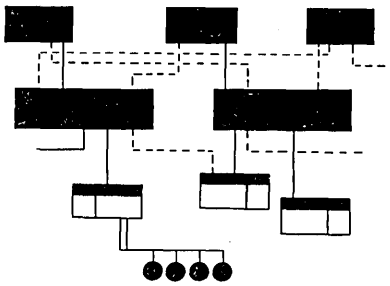
GENERAL ELECTRIC
COMPUTERS

GE-625/635 GIFT
General Internal
FORTRAN Translator

SYSTEM
SUPPORT
INFORMATION



ABSTRACT

This document describes the Compatibles/600 General Internal FORTRAN Translator (GIFT). Part I discusses the differences between FORTRAN II and the GIFT translation to FORTRAN IV, and significant features of GIFT. Part II discusses requirements for FORTRAN II programs to be translated and how to use GIFT, including deck setup and program options. Part III provides technical specifications.

GENERAL ELECTRIC

CPB-1077A

INSTRUCTIONS FOR RUNNING GIFT

At the T.I.P. installation, they have their own method of running GIFT. The following GECOS Control Cards will be required for each program:

Column 1	Column 8	Column 16
$	SNUMB	(A sequential number scheme of 5 Digit Intergers)
$	IDENT	Prog. No., S. Elzam, GIFT, Comments (5 numeric digits)
$	USE	.LHSF
$	ENTRY	.LHSF
$	EXECUTE	
$	TAPE	H*,A1S,,,,GIFT*
$	TAPE	02,A2R
$	TAPE	03,B2R
$	TAPE	08,B2R
$	SYSOUT	43
$	INCODE	IBMF
SYMBOLIC FORTRAN II DECK		
$	ENDJOB	

Column 1-6

***EOF

If any questions exist on the use of GIFT, please call S. Elzam or J. Seefranz at G.E. - PL1-1311, x-3266
Allied - HA2-7300, x-3477

12/7/65

GE-625/635

GIFT

General Internal

FORTRAN Translator

September 1964

Rev. June 1965

GENERAL ELECTRIC

COMPUTER DEPARTMENT

PREFACE

This manual describes the General Internal FORTRAN Translator (GIFT) for use with the Compatibles/600 computers. This program automatically translates FORTRAN II source programs into FORTRAN IV language. The level of presentation assumes reader familiarity with FORTRAN.

For additional information concerning FORTRAN IV for the Compatibles/600, refer to the GE-625/635 FORTRAN IV Reference Manual, CPB-1006B.

GIFT is the General Electric Computer Department version of the SHARE Internal FORTRAN Translator (SIFT), described in the SHARE Internal FORTRAN Translator Users Manual, SHARE Distribution #1367 HS SIFT (PA), prepared by members of the SHARE FORTRAN Project, September, 1962.

This manual supersedes CPB-1077. Part III and the Appendix have been added to this edition and other changes are indicated by a line in the margin.

Comments on this publication may be addressed to Technical Publications, Computer Department, General Electric Company, P. O. Box 2961, Phoenix, Arizona, 85002.

CONTENTS

Page

APPENDIX

1. PROGRAM FEATURES

The General Internal FORTRAN Translator (GIFT) automatically translates a FORTRAN II source program into a FORTRAN IV source program. It eliminates certain incompatibilities between the two languages and modifies certain statements to conform with recommended or alternate formats. These changes brought about by the program are discussed in this chapter.

TRANSLATIONS EFFECTED BY GIFT

F Card

The F card, used to specify function or subroutine names which are arguments to other functions or subroutines, is replaced by an EXTERNAL Type statement. For example:

FORTRAN II	GIFT Translation
F SIN, COS, FUNC	EXTERNAL SIN, COS, FUNC

Function Names

- Elimination of Terminal F. Built-in, library, and arithmetic statement functions are no longer identified by a terminal F. GIFT therefore removes the terminal F from every function name. For example:

FORTRAN II	GIFT Translation
X = Y + SINF (A) *ADFUNF (Z**2)	X = Y + SIN (A) *ADFUN (Z**2)

- Fixed-Point Function Names. Fixed-point function names normally begin with I, J, K, L, M, or N rather than X. GIFT therefore manufactures INTEGER and REAL Type statements where appropriate. For example:

FORTRAN II	GIFT Translation
X = LASTF (Y)	REAL LAST X = LAST (Y)
I = XTRAF (1, K)	INTEGER XTRA I = XTRA (1, K)

- Changed Function Names. Many of the function names have been changed. GIFT replaces each of the FORTRAN II names listed below by the FORTRAN IV equivalent shown.

FORTRAN II Name	GIFT Translation	FORTRAN II Name	GIFT Translation
XABSF	IABS	INTF	AINT
XINTF	INT	MODF	AMOD
XMODF	MOD	XMAX1F	MAX1
XFIXF	IFIX	XSIGNF	ISIGN
XDIMF	IDIM	MAX0F	AMAX0
MAX1F	AMAX1	MIN0F	AMIN0
MIN1F	AMIN1	XMIN0F	MIN0
XMIN1F	MIN1	XMAX0F	MAX0
LOGF	ALOG		

- Other New Function Names. Many other new function names have been created. Every time GIFT encounters a variable whose name is the same as one of the new FORTRAN IV function names, it creates a name not used in the program being translated. It uniformly replaces all occurrences of the conflicting name with this created name, which is called an "insert variable." The insert variable is chosen so that its first letter determines the correct type for replacement.

The list below includes the new FORTRAN IV function names recognized by GIFT:

SLITE	SLITET	SSWTCH	OVERFL
DVCHK	IABS	AINT	INT
AMOD	MOD	AMAX0	AMAX1
AMIN0	AMIN1	MIN0	MIN1
MAX0	MAX1	IFIX	ISIGN
IDIM	ALOG	CABS	STORE
CSIGN	PART	CEXP	CLOG
CSIN	CCOS	CSQRT	OR
AND	BOOL	COMPL	COM

However, names of subroutines and functions which conflict with new FORTRAN IV names cannot be merely replaced by insert variables. GIFT recognizes two degrees of severity of conflict:

1. Nonfatal example:

```
CALL ALOG (A, B)
CALL EXIT
END
```

ALOG conflicts with the FORTRAN IV subroutine of the same name, but no usage of ALOG is constructed by the translator.

2. Fatal example:

```
CALL ALOG (A, B)
X = LOGF (A)
    becomes
CALL ALOG (A, B)
X = ALOG (A)
```

This is a definite and fatal error. The program must be corrected by hand.

Boolean Statements

- Arithmetic, IF, and CALL Statements. Arithmetic, IF, and CALL statements with a B in column 1 are modified so that the Boolean operators +, *, and - are replaced by OR, AND, and COMPL functions, respectively. For example:

FORTRAN II	GIFT Translation
B A = C + D	A = OR (C, D)
B F = G* (-H)	F = AND (G, COMPL (H))
B IF (A*B) 10, 20, 10	IF (BOOL (AND (A, B)))10, 20, 10

The call to BOOL is inserted to provide a zero-vs-nonzero test of all 36 bits of the result. In the FORTRAN II program, a transfer of control to the negative branch of the IF statement is impossible.

- Octal Constants. Each octal constant is replaced by an insert variable name; and a DATA statement is manufactured, specifying the variable and its corresponding value. Thus:

FORTRAN II	GIFT Translation
B P = R*77777	DATA Q000CT/O77777/ P = AND (R, Q000CT)

- Return Statements. Since the AND, OR, and COMPL functions obtain arguments from and return results to the algebraic rather than the logical accumulator, the Boolean RETURN statement is no longer meaningful. Thus, for example:

FORTRAN II	GIFT Translation
B RETURN	RETURN

Double-Precision and Complex Statements

- Alteration of Variable and Function Names. In FORTRAN IV, variable and function names rather than statements are specified as double-precision or complex. GIFT includes in an appropriate Type statement every variable and function name which appears in a statement containing a D or an I in column 1 and removes the D or I. For example:

FORTRAN II	GIFT Translation
I DIMENSION XI (2, 3), YI (5, 5, 5)	DIMENSION XI (2, 3), YI (5, 5, 5)
D ALPHA = BETA * GAMMA	DOUBLE PRECISION ALPHA, BETA, GAMMA
	COMPLEX XI, YI
	ALPHA = BETA * GAMMA

In FORTRAN IV, every function name in a double-precision or complex statement is prefixed by a D in double-precision statements and a C in complex statements. In addition, the terminal F is removed as explained on page 1; and the name is included in a Type statement, as explained above. For example:

FORTRAN II	GIFT Translation
I Y = SINF (X)	COMPLEX Y, CSIN, X Y = CSIN (X)

- References to Double-Precision and Complex Variables. GIFT handles references to double-precision and complex variables in the following manner:

1. In statements not preceded by D or I--If a reference to a double-precision or complex variable name appears in an arithmetic expression in a statement without a D or an I in column 1, that variable becomes an argument to the function SNGL (or function REAL) if it is unsubscripted or to the function PART if it is subscripted. The PART function also contains as arguments the name of the array and its length. At execution time, then, the desired part of the double-precision or complex pair is returned to the object program.

   If, however, the reference is an unsubscripted variable name used as an explicit argument to a function reference or a CALL statement, it is passed on unaltered. For example:

	FORTRAN II	GIFT Translation
I	DIMENSION A (5, 5)	DIMENSION A (5, 5)
D	B = C*D	COMPLEX A
	BB = A (I, J)**2	DOUBLE PRECISION B, C, D
	IF (B) 5, 5, 10	BB = PART (A, A (I, J), 25)**2
5	CALL XYZ (A, B, C**2)	IF (SNGL (B)) 5, 5, 10
10	AB = C*D - A - SINF (A)	5 CALL XYZ (A, B, SNGL (C)**2)
		10 AB = SNGL (C) *SNGL (D) - REAL (A) -SIN (A)

In construction of the PART function in the preceding example, if J ≤ 5, the arithmetic statement is equivalent to:

```
BB = REAL (A (I, J) )**2
```

But if J > 5, it is equivalent to:

```
BB = IMAG (A(I, J) )**2
```

2. On left side of arithmetic statement--If a reference similar to that described above occurs on the left side of an arithmetic statement, then the statement specifies that a quantity be stored in one part of the number pair. To accomplish this operation in FORTRAN IV, GIFT replaces the arithmetic statement by a call to the STORE subroutine.

   The first three arguments are the same as in a call to PART, and the fourth argument is the expression whose value is to be stored. For example:

FORTRAN II	GIFT Translation
I DIMENSION A (5, 5, 5)	DIMENSION A (5, 5, 5)
A (I, J, K) = B**2	COMPLEX A
	CALL STORE (A, A (I, J, K), 125, B**2)

- Floating-Point Constants. In double-precision statements all floating-point constants are made double-precision by the use of a letter D followed by an exponent. Thus, for example:

FORTRAN II	GIFT Translation
D A = 1567.0 E 15 * X+2.4	DOUBLE PRECISION A, X
	A = 1567.0 D 15*X+2.4D0

- Constants in Single-Precision Statements. In single-precision statements, constants containing more than nine significant digits are truncated to nine significant digits so as not to be taken as double-precision in FORTRAN IV. For example:

FORTRAN II	GIFT Translation
X = 1234567895.	X = .123456789 E + 10

- Floating-Point Constants Ended with Non-Numerics. All floating-point constants whose last character is a non-numeric have a zero appended when they are translated. For example:

FORTRAN II	GIFT Translation
X = 2.4E	X = 2.4E0
D Y = 2.4E	Y = 2.4D0

COMMON STATEMENTS

- Manufactured COMMON Statements. Because EQUIVALENCE statements no longer affect COMMON storage, GIFT manufactures a COMMON statement, inserting dimensioned artificial variables where necessary, to preserve the order of COMMON storage. For example:

FORTRAN II	GIFT Translation
COMMON A, B, C, D EQUIVALENCE (B (2), E) DIMENSION B(2)	COMMON B, Q000CM (1), A, C, D EQUIVALENCE (B (2), E) DIMENSION B(2)

Note that the manufactured COMMON statement includes dimension information (but only for the artificially inserted variables), as allowed in FORTRAN IV.

- Addition of Undimensioned Artificial Variables. If in COMMON storage a double-precision or complex COMMON variable would begin in an odd-numbered memory location, an undimensioned artificial variable is inserted in the COMMON statement to cause the variable to begin in an even-numbered location. For example:

FORTRAN II	GIFT Translation
COMMON R, S, T	DOUBLE PRECISION S, X, Y
D S = X + Y	COMMON R, Q001CM, S, T S = X + Y

A similar insertion is made, if possible, when a double-precision or complex variable related to a COMMON variable through an EQUIVALENCE statement would begin in an odd-numbered location. However, certain cases of this type cannot be translated. See page 14 for further details.

- Modification of EQUIVALENCE Statements. An EQUIVALENCE Statement involving a double-precision or complex array must be modified, since subscripts in such equivalences are interpreted differently in FORTRAN II and FORTRAN IV. For example, in FORTRAN II the following EQUIVALENCE statement specifies that E shares storage with the imaginary part of A, assuming that A is not dimensioned.

```
      EQUIVALENCE (A (2), E)
C     A = B**2
```

In FORTRAN IV, an EQUIVALENCE reference to A (2) is interpreted as referring to the first word of the second number pair. To achieve the same storage allocation as the FORTRAN II program, GIFT (1) creates an insert variable, (2) generates an EQUIVALENCE between the insert variable and the double-precision or complex variable, and (3) replaces each occurrence of the latter variable in an EQUIVALENCE statement by the insert variable. Occurrences of the variable outside of EQUIVALENCE statements are not altered. If, for example, the insert variable Q000EQ is chosen to replace A, GIFT translates the last example as follows:

```
      DOUBLE PRECISION A, B
      EQUIVALENCE (A, Q000EQ)
      EQUIVALENCE (Q000EQ (2), E)
      A = B**2
```

- Treatment of Repeated Variables. In FORTRAN II, a variable can appear more than once in a COMMON statement. This situation is illegal in FORTRAN IV, and GIFT always reconstructs such COMMON statements. For example:

FORTRAN II	GIFT Translation
COMMON A, B, C, A, D, E	COMMON A, B, C, D, E

Arithmetic Statement Functions

In FORTRAN II, argument variables to arithmetic statement functions are dummies. No conflict arises if the same variable names appear as arguments to arithmetic statement functions and also as genuine variable names in the body of the program. In FORTRAN IV, a conflict may arise involving Type statements. Consider the following FORTRAN II example:

```
D   DIMENSION X (10, 10)
    FIRSTF (X, I) = A + X**I
```

The variable X in the DIMENSION statement is a double-precision variable. The variable X in the arithmetic statement function definition is a single-precision dummy. However, in FORTRAN IV any Type statement applied to X holds wherever X is used, even in a dummy argument. GIFT anticipates this possible conflict by replacing all dummy arguments in arithmetic statement functions with insert variables. Thus, if the insert variable Q000FL is chosen to replace X, and K000FX is chosen to replace I, GIFT translates the last example as follows:

```
DIMENSION X (10, 10)
DOUBLE PRECISION X
FIRST (Q000FL, K000FX) = A+Q000FL**K000FX
```

DIMENSION Statements

To ensure that every array is mentioned in a DIMENSION statement before it appears in an executable statement, every DIMENSION statement is relocated near the beginning of the program.

Hollerith Literals

Each Hollerith literal in an arithmetic or IF statement is replaced by an artificial variable name; and a DATA statement is manufactured, specifying the variable name and its corresponding value. For example (where b denotes a blank):

FORTRAN II	GIFT Translation
WORD = 4HbEND	DATA Q000HL/6HbENDbb/ WORD = Q000HL

Hollerith arguments in CALL statements are not altered.

Implicit Multiplication

Certain cases of implicit multiplication are allowed in FORTRAN II. GIFT inserts an * wherever necessary to remove the ambiguity in accordance with the rules for FORTRAN IV. For example:

FORTRAN II

```
X = (A + B) C + 3. D - (E + F) (G + H) 5. (U-V)
```

GIFT Translation

```
X = (A + B) * C + 3. *D - (E + F) * (G + H) *5. * (U-V)
```

Variables with Too Few Subscripts

The FORTRAN II compiler accepts singly-subscripted references to multiple-dimensioned variables. When GIFT encounters such a reference in statements other than EQUIVALENCE statements, it appends sufficient trailing subscripts (with value 1) to make the number of subscripts in the reference equal to the number of dimensions in the DIMENSION statement. Trailing subscripts are also added to unsubscripted references to dimensioned variables in arithmetic, IF, and CALL statements, except when they are explicit arguments to functions or to CALL statements. For example:

FORTRAN II

```
DIMENSION X (10, 10, 5), Y (10, 10), Z (10, 10)
EQUIVALENCE (X (3), XX)
X (I) = Y (I) *Z-SINF (Y) - COSF (Y (3)) *SINF (Y**2)
CALL XYZ (X, Y (3), Z**2)
```

GIFT Translation

```
DIMENSION X (10, 10, 5), Y (10, 10), Z (10, 10)
EQUIVALENCE (X (3), XX)
X (I, 1, 1) = Y (I, 1) *Z (1, 1) - SIN (Y) - COS (Y (3, 1)) *SIN(Y**2)
CALL XYZ (X, Y (3, 1), Z (1, 1)**2)
```

DO Statement Indexing Parameter

Several incompatibilities exist between FORTRAN II and FORTRAN IV in regard to the indexing parameter of a DO statement or an implied DO in input/output lists. These inconsistences, which cannot be resolved by GIFT and must be corrected by hand, are listed below:

1. In FORTRAN II, the name of the indexing parameter within a DO loop may be the same as the name of a dimensioned fixed-point variable used outside the DO loop.

FORTRAN IV, however, objects to this and issues a level 3 error message which suppresses assembly. The following is an example of this incompatibility:

```
   DIMENSION J (10, 2)
   .
   .
   .
   .
   DO 10 J = 5, 100, 5
   LL = J
10 PRINT 35, LL
   .
   .
   .
   .
   END
```

2. In FORTRAN II, if the program contains a storage location having the same fixed-point variable name as the indexing parameter of a DO (or an implied DO) statement, the storage location is not affected after a normal exit unless the indexing parameter was used as a variable within the DO range or it was used as a subscript in combination with a relative constant whose value changes within the range of the DO statement. In FORTRAN IV, however, the storage location is always updated and (after a normal exit from the DO range) contains the highest value of the indexing parameter.

MODIFICATIONS FOR FORMAT CONFORMITY

READ/WRITE Statement

READ/WRITE statements are modified according to the FORTRAN IV specification. For example:

FORTRAN II	GIFT Translation
READ INPUT TAPE IN, 100, A, B, C	READ (IN, 100) A, B, C
WRITE TAPE IOUT, D, E, F	WRITE (IOUT) D, E, F

At the option of the user, references to file numbers can be replaced either by other variable names or by file constants by means of an *REPLACE card. (p. 18) If the replacements are variables, a DATA statement is constructed assigning the proper values to the variables, and INTEGER Type statements are manufactured for these variable names if they begin with A through H or with O through Z. For example:

FORTRAN II	GIFT Translation
*REPLACE (6, AOUT), (3, 5) WRITE OUTPUT TAPE 6, 120, G, H, I REWIND 3	INTEGER AOUT DATA AOUT/6/ WRITE (AOUT, 120)G, H, I REWIND 5

READ DRUM/WRITE DRUM statements are flagged as errors and are not translated by GIFT.

Double-precision and complex variables included in the list of input/output statements are flagged by GIFT, but the statements are not considered in error and are translated.

Statements Concerning Internal Switches

Each statement which sets or tests an internal switch (sense switch, overflow indicator, etc.) is translated into a call to an appropriate subroutine. This is followed, in the case of tests, by a computed GO TO statement. For example, assume an insert variable with the name K000FX. Then the following set of FORTRAN II statements is translated as shown:

FORTRAN II

```
100   SENSE LIGHT 3
110   IF (SENSE LIGHT 2) 111, 112
120   IF (SENSE SWITCH 5) 121, 122
130   IF DIVIDE CHECK 131, 132
140   IF ACCUMULATOR OVERFLOW 141, 142
150   IF QUOTIENT OVERFLOW 151, 152
```

GIFT Translation

```
100   CALL SLITE (3)
110   CALL SLITET (2, K000FX)
      GO TO (111, 112), K000FX
120   CALL SSWTCH (5, K000FX)
      GO TO (121,122), K000FX
130   CALL DVCHK (K000FX)
      GO TO (131,132), K000FX
140   CALL OVERFL (K000FX)
      GO TO (141,142), K000FX
150   CALL OVERFL (K000FX)
      GO TO (151,152), K000FX
```

FORMAT Generator

GIFT recognizes sets of FORMAT Generator cards and changes the first card of the set to comply with GE-600 Series FORTRAN IV requirements.

RIT and WOT

GIFT recognizes RIT and WOT as equivalent to READ INPUT TAPE and WRITE OUTPUT TAPE, respectively, and translates them as explained on page 9.

Characters in Column 1

Except on FORMAT Generator cards, nonnumeric characters other than *, $, and C are removed from column 1. Every * or $ is replaced by a C, and all comment cards remain unchanged.

FAP Programs

Programs preceded by a *FAP card are ignored by the translator. The end of the FAP program is recognized when the number of FAP END cards encountered exceeds the number of MACRO and MOP cards encountered. (ENDM is recognized as an END card for this purpose.) Inclusion of FAP UPDATE decks in GIFT input causes errors if the UPDATE decks do not contain the proper number of END cards.

$ DATA Card

GIFT recognizes a $ DATA card and considers all subsequent cards as data until it encounters either a FORTRAN or a GECOS control card ($ in column 1). Data cards are listed as part of the input programs but are ignored by the translator.

2. PROGRAM REQUIREMENTS AND USAGE

GIFT is a program written in FORTRAN IV and macro assembly language that is designed to run under control of standard GE-600 Series software. The subprograms to be translated are considered data and are placed behind a $ GIFT control card in the deck. When the job is run, output is written on the SYSOUT file.

REQUIREMENTS

Characteristics of Subprograms to be Translated

A subprogram that is to be translated must fulfill the following requirements:

1. It must be programmed in the FORTRAN II language and be capable of being compiled successfully by the IBM 7000 Series FORTRAN II Compiler. GIFT does little diagnostic checking, and an incorrect translation may result for a program which cannot be compiled by the FORTRAN II system.

2. It must terminate in the normal manner with the FORTRAN END card.

3. If the first card in the input source deck has a C in column 1, then the contents of columns 2-5 of this card are punched in columns 73-76 of the output source program with zeros completing the field when necessary. If this comment card is missing or card columns 2-5 are blank, the subroutine name (or function name) will be used for a subprogram and a zero field used for the main program. The output source deck is sequenced by tens in columns 77-80, recycling to zero when the field reaches maximum value.

   All labeling and sequencing in the input source program is replaced in the above manner.

   Any $ control cards included before the first statement of the FORTRAN II deck, however, are labeled with zeros in columns 73-79 and sequenced by 2 in column 80, starting with 1-9 and continuing from A-Z.

Table Sizes

The following table sizes are allowed:

1. GIFT allows for a total of 4000 COMMON, DIMENSION, EQUIVALENCE, double-precision, and complex variables in a single program or subprogram. Currently FORTRAN II allows 2400 COMMON, 1160 DIMENSION, 6000 EQUIVALENCE (that is, literal appearances), and 550 double-precision and complex variables--a total of 10110. Thus, GIFT can handle 6110 fewer variables than the maximum case.

2. GIFT allows for replacement of a maximum of 20 different variable names in any one program under the REPLACE option.

3. GIFT provides for a maximum of 100 library, built-in, and arithmetic statement function names beginning with X or letters from I through N.

4. GIFT allows a maximum of 100 function names appearing on F cards.

Chain Jobs

GIFT flags the appearance of a CALL CHAIN statement by inserting a comment card containing asterisks in columns 2-72. IN THE UNUSED area of

Double-Precision and Complex EQUIVALENCE Variables

- Odd-Even Memory Storage Inconsistencies. Because the most significant part of a double-precision variable--or the real part of a complex variable--must be stored in an even-numbered memory location, certain EQUIVALENCE specifications may result in odd-even inconsistencies. For example, consider the following:

```
      EQUIVALENCE (A, B), (A (6), C)
D     G = B + C
```

  Clearly, the more significant parts of B and C cannot both begin in an even-numbered memory location. In such cases, GIFT prints out a diagnostic message. The inconsistency must be resolved manually and the revised program retranslated.

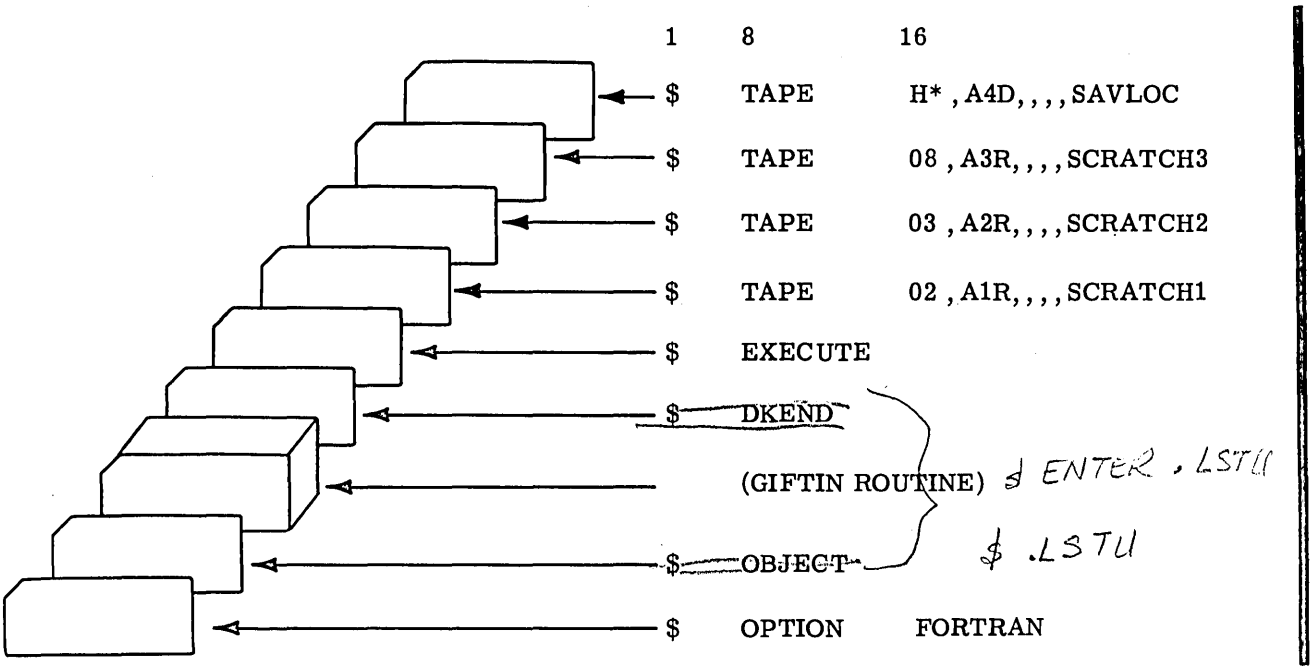
- Implied Synonym Relationships. FORTRAN II stores the imaginary parts of a complex array--or the low-order parts of a double-precision array--in a separate block, while FORTRAN IV stores the two parts of each array element in consecutive memory words. GIFT maintains the relative positions of arrays linked through EQUIVALENCE statements, but it does not always maintain implied synonym relationships. For example:

```
I     DIMENSION A (5)

      EQUIVALENCE (A (2), B)
```

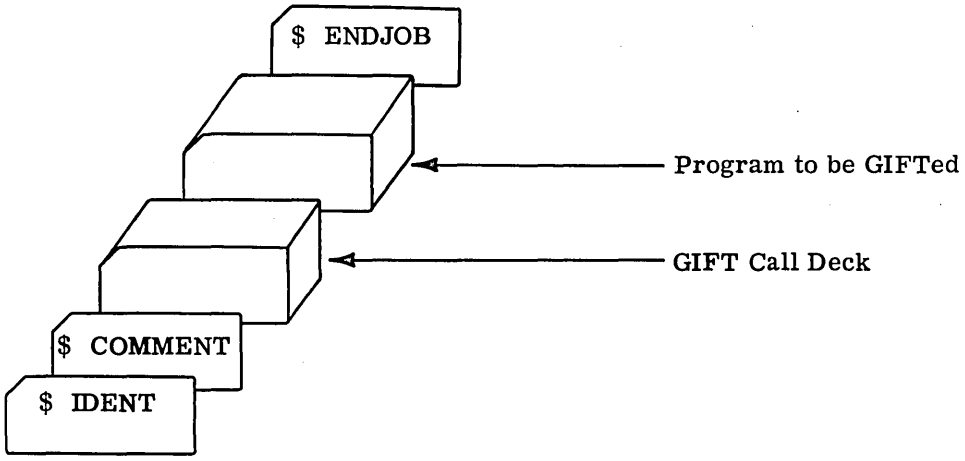
  This FORTRAN II expression specifies that B shares storage with the second word of the 10-word block A. These statements also imply that B shares storage with the real part of the second complex number in A. When the FORTRAN IV compiler processes the GIFT/Translated version of this program, B still shares storage with the second word of block A; but B also implicitly shares storage with the imaginary part of the first complex number in A.

GIFT Call Deck



GIFT Usage Procedure

1. Operator will place the GIFT call deck in user's deck as shown below.
2. All other instructions to the operator will appear on the console.



Double-Precision and Complex COMMON Variables

An equivalence relationship implied through different COMMON statements in two or more programs of a job may be destroyed if one of the COMMON statements contains double-precision or complex variables. For example:

```
                SUBROUTINE SUB1          SUBROUTINE SUB2
                COMMON R, S, T           COMMON X, Y, Z
I               A = S**2                 DIMENSION Y (2)
```

In this example, R is equivalent to X, S to Y, and T to Z. The beginning of the GIFT/Translated version of SUB1 would be:

```
SUBROUTINE SUB1
COMMON R, Q000CM, S, T
COMPLEX S, A
A = S**2
```

The artificial variable Q000CM is inserted in COMMON to cause the real part of the complex variable S to be stored in an even-numbered location. The implicit equivalence relationship between S and Y and between T and Z is thus destroyed.

GIFT processes each subprogram of a job separately and therefore does not modify the COMMON statement of SUB2 in the above example.

Continuation Cards

The GIFT/Translated version of certain statements is sometimes longer than the original statement. In such cases, GIFT generates up to 19 continuation cards. If a statement requires more than 20 such cards, a diagnostic message is written and the statement is ignored.

USAGE

GIFT will exist on a file with file code H*. A program named GIFTIN may be written to call GIFT from H*. The object deck of this program plus control cards will be referred to as the GIFT call deck. To use GIFT, the user places the call deck preceding the program to be GIFTed. The GIFT call deck will then call GIFT overlays from H*. This process will recycle until there are no more cards to be GIFTed. GIFT will terminate with the message END OF FILE READING FRTRN UNIT I*.

Library subroutine .LHSF may be used to read an H* tape. Substitute the following cards for the GIFTIN routine on page 15.

```
$    USE          .LHSF
$    ENTRY        .LHSF
```

The control cards in the GIFT call deck may be changed for various options in accordance with GECOS specifications.

Programmer Option Cards

For most translation jobs, no special options need be specified. However, file control, *ASSIGN, and *REPLACE cards allow the user to modify the GIFT file assignments, the insert variables, and the existing variables in the source program.

- File Control Cards--Altering File Assignments. In addition to required system files, GIFT requires the files listed below during the translation of a subprogram:

F_1--Input file
F_2--Output file
F_3, F_4, F_5--Intermediate or scratch files

Usually, any modification of file assignments is permanent and is performed by an installation for its particular requirements. The user can modify the assignment of specific physical devices for files by means of file control cards as described in Chapter III of the GE-635 General Comprehensive Operating Supervisor Reference Manual, CPB-1002. The *ASSIGN card described below is used to define the logical files to be used by GIFT.

- *ASSIGN Cards--Changing Insert Variable Names. During the process of subprogram translation, it is sometimes necessary to replace the names of variables and constants in the source program with new ones and to insert dummy variables in COMMON statement translations. For instance, the following statement could appear in the source program:

```
READ INPUT TAPE 5, 20, A, B
```

This might then be translated into:

```
INTEGER Q000TP
DATA Q000TP/5/
READ (Q000TP, 20) A, B
```

To make the insert variable names as unique as possible, the following list is used by GIFT.

Insert Variable Name	Use
Q000TP	To replace numeric tape references
Q000CM	To use as dummy inserts in COMMON
Q000HL	To replace Hollerith literals in arithmetic statements
Q000CT	To replace octal constants in Boolean statements
Q000FL	To replace or insert a floating-point variable
K000FX	To replace or insert a fixed-point variable
Q000EQ	To replace double-precision or complex equivalences
QQ000Q, Z000QQ, Z00Q0Q, Z0Q00Q, ZQ000Q, K000ZQ, Q00Z0Q, Q0Z00Q, QZ000Q, Q000QZ, K00Q0Z, Q0Q00Z, QQ000Z	Spares for all above

Numeric file references are not replaced by GIFT unless a file variable replacement name was given as the first variable name on the *ASSIGN card currently in effect. Individual file constants may still be replaced by use of *REPLACE card.

The user who wishes to replace one or more of these variable names by using an *ASSIGN card must be familar with the method used by GIFT when selecting an insert variable. This is best described by an example.

Suppose that four octal constants in a Boolean statement must be replaced by variable names. The names selected by GIFT might be (in succession from left to right):

Q000CT Q001CT Q002CT Q003CT

Construction of insert variables would continue in this manner until Q999CT was reached. If more replacements were necessary, one of the spares would be taken and the modification procedure would begin anew.

Replacement names are assigned in the order of appearance of the item to be replaced in the input source program. Therefore, file constants, Hollerith literals, and octal constants may be assigned different replacement names in different subprograms.

The following rules apply when changing insert variables in the list:

1. If the new variable contains fewer than six characters, it is left-adjusted and filled with trailing zeros.
2. The new variable should contain some numeric characters so that incrementing of the characters can occur.
3. If the Hollerith literal, octal constant, or floating-point replacement is changed, it must not begin with any letter from I through N.
4. If the fixed-point replacement is changed, it must begin with any letter from I through N.

The *ASSIGN card--used for changing logical file assignments and/or inserting variable names, as just discussed--should be prepared and used in the following manner:

1. Format:

   Column 1 7

   * ASSIGN (F_1, F_2,... F_5) V_1, V_2,... V_n

   Where:

   F_1, F_2,... F_5 refer to the respective files listed in the table on page 16.

   V_1, V_2,... V_n refer to the respective insert variable names listed in the table on page 16.

   Blanks are ignored.

   Example: To change the output file to file 11 and to change the file constant and floating-point variable replacements:

   * ASSIGN (,11) K00T,,,,AAX009

2. Placement: The *ASSIGN card must be the first card in the source deck to be translated.

3. Persistence: An *ASSIGN card remains in effect until changed by another *ASSIGN card. To return to the original GIFT configuration, a blank *ASSIGN card may be used.

- *REPLACE Card. The user may change the variable names used in a source program by means of the *REPLACE card. Also, it may be used to replace a file designator constant with a variable or another constant.

For example, it may be desirable to change the name of a table of fixed-point items from ITABLE to TABLE. If the replacement is requested via the *REPLACE card, the translator generates the statement INTEGER TABLE and changes all references of ITABLE to TABLE. A maximum of 20 variable names may be replaced.

The card should be prepared and used in the following manner:

1. Format:

   Column 1 7

   * REPLACE (A_1, B_1), (A_2, B_2),...,(A_n, B_n)

   Where:

   A_1, A_2,... A_n are the variable names or file constants to be replaced and B_1, B_2,... B_n are the replacement names or constants.

Blanks are ignored.

Example: A *REPLACE card is punched as follows:

*REPLACE (S, T), (T, S)

This changes the statement S = T in the source program to T = S in the translation.

2. Placement: *REPLACE cards must follow directly behind the *ASSIGN card or be the first cards in the source program if there is no *ASSIGN card.

3. Persistence: A *REPLACE card affects only a single input program.

3. TECHNICAL SPECIFICATIONS

INTRODUCTION

The following description and flow charts are on a general level of information. The listing of the GIFT program provides many comments and descriptions for detailed information.

FILE NAMES

GIFT uses the following file designations:

Name	Use	File Code
ITAPE	INPUT FILE	05
IOTAPE	OUTPUT FILE (Printer)	06
NTAPEA	INTERMEDIATE	02
NTAPEB	INTERMEDIATE	03
NTAPEC	INTERMEDIATE	08
(No Name)	OUTPUT FILE (Punch)	43

The logical files are set in subroutine INIT, Link 1.

OUTPUT

In Pass 1, subroutine READIN prints a listing of the input source program on the IOTAPE. If this listing is not desired, the appropriate statements may be removed from the READIN subroutine, and the subroutine recompiled.

The output of the translated program is put on the IOTAPE and punched by means of the FORTRAN statement PUNCH in subroutine OUTPUT, Pass 3. If this method of output is not desired, the appropriate modifications may be made by proper use of file control cards.

The listing is provided with numbered pages. The total number of lines per page, including heading lines, is carried in IPATCH (19). This COMMON location is initialized in the subroutine 1INIT. As distributed, IPATCH (19) = 55.

ADDITIONAL STATEMENT TYPES

To modify GIFT to recognize statement types other than those in the Pass 1 main program listing, one must make the following changes:

1. Add the statement type to the dictionary by inserting a GMAP card of the form BCI n, xxx $ k where xxx is the statement name, k is the type number (48, 49, or 50) and n is the number of 36-bit words required to contain in BCD the name, dollar sign, type number, and at least one terminating blank. For example,

```
BCI  2, TABLE $ 48
```

2. Modify the coding in Pass 1 routine (PASS1) and/or Pass 3 routine (PASS3) to process statements of the added type when the variable ITYPE contains the value assigned to the added statement. In conjunction with the above example,

```
                GO TO 4000          1PS12095
3048            CALL PTABLE         1PS12100
```

   might be inserted in PASS1 and a routine PTABLE added to process TABLE statements.

3. If more than three additional statement types are to be recognized, additional type numbers may be assigned starting from 54. The long computed GO TO in PASS1 and/or PASS3 must be suitably expanded.

ROUTINE INFORMATION

GIFT is a three-pass overlay program. Pass 1 preprocesses the input program. Pass 2 constructs TYPE, DATA, and COMMON statements from information gathered during Pass 1. Pass 3 contains the third pass over the input program and outputs the GIFTed program. Following is a list of each subroutine and its function. Further details may be found in the program listings and flow charts. An explanation of COMMON variables, and overlay map and routine category as to overlay may be found in the Appendix.

SUPERVISOR

Main Program--Calls INIT, PASS1, FAPIN2, PASS 2, INIT 3, PREP, PASS3.

PASS 1

During Pass 1 *ASSIGN cards, if any, are processed and the tape configuration and/or insert variable list are modified accordingly. The *REPLACE cards, if any, are then processed and a table formed for later use. Each FORTRAN statement is then read in and classified according to type (arithmetic, DO, GO TO, etc.).

Every variable name is then checked against the insert variable list and the replacement list. If a variable name is identical to an entry in the insert variable list, the insert variable entry is removed from the list. If a variable name is identical to an entry in the replace list, it is replaced by the corresponding name. Variables in double-precision and complex statements are placed in a list of double-precision and complex variables. Lists of tape constants, function names beginning with X or I-N, and COMMON, EQUIVALENCE, DIMENSION, double-precision, and complex variables are also formed.

1. Initialization Routines

   a. INIT--Initializes parameters, rewinds tapes, etc. Calls FAPIN1, READIN, SCAN, and CCLASS. Calls PASIGN or PRPLCE if card is *ASSIGN or *REPLACE card respectively.

   b. FAPIN1 (GMAP Routine)--Sets up COMMON parameters, encodes statement dictionary, etc.

2. Supervisory Routine and Statement Processors

   a. PASS1--Reads statements, calls SCAN, CCLASS and proper processor for type of statement and then calls SHIP.

   b. PASIGN--Processes *ASSIGN cards and modifies the tape configuration and insert variable list accordingly.

   c. PRPLCE--Processes *REPLACE cards and forms a table of names to be replaced and their replacements. Writes Type statements (REAL or INTEGER) on NTAPEB if necessary.

   d. PFCARD--Processes F cards. Constructs an EXTERNAL statement and writes it on NTAPEB.

   e. ARITH--Processes arithmetic, IF, CALL, and double-precision and complex DIMENSION statements. Forms a list of double-precision and complex variable names. Writes out TYPE statement (REAL or INTEGER) for all function names beginning with I-N or X.

   f. EQV--Processes COMMON, DIMENSION and EQUIVALENCE statements.

      (1) COMMON--Records name and order in COMMON of each COMMON variable.

      (2) DIMENSION--Records name and length of each array and calls ARITH if statement has a D or an I in column 1.

      (3) EQUIVALENCE--Records name, group number and subscript for each literal appearance of a variable.

   g. INOUT--Processes all input/output statements that include a tape reference. Forms table of tape constants used in program.

   h. FORGEN--Processes set of FORMAT Generator cards. Calls READIN, SCAN, CCLASS, and SHIP until END of FORMAT is encountered.

   i. ENDCRD--Rewinds NTAPEA, finalizes NTAPEC, and transfers control to Link 2 via PASS1 and CONVRT.

3. Utility Routines

   a. SCAN--Scans each statement and marks the beginning column and type of each item: for example, variable, constant, punctuation.

   b. CLASS--Determines type of statement.

   c. ERROR (GMAP Routine)--Internal diagnostic routine.

d. ENCODE (GMAP Routine)--Unpacks and encodes a block of packed BCD words to a one-character-per-word string in an internal code.

e. DECODE (GMAP Routine)--Decodes and packs an internally coded one-character-per-word string into a block of packed BCD words.

f. PACK (GMAP Routine)--Packs two signed fixed-point integers into one computer word.

g. UNPACK (GMAP Routine)--Unpacks one computer word into two signed fixed-point integers.

h. OUTCOM--Writes out every comment card with an identification number onto NTAPEC.

i. READIN--Reads a card. Calls ENCODE. Calls OUTCOM if card is comment card.

j. SHIP--Outputs statement and auxiliary information onto NTAPEA.

k. TIVAR--Checks each variable name against the insert variable list and deletes insert entry if a match is found. Checks each variable name against replace list and substitutes replacement if a match is found.

l. UNPAR (FUNCTION Subprogram)--Searches for unparenthesized character. Used by CLASS to determine whether statement is arithmetic.

m. PAGE1--Formats listing of input program.

n. SKFP--Skips to end of FAP deck.

PASS 2

During Pass 2 tables formed by Pass 1 subroutines EQV, ARITH, and INOUT are processed and COMMON, and DATA statements are written onto NTAPEB.

1. Supervisory Routine and Statement Generating Routines

a. PASS2--Calls CONCOM, DPORC, and ASTAPE, finalizes NTAPEB and then transfers control to Pass 3 via the main program.

b. CONCOM--Calls SORTAC. Constructs new COMMON statements, if necessary, to compensate for the fact that EQUIVALENCE does not reorder COMMON in FORTRAN IV. Calls GETVAR. Checks for odd-even inconsistency of double-precision and complex variables and prints out diagnostic message if inconsistency is detected.

c. DPORC--Generates DOUBLE PRECISION and/or COMPLEX TYPE statement(s) for double-precision and/or complex variable and function names, if any.

d. ASTAPE--Calls GETVAR. Generates DATA statements for tape constants and insert variables which replace them, if any.

e. GETVAR--Finds next available insert variable name of specified type. Generates INTEGER TYPE statement for tape constant replacement if name does not begin with I-N. If no more insert variable names are available, call NOIVAR.

2. Utility Routines

   a. DECODE (GMAP Routine)--Same as Pass 1.

   b. PACK (GMAP Routine)--Same as Pass 1.

   c. UNPACK (GMAP Routine)--Same as Pass 1.

   d. NOIVAR--Writes out message that no more insert variable names of specified types are available. Terminates execution.

   e. SORTAC (GMAP Routine)--Sorting routine.

PASS 3

During Pass 3, a second pass is made over the source program and auxiliary information written on NTAPEA. This information and the other information gathered during Pass 1 and Pass 2 is processed and merged with the comments on NTAPEC and the GIFTed program is written out.

1. Supervisory Routine and Statement Processors

   a. PASS3--Reads statement and calls proper processor depending on type of statement.

   b. TRIGG--Processes machine-oriented trigger tests.

   c. FGEN--FORMAT Generator processor. Outputs statements between FORMAT and END OF FORMAT unaltered.

   d. COMOUT--Outputs comment (C in column 1) cards.

   e. NXMIT--Non-transmit tape select processor. Replaces numeric tape references in REWIND, BACKSPACE and ENDFILE.

   f. ARTEN--Arithmetic Statement Function Processor. Replaces all variable names in arithmetic statement functions with insert variables of the proper type.

   g. RWTAPE--Constructs FORTRAN IV format for READ TAPE, WRITE TAPE, READ INPUT TAPE, WRITE OUTPUT TAPE, RIT and WOT.

   h. ENDPR--Outputs END statement.

   i. ARTPR--Processes arithmetic statements.

   j. BOOLE--Processes Boolean statements.

   k. PSCAN--Replaces variables which conflict with FORTRAN IV functions and changes built-in function names to their FORTRAN IV equivalent.

   l. DMSCAN--Flag dimensioned variables with an IP code of 7 for BOOLE processing.

   m. VREPL--Replaces variable names in IEQV, IDBLDM, and IDPC tables.

2. Subprocessors to Handle Individual Items within Statements

   a. PARTY--Constructs PART function and CALL STORE for references to part of a double-precision or complex pair.

   b. CKCNST--Examines and truncates floating-point constants to nine significant digits in single precision statements.

3. Utility Routines

   a. POINT--Determines the column number in statement, the length and the type for an item in the current statement.

   b. MOUT--Moves from 1 to N characters to the output area, IOUT.

   c. GETVAR--Same as Pass 2.

   d. PACKP--Moves a specified group of characters from one area to another, omitting blanks.

   e. OUTPUT--Standard output subroutine. Outputs a 20-word BCD string to the IOTAPE and punches the first 14 words by means of PUNCH.

   f. OUTIT--Decodes and outputs an encoded statement string.

   g. CCLASS (GMAP Routine)--Same as Pass 1.

   h. ENCODE (GMAP Routine)--Same as Pass 1.

   i. DECODE (GMAP Routine)--Same as Pass 1.

   j. SEQU (GMAP Routine)--Appends label and sequence number in columns 73-80 of output cards.

   k. DIAG (GMAP Routine)--Prints statement in error diagnostic and outputs statement untranslated.

   l. TBIG (GMAP Routine)--Outputs diagnostic that translated statement exceeds 19 continuation cards.

   m. BINBCD (GMAP Routine)--FORTRAN integer to BCD conversion.

   n. SORTAC (GMAP Routine)--Same as Pass 2.

   o. UNPAR--Same as Pass 1.

   p. NOIVAR--Same as Pass 2.

   q. SCAN--Same as Pass 1.

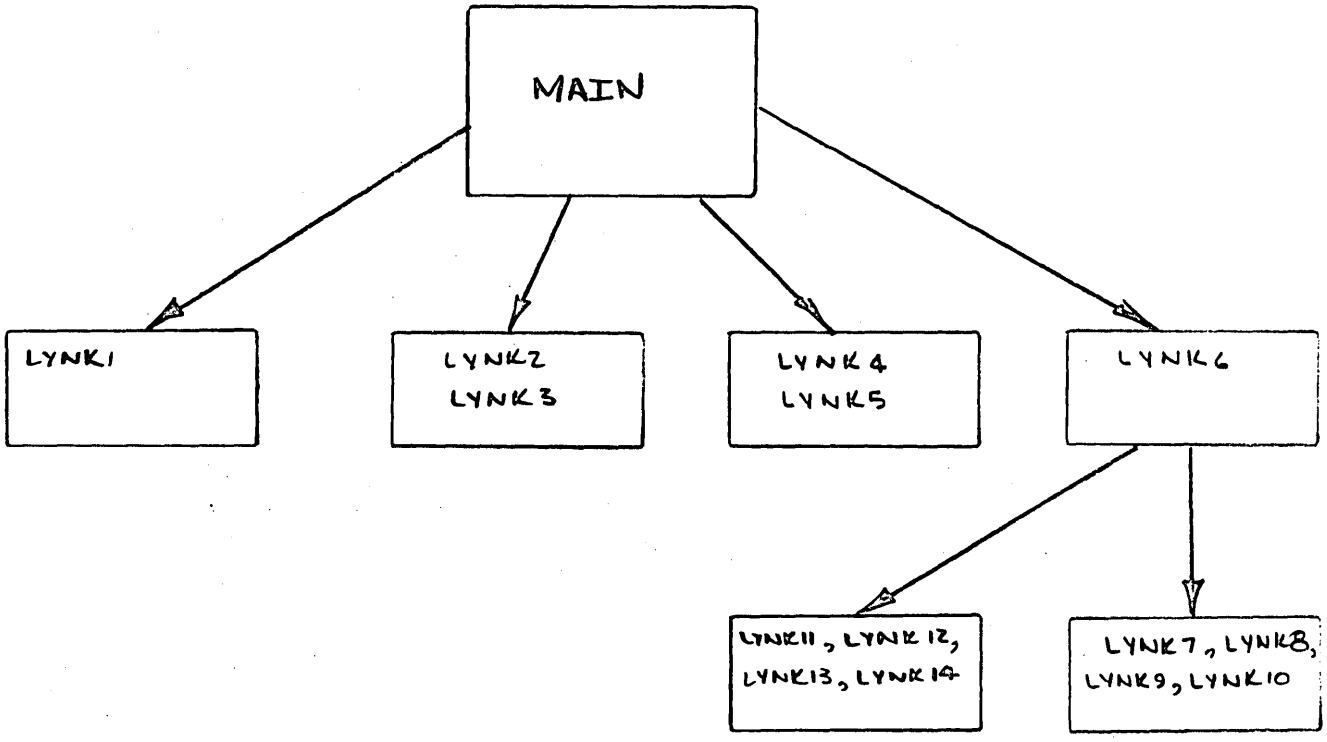
   r. CLASS--Same as Pass 1.

   s. INTIN--Reads input statement from NTAPEB.

   t. PAGE3--Formats listing of GIFTed program.

APPENDIX

MASTER FLOW CHART



GIFT OVERLAY LIST

MAIN

- MAIN
- INHOV
- CLASS
- ERROR
- SCAN
- UNPAIR
- DECODE
- ENCODE
- PACK
- UNPACK
- LLINKI
- LINKI

LYNKI

- FAPINI
- ARITH
- ENDCRD
- EQV
- FORGEN
- INIT
- INOUT
- OUTCOM
- PAGEI
- PASIGN
- PASSI
- PFCARD
- PRPLCE
- READIN
- SHIP
- SKPFAP
- TIVAR

LYNK2

- GETVAR
- NOIVAR
- SORTAC

2

LYNK3

ASTAPE
CONCOM
DPORC
PASS2

LYNK4

BINBCD
DIAG
INIT3
SEQU
TBIG
COMOUT
INTIN
MOUT
OUTIT
OUTPUT
PACKP
PAGE3
POINT
PSCAN

LYNK5

PREP
PTAPEB
VREPL

LYNK6

BOOLE
CKCNST
DMSCAN
PARTY
PASS3

LYNK7

ARTFN

LYNK8
ARTPR

LYNK9
ENDPR

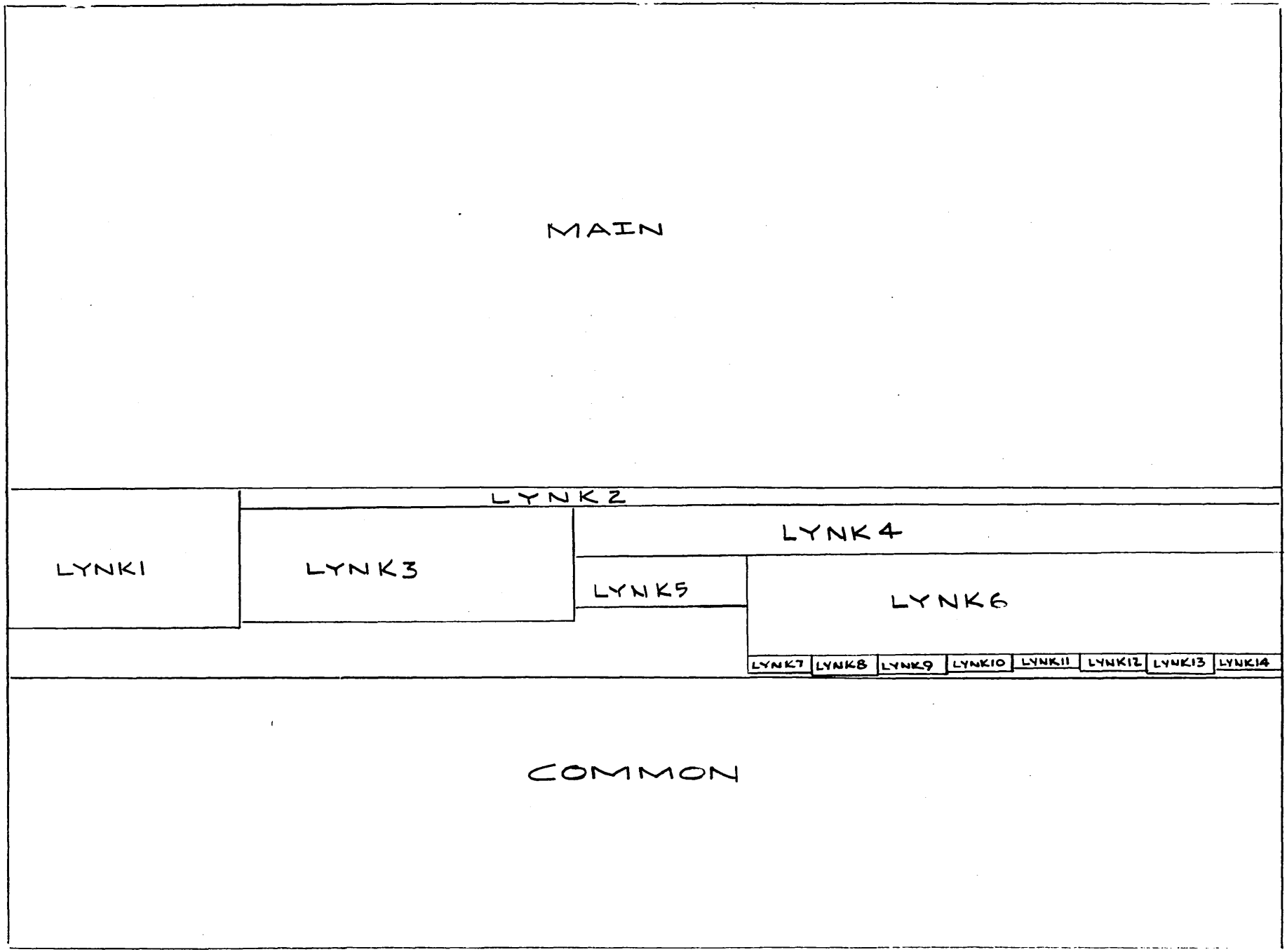
LYNK10
EQUIV

LYNK11
FGEN

LYNK12
NXMIT

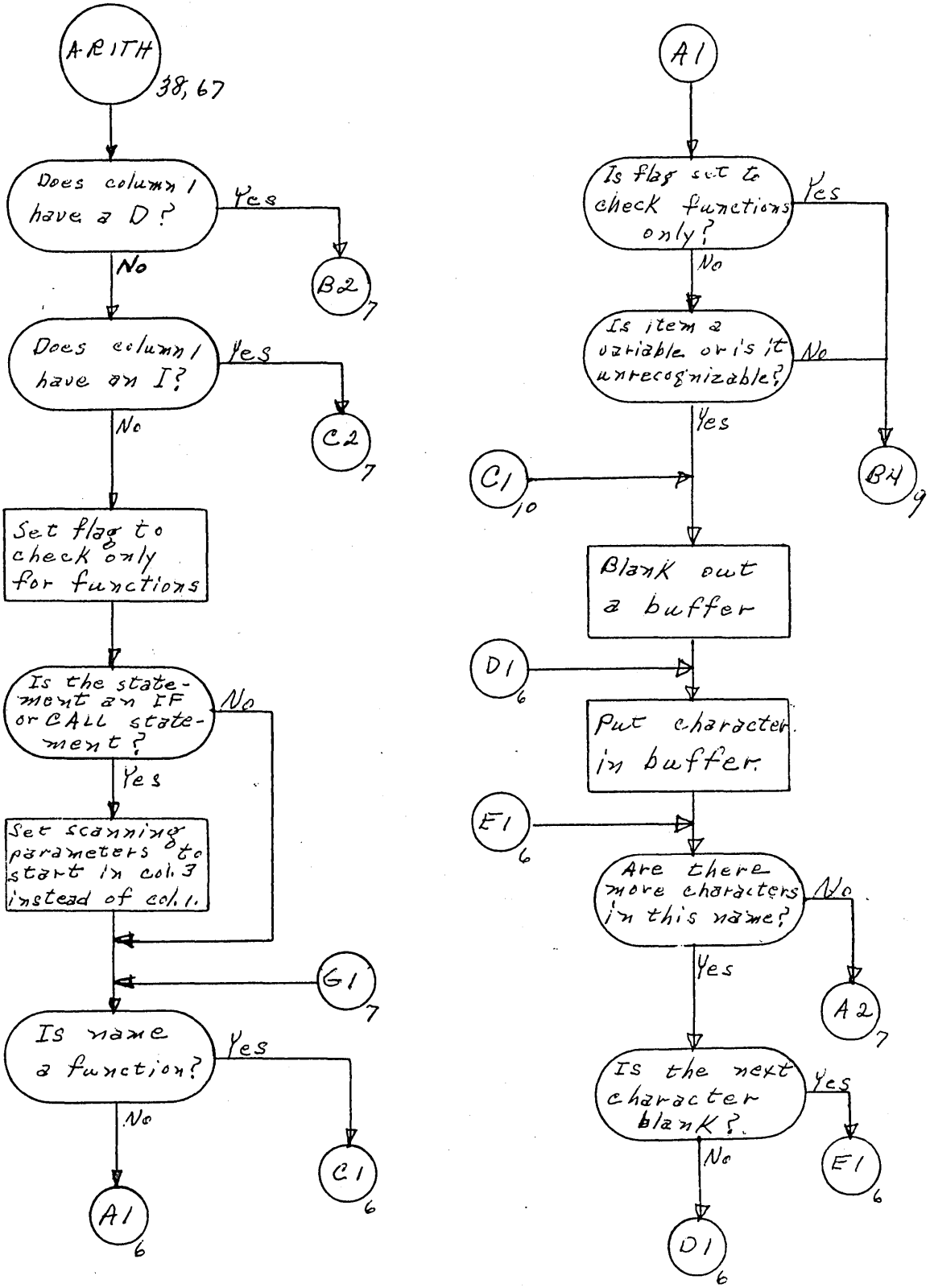
LYNK13
RWTAPE

LYNK14
TRIGG

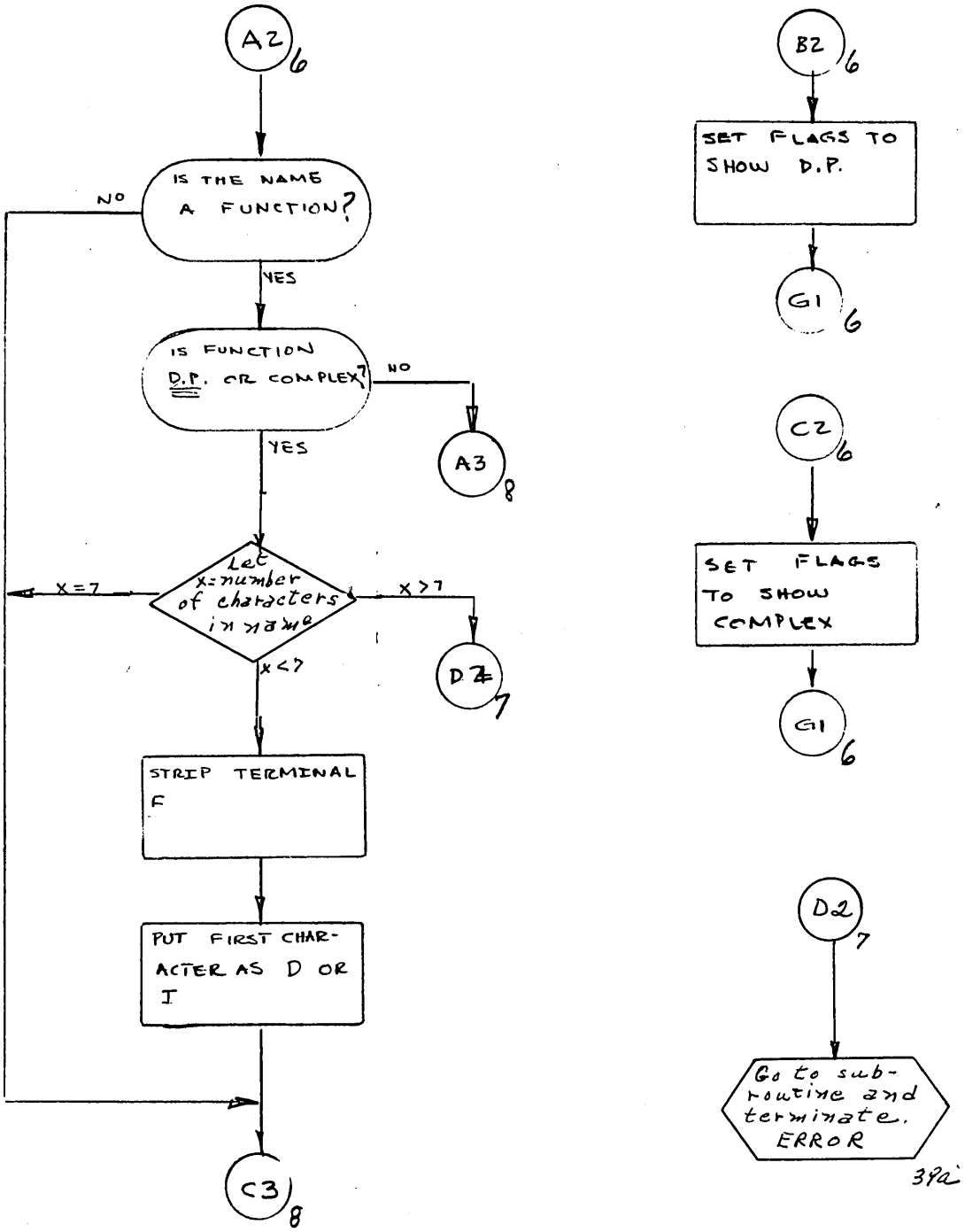


GIFT OVERLAY MAP

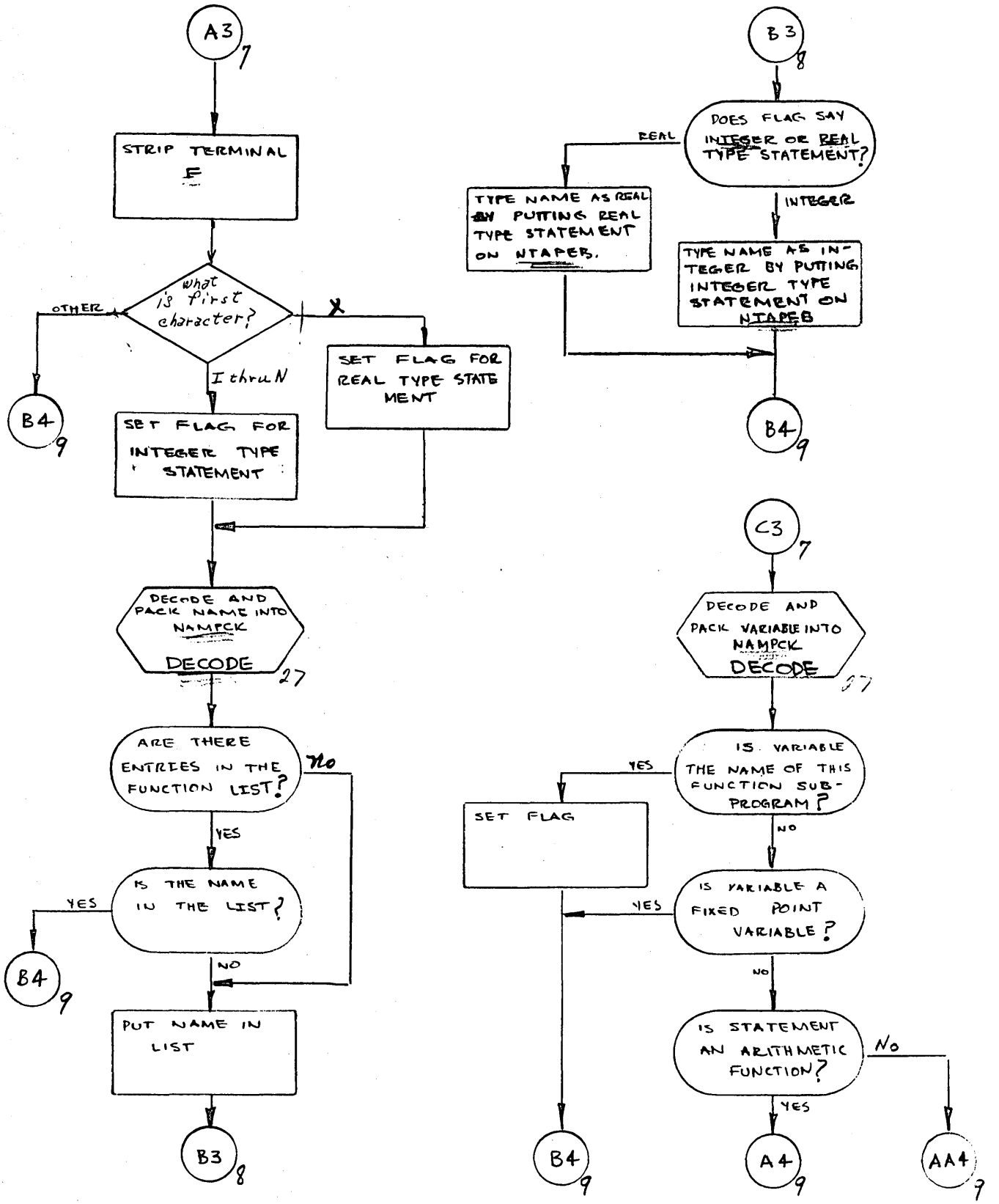
GIFT
SUBROUTINE ARITH
PROCESS ARITHMETIC STATEMENTS



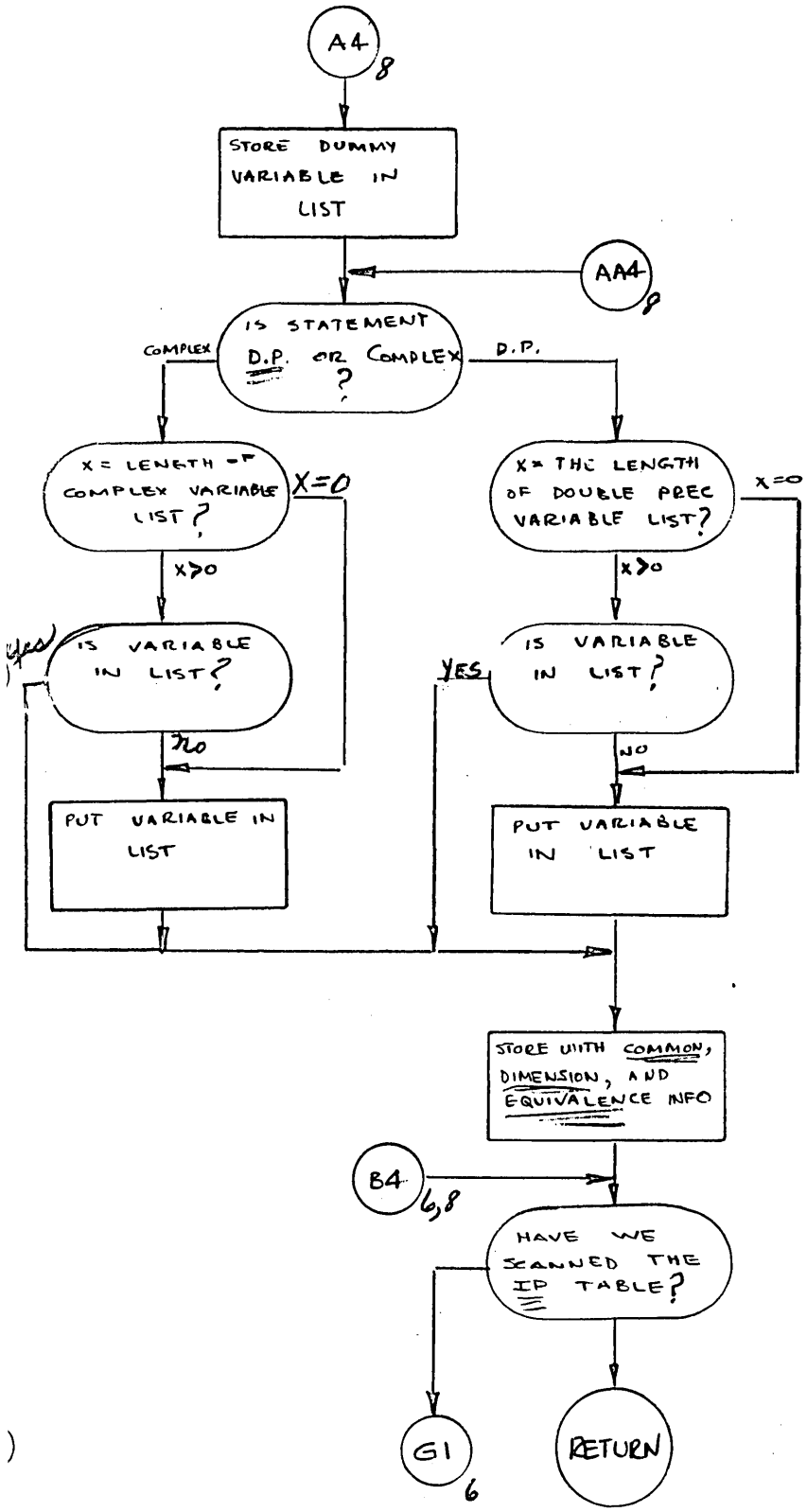
GIFT
SUBROUTINE ARITH



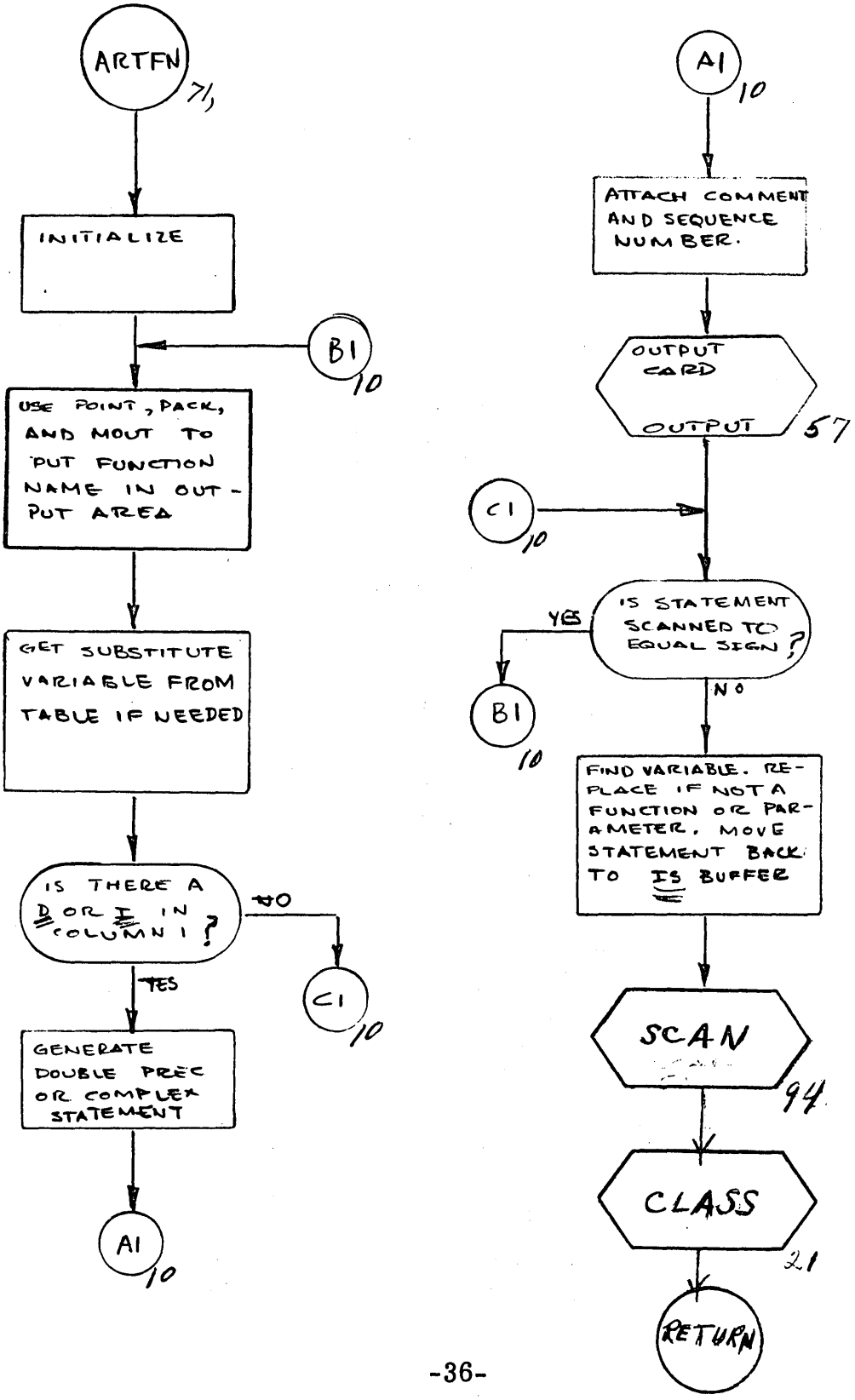
GIFT
SUBROUTINE ARITH



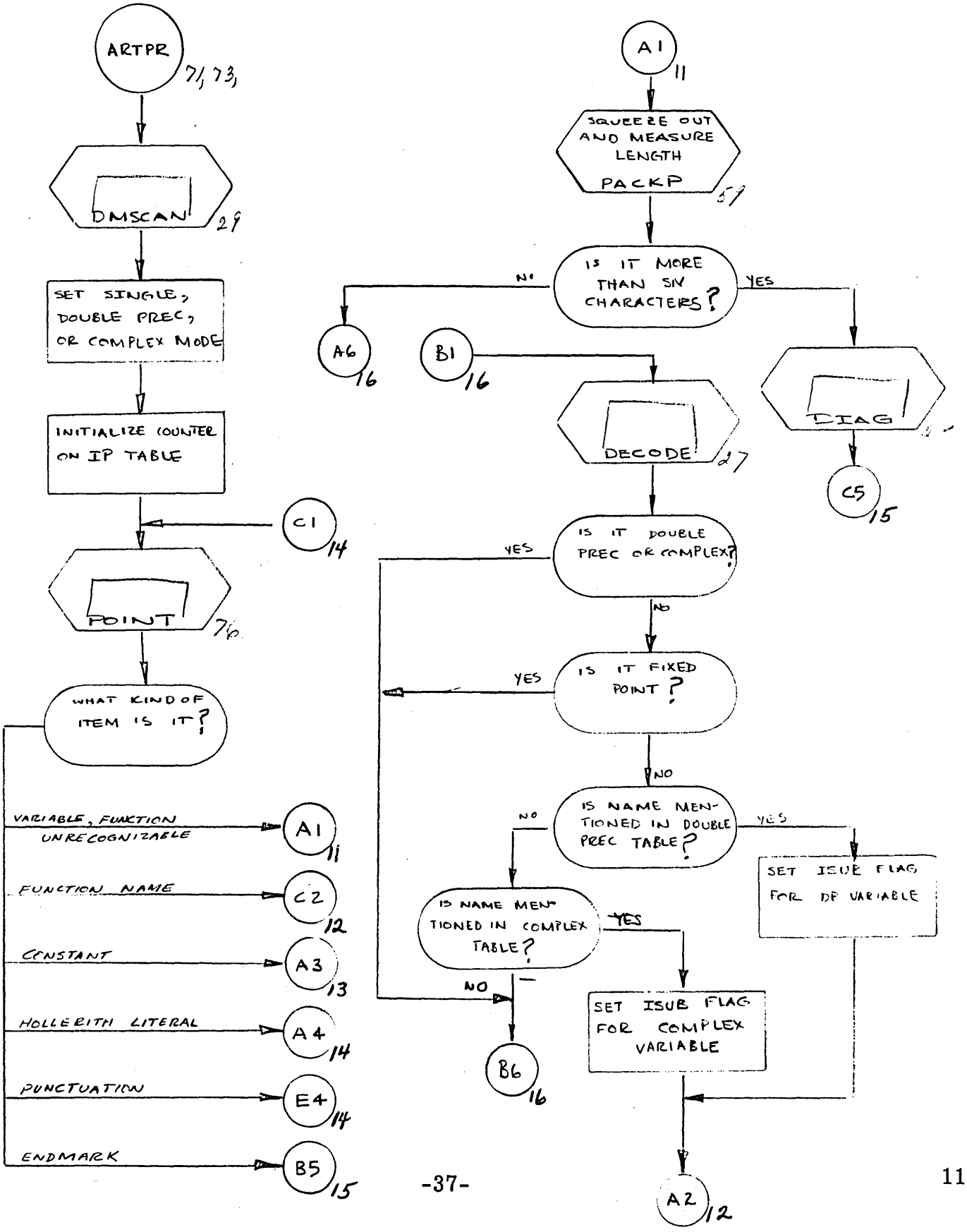
GIFT
SUBROUTINE ARITH



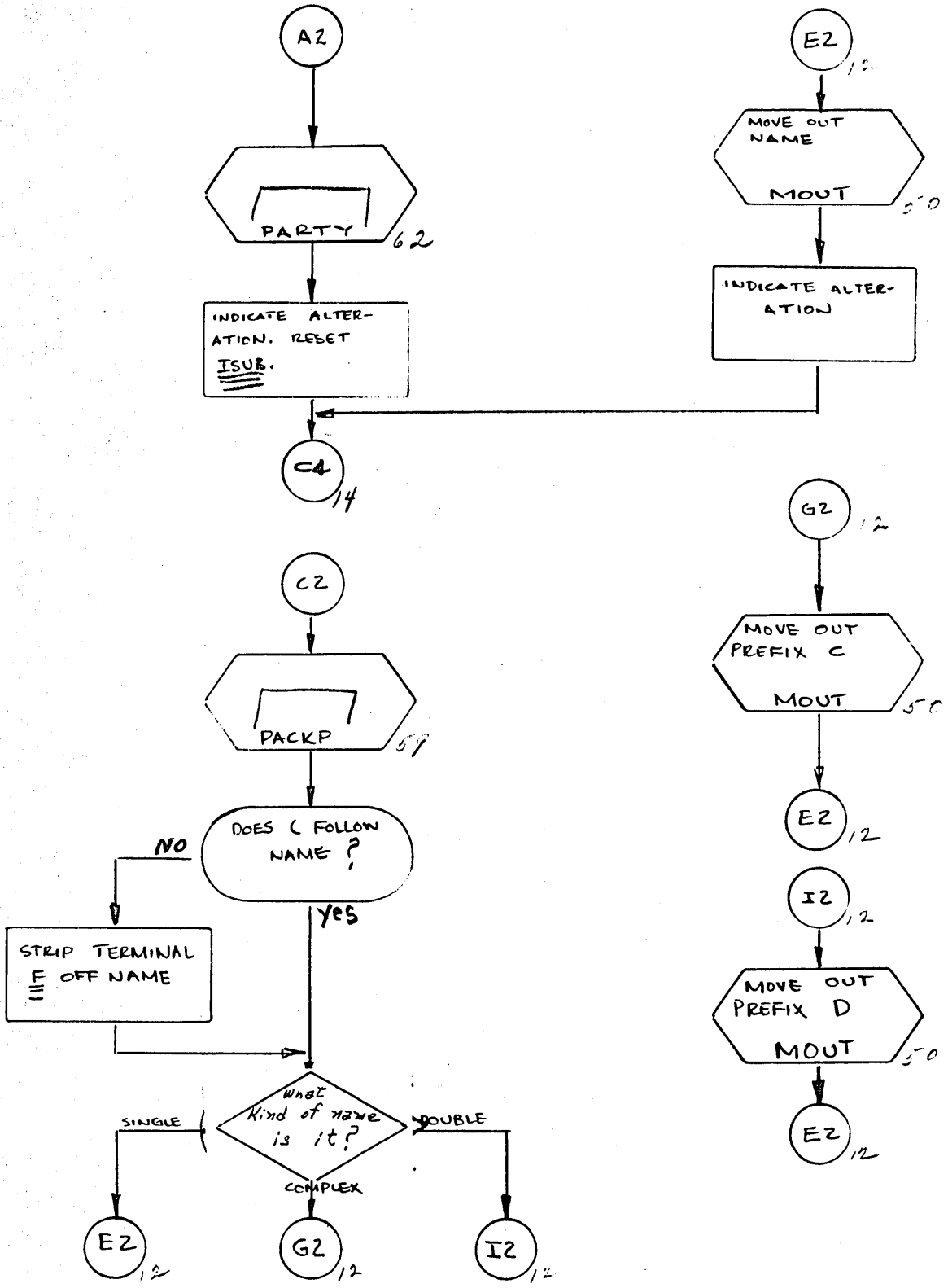
GIFT
SUBROUTINE ARTFN
PROCESS ARITHMETIC STATEMENT FUNCTIONS



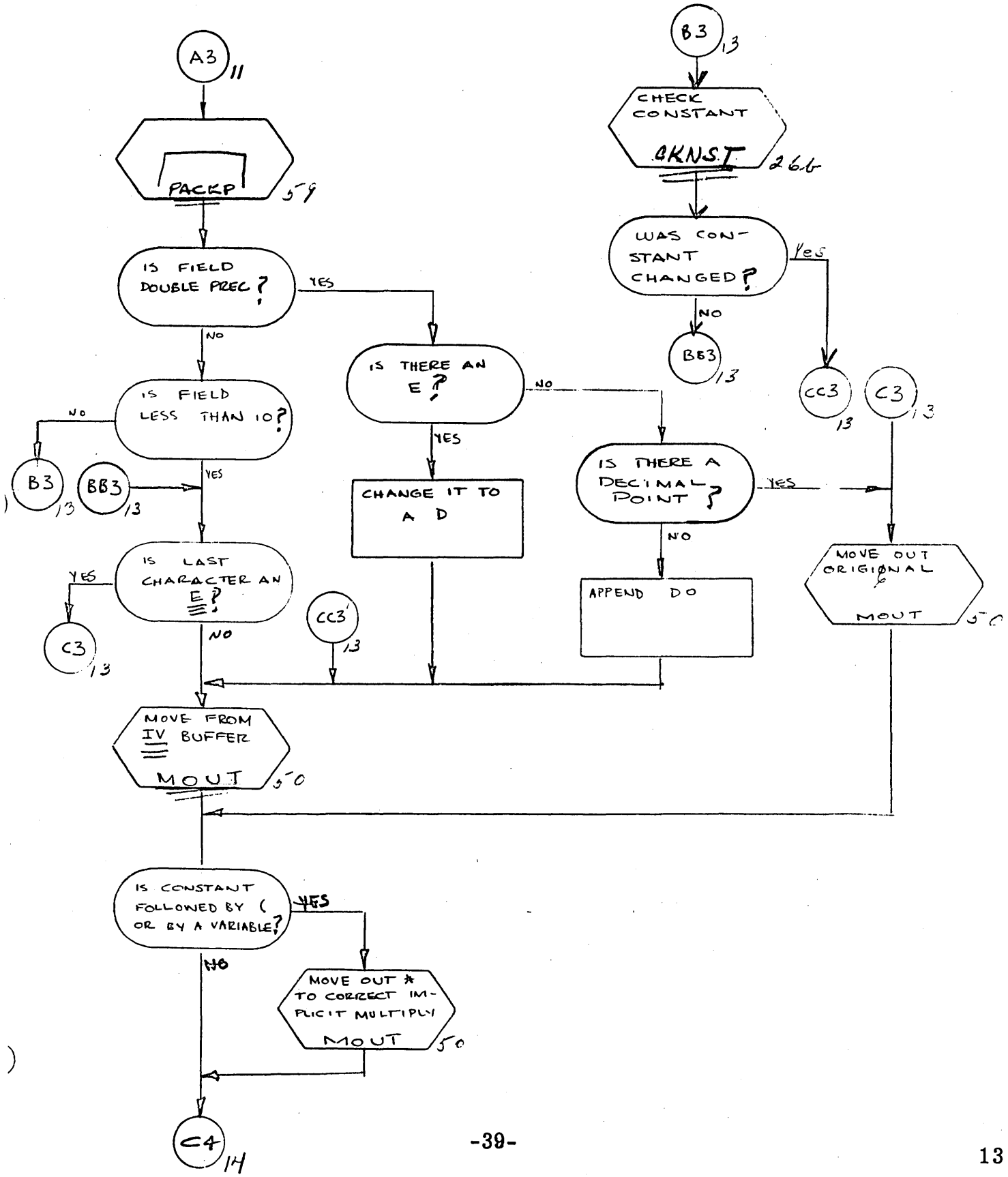
GIFT
SUBROUTINE ARTPR
PROCESS ARITHMETIC, IF, AND CALL STATEMENTS



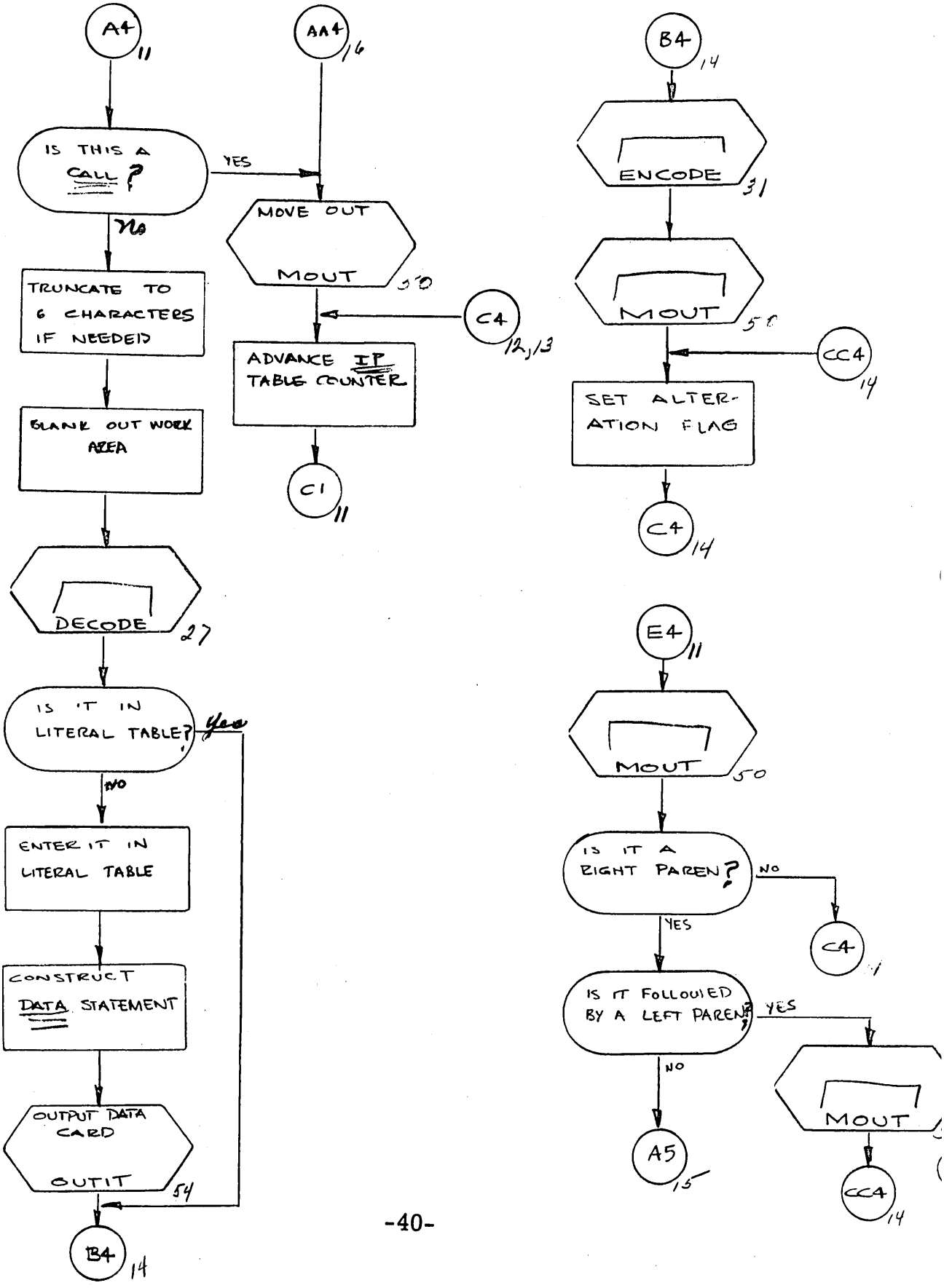
GIFT
SUBROUTINE ARTPR



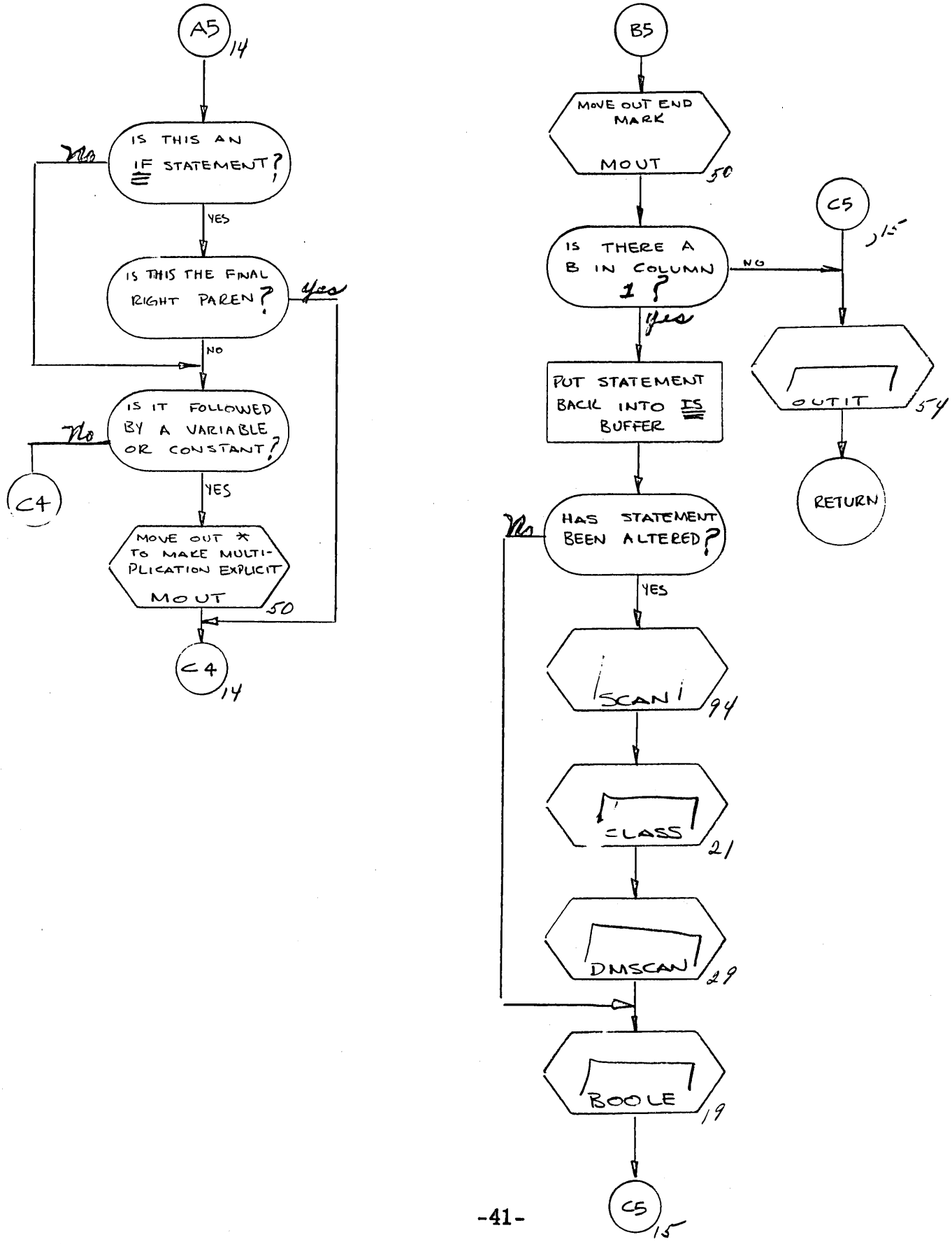
GIFT
SUBROUTINE ARTPR



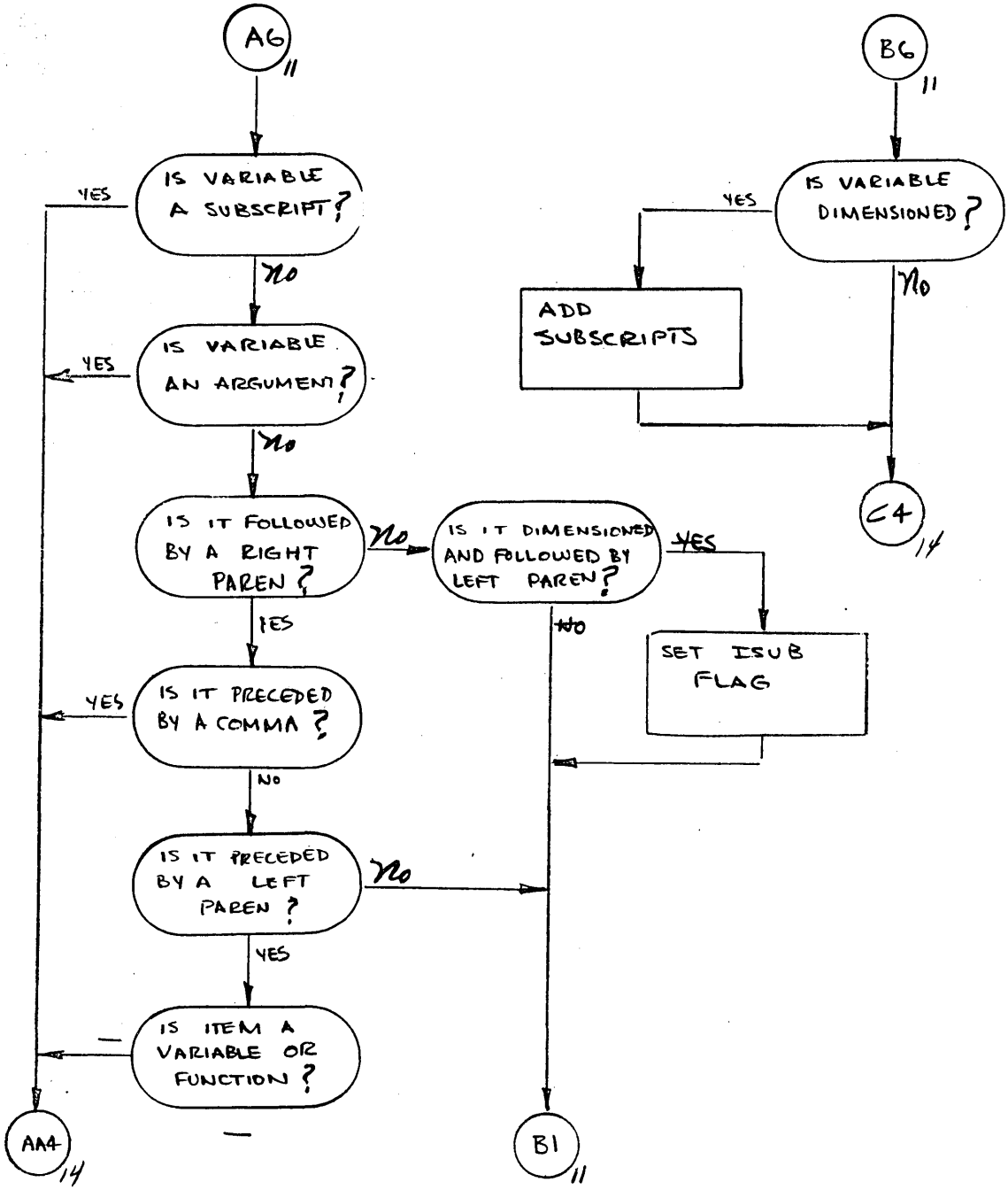
GIFT
SUBROUTINE ARTPR



GIFT
SUBROUTINE ARTPR



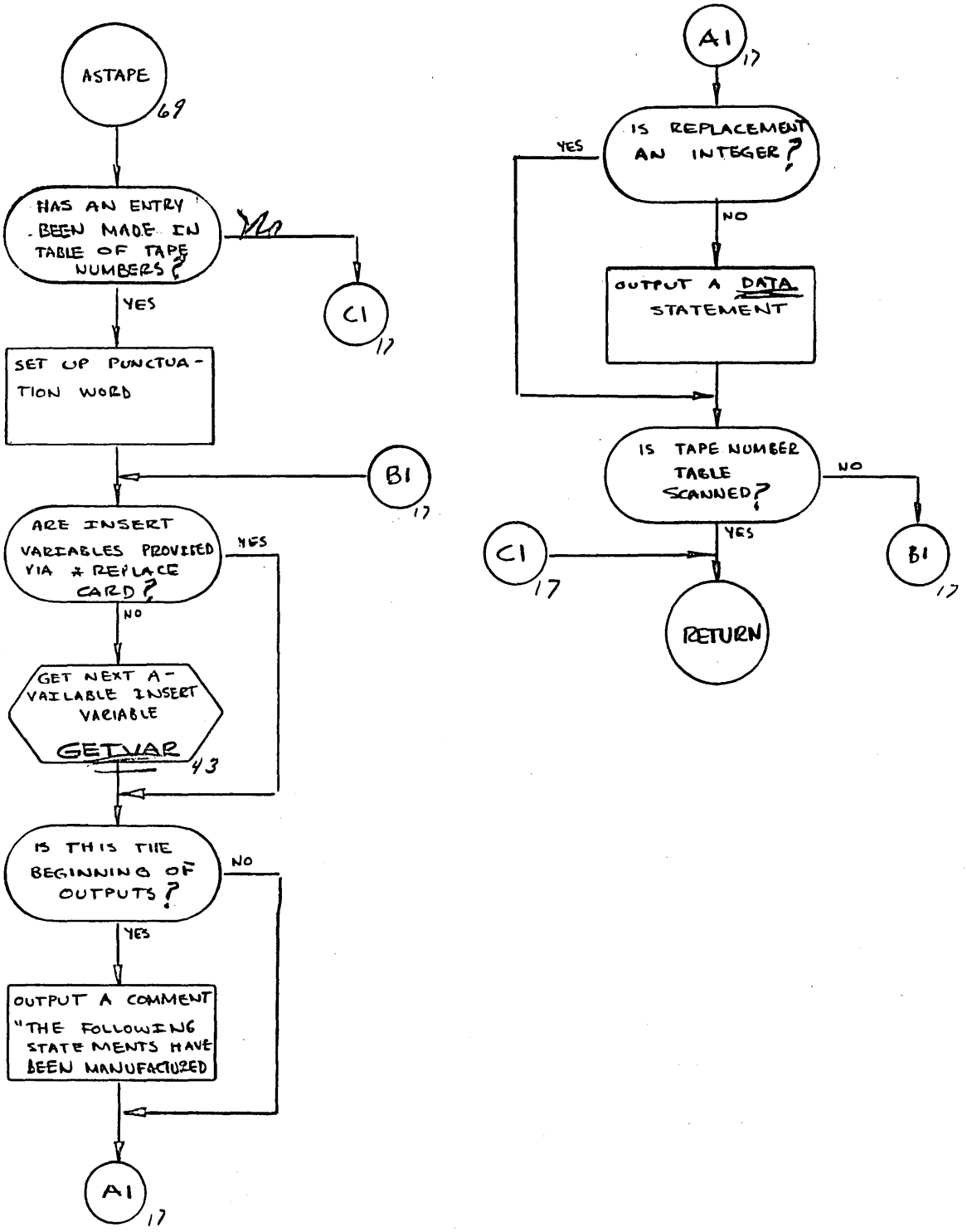
GIFT
SUBROUTINE ARTPR



GIFT

SUBROUTINE ASTAPE

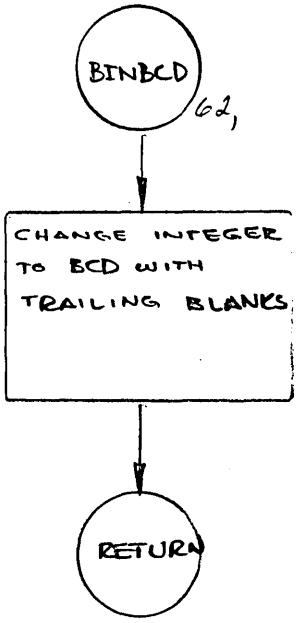
CONSTRUCT DATA STATEMENT FOR TAPE ASSIGNMENTS WHEN NEEDED



GIFT

SUBROUTINE BINBCD

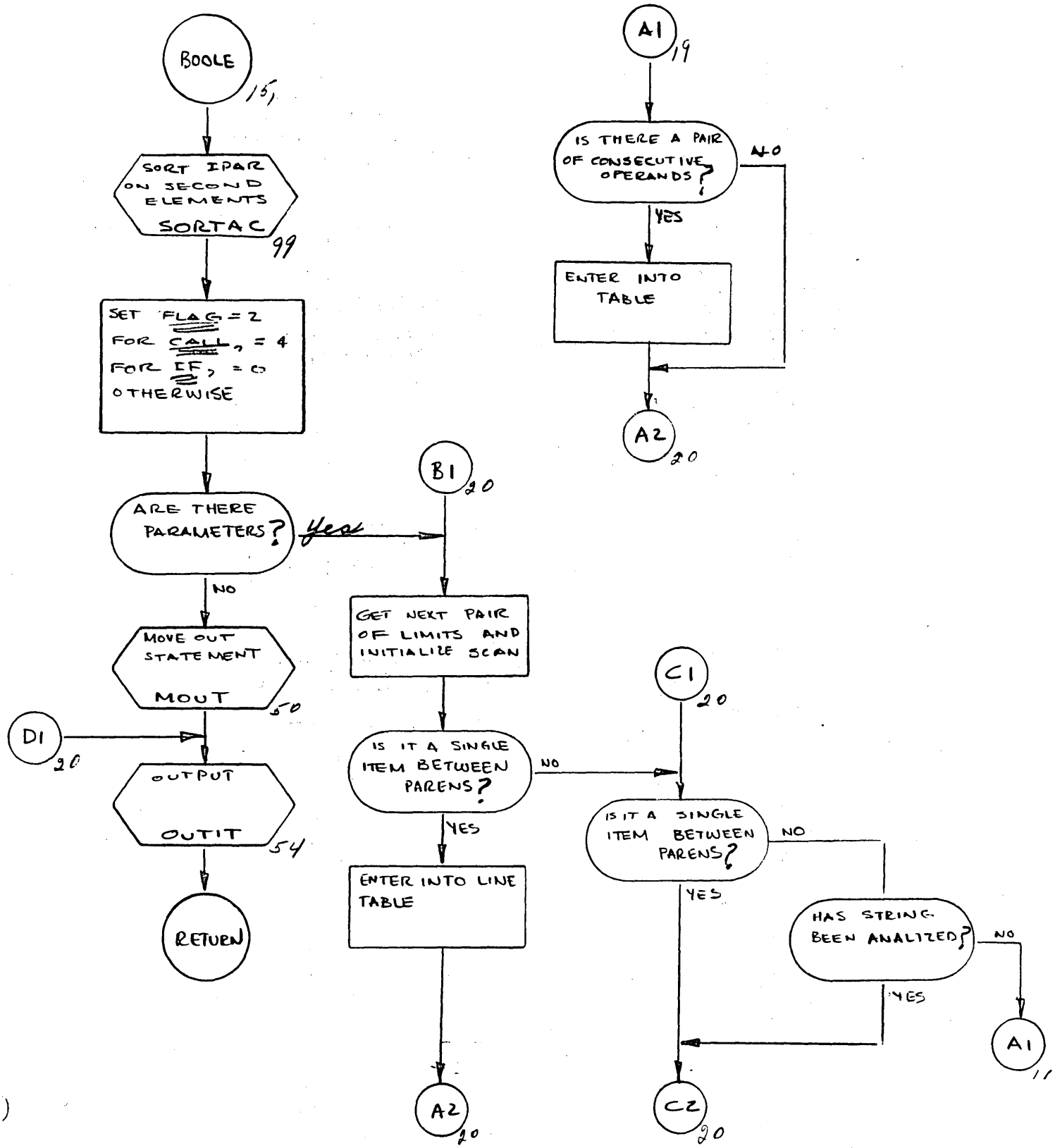
FORTRAN INTEGER CHANGED TO BCD A6 FORMAT



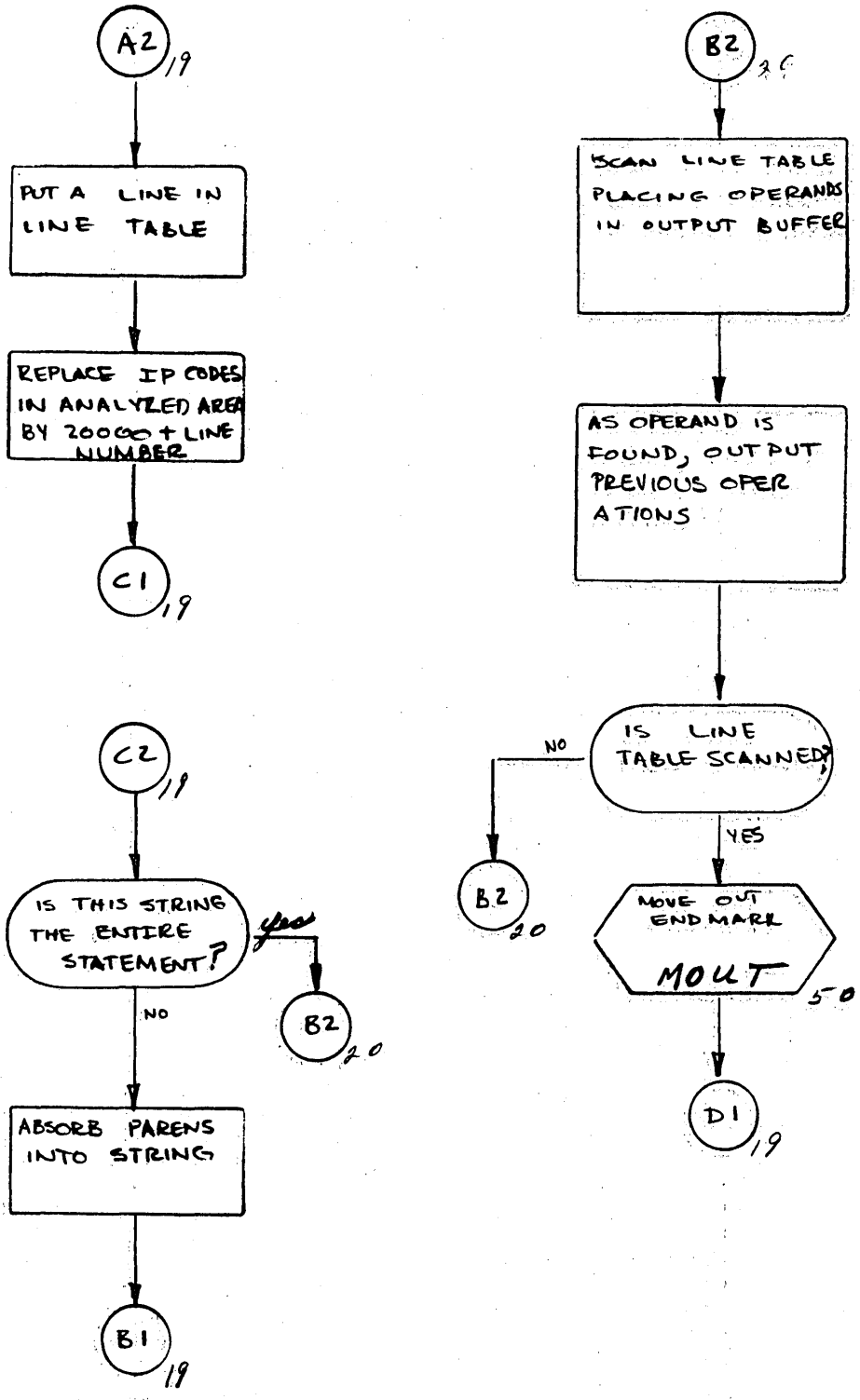
GIFT

SUBROUTINE BOOLE

PROCESS BOOLEAN STATEMENTS



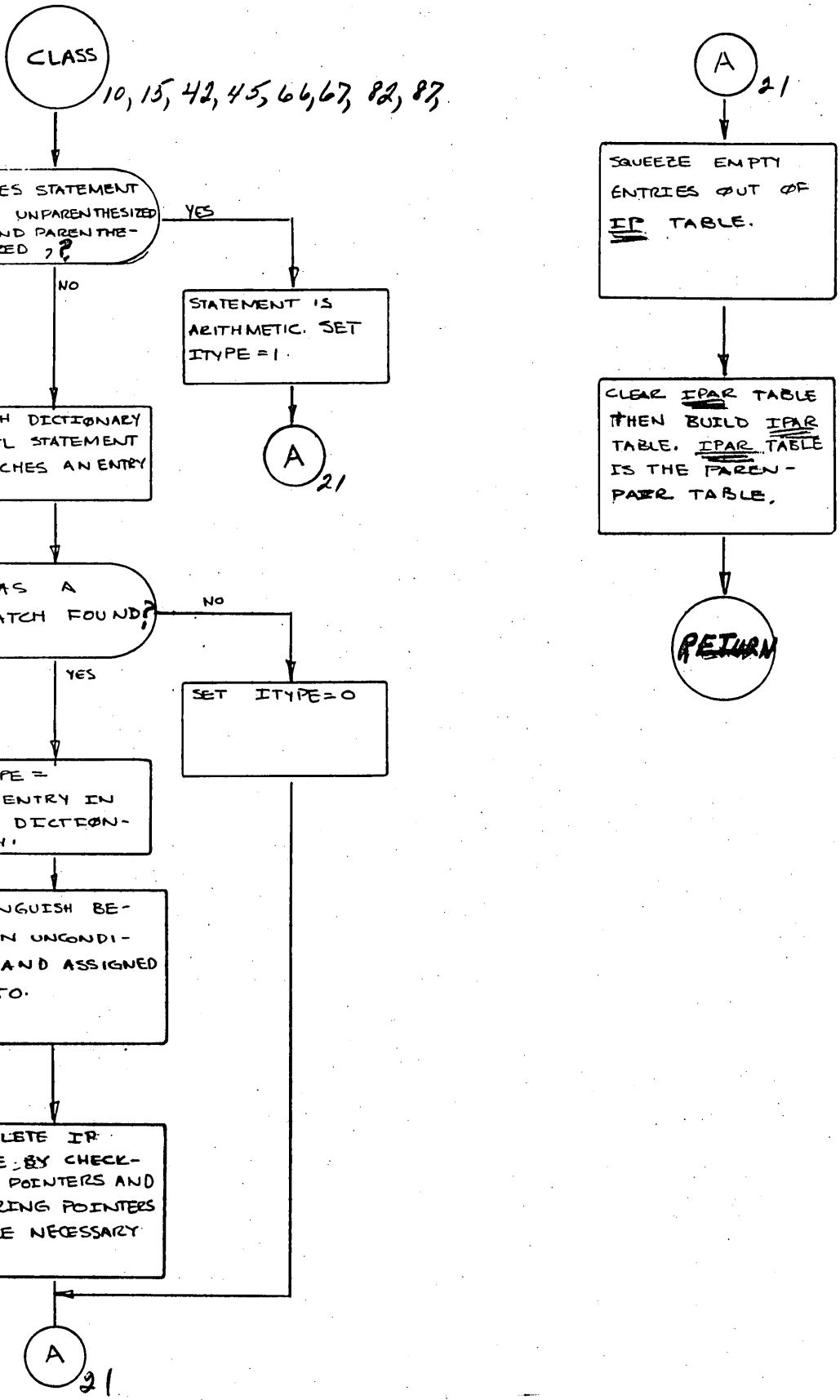
GIFT
SUBROUTINE BOOLE



SIFT

SUBROUTINE CLASS

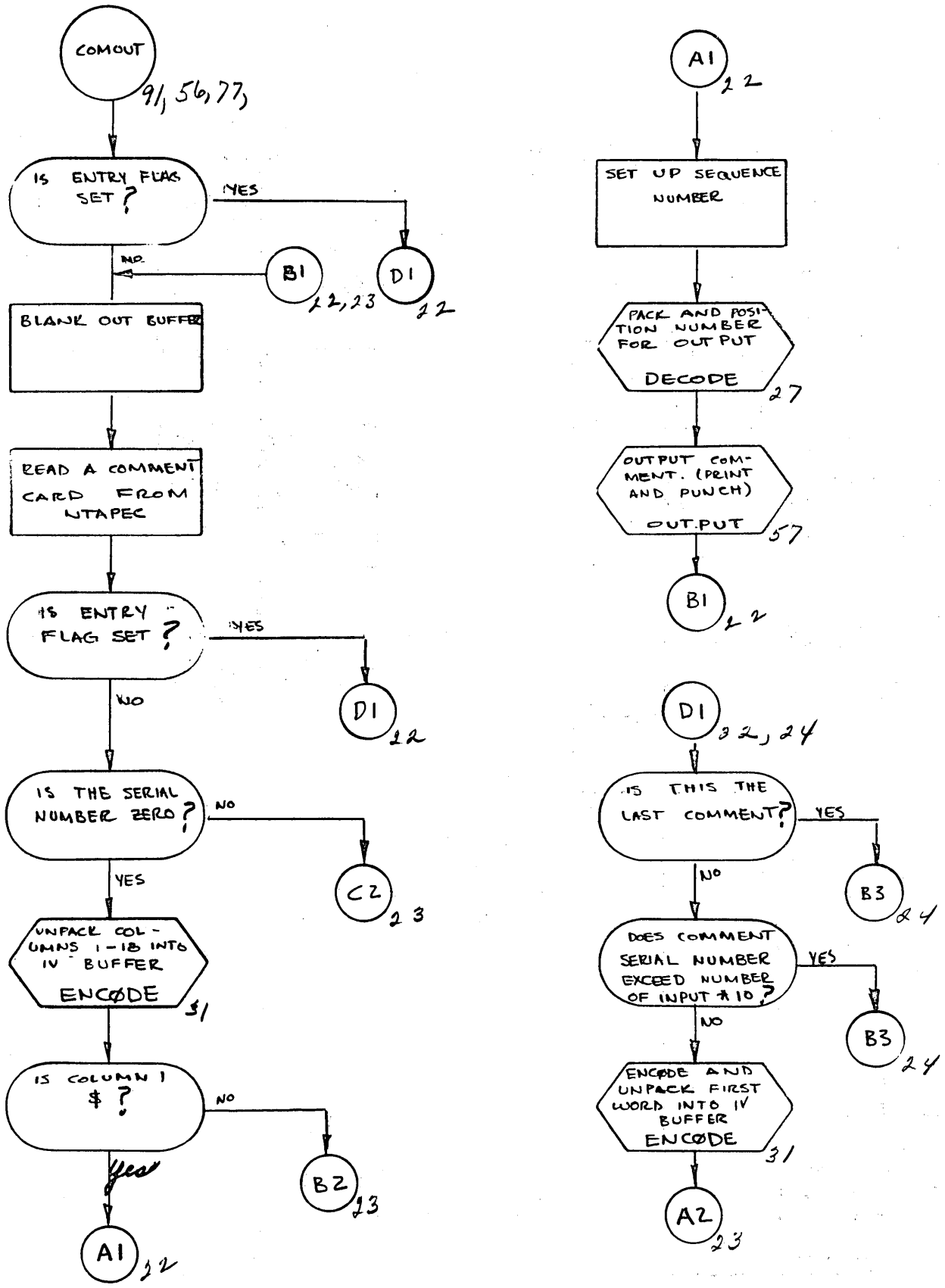
CLASSIFY STATEMENT, DETERMINE ITYPE, AND UPDATE IP TABLE



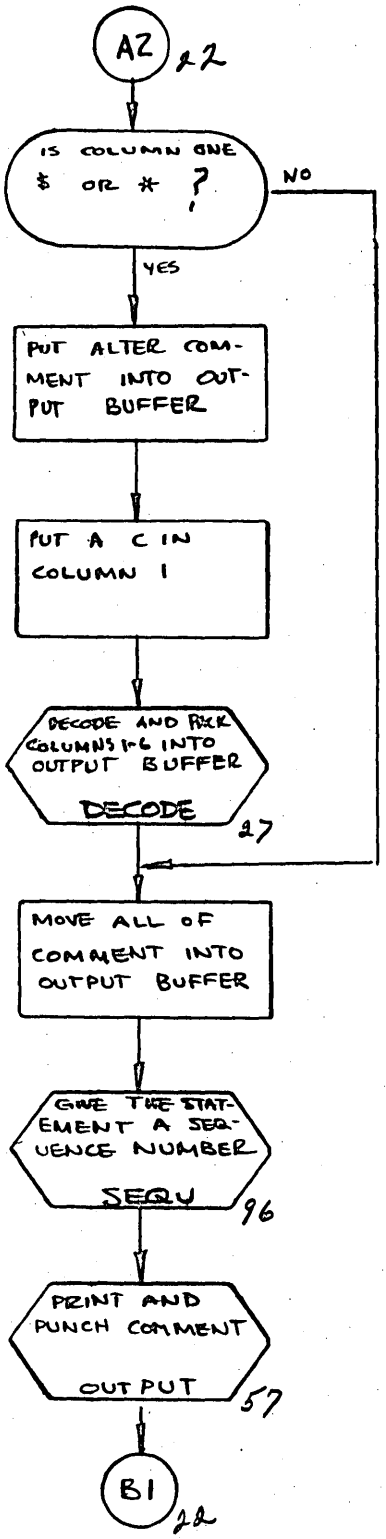
GIFT

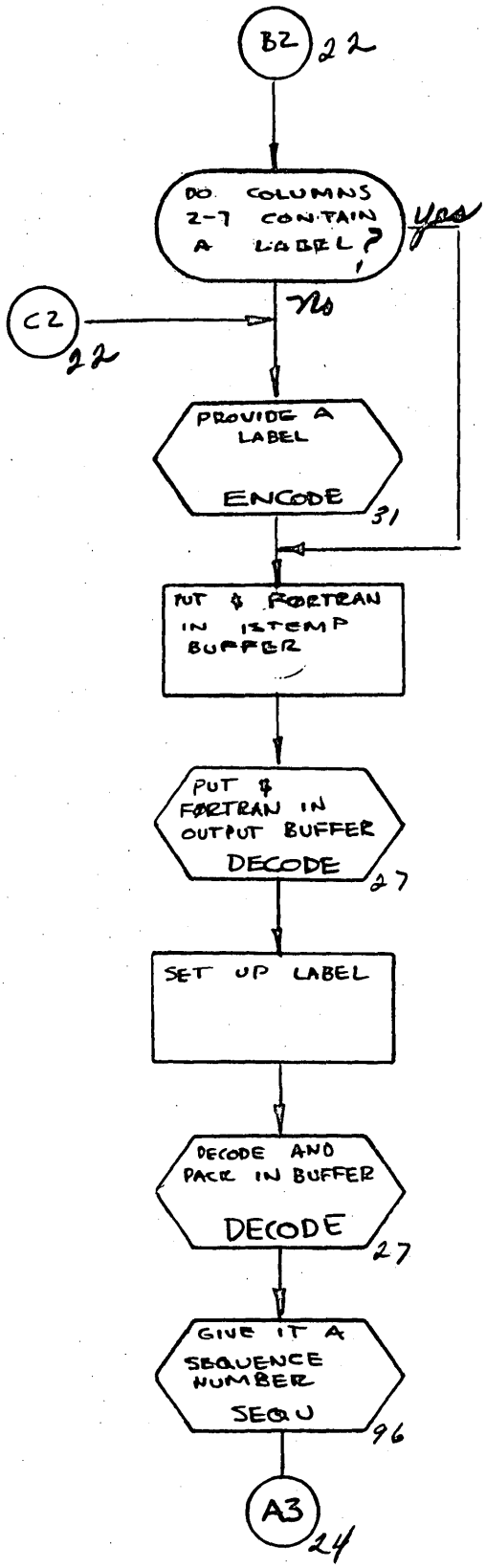
SUBROUTINE COMOUT

OUTPUT COMMENT CARDS FROM NTAPEC

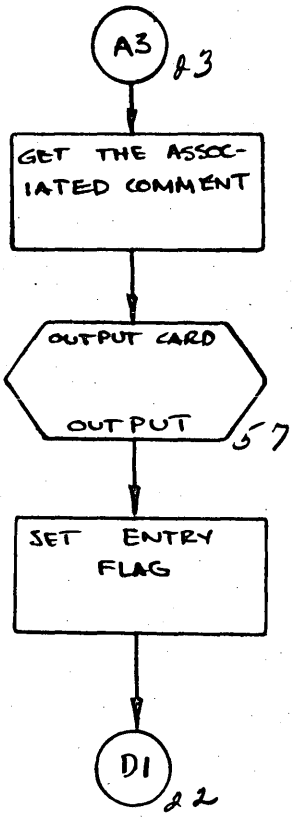


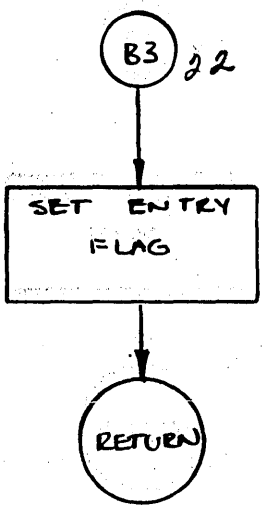
SUBROUTINE COMOUT





GIFT
SUBROUTINE COMOUT

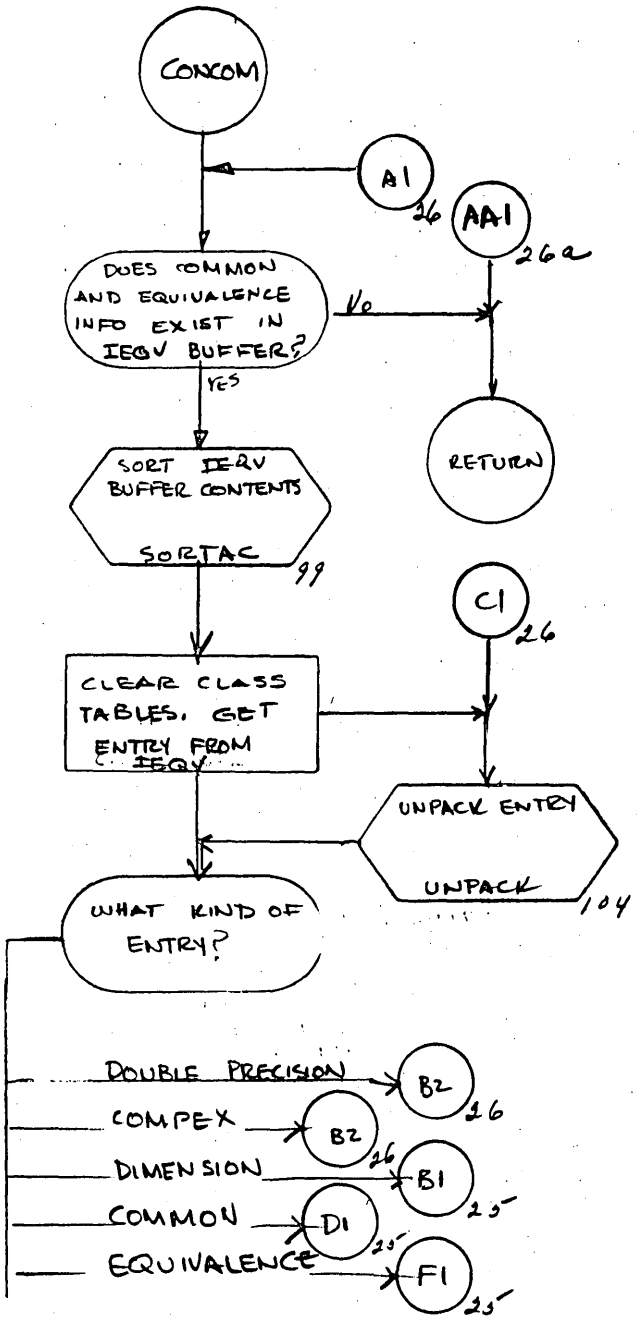


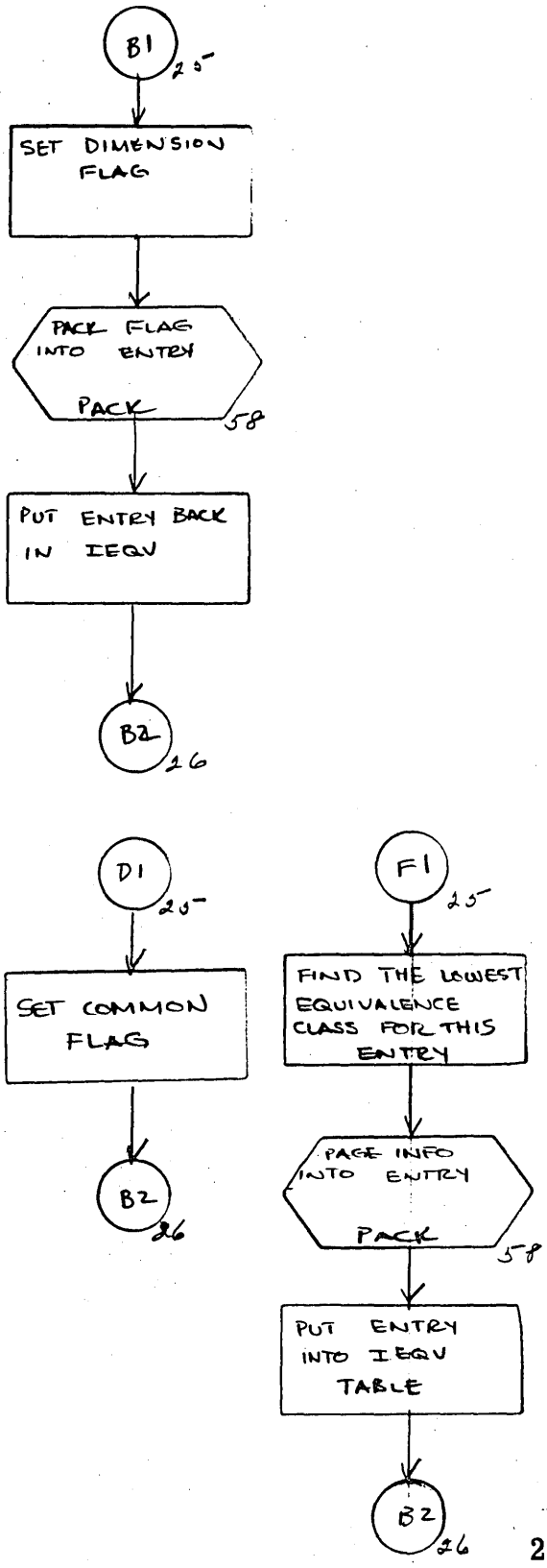


SIFT

SUBROUTINE CONCOM

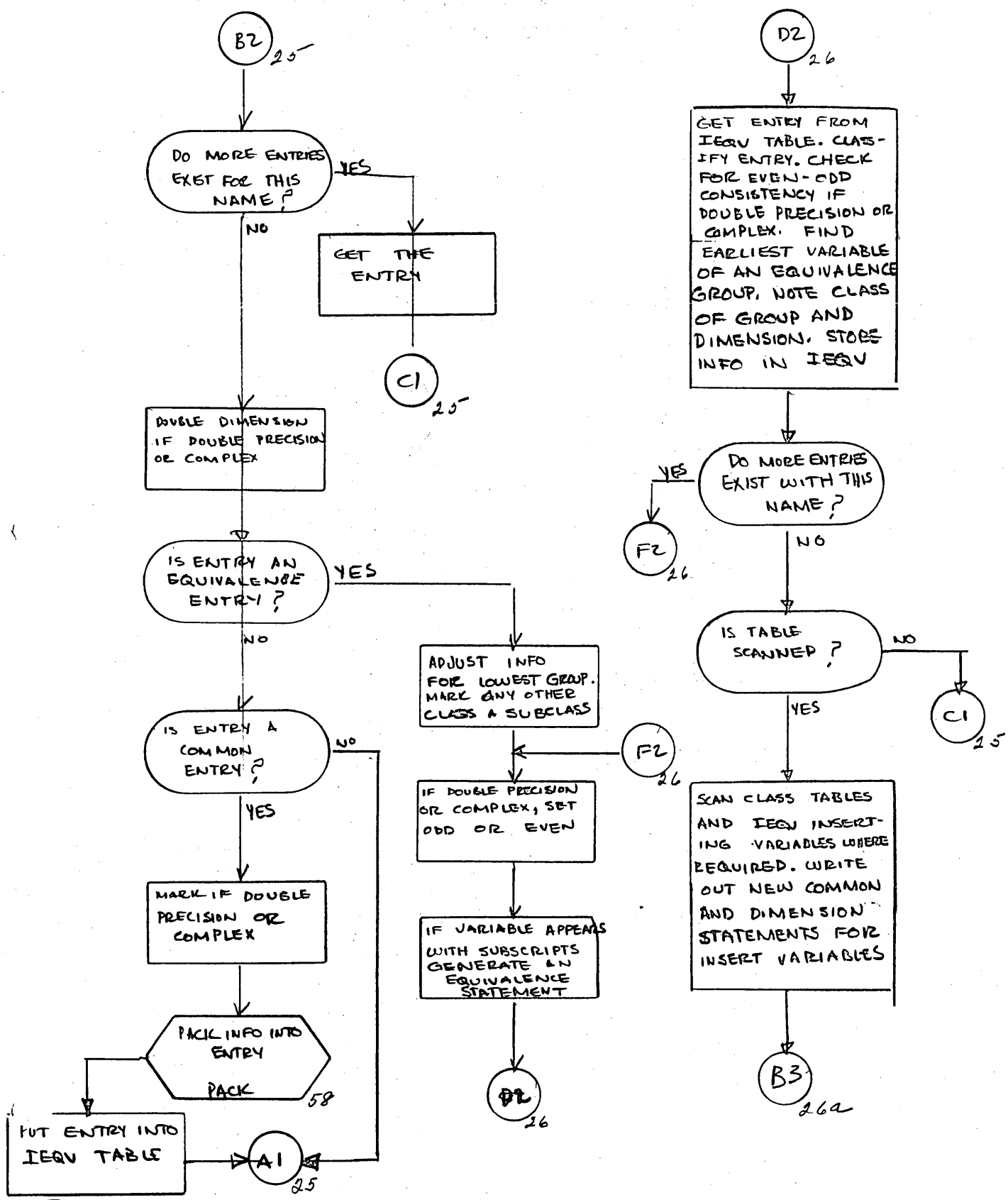
PROCESS EQV TABLE, CONSTRUCT NEW EQUIVALENCE, COMMON, DIMENSION, AND NECESSARY ARTIFICIAL VARIABLES. EQV TABLE WAS CONSTRUCTED BY SUBROUTINE EQV.



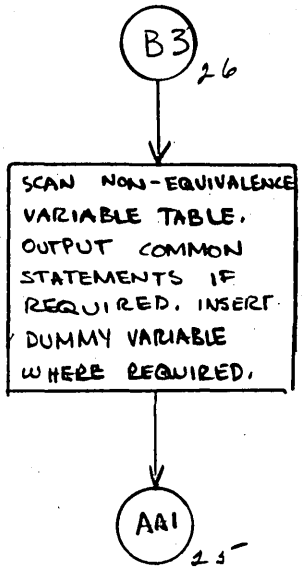


GIFT

SUBROUTINE CONCOM



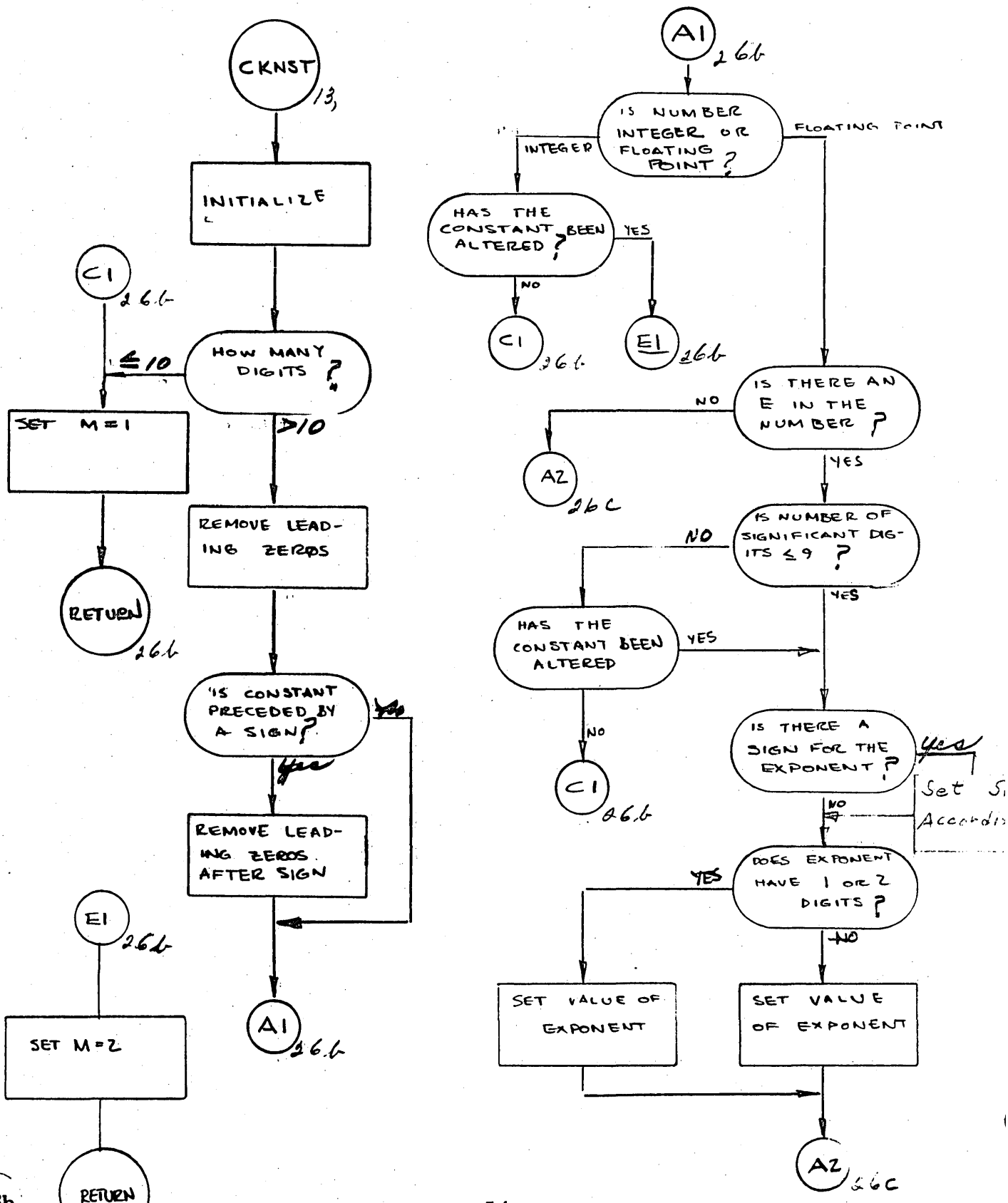
GIFT
SUBROUTINE CONCOM



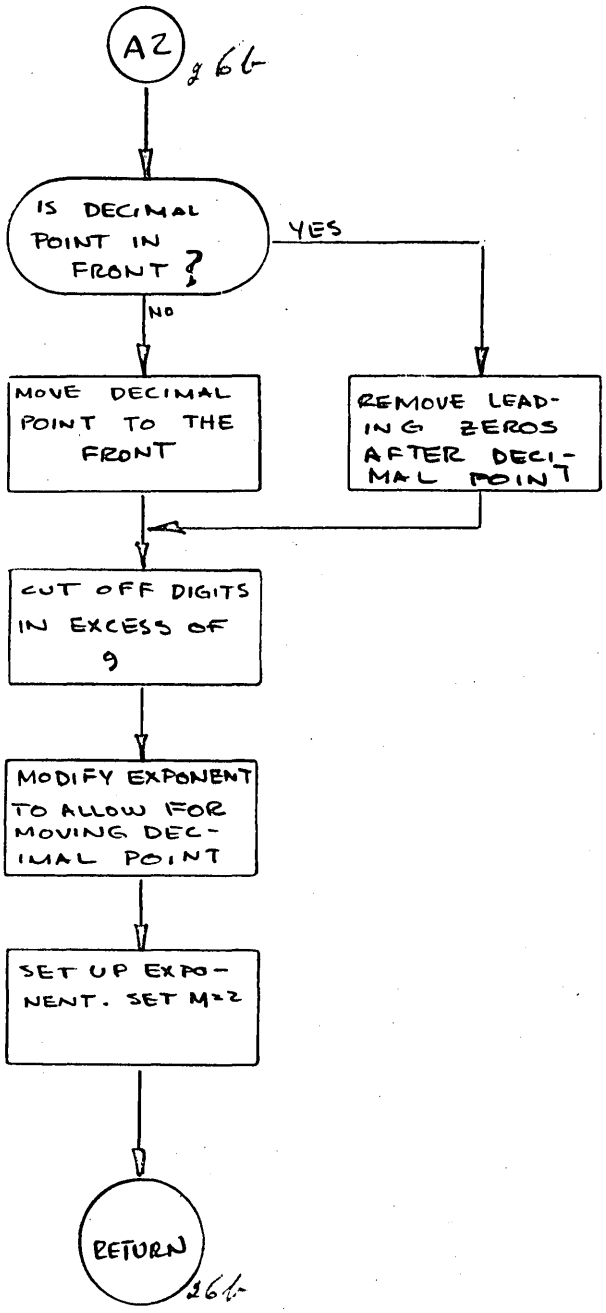
GIFT

SUBROUTINE CKNST

CHECK AND STRIP SINGLE PRECISION CONSTANTS



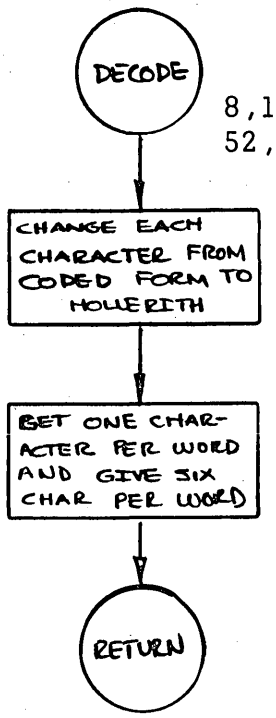
GIFT
SUBROUTINE CKNST



GIFT

SUBROUTINE DECODE

DECODE AND PACK INFORMATION

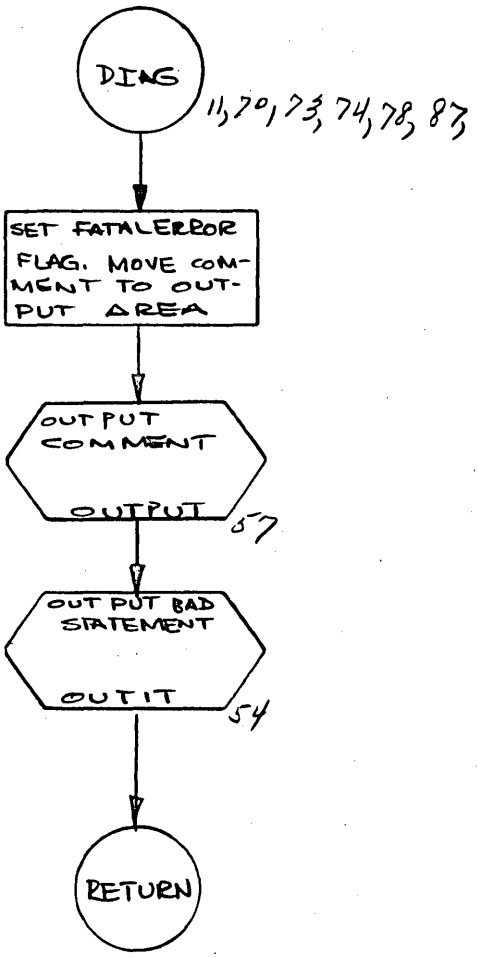


8,11,14,22,23,29,34,37,38,39,43,48,
52,56,65,68,75,79,82,83,88,92,97

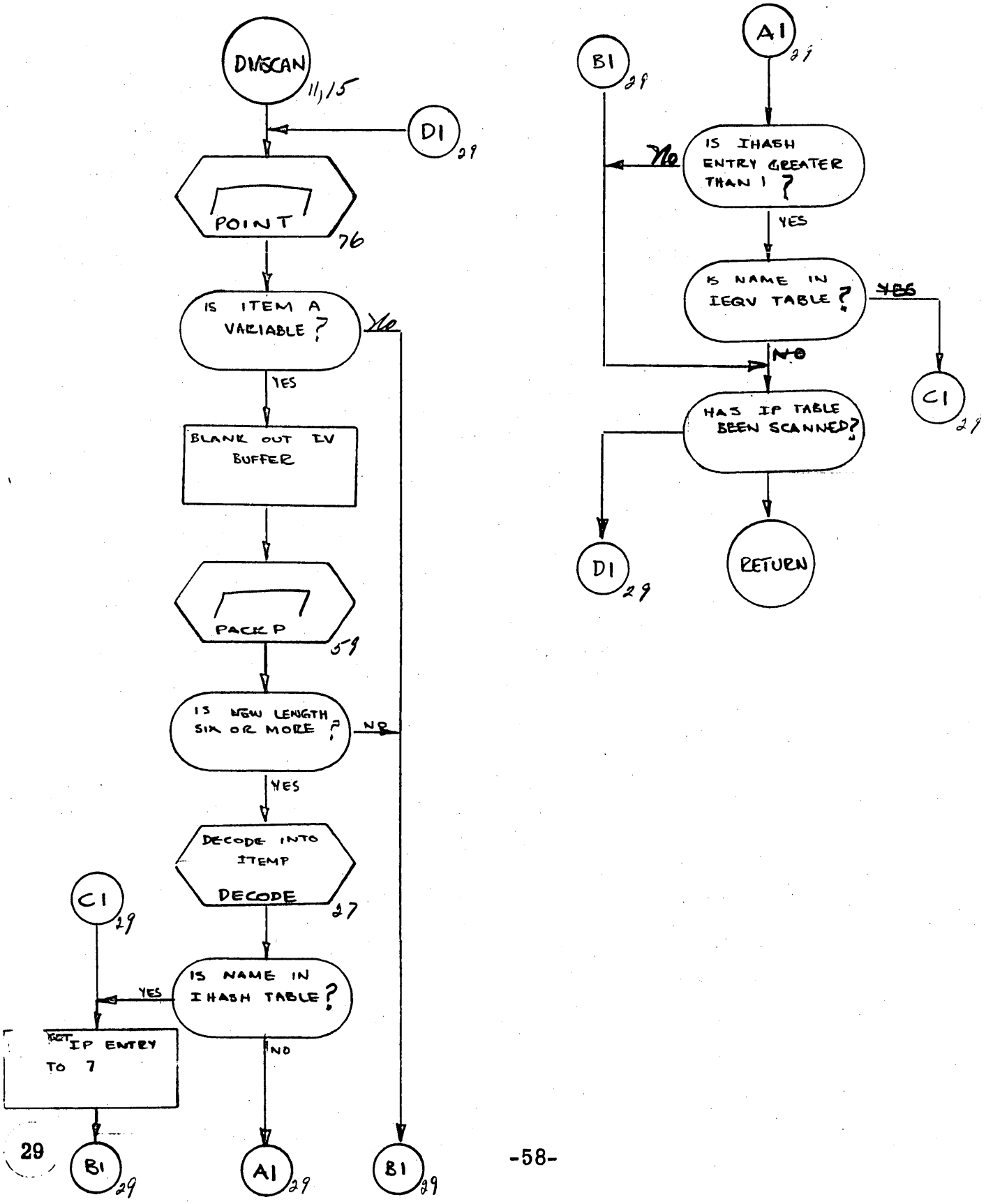
GIFT

SUBROUTINE DIAG

SIGNAL AN INCONSISTENCY IN STATEMENT



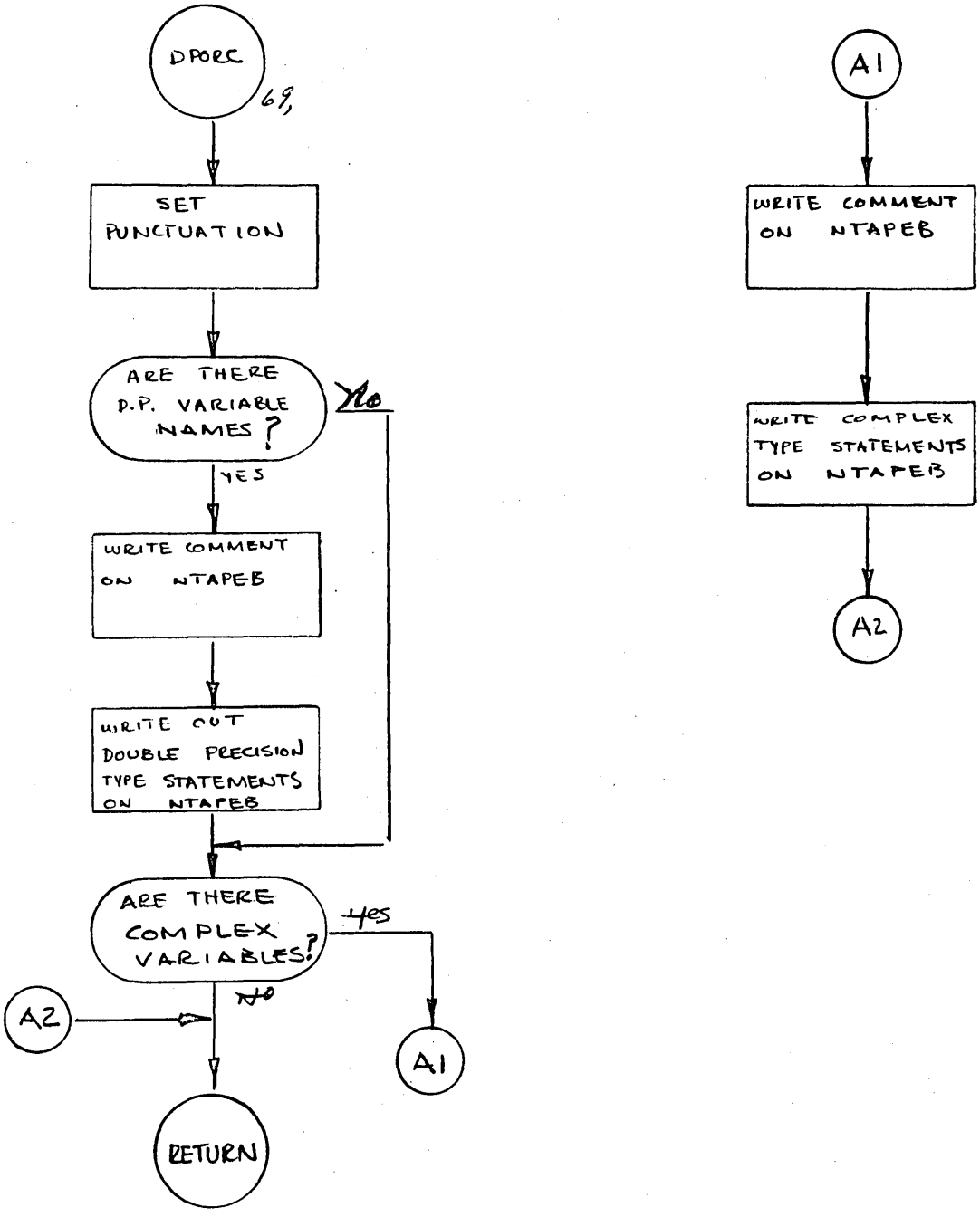
GIFT
SUBROUTINE DMSCAN
CHANGE IP CODE OF DIMENSIONED VARIABLES FROM 1 TO 7



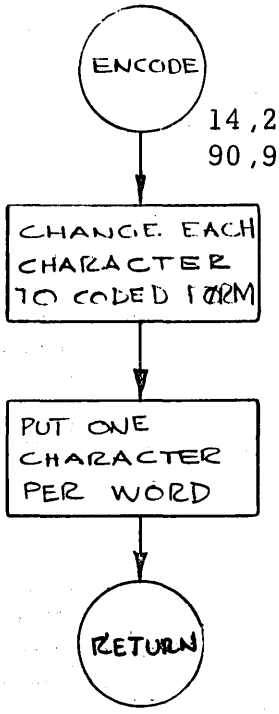
GIFT

SUBROUTINE DPORC

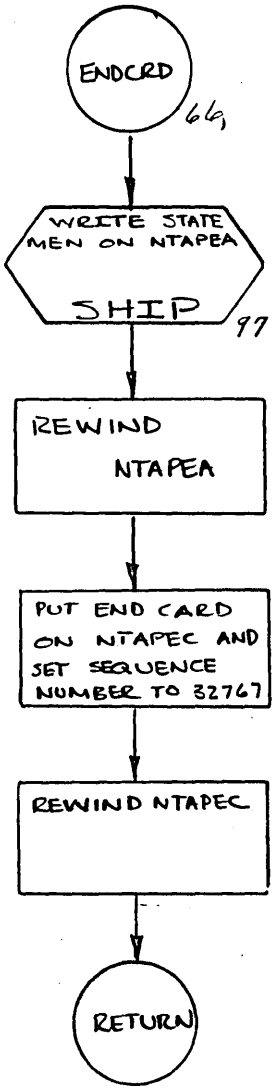
CONSTRUCT TYPE STATEMENTS FOR DOUBLE PRECISION AND/OR COMPLEX VARIABLE NAMES



GIFT
ENCODE
CODE AND UNPACK



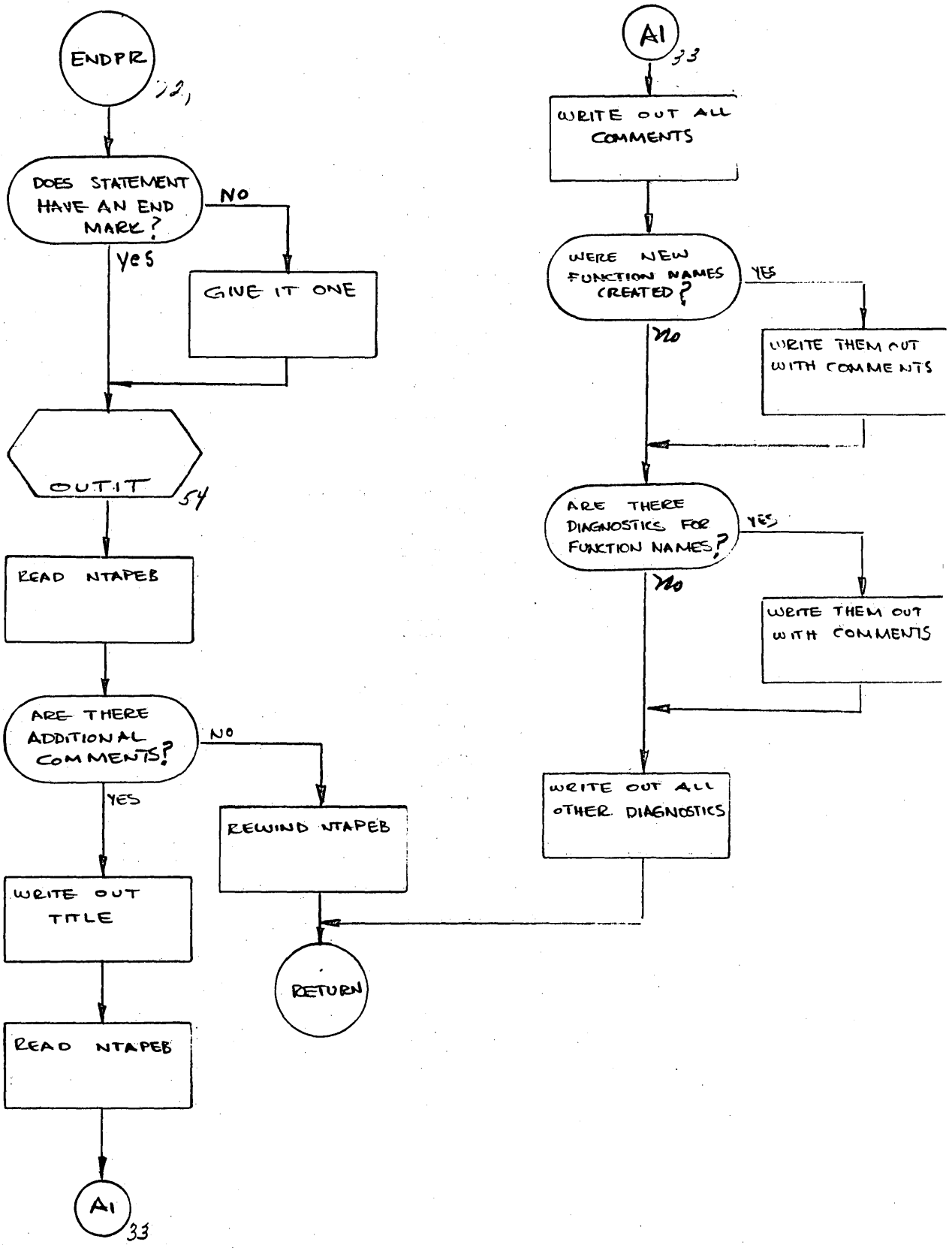
GIFT
SUBROUTINE ENDCRD
PROCESS END CARD



GIFT

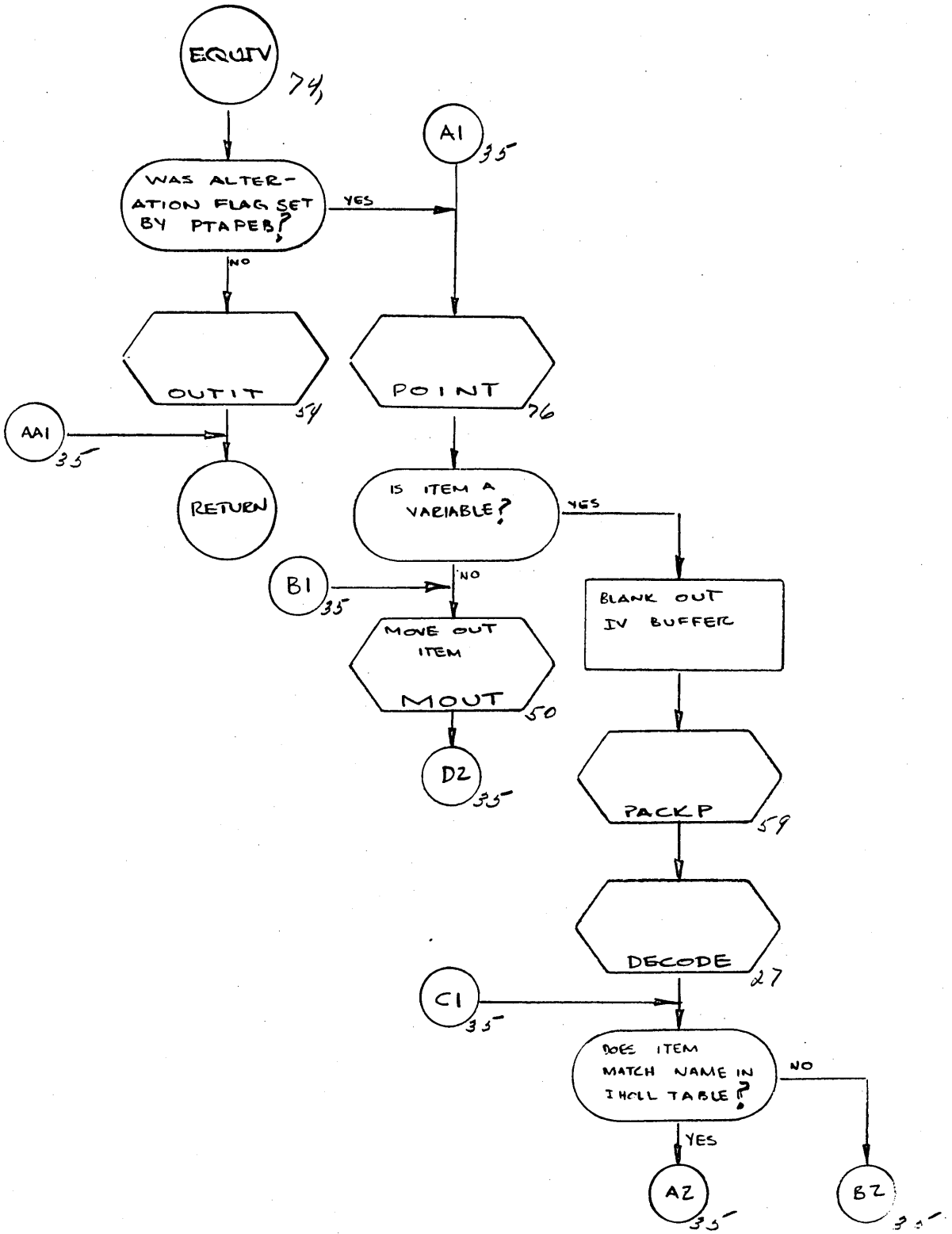
SUBROUTINE ENDPR

PROCESS END CARD, OUTPUT FINAL COMMENTS

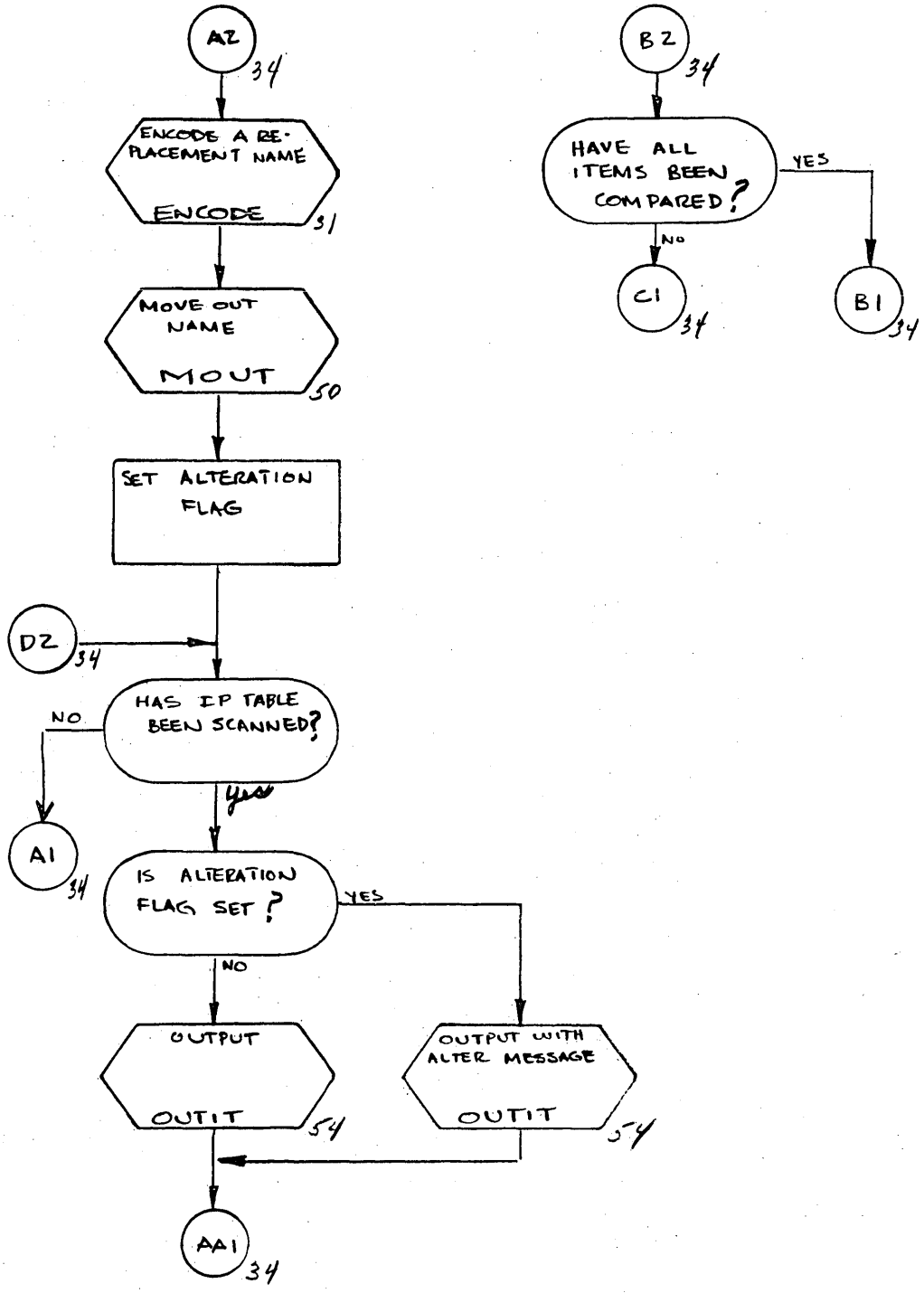


GIFT
SUBROUTINE EQUIV

REPLACE NAMES IN EQUIVALENCE STATEMENTS TO EQUATE DOUBLE PRECISION AND INTEGER VARIABLES



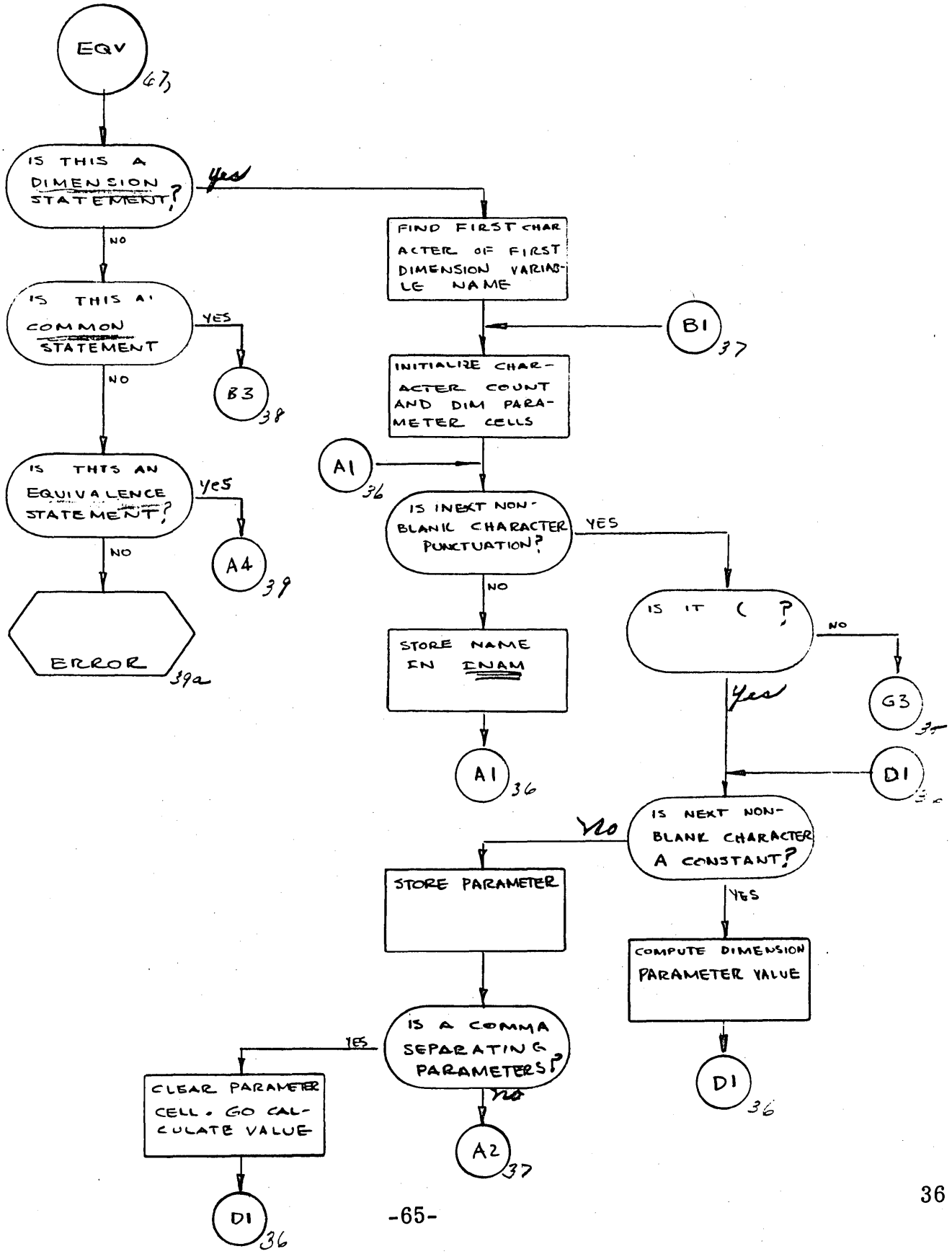
GIFT
SUBROUTINE EQUIV

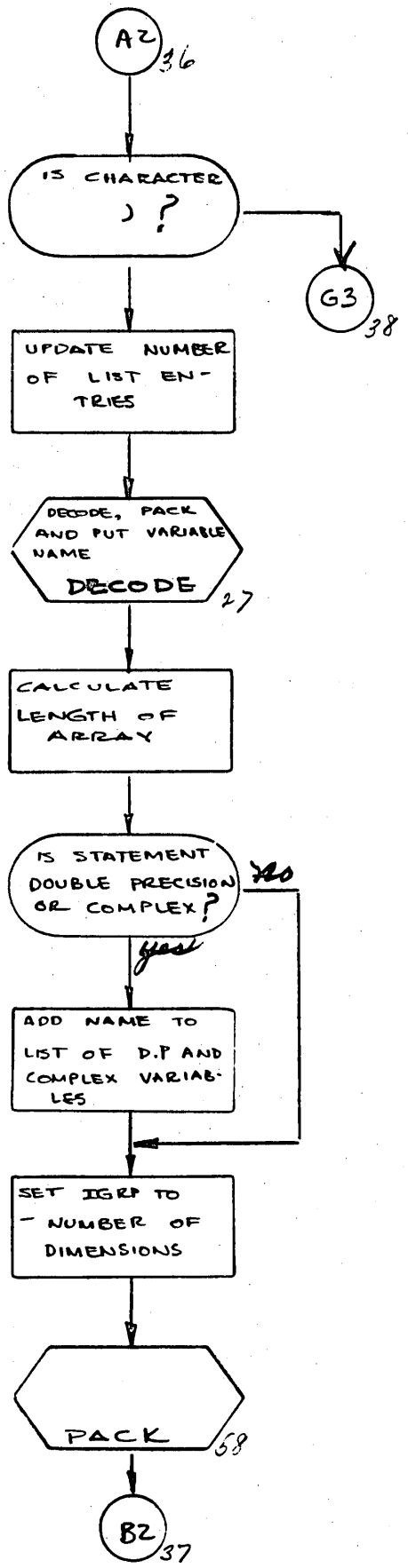


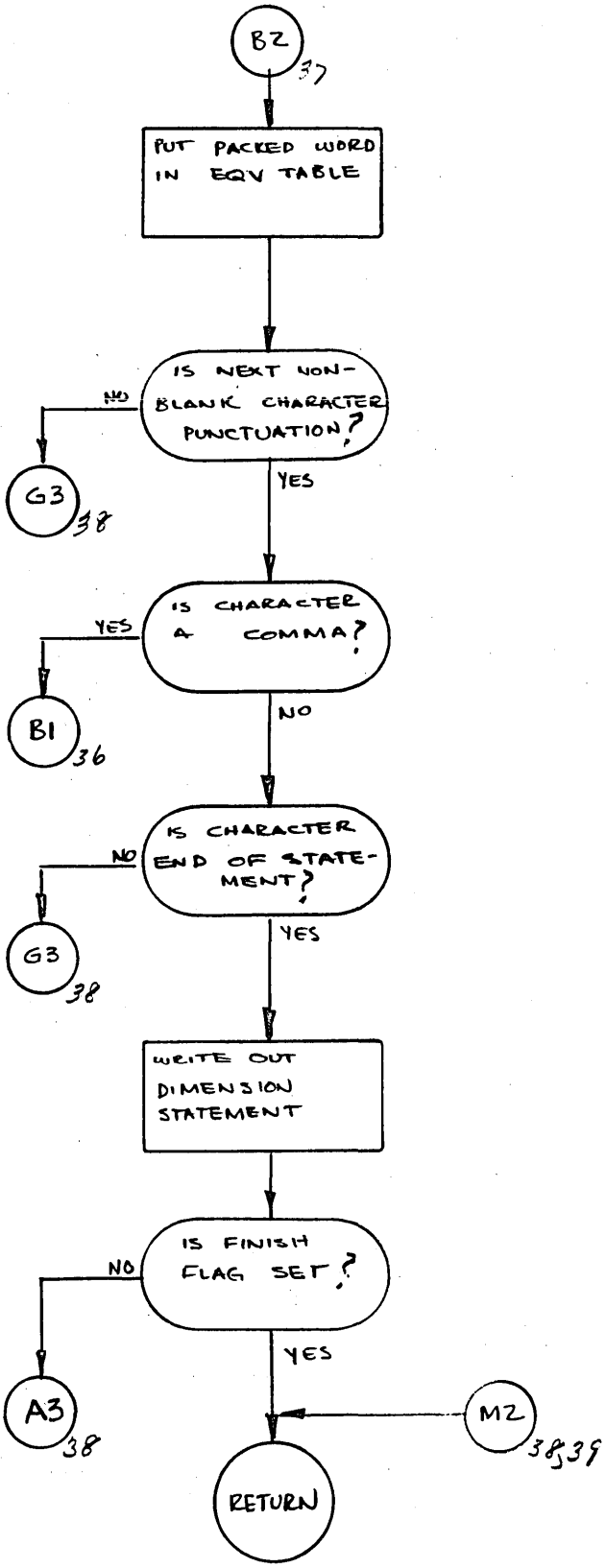
GIFT

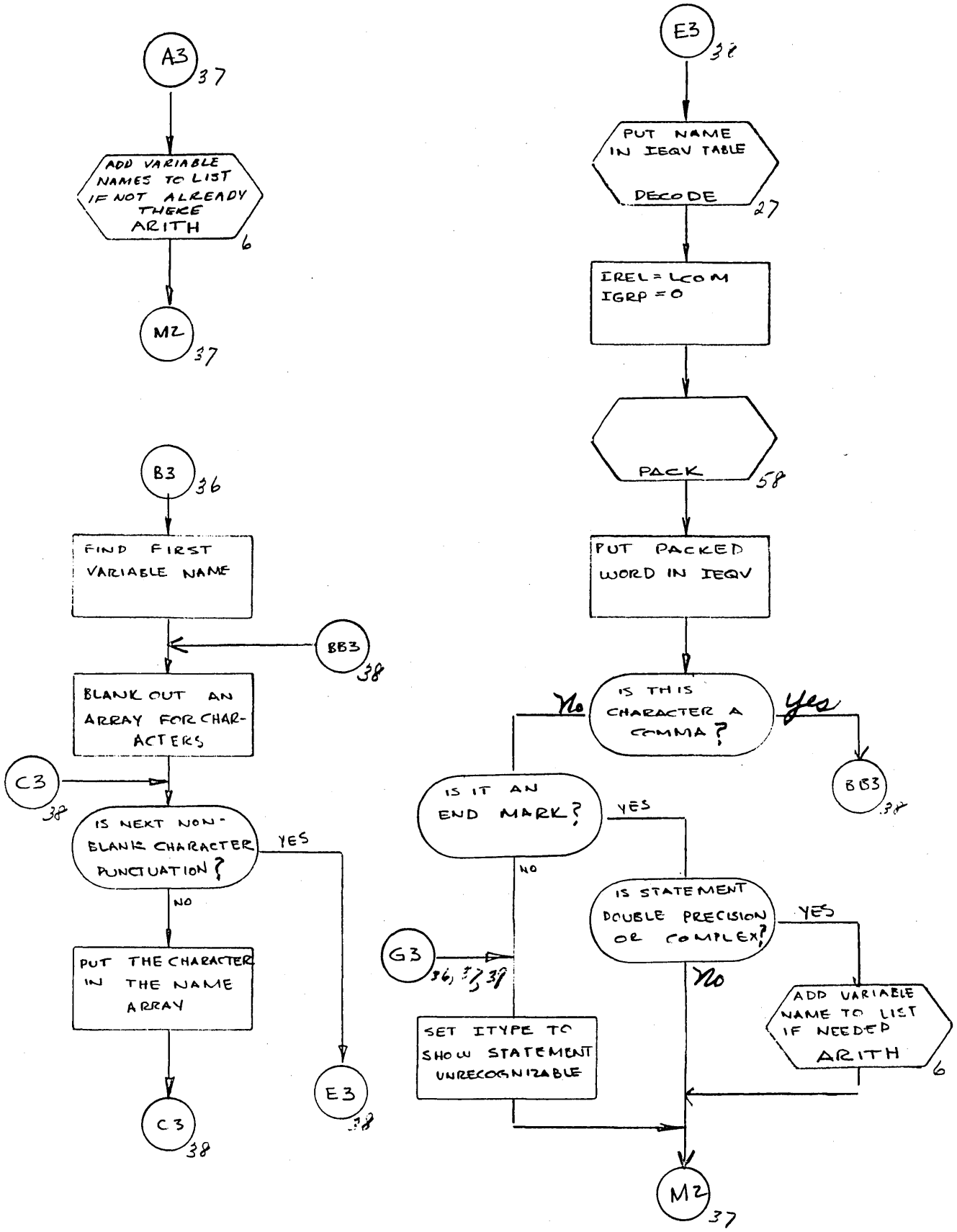
SUBROUTINE EQV

TABULATE DIMENSION, EQUIVALENCE, AND COMMON INFORMATION FOR PROCESSING BY SUBROUTINE CONCOM

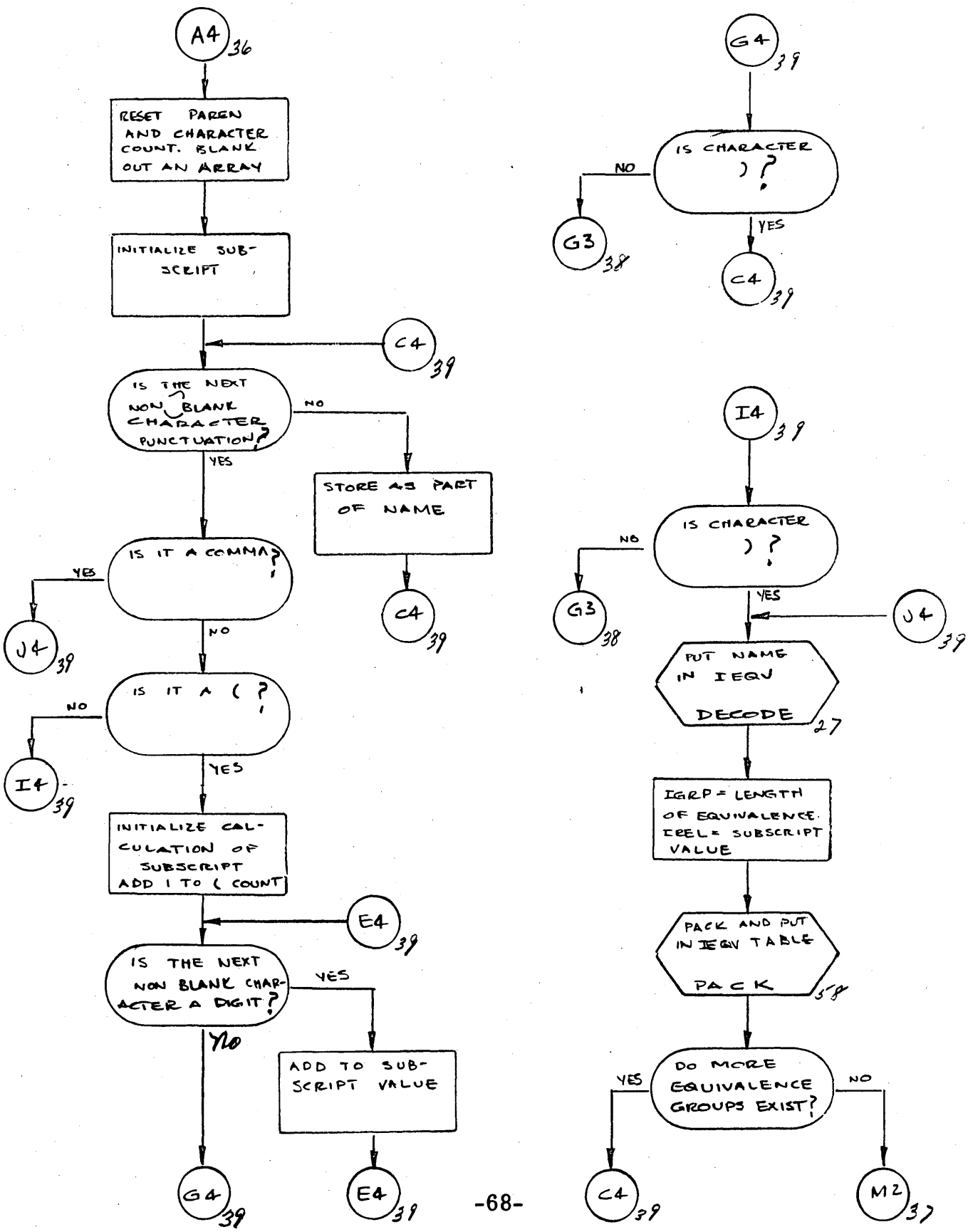



A2
36
IS CHARACTER) ?
G3
38
UPDATE NUMBER OF LIST ENTRIES
DECODE, PACK AND PUT VARIABLE NAME
DECODE
27
CALCULATE LENGTH OF ARRAY
IS STATEMENT DOUBLE PRECISION OR COMPLEX?
NO
YES
ADD NAME TO LIST OF D.P AND COMPLEX VARIABLES
SET IGRP TO - NUMBER OF DIMENSIONS
PACK
68
B2
37

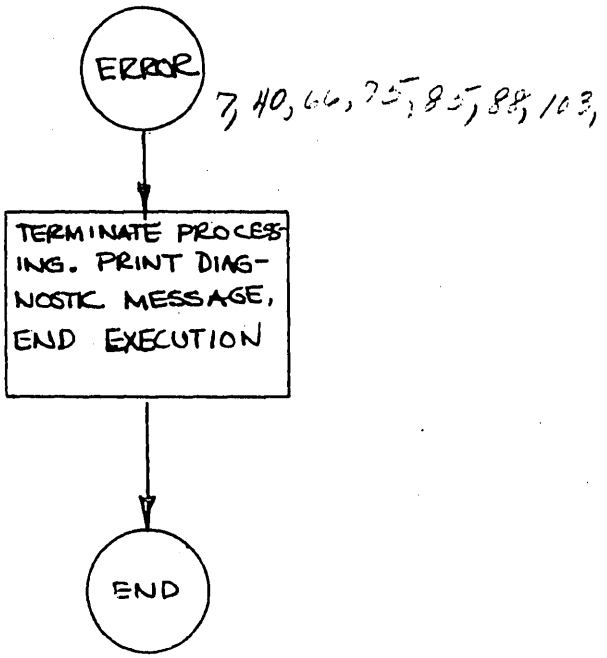

B2
37
PUT PACKED WORD IN EQV TABLE
IS NEXT NON-BLANK CHARACTER PUNCTUATION?
NO
G3
38
YES
IS CHARACTER A COMMA?
YES
B1
36
NO
IS CHARACTER END OF STATEMENT?
NO
G3
38
YES
WRITE OUT DIMENSION STATEMENT
IS FINISH FLAG SET?
NO
A3
38
YES
M2
38,39
RETURN


A3
37
ADD VARIABLE NAMES TO LIST IF NOT ALREADY THERE
ARITH
6
M2
37
B3
36
FIND FIRST VARIABLE NAME
BB3
38
BLANK OUT AN ARRAY FOR CHARACTERS
C3
38
IS NEXT NON-BLANK CHARACTER PUNCTUATION?
YES
NO
PUT THE CHARACTER IN THE NAME ARRAY
C3
38
E3
38
E3
38
PUT NAME IN IEQV TABLE
DECODE
27
IREL = LCOM
IGRP = 0
PACK
58
PUT PACKED WORD IN IEQV
IS THIS CHARACTER A COMMA?
No
Yes
BB3
38
IS IT AN END MARK?
YES
NO
G3
36, 37, 39
IS STATEMENT DOUBLE PRECISION OR COMPLEX?
YES
No
ADD VARIABLE NAME TO LIST IF NEEDED
ARITH
6
SET ITYPE TO SHOW STATEMENT UNRECOGNIZABLE
M2
37

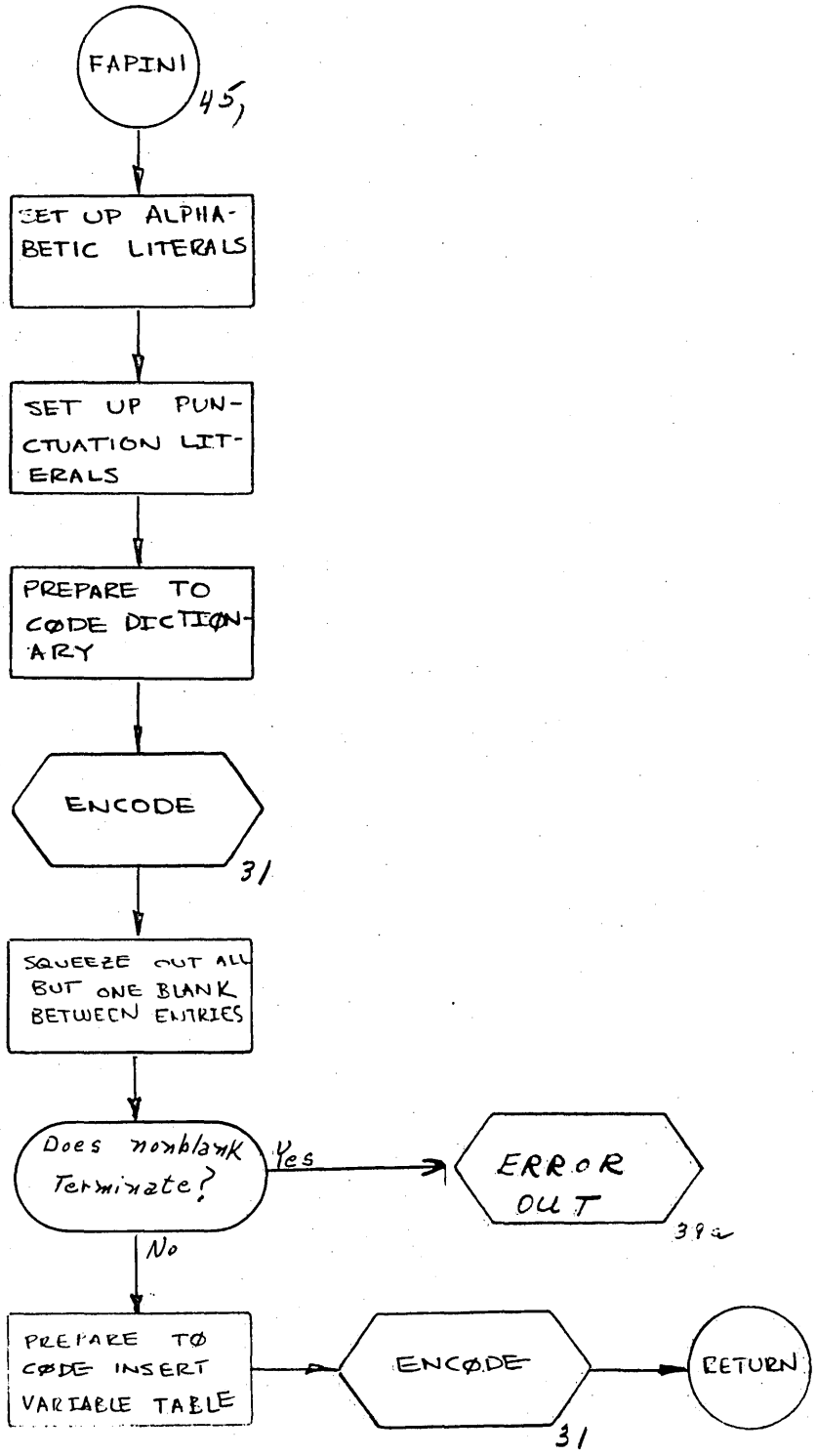
GIFT
SUBROUTINE EQV



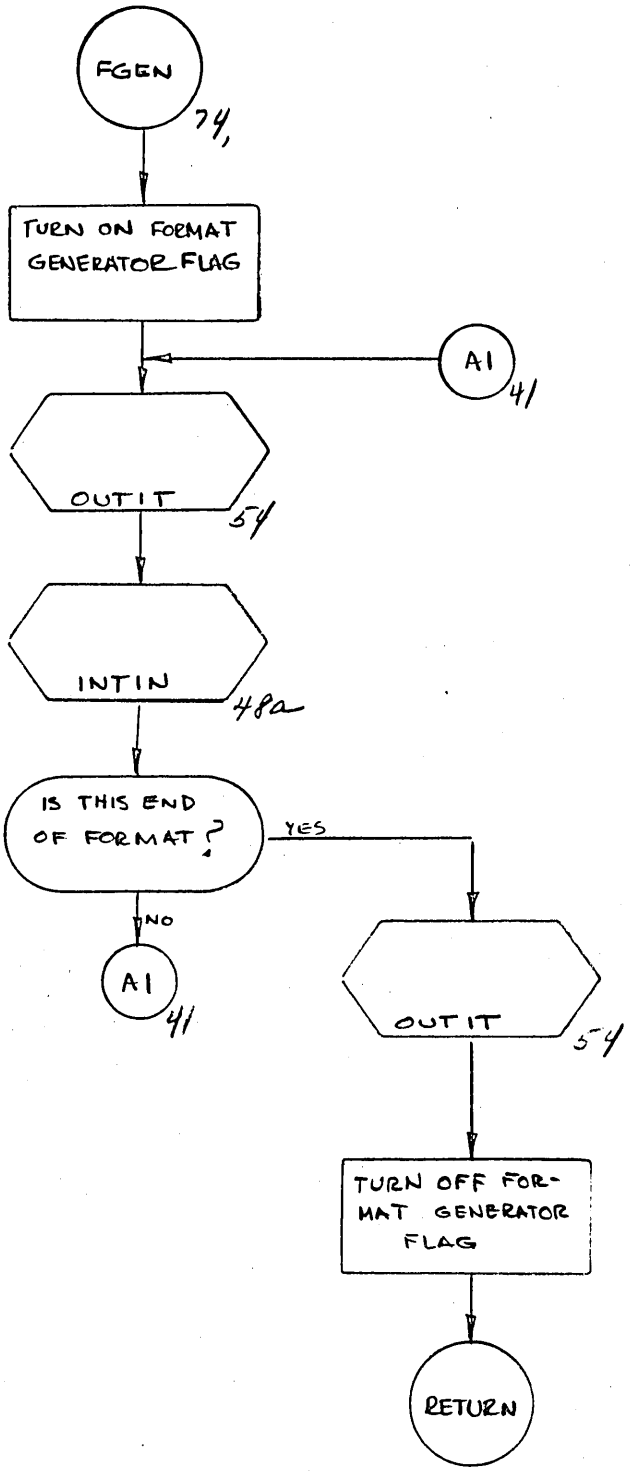
GIFT
SUBROUTINE ERROR



GIFT
FAPIN 1
INITIALIZE LITERALS AND DICTIONARY



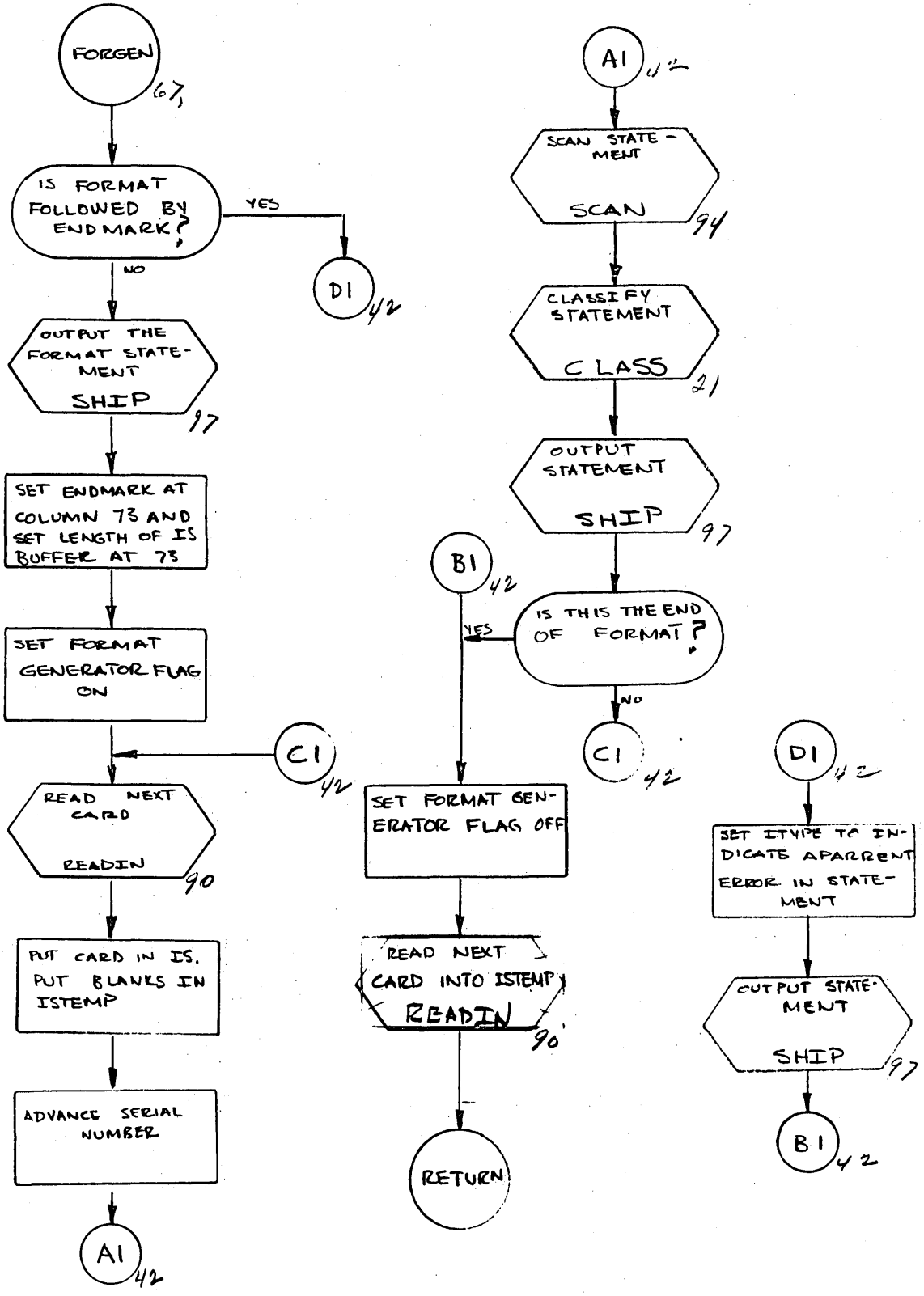
GIFT
SUBROUTINE FGEN
PROCESS STATEMENTS FOR FORMAT GENERATOR



GIFT

SUBROUTINE FORGEN

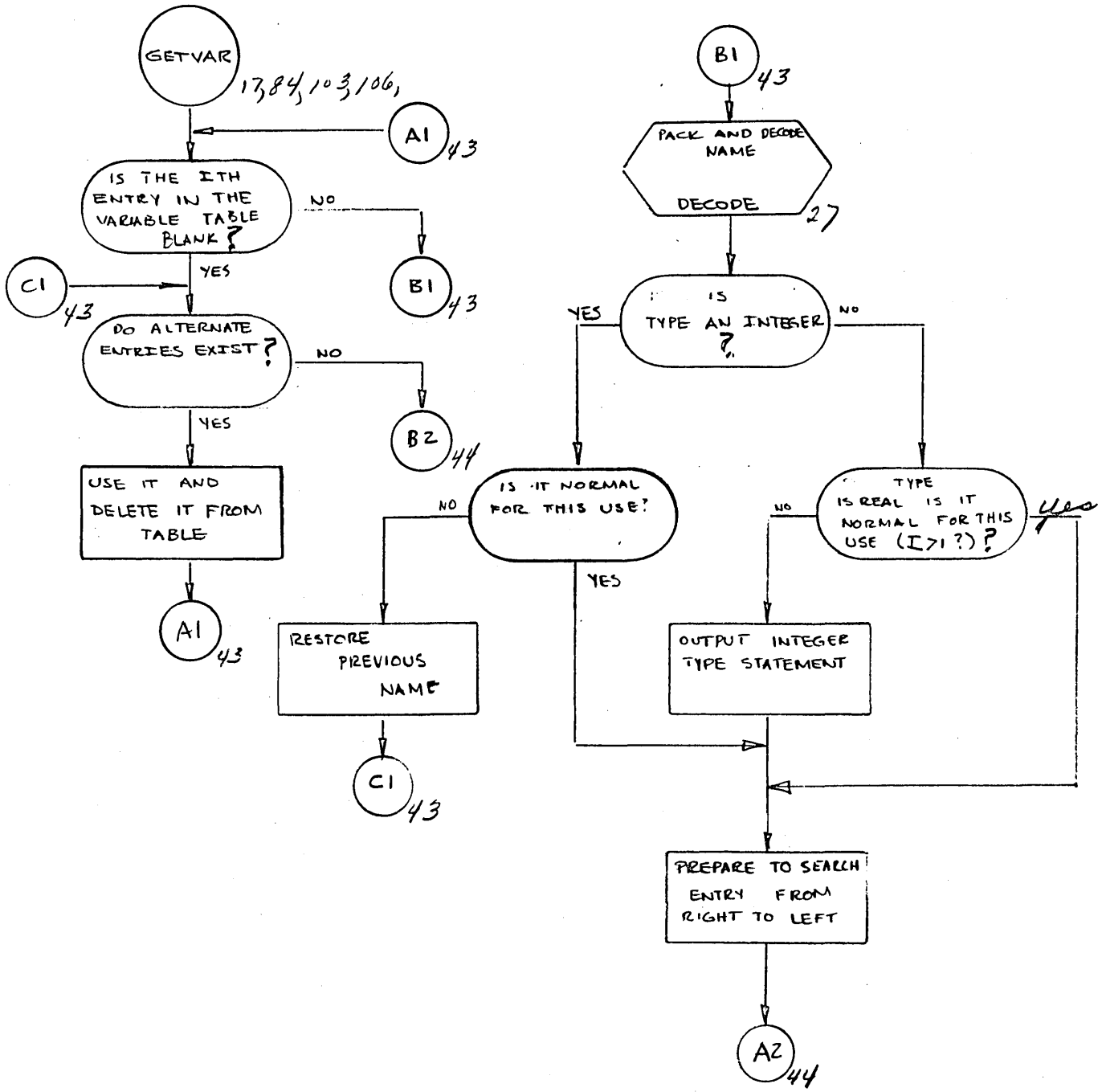
PROCESS STATEMENTS INTENDED FOR FORMAT GENERATOR



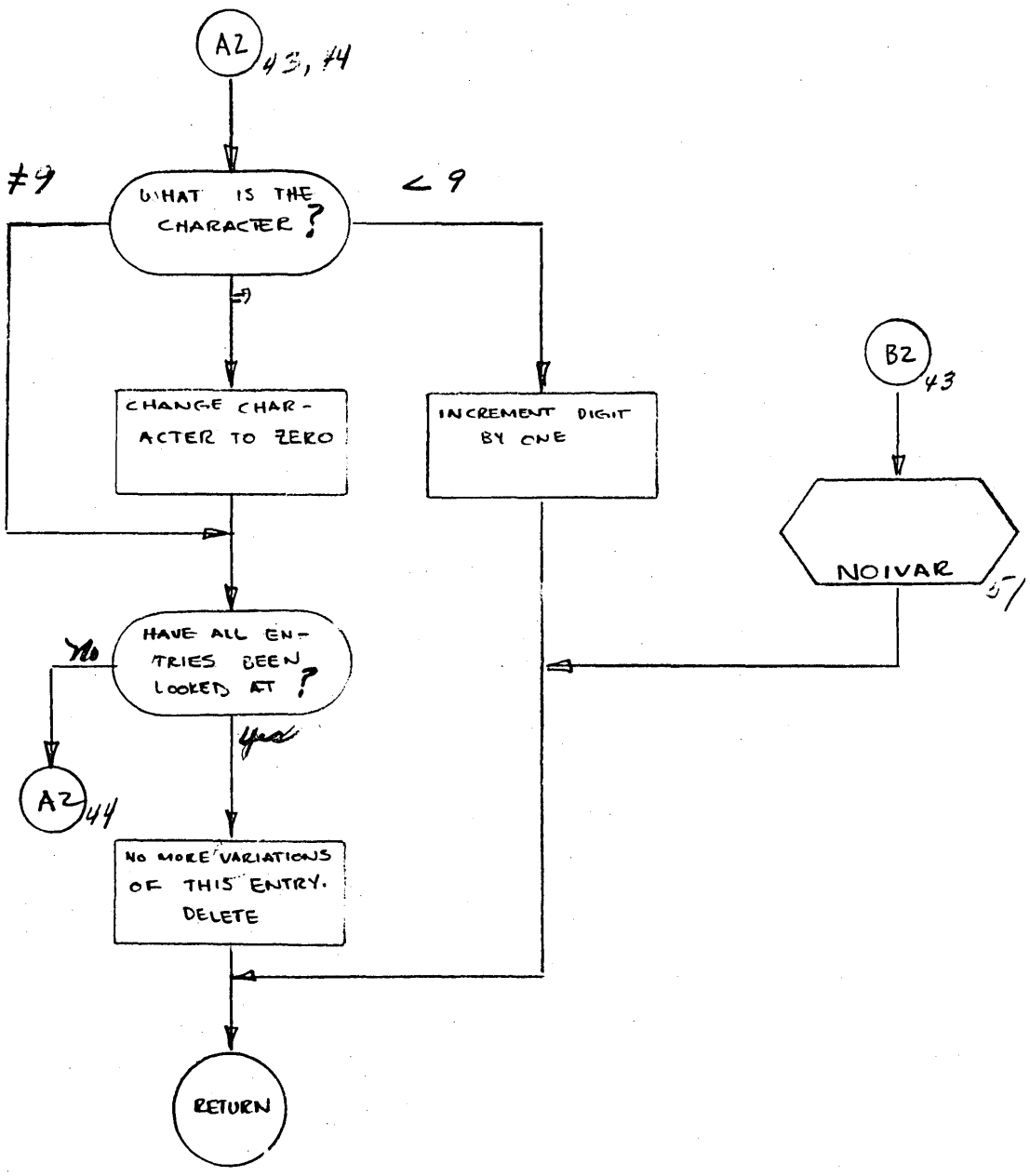
GIFT

SUBROUTINE GETVAR

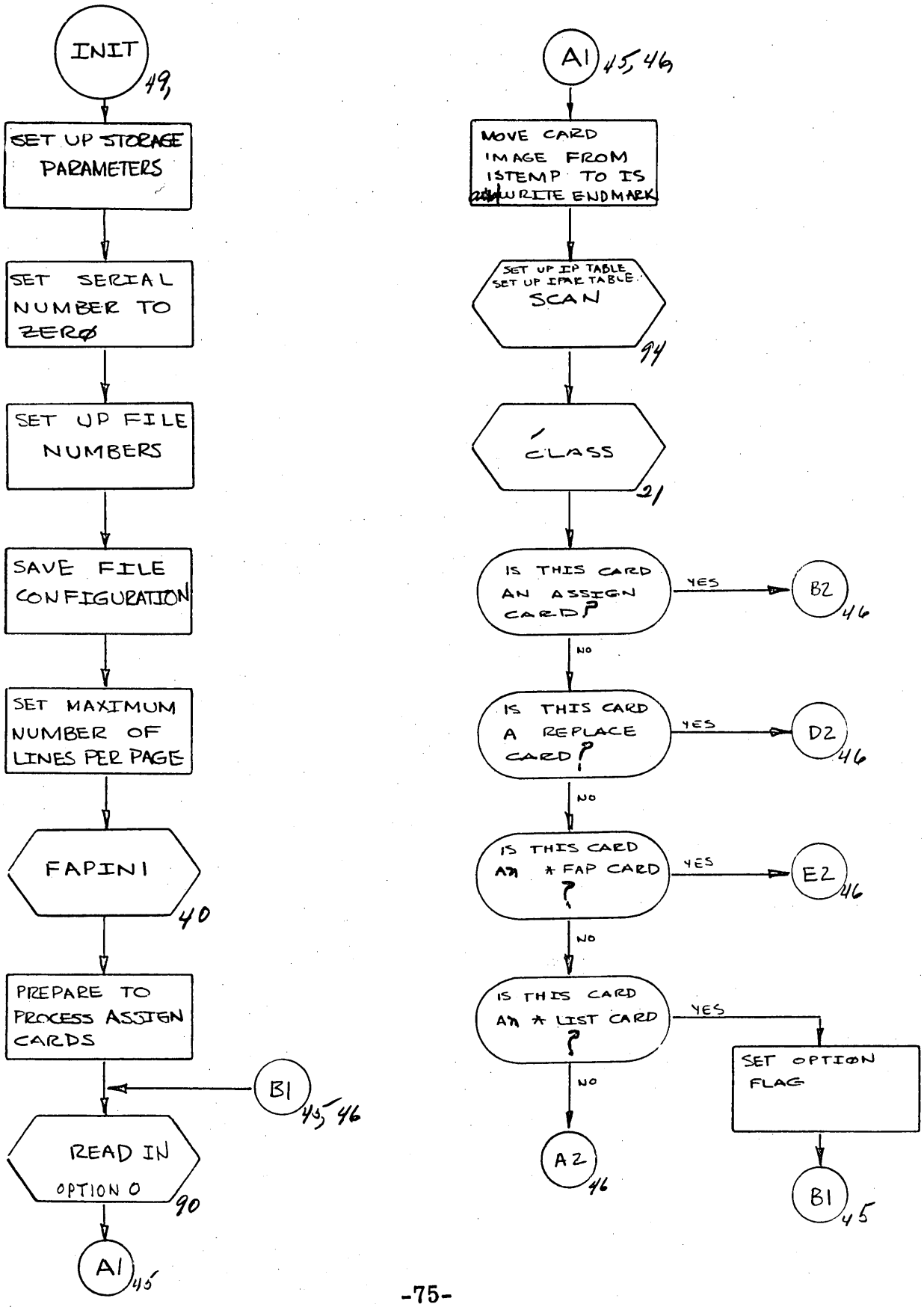
GETS NEXT AVAILABLE INSERT VARIABLE OF SPECIFIED TYPE AND RETURNS PACKED AND DECODED NAME. OUTPUTS TYPE STATEMENT (REAL OR INTEGER) IF TYPE OF INSERT VARIABLE IS NOT NORMAL FOR KIND OF ITEM TO BE REPLACED. CALLS SUBROUTINE NOIVAR IF NO MORE NAMES EXIST



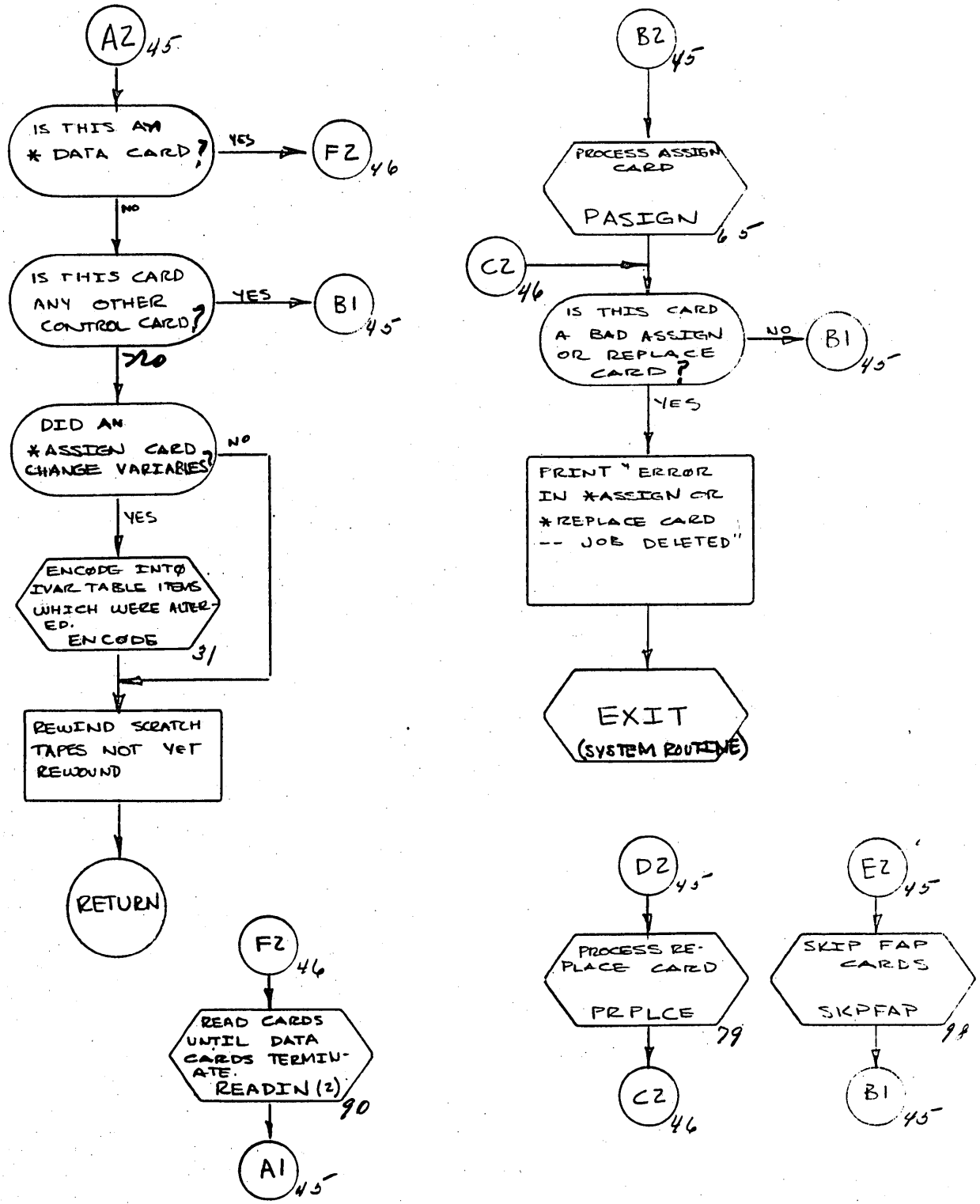
GIFT
SUBROUTINE GETVAR



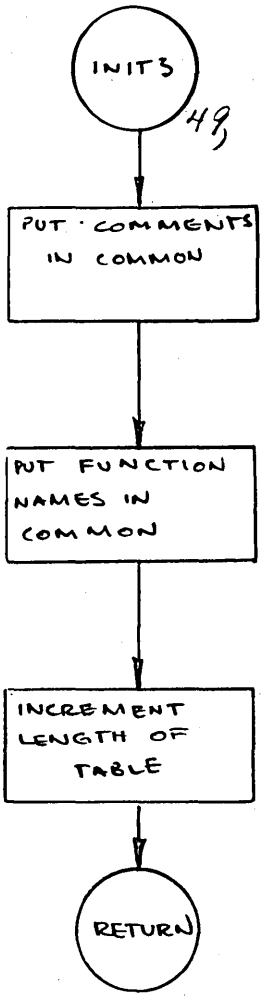
GIFT
INIT
INITIALIZE FOR INPUT



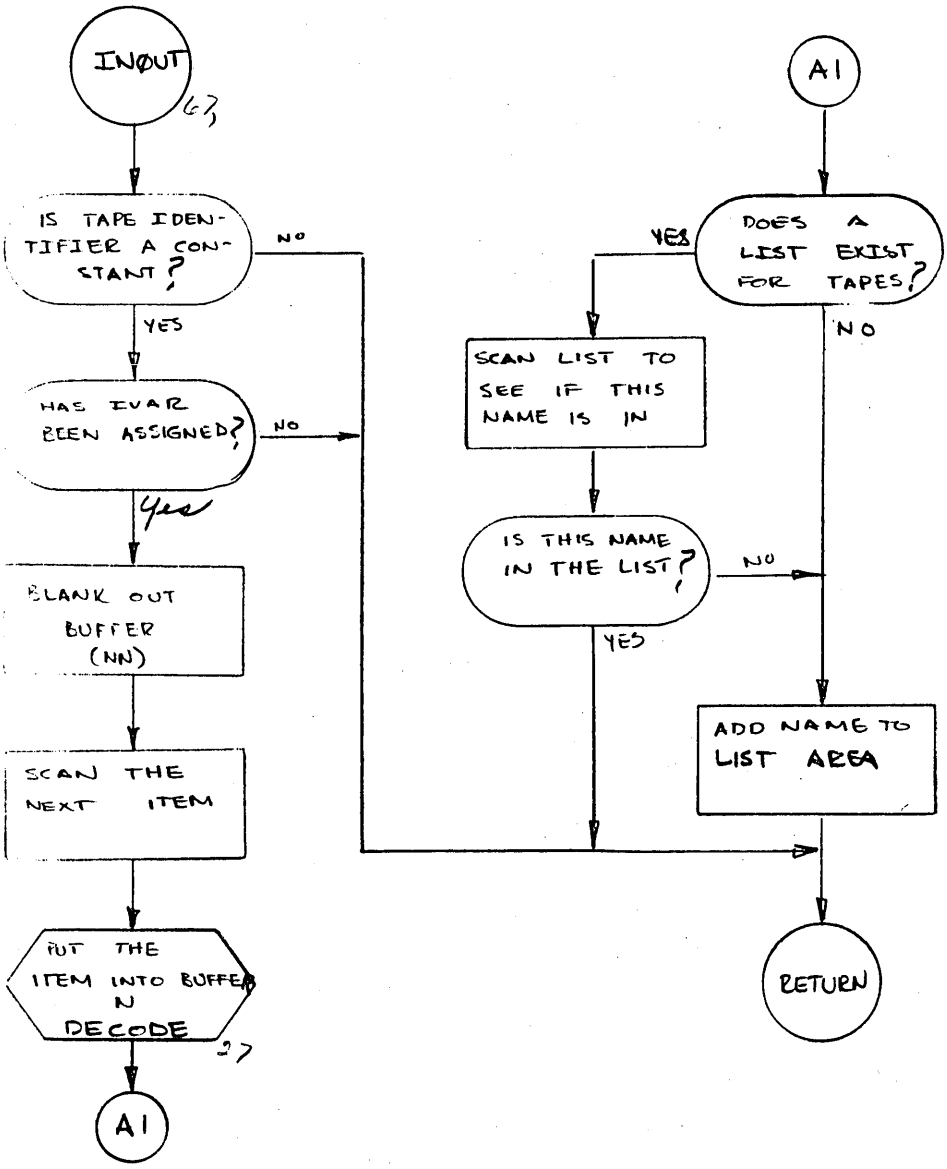
GIFT
SUBROUTINE INIT
INITIALIE FOR INPUT



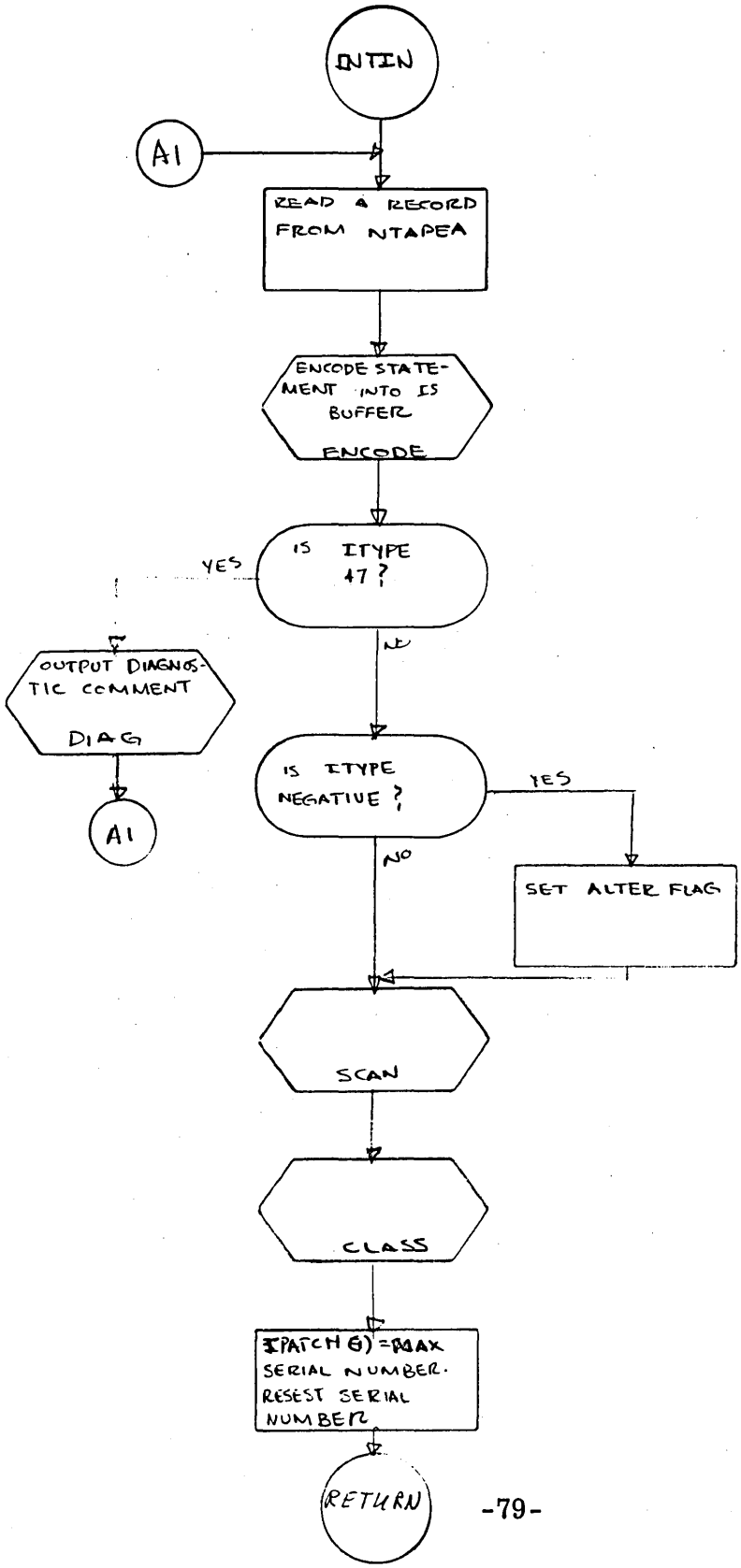
GIFT
SUBROUTINE INIT3
INITIALIZE FOR FAP3



GIFT
SUBROUTINE INOUT
PROCESS INPUT/OUTPUT STATEMENTS

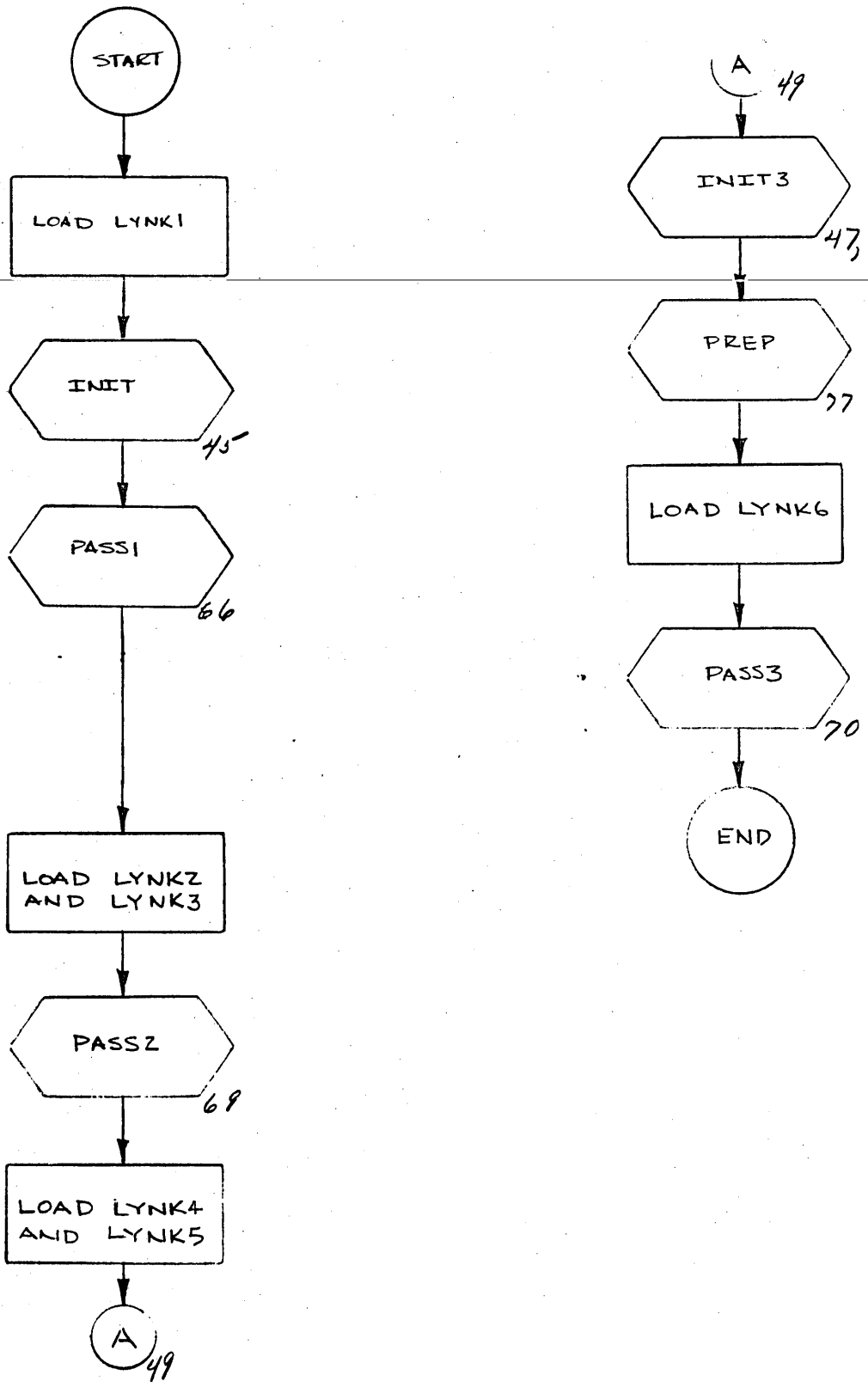


GIFT
SUBROUTINE INTIN
READ NEXT STATEMENT FROM SHIP TAPE (NTAPEA)

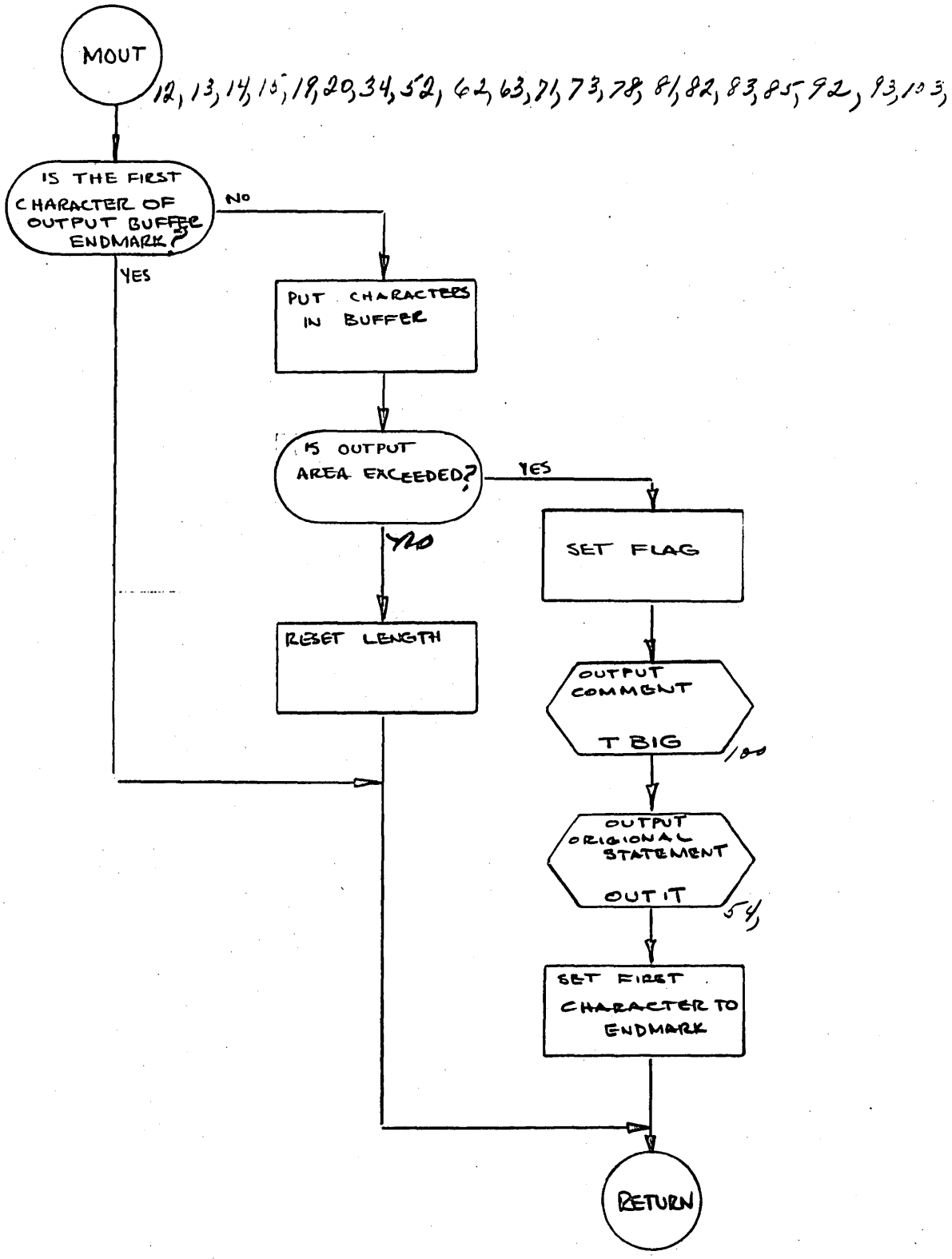


48a

GIFT
MAIN SUPERVISOR



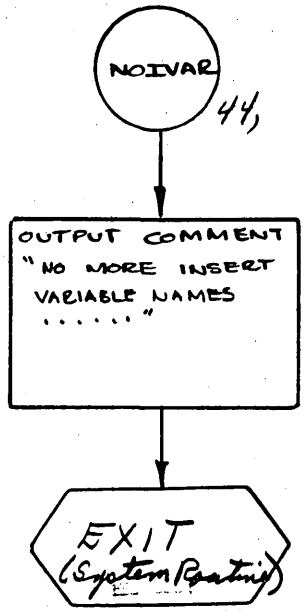
GIFT
SUBROUTINE MOUT
MOVE CHARACTERS TO OUTPUT BUFFER



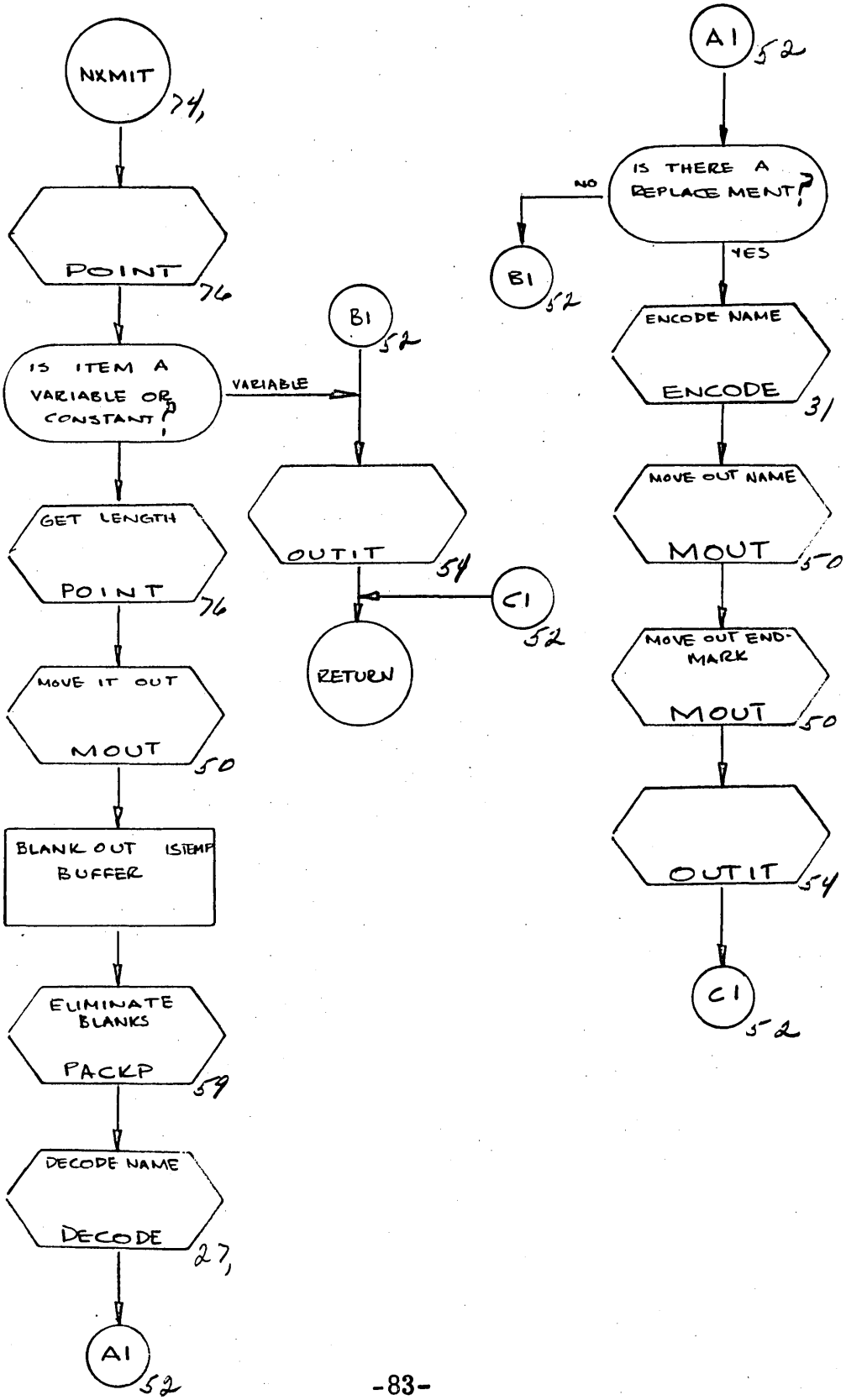
GIFT

SUBROUTINE NOIVAR

TERMINATE EXECUTION BECAUSE NO MORE INSERT VARIABLE NAMES ARE AVAILABLE



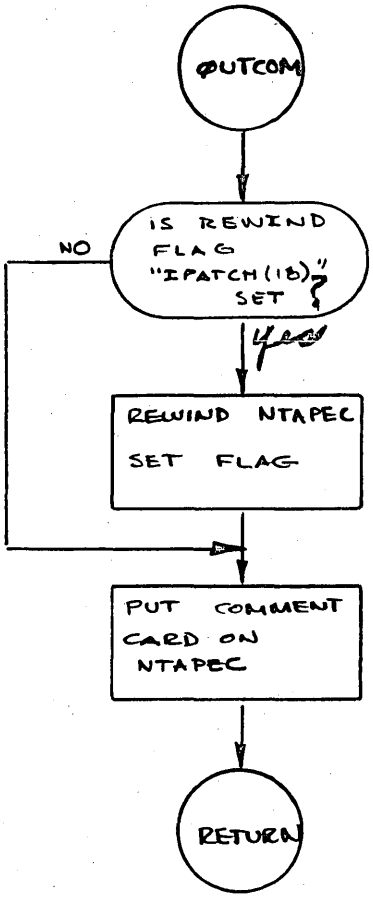
GIFT
SUBROUTINE NXMIT
PROCESS REWIND, BACKSPACE, AND ENDFILE STATEMENTS



GIFT

SUBROUTINE OUTCOM

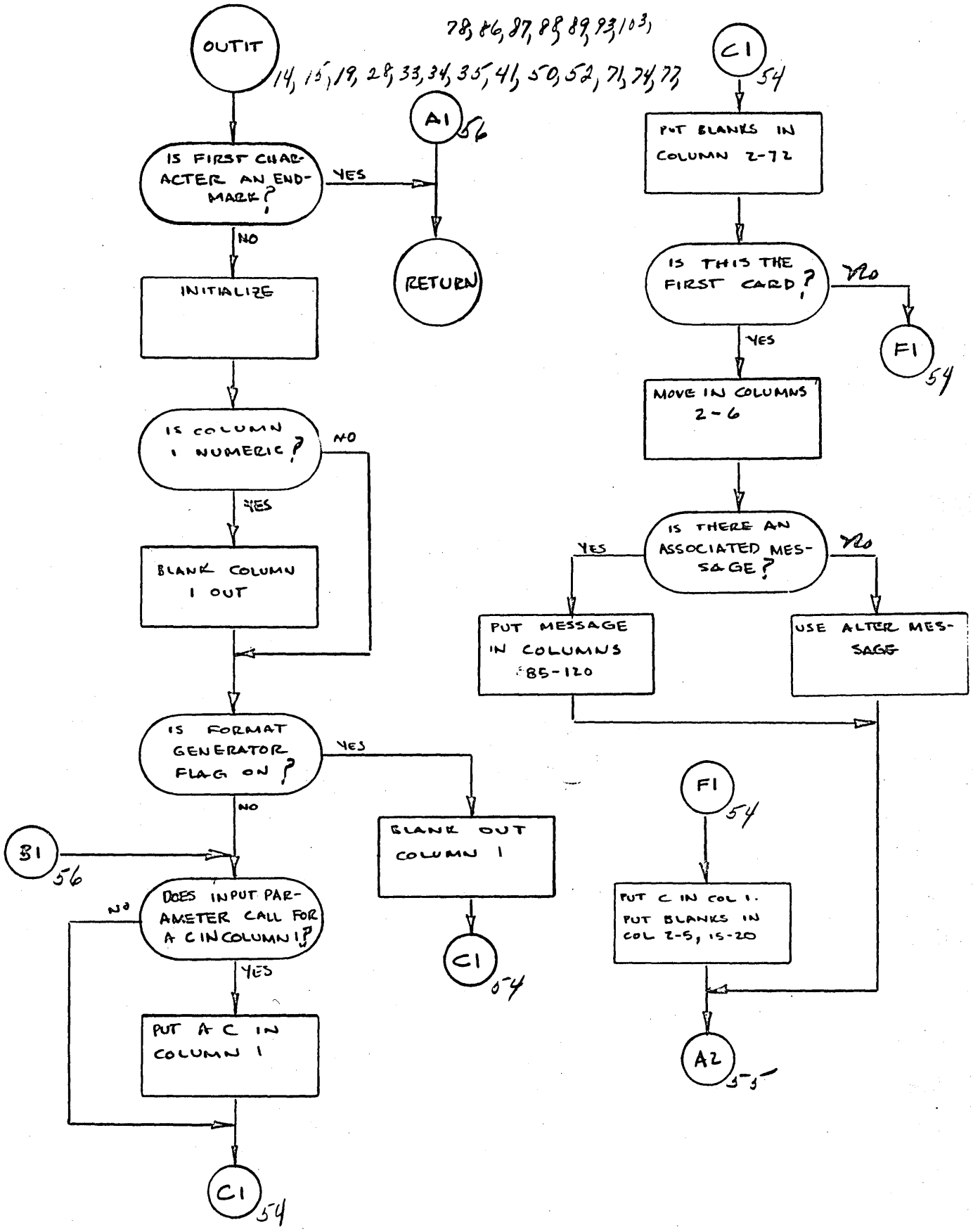
OUTPUTS COMMENT CARDS TO NTAPEC APPENDING SERIAL NUMBER



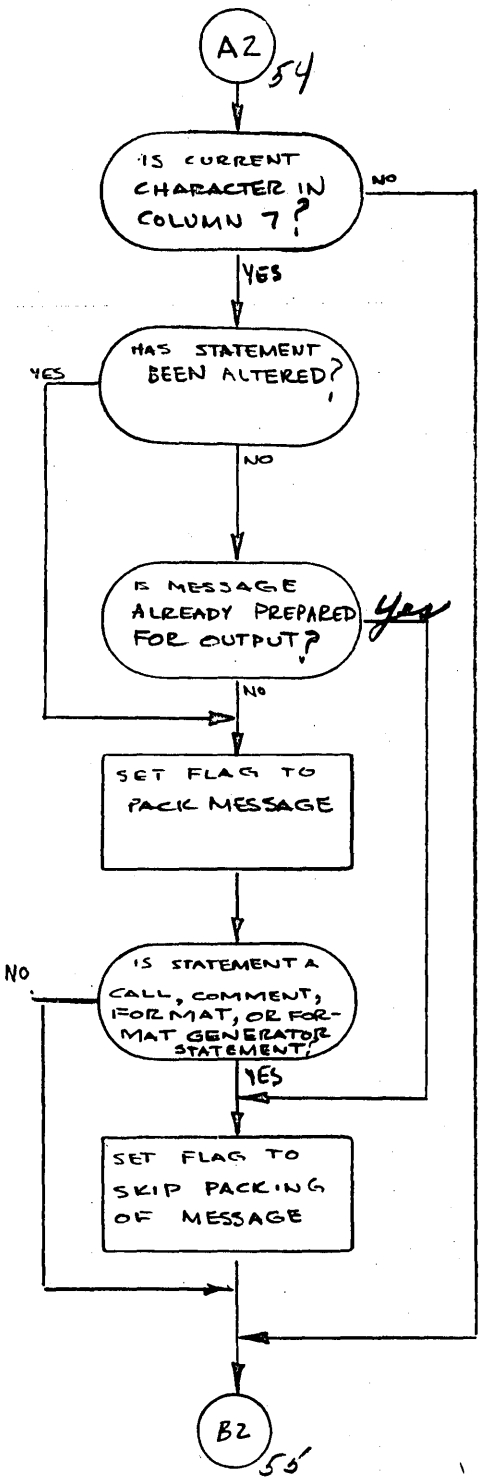
GIFT

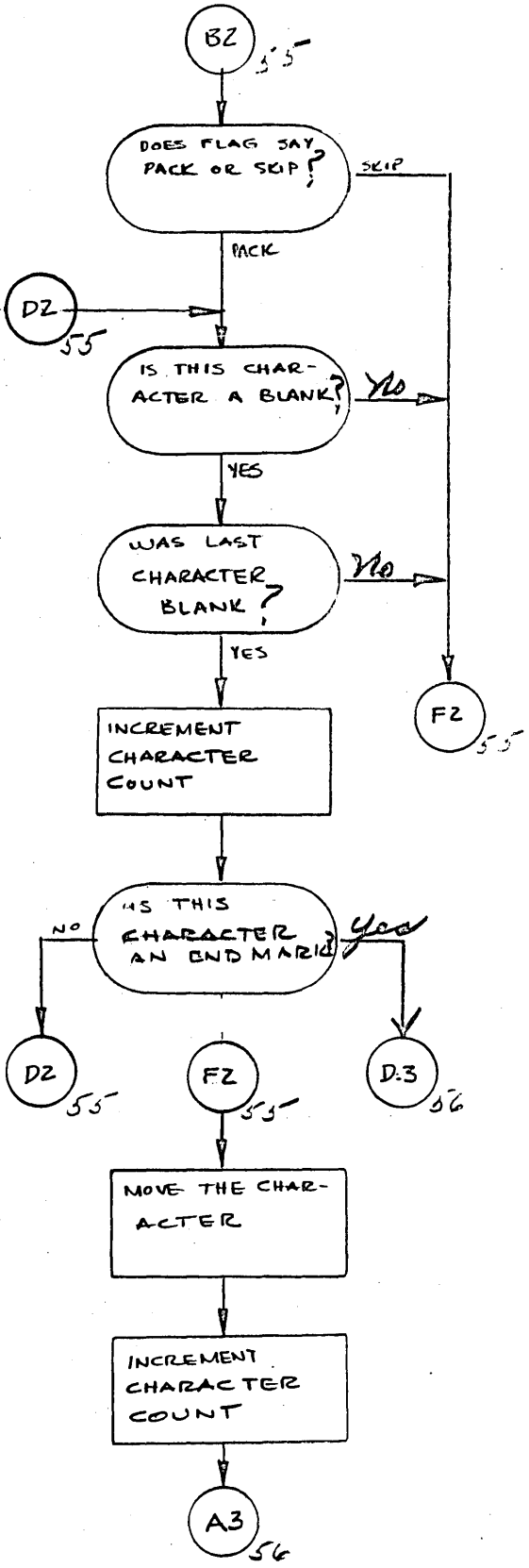
SUBROUTINE OUTIT

DECODE AND OUTPUT A STATEMENT

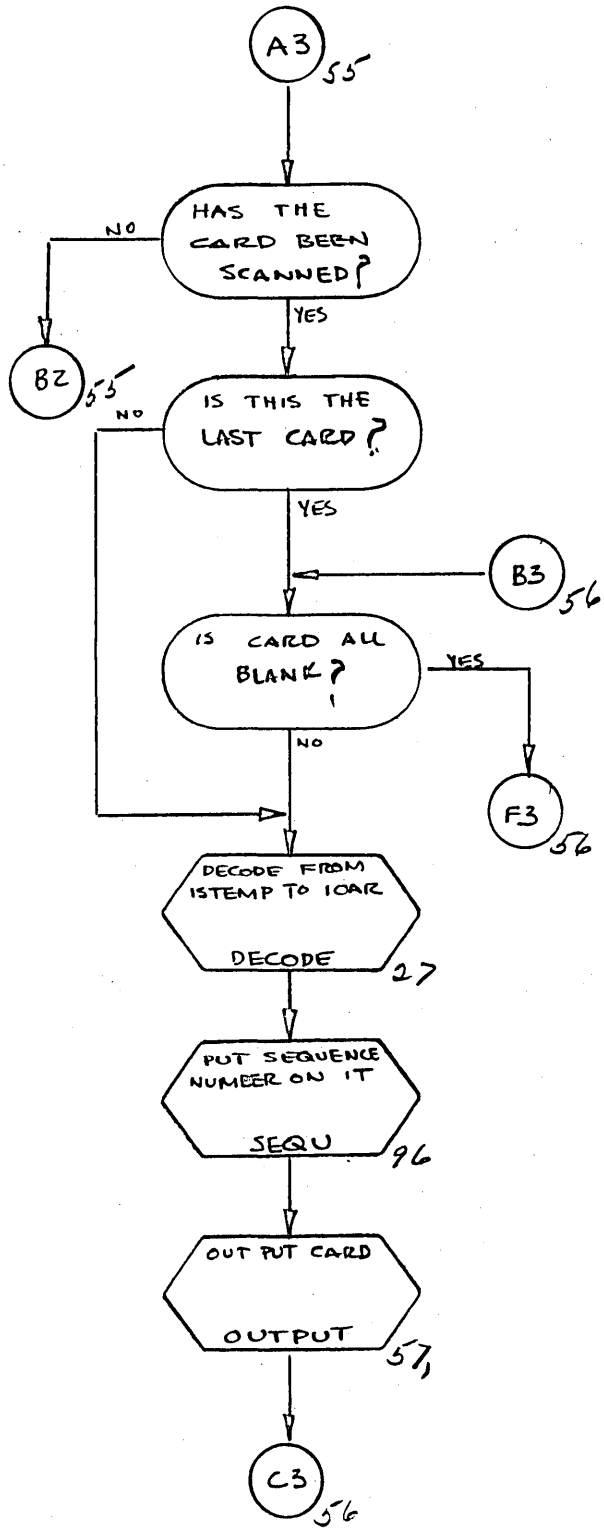


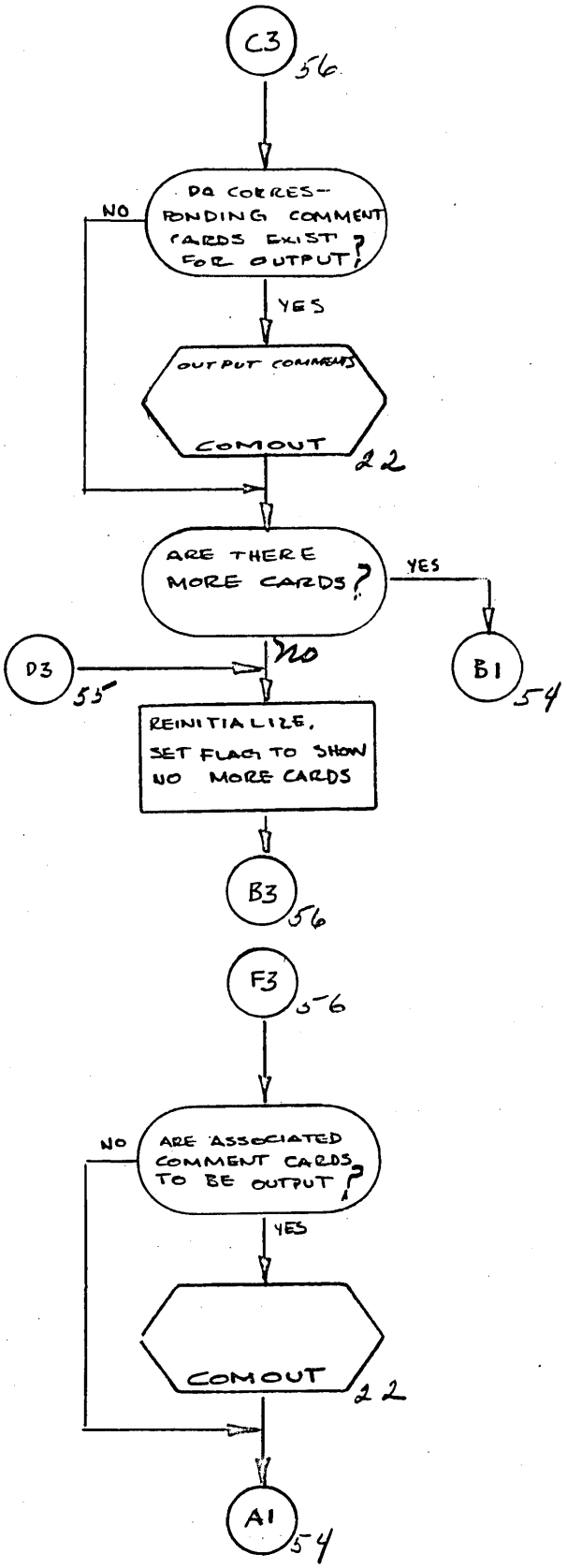
GIFT
SUBROUTINE OUTIT



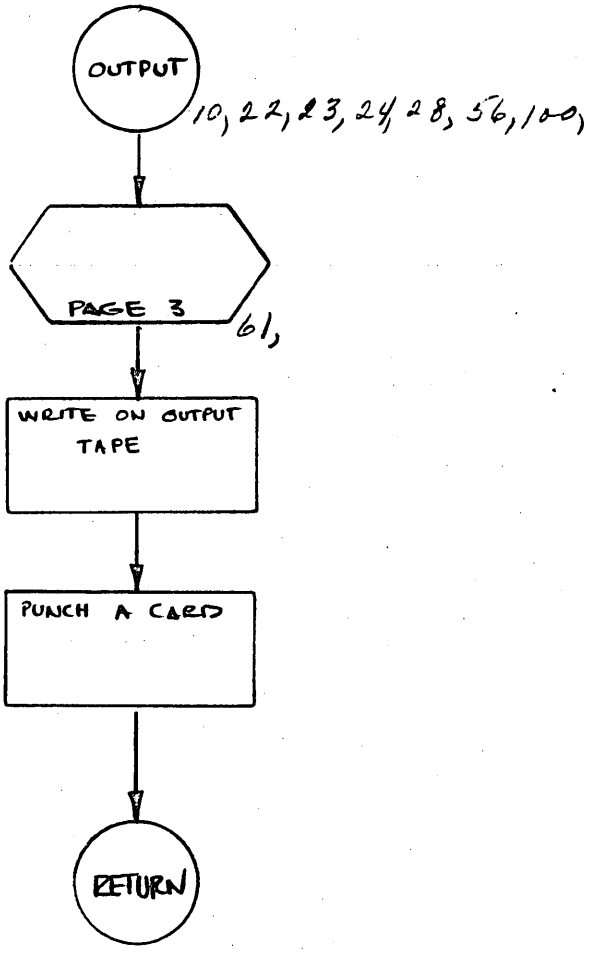


GIFT
SUBROUTINE OUTIT



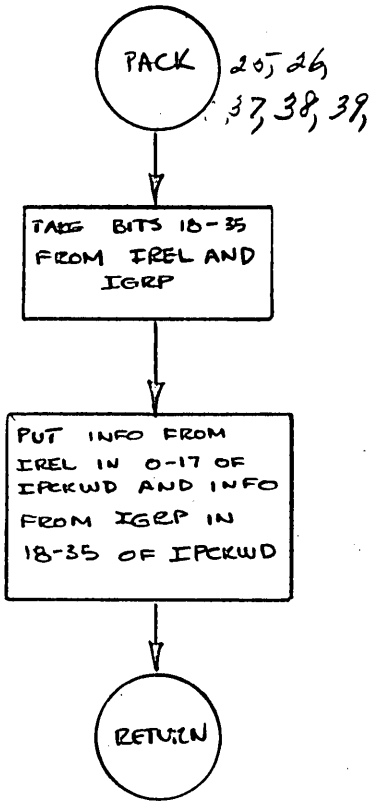


GIFT
SUBROUTINE OUTPUT
LIST AND PUNCH ONE CARD, CLEAR OUTPUT AREA

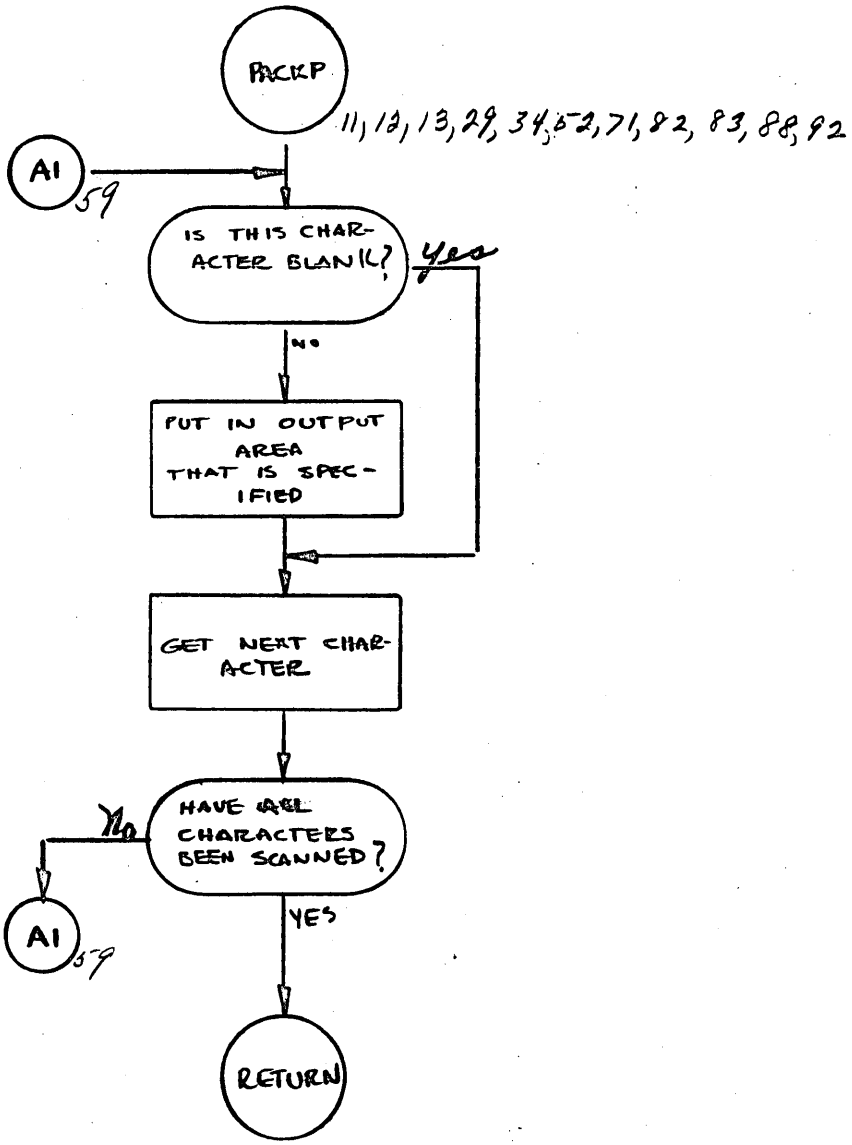


57

GIFT
SUBROUTINE PACK
PUT CONTENTS OF IGRP AND IREL IN IRCKWD



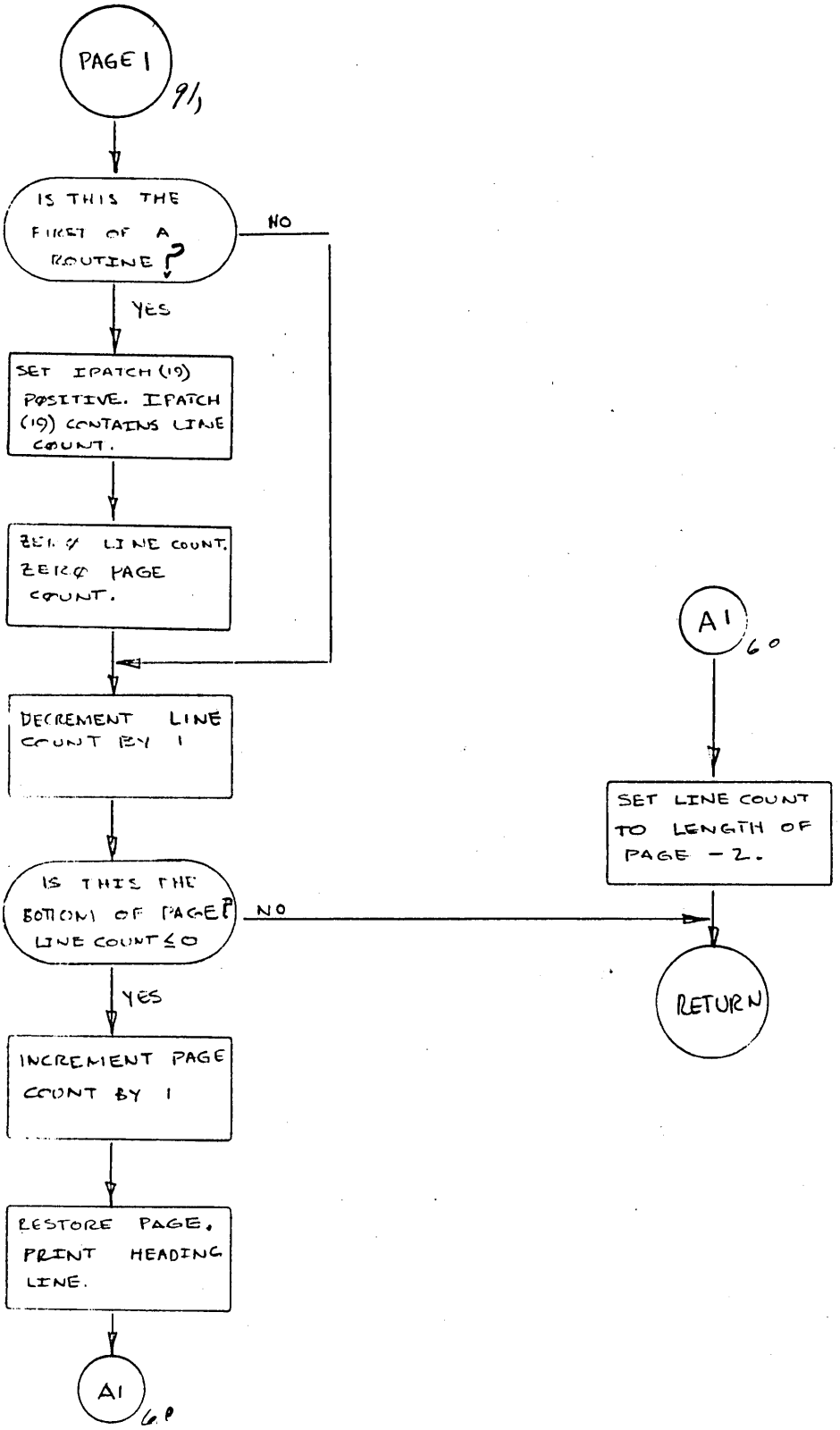
GIFT
SUBROUTINE PACKP
PACK AND OMIT BLANKS



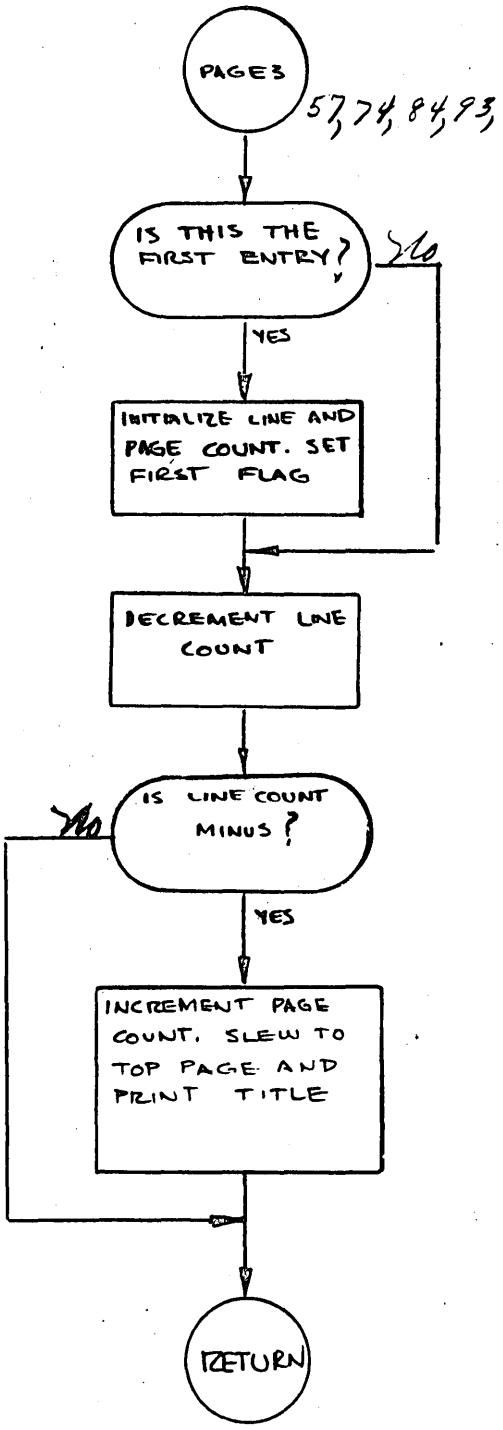
EIFT

SUBROUTINE PAGE 1

COUNT LINES AND RESTORE PAGE



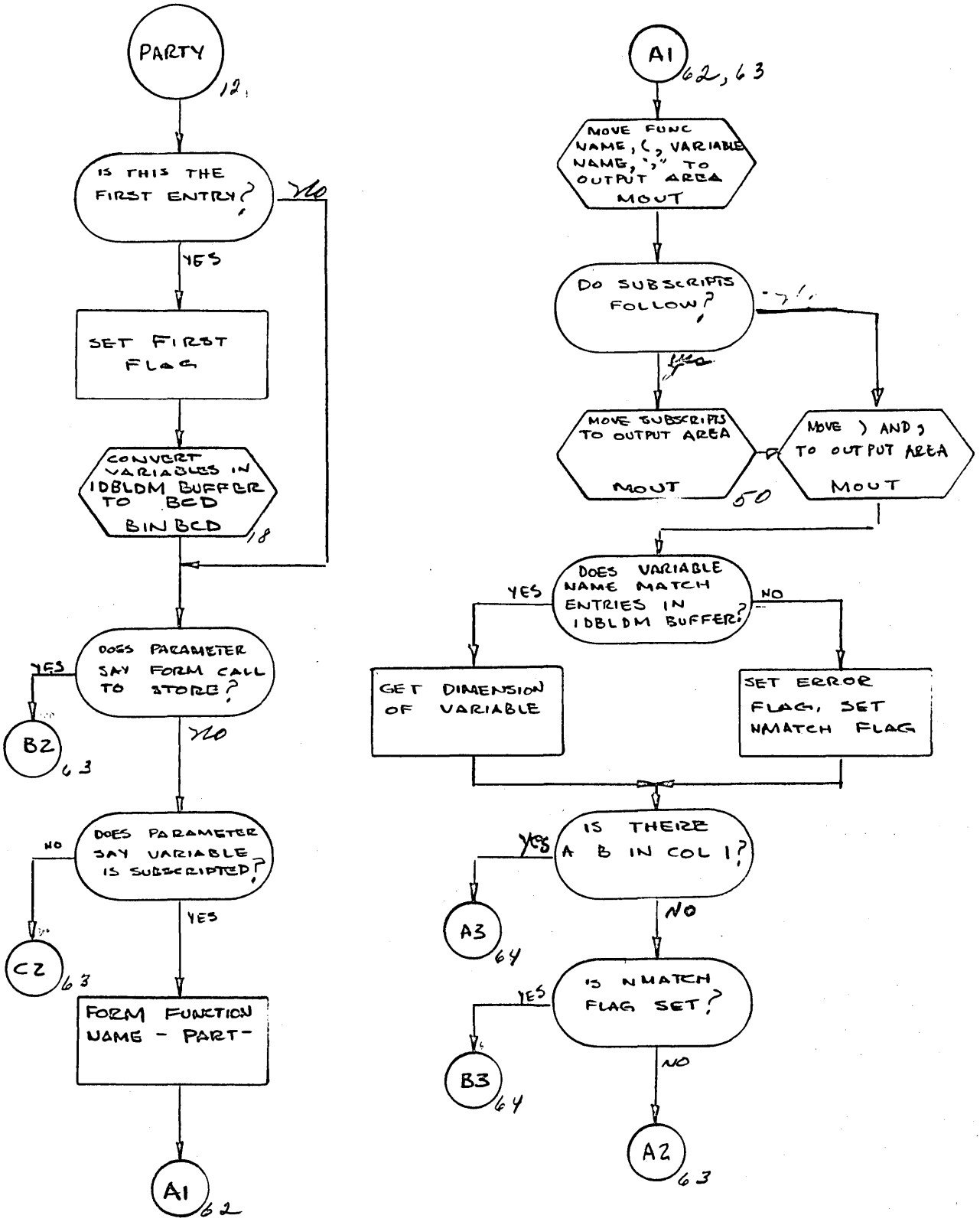
GIFT
SUBROUTINE PAGE3
COUNT LINES AND RESTORE PAGE



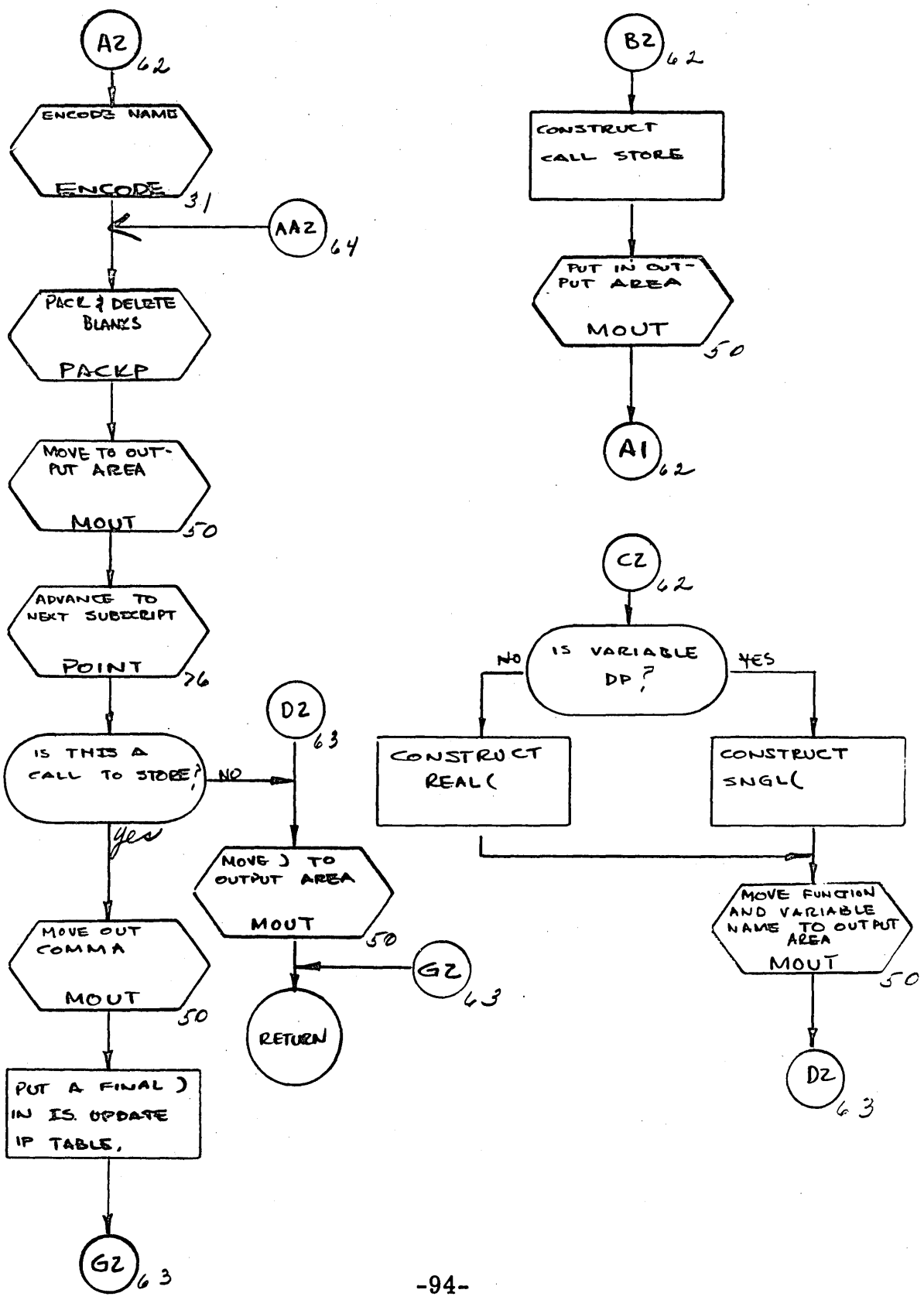
GIFT

SUBROUTINE PARTY

CONSTRUCT CALLS TO PART FUNCTION AND STORE SUBROUTINE

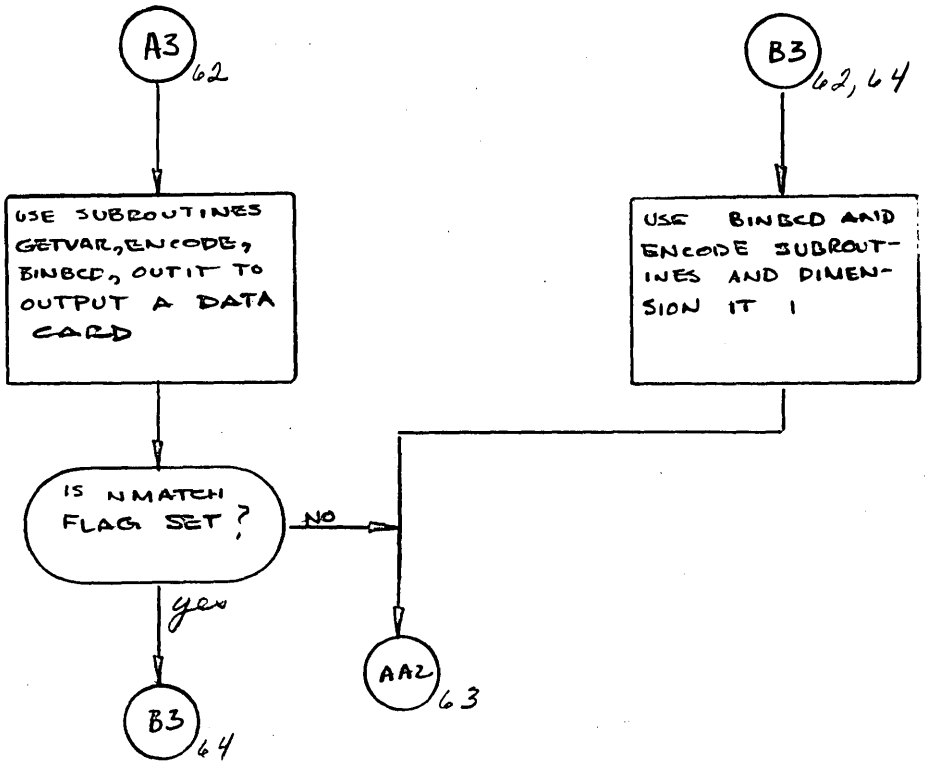


GIFT
SUBROUTINE PARTY



63

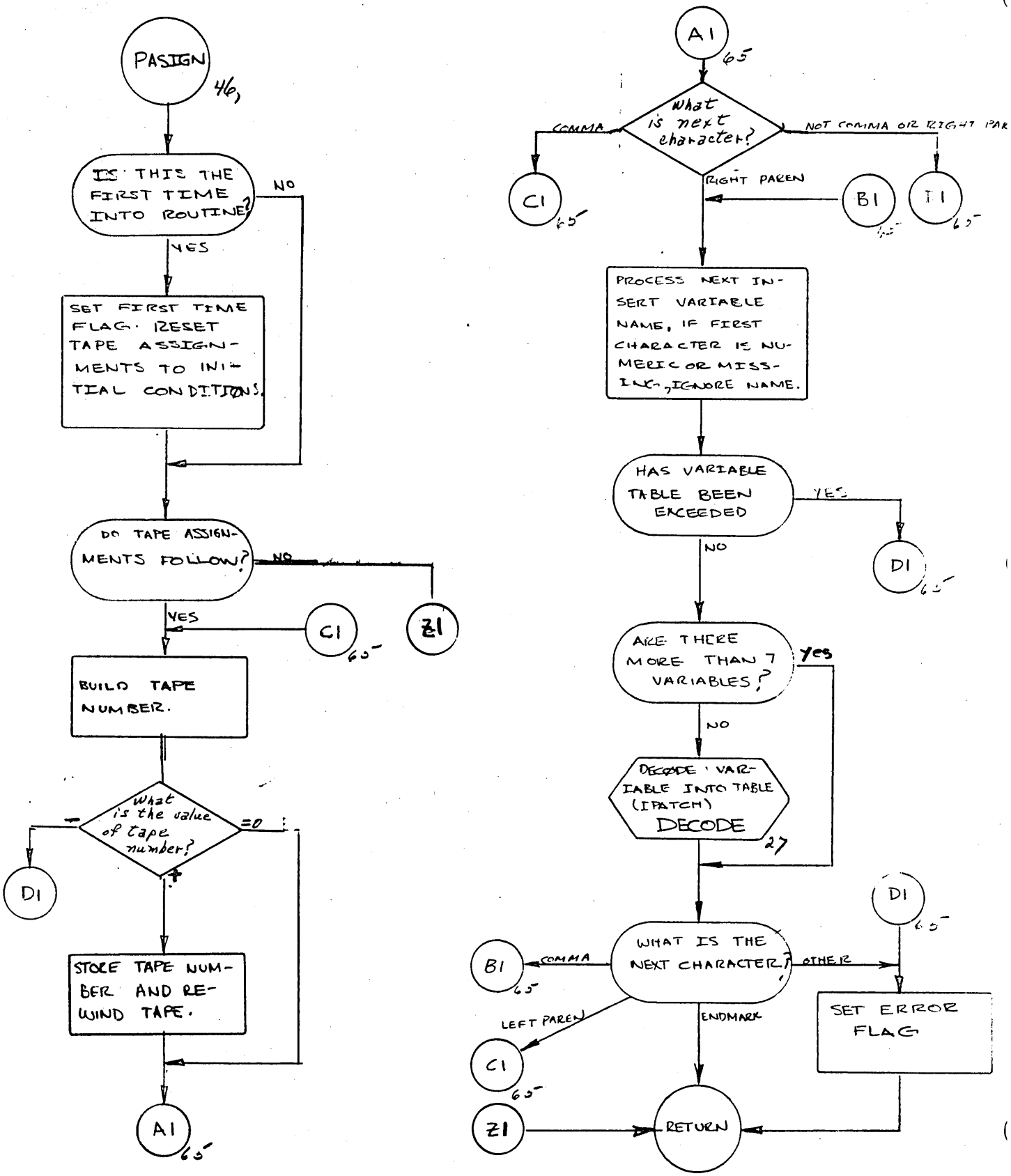
GIFT
SUBROUTINE PARTY



GLF 1

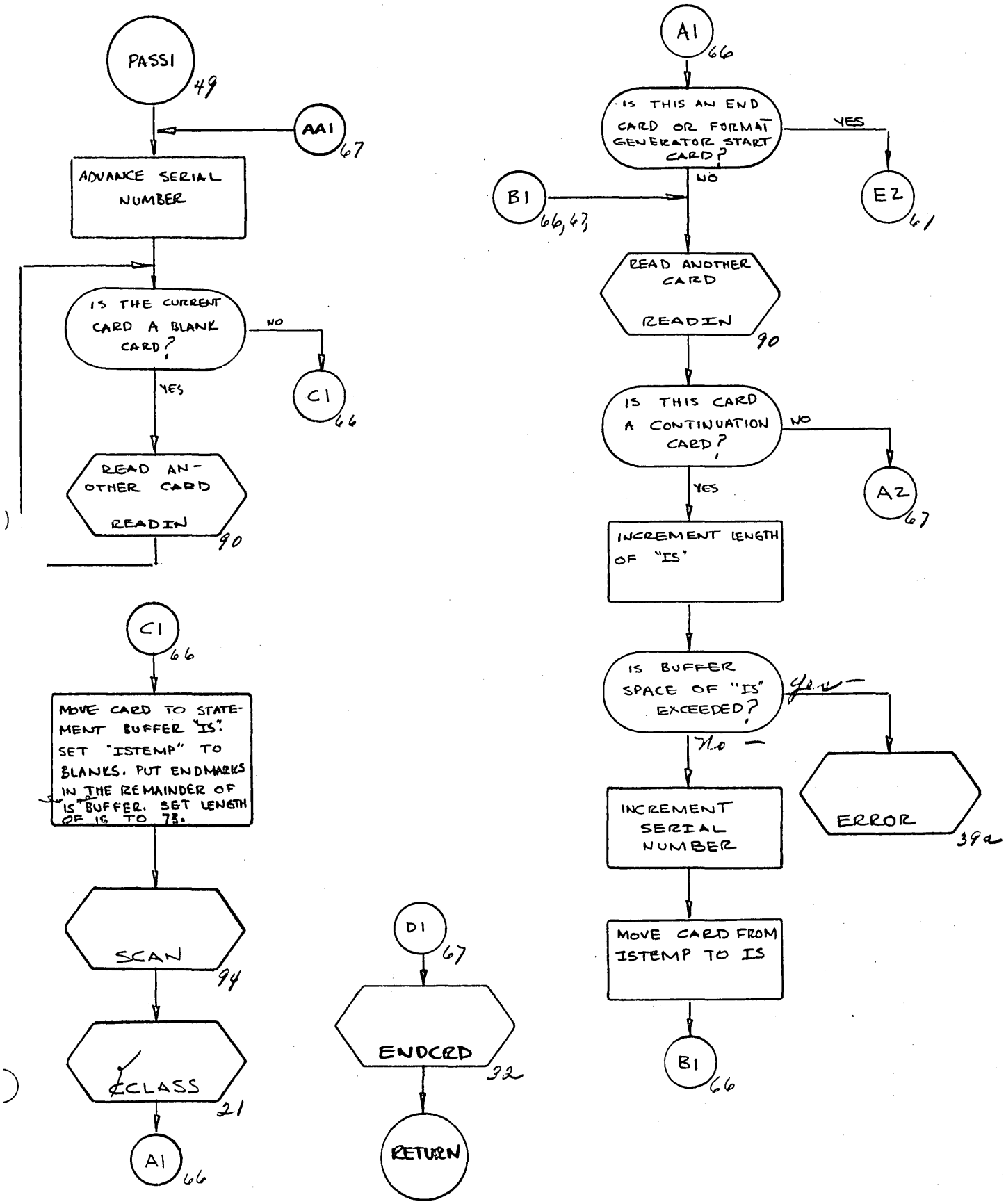
SUBROUTINE PASIGN

PROCESS *ASSIGN CARDS

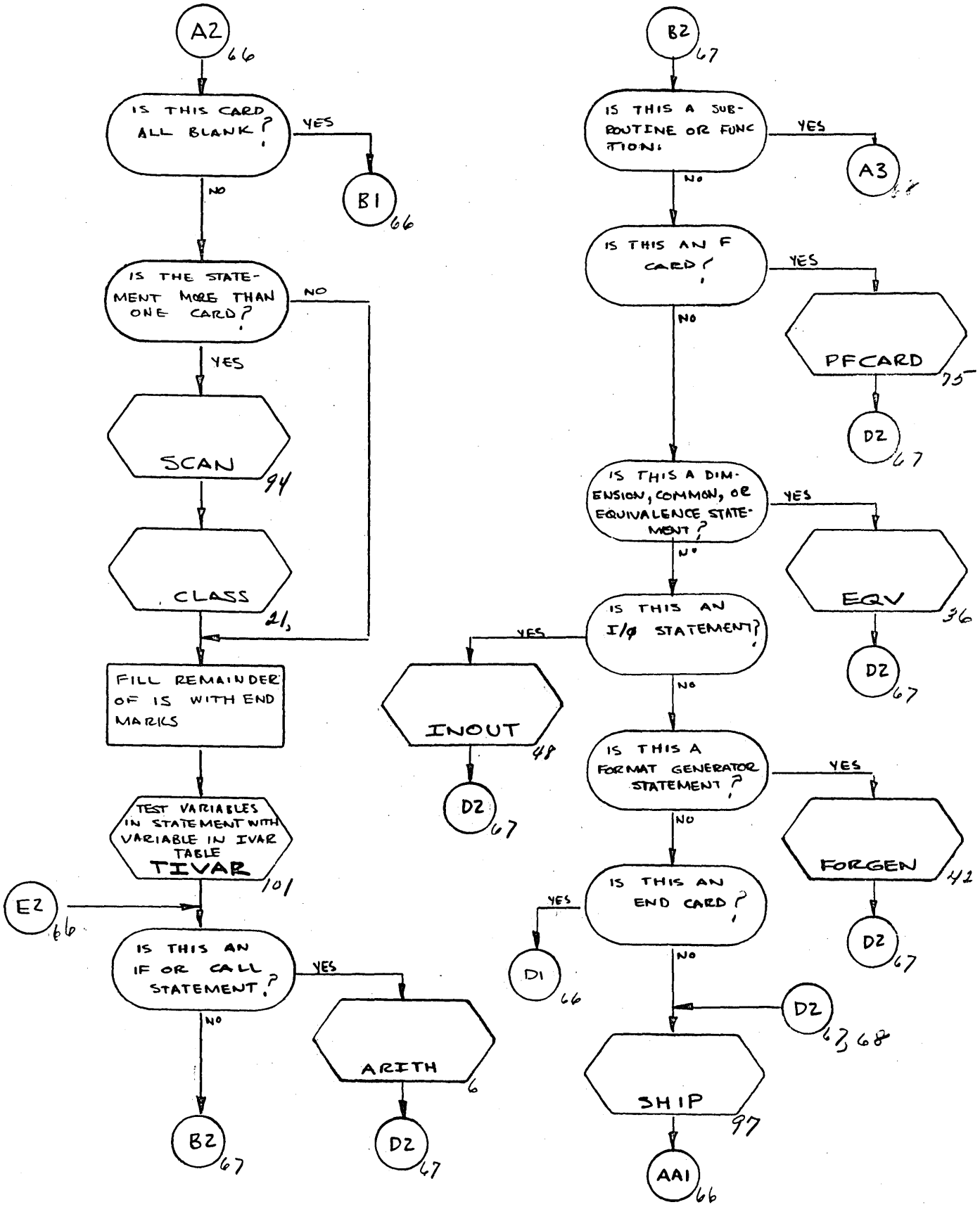


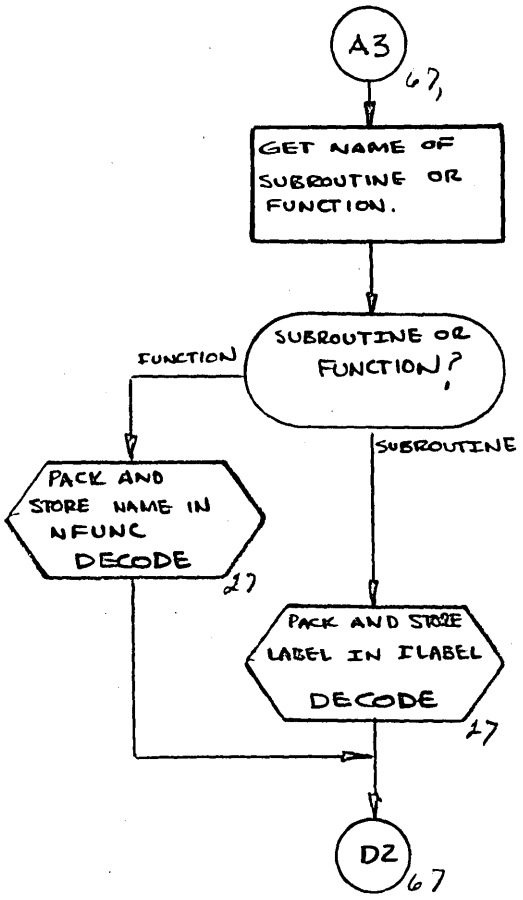
65

SIFT
SUBROUTINE PASS 1
SUPERVISE PROCESSING OF EACH INPUT STATEMENT

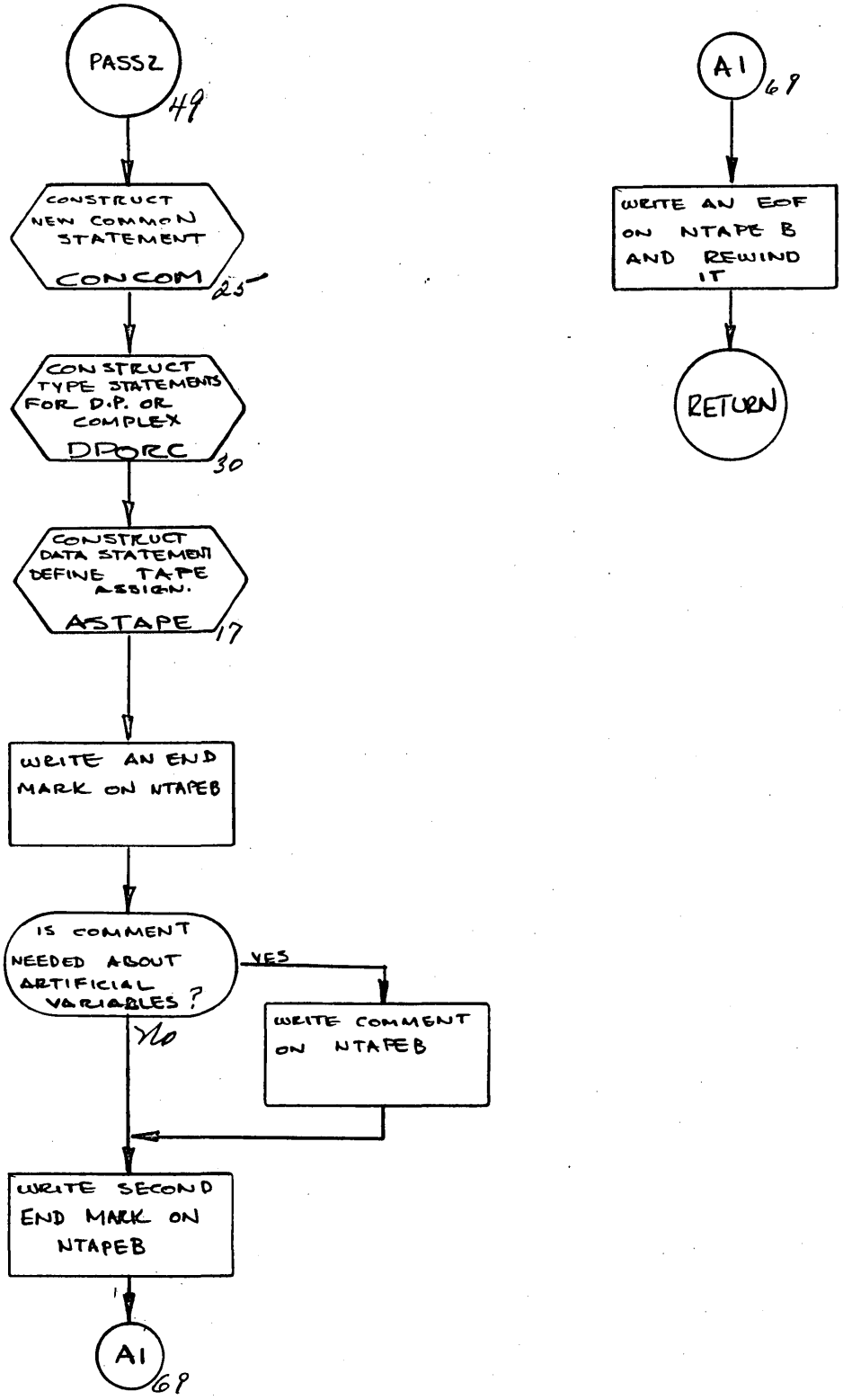


SUBROUTINE PASS1

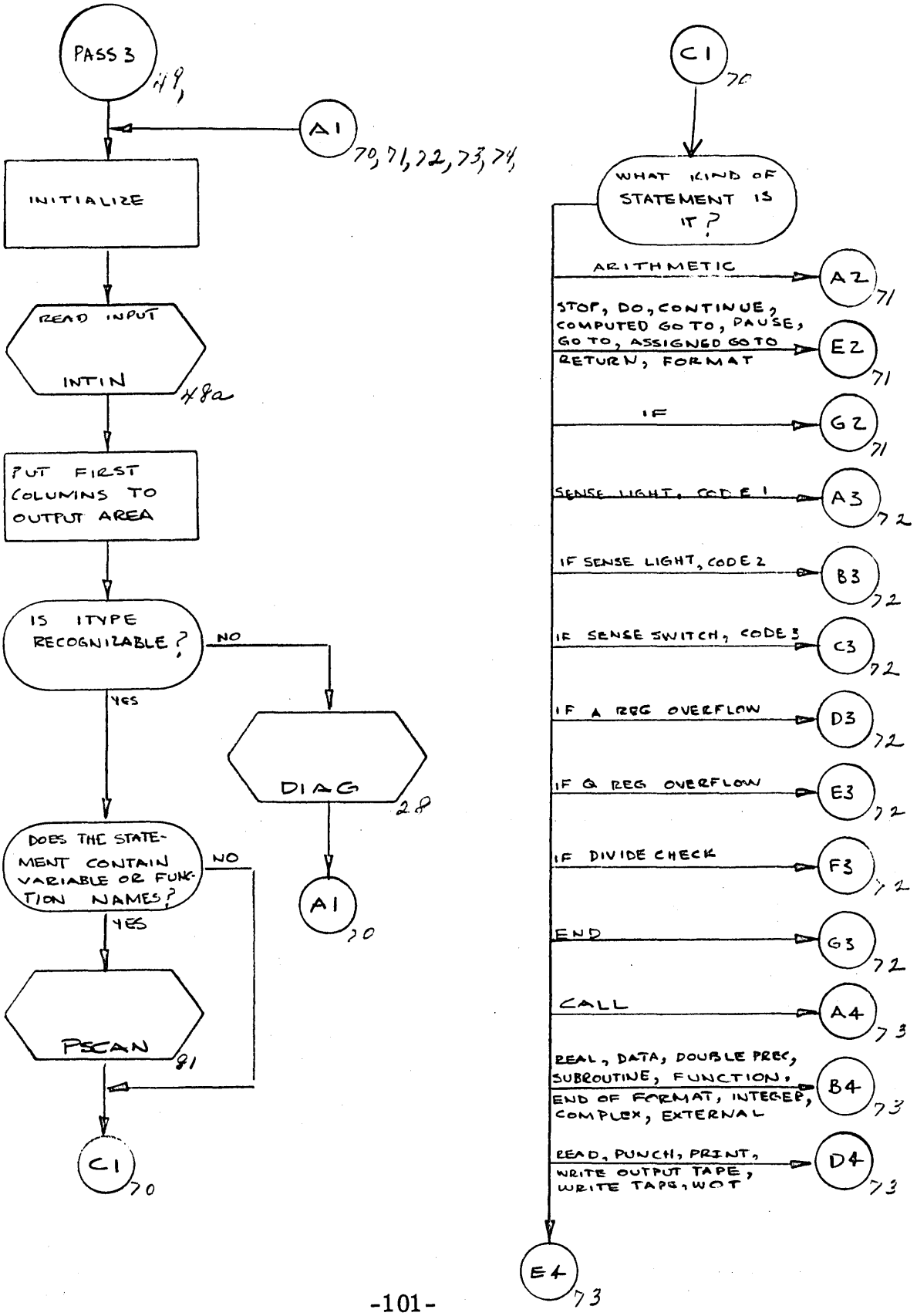



A3
67
GET NAME OF SUBROUTINE OR FUNCTION.
SUBROUTINE OR FUNCTION?
FUNCTION
SUBROUTINE
PACK AND STORE NAME IN NFUNC
DECODE
27
PACK AND STORE LABEL IN ILABEL
DECODE
27
D2
67

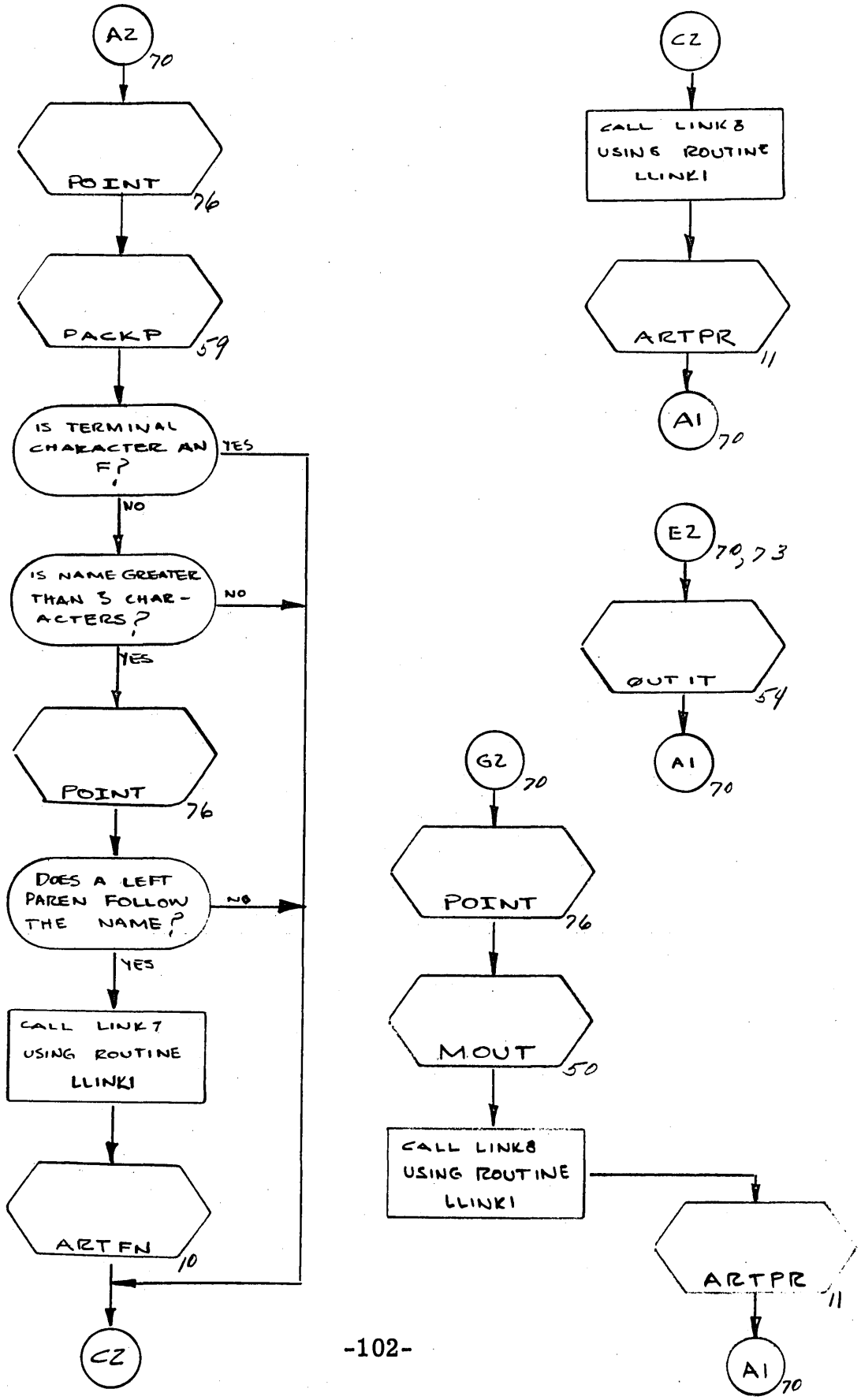
GIFT
SUBROUTINE PASS2
SUPERVISE PROCESSING DURING PASS 2.



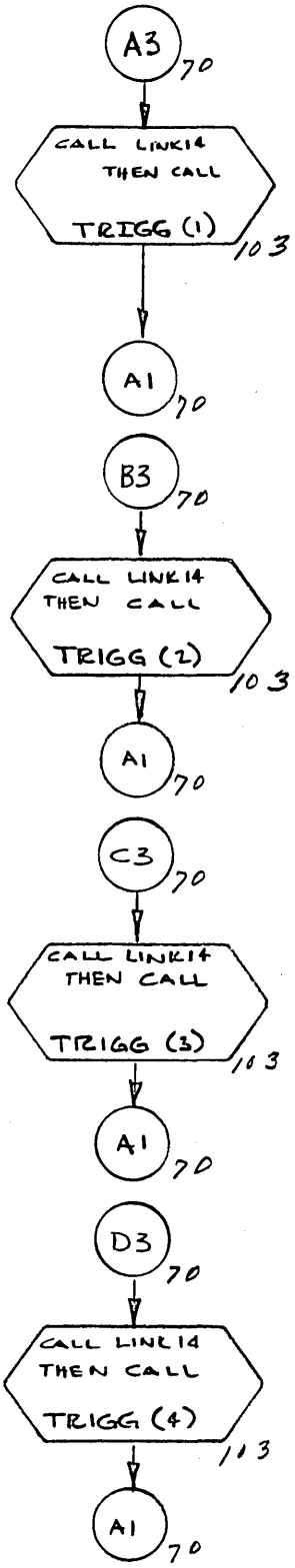
GIFT
SUBROUTINE PASS3
SUPERVISE PROCESSING OF EACH STATEMENT ON

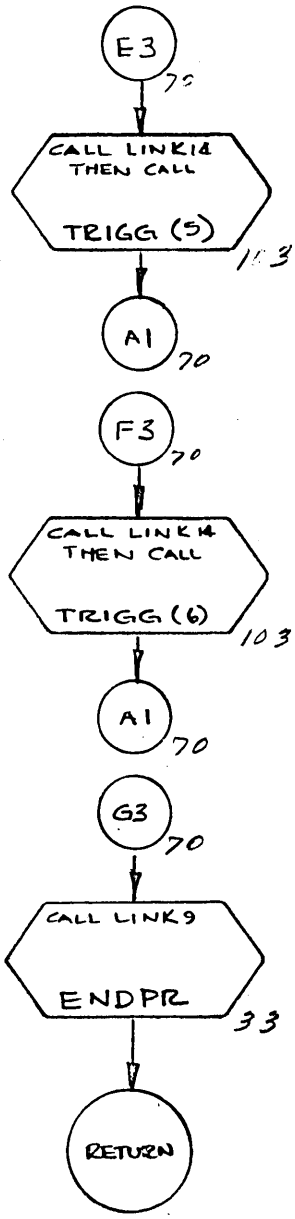


GIFT
SUBROUTINE PASS 3

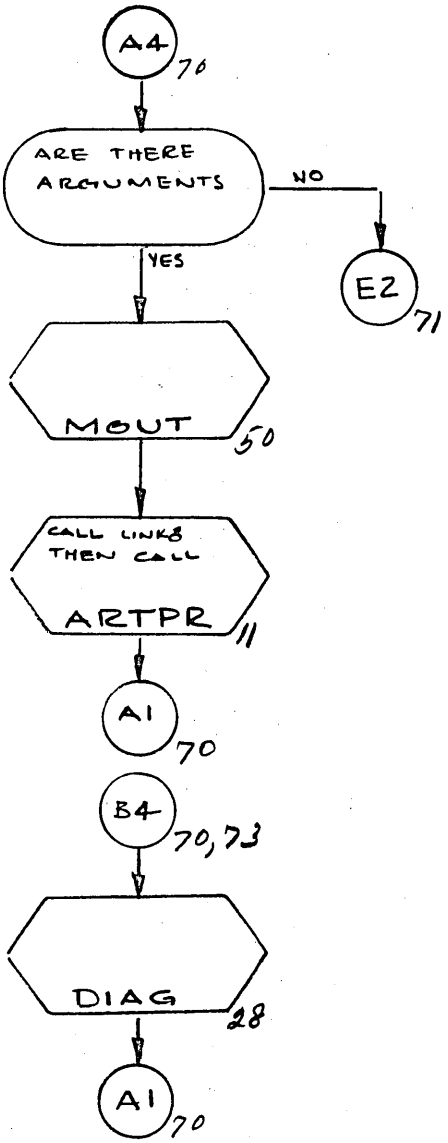


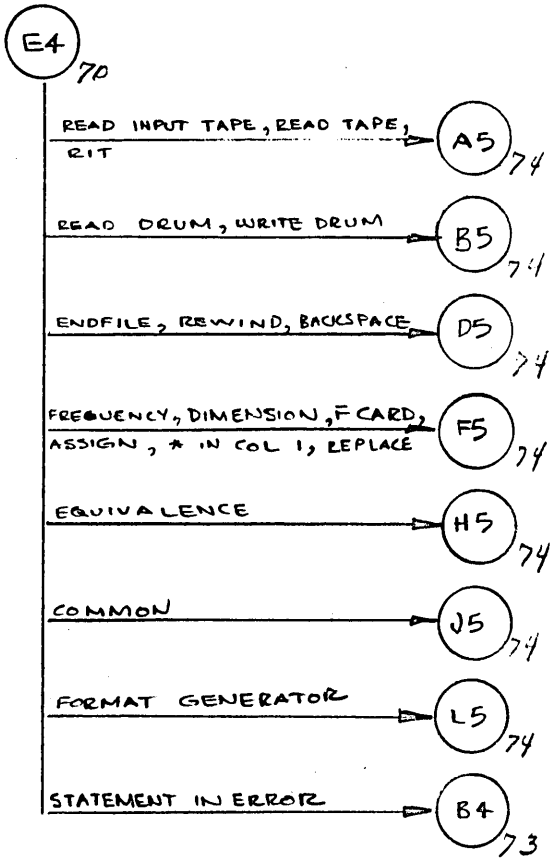
GIFT
SUBROUTINE PASS3

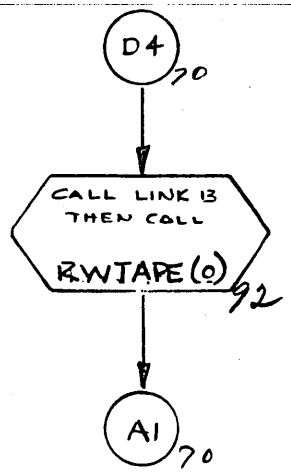




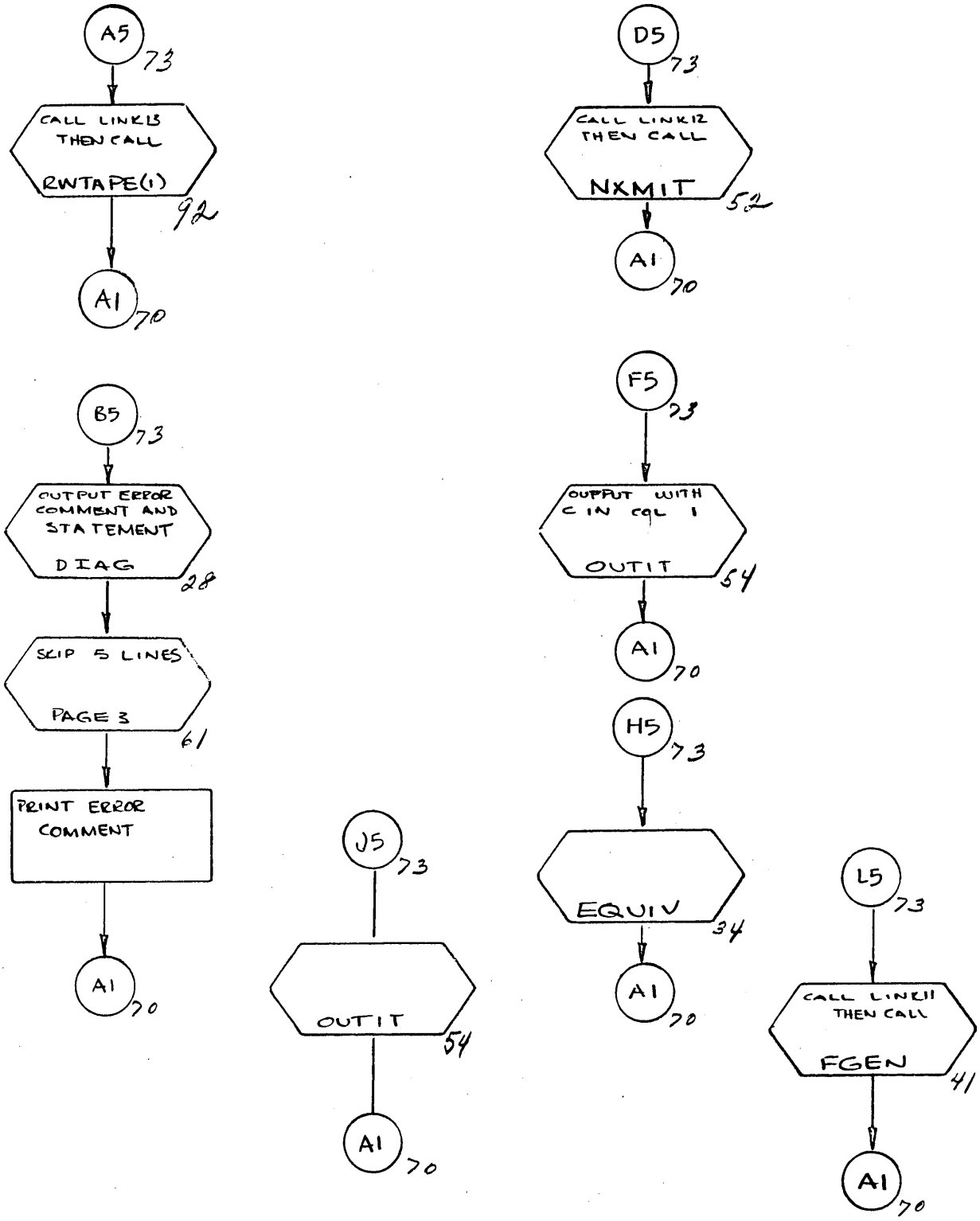
GIFT
SUBROUTINE PASS3





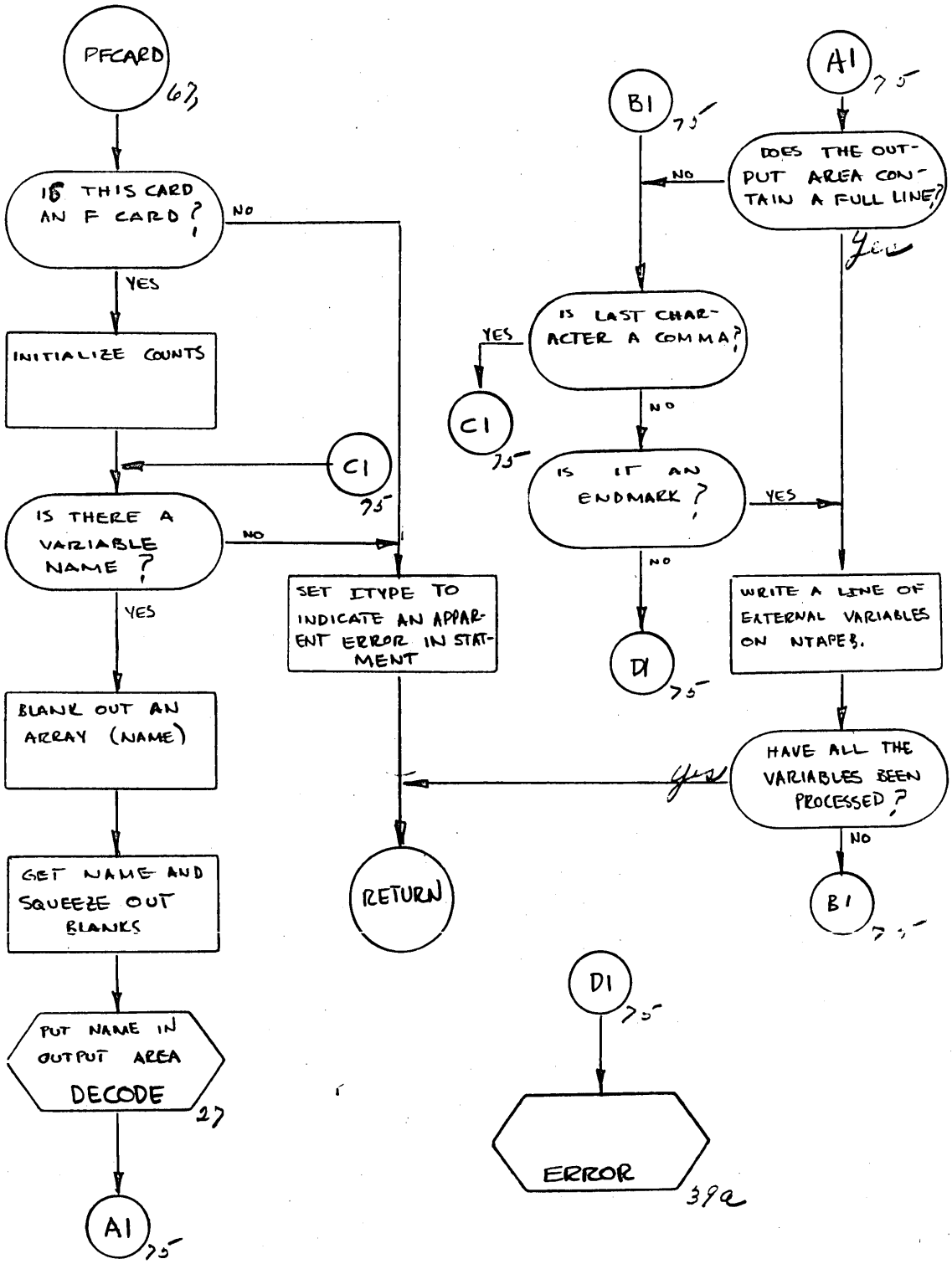


GIFT
SUBROUTINE PASS3

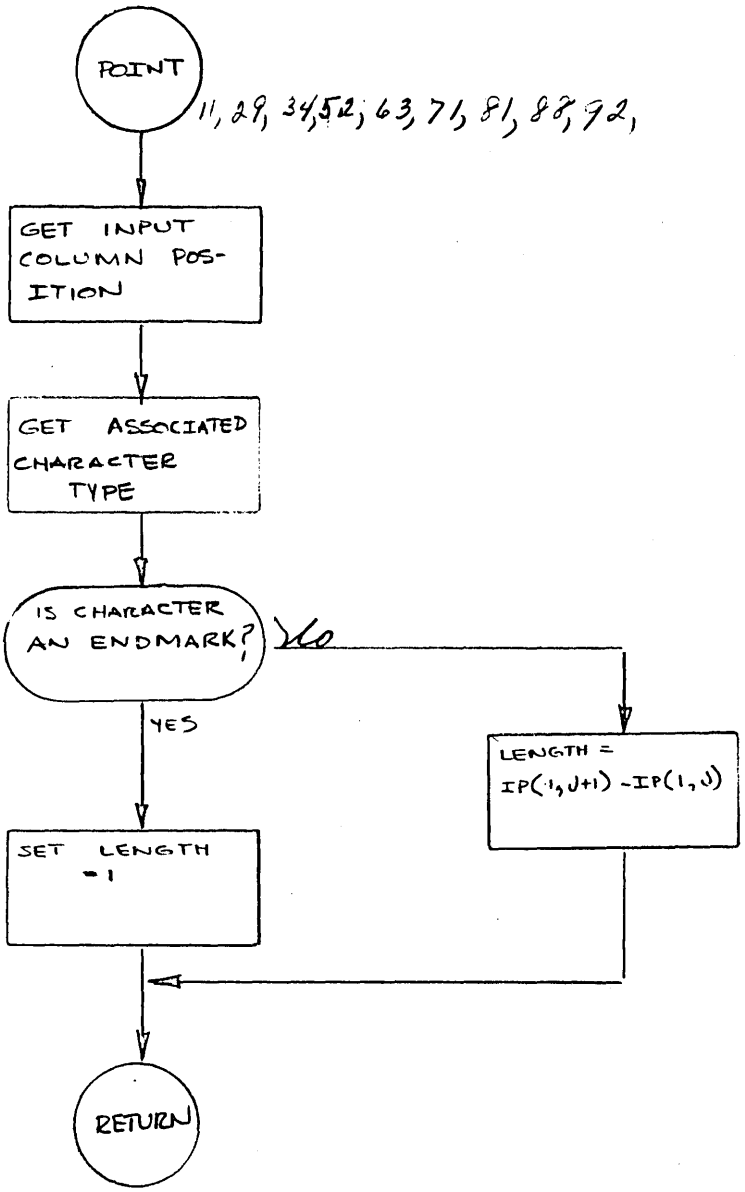


74

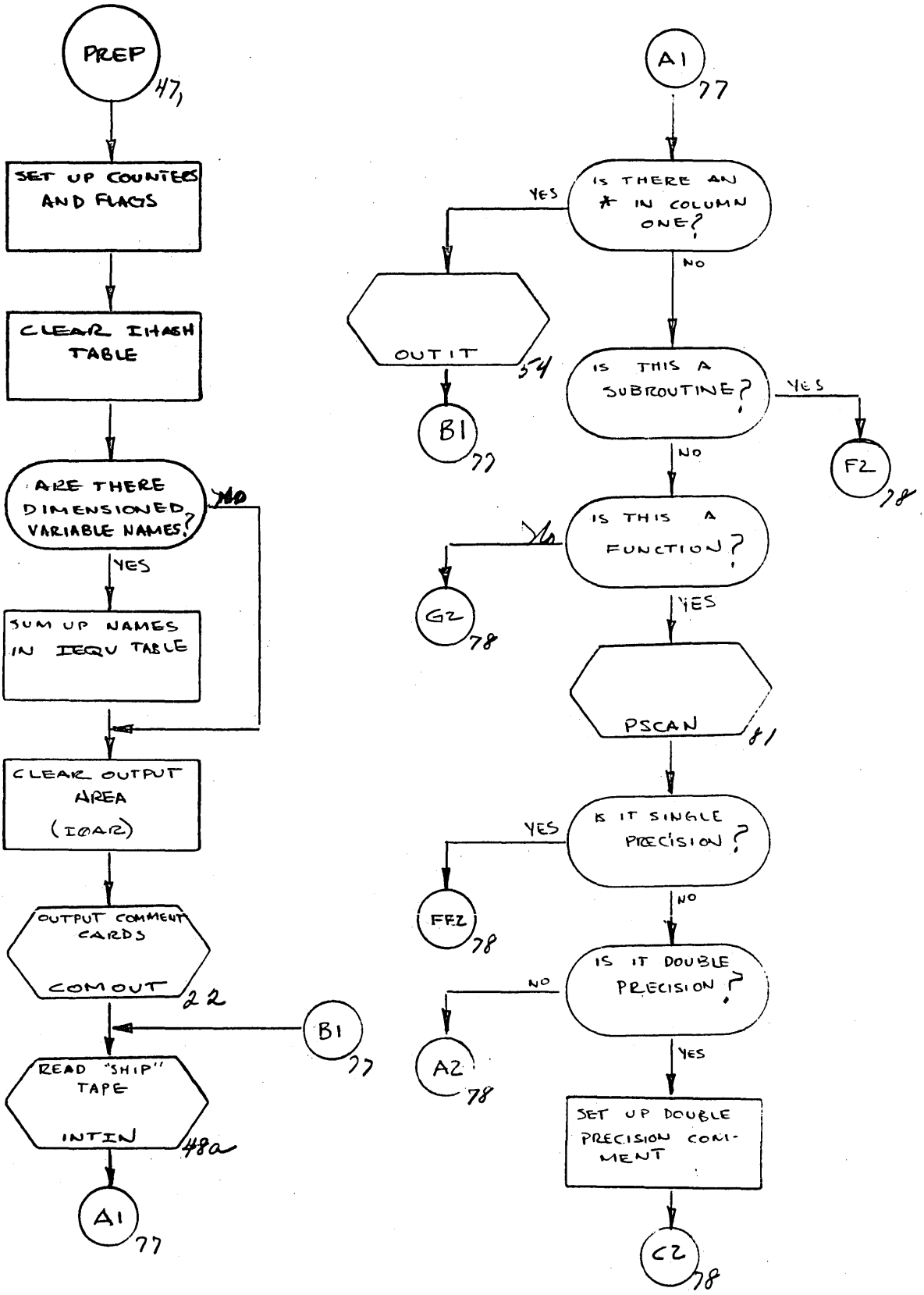
GIFT
SUBROUTINE PFCARD
PROCESS F CARDS



GIFT
SUBROUTINE POINT
IDENTIFY A FIELD WITH AN INDEX IN IP
INPUT = COLUMN NO.. OUTPUT = COL. NO., LENGTH OF PARAMETER, TYPE CODE.

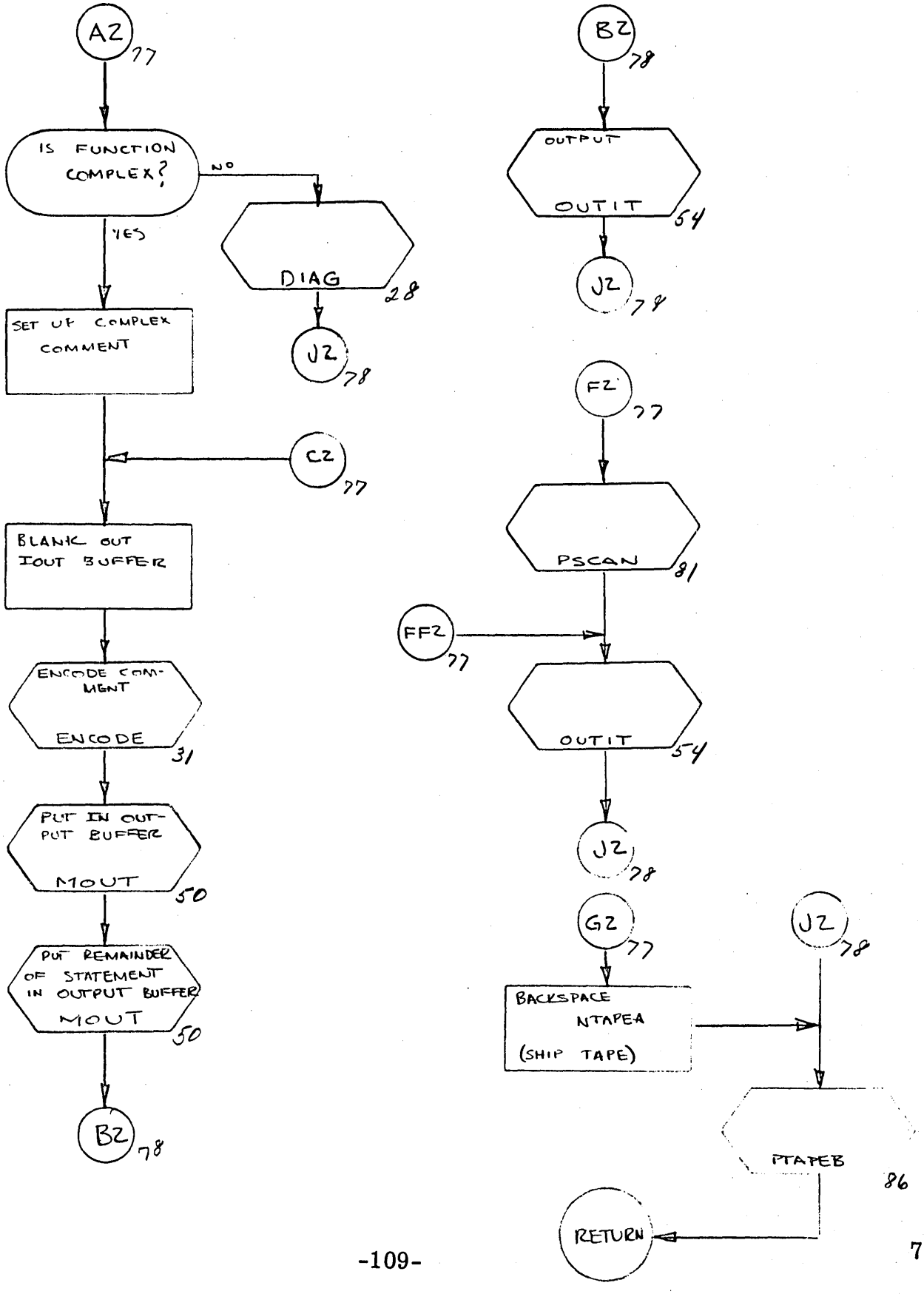


GIFT
SUBROUTINE PREP
PREPROCESS TABLES FOR THIRD PASS



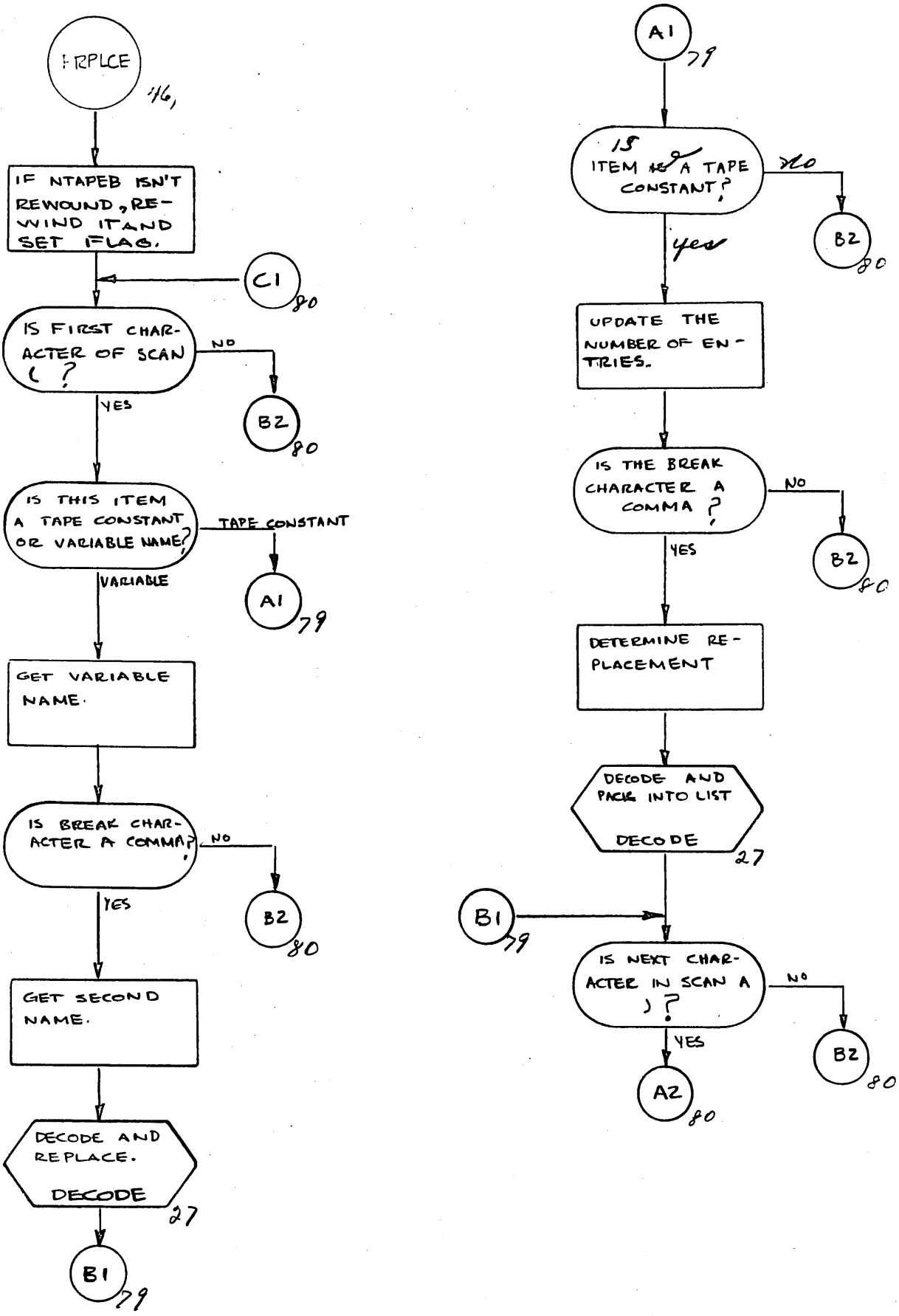
77

GIFT
SUBROUTINE PREP



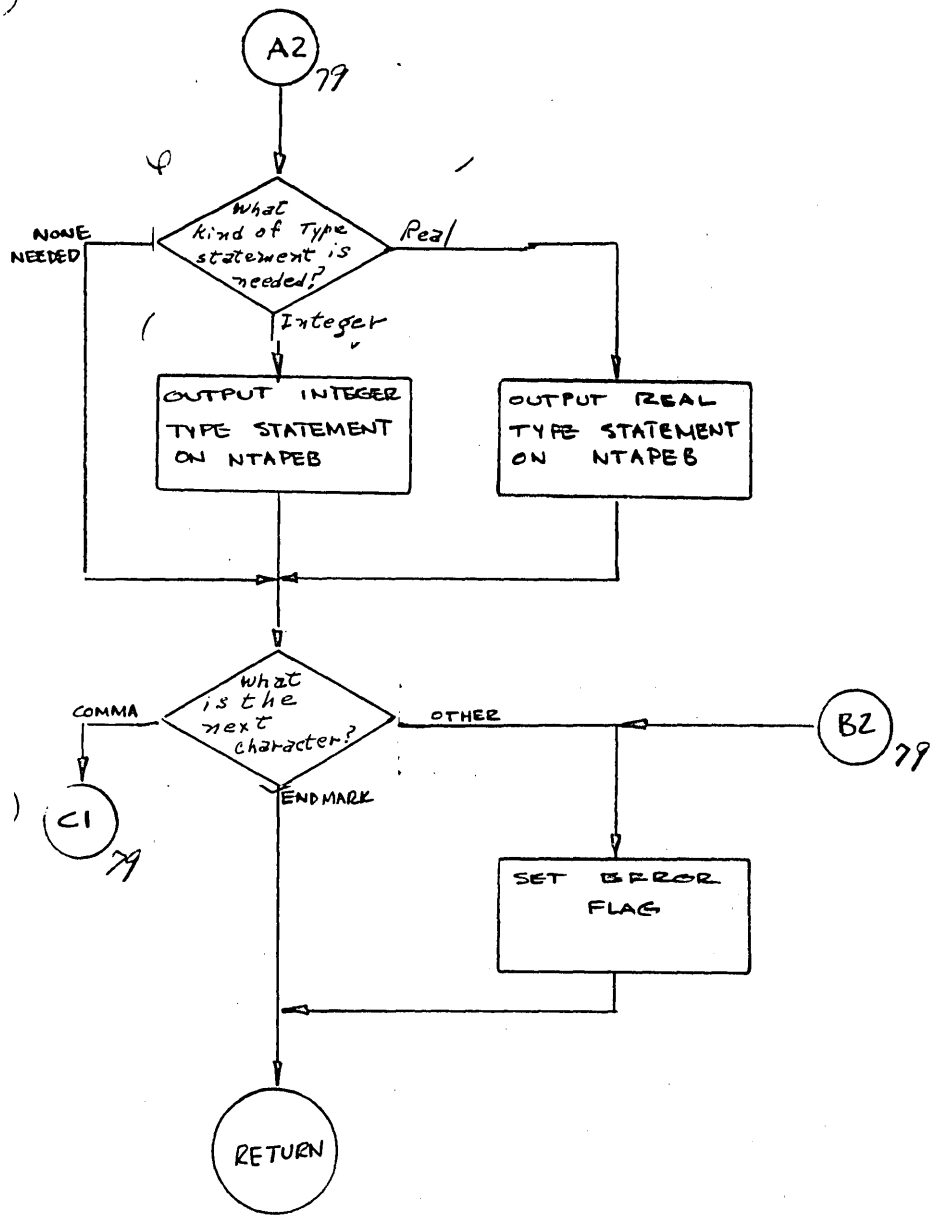
SUBROUTINE PRPLCE

PROCESS * REPLACE CARDS

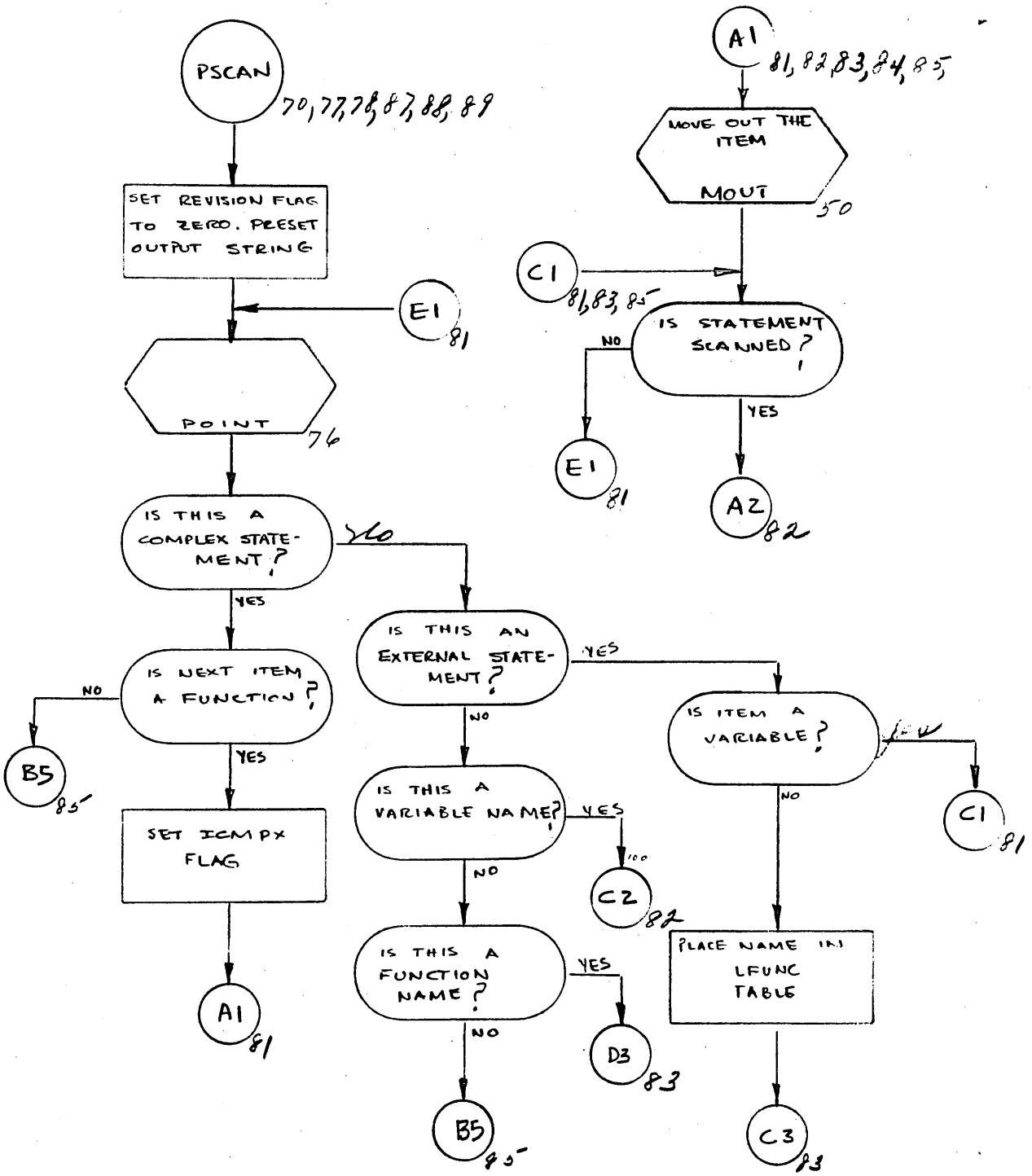


79

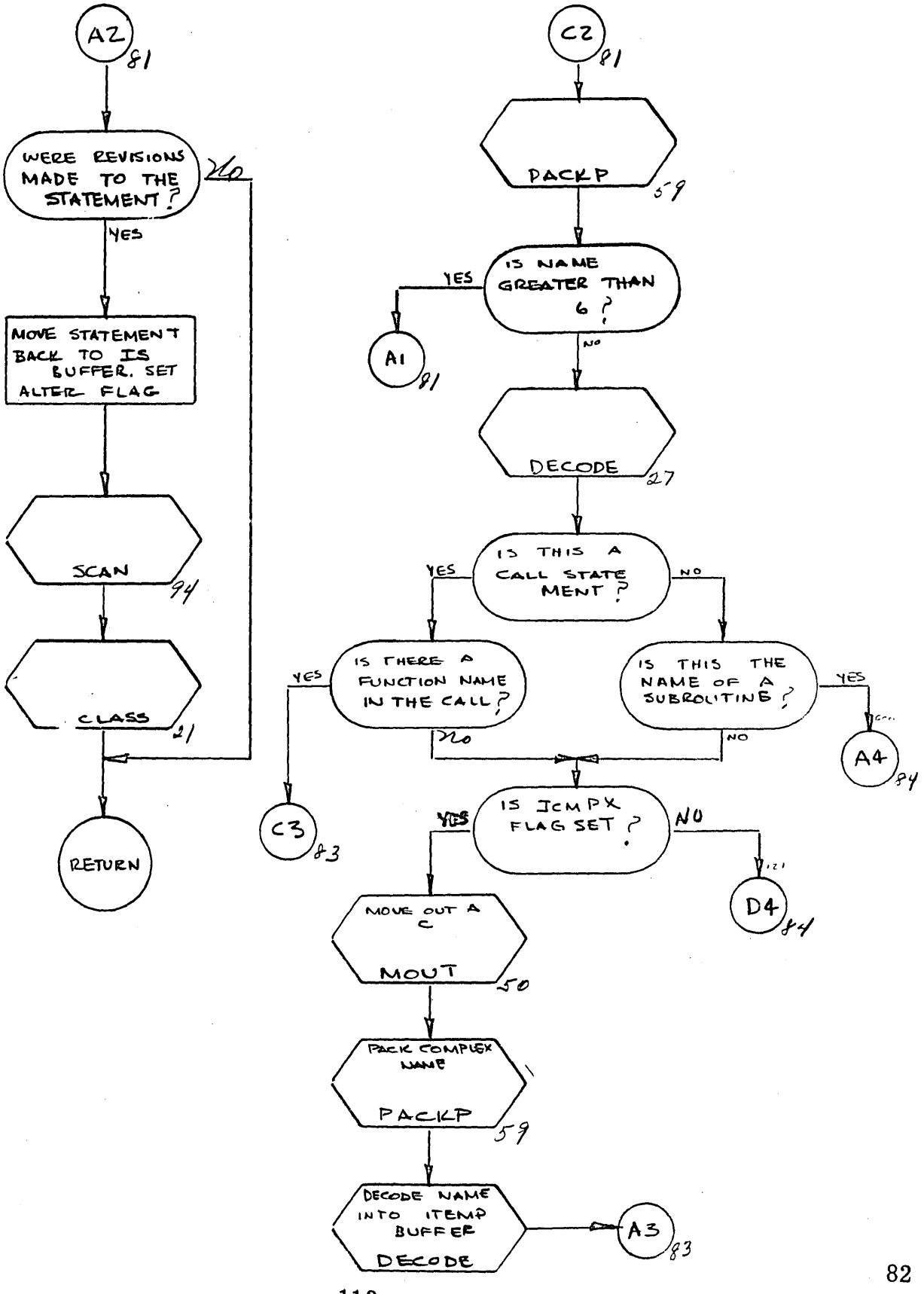
SUBROUTINE PRPLCE



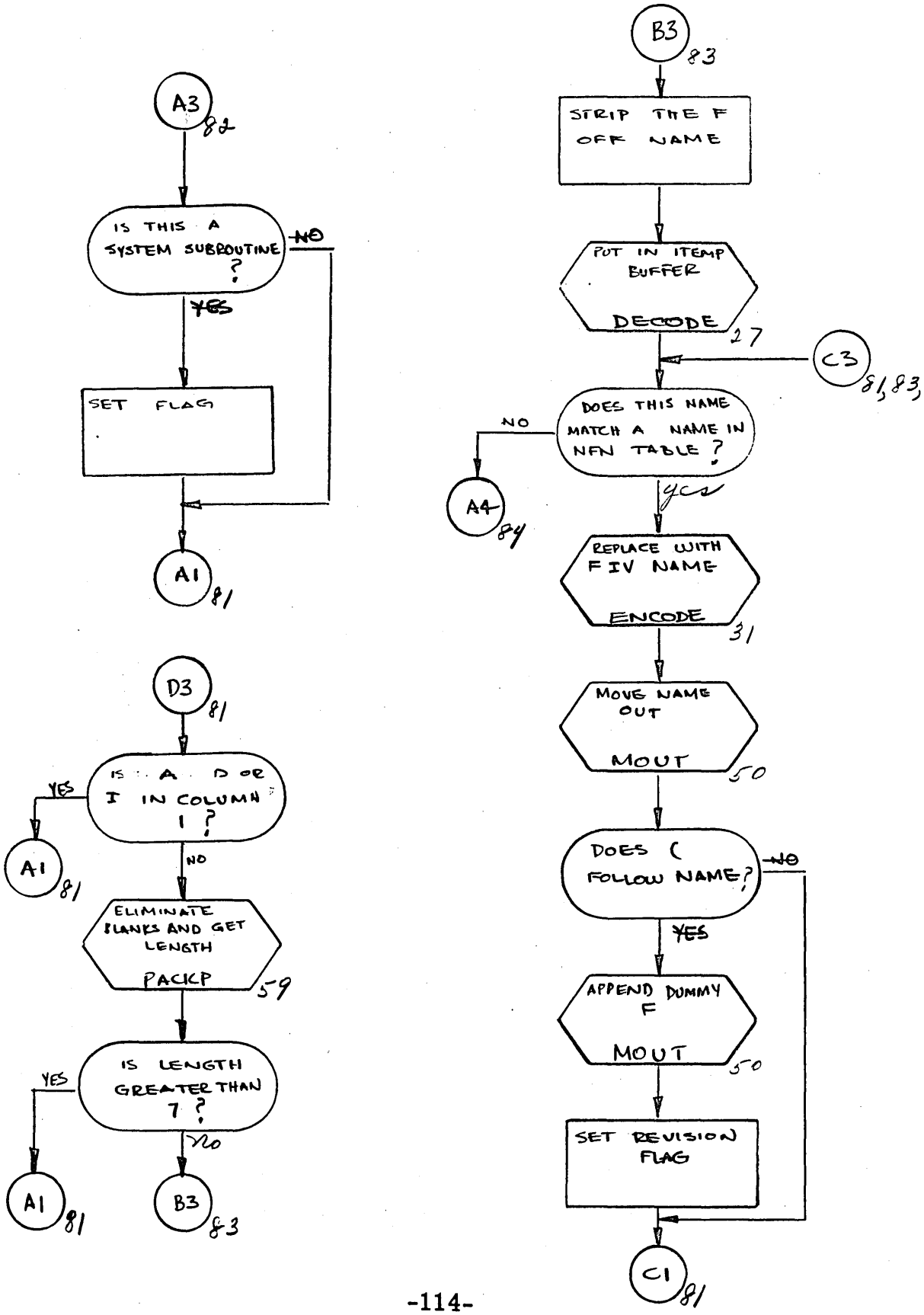
GIFT
SUBROUTINE PSCAN
REPLACE NAMES WHICH CONFLICT WITH FORTRAN IV NAMES



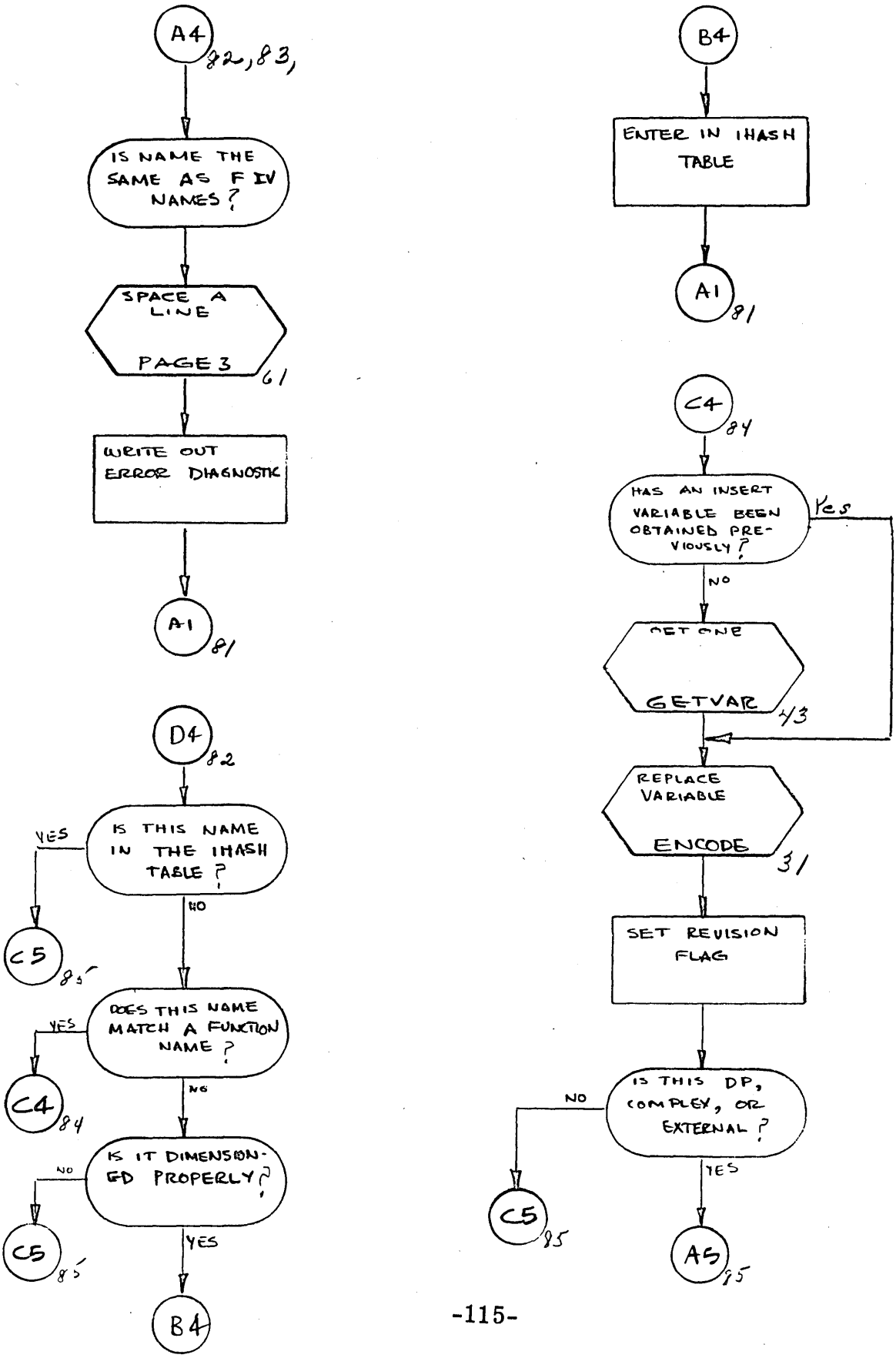
GIFT
SUBROUTINE PSCAN



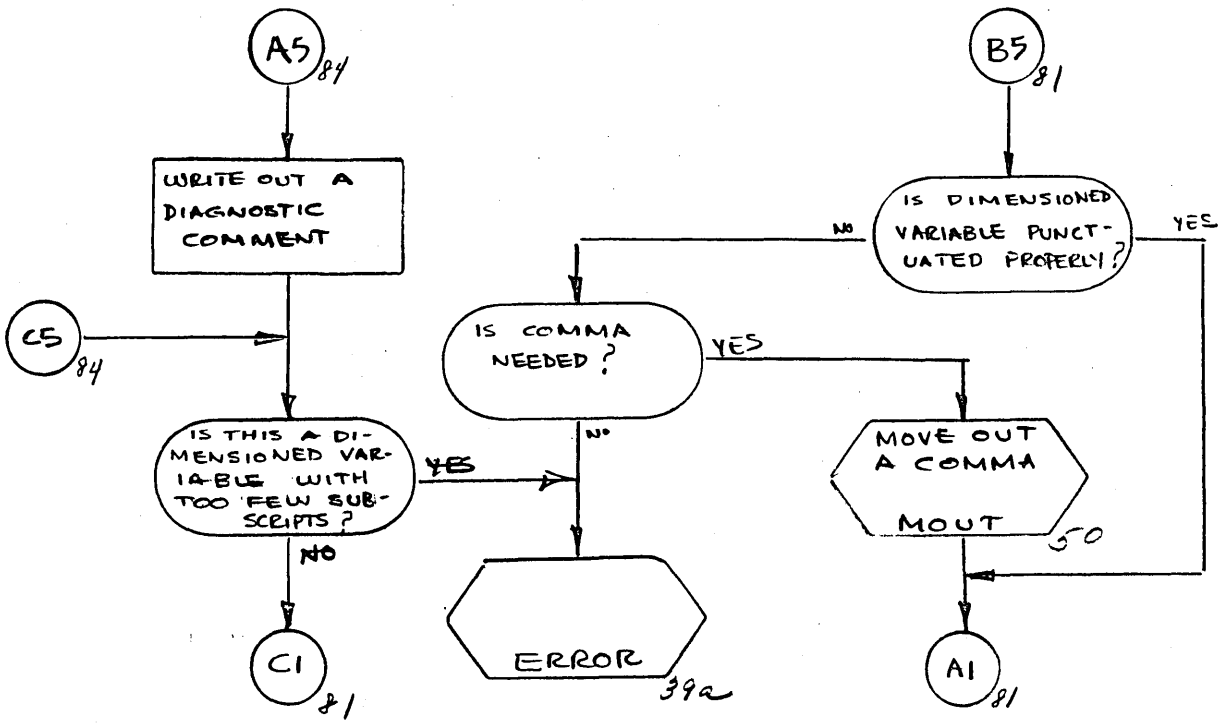
GIFT
SUBROUTINE PSCAN



GIFT
SUBROUTINE PSCAN



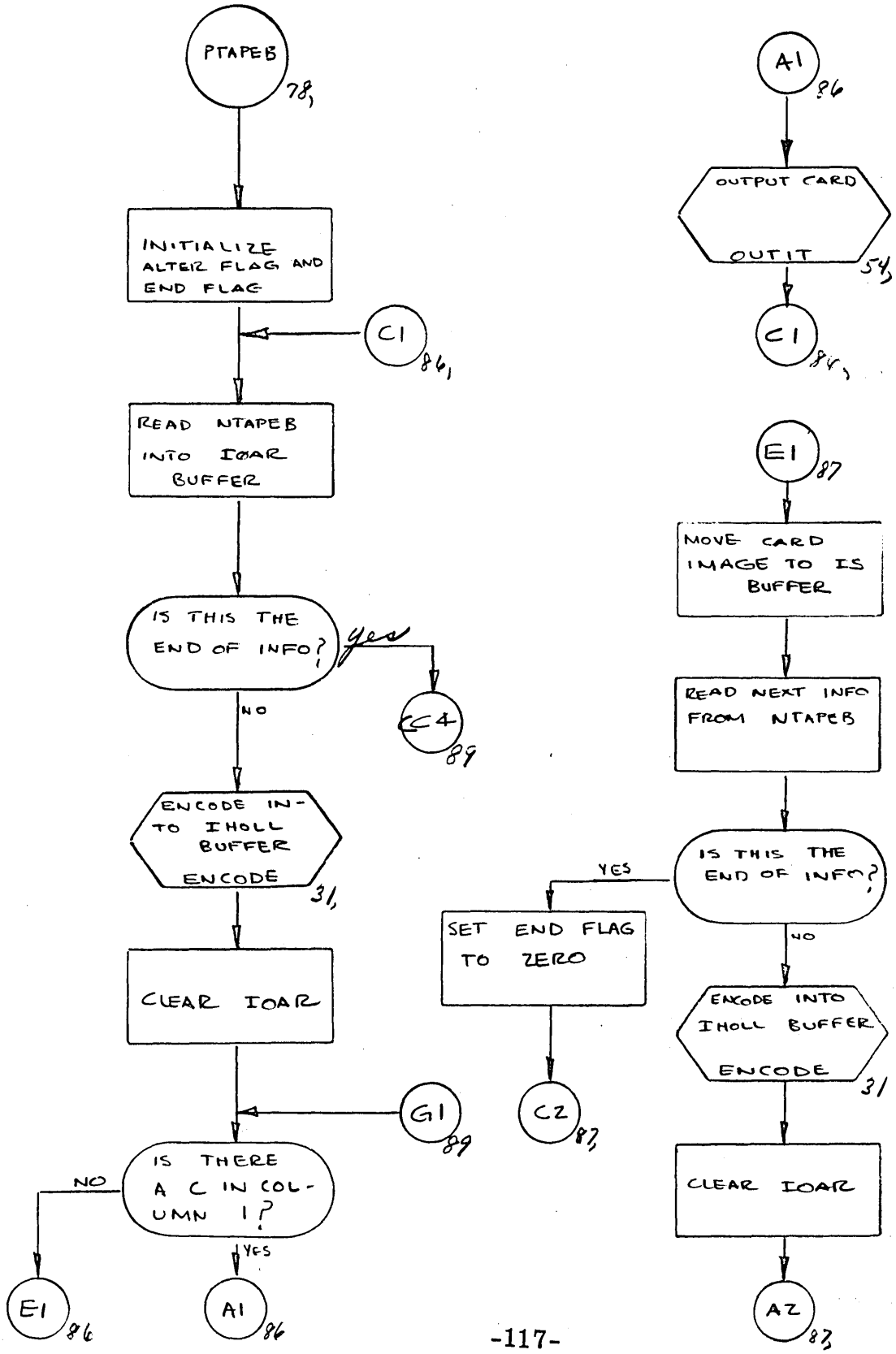
GIFT
SUBROUTINE PSCAN



GIFT

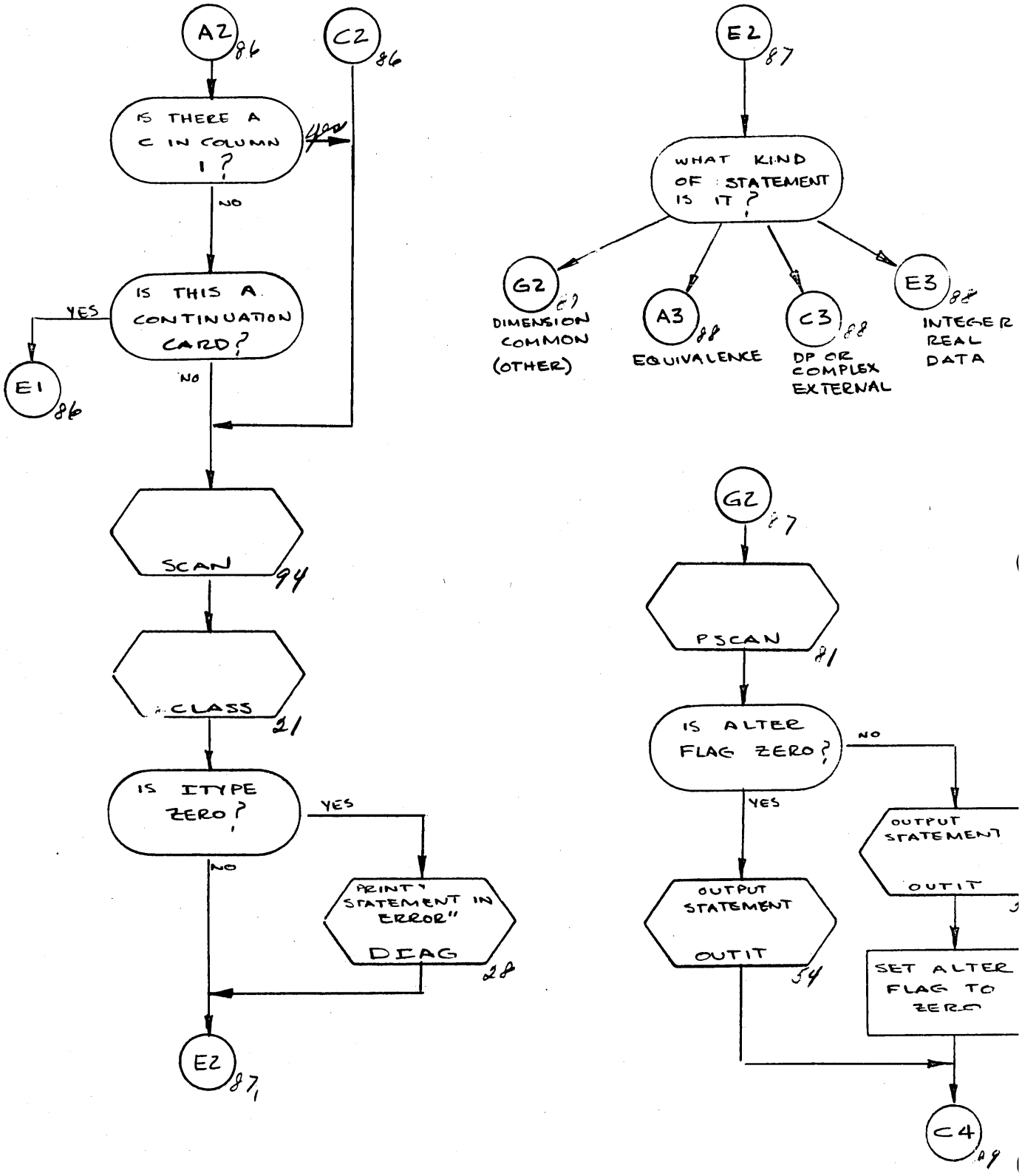
SUBROUTINE PTAPEB

PROCESS INFORMATION WRITTEN ON NTAPEB DURING PASS 1 AND PASS 2.

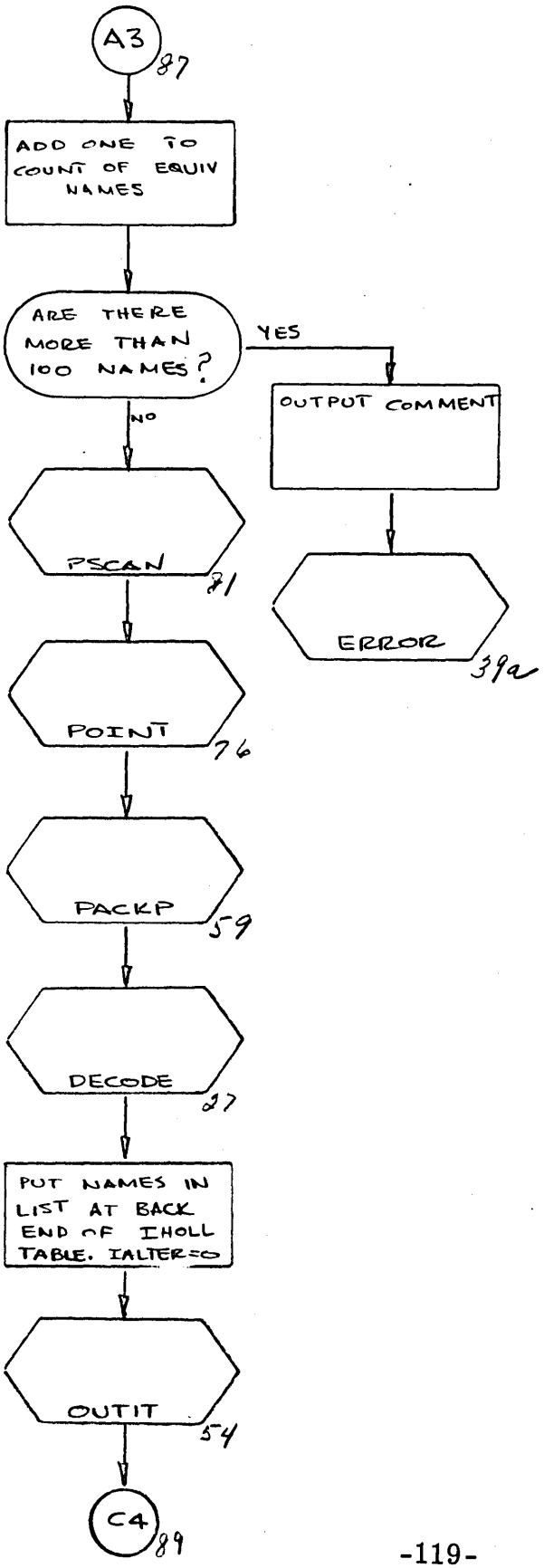


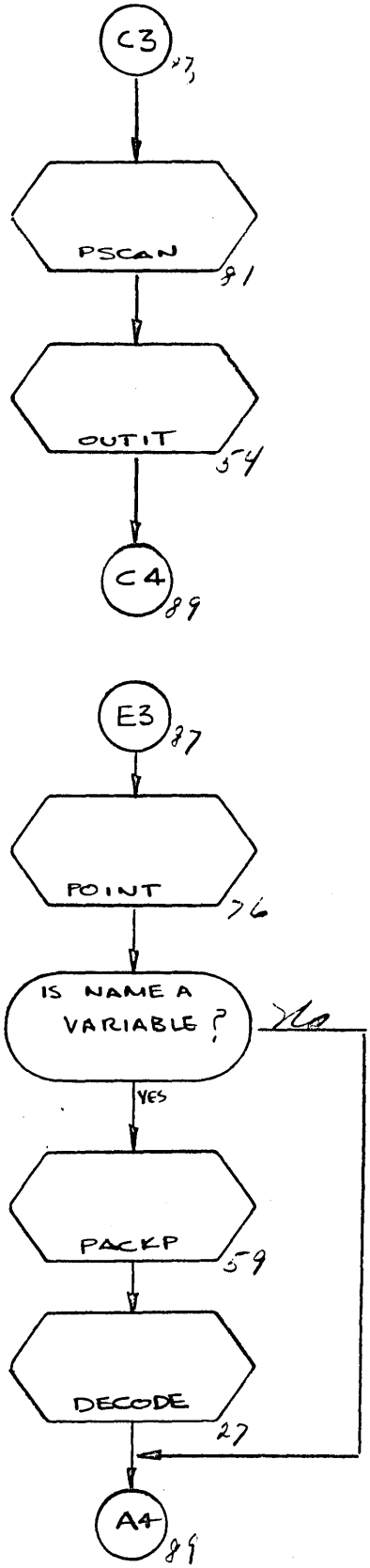
GIFT

SUBROUTINE PTAPEB

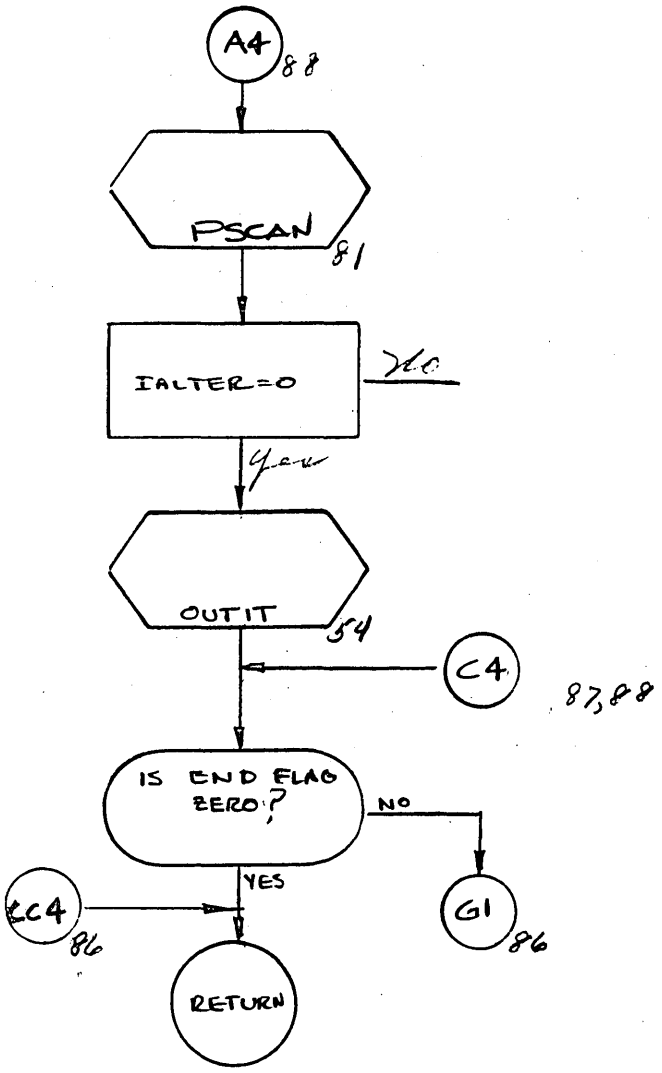


GIFT
SUBROUTINE PTAPEB





GIFT
SUBROUTINE PTAPEB



GIFT

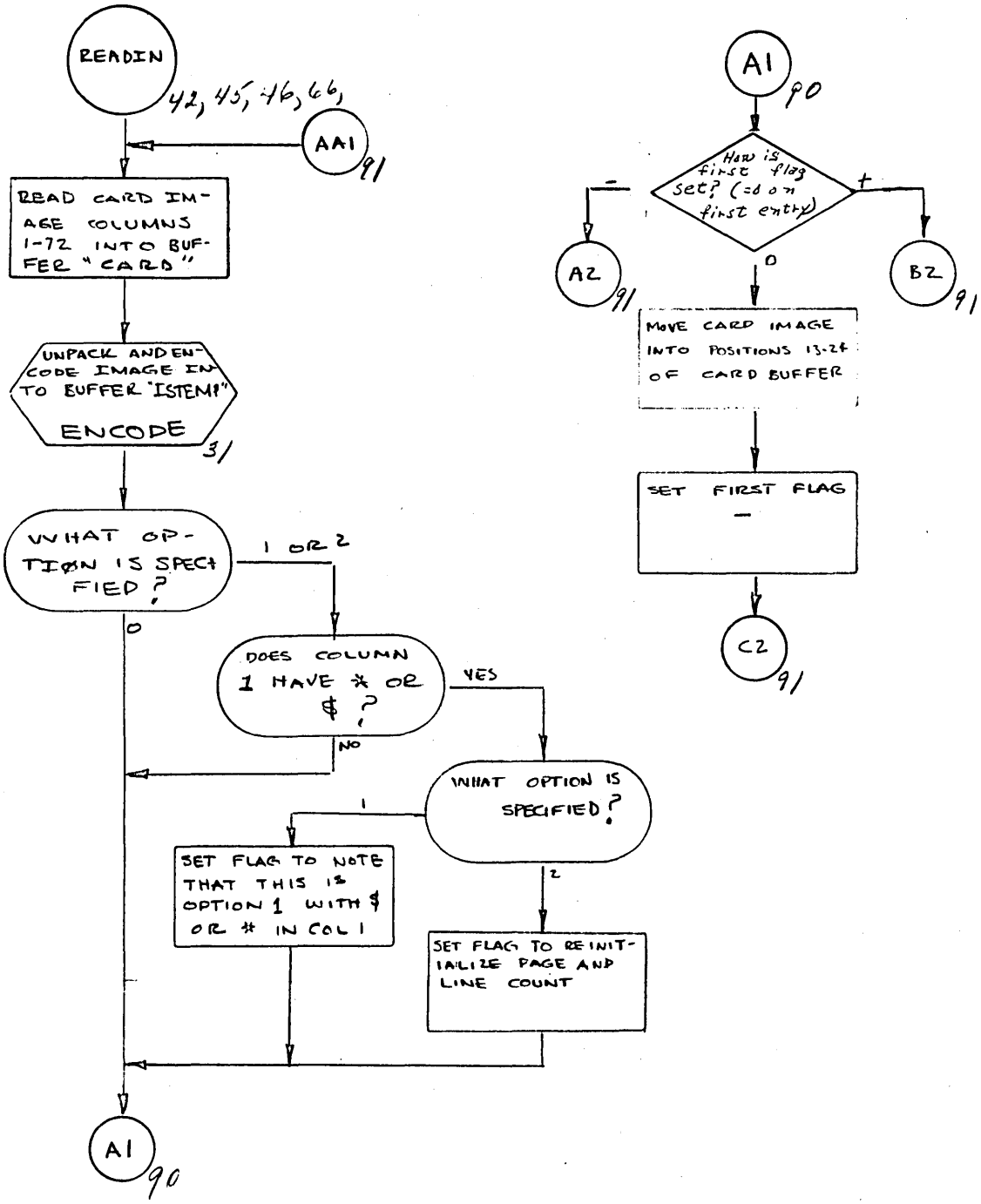
SUBROUTINE READIN

READ INPUT TAPE, PASS DATA CARDS ON OPTION 2; PASS COMMENT CARDS ON OPTION 1; PASS *ASSIGN, FORMAT GEN ON 0.

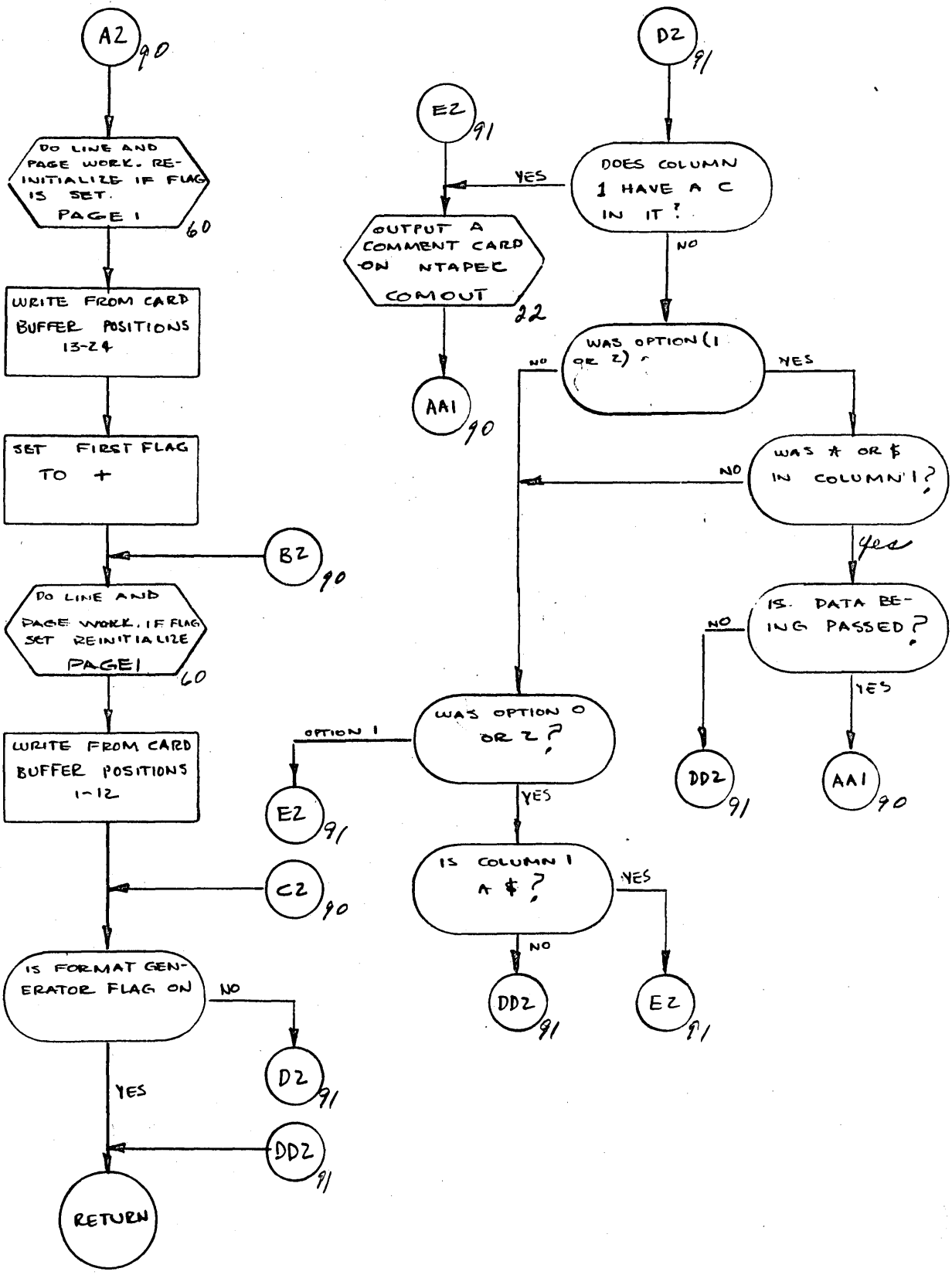
OPTION 0 - CARDS WITH * IN COLUMN 1 ACCEPTABLE

1 - CARDS WITH * IN COLUMN 1 ARE COMMENTS

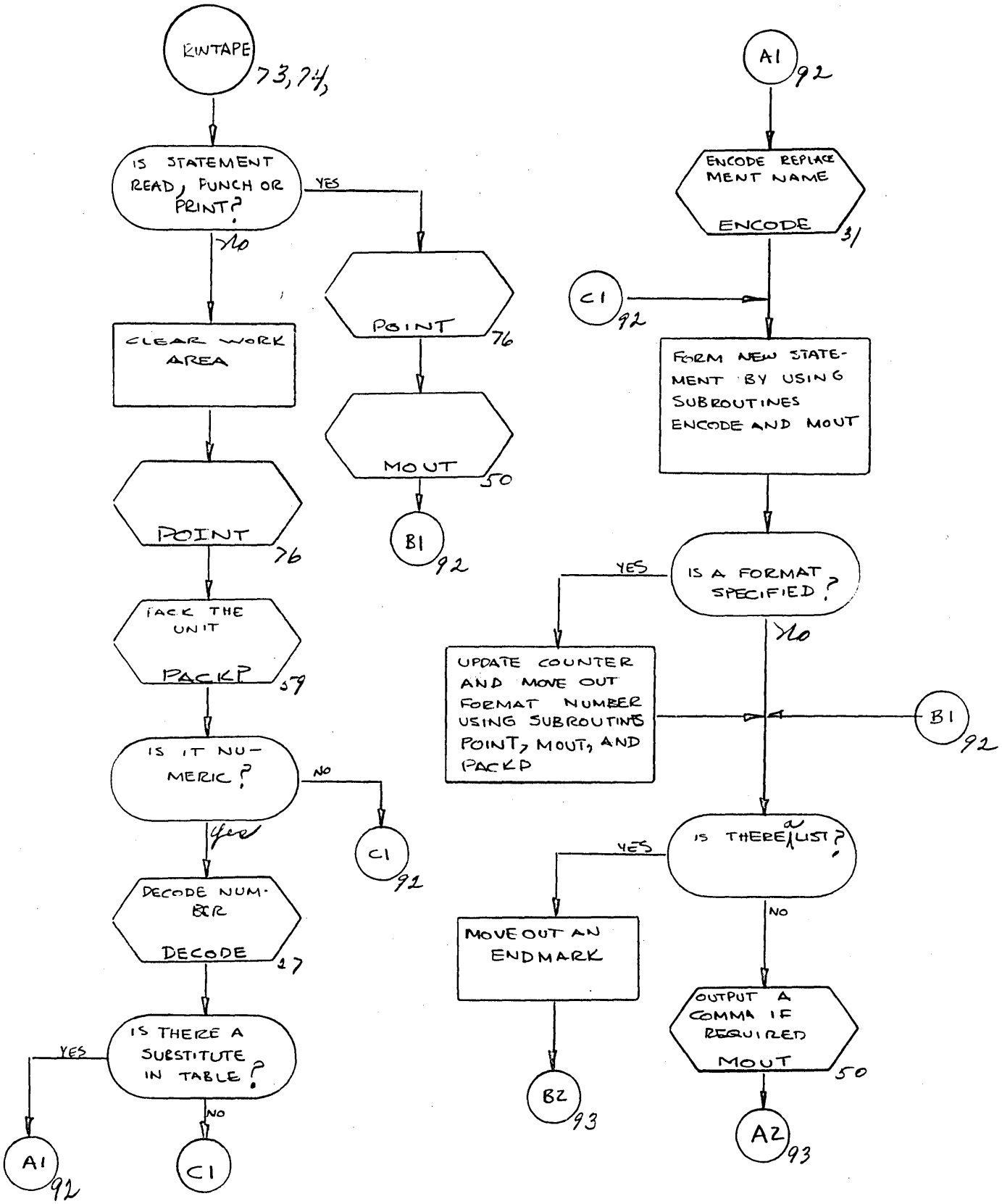
2 - CARDS WITH * OR $ IN COLUMN 1 END DATA



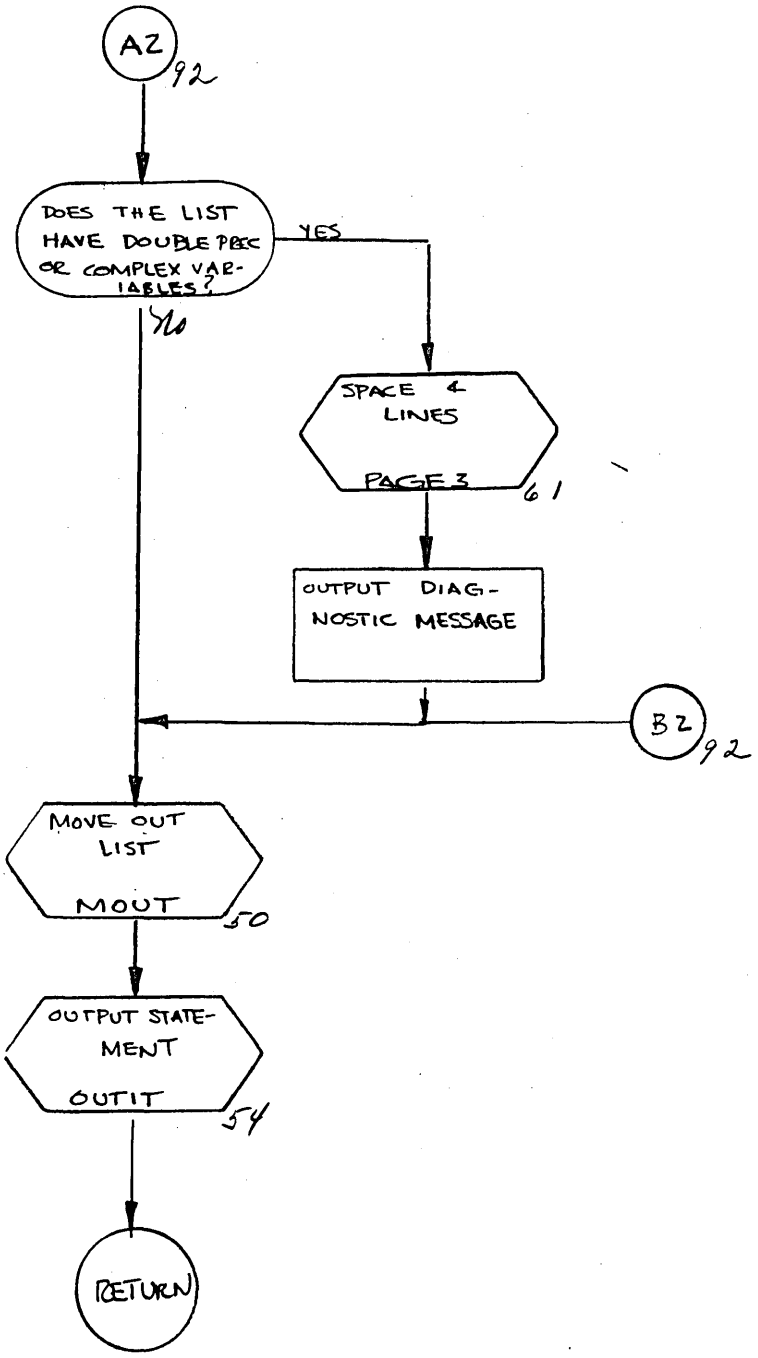
SUBROUTINE READIN



GIFT
SUBROUTINE RWTAPE
PROCESSES TAPE TRANSMISSION STATEMENTS



GIFT
SUBROUTINE RWTAPE

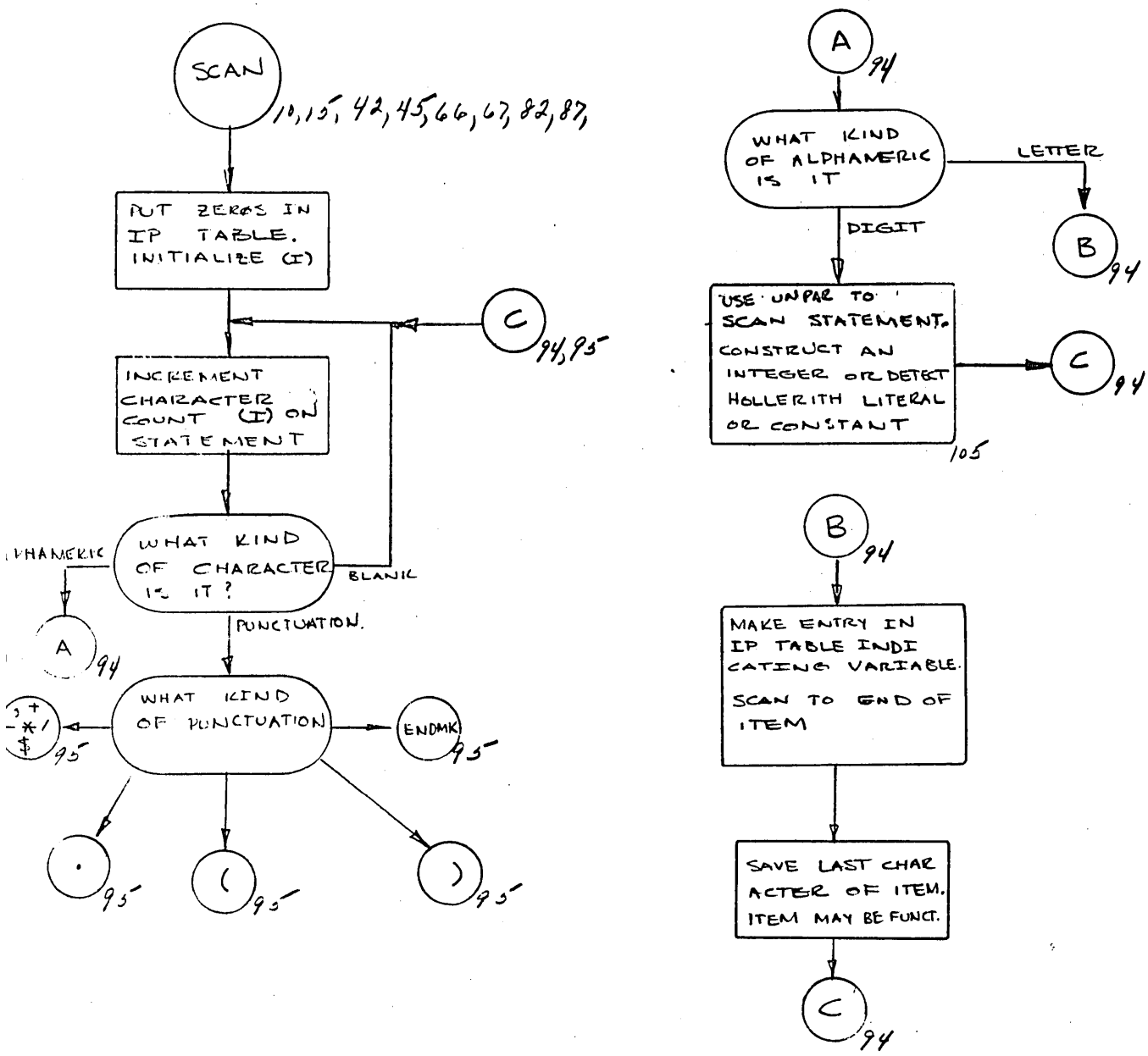


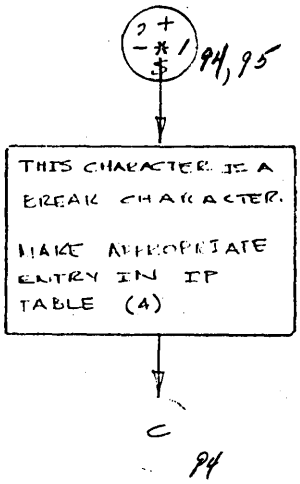
93

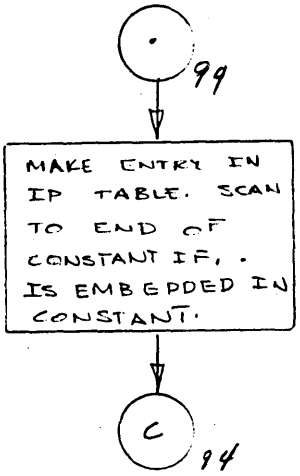
GIFT

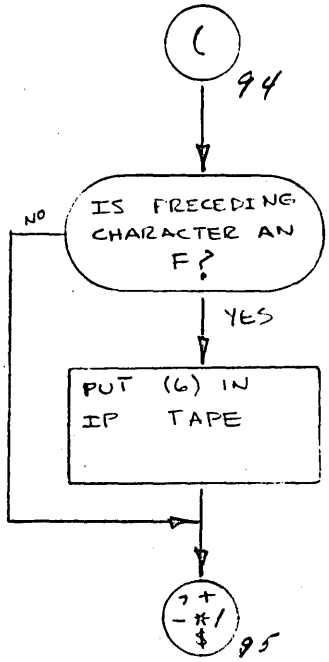
SUBROUTINE SCAN

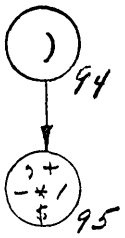
SCAN A STATEMENT. CONSTRUCT IP AND IPAR TABLE.

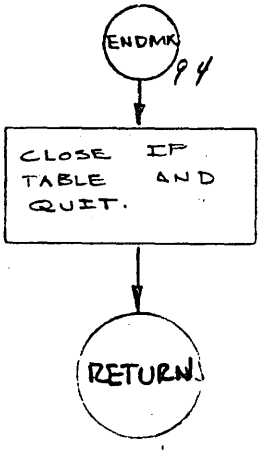



) + - * / $
94, 95
THIS CHARACTER IS A BREAK CHARACTER.
MAKE APPROPRIATE ENTRY IN IP TABLE (4)
C
94

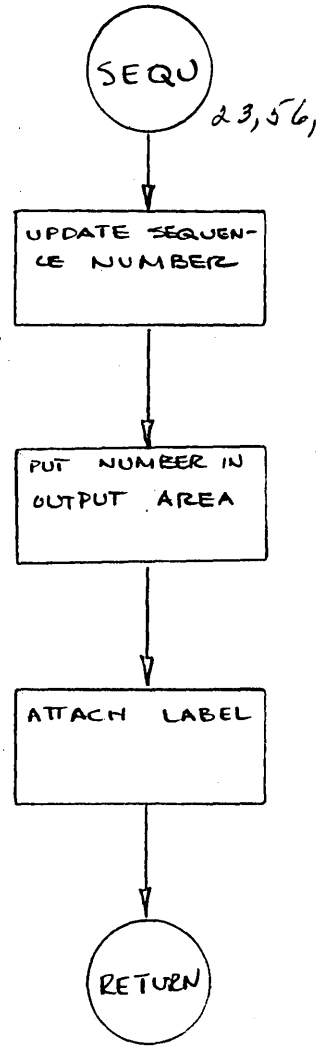

.
99
MAKE ENTRY IN IP TABLE. SCAN TO END OF CONSTANT IF, . IS EMBEDDED IN CONSTANT.
C
94


(
94
NO
IS PRECEDING CHARACTER AN F?
YES
PUT (6) IN IP TAPE
) + - * / $
95

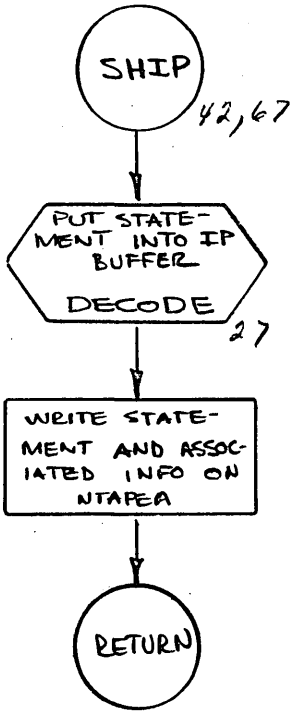

)
94
) + - * / $
95


ENDMK
94
CLOSE IP TABLE AND QUIT.
RETURN

GIFT
SUBROUTINE SEQU
PUT CORRECT SEQUENCE NUMBER IN OUTPUT AREA

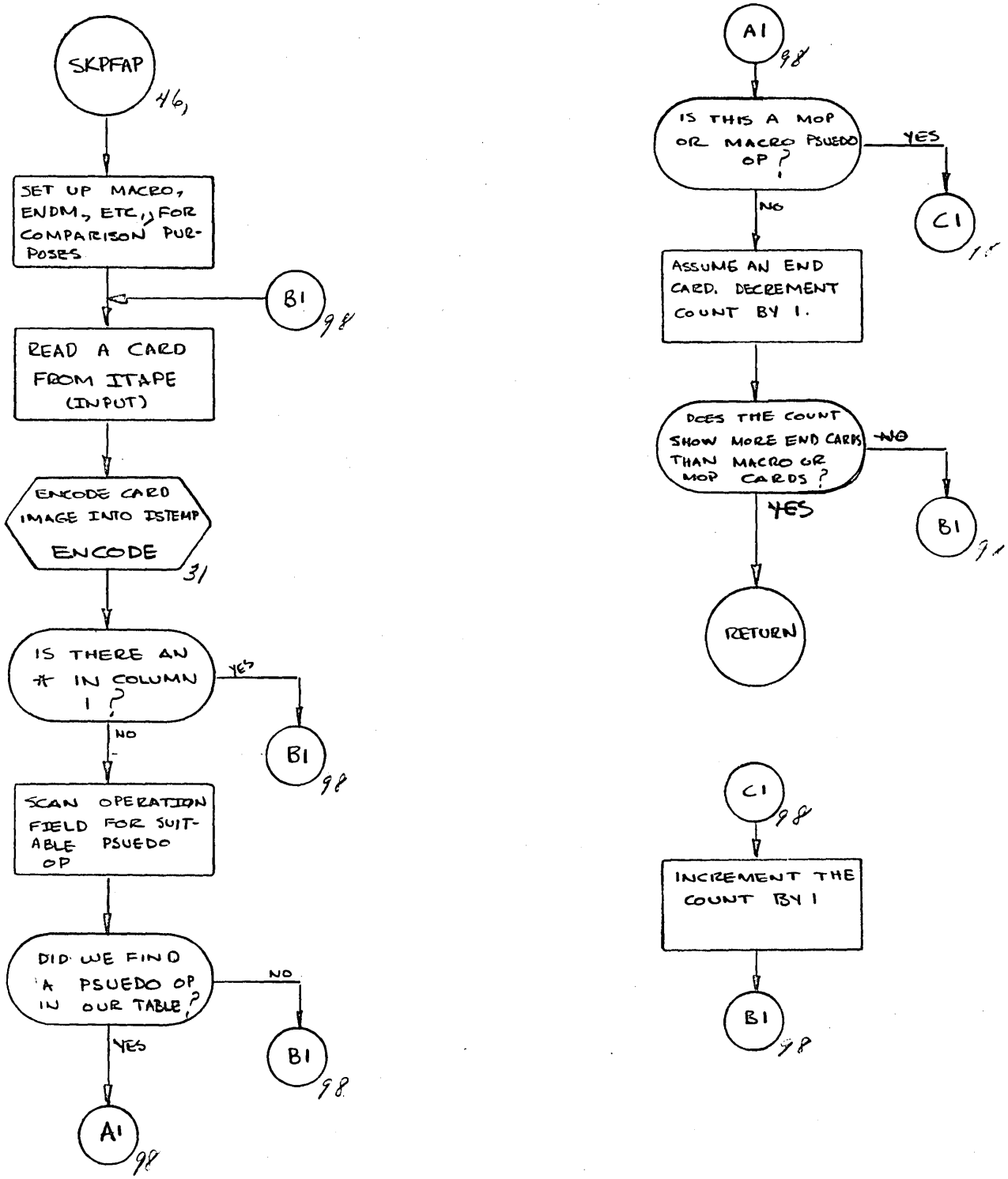


GIFT
SUBROUTINE SHIP
WRITE STATEMENT PROCESSED ON NTAPEA



SUBROUTINE SKPFAP

SKIP OVER FAP PROGRAM BY READING TO END CARD. END CARDS WHICH TERMINATE MACRO DEFINITIONS ARE IGNORED.



GIFT

SUBROUTINE SORTAC

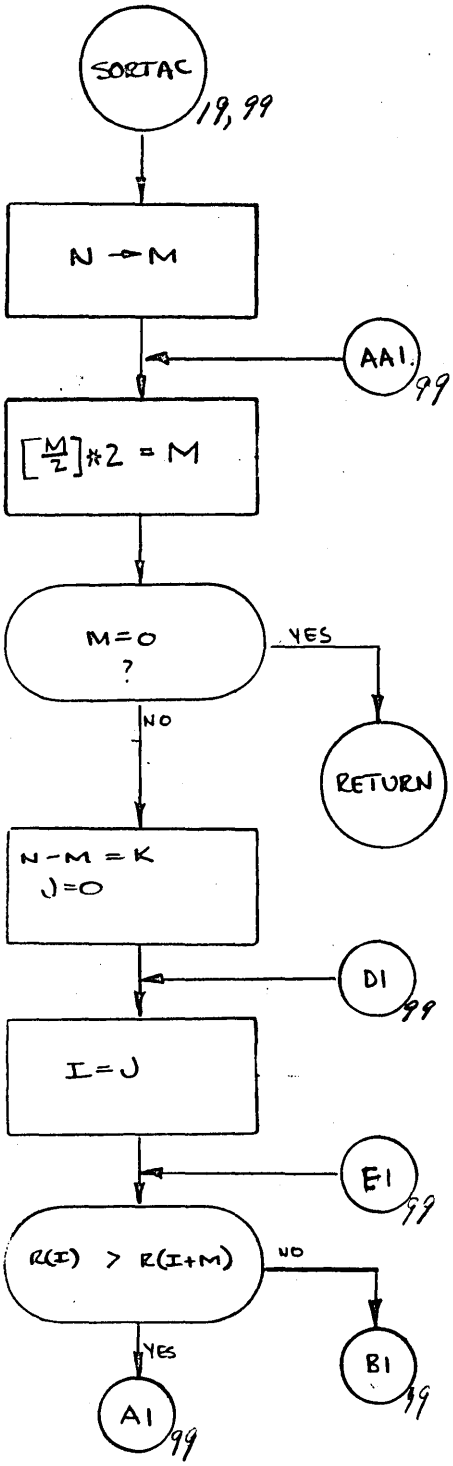
SORT SPECIFIED TABLE. DOES ASCENDING SORT.

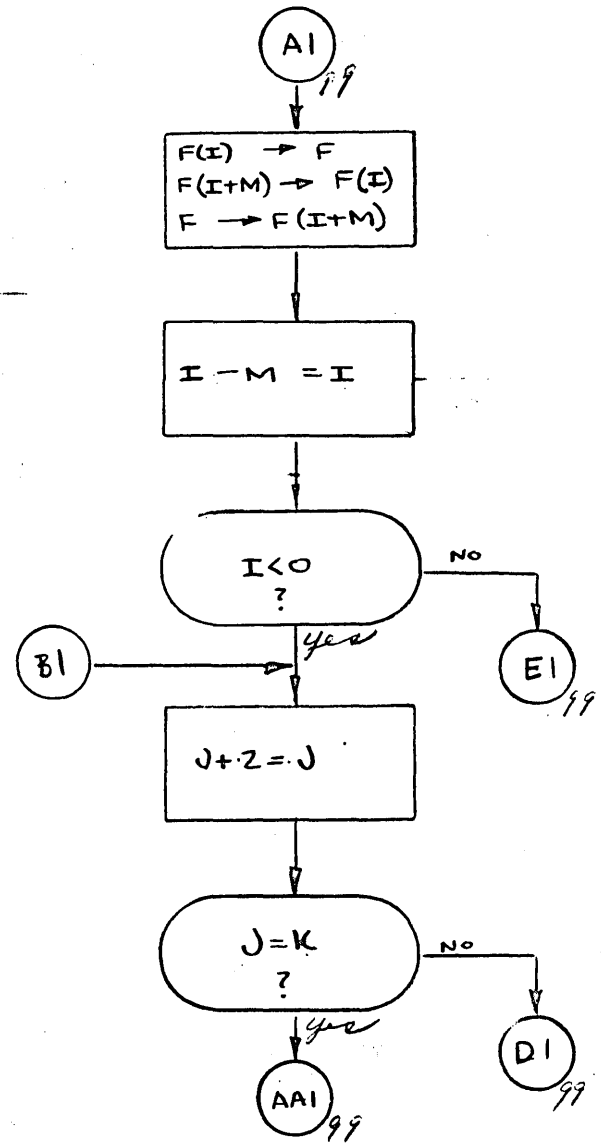
N= LENGTH OF TABLE

R(I) = AREA OF SORT IN EACH ELEMENT

F(I) = ELEMENT OF TABLE

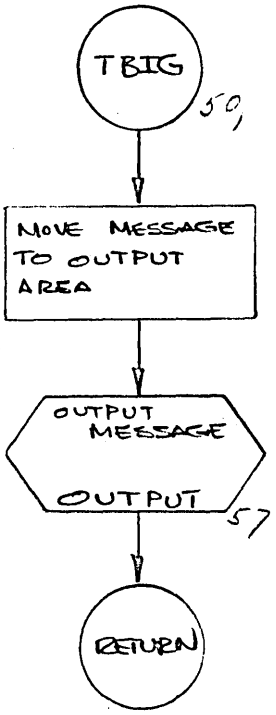
[x] = LARGEST INTEGER LESS THAN OR EQUAL TO X





99

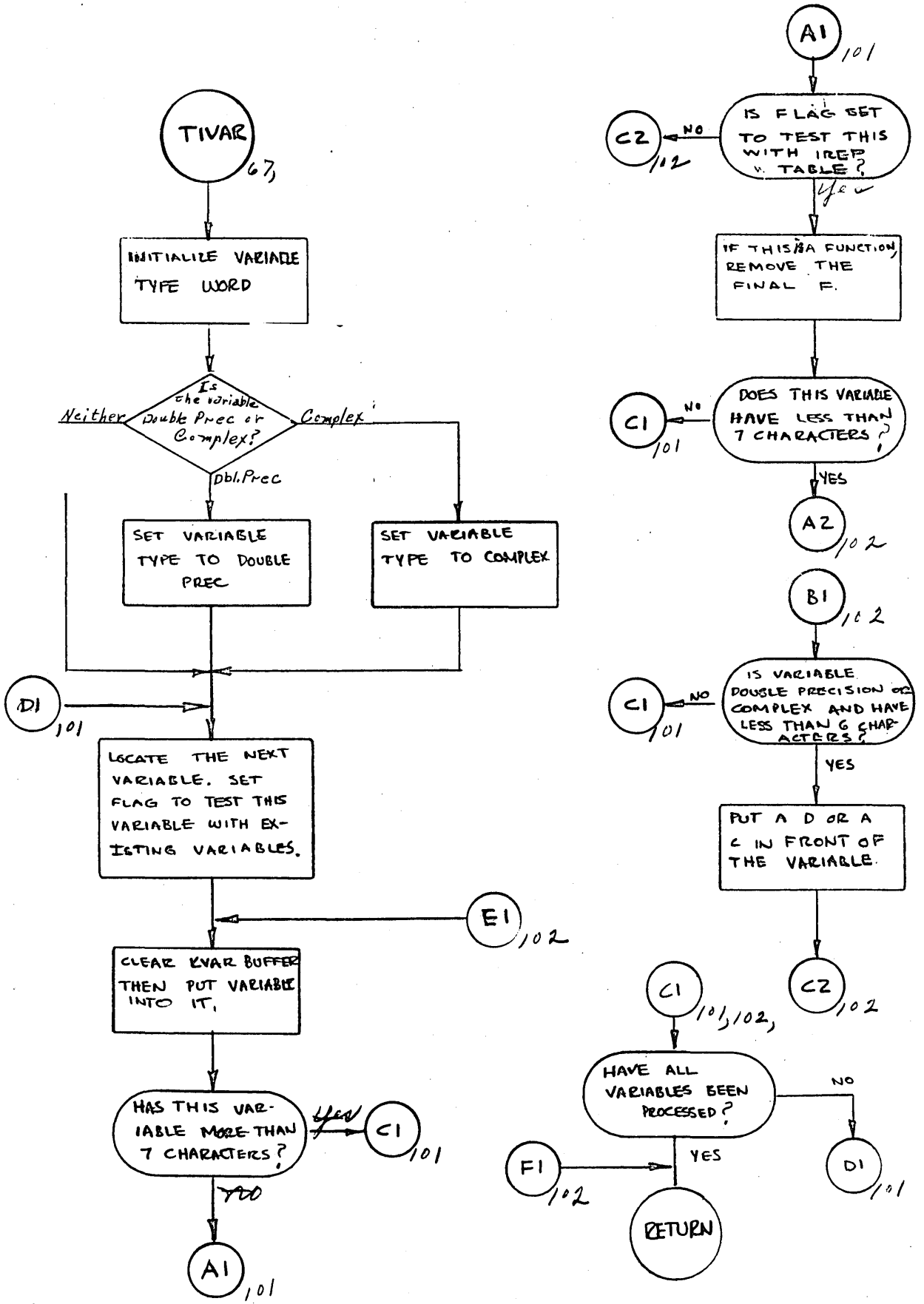
GIFT
SUBROUTINE TBIG
OUTPUT DIAGNOSTIC WHEN STATEMENT EXCEEDS TWENTY CARDS



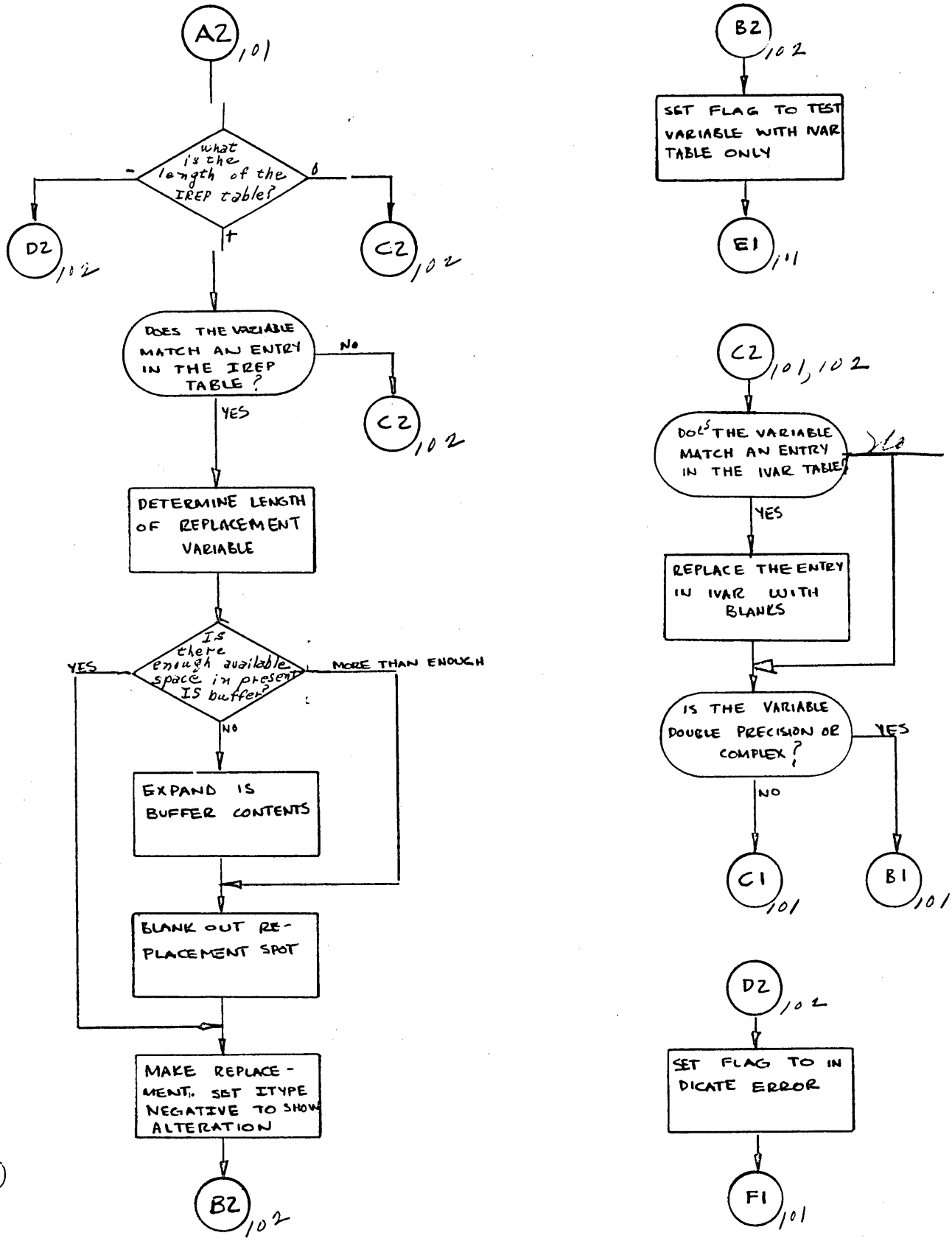
GIFT

SUBROUTINE TIVAR

EXAMINE VARIABLE NAMES. REPLACE AND DELETE CLASHING VARIABLES IN THE IVAR TABLE.

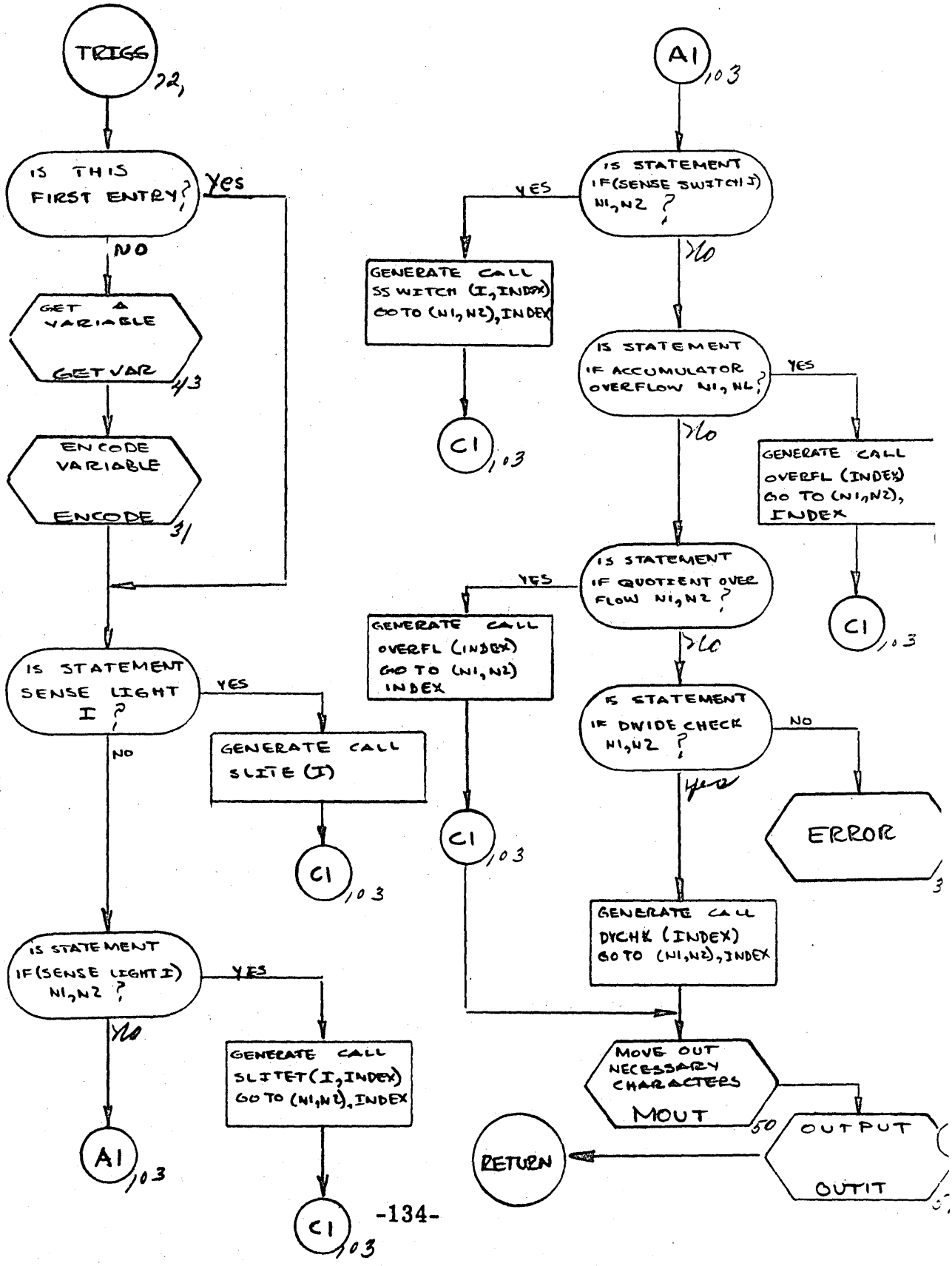


101

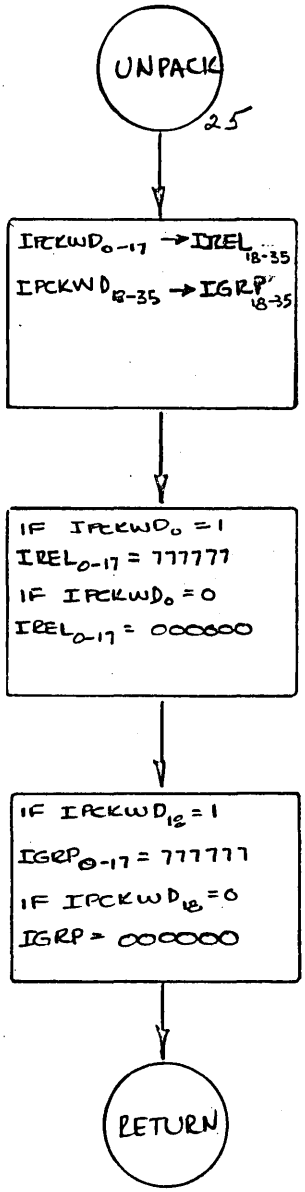

A2
101
what is the length of the IREP table?
-
0
+
D2
102
C2
102
DOES THE VARIABLE MATCH AN ENTRY IN THE IREP TABLE ?
No
YES
C2
102
DETERMINE LENGTH OF REPLACEMENT VARIABLE
Is there enough available space in present IS buffer?
YES
MORE THAN ENOUGH
NO
EXPAND IS BUFFER CONTENTS
BLANK OUT REPLACEMENT SPOT
MAKE REPLACEMENT. SET ITYPE NEGATIVE TO SHOW ALTERATION
B2
102
B2
102
SET FLAG TO TEST VARIABLE WITH IVAR TABLE ONLY
E1
101
C2
101, 102
DOES THE VARIABLE MATCH AN ENTRY IN THE IVAR TABLE?
YES
REPLACE THE ENTRY IN IVAR WITH BLANKS
IS THE VARIABLE DOUBLE PRECISION OR COMPLEX?
YES
NO
C1
101
B1
101
D2
102
SET FLAG TO INDICATE ERROR
F1
101

SUBROUTINE TRIGG

PROCESS STATEMENTS WHICH REFER TO MACHENE TRIGGERS



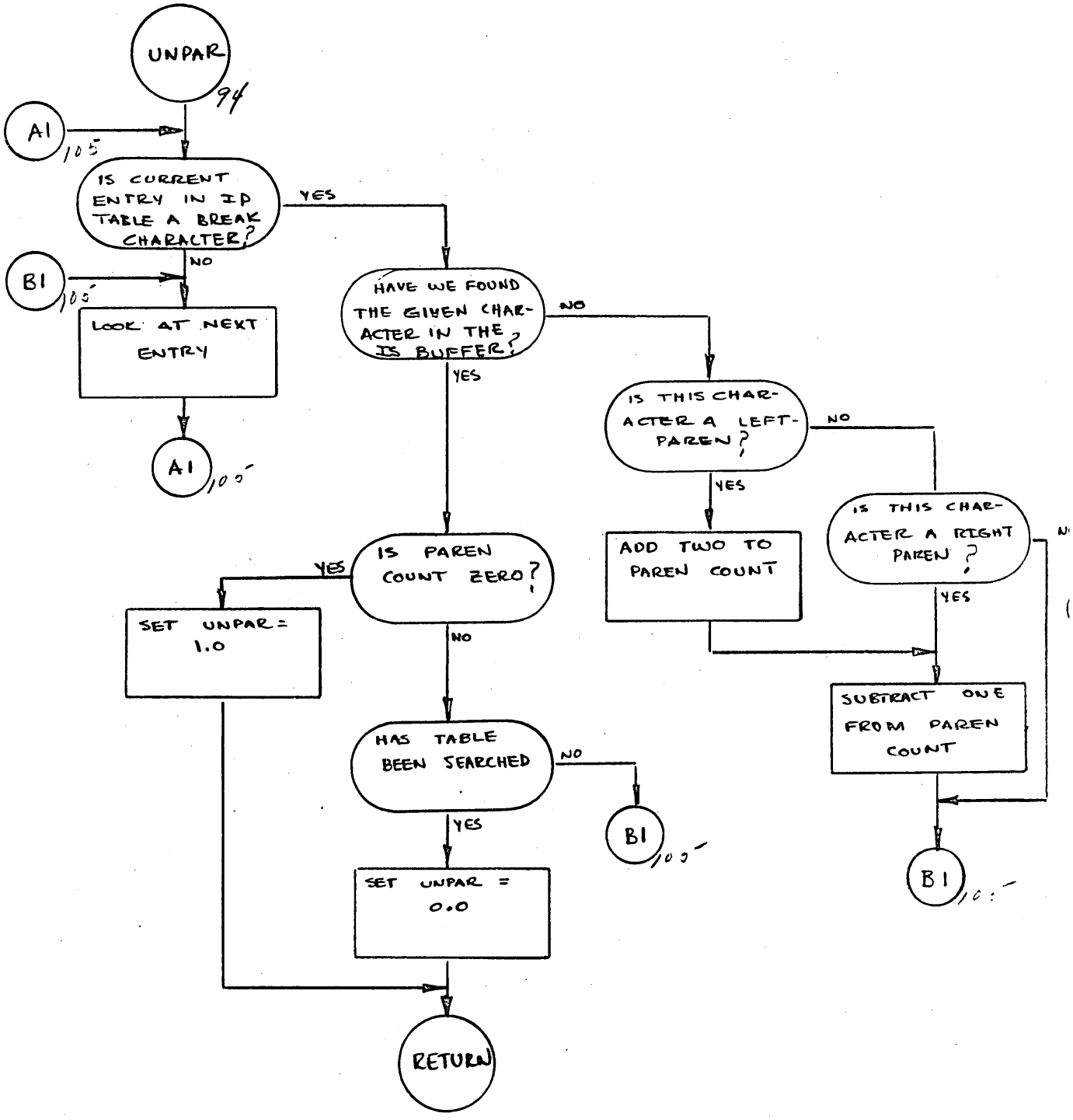
GIFT
SUBROUTINE UNPACK
PUT CONTENTS OF IPCKWD INTO IGRP AND IREL



GIT 1

FUNCTION UNPAR

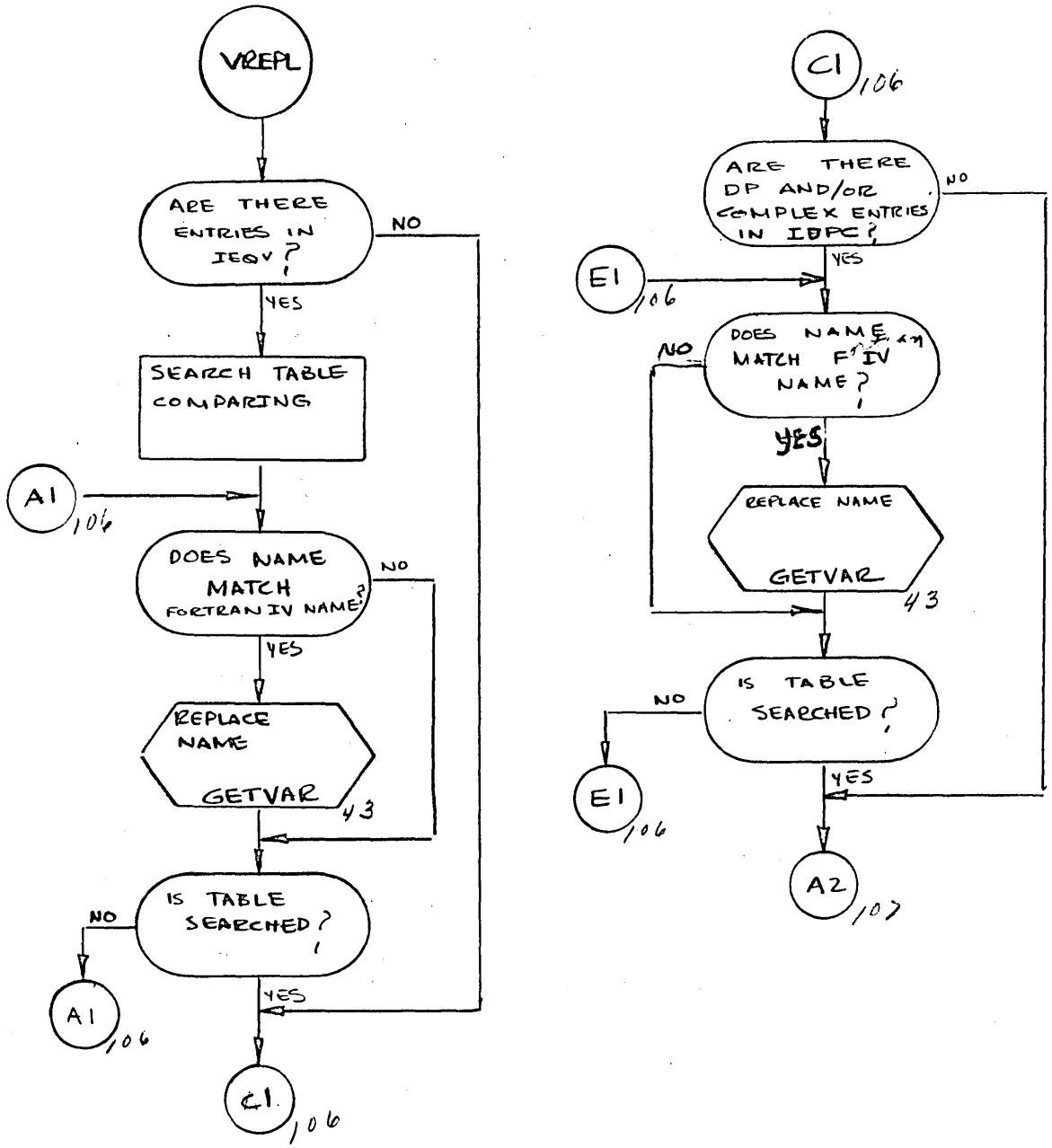
SEARCH FOR UNPARENTHESIZED INSTANCE OF GIVEN CHARACTER



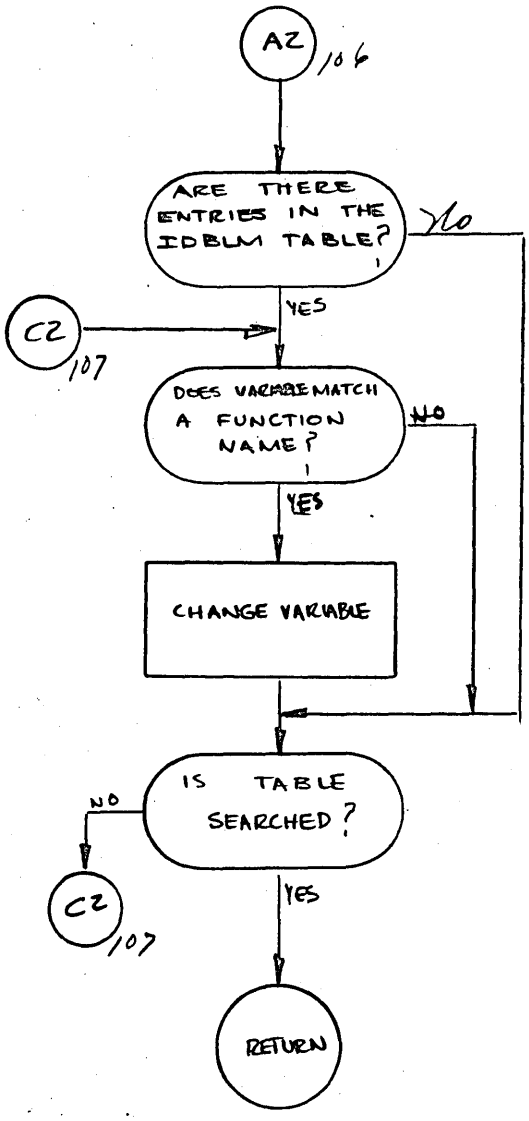
GIFT

SUBROUTINE VREPL

REPLACE NAMES IN TABLES WHICH CONFLICT WITH FORTRAN IV NAMES. CHECK TABLES IEQV, IDPC, IDBLM AND REPLACE VARIABLES WHICH CONFLICT WITH FUNCTION NAMES.



GIFT
SUBROUTINE VREPL



Progress Is Our Most Important Product

GENERAL ELECTRIC

Computer Department • Phoenix, Arizona

LITHO IN U.S.A.