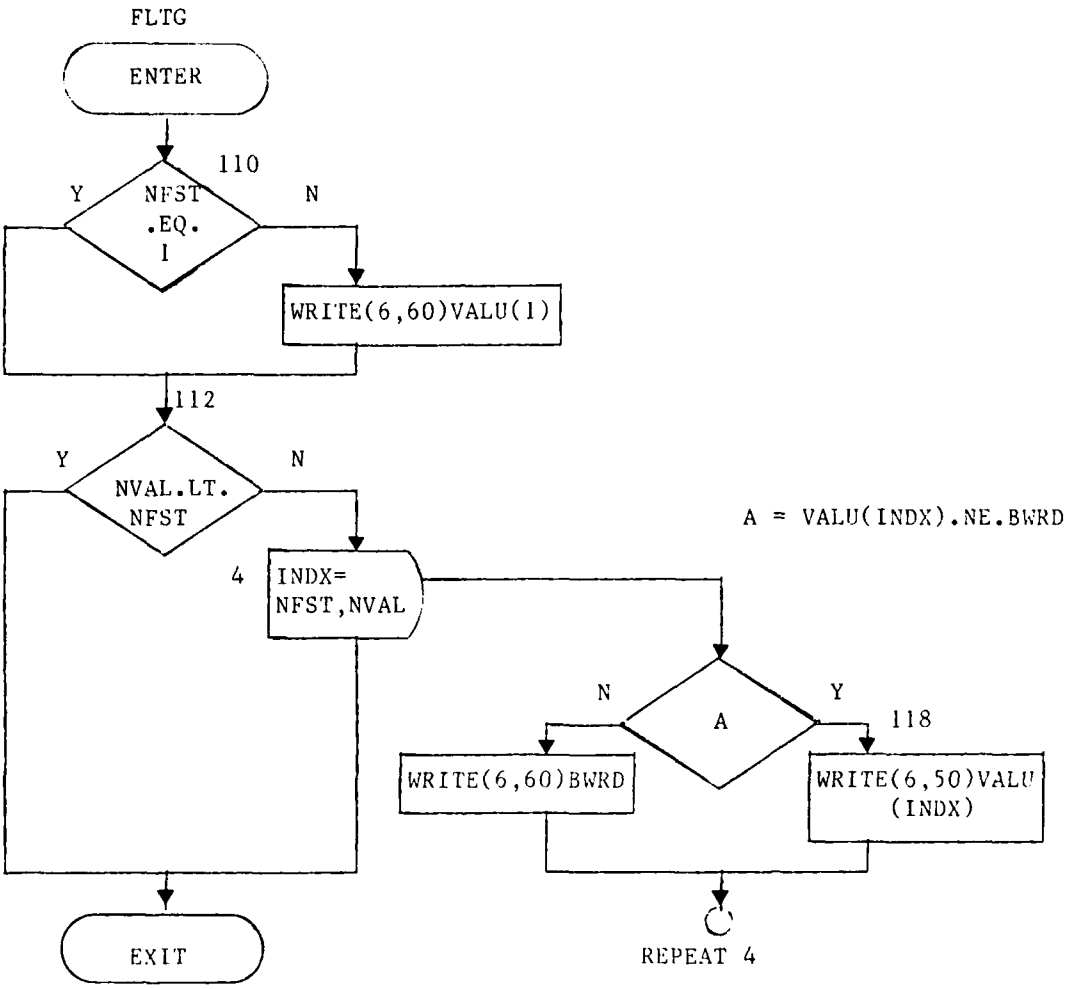


Figure 4. Processing Floating Point in the Test Algorithm.

A complete list of variables and definitions used in subroutine FREFORM is presented in Table 3. The general structure of FREFORM is shown in Figure 5. The single alternative decision block at line 260 is part of the diagnostic package. The rest of the diagnostic package is contained in the decision block in Figure 6, and in "DIAGNOSE1" and "DIAGNOSE2"(Figure 7). Note that "DIAGNOSE1" is executed only on the first call to the subroutine.

The major loop (number 1) in Figure 5 is executed until the end of the current sequence string. Section 2.2 describes the basic steps which occur inside this loop. Steps (1) and (2) mentioned there occur in the "FILLSTRING" algorithm (Figure 8). The test described in Step (3) is shown in Figure 5 at line 240. Step (4) occurs in either "NUMBER" (Figure 12) or "CHARACTER" (Figure 19).

Figure 8 shows the "FILLSTRING" algorithm. In the first loop (number 2), the array JCST is filled with blanks. The process string loop (number 3) is executed until the process string is terminated. The decision block at the top of the loop fetches a new record image when needed (see "READCARD" in Figure 9) or terminates the sequence string when NIPS records have been read. Each time through the loop, one character from the current record image is tested and may be placed in the process string. Testing and storing occurs in "TEST1" (Figure 10) and "TEST2" (Figure 11). Note that the process string may be terminated by a blank, a comma, or a slash, or by the test at line 380, when NCPS characters have been read.

The "NUMBER" algorithm is shown in Figure 12. Integers, as well as the integer portion of fixed point numbers and floating point number mantissas are all processed in "INTEGER" (Figure 13). The portion of any number to the right of a decimal point is processed in "DECIMAL" (Figure 15) and stored in "STORDEC" (Figure 16). Exponents are processed in "EXPONENT" (Figure 17) and stored in "STOREXP" (Figure 18). In "STORNUM" (Figure 19), valid processed numbers are placed in the VALU array and an error message is printed for incorrectly formatted numbers.

TABLE 3
VARIABLE DEFINITIONS FOR FREFORM

DFCT - factor used to compute DSTO
DSTO - variable to store decimal portion of number
ESGN - sign of exponent
ESTO - variable to store exponent
EXPN - exponential multiplier for number processing
IALP - flag for alphanumeric character string processing
ICIM - image position of first character in the process string
ICOM - flag to detect sequential commas
IERR - flag to indicate error in numerical value
IFLD - flag to indicate text processing
IND0 - index for general use
IND1 - index for general use
IONC - flag to indicate first execution of subroutine
IPRO - flag to initiate word processing
IRET - flag to exit subroutine after sequence completion
ISTO - variable to store integer portion of number
ISTR - flag to transfer character from image to process string
ITST - flag to produce system diagnostic
JCAE - ASCII character for fifth alphabetical character, (E)
JCAF - ASCII character for final alphabetical character, (Z)
JCAI - ASCII character for initial alphabetical character, (A)
JCAS - ASCII character for asterisk symbol, (*)
JCCO - ASCII character for comma symbol, (,)
JCDP - ASCII character for decimal point, (.)
JCHA - current character in the record image
JCIM - array of characters in the record image
JCMI - ASCII character for minus symbol, (-)
JCNF - ASCII character for final decimal character, (9)
JCNI - ASCII character for initial decimal character, (0)

TABLE 3 - continued

JCPL - ASCII character for plus symbol, (+)
JCSL - ASCII character for slash symbol, (/)
JCSP - ASCII character for space symbol, ( )
JCST - array of characters in the process string
JCTF - character transfer variable in number processing
JNUM - array of decoded digits for number processing
NCCT - number of characters counted
NCIM - pointer for the image string character array
NCIW - maximum number of characters per integer word
NCLW - number of characters in the last word processed
NCPI - maximum number of characters per image string
NCPS - maximum number of characters per process string
NCRW - maximum number of characters per real word
NCST - actual number of characters in the process string
NCTF - number of characters transferred in number processing
NFST - position of first character to be transferred
NIMS - number of images processed in current sequence string
NIMT - total number of record images processed
NIPS - maximum number of images per sequence
NLST - position of last character to be transferred
NPRO - pointer for the process string character array
NSEQ - total number of sequence strings processed
NSGN - sign of number
NVAL - pointer for the output array VALU
NWTF - number of words transferred in number processing
RNUM - array containing decoded digits in number processing
VALU - output array containing processed data

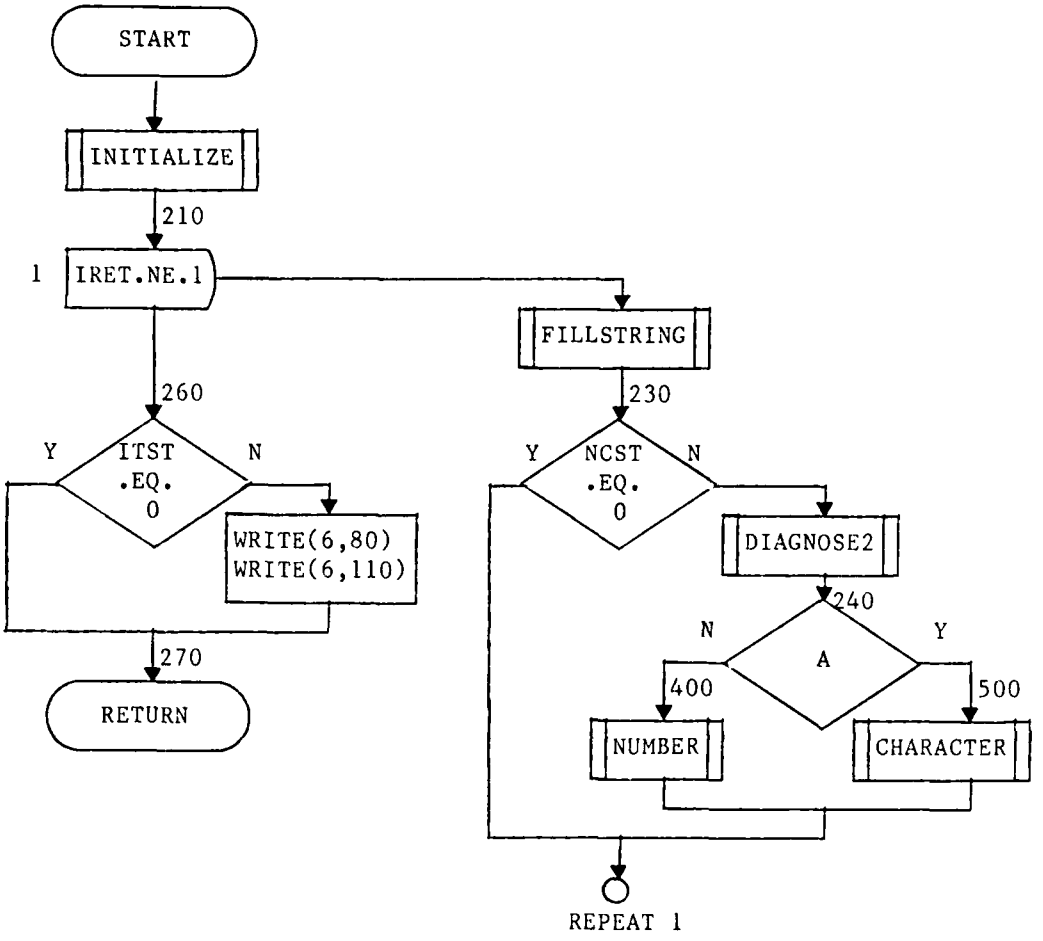
The "INTEGER" and "EXPONENT" algorithms have similar structures. A decision block at the beginning detects an algebraic sign, and a counting loop then determines the number of numerals before the next non-numerical character. Note that the character which terminates the counting loop may be the blank which is appended to every process string. Each of the numerals counted is used to form an integer value in "STORINT" or "STOREXP".

The formation of an integer value from ASCII characters is accomplished by an ENCODE/DECODE statement pair and a small loop, all located within a larger outer loop. ASCII characters in the elements of JCST are packed into a temporary variable, JCTF, by the ENCODE statement. The characters are then converted to integer representation one at a time, and placed in the elements of JNUM by the DECODE statement. The integer representations in the elements of the JNUM array are multiplied by decreasing powers of (10) and summed.

"DECIMAL" is similar to "INTEGER" and "EXPONENT", with two exceptions. First, no tests are made for an algebraic sign. Second, each of the numerals counted is used to form a decimal fraction. Note that "DECIMAL" is not executed unless the character terminating the counting loop in "INTEGER" is a decimal point.

Error analysis occurs in two places in the number processing algorithm. The first check occurs after the integer and decimal processing algorithms, at line 450 of "NUMBER". A test is made to determine whether the next character is the letter "E". If so, then the exponent is processed; if not, then an error flag is set. The second check occurs at the end of "EXPONENT". Again, a test is made to determine whether the end of the process string has been reached. If not, then an error flag is set.

In the "CHARACTER" algorithm (Figure 19), every character in the process string (except the appended blank) is packed into the VALU array with an ENCODE statement. The counter NVAL is then incremented by the number of words stored in VALU.



A = (IALP.EQ.1).OR. (JCST(1).LE.JCAF.AND.JCST(1).GE.JCAI)

Figure 5. The General Structure of FREFORM.

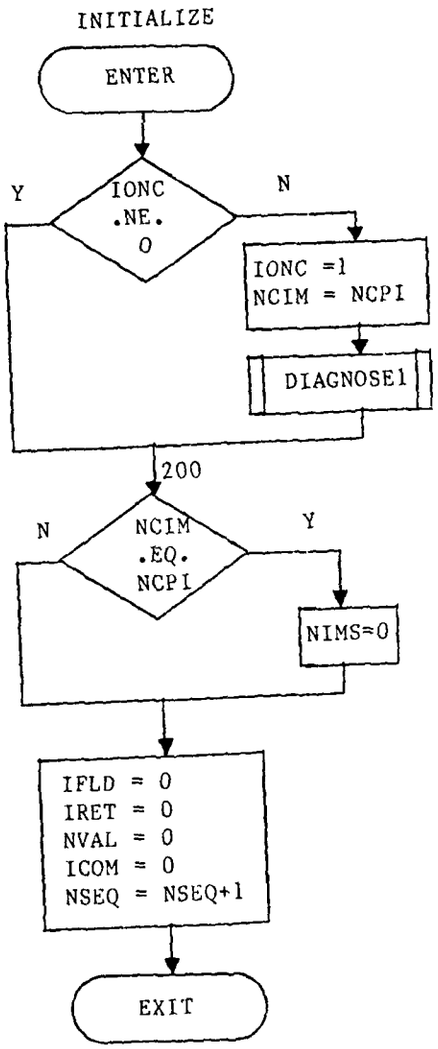


Figure 6. Initializing FREFORM

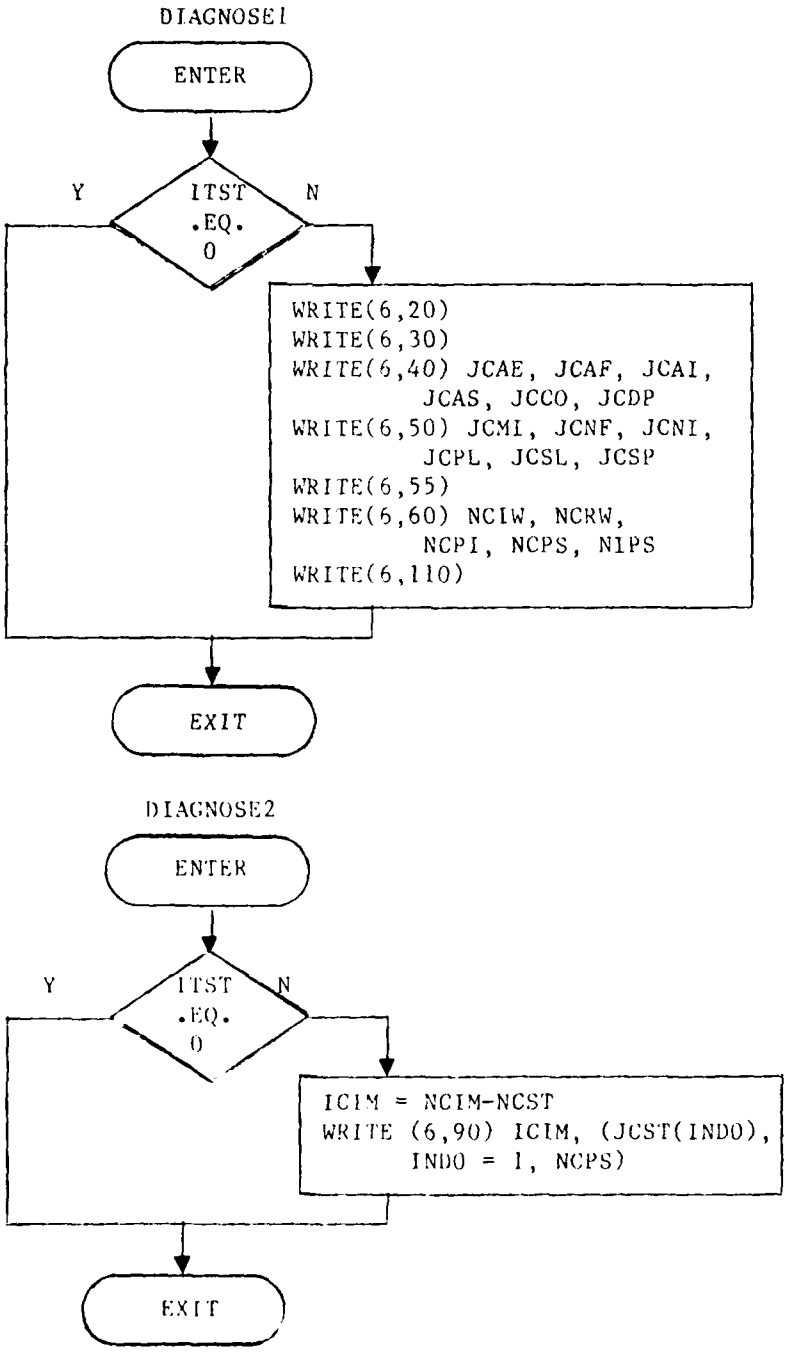


Figure 7. The Diagnostic Package.

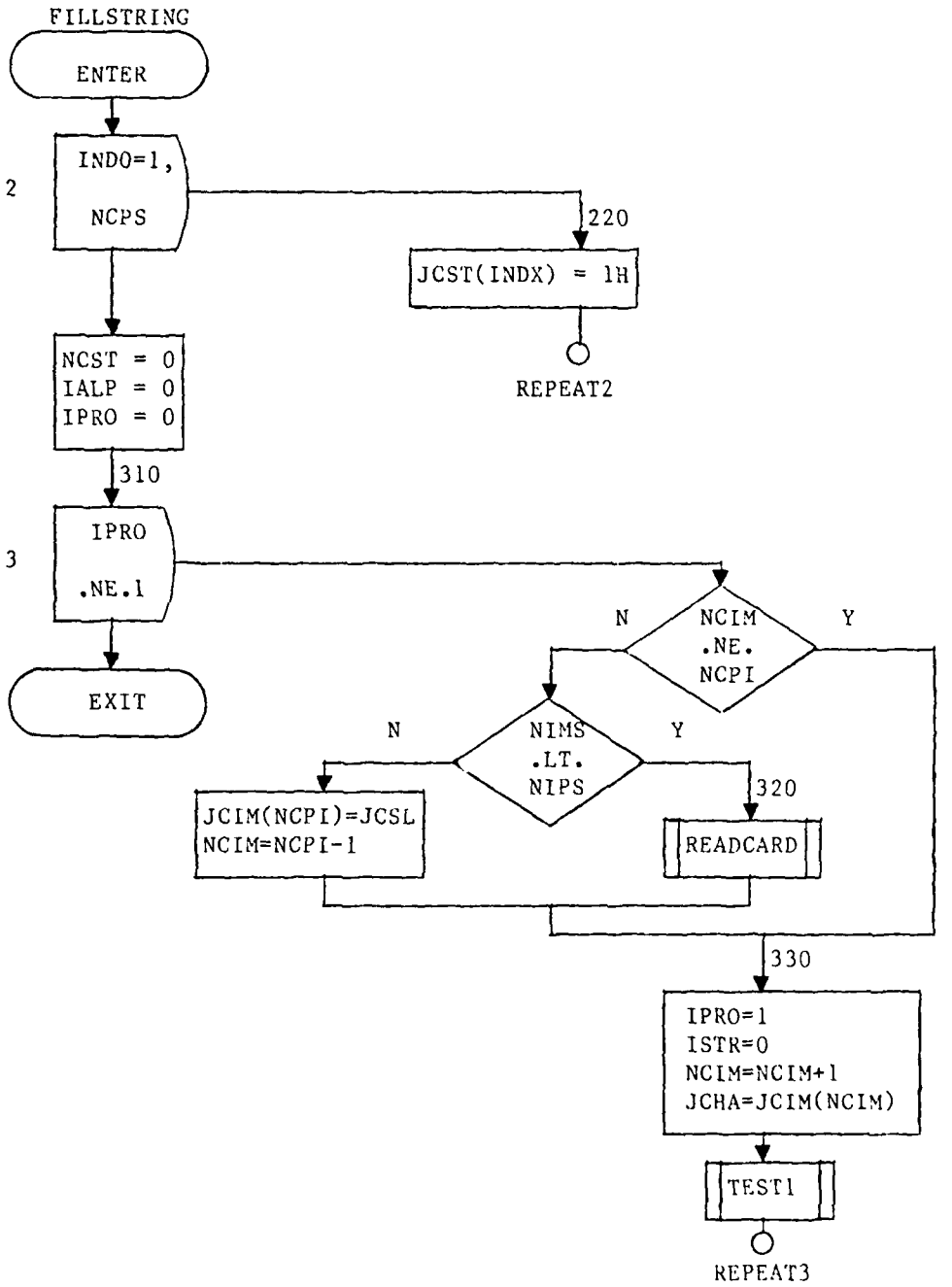


Figure 8. Filling the Process String.

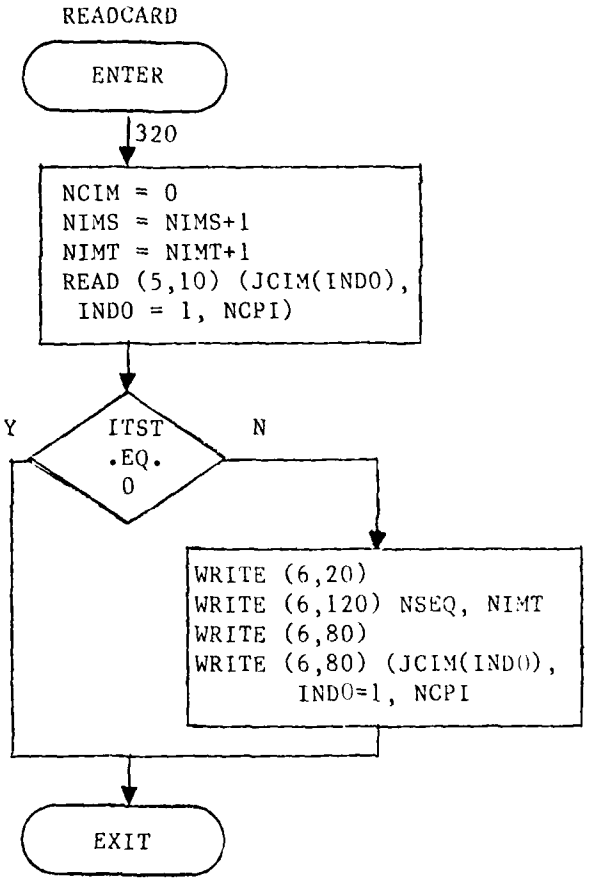
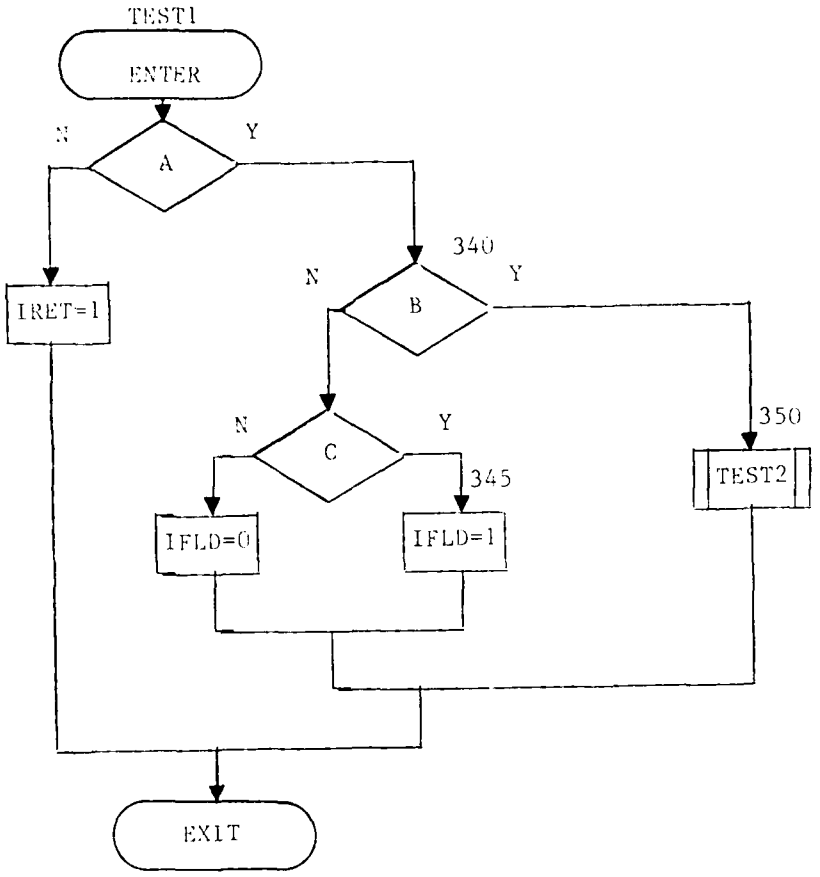
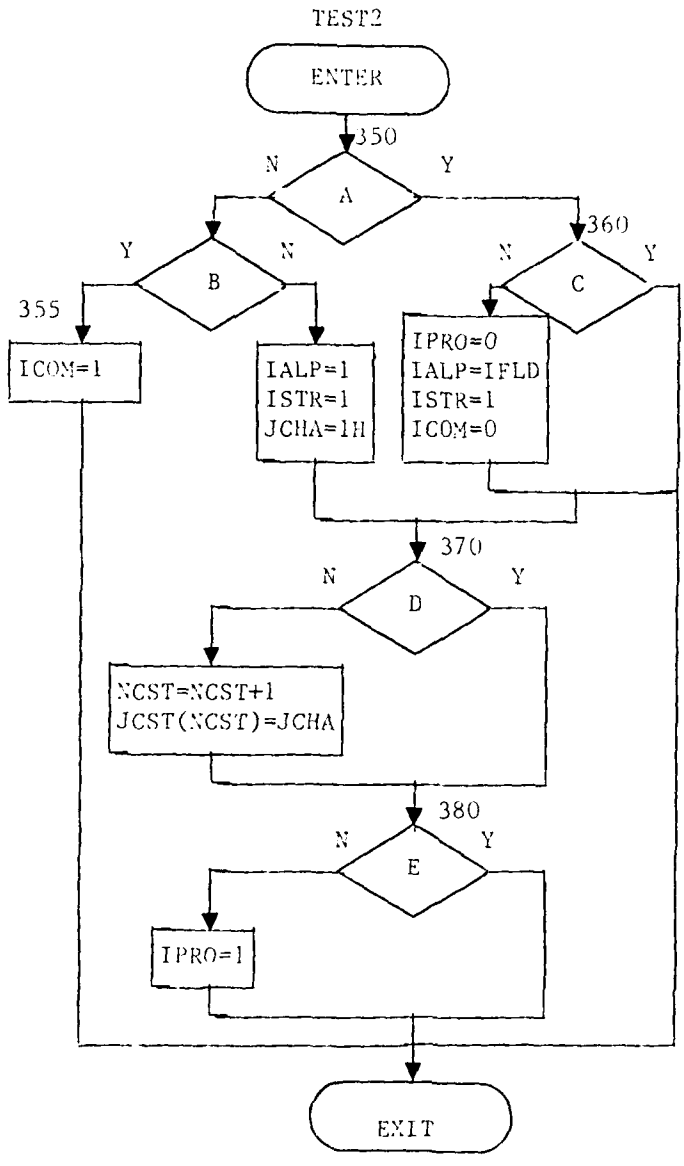


Figure 9. Obtaining a New Record Image.



A = (JCHA.NE.JCSL)
B = (JCHA.NE.JCAS)
C = (IFLD.NE.1)

Figure 19. Testing and Storing Characters.



A = (.NOT.(JCHA.EQ.JCCO.AND.IFLD.EQ.0))
B = (ICOM.NE.1)
C = (JCHA.EQ.JCSP.AND.IFLD.EQ.0)
D = (ISTR.EQ.1)E = (NCST.LT.NCPS)

Figure 11. Testing and Storing Characters (continued).

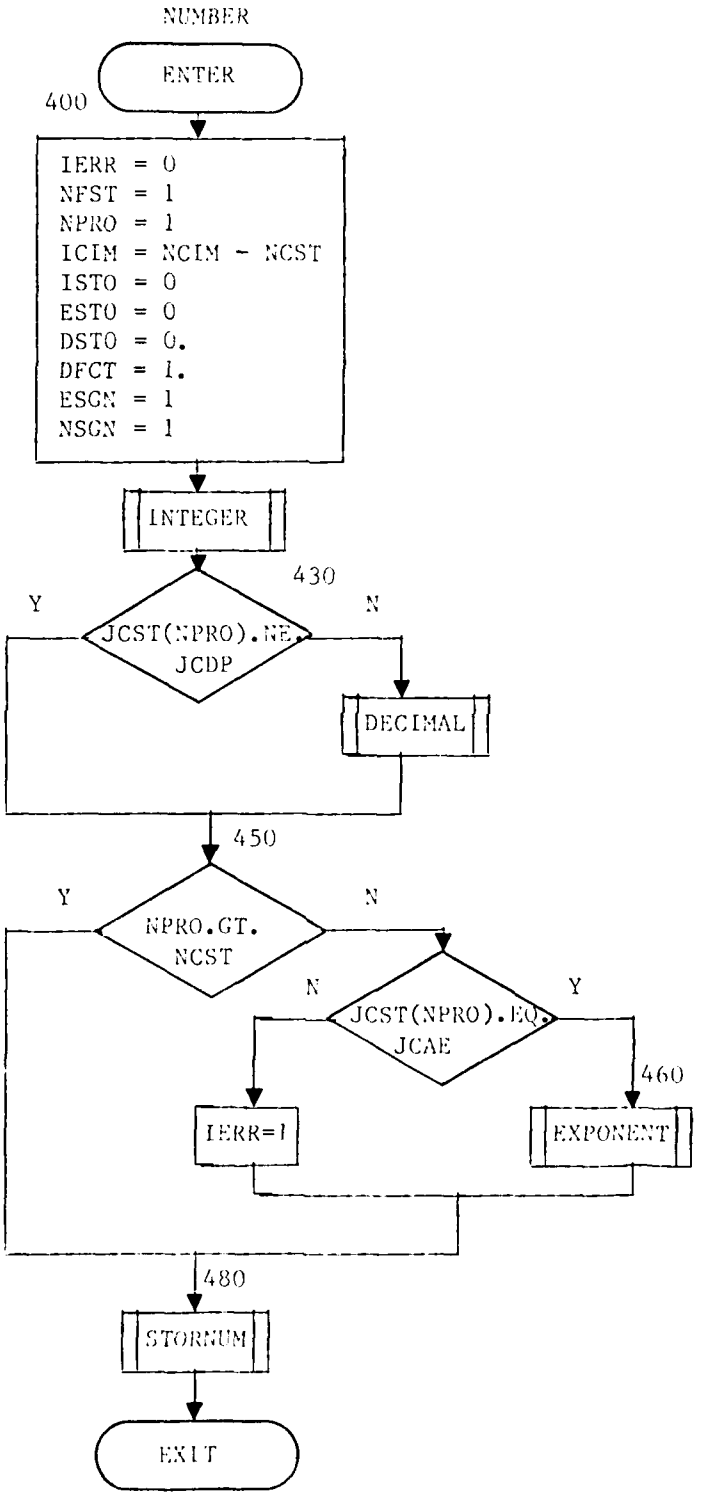


Figure 12. Processing Real Numbers.

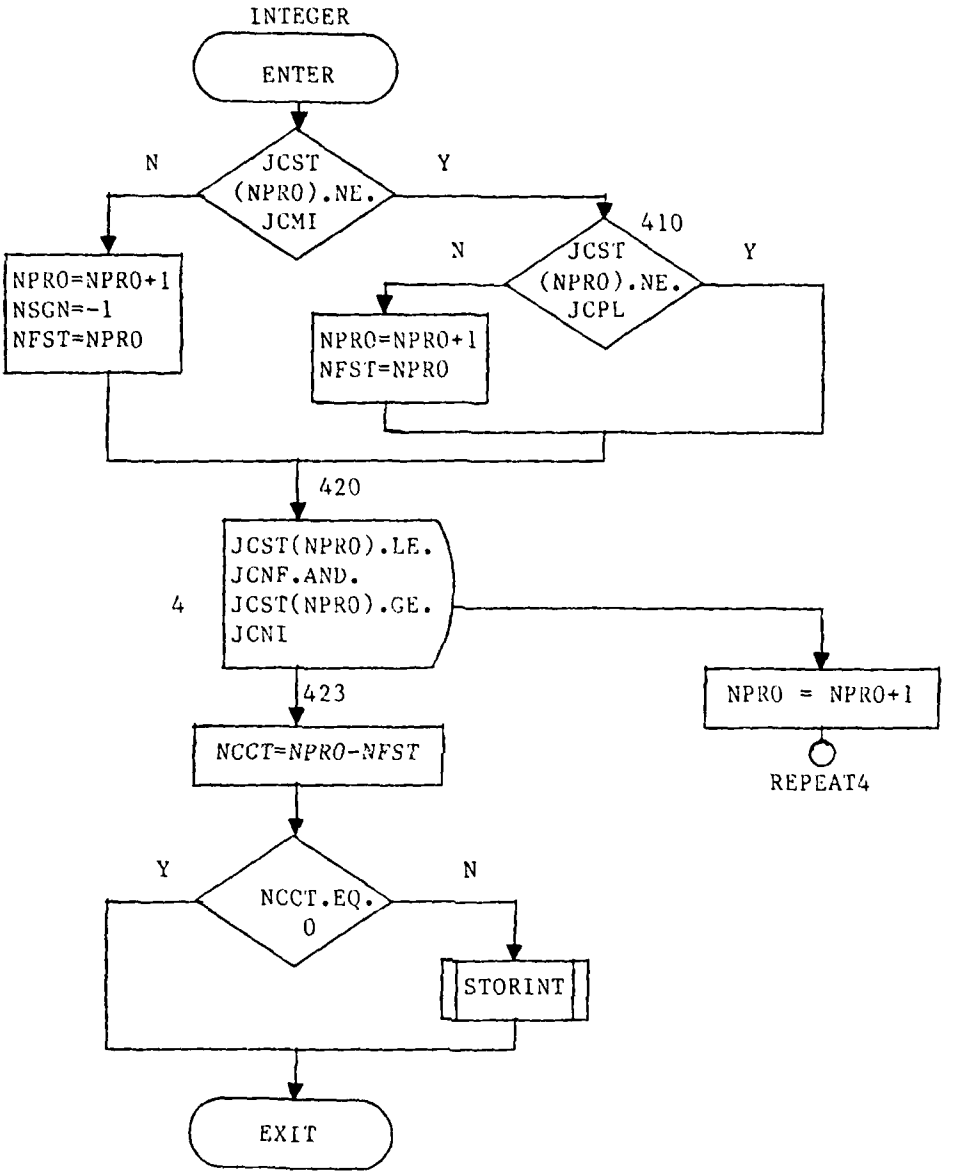


Figure 13. Processing Integer Portion.

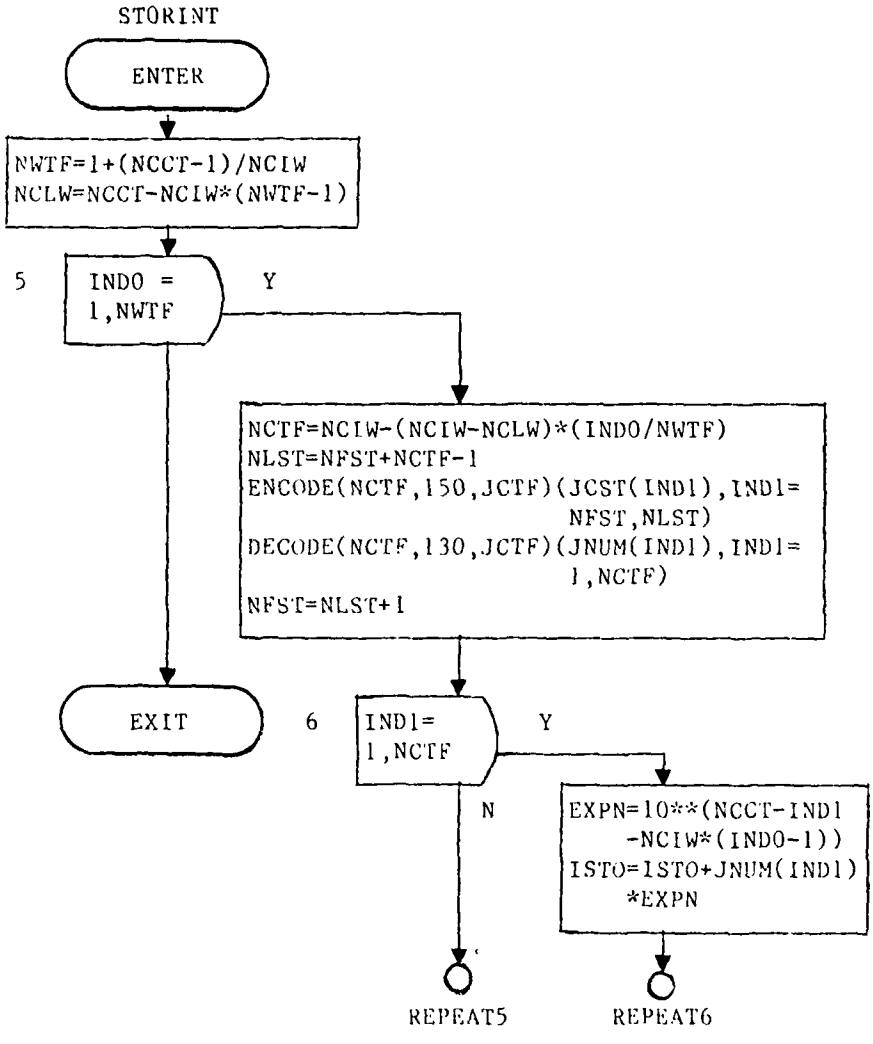


Figure 14. Storing Integer Portion.

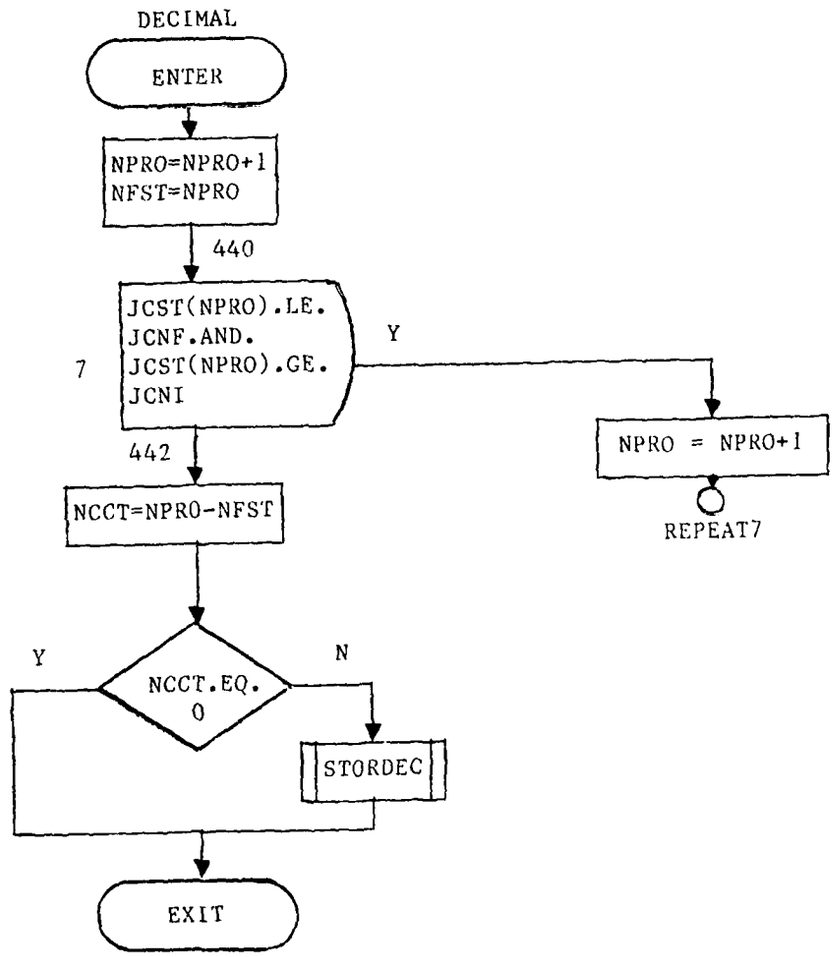


Figure 15. Processing Decimal Portion.

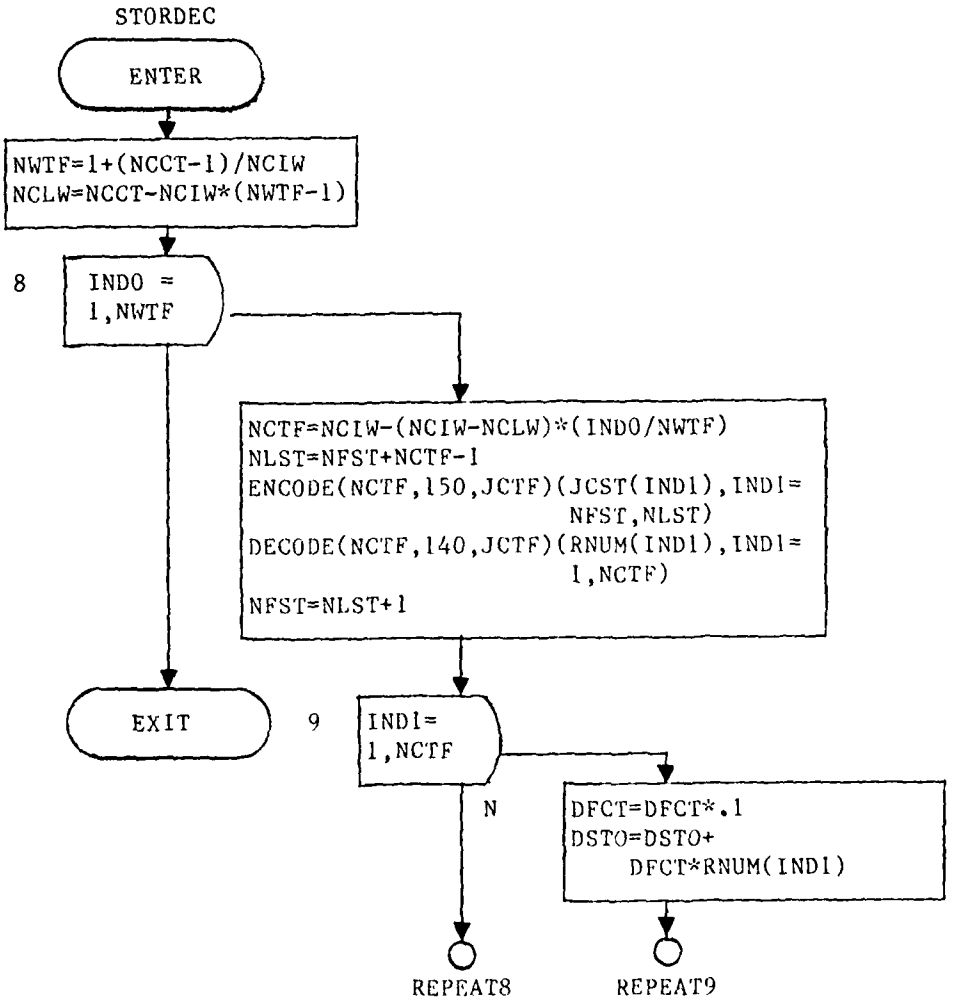


Figure 15. Storing the Decimal Portion.

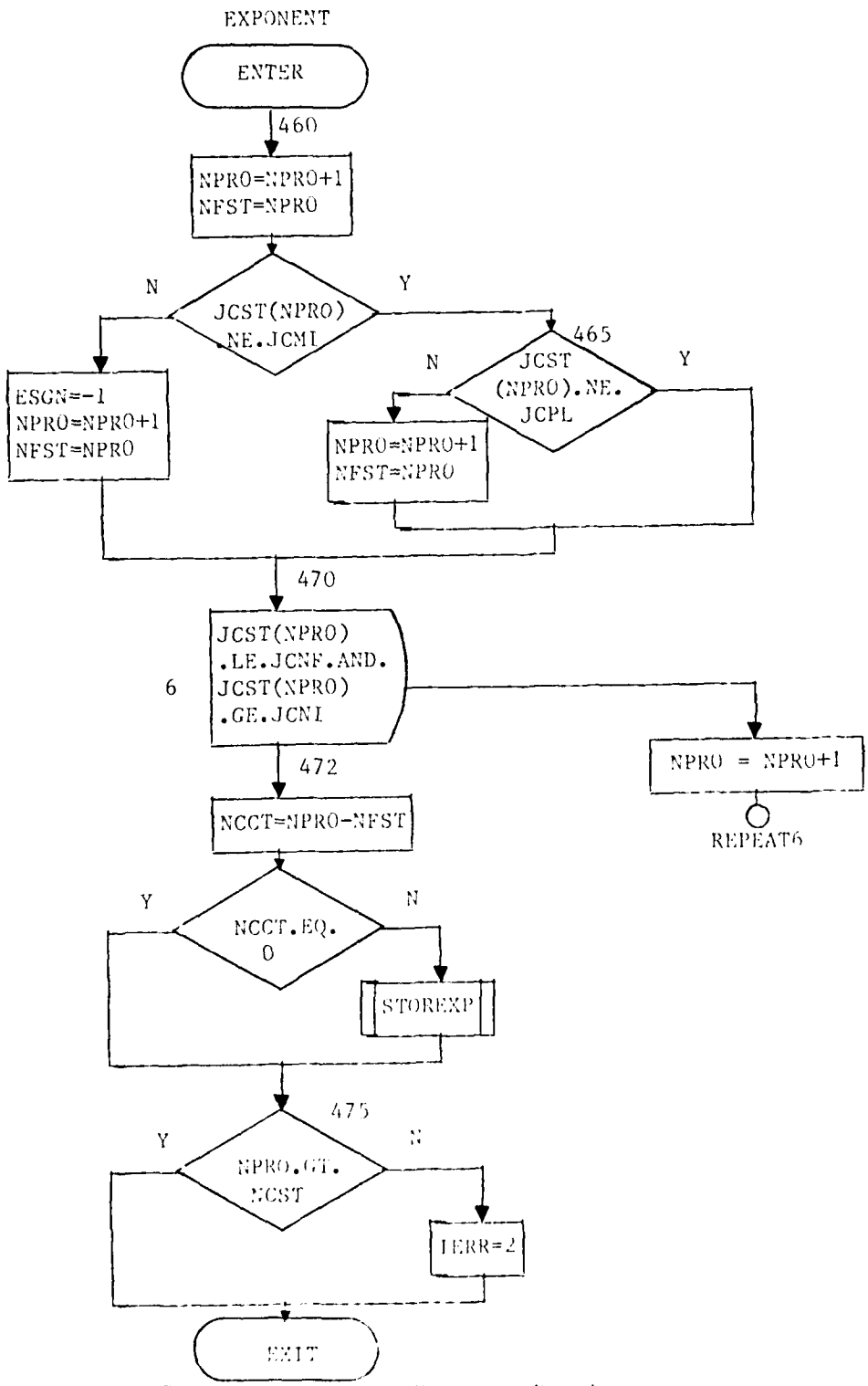


Figure 17. Processing the Exponent Portion.

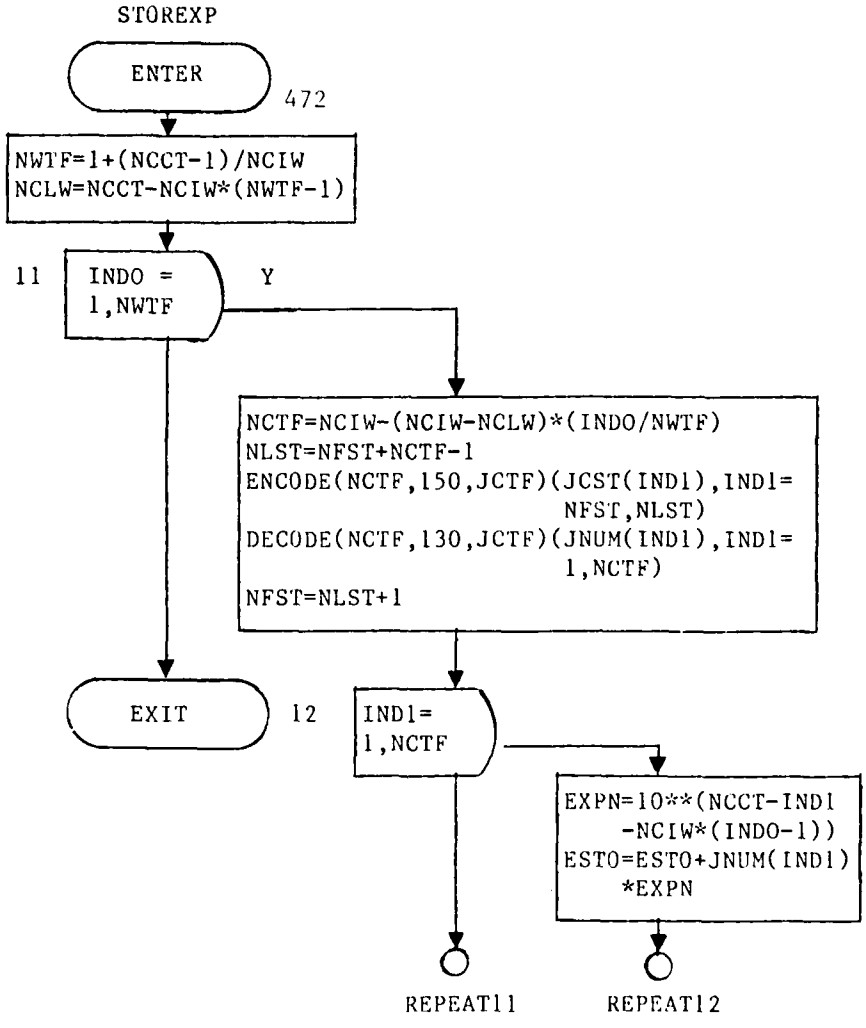


Figure 18. Storing the Exponent Portion.

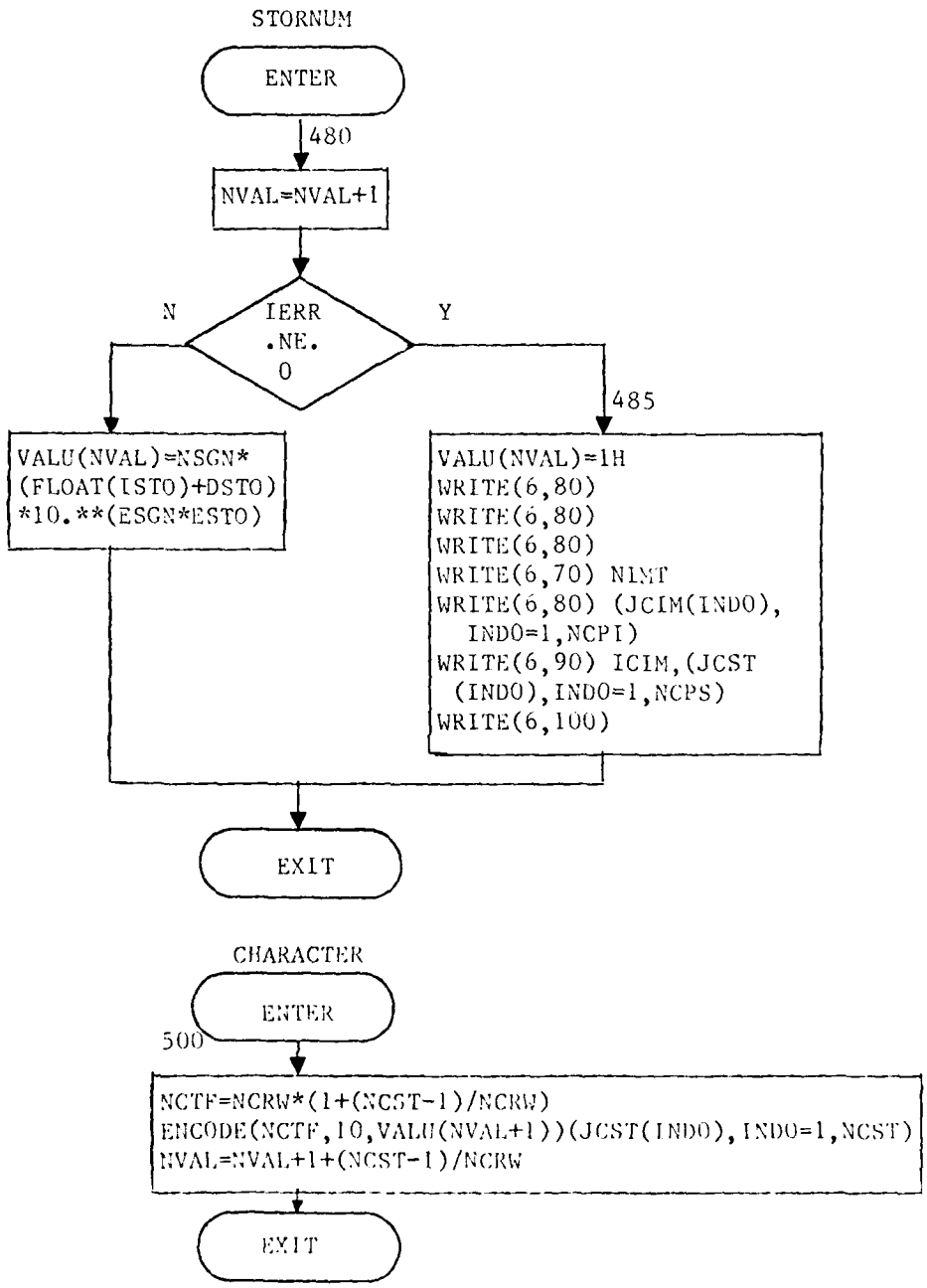


Figure 19. Storing Real Numbers and Characters.

SECTION 5

FORTRAN LISTING

This section presents FORTRAN listings for the program, TESTFF, and the subroutine, FREFORM, exactly as implemented on the DEC-10 computer.

```
      PROGRAM TESTFF
C
C     ********** VALIDATES PERFORMANCE OF SUBROUTINE FREFOR **********
C
C     THE PURPOSE OF PROGRAM TESTFF IS TO VALIDATE THE PERFORMANCE OF
C     SUBROUTINE FREFOR.  TESTFF ALLOWS DATA TO BE SENT TO FREFOR,
C     AND PRINTS THE VALUES RETURNED.  ALL OF THE VALUES RETURNED FROM
C     EACH SEQUENCE STRING ARE PRINTED ACCORDING TO THE SAME FORMAT
C     SPECIFICATON. THIS IMPOSES ONE RESTRICTION ON THE NORMAL OPERATION
C     OF SUBROUTINE FREFOR: ALL OF THE VALUES RETURNED FROM EACH
C     SEQUENCE STRING MUST BE OF THE SAME DATA TYPE.
C
C     THE TYPE MUST BE SPECIFIED BY THE USER BY A KEYWORD AT THE
C     BEGINNING OF EACH SEQUENCE STRING.  IF THE DATA TYPE IS NOT
C     SPECIFIED IN THE FIRST SEQUENCE STRING, PROGRAM EXECUTION IS
C     TERMINATED.  ONCE A DATA TYPE HAS BEEN SPECIFIED, ALL SEQUENCE
C     STRINGS ARE ASSUMED TO BE OF THAT TYPE UNTIL A NEW TYPE IS
C     SPECIFIED.  THREE DATA TYPE KEYWORDS ARE RECOGNIZED: "INTG"
C     (FOR INTEGER VALUES), "FLTG" (FOR FLOATING POINT AND FIXED POINT
C     VALUES), AND "TEXT" (FOR ASCII CHARACTER STRINGS).
C
C     NUMERICAL FORMAT SPECIFICATIONS IN PROGRAM TESTFF MATCH THE RANGE
C     OF VALUES ACCEPTED BY SUBROUTINE FREFOR.  THE TEXT FORMAT
C     SPECIFICATION IN TESTFF WILL ACCOMODATE 10 ASCII CHARACTERS, BUT
C     THE MAXIMUM LENGTH OF TEXT STRINGS RETURNED BY SUBROUTINE FREFOR
C     IS DETERMINED BY THE WORD LENGTH AND NUMBER OF BITS PER CHARACTER
C     OF THE SYSTEM BEING USED.
C
C     PROGRAM EXECUTION IS TERMINATED BY THE KEYWORD "STOP", WHICH
C     MUST OCCUR AT THE BEGINNING OF A SEQUENCE STRING.  ANY DATA
C     FOLLOWING THE WORD "STOP" WILL NOT BE PRINTED BY PROGRAM TESTFF.
C
C
C
C     ************************ FORMAT DEFINITIONS ************************
C
   10 FORMAT(/1X,24(1H*),31H PROGRAM TESTFF  -  MARCH 1981 ,25(1H*)//)
   20 FORMAT(/1X,26(1H*),28H OUTPUT FROM PROGRAM TESTFF ,26(1H*))
   30 FORMAT(/1X,24(1H*),32H END OUTPUT FROM PROGRAM TESTFF ,24(1H*))
   40 FORMAT(1X,I10)
   50 FORMAT(1X,E14.6)
   60 FORMAT(1X,A10)
   70 FORMAT(/1X,20(1H*),40H PROGRAM TESTFF  -  EXECUTION COMPLETED ,
     1       20(1H*)/)
```

```
C
C
      REAL VALU(50),VWRD(4)
C
      DATA VWRD/4HINTG,4HFLTG,4HTEXT,4HSTOP/
      DATA BWRD/1H /
C
C     INITIALIZE I/O UNIT 6 TO TERMINAL.
      OPEN(UNIT=6, DEVICE='TTY')
C
C     PRINT PROGRAM HEADER.  INITIALIZE FLAGS.
      WRITE(6,10)
      JFLG=4
      ISTP=0
C
C     ********************* LOOP UNTIL STOP REQUESTED ******************
C
   80 CALL FREFOR (VALU,NVAL)
      NFST=1
C
C     TEST FIRST VALUE FOR KEYWORD, SET JFLG.
      DO90INDX=1,4
      IF(VWRD(INDX).NE.VALU(1))GOTO90
      JFLG=INDX
      NFST=2
   90 CONTINUE
C
C     PRINT OUTPUT HEADER, THEN BRANCH TO PRINT VALUES.
      WRITE(6,20)
      GOTO(100,110,120,130)JFLG
C
C     INTEGER PROCESSING.
C
C     PRINT KEYWORD IN CHARACTER FORMAT.
  100 IF(NFST.EQ.1)GOTO105
      WRITE(6,60)VALU(1)
C
C     PRINT VALUES IN INTEGER FORMAT.
  105 IF(NVAL.LT.NFST)GOTO140
      DO109INDX=NFST,NVAL
      IF(VALU(INDX).NE.BWRD)GOTO108
      WRITE(6,60)BWRD
      GOTO109
  108 INTG=VALU(INDX)
      WRITE(6,40)INTG
  109 CONTINUE
      GOTO140
C
```

```
C     FLOATING POINT NUMBER PROCESSING.
C
C     PRINT KEYWORD IN CHARACTER FORMAT.
  110 IF(NFST.EQ.1)GOTO112
      WRITE(6,60)VALU(1)
C
C     PRINT VALUES IN FLOATING POINT FORMAT.
  112 IF(NVAL.LT.NFST)GOTO140
      DO119INDX=NFST,NVAL
      IF(VALU(INDX).NE.BWRD)GOTO118
      WRITE(6,60)BWRD
      GOTO119
  118 WRITE(6,50)VALU(INDX)
  119 CONTINUE
      GOTO140
C
C     TEXT PROCESSING.
C
C     PRINT VALUES IN CHARACTER FORMAT.
  120 WRITE(6,60)(VALU(INDX),INDX=1,NVAL)
      GOTO140
C
C     TERMINATE PROGRAM EXECUTION.
C
C     PRINT "STOP", SET ISTP.
  130 WRITE(6,60)VWRD(4)
      ISTP=1
C
C     PRINT OUTPUT FOOTING, TEST ISTP TO CONTINUE.
  140 WRITE(6,30)
      IF(ISTP.EQ.0)GOTO80
C
C
C     PRINT PROGRAM FOOTING.
      WRITE(6,70)
C
C
C
      STOP
      END
```

```
      SUBROUTINE FREFOR (VALU,NVAL)
C
C     ********* PROCESSES FREE-FORMAT DATA TO INTERNAL FORMAT **********
C
C     SUBROUTINE FREFOR IS DESIGNED TO PROCESS FREE-FORMAT DATA WHICH
C     IS OBTAINED FROM AN EXTERNAL RECORD, FOR EXAMPLE, A CARD READER
C     OR AN INTERACTIVE CONSOLE.  THE DATA MUST CONSIST OF A STRING
C     OF ASCII CHARACTERS, INCLUDING THE DELIMITERS, SPACE( ), COMMA(,),
C     ASTERISK(*), AND SLASH(/).  FREFOR INTERPRETS DATA SEQUENTIALLY
C     ACCORDING TO ITS INTERNAL FORMAT, AND STORES INTERNAL VALUES IN
C     AN ARRAY (VALU).  FOUR TYPES OF VALUES ARE RECOGNIZED.
C
C     (1) NUMERICAL CONSTANT  -  RATIONAL NUMBER CONSTANTS ARE
C     RECOGNIZED IN FIXED, FLOATING, AND EXPONENTIAL FORMATS
C     (EXAMPLES  -  143, 3.14159, 1.E-07).  VALUES MUST BE
C     SEPARATED BY SPACE OR COMMA DELIMITERS.  NONESSENTIAL
C     CHARACTERS WILL BE AUTOMATICALLY SUPPLIED DURING
C     INTERPRETATION (THUS, FOR EXAMPLE, 1E-7 AND 1.0E-07 WILL
C     BE INTERPRETED IDENTICALLY).  THE NUMBER OF CHARACTERS
C     DEVOTED TO A SINGLE CONSTANT IS LIMITED TO NCPS (DEFINED IN A
C     DATA STATEMENT).
C
C     (2) ALPHANUMERIC CONSTANT  -  ALPHANUMERIC CONSTANTS ARE
C     RECOGNIZED BY THE FIRST CHARACTER, WHICH MUST BE
C     ALPHABETICAL.  SUCCEEDING CHARACTERS MAY BE ALPHABETICAL,
C     NUMERICAL, OR ANY ASCII SPECIAL CHARACTER OTHER THAN SLASH
C     OR ASTERISK (EXAMPLES  -  A123, Z$45, START).  VALUES
C     MUST BE SEPARATED BY SPACE OR COMMA DELIMITERS.  THE
C     NUMBER OF CHARACTERS DEVOTED TO A SINGLE CONSTANT IS
C     LIMITED TO NCPS.
C
C     (3) NULL VALUE  -  A NULL VALUE IS CREATED BY RECOGNITION OF
C     SUCCESSIVE COMMAS, WITH NO INTERVENING CHARACTERS.
C     (BLANKS, OR SPACES, BETWEEN ADJACENT COMMAS WILL BE
C     IGNORED.  FOR EXAMPLE, THE CHARACTER STRING (, ,,) WILL
C     PRODUCE TWO NULL VALUES.)
C
C     (4) TEXT  -  ANY CHARACTER STRING DELIMITED BY TWO ASTERISKS
C     (AT BEGINNING AND END) WILL BE RECOGNIZED AS TEXT, AND WILL
C     BE RETAINED INTACT WITH EMBEDDED SPACES, COMMAS, AND ALL
C     ASCII CHARACTERS EXCEPT THE SLASH AND ASTERISK.  THE
C     TEXT WILL BE CODED IN AS MANY VALUES AS ARE REQUIRED TO
C     PROCESS THE STRING.
C
```

```
C     THE OPERATION OF FREFOR RELIES UPON DEFINITION OF FOUR TYPES
C     OF CHARACTER STRINGS, EACH IN GENERAL CONTAINING A DIFFERENT
C     NUMBER OF CHARACTERS.  THE RELATIONSHIPS BETWEEN THE FOUR TYPES
C     OF CHARACTER STRINGS ARE AS FOLLOWS.
C
C     (1) RECORD STRING  -  THIS IS THE COMPLETE SET OF ALL
C     SEQUENCE STRINGS TO BE PROCESSED THROUGH SEQUENTIAL
C     EXECUTIONS OF FREFOR.
C
C     (2) SEQUENCE STRING  -  WITHIN THE RECORD STRING MAY EXIST
C     ANY NUMBER OF SEQUENCE STRINGS, DELIMITED BY A SLASH
C     AT THE TRAILING END.  NO DELIMITER IS REQUIRED IF THE
C     SEQUENCE STRING IS LIMITED BY THE SPECIFIED NUMBER OF
C     RECORD IMAGES (NIPS).  OTHERWISE, THERE MAY BE ANY
C     NUMBER OF IMAGES PER SEQUENCE STRING.
C
C     (3) RECORD IMAGE  -  THIS IS THE CHARACTER STRING WHICH
C     EXISTS ON A SINGLE IMAGE.  FOR EXAMPLE, AN IMAGE MIGHT
C     CONSIST OF A FORTRAN CARD, IN WHICH CASE THE RECORD
C     IMAGE CONSISTS OF 80 CHARACTERS.
C
C     (4) PROCESS STRING  -  THIS IS AN INTERNAL CHARACTER
C     STRING USED IN FREFOR TO PROCESS DELIMITED DATA.
C     THE LENGTH OF THE STRING IS SPECIFIED BY NCPS, WHICH
C     MUST BE AN INTEGER MULTIPLE OF THE NUMBER OF ASCII
C     CHARACTERS WHICH CAN BE STORED IN A SINGLE MACHINE
C     WORD.  IN EFFECT, THE PROCESS STRING LIMITS THE
C     LENGTH OF ALPHANUMERIC CONSTANTS AND THE ACCURACY OF
C     NUMERICAL CONSTANTS.
C
C     SUBROUTINE FREFOR RECOGNIZES INCORRECTLY FORMATTED NUMBERS AND
C     PRINTS AN ERROR DIAGNOSTIC.  TWO ERRORS ARE RECOGNIZED.
C
C     (1) SYNTAX ERROR IN THE MANTISSA  -  ANY STRING (EXCLUDING TEXT
C     DELIMITED BY ASTERISKS) BEGINNING WITH A NONALPHABETICAL
C     CHARACTER IS ASSUMED TO BE A NUMERICAL CONSTANT.  IF THIS
C     CONSTANT CANNOT BE INTERPRETED, A SYNTAX ERROR RESULTS.
C     (EXAMPLES: 25.00F+02,1 +15, &STRING)
C
C     (2) SYNTAX ERROR IN THE EXPONENT  -  THE EXPONENT OF A NUMERICAL
C     CONSTANT MUST BE AN INTEGER VALUE.  (IT MAY INCLUDE AN
C     ALGEBRAIC SIGN).  IF AN EXPONENT CANNOT BE INTERPRETED A
C     SYNTAX ERROR RESULTS.  (EXAMPLES: 2E2., 2EE3, 2E-I)
C
C     THE PROCESSING OF DATA BY SUBROUTINE FREFOR OCCURS IN FOUR
C     MAJOR STEPS.
C
C     (1) A RECORD IMAGE IS READ INTO AN INTEGER ARRAY, JCIM.
```

```
C     ONE CHARACTER IS PLACED IN EACH ARRAY ELEMENT.  A NEW
C     RECORD IMAGE IS OBTAINED WHEN ALL OF THE CHARACTERS ON
C     THE PREVIOUS IMAGE HAVE BEEN INTERPRETED.
C
C     (2) EACH CHARACTER IN JCIM IS TESTED AGAINST EACH OF THE
C     FOUR STRING DELIMITERS AND THEN STORED IN AN INTEGER ARRAY,
C     JCST, WHICH CONTAINS THE PROCESS STRING.  ONE CHARACTER IS
C     STORED IN EACH ARRAY ELEMENT.  WHEN A DELIMITER IS DETECTED,
C     TESTING STOPS AND PROCESSING IS INITIATED.
C
C     (3) A TEST IS MADE TO DETERMINE WHETHER THE PROCESS STRING
C     CONTAINED IN JCST IS A NUMERICAL VALUE OR AN ALPHANUMERIC
C     CHARACTER STRING.
C
C     (4) THE CONTENTS OF JCST ARE INTERPRETED AND THEN STORED IN
C     THE ARRAY VALU BY EITHER A NUMBER DECODING ALGORITHM OR A
C     CHARACTER STRING PACKING ALGORITHM, ACCORDING TO THE RESULT
C     OF THE TEST.  THE ARRAY VALU IS RETURNED TO THE CALLING
C     PROGRAM UNIT WHEN THE ENTIRE SEQUENCE STRING HAS BEEN READ,
C     TRANSLATED AND STORED.
C
C     SUBROUTINE FREFOR CONTAINS A SYSTEM DIAGNOSTIC MODE.  IT IS
C     SELECTED BY SPECIFYING ITST=1.  THE DIAGNOSTIC MODE CAUSES A
C     LISTING OF THE SYSTEM-DEPENDENT PARAMETERS AND ASCII CHARACTER
C     VARIABLES TO BE PRINTED ON THE FIRST CALL.  SUBSEQUENTLY, ALL
C     RECORD IMAGES AND PROCESS STRINGS WILL BE PRINTED PRIOR TO
C     TRANSLATION.
C
C
C     ********************* VARIABLE DEFINITIONS **********************
C
C     DFCT  -  FACTOR USED TO COMPUTE DSTO.
C     DSTO  -  VARIABLE TO STORE DECIMAL PORTION OF NUMBER.
C     ESGN  -  SIGN OF EXPONENT.
C     ESTO  -  VARIABLE TO STORE EXPONENT.
C     EXPN  -  EXPONENTIAL MULTIPLIER FOR NUMBER PROCESSING.
C     IALP  -  FLAG FOR ALPHANUMERIC CHARACTER STRING PROCESSING.
C     ICIM  -  IMAGE POSITION OF FIRST CHARACTER IN THE PROCESS STRING.
C     ICOM  -  FLAG TO DETECT SEQUENTIAL COMMAS.
C     IERR  -  FLAG TO INDICATE ERROR IN NUMERICAL VALUE.
C     IFLD  -  FLAG TO INDICATE TEXT PROCESSING.
C     IND0  -  INDEX FOR GENERAL USE.
C     IND1  -  INDEX FOR GENERAL USE.
C     ICNC  -  FLAG TO INDICATE FIRST EXECUTION OF SUBROUTINE.
C     IPRO  -  FLAG TO INITIATE WORD PROCESSING.
C     IRET  -  FLAG TO EXIT SUBROUTINE AFTER SEQUENCE COMPLETION.
C     ISTO  -  VARIABLE TO STORE INTEGER PORTION OF NUMBER.
C     ISTP  -  FLAG TO TRANSFER CHARACTER FROM IMAGE TO PROCESS STRING
```

```
C     ITST  -  FLAG TO PRODUCE SYSTEM DIAGNOSTIC.
C     JCAE  -  ASCII CHARACTER FOR FIFTH ALPHABETICAL CHARACTER, (E).
C     JCAF  -  ASCII CHARACTER FOR FINAL ALPHABETICAL CHARACTER, (Z).
C     JCAI  -  ASCII CHARACTER FOR INITIAL ALPHABETICAL CHARACTER, (A).
C     JCAS  -  ASCII CHARACTER FOR ASTERISK SYMBOL, (*).
C     JCCO  -  ASCII CHARACTER FOR COMMA SYMBOL, (,).
C     JCDP  -  ASCII CHARACTER FOR DECIMAL POINT, (.).
C     JCHA  -  CURRENT CHARACTER IN THE RECORD IMAGE.
C     JCIM  -  ARRAY OF CHARACTERS IN THE RECORD IMAGE.
C     JCMI  -  ASCII CHARACTER FOR MINUS SYMBOL, (-).
C     JCNF  -  ASCII CHARACTER FOR FINAL DECIMAL CHARACTER, (9).
C     JCNI  -  ASCII CHARACTER FOR INITIAL DECIMAL CHARACTER, (0).
C     JCPL  -  ASCII CHARACTER FOR PLUS SYMBOL, (+).
C     JCSL  -  ASCII CHARACTER FOR SLASH SYMBOL, (/).
C     JCSP  -  ASCII CHARACTER FOR SPACE SYMBOL, ( ).
C     JCST  -  ARRAY OF CHARACTERS IN THE PROCESS STRING.
C     JCTF  -  CHARACTER TRANSFER VARIABLE IN NUMBER PROCESSING.
C     JNUM  -  ARRAY OF DECODED DIGITS IN NUMBER PROCESSING.
C     NCCT  -  NUMBER OF CHARACTERS COUNTED.
C     NCIM  -  POINTER FOR THE IMAGE STRING CHARACTER ARRAY.
C     NCIW  -  MAXIMUM NUMBER OF CHARACTERS PER INTEGER WORD.
C     NCLW  -  NUMBER OF CHARACTERS IN THE LAST WORD PROCESSED.
C     NCPI  -  MAXIMUM NUMBER OF CHARACTERS PER IMAGE STRING.
C     NCPS  -  MAXIMUM NUMBER OF CHARACTERS PER PROCESS STRING.
C     NCRW  -  MAXIMUM NUMBER OF CHARACTERS PER REAL WORD.
C     NCST  -  ACTUAL NUMBER OF CHARACTERS IN THE PROCESS STRING.
C     NCTF  -  NUMBER OF CHARACTERS TRANSFERRED BY ENCODE/DECODE.
C     NFST  -  POSITION OF FIRST CHARACTER TO BE TRANSFERRED.
C     NIMS  -  NUMBER OF IMAGES PROCESSED IN CURRENT SEQUENCE STRING.
C     NIMT  -  TOTAL NUMBER OF RECORD IMAGES PROCESSED.
C     NIPS  -  MAXIMUM NUMBER OF IMAGES PER SEQUENCE.
C     NLST  -  POSITION OF LAST CHARACTER TO BE TRANSFERRED.
C     NPRO  -  POINTER FOR THE PROCESS STRING CHARACTER ARRAY.
C     NSEQ  -  TOTAL NUMBER OF SEQUENCE STRINGS PROCESSED.
C     NSGN  -  SIGN OF NUMBER.
C     NVAL  -  POINTER FOR THE OUTPUT ARRAY VALU.
C     NWTF  -  NUMBER OF WORDS TRANSFERRED BY ENCODE/DECODE.
C     RNUM  -  ARRAY CONTAINING DECODED DIGITS IN NUMBER PROCESSING.
C     VALU  -  OUTPUT ARRAY CONTAINING PROCESSED DATA.
C
C
C
      INTEGER ESGN,ESTO,JCIM(80),JCST(21),JNUM(10)
      REAL    RNUM(10),VALU(5)
C
      DATA NIMT/0/,IGNC/0/,NSEQ/0/
C
C
```

```
C
C     ******************** SYSTEM-DEPENDENT PARAMETERS ****************
C
      DATA NCIN/5/,NCRW/5/,NCPI/80/,NCPS/20/,NIPS/1/
C
C
C
C     ********************* ASCII CHARACTER VARIABLES ******************
C
      DATA JCAE/1HE/,JCAF/1HZ/,JCAI/1HA/,JCAS/1H*/,JCCO/1H,/,JCDP/1H./,
     1     JCMI/1H-/,JCNF/1H9/,JCNI/1H0/,JCPL/1H+/,JCSL/1H//,JCSP/1H /
C
C
C
C     ************************ DIAGNOSTIC SWITCH ***********************
C
      DATA ITST/1/
C
C
C
C     ************************ FORMAT DEFINITIONS **********************
C
   10 FORMAT(80A1)
   20 FORMAT(/1X,25(1H*),30H SUBROUTINE FREFOR DIAGNOSTIC ,25(1H*))
   30 FORMAT(1X,26H ASCII CHARACTER VARIABLES/)
   40 FORMAT(1X,5H JCAE,4X,4HJCAF,4X,4HJCAI,4X,4HJCAS,
     1          4X,4HJCCO,4X,4HJCDP/
     1          4H    ,A1,5(7X,A1)/)
   50 FORMAT(1X,5H JCMI,4X,4HJCNF,4X,4HJCNI,4X,4HJCPL,
     1          4X,4HJCSL,4X,4HJCSP/
     1          4H    ,A1,5(7X,A1)/)
   55 FORMAT(1X,28H SYSTEM-DEPENDENT PARAMETERS/)
   60 FORMAT(1X,5H NCIN,4X,4HNCRW,4X,4HNCPI,4X,4HNCPS,4X,4HNIPS/
     1       3H   ,I2,4(6X,I2))
   70 FORMAT(1X,11(1H*),36H ERROR DETECTED BY SUBROUTINE FREFOR,
     1               17H IN RECORD IMAGE ,I4,1X,11(1H*))
   80 FORMAT(1X,80A1)
   90 FORMAT(1X,8H COLUMN=,I3,2X,16HPROCESS STRING= ,20A1)
  100 FORMAT(1X,80(1H*)///)
  110 FORMAT(/1X,32(1H*),16H END DIAGNOSTIC ,32(1H*))
  120 FORMAT(1X,17H SEQUENCE STRING=,I4,2X,13HRECORD IMAGE=,I4)
  130 FORMAT(10I1)
  140 FORMAT(10F1.0)
  150 FORMAT(10A1)
C
C
C
C     ************** BEGIN PROCESSING A NEW SEQUENCE STRING **************
```

```
C
C     INITIALIZE I/O UNIT 6 TO TERMINAL.
      OPEN(UNIT=6, DEVICE='TTY')
C
C     THIS SECTION FIRST TIME ONLY.
C
      IF(IONC.NE.0)GOTO200
C
C     INITIALIZE FLAGS FOR FIRST CALL.
      IONC=1
      NCIM=NCPI
C
C     DIAGNOSTIC: ASCII CHARACTER VARIABLES
C     SYSTEM DEPENDENT PARAMETERS.
      IF(ITST.EQ.0)GOTO200
      WRITE(6,20)
      WRITE(6,30)
      WRITE(6,40)JCAE,JCAF,JCAI,JCAS,JCCO,JCDP
      WRITE(6,50)JCMI,JCNF,JCNI,JCPL,JCSL,JCSP
      WRITE(6,55)
      WRITE(6,60)NCIW,NCRW,NCPI,NCPS,NIPS
      WRITE(6,110)
C
C     INITIALIZE VARIABLES FOR NEW SEQUENCE STRING.
C
  200 IF(NCIM.EQ.NCPI)NIMS=0
      IFLD=0
      IRET=0
      NVAL=0
      ICOM=0
      NSEQ=NSEQ+1
C
C
C
C     ***************** LOOP UNTIL END OF SEQUENCE STRING ***************
C
  210 IF(IRET.EQ.1)GOTO260
C
C     LOAD PROCESS STRING ARRAY ELEMENTS WITH BLANKS.
C
      DO220IND0=1,NCPS
  220 JCST(IND0)=1H
C
C     INITIALIZE VARIABLES FOR NEW PROCESS STRING.
C
      NCST=0
      ILP=0
      IPP0=0
```

```
C
C
C
C     ***************** LOOP UNTIL END OF PROCESS STRING *****************
C
  310 IF(IPRO.EQ.1)GOTO230
C
C     GET A CHARACTER FROM THE RECORD IMAGE.
C
      IF(NCIM.NE.NCPI)GOTO330
      IF(NIMS.LT.NIPS)GOTO320
C
C     MAXIMUM IMAGE NUMBER EXCEEDED. TERMINATE SEQUENCE.
      JCIM(NCPI)=JCSL
      NCIM=NCPI-1
      GOTO330
C
C     END OF IMAGE ENCOUNTERED. READ A NEW RECORD IMAGE.
  320 NCIM=0
      NIMS=NIMS+1
      NIMT=NIMT+1
      READ(5,10)(JCIM(IND0),IND0=1,NCPI)
C
C     DIAGNOSTIC: PRINT RECORD IMAGE.
      IF(ITST.EQ.0)GOTO330
      WRITE(6,20)
      WRITE(6,120)NSEQ,NIMT
      WRITE(6,80)
      WRITE(6,80)(JCIM(IND0),IND0=1,NCPI)
C
C     INITIALIZE VARIABLES FOR CHARACTER COMPARISON TESTS.
C
  330 IPRO=1
      ISTR=0
      NCIM=NCIM+1
      JCHA=JCIM(NCIM)
C
C     TEST CHARACTER AGAINST DELIMITERS WHICH TERMINATE PROCESS STRING.
C
C     TEST FOR SLASH DELIMITER, (/).
      IF(JCHA.NE.JCSL)GOTO340
      IPET=1
      GOTO310
C
C     TEST FOR ASTERISK DELIMITER, (*).
  340 IF(JCHA.NE.JCAS)GOTO350
      IF(IPLS.NE.1)GOTO345
      IPLS=0
```

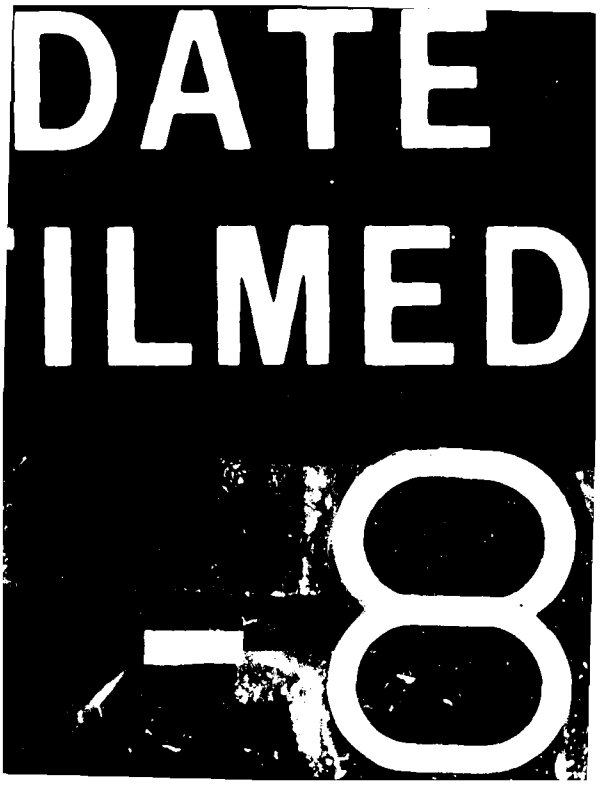
```
      GOTO310
  345 IFLD=1
      GOTO310
C
C     TEST FOR COMMA DELIMITER, (,).
  350 IF(.NOT.(JCHA.EQ.JCCO.AND.IFLD.EQ.0))GOTO360
      IF(ICOM.NE.1)GOTO355
      IALP=1
      ISTR=1
      JCHA=1H
      GOTO370
  355 ICOM=1
      GOTO310
C
C     TEST FOR SPACE DELIMITER, ( ).
  360 IF(JCHA.EQ.JCSP.AND.IFLD.EQ.0)GOTO310
      IPRO=0
      IALP=IFLD
      ISTR=1
      ICOM=0
C
C     STORE CHARACTER IN PROCESS STRING.
  370 IF(ISTR.NE.1)GOTO380
      NCST=NCST+1
      JCST(NCST)=JCHA
C
C     TEST NUMBER OF CHARACTERS STORED IN PROCESS STRING.
  380 IF(NCST.LT.NCPS)GOTO310
      IPRO=1
      GOTO310
C
C
C
C     ****************** TRANSLATE THE PROCESS STRING ******************
C
C     TEST FOR EMPTY PROCESS STRING, BYPASS TRANSLATION.
  230 IF(NCST.EQ.0)GOTO210
C
C     DIAGNOSTIC: PRINT PROCESS STRING.
      IF(ITST.EQ.0)GOTO240
      ICIM=NCIM-NCST
      WRITE(6,90)ICIM,(JCST(IND0),IND0=1,NCPS)
C
C     TEST FOR NUMBER PROCESSING OR CHARACTER STRING PROCESSING.
  240 IF((IALP.EQ.1).OR.(JCST(1).LE.JCAF.AND.JCST(1).GE.JCAI))
     1GOTO500
C
```

```
C
C     ********************** NUMBER PROCESSING **********************
C
C
C     INITIALIZE VARIABLES.
  400 IERR=0
      NFST=1
      NPRO=1
      ICIM=NCIM-NCST
      ISTO=0
      ESTO=0
      DSTO=0.
      DFCT=1.
      ESGN=1
      NSGN=1
C
C     PROCESS INTEGER PORTION.
C
C     TEST FOR MINUS SIGN.
      IF(JCST(NPRO).NE.JCMI)GOTO410
      NPRO=NPRO+1
      NSGN=-1
      NFST=NPRO
      GOTO420
C
C     TEST FOR PLUS SIGN.
  410 IF(JCST(NPRO).NE.JCPL)GOTO420
      NPRO=NPRO+1
      NFST=NPRO
C
C     DETERMINE LENGTH OF INTEGER PORTION.
  420 IF(.NOT.(JCST(NPRO).LE.JCNF.AND.JCST(NPRO).GE.JCNI))GOTO423
      NPRO=NPRO+1
      GOTO420
C
C     STORE INTEGER PORTION.
  423 NCCT=NPRO-NFST
      IF(NCCT.EQ.0)GOTO430
      NWTF=1+(NCCT-1)/NCIW
      NCLW=NCCT-NCIW*(NWTF-1)
      DO424IND0=1,NWTF
      NCTF=NCIW-(NCIW-NCLW)*(IND0/NWTF)
      NLST=NFST+NCTF-1
      ENCODE(NCTF,150,JCTF)(JCST(IND1),IND1=NFST,NLST)
      DECODE(NCTF,130,JCTF)(JNUM(IND1),IND1=1,NCTF)
      NFST=NLST+1
      DO424IND1=1,NCTF
      EXPN=10**(NCCT-IND1-NCIW*(IND0-1))
```

```
      ISTO=ISTO+JNUM(IND1)*EXPN
  424 CONTINUE
C
C     PROCESS DECIMAL PORTION.
C
C     TEST FOR DECIMAL POINT.
  430 IF(JCST(NPRO).NE.JCDP)GOTO450
      NPRO=NPRO+1
      NFST=NPRO
C
C     DETERMINE LENGTH OF DECIMAL PORTION.
  440 IF(.NOT.(JCST(NPRO).LE.JCNF.AND.JCST(NPRO).GE.JCNI))GOTO442
      NPRO=NPRO+1
      GOTO440
C
C     STORE DECIMAL PORTION.
  442 NCCT=NPRO-NFST
      IF(NCCT.EQ.0)GOTO450
      NWTF=1+(NCCT-1)/NCIW
      NCLW=NCCT-NCIW*(NWTF-1)
      DO444IND0=1,NWTF
      NCTF=NCIW-(NCIW-NCLW)*(IND0/NWTF)
      NLST=NFST+NCTF-1
      ENCODE(NCTF,150,JCTF)(JCST(IND1),IND1=NFST,NLST)
      DECODE(NCTF,140,JCTF)(RNUM(IND1),IND1=1,NCTF)
      NFST=NLST+1
      DO444IND1=1,NCTF
      DFCT=DFCT*.1
      DSTO=DSTO+DFCT*RNUM(IND1)
  444 CONTINUE
C
C     TEST FOR SYNTAX ERROR IN MANTISSA.
  450 IF(NPRO.GT.NCST)GOTO480
      IF(JCST(NPRO).EQ.JCAE)GOTO460
      IERR=1
      GOTO480
C
C     PROCESS EXPONENT.
C
  460 NPRO=NPRO+1
      NFST=NPRO
C
C     TEST FOR MINUS SIGN.
      IF(JCST(NPRO).NE.JCMI)GOTO465
      ESGN=-1
      NPRO=NPRO+1
      NFST=NPRO
      GOTO470
```

```
C
C     TEST FOR PLUS SIGN.
  465 IF(JCST(NPRO).NE.JCPL)GOTO470
      NPRO=NPRO+1
      NFST=NPRO
C
C     DETERMINE LENGTH OF EXPONENT.
  470 IF(.NOT.(JCST(NPRO).LE.JCNF.AND.JCST(NPRO).GE.JCNI))GOTO472
      NPRO=NPRO+1
      GOTO470
C
C     STORE EXPONENT.
  472 NCCT=NPRO-NFST
      IF(NCCT.EQ.0)GOTO475
      NWTF=1+(NCCT-1)/NCIW
      NCLW=NCCT-NCIW*(NWTF-1)
      DO474IND0=1,NWTF
      NCTF=NCIW-(NCIW-NCLW)*(IND0/NWTF)
      NLST=NFST+NCTF-1
      ENCODE(NCTF,150,JCTF)(JCST(IND1),IND1=NFST,NLST)
      DECODE(NCTF,130,JCTF)(JNUM(IND1),IND1=1,NCTF)
      NFST=NLST+1
      DO474IND1=1,NCTF
      EXPN=10**(NCCT-IND1-NCIW*(IND0-1))
      ESTO=ESTO+JNUM(IND1)*EXPN
  474 CONTINUE
C
C     TEST FOR SYNTAX ERROR IN EXPONENT.
  475 IF(NPRO.GT.NCST)GOTO480
      IERR=2
C
C     COMPUTE NUMBER AND STORE IN VALU ARRAY.
C
  480 NVAL=NVAL+1
      IF(IERR.NE.0)GOTO485
C
C     STORE VALID NUMBER.
      VALU(NVAL)=NSGN*(FLOAT(ISTO)+DSTO)*10.**(ESGN*ESTO)
C
C     RETURN TO BEGINNING OF PROCESS STRING LOOP.
      GOTO210
C
C     PROCESS ERROR.
C
C     STORE BLANK.
  485 VALU(NVAL)=1H
C
C     ERROR MESSAGE
```

```
      WRITE(6,80)
      WRITE(6,80)
      WRITE(6,80)
      WRITE(6,70)NIMT
      WRITE(6,80)(JCIM(IND0),IND0=1,NCPI)
      WRITE(6,90)ICIM,(JCST(IND0),IND0=1,NCPS)
      WRITE(6,100)
C
C     RETURN TO BEGINNING OF PROCESS STRING LOOP.
      GOTO210
C
C
C
C     ******************** CHARACTER STRING PROCESSING ********************
C
C
C     PLACE CHARACTER STRING IN VALU USING ENCODE.
  500 NCTF=NCRW*(1+(NCST-1)/NCRW)
      ENCODE(NCTF,10,VALU(NVAL+1))(JCST(IND0),IND0=1,NCST)
C
C     ADJUST NVAL.
      NVAL=NVAL+1+(NCST-1)/NCRW
C
C     RETURN TO BEGINNING OF PROCESS STRING LOOP.
      GOTO210
C
C
C
C     END OF SEQUENCE STRING, RETURN TO CALLING PROGRAM UNIT.
C
  260 IF(ITST.EQ.0)GOTO270
      WRITE(6,110)
      WRITE(6,80)
C
C
C
  270 RETURN
      END
```


DATE
ILMED
-8