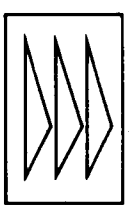
TAG TECHNICAL MANUAL

PRC R-939

22 January 1967

Prepared for

Jet Propulsion Laboratory



PLANNING RESEARCH CORPORATION
LOS ANGELES, CALIFORNIA WASHINGTON, D.C.

This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the National Aeronautics and Space Administration under Contract NAS7-100.

TAG TECHNICAL MANUAL

PRC-R-939

22 January 1967

Prepared for

Jet Propulsion Laboratory
Under Contract Number 951553

By

Kenneth Gillett
Daniel W. LaDage
Genevieve L. Michalski
John L. Overbey

PLANNING RESEARCH CORPORATION
LOS ANGELES, CALIF. WASHINGTON, D.C.

This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the National Aeronautics and Space Administration under Contract NAS7-100.

ABSTRACT

The TAG Technical Manual presents a detailed description of the Transient Analysis Generator (TAG) program, which was developed at the Jet Propulsion Laboratory of the California Institute of Technology. This manual was written primarily for the programmer who might be required to maintain or modify the TAG program; however, it should also be of significant interest to the user who desires more than a superficial understanding of the program. Included in the text are (1) a description of the analytical processes employed by TAG to generate and solve the circuit equations; (2) a brief description of the list processing techniques used to generate the equations and the FORTRAN code for the network solution program; and (3) a detailed writeup and flow chart of the two main routines and all the subroutines that comprise the TAG system.

TABLE OF CONTENTS

LIST OF EXHIBITS

I. INTRODUCTION

A. Background Summary

1. Label

TAG

2. Name

Transient Analysis Generator

3. Author

Mr. William J. Thomas

4. Sponsor

The Jet Propulsion Laboratory of the California Institute of Technology

5. Effective Date

2 August 1966

B. Program Capabilities

1. Field of Applicability

TAG is an electrical network analyzer which performs its function by means of mathematical simulation.

2. Types of Simulation

TAG is primarily a transient performance simulator; however, it will perform DC steady-state analysis as a special case of transient simulation.

3. Types of Networks

TAG will provide accurate simulation of a large class of linear and nonlinear networks that can be described by some connected set of the allowed lumped parameter elements.

4. Standard Elements

As standard circuit elements, TAG allows only the following: voltage sources, current sources, capacitors, conductances, reciprocal inductances, and ideal transformer windings.

5. Element Modifications

Any of the standard element parameters (with the exception of transformer windings) may be modified during the simulation process as a continuous or piece-wise continuous function of time, node pair voltage, or node pair voltage integral. Such functions are provided to allow modeling of nonstandard devices and must be expressed in terms of FORTRAN II arithmetic statements or closed subroutines.

6. Output

Both listed and plotted outputs may be generated by TAG. Directly available for output are time, any node pair voltage, any element value, and any of several TAG execution control variables. In addition, any variable that may be calculated from the above list of primary variables by FORTRAN arithmetic statement or closed subroutine is also available for output.

7. Organization

TAG is organized into (1) a Preprocessing program, which generates the simulation equations and imbeds them in a FORTRAN II solution program; and (2) an Execution program, which provides all the subroutines and parameter values for performing the actual network simulation specified in the solution program.

8. Computational Techniques

Equation Generation:	List Processing
Program Generation:	List Processing
Transient Solutions:	Adams-Moulton Variable Step-Size Integration
Nonlinear DC Solution:	Newton-Raphson Iteration

C. Programming and Computer System Requirements

1. Program Language

Primary: FORTRAN II

Secondary: FAP

2. Computing System

Computer:	IBM 7090/7094
Programming System:	FORTRAN II Mod. III
Auxiliary Storage:	4 scratch tapes or equivalent disk file logical units
Auxiliary Hardware:	SC4020 Stromberg-Carlson plotter

II. TAG PROGRAM DESCRIPTION

A. Equation Formulation

1. Introduction

The TAG User's Manual describes TAG descriptors and the connection list. Pass I of the preprocessor reads the connection list into the computer and forms the network equations. In this section, the important steps in equation formulation will be discussed in the order in which they are performed by the program.

2. The Connection List

We will call the connection list WLIST to conform with the TAG internal designation. WLIST consists of a sequence of descriptors which define the network.

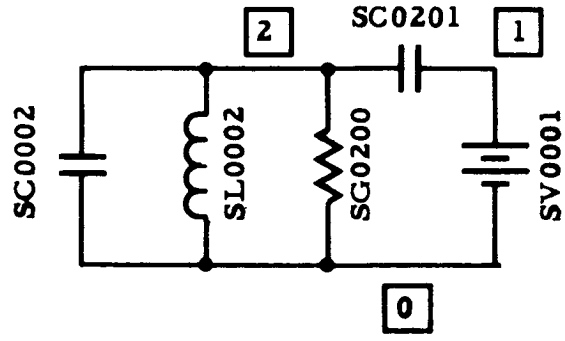


EXHIBIT 1 - SAMPLE NETWORK

Exhibit 1 has a WLIST of the following form.

WLIST = [SL0002; SG0200; SC0201; SV0001; SC0002]

The ordering of the descriptors on the input cards has an important effect upon the final network equations. We shall call the ordering of descriptors on the cards the input sequence. WLIST preserves the input sequence of descriptors.

3. The Sorted Connection List

The algorithm for selecting a proper tree, to be described in the next subsection, requires that WLIST be sorted into a new sequence. The sequence is V, C, G, L, N, and I. The new list will be called PLIST and WLIST will be saved. PLIST for Exhibit 1 is

$$\text{PLIST} = \left[\text{SV0001; SC0201; SC0002; SG0200; SL0002}\right]$$

4. Selection of the Proper Tree

A tree in a network is a subset of the branches of the network which includes all nodes but has no loops. Three possible trees of the network of Exhibit 1 are shown in Exhibit 2.

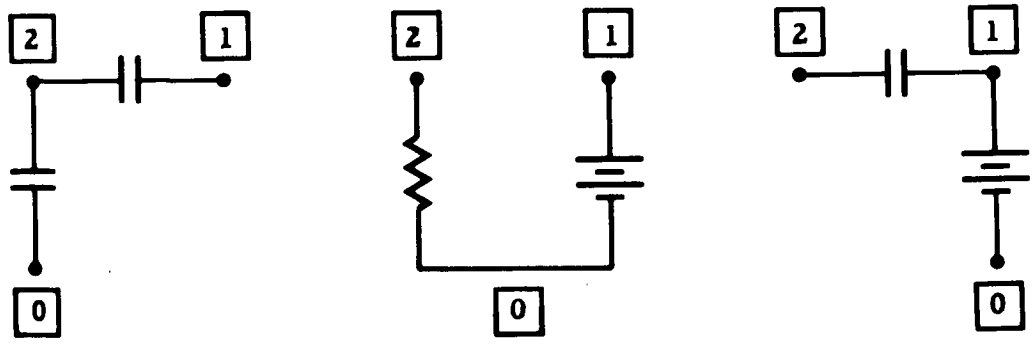


EXHIBIT 2 - TREES OF A NETWORK

A TAG proper tree is the unique tree in a network defined by the following algorithm.

Step 1. Starting from the left in PLIST, find the first descriptor which has zero as one of its nodes. Place this descriptor in a new list, FLIST. Underline the descriptor in WLIST. Underline the descriptor in PLIST only if its second node is zero.

Step 2. Starting from the left in PLIST, find a descriptor which is not underlined and has either node number equal to either node number of a descriptor in FLIST. There are three possibilities.

a. The first node only is found in FLIST. Place the descriptor in FLIST; underline it in PLIST.

b. The second node only is found in FLIST. Place the descriptor in FLIST; underline it in both FLIST and PLIST.

c. Both nodes are in FLIST. Underline the descriptor in PLIST only.

Step 3. Repeat step 2 until FLIST contains a number of descriptors equal to one less than the number of nodes.

Step 4. Sort FLIST into V, C, G, L, N, I sequence.

As an example, the proper tree for Exhibit 1 will be formed.

PLIST = [SV0001; SC0201; SC0002; SG0200; SL0002]

Step 1: PLIST = [<u>SV0001</u>; SC0201; SC0002; SG0200; SL0002]

FLIST = [SV0001]

Step 2: PLIST = [<u>SV0001</u>; <u>SC0201</u>; SC0002; SG0200; SL0002]

FLIST = [SV0001; <u>SC0201</u>]

The proper tree is shown in Exhibit 3.

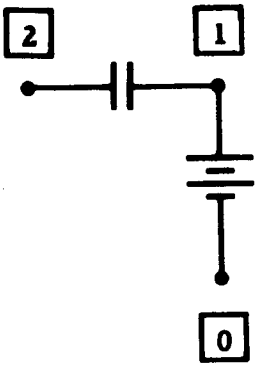


EXHIBIT 3 - PROPER TREE

5. The Voltage Basis in Networks Without Transformers

With each proper tree branch, we associate one basis (coordinate) voltage. The voltages are called a basis, in the vector sense, because they are a minimum set from which every voltage in the network can be determined. The basis voltages will be named in the following way.

V11(i)	The basis voltage associated with the ith voltage source in FLIST
V21(j)	The basis voltage associated with the jth capacitor in FLIST
V31(k)	The basis voltage associated with the kth conductance in FLIST
V41(l)	The basis voltage associated with the lth inductor in FLIST
V51(m)	The basis voltage associated with the mth ideal transformer winding in FLIST

A current source may never enter FLIST and thus there is no assigned basis voltage name.

Exhibit 4 gives the relationship between the basis voltages and the corresponding tree branch voltages.

EXHIBIT 4 - RELATIONSHIP BETWEEN BASIS VOLTAGES AND BRANCH VOLTAGES

Relationship	FLIST Entry
V11(i) = SVNNNP	SVNNNP (ith V)
V11(i) = -SVNNNP	SVNNNP
V21(j) = SVNNNP	SCNNNP (jth C)
V21(j) = -SVNNNP	SCNNNP
V31(k) = SVNNNP	SGNNNP (kth G)
V31(k) = -SVNNNP	SGNNNP
V41(l) = SVNNNP	SLNNNP (lth L)
V41(l) = -SVNNNP	SLNNNP
V51(m) = SVNNNP	SNNNNP (mth N)
V51(m) = -SVNNNP	SNNNNP

As an example, the FLIST for Exhibit 3 was

$$\text{FLIST} = \left[\text{SV0001; } \underline{\text{SC0201}}\right]$$

The basis voltages are then V11(1) and V21(1). They are related to the branch voltages as follows.

$$\text{V11(1)} = \text{SV0001}$$

$$\text{V21(1)} = \text{-SV0201}$$

The minus sign occurs because SC0201 was underlined in FLIST.

6. The Tree Branch and Voltage Basis Reference Systems

From the above discussion it is apparent that the basis voltage reference system is different from the tree branch voltage reference system.

The second node in the descriptor is always assumed to be the positive reference for the branch voltage. Exhibit 5 shows a tree with its user assigned branch reference system.

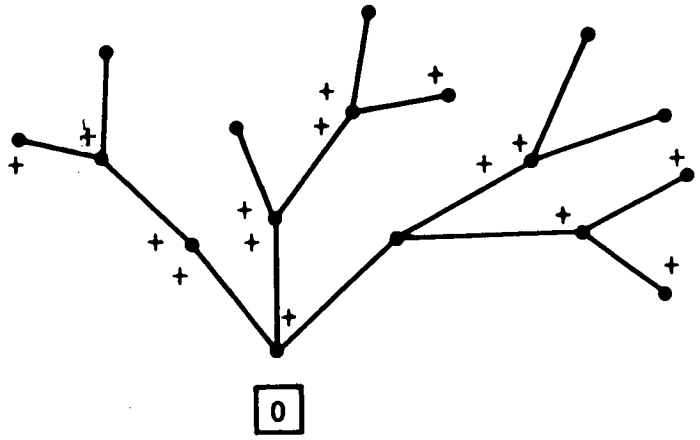


EXHIBIT 5 - TREE WITH USER ASSIGNED BRANCH REFERENCE SYSTEM

Note that the two nodes of every branch in a tree can be classified as closest to the zero node or farthest from the zero node. In choosing the references for the voltage basis, TAG always puts the plus sign on the node of a branch which is farthest from the zero node. Exhibit 6 shows the tree of Exhibit 5 with the TAG voltage basis reference system placed upon it.

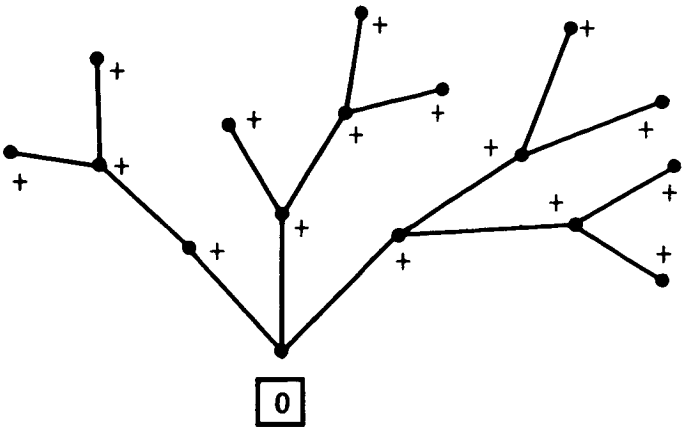
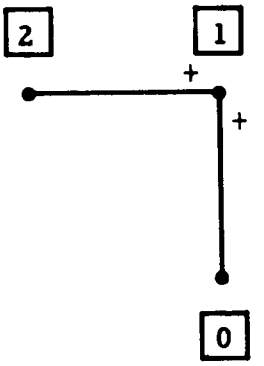
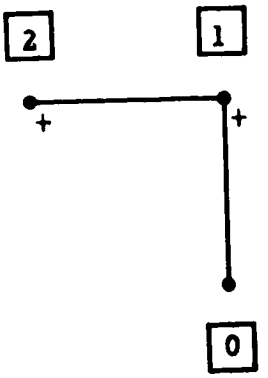


EXHIBIT 6 - TAG VOLTAGE BASIS REFERENCE SYSTEM

Using Exhibit 3 as an example, we have the following user assigned branch reference system



and the following TAG voltage basis reference system.



7. The Initial Coordinate Transformation Matrix

Since a tree touches every node in a network, and is connected, we can always find the value of a node voltage by summing the basis voltages along the path which connects the node in question with the zero node. Using Exhibit 3, we have as an example

$$V_2 = V21(1) + V11(1)$$

$$V_1 = V11(1)$$

Note that, because of the TAG reference system, the summations will always have terms with positive coefficients only.

If the network has N nodes, one of which is the zero or ground node, there are N - 1 node voltages which can be expressed as sums of the N - 1 basis voltages. The N - 1 by N - 1 matrix TC expresses these relationships.

$$VN = TC * V$$

where VN is the node voltage vector and V is the basis voltage vector. The matrices and vectors are defined as follows.

$$VN = \left[V_i \right] \qquad i = 1, 2, \cdots, N - 1$$

$$V = \begin{bmatrix} V11 \\ V21 \\ V31 \\ V41 \\ V51 \end{bmatrix}$$

$$V11 = V11(i) \qquad i = 1, 2, \cdots, NV$$

$$V21 = V21(i) \qquad i = 1, 2, \cdots, NC$$

$$V31 = V31(i) \qquad i = 1, 2, \cdots, NG$$

$$V41 = V41(i) \qquad i = 1, 2, \cdots, NL$$

$$V51 = V51(i) \qquad i = 1, 2, \cdots, NX$$

$$TC = \left[tc_{ij}\right] \qquad i = 1, 2, \cdots, N - 1 \; ; \; j = 1, 2, \cdots, N - 1$$

$tc_{ij} = 1$ if the jth basis voltage is in the path from node i to node 0. In all other cases $tc_{ij} = 0$.

N = the number of nodes in the network

NV = the number of voltage sources in the tree

NC = the number of capacitors in the tree

NG = the number of conductances in the tree

NL = the number of inductors in the tree

NX = the number of transformer windings in the tree

The previous example may now be expressed in matrix notation.

$$VN = TC * V$$

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} V11(1) \\ V21(1) \end{bmatrix}$$

8. The Ideal Transformer Constraint Matrix

The equations of an ideal transformer require that winding voltages be related in the following way.

$$n_j v_{ti} - n_i v_{tj} = 0$$

where n_i and v_{ti} are the turns and voltage of the ith winding. The symbols n_j and v_{tj} represent the turns and voltage for the jth winding. It is assumed here that a set of correct references has been assigned to the transformer windings. If a transformer has W windings, there are W - 1 independent voltage equations which may be written for its windings. The number of sets of W - 1 equations for a given transformer is large. It is convenient to choose a set of the form

$$\frac{v_{t1}}{n_1} - \frac{v_{ti}}{n_i} = 0 \qquad i = 2, 3, \cdots, W$$

Since each winding voltage may be expressed as the difference between two node voltages, in terms of node voltages, the above equations become the following

$$\frac{V_a - V_b}{n_1} - \frac{V_c - V_d}{n_i} = 0 \qquad i = 2, 3, \cdots, W$$

V_a and V_b are the voltages of the nodes to which the first winding is connected; V_c and V_d are similar quantities for the ith winding. a and c are the positive terminals of the two windings.

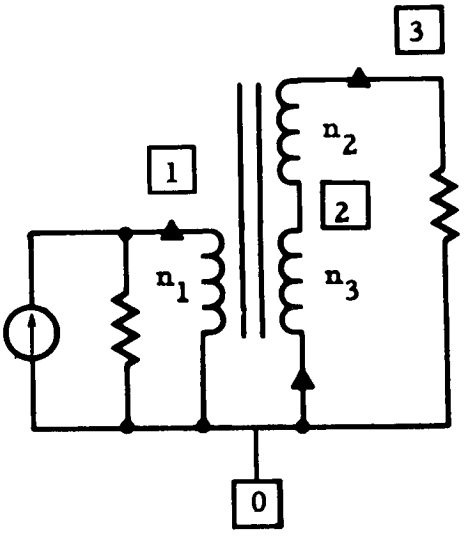


EXHIBIT 7 - NETWORK WITH IDEAL TRANSFORMER

As an example, the transformer equations will be written for Exhibit 7.

$$n_2 v_{t1} - n_1 v_{t2} = 0$$

$$n_3 v_{t1} - n_1 v_{t3} = 0$$

The winding voltages may be expressed in terms of node voltages

$$v_{t1} = V_1$$

$$v_{t2} = V_3 - V_2$$

$$v_{t3} = V_0 - V_2 = 0 - V_2 = -V_2$$

and substituted into the transformer equations

$$n_2 V_1 - n_1 (V_3 - V_2) = 0$$

$$n_3 V_1 - n_1 (-V_2) = 0$$

$$n_2 V_1 + n_1 V_2 - n_1 V_3 = 0$$

$$n_3 V_1 + n_1 V_2 = 0$$

If we have W ideal transformer windings in the network and T transformers, there will be W - T independent transformer equations which can be written. In TAG this number is called NRR.

$$NRR = W - T$$

In matrix notation the transformer equations are

$$TTR * VN = 0$$

The matrix TTR has NRR rows and N - 1 columns. Each row of TTR represents one transformer voltage equation. The transformer equations for Exhibit 7 are then

$$TTR * VN = 0$$

$$\begin{bmatrix} n_2 & n_1 & -n_1 \\ n_3 & n_1 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = 0$$

The ordering of the rows of TTR is not mathematically important. However, TAG groups them by transformer as described in the XFORM subroutine writeup.

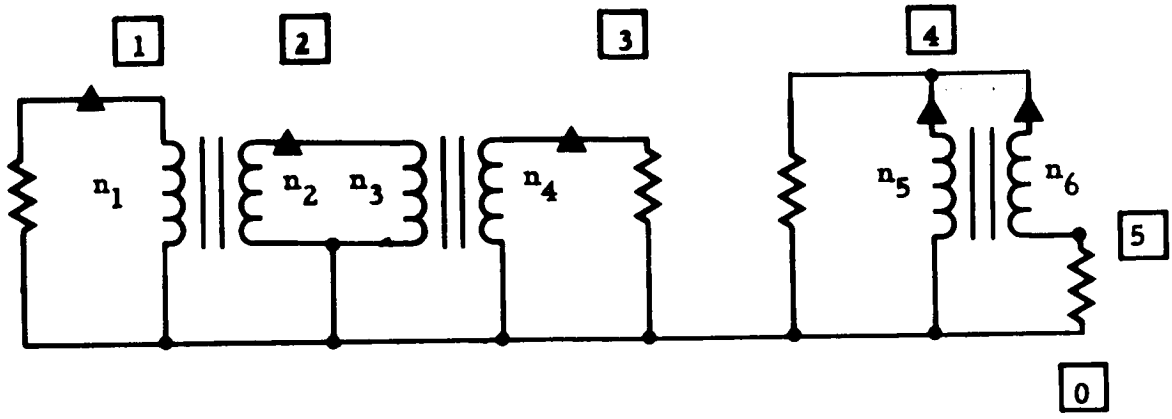


EXHIBIT 8 - MULTIPLE TRANSFORMERS

As a final example, Exhibit 8 shows a network with three transformers and six windings. There are then three transformer equations and five node voltages.

$$N = 6$$
$$T = 3$$
$$NRR = 6 - 3 = 3$$
$$N = 6$$
$$N - 1 = 5$$
$$TTR * VN = 0$$

$$\begin{bmatrix} n_2 & -n_1 & 0 & 0 & 0 \\ n_4 & 0 & -n_3 & 0 & 0 \\ 0 & 0 & 0 & (n_6 - n_5) & n_5 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{bmatrix} = 0$$

9. Elimination of Voltages From the Basis

Since there are NRR independent transformer equations, it is possible to express NRR node voltages in terms of the remaining node voltages. TAG, however, expresses the transformer equations in terms of the basis voltages

$$TTR * VN = 0$$

$$VN = TC * V$$

$$TTR * TC * V = TQ * V = 0$$

and then eliminates basis voltages. Because TTR has a rank of NRR and TC is nonsingular, TQ will have rank NRR. Thus NRR basis voltages may be expressed as linear combinations of N - 1 - NRR remaining voltages.

The first step in the elimination is to apply Jordan's[1] method to the equations. We try to solve for voltages on the bottom of V and work up until we have solved for NRR basis voltages. The process is best illustrated by an example.

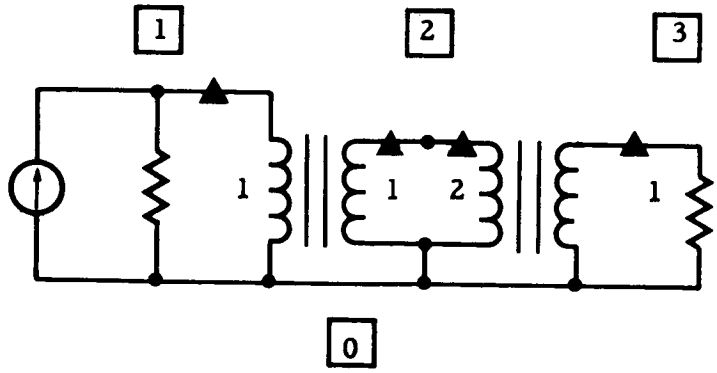


EXHIBIT 9 - EXAMPLE CIRCUIT FOR BASIS VOLTAGE ELIMINATION

[1] See Hildebrand, F.B., Introduction to Numerical Analysis, McGraw-Hill, 1956, p. 429.

Assume the connection list for Exhibit 9 is

$$\text{WLIST} = \lfloor \text{SI0001; SG0001; SN0001/1-1; SN0002/1-1; SN0002/2-2; SN0003/2-1; SG0003} \rfloor$$

The tree list is

$$\text{FLIST} = \lfloor \text{SG0001; SG0003; SN0002} \rfloor$$

The relationship between the node voltages and basis voltages is

$$VN = TC * V$$

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V31(1) \\ V31(2) \\ V51(1) \end{bmatrix}$$

There are two transformer equations.

$$TTR * VN = 0$$

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = 0$$

TQ is the product of TTR and TC .

$$TQ = TTR * TC = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2 \end{bmatrix}$$

We now have the equations

$$TQ * F = 0 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} V31(1) \\ V31(2) \\ V51(1) \end{bmatrix}$$

$$= \begin{bmatrix} V31(1) - V31(2) \\ V31(2) - 2\,V51(1) \end{bmatrix}$$

In Jordan's method, we make the coefficient of a variable in some equation +1 and then use that equation to eliminate the variable from all of the other equations. This is repeated for each variable we wish to eliminate. The TAG rule is that we start at the bottom of the V vector. In our example, V51(1) is the first candidate.

$$\begin{aligned} V31(1) - V31(2) &= 0 \\ -.5\,V31(2) + V51(1) &= 0 \end{aligned}$$

$$\begin{aligned} -V31(1) + V31(2) &= 0 \\ -.5\,V31(2) + V51(1) &= 0 \end{aligned}$$

$$\begin{aligned} -V31(1) + V31(2) &= 0 \\ -.5\,V31(1) + V51(1) &= 0 \end{aligned}$$

$$\begin{aligned} V31(2) &= +V31(1) \\ V51(1) &= +.5\,V31(1) \end{aligned}$$

We can now express the NRR basis voltages in terms of the N - 1 - NRR remaining basis voltages. In general we have

$$V = TL * FV$$

where TL has N - 1 rows and N - 1 - NRR columns and FV is the reduced basis vector. In our example, the equation is

$$\begin{bmatrix} V31(1) \\ V31(2) \\ V51(1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ .5 \end{bmatrix} V31(1)$$

Since the node voltages were expressed in terms of the basis voltages, we may now express the node voltages in terms of the reduced set of basis voltages.

$$VN = TC * V = TC * TL * FV$$

For the example, we have

$$VN = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = TC * TL * FV$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ .5 \end{bmatrix} V31(1)$$

$$= \begin{bmatrix} 1 \\ 1 \\ .5 \end{bmatrix} V31(1)$$

10. The Final Voltage Basis and Coordinate Transformation Matrix

a. No Transformers in the Network

In the case of a network with no transformers, no basis voltage elimination is required. The various lists, matrices, and constants are renamed in the following way.

TR — The final coordinate transformation matrix which is equal to TC

VC	The final voltage basis list which is equal to FLIST
FV	The final voltage basis vector which is identical to V
FV11, FV21, FV31, FV41, FV51	Subvectors of FV equal to V11, V21, V31, V41, and V51, respectively
LNV, LNC, LNG, LNL, LNX	The dimensions of the above subvectors which are equal to NV, NC, NG, NL, and NX, respectively
FS	The voltage integral vector which is equal to the integral of FV

$$FS = \int_0^t FV \, d\tau$$

FS11, FS21, FS31, FS41, FS51	The integrals of FV11, FV21, FV31, FV41, and FV51, respectively.

b. Transformers in the Network

When transformers are present, the node voltages may be expressed as linear combinations of less than N - 1 basis voltages. As shown in subsection 9, it is TC * TL that relates the node voltages to a reduced voltage basis vector FV . The following definitions are made for the network with transformers.

VC	A list of those descriptors in FLIST whose corresponding basis voltages have not been eliminated by the transformer constraints. VC is in the same sequence as FLIST
LNV, LNC, LNG, LNL, LNX	The number of voltage sources, capacitors, conductances, inductors, and transformer winding descriptors in FVC; also the dimensions of FV11, FV21, FV31, FV41, and FV51, respectively

FV	The voltage basis vector related to VC in the same way that V is related to FLIST (see subsection 7)
FV11, FV21, FV31, FV41, FV51	Subvectors of FV related to VC and the branch voltages in the same way that V11, V21, V31, V41, and V51 are related to FLIST and the branch voltages
TF	The final coordinate transformation matrix, which is TL multiplied by TC TF = TC * TL TF has N - 1 rows and N - NRR - 1 columns
N	The number of nodes in the network
NRR	The number of transformer windings minus the number of transformers
FS	The voltage integral vector equal to the integral of FV
FS11, FS21, FS31, FS41, FS51	The integrals of FV11, FV21, FV31, FV41, and FV51, respectively

11. The TAG Node Equations

In TAG, node equations are first written and then modified by the TF matrix described above.

Kirchoff's current law is written for each node in terms of node to ground (zero node) voltages. Branch equations for each element type are described in the TAG User's Manual.

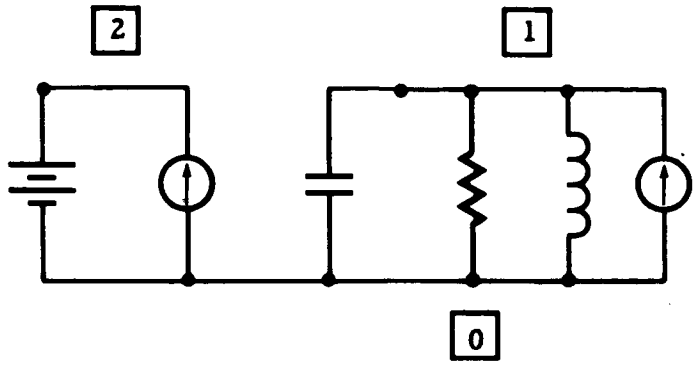


EXHIBIT 10 - EXAMPLE NETWORK DEMONSTRATING GENERATION OF NODE SYSTEM EQUATIONS

The node equations for the network shown in Exhibit 10 are

$$\text{WLIST} = \left[\text{SV0002; SC0001; SG0001; SL0001; SI0001}\right]$$

$$\text{FLIST} = \left[\text{SV0002; SC0001}\right]$$

$$\text{SC0001} * \frac{dV_1}{dt} + \text{SG0001} * V_1 + \text{SL0001} \int_0^t V_1 \, d\tau - \text{SI0001} = 0$$

$$I_{\text{SV0002}} = -\text{SI0002}$$

Note that no equation is written for the zero node and that the symbol L is TAG reciprocal inductance. Also note that ideal transformers are ignored in writing the node equations.

For an N-node network, we have the following matrix equation.

$$C_n \frac{d}{dt} VN + G_n VN + L_n \int_0^t VN \, d\tau - I_n - I_{en} = 0$$

where VN = the N - 1-element node voltage vector

C_n = the N - 1 by N - 1 node capacitance matrix

G_n = the N - 1 by N - 1 node conductance matrix
L_n = the N - 1 by N - 1 node reciprocal inductance matrix
I_n = the N - 1 by 1 node current source vector
I_{en} = the N - 1 by 1 current in voltage source vector

In order to solve the equations numerically, we must manipulate them into a standard form in which there are only first derivatives of variables on the left and functions of the same variables on the right.

$$\frac{dY}{dt} = F(Y)$$

$$\begin{bmatrix} \frac{dY_a}{dt} \\ \cdot \\ \cdot \\ \cdot \\ \frac{dY_k}{dt} \end{bmatrix} = \begin{bmatrix} f_1(Y_a, \cdots, Y_k) \\ \\ \\ \\ f_k(Y_a, \cdots, Y_k) \end{bmatrix}$$

If C_n is nonsingular and I_{en} is zero, then the equations may be manipulated easily into the required form.

$$\frac{dP}{dt} = VN$$

$$C_n \frac{d}{dt} VN + G_n VN + L_n \int_0^t \frac{dP}{d\tau} d\tau - I_n - 0 = 0$$

$$\frac{dP}{dt} = VN$$

$$\frac{d}{dt} VN = - C_n^{-1} (G_n VN + L_n P) + C_n^{-1} I_n$$

Unfortunately, C_n is often singular and I_{en} is not always zero. It is for these reasons that TAG must modify the node equations. A network for which C_n is singular and I_{en} is nonzero is shown in Exhibit 11.

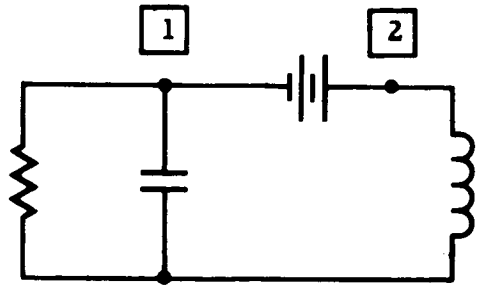


EXHIBIT 11 - EXAMPLE NETWORK DEMONSTRATING SINGULARITY OF NODE SYSTEM CAPACITANCE MATRIX

$$\text{WLIST} = \left[\text{SV0102; SC0001; SG0001; SL0002}\right]$$

$$\text{FLIST} = \left[\text{SV0102; SC0001}\right]$$

$$\begin{bmatrix} \text{SC0001} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{V}_1 \\ \dot{V}_2 \end{bmatrix} + \begin{bmatrix} \text{SG0001} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \text{SL0002} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \end{bmatrix}$$

$$+ \begin{bmatrix} I_{\text{SV0001}} \\ -I_{\text{SV0001}} \end{bmatrix} = 0$$

$$\begin{bmatrix} \dot{P}_1 \\ \dot{P}_2 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

The equations may be put into the correct form by making the substitution

$$V_2 = V_1 + \text{SV0102}$$

and then adding the first two equations.

$$\dot{P}_1 = V_1$$

$$\dot{P}_2 = V_1 + SV0102$$

$$SC0001 * \dot{V}_1 + SG0001 * V_1 + SL0002 * P_2 = 0$$

$$I_{SV0001} = SL0002 * P_2$$

$$\dot{V}_1 = -\frac{1}{SC0001}(SG0001 * V_1 + SL0002 * P_2)$$

$$I_{SV0001} = SL0002 * P_2$$

The above is a simple example of the transformation of the node equations carried out by TAG.

12. Transformation of the Node Equations

The TAG final equations are formed by making the following substitutions into the node equations

$$VN = TF * FV$$

$$\frac{d}{dt} VN = TF * \frac{d}{dt} FV$$

$$\int_0^t VN \, d\tau = TF * \int_0^t FV \, d\tau$$

$$FS = \int_0^t FV \, d\tau$$

$$FVD = \frac{d}{dt} FV$$

and then multiplying through by TF transposed.

$$(TF)^T C_n(TF) * FVD + (TF)^T G_n(TF) \overline{FV} + (TF)^T L_n(TF) FS$$

$$- (TF)^T I_n - (TF)^T I_{en} = 0$$

$$\frac{dFS}{dt} = FS$$

If the following substitutions are made

$$FC = (TF)^T C_n(TF)$$

$$FG = (TF)^T G_n(TF)$$

$$FL = (TF)^T L_n(TF)$$

$$FI = (TF)^T I_n$$

$$I_e = (TF)^T I_{en}$$

the equations become

$$FC * FVD + FG * FV + FL * FS - FI - I_e = 0$$

13. Form of Final Matrices

Lock[1] and others have shown that the matrices will always have properties to be described below.

a. Partition Dimensions

The partition dimensions are taken from the final basis list, VC.

LNV = the number of voltage source descriptors in VC
LNC = the number of capacitor descriptors in VC
LNG = the number of conductance descriptors in VC
LNL = the number of inductor descriptors in VC
LNX = the number of transformer winding descriptors in VC

[1] Lock, K., A Digital Computer Programmed Topological Method of Coordinate Selection for Numerical Computations in an Electrical Network, Ph.D., Thesis, California Institute of Technology, 1962.

b. Partitions of C

$$FC = \begin{array}{c} \begin{matrix} \text{LNV} & \text{LNC} & \text{LNG} & \text{LNL} \end{matrix} \\ \begin{bmatrix} FC11 & FC12 & 0 & 0 \\ FC21 & FC22 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} \text{LNV} \\ \text{LNC} \\ \text{LNG} \\ \text{LNL} \end{matrix}$$

The matrix FC has the following characteristics:

- Columns and rows with index greater than LNV + LNC are always zero.
- The FC22 submatrix, which is LNC by LNC, is always nonsingular when it is not null.

c. Partitions of FG

$$FG = \begin{array}{c} \begin{matrix} \text{LNV} & \text{LNC} & \text{LNG} & \text{LNL} \end{matrix} \\ \begin{bmatrix} FG11 & FG12 & FG13 & 0 \\ FG21 & FG22 & FG23 & 0 \\ FG31 & FG23 & FG33 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} \text{LNV} \\ \text{LNC} \\ \text{LNG} \\ \text{LNL} \\ \text{LNX} \end{matrix}$$

FG has the following characteristics:

- Columns and rows with index greater than LNV + LNC + LNG are always zero.
- FG33, which is LNG by LNG, is nonsingular if it is not null.

d. Partitions of FL

$$FL = \begin{array}{c} \begin{matrix} LNV & LNC & LNG & LNL & \end{matrix} \\ \begin{bmatrix} FL11 & FL12 & FL13 & FL14 & 0 \\ FL21 & FL22 & FL23 & FL24 & 0 \\ FL31 & FL32 & FL33 & FL34 & 0 \\ FL41 & FL42 & FL43 & FL44 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} LNV \\ LNC \\ LNG \\ LNL \\ LNX \end{array}$$

FL has the following characteristic:

- Matrix FL44 is nonsingular when it is not null.

e. Partitions of FV

$$FV = \begin{bmatrix} FV11 \\ FV21 \\ FV31 \\ FV41 \\ FV51 \end{bmatrix} \begin{array}{l} LNV \\ LNC \\ LNG \\ LNL \\ LNX \end{array}$$

$$FV11 = \begin{bmatrix} FV11(1) \\ \vdots \\ FV11(LNV) \end{bmatrix}$$

$$FV21 = \begin{bmatrix} FV21(1) \\ \vdots \\ FV21(LNC) \end{bmatrix}$$

$$FV31 \begin{bmatrix} FV31(1) \\ \vdots \\ FV31(LNG) \end{bmatrix}$$

$$FV41 = \begin{bmatrix} FV41(1) \\ \vdots \\ FV41(LNL) \end{bmatrix}$$

f. Partitions of FVD

$$FVD = \begin{bmatrix} FVD11 \\ FVD21 \\ FVD31 \\ FVD41 \\ FVD51 \end{bmatrix} \begin{matrix} LNV \\ LNC \\ LNG \\ LNL \\ LNX \end{matrix}$$

$$FVD11 = \begin{bmatrix} FVD11(1) \\ \vdots \\ FVD11(LNV) \end{bmatrix}$$

$$FVD21 = \begin{bmatrix} FVD21(1) \\ \vdots \\ FVD21(LNC) \end{bmatrix}$$

$$FVD31 \quad \begin{bmatrix} FVD31(1) \\ \vdots \\ FVD31(LNG) \end{bmatrix}$$

$$FVD41 \quad \begin{bmatrix} FVD41(1) \\ \vdots \\ FVD41(LNL) \end{bmatrix}$$

$$FVD51 = \begin{bmatrix} FVD51(1) \\ \vdots \\ FVD51(LNX) \end{bmatrix}$$

g. Partitions of FI

$$FI = \begin{bmatrix} FI11 \\ FI21 \\ FI31 \\ FI41 \\ FI51 \end{bmatrix} \begin{matrix} LNV \\ LNC \\ LNG \\ LNL \\ LNX \end{matrix}$$

$$FI11 = \begin{bmatrix} FI11(1) \\ \vdots \\ FI11(LNV) \end{bmatrix}$$

$$FI21 = \begin{bmatrix} FI21(1) \\ \vdots \\ FI21(LNC) \end{bmatrix}$$

$$FI31 = \begin{bmatrix} FI31(1) \\ \vdots \\ FI31(LNG) \end{bmatrix}$$

$$FI41 = \begin{bmatrix} FI41(1) \\ \vdots \\ FI41(LNL) \end{bmatrix}$$

$$FI51 = \begin{bmatrix} FI51(1) \\ \vdots \\ FI51(LNX) \end{bmatrix}$$

h. Partitions of FS

$$FS = \begin{bmatrix} FS11 \\ FS21 \\ FS31 \\ FS41 \\ FS51 \end{bmatrix} \begin{matrix} LNV \\ LNC \\ LNG \\ LNL \\ LNX \end{matrix}$$

$$FS11 = \begin{bmatrix} FS11(1) \\ \vdots \\ FS11(LNV) \end{bmatrix}$$

$$FS21 = \begin{bmatrix} FS21(1) \\ \vdots \\ FS21(LNC) \end{bmatrix}$$

$$FS31 = \begin{bmatrix} FS31(1) \\ \vdots \\ FS31(LNG) \end{bmatrix}$$

$$FS41 = \begin{bmatrix} FS41(1) \\ \vdots \\ FS41(LNL) \end{bmatrix}$$

$$FS51 = \begin{bmatrix} FS51(1) \\ \vdots \\ FS51(LNX) \end{bmatrix}$$

i. Partitions of I_e

$$I_e = \begin{bmatrix} FIE11 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} LNV \\ LNC \\ LNG \\ LNL \end{matrix}$$

$$FIE11 = \begin{bmatrix} FIE11(1) \\ \vdots \\ FIE11(LNV) \end{bmatrix}$$

14. Summary of Equation Formulation Procedure

To summarize, the TAG equations are written in the following way.

a. PLIST is formed by sorting the descriptors in the connection list into V, C, G, L, N, I order. Connection list order is maintained between descriptors of the same type.

b. Descriptors corresponding to a TAG proper tree are selected from PLIST and placed in FLIST. If the second (positive) node of an element descriptor is closest to the zero node, then the descriptor is underlined in FLIST.

c. FLIST is sorted into V, C, G, L, N, I order.

d. The coordinate transformation matrix, TC , is formed from FLIST.

e. If no ideal transformers are present, TC is renamed TF, the final coordinate transformation matrix. In addition, FLIST is renamed VC.

f. If ideal transformers are present, the following steps are performed.

(1) The ideal transformer constraint matrix, TTR, is formed.

(2) The matrix TQ is formed by multiplying TC by TTR.

(3) Dependent basis voltages are solved for by applying Jordan's method to TQ.

(4) From the solution in (3) above, a new matrix TL is formed which expresses the old basis voltages as linear combinations of a smaller set of voltages.

(5) The final coordinate transformation matrix is formed by multiplying TL by TC.

(6) A new list, VC, is formed which consists of only those entries in FLIST which were not associated with dependent basis voltages (step (3), above).

g. Ignoring ideal transformers and voltage sources, the node equations for the network are written. These equations are completely specified by forming only the C_n , G_n , L_n , and I_n matrices.

h. The final TAG equations are formed by forming $(TF)^T C_n TF$, $(TF)^T G_n TF$, $(TF)^T L_n TF$, and $(TF)^T I_n$, which are named FC , FG , FL , and FI , respectively.

i. The dependent variables for the equations are determined from VC. They are vectors with names FV11, FV21, FV31 , FV41 , and FV51 , corresponding to V, C, G, L, and N entries in the VC list. Whenever a descriptor is underlined in VC , the branch voltage and dependent variable for that branch have opposite signs.

15. Sample Network With No Transformers

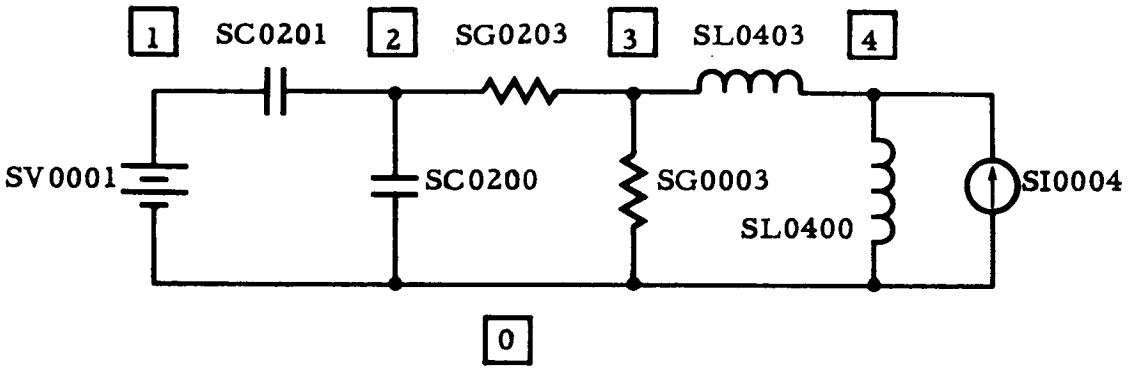


EXHIBIT 12 - SAMPLE NETWORK WITH NO TRANSFORMERS

The equations for the network of Exhibit 12 will be written using the previously outlined procedures.

Connection list: SL0403, SL0400, SG0203, SV0001, SC0201, SC0200, SG0003, SI0004*

WLIST = [SL0403; SL0400; SG0203; SV0001; SC0201; SC0200; SG0003; SI0004]

PLIST = [SV0001; SC0201; SC0200; SG0203; SG0003; SL0403; SL0400; SI0004]

FLIST = [SV0001; SC0201; SG0203; SL0403]

The TAG proper tree is shown in Exhibit 13. Basis references are circled; branch references are not. Note that the basis and branch references differ only where the corresponding descriptor is underlined in FLIST.

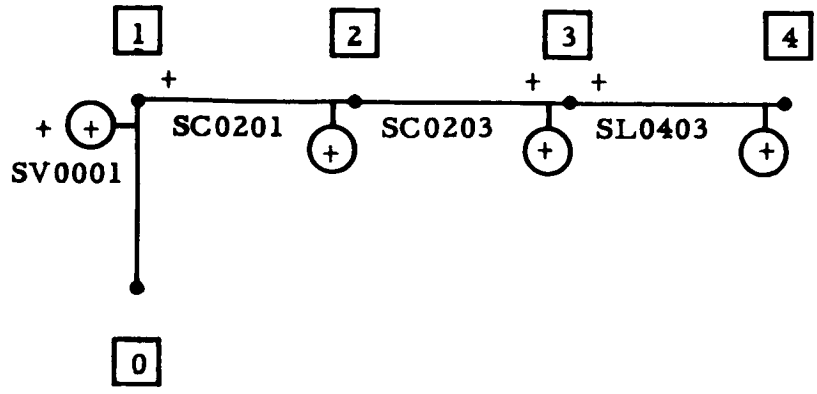


EXHIBIT 13 - TAG PROPER TREE

$$TC = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Since there are no ideal transformers in the network, we have

$$TF = TC$$

$$VC = FLIST$$

The coefficient matrices for the node equations are

$$C_n = \begin{bmatrix} SC0201 & -SC0201 & 0 & 0 \\ -SC0201 & SC0201 + SC0200 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G_n = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & SG0203 & -SG0203 & 0 \\ 0 & -SG0203 & SG0203 + SG0003 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$L_n = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & SL0403 & -SL0403 \\ 0 & 0 & -SL0403 & SL0403 + SL0400 \end{bmatrix}$$

$$I_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ SI0004 \end{bmatrix} \qquad I_{en} = \begin{bmatrix} -I_{SV0001} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Applying the TF matrix to the above coefficient matrices, we obtain

$$FC = \begin{bmatrix} SC0200 & SC0200 & 0 & 0 \\ SC0202 & SC0201 + SC0200 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$FG = \begin{bmatrix} SG0003 & SG0003 & SG0003 & 0 \\ SG0003 & SG0003 & SG0003 & 0 \\ SG0003 & SG0003 & SG0203 + SG0003 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$FL = \begin{bmatrix} SL0400 & SL0400 & SL0400 & SL0400 \\ SL0400 & SL0400 & SL0400 & SL0400 \\ SL0400 & SL0400 & SL0400 & SL0400 \\ SL0400 & SL0400 & SL0400 & SL0400 + SL0403 \end{bmatrix}$$

$$FI = \begin{bmatrix} SI0004 \\ SI0004 \\ SI0004 \\ SI0004 \end{bmatrix} \qquad I_e = \begin{bmatrix} -I_{SV0001} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The dependent variable names are

$$FV = \begin{bmatrix} FV11(1) \\ FV21(1) \\ FV31(1) \\ FV41(1) \end{bmatrix} = \begin{bmatrix} SV0001 \\ -SV0201 \\ SV0203 \\ -SV0403 \end{bmatrix}$$

$$FS = \begin{bmatrix} FS11(1) \\ FS21(1) \\ FS31(1) \\ FS41(1) \end{bmatrix} = \int_0^t \begin{bmatrix} FV11(1) \\ FV21(1) \\ FV31(1) \\ FV41(1) \end{bmatrix} d\tau$$

B. The Solution of TAG Equations

1. Introduction

The second pass of the TAG preprocessor takes the equations generated by the first pass and forms a solution program. The TAG User's Manual describes the solution program from the user's point of view. A more technical approach will be taken here; the steps

that are executed by the solution program in numerically solving the network problem will be discussed.

2. The TAG Matrix Equation

The TAG network equations, in their most general form, may be represented by two matrix equations.

$$C * FVD + G * FV + L * FS = FI + I_e + I_t$$

$$\frac{d}{dt} FS = FV$$

Exhibit 14 shows the equations in more detailed form. Performing the indicated multiplication, we obtain the following five sets of equations.

LNV Voltage Source Equations

```
  FC11 * FVD11 + FC12 * FVD21 + FG11 * FV11
+ FG12 * FV21  + FG13 * FV31  + FL11 * FS11
+ FL12 * FS21  + FL13 * FS31  + FL14 * FS41 = FI11 + FIE11
```

The above equations may be used to determine the currents in the voltage sources at any instant of time. TAG, however, does not evaluate them.

LNC Capacitor Equations

```
  FC21 * FVD11 + FC22 * FVD21 + FG21 * FV11
+ FG22 * FV21  + FG23 * FV31  + FL21 * FS11
+ FL22 * FS21  + FL23 * FS31  + FL24 * FS41 = FI21
```

LNG Conductance Equations

```
  FG31 * FV11 + FG32 * FV21 + FG33 * FV31 + FL31 * FS11
+ FL32 * FS21 + FL33 * FS31 + FL34 * FS41              = FI31
```

LNL Inductor Equations

```
FL41 * FS11 + FL42 * FS21 + FL43 * FS31 + FL44 * FS41 = FI41
```

LNX Transformer Equations

```
O = FI51 + FIT51
```

TAG does not evaluate the transformer equations.

$$\begin{bmatrix} FC11 & FC12 & 0 & 0 & 0 \\ FC21 & FC22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} FVD11 \\ FVD21 \\ FVD31 \\ FVD41 \\ FVD51 \end{bmatrix} + \begin{bmatrix} FG11 & FG12 & FG13 & 0 & 0 \\ FG21 & FG22 & FG23 & 0 & 0 \\ FG31 & FG32 & FG33 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} FV11 \\ FV21 \\ FV31 \\ FV41 \\ FV51 \end{bmatrix}$$

$$\begin{bmatrix} FL11 & FL12 & FL13 & FL14 & 0 \\ FL21 & FL22 & FL23 & FL24 & 0 \\ FL31 & FL32 & FL33 & FL34 & 0 \\ FL41 & FL42 & FL43 & FL44 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} FS11 \\ FS21 \\ FS31 \\ FS41 \\ FS51 \end{bmatrix} = \begin{bmatrix} FI11 \\ FI21 \\ FI31 \\ FI41 \\ FI51 \end{bmatrix} + \begin{bmatrix} FIE11 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ FIT51 \end{bmatrix}$$

$$\begin{bmatrix} FSD11 \\ FSD21 \\ FSD31 \\ FSD41 \\ FSD51 \end{bmatrix} = \begin{bmatrix} FV11 \\ FV21 \\ FV31 \\ FV41 \\ FV51 \end{bmatrix}$$

$$FV = \int_0^t FVD \, d\tau$$

$$FS = \int_0^t FV \, d\tau$$

EXHIBIT 14 - TAG NETWORK EQUATIONS IN PARTITIONED FORM

N - NRR - 1 Voltage Integral Equations

FSD11 = FV11

FSD21 = FV21

FSD31 = FV31

FSD41 = FV41

FSD51 = FV51

The last two equations, for FS41 and FS51, are not used by TAG. From this point on, we will concern ourselves only with the LNC capacitor equations, LNG conductance equations, LNL inductance equations, and a subset of the voltage integral equations. It will be shown that these equations, when solved, specify every voltage in the network at every instant of time.

The equations for Exhibit 12 are

LNC = 1

```
  SC0200 * FVD11(1) + (SC0200 + SC0201) * FVD21
+ SG0003 * FV11(1)  + SG0003  * FV21(1)
+ SG0003 * FV31(1)  + SL0400  * FS11(1)
+ SL0400 * FS21(1)  + SL0400  * FS31(1)
+ SL0400 * FS41(1)                              = SI0004
```

LNG = 1

```
  SG0003 * FV11(1) + SG0003  * FV21(1)
+(SG0203 + SG0003) * FV31(1) + SL0400 * FS11(1)
+ SL0400 * FS21(1) + SL0400  * FS31(1)
+ SL0400 * FS41(1)                              = SI0004
```

LNL = 1

```
  SL0400 * FS11(1) + SL0400  * FS21(1)
+ SL0400 * FS31(1) + (SL0400 + SL0403) * FS41(1)
                                                = SI0004
```

FSD11(1) = FV11(1)

FSD21(1) = FV21(1)

FSD31(1) = FV31(1)

3. Nonconstant Parameters and Current Generators

The TAG User's Manual describes, in detail, the rules for specifying nonconstant parameters. It is sufficient to note here that the following parameters may depend upon voltage, voltage integral, and time:

Capacitors
Conductances
Inductors
Current Generators

Whenever a voltage appears in the expression for a nonconstant parameter, we use a symbol which is similar to the voltage generator descriptor. Thus, SVXXYY represents the voltage between nodes XX and YY , where node YY is the positive node.

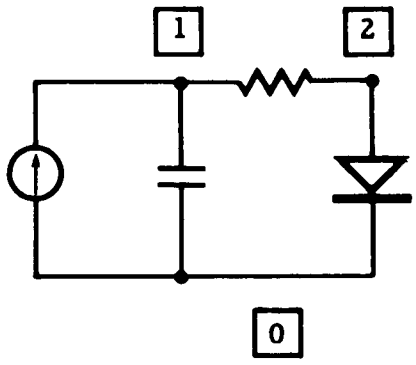


EXHIBIT 15 - EXAMPLE NETWORK DEMONSTRATING SINGULARITY OF NODE SYSTEM CAPACITANCE MATRIX

In Exhibit 15, the voltage between nodes 2 and 0 is SV0002. The diode between nodes 2 and 0 may be represented by a current generator having the following branch equation.

$$SI0200 = XIS(e^{SV0002/VO} - 1)$$

The dependence of the current generator does not change the equation writing procedure. When the equations are completed, we simply substitute the expression for the current generator symbol, wherever it appears. In addition, SV0002 must be expressed as a sum of basis voltages.

$$SV0002 = FV21(1) + FV31(1)$$

This is always possible because the basis voltages specify every other voltage in the network.

4. Form of the Final Differential Equations

As stated previously, we wish to obtain a system of equations having the form

$$\frac{dY}{dt} = f(Y, t)$$

where Y and f are vectors and t is time. In the case of TAG, the final form is

$$\frac{d}{dt}\begin{bmatrix} FV21 \\ FS11 \\ FS21 \\ FS31 \end{bmatrix} = f(FV21, FS11, FS21, FS31, FT)$$

where FT is time and the other symbols have been defined in earlier sections.

Since TAG must generate a computer program for the solution of the equations, it recognizes a large number of different forms of the equations. For the discussion here, we will recognize only two basic equation forms.

Form 1: The equations are in the first TAG form if

- o No element in the inductance or current generator matrices of the LNL inductance equations is a function of FV41, FS41, or FV31
- o No element in the conductance, inductance, or current generator matrices of the LNG conductance equations is a function of FV31

Form 2: The equations are in the second TAG form if they cannot be classified as being in the first form.

When the equations are in the first form, we can solve for FS41 and FV31 at every instant of time by inverting FG33 and FL44 and solving two matrix equations.

$$FS41 = (FL44)^{-1} * (FI41 - FL41 * FS11 - FL42 * FS21 - FL43 * FS31)$$

$$FV31 = (FG33)^{-1} * (FI31 - FG31 * FV11 - FG32 * FV31 - FL31 * FS11 - FL32 * FS21 - FL33 * FS31 - FL34 * FS41)$$

In order to allow parameters to be dependent upon FV41, we numerically differentiate FS41.

$$FV41 \cong \frac{FS41_t - FS41_{t - \Delta t}}{\Delta t}$$

Thus, functions of FV41 become functions of FS41. It is apparent that the above equations are explicit in FS41 and FV31 if the right-hand side of the FS41 equation is not a function of FV41, FS41, or FV31. In addition, the FV31 equation cannot have a function of FV31 on the right-hand side. There are a number of special cases in which FV31 and FS41 may be solved for directly; they will not be considered here.

When the equations are in the second form, we assume that the right-hand sides of the FS41 and FV31 equations are functions of FS41 and FV31. We may rewrite the equations as

$$FS41 - (FL44)^{-1} * (FI41 - FL41 * FS11 - FL42 * FS21 - FL43 * FS31) = 0$$

$$FV31 - (FG33)^{-1} * (FI31 - FG31 * FV11 - FG32 * FV31 - FL31 * FS11 - FL32 * FS21 - FL33 * FS31 - FL34 * FS21) = 0$$

In shorthand form, we have

$$f_1 (FS41, FV31) = 0 \qquad \text{(LNL equations)}$$

$$f_2 (FS41, FV31) = 0 \qquad \text{(LNG equations)}$$

For the solution of these equations, all other variables and time are assumed constant. The FV41 variable is assumed to have been transformed into a function of FS41 by numerical differentiation.

Although equations in the second TAG form may be linear, it will be assumed that an iterative technique, such as the Newton-Raphson method, is used to solve them.

Regardless of the method used to solve for FS41 and FV31, the LNC capacitor equations can always be solved for FVD21 by inverting FC22. Assuming FS41 and FV31 are known, we have the final differential equations.

$$\begin{aligned}
FVD21 = (FC22)^{-1} * (&FI21 - FC21 * FVD11 - FG21 * FV11 \\
&- FG22 * FV21 - FG23 * FV31 - FL21 * FS11 \\
&- FL22 * FS11 - FL23 * FS31 - FL24 * FS41)
\end{aligned}$$

$$FSD11 = FV11$$

$$FSD21 = FV21$$

$$FSD31 = FV31$$

In the above equations, it is assumed that FV11 is a known function of time and that the derivative of FV11 is determined by numerical differentiation.

$$FVD11 \cong \frac{FV11_t - FV11_{t - \Delta t}}{\Delta t}$$

5. Solution of the Equations

The equation formulation procedure results in LNL + LNG linear or nonlinear equations and LNC + LNV + LNG + LNC first order differential equations.

When the LNL + LNG equations are classified TAG type 1, FV31 and FS41 are found by matrix inversion, addition, subtraction, and multiplication. When they are type 2, we find FV31 and FS41 by the Newton-Raphson method. Writing the equations in shorthand form, we have

$$\begin{bmatrix} f_1 (FV31,\ FS41) \\ f_2 (FV31,\ FS41) \end{bmatrix} = \begin{bmatrix} R \end{bmatrix} = \begin{bmatrix} r_i \end{bmatrix}$$

where R is the vector of residuals whose values are all zero only at the exact solution of the set of equations.

The Newton-Raphson method defines a sequence of values of FV31 and FS41 which, hopefully, converges to the solution of the equations.

$$\begin{bmatrix} FV31_k \\ FS41_k \end{bmatrix} = \begin{bmatrix} FV31_{k-1} \\ FS41_{k-1} \end{bmatrix} - P_{k-1}^{-1} R_{k-1}$$

The subscript k represents evaluation for the kth step and k - 1 represents evaluation for the (k - 1)th step. The matrix P is the partial derivative matrix defined as follows.

$$P = \left[p_{ij} \right] \qquad 1 \le i \le LNL + LNG, \; 1 \le j \le LNL + LNG$$

$$p_{ij} = \frac{\partial r_i}{\partial v_j}$$

An obvious necessary condition for convergence of the iteration is that P^{-1} exist at the solution.

In TAG, the partial derivative matrix is computed by numerical differentiation.

$$p_{ij} \cong \frac{r_i (v_j + \Delta v_j) - r_i(v_j)}{\Delta v_j}$$

The Newton-Raphson method must be given initial guesses for FV31 and FS41 which it will refine to the solution. The closer the initial guesses are to the solution, the more rapidly the iteration will converge, in most instances.

The differential equations are in a form that allows the computation of the derivatives of FV21, FS11, FS21, and FS31 from their previous (or initial) values and the computed values of FS41 and FV31. The numerical integration program may then extrapolate all voltages, whose derivatives are known, to their approximate values at the next instant of time.

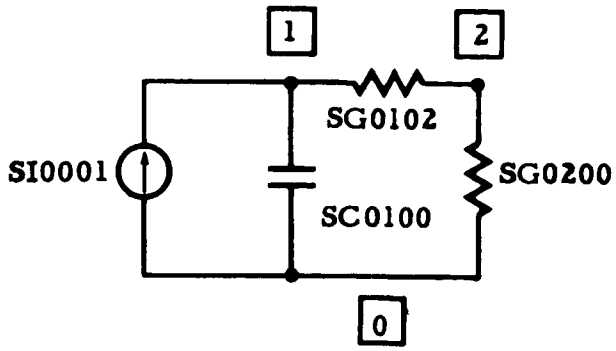


EXHIBIT 16 - EXAMPLE NETWORK USED IN DEMONSTRATION OF NUMERICAL INTEGRATION TECHNIQUES

Exhibit 16 will be used to illustrate the numerical solution of a network problem. Instead of the sophisticated TAG integration technique, we will use a very simple method which adequately demonstrates the procedure. The TAG equations are assumed to be

$$- FV21(1) * SG0102 + FV31(1)\,(SG0102 + SG0200) = 0$$

$$SC0100 * FVD21(1) + FV21(1) * SG0102 - FV31(1) * SG0102 = SI0001$$

Solving for FV31(1) and then FVD21(1), we obtain

$$FV31(1) = \frac{1}{SG0102 + SG0200}(FV21(1) * SG0102)$$

$$FVD21(1) = \frac{1}{SC0100}(SI0001 - FV21(1) * SG0102 + FV31(1) * SG0102)$$

For the solution, let

$$SG0102 = SG0200 = SC0100 = SI0001 = 1\ .$$

The voltages will be evaluated at FT = 0, .5, 1, 1.5, and 2 seconds. The initial voltage on the capacitor is -1 volt.

FT = 0

FV21(1) = -1 volt

FV31(1) = 1/2 (-1) = -(1/2) volts

FVD21(1) = 1 - (-1) + (-(1/2)) = 1.5 volts/sec

FT = .5

FV21(1) ≅ FVD21(1) * .5 + (-1) = .75 - 1 = - .25 volts

FV31(1) = 1/2 (-.25) = - .125 volts

FVD21(1) = 1 + .25 - .125 = 1.125 volts/sec

FT = 1.0

FV21(1) ≅ -.25 + 1.125 * .5 = .3125 volts

FV31(1) = 1/2 (.3125) = .1563 volts

FVD21(1) = 1 - .3125 + .1563 = .8438 volts/sec

FT = 1.5

FV21(1) ≅ .3125 + .5 * .8438 = .7344 volt

FV31(1) = 1/2 (.7344) = .3672 volts

FVD21(1) = 1 - .7344 + .3672 = .6328 volts/sec

FT = 2.0

FV21 ≅ .7344 + .5 * .6328 = 1.0508 volts

If the equation for FV31(1) had been nonlinear, then the Newton-Raphson iteration method would have been used to solve for FV31(1) at each instant of time.

6. Summary of Equation Solution Technique

The TAG solution technique may be summarized as follows.

a. Substitute sums of basis voltages for all voltage symbols SVXXYY which appear in nonconstant parameter equations.

b. Substitute the expression for the parameter into the TAG equations.

c. Classify the FV31 and FS41 equations as type 1 or type 2.

- o If they are type 1, solve explicitly for FV31 and FS41 by matrix manipulation.
- o If they are type 2, put the equations in the implicit form for Newton-Raphson iteration

$$f_1(FS41,\ FV31) = \left[R_1\right]$$
$$f_2(FS41,\ FV31) = \left[R_2\right]$$

d. Solve the LNC capacitor equations for the derivative of FV21 (FVD21).

e. Write down, directly, the equations for FS11, FS21, and FS31, if they appear in the above equations.

f. Specify one initial condition for each differential equation.

g. Obtain the system solution at each instant of time by numerical methods.

III. TAG LIST PROCESSING

A. Definition of TAG List Properties

1. Introduction

The basic data structuring of TAG is in the format of lists and list structures. A list is any sequence of elements linked together in a fixed order. A list structure is a list whose elements may themselves be lists. An array-list is a one-dimensional FORTRAN array of which each element may be a list or a list structure. Input and output of card images, construction and storing of special symbolic labels, and maintenance of parameter arrays are accomplished by applying list-manipulation techniques to these lists.

This section will contain a description of the structuring technique used and will define some terms useful in referring to peculiar structures and their properties. Following sections will examine the basic FORTRAN functions operating upon the lists and some of the special-format lists that are important in TAG.

2. Definitions

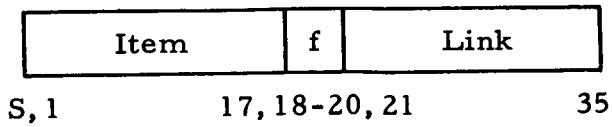
The following definitions are, for the most part, common in list-processing literature. The format of TAG lists is peculiar to TAG, however, and therefore a hybrid set of terms has been compiled.

AVS is available space reserved in core as a list of elements linked to each other by pointers. This simple list initially occupies sequentially descending positions in core. AVS is maintained as a "free storage list" whose elements are acquired by the program as they are needed. When elements are no longer needed for computation, they are returned to AVS.

A head cell, or head of a list, is a cell in core that has a FORTRAN name or symbol associated with it and whose decrement contains a pointer to the first element of a list.

A pointer is the 15-bit address of the memory cell or element to which it points. A pointer in the head cell of a list (as used in TAG) gives access to the first element of the list.

An element is one "piece" of information in a list. In TAG, it occupies one register (cell; word) of core storage and has the following format:



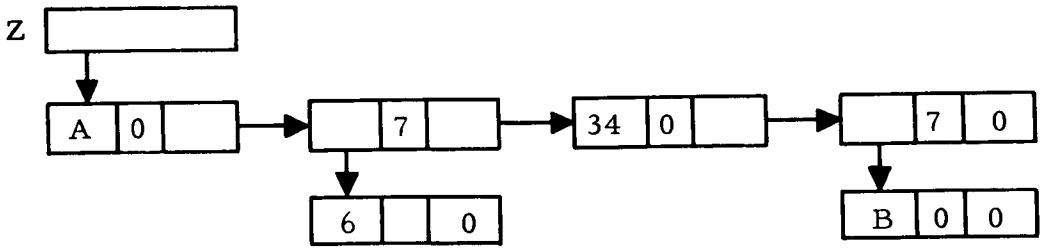
A link is one of a class of pointers. It occupies bits 21-35 of an element, and, in a list, is a pointer to the location in core of the next element of the list.

An item is bits S, 1-17 of an element, and constitutes the basic nonstructural data of an element. An item may be either an atom or a pointer (nonatomic item) to another list.

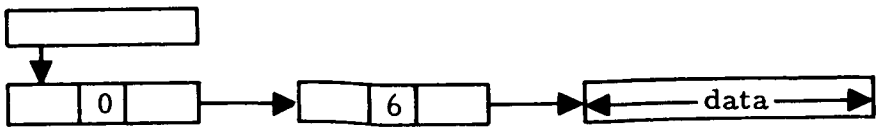
An atom is an item that may be any useful piece of nonstructural information (data). Depending on the context and the purpose of the list, an atom may be an integer, the BCD code for a Hollenith character, etc.

f is a flag in bits 18-20 of an element that gives supplemental information about the uses of the item and link.

Type 7 element: A flag of 7 in an element indicates that its item is nonatomic and is a pointer to another list.

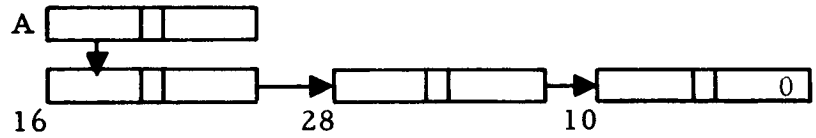


Type 6 element: A flag of 6 in an element indicates that the link points to a full word of data. This word does not have the item : flag : link structure of normal list elements, but contains whole word data. Note that any such word must be the last word in a list. (For example, this is useful in storing floating-point numbers in lists.)



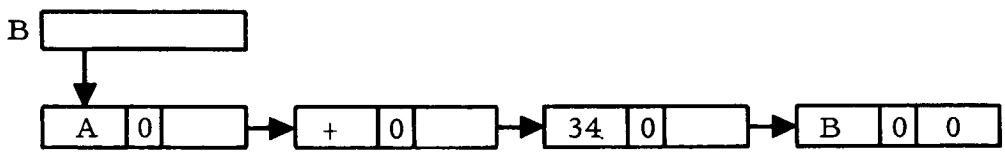
Type 5 element: A flag of 5 in an element indicates that the item portion of the word contains one decimal character of an eight-element BCD string (used when reformatting floating-point numbers for output in the second pass of the Preprocessor).

A list is any sequence of elements, linked in order.



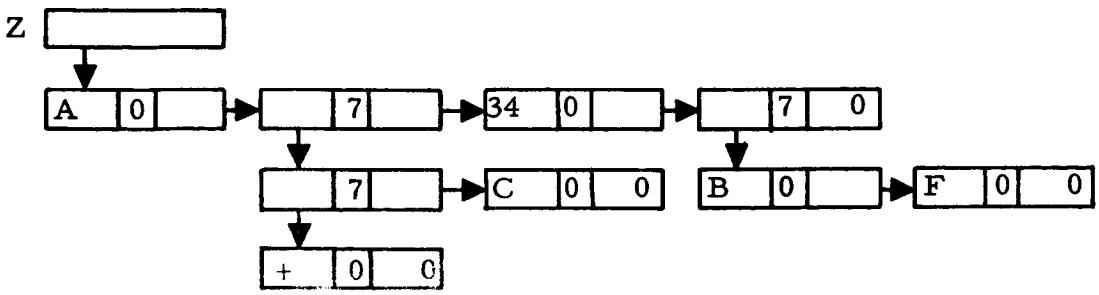
A is head of a list whose elements are located in cells 16, 28, and 10. These elements may be referred to as "list A," using the name associated with the head cell which points to the list. The above list contains several elements. Each element is linked to its successor by a pointer in the link portion of the word. The list is terminated with an element whose link portion contains zero. This indicates that there are no more elements to follow. The elements of a list need not be sequential cells in core. They are associated through their links. If the head of a list contains zero, the list has no elements at all. Such a list is said to be a null list.

A string, or simple list, is a list containing only atoms as items.



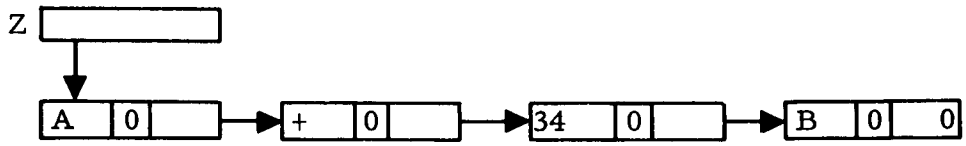
In the example above, A + 34 B are atoms in list B.

A list structure is a list that may have items which are not atoms:



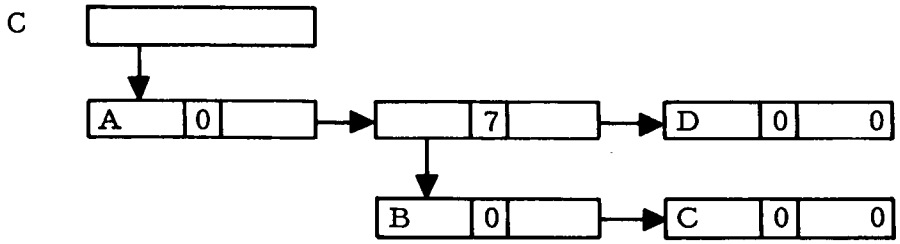
In some of the following descriptions, lists will be represented in a nongraphic manner according to the following format:

A simple list will be written as the sequence of atoms, separated by semicolons, and enclosed in brackets.



List Z in the example above may be represented as [A ; + ; 34 ; B]

If the list contains a nonatomic item, this item will be written according to the regular pattern for lists.



List C above is written as [A [B ; C] D]

Note that the "outer" list contains three items:

A

[B ; C]

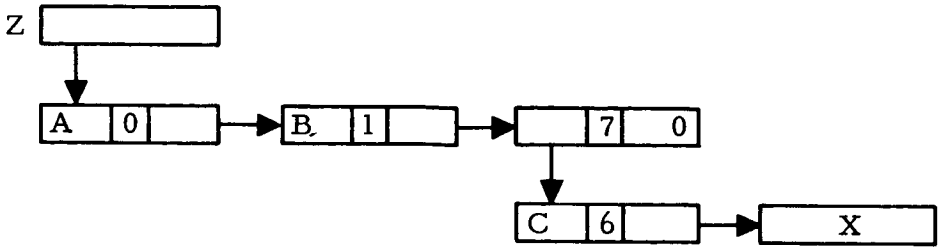
D

The second item is itself a list of two atomic items:

B

C

If the list contains peculiar flags, these may be indicated by placing them with the item of the same element, separated by a colon.



List Z above may be represented as $\left[A\ ;\ B : 1\ \left[C : 6\ ;\ X\right]\right]$

B. Basic TAG List Operations

Five subroutines that perform basic functions on the elements of AVS are ERASE, ERASEA, ADDLOC, BACK and NEWLOC.

ADDLOC determines the number of cells in core allocated for lists and connects all the allocated elements into a simple list called SPACE. Each element is linked to the next element by a pointer stored in the address portion of the word.

NEWLOC acquires a new element from AVS.

BACK returns an unneeded element to AVS.

ERASE restores all elements associated with a given list back to AVS.

ERASEA restores an array list back to AVS.

Seven subroutines that operate on the lists themselves are LINK, POPUP, DOWN, UPDWN, DOWNS, FROM and INTO.

LINK gets the link of the first element in a list (location of the 2nd element.)

FROM extracts the item and flag, if any, from the first element of a list.

INTO replaces the item and flag, if one is given, of the first element of a list. The address or link portion of the element is unchanged.

POPUP performs the basic "pop up" list function: the first element of a list is removed and its contents are saved. This element is then restored to AVS.

DOWN acquires a new element from AVS and pushes it down into a list with a specified item, flag, and link.

UPDWN pops up all elements of a given list and pushes them down (in reverse order) into a new list. The elements of the list that is "popped up" are returned to AVS.

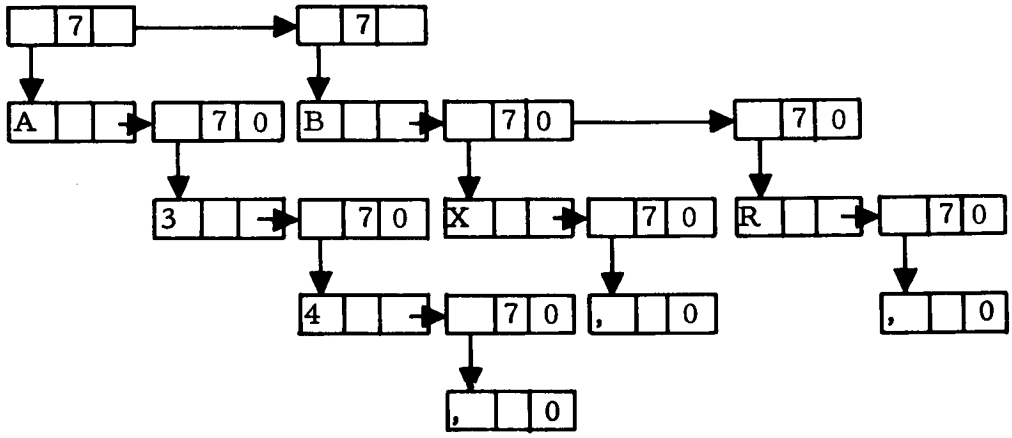
DOWNS performs the same function as UPDWN except that DOWNS does not destroy the list that was popped up.

C. Examples of List Structures Found in TAG

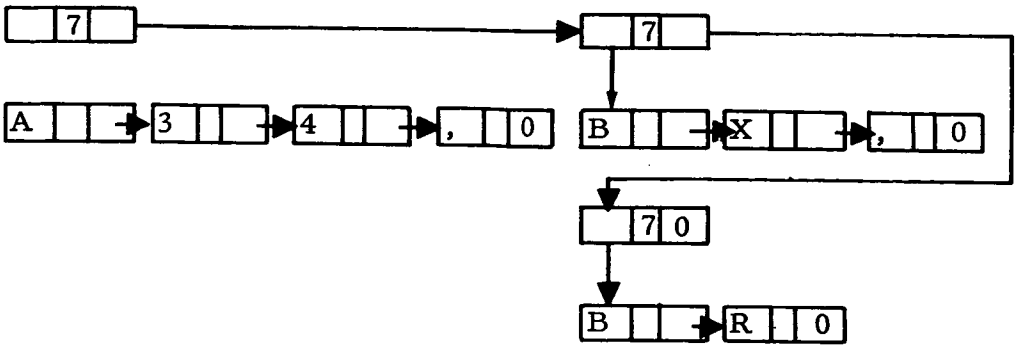
Type A: Simple string of characters.

Type B: Symbol string; like type A, but symbols represent variable names and are separated by commas (like output from RECOVER).

Type C: Symbol list structure (like output from ELIM).

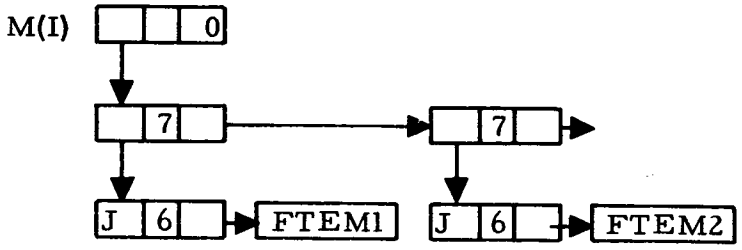


Type D: Two-dimensional list structure of simple sublists (used in PRPTG and PRPTR in first pass).

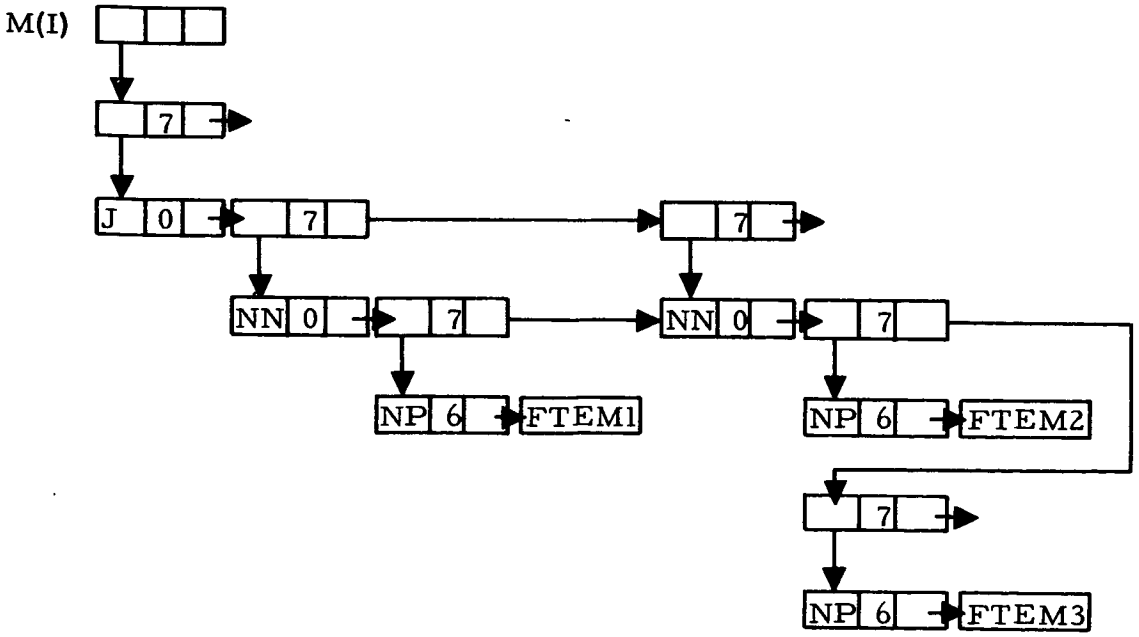


Type E: Array list (as used in LOCATA, ERASEA, SNATCH, STASH).

1. Two-dimensional array list, I, J are two dimensions of the array, M, of coefficients FTEM(i). I = 1, 2, ... 100.



2. Four-dimensional array list. I, J, NN, NP are four dimensions of the array, M, of coefficients FTEM(i). I = 1, 2, ... 100.



IV. TAG PREPROCESSOR MAIN PROGRAM

A. Overall Description of the TAG Preprocessor

1. Function

The TAG Preprocessor generates a FORTRAN source program specifically tailored to simulate the transient or DC steady-state behavior of a certain class of electronic circuits which can be described to it by means of a special input language. The input language is a combination of a special TAG vocabulary for describing the topology, variables, and parameters of circuits and a subset of FORTRAN code for describing special functions needed for nonstandard component modeling, information output, and program control. Using this language, the user creates a Circuit Description Deck, which specifies the particular circuit and particular analysis to be run. From the Circuit Description Deck the Preprocessor creates the simulation equations and imbeds them in a generated simulation program.

The TAG Preprocessor is composed of two main programs, which are themselves written in FORTRAN. The first pass program examines only the topological properties of the network, and, by treating all branch elements as linear, passive, and bilateral, creates a set of algebraic and/or first-order differential equations to simulate the network. The second pass program adds all nonlinear side constraints to these equations and generates the FORTRAN code for a program that will solve the simulation equations and output the desired variables.

2. Organization

The Preprocessor, which is the heart of the TAG circuit analysis concept, is a system of programs consisting of the 2 main routines and over 100 subroutines. The major manipulative technique employed in the Preprocessor is list processing. Even in the first pass, where many of the operations are arithmetic in nature, list processing is used predominantly. The second pass operations, which are, for the most part, symbol manipulative in nature, fit very well into the list

processing structure. Most of the subroutines of the Preprocessor are written in FORTRAN; however, a significant number of the lower level routines have had to be mechanized in FAP due to the limitations of FORTRAN.

The two passes of the Preprocessor are run as separate programs chained together through the FORTRAN Monitor System. Both programs are stored on a special tape in precompiled binary form. The first pass is called in by the FORTRAN Monitor System as commanded by a * LOAD 12 card provided by the user just ahead of the first card of the TAG Description Deck. At the end of the first pass, the second pass is called in through the FMS chaining feature. The equation matrices, variable vectors, and parameter symbol lists that are generated during the first pass are transferred to the second pass via three scratch tapes. Certain matrix dimensions and flags calculated in the first pass are passed along in common. At the end of the second pass, the final simulation program is written onto the FORTRAN print and punch tapes, and control is returned to the FMS in such a way as to provide an immediate compilation of the generated program. From that point on, control is under the FORTRAN Monitor system, and the program, if it compiles properly, may be executed at will.

B. First Pass Writeup and Flow Chart

1. Program Description

2. Identification

a. Routine Label

TAG

b. Name

Preprocessor, Main Program, Pass 1.

3. Function

The first pass of the Preprocessor reads in the connection list down to the final asterisk, selects a proper TAG tree from this list, and generates the standard form, linear differential and algebraic, simulation equations. If the network contains ideal transformers, the transformer voltage constraint equations are used to reduce the number of simulation equations. The equations, the first pass intput symbol table, and the final reduced tree voltage vector are written onto scratch tapes to be used by the second pass. The constants KIND, NM, NMR, and arrays NPT and SPACE are established in Pass 1 and passed on to Pass 2 in COMMON.

4. Programming System

FORTRAN II

5. Usage

a. Calling Sequence

The first pass routine of the TAG Preprocessor is loaded from tape in precompiled form by the FORTRAN Monitor. The FORTRAN Monitor control card, * LOAD 12 , is placed just ahead of the TAG Circuit Description Deck to initiate the load operation.

b. Entry Conditions

The TAG Circuit Description Deck has been written on a standard FORTRAN input tape.

The FORTRAN Monitor reads the Preprocessor Main Program for Pass 1 from tape into the computer memory and turns control of the computer over to it.

c. Exit Conditions

Scratch tape, NTAPE1, contains the final cut-set equation matrices for the capacitive, resistive, inductive and current source topologies stored in that order in a list-type format. Scratch tape, NTAPE2, contains the partially transformed node basis equation matrices for the capacitive, resistive, inductive, and current source topologies stored in that order in a list-type format. Scratch tape, NTAPE3, contains, in order, the final tree voltage basis to node voltage basis transformation matrix, TF; the input symbol table for Pass 1, which contains the names of each branch element (except transformer elements) plus the initial conditions voltages for all capacitors; and the final tree voltage vector corresponding to VC. In COMMON is stored:

KIND = 1 (for no inductors)
KIND = 2 (for inductors)
NM = Maximum number of nodes less 1
NMR = Number of variables in vector VC
NPT(NE) = Number of elements of type NE in VC
SPACE = Total available memory cells for lists

d. Error Exits

If a member of the final tree voltage vector does not match at least one branch in the connection list, WLIST, a DUMP is called and control is returned to the FORTRAN Monitor system.

6. Definition of Identifiers

SPACE is the total memory space allocated for list structures.

WLIST is a type D list representation of the TAG connection list. Each branch descriptor is represented by a sublist of 3 or 5 elements. Descriptor ordering is unchanged. All items of the branch descriptors are represented by binary integers. The element type characters are replaced by the following integers: V = 1; C = 2; G = 3; L = 4; N = 5; I = 6. Node numbers, transformer turns, and transformer numbers are converted directly to binary. The members of each descriptor

sublist are in general denoted by NE, NN, NP, NTRN, NNTR, and are placed in the order shown.

NE is the integer that represents the element type classification for a particular branch, and is assigned according to the rules given under WLIST above.

NN is the number of the first or negative node of a descriptor.

NP is the number of the second or positive node of a descriptor.

NTRN is the transformer core number of a descriptor.

NNTR is the number of turns on a transformer winding of a descriptor.

PLIST is a modified copy of WLIST used as input to the TREE subroutine. In PLIST, the branch descriptor sublists are reordered by element type into a V, C, G, L, N, I sequence. Input ordering within the element types is maintained.

XLIST is the tree branch list generated by the TREE subroutine. This is a two-dimensional type D list containing the descriptors of the branches whose node pair voltages will be used to form a voltage basis for the network. The order of PLIST is not maintained in XLIST.

KIND is an inductor flag which equals 1 if there are no inductors in the circuit and 2 if there are inductors in the circuit.

FLIST is a copy of XLIST reordered into a V, C, G, L, N, I sequence.

NM is the maximum node number of the network which also equals the total number of nodes less one, or the total number of tree branches or network equations with all transformer windings treated as open circuits.

NPT is an array in common which stores the number of each type of element that makes up the TAG tree by the following rule: NPT(NE) = the number of type NE elements in the tree.

TC is a two-dimensional type E list representation of the voltage coordinate transformation matrix. TC has NM rows and NM columns. Entries in TC are either +1 or 0. Each row in TC represents a path, in the tree, from some node to the zero node. Rows and row numbers correspond directly to nodes and node numbers.

TTR is a two-dimensional type E list representation of the transformer voltage constraint matrix. TTR has NM columns and NRR rows. Each row of TTR represents one transformer voltage constraint equation expressed in terms of node voltages. The columns of TTR are in one-to-one correspondence with the nodes.

NRR is the total number of transformer constraint equations which is equal to the total number of transformer windings less the total number of transformers.

TQ is always equivalent to TTR * TC , by which it is formed. TQ is an NRR x NM matrix represented as a two-dimensional type E list. Each row of TQ represents a transformer constraint equation expressed in terms of tree voltage variables. The columns of TQ are in one-to-one correspondence with the tree branch node pair voltages.

TL is a two-dimensional type E list representation which expresses all of the unconstrained tree voltage variables in terms of a smaller independent set of tree voltage variables. TL is NM x NMR.

NMR is the number of independent tree voltage variables under the transformer constraint equations. NMR = NM-NRR.

XS is a four-dimensional type E list representation of either the node basis or tree basis current equations.

TF is the final coordinate transformation matrix having NM rows and NMR columns. TF expresses all of the node voltages as linear combinations of the reduced set of tree voltages. TF = TC * TL.

TFT is the transpose of TF.

XF is a four-dimensional type E list representation of a matrix which temporarily stores the partial transformations of the XS matrices. XF = XS * TF.

XZ is a four-dimensional list representation of a matrix which expresses the final tree branch voltage vector. XZ has +1 or -1 entries only.

INSTP is an input symbol table for the Pass 1 circuit parameters and is represented as a four-dimensional type E list having +1 entries only. The table contains all the descriptors in WLIST in addition to a voltage source descriptor to initialize each capacitor voltage.

7. Method

The first pass of the Preprocessor operates on the connection list only, and produces the linear differential and algebraic simulation equations in matrix format. These equations are passed from the first to the second pass stored on tape in a special list format. The following series of steps describes the first pass operations.

a. Initialize DIMENSION, and COMMON required variables.

b. Set up scratch tapes, NTAPE1, NTAPE2, and NTAPE3 and rewind each.

c. Set up the storage area, SPACE, to be available for list structures (subroutine ADDLOC).

d. Write on the output tape, "CIRCUIT DESCRIPTION."

e. Set the inductor element flag KIND to 1.

f. Read in connection list, remove punctuation and excess characters, convert BCD to binary, and store a descriptor at a time in the two-dimensional type D list, WLIST.
Set NM equal to the maximum node number.
Set NT to equal the total number of branch descriptors.
Write on output tape a copy of the original connection list (subroutine GOBLE).

g. Write on output tape, "NUMBER OF NODES IS < NM >"

h. Erase PLIST and FLIST.

i. Reorder WLIST in INLGCV sequence and place the result in FLIST. Maintain WLIST intact. Set KIND to 2 if inductive descriptors appear in WLIST.
Reverse order of FLIST and place result in PLIST.
PLIST is then in VCGLNI sequence, while, within the same element type, the order of WLIST is maintained.
Erase FLIST.

j. Choose a proper TAG tree from PLIST and place the result in XLIST. Erase PLIST (subroutine TREE).

k. Reorder the elements of XLIST into VCGLNI sequence and place the resultant list in FLIST. Count the number of

each element type in FLIST and store in NPT such that NPT(NE) equals the number of type NE elements in FLIST. Erase XLIST.

l. Form TC, the tree to node system change of basis matrix, from FLIST such that VN = TC * VT. VN is the node voltage vector and VT is the tree voltage vector (subroutine COTRN).

m. Form TTR, the ideal transformer node voltage constraint matrix, from WLIST such that TTR * VN = 0. Set NRR to equal number of transformer constraint equations or the number of rows in TTR (subroutine XFORM).

n. If NRR equals zero, there are no transformers in the circuit and the process continues at step o. If NRR is greater than zero, there are transformers in the circuit and the process must jump to step r.

o. Change the name of FLIST to VC, which is the name of the final tree voltage vector.

p. Change the name of TC to TF, which is the name of the final tree to node system change of basis matrix.

q. Set NMR, the final number of simulation equations, to equal NM, the total number of nodes less 1. Proceed to step w.

r. Form TQ, the transformer tree voltage constraint matrix, by TQ = TRR * TC (subroutine MULTS).

s. Solve TQ for a set of NRR dependent tree voltages in terms of NMR = NM - NRR independent tree voltages by Jordan elimination. Priority for dependence is greatest at the bottom of the tree branch list, FLIST (subroutine BAKELM).

t. From TQ form TL, the matrix which transforms the final independent set of tree voltages, VC, to the original list of tree voltages, VT, such that VT = TL * VC. Form VC from FLIST. Recalculate NPT(NE) to conform to the constituents of VC. Calculate NMR, the final number of simulation equations, from NM-NRR (subroutine STRIK).

u. Calculate TF, the final tree to node system change of basis matrix, from TF = TC * TL (subroutine MULTS).

v. Erase TTR, TQ, FLIST, TC, TL.

w. The node system current equilibrium equation matrices (CN, GN, LN, and IN) are formed for each element type C, G, L, and I appearing in WLIST. Each matrix is temporarily formed in four-dimensional type E list format in XS and written onto NTAPE1 in the order given above. Immediately following the I matrix, an END OF FILE is written onto NTAPE1 and NTAPE1 is rewound (subroutine PARAM).

x. In sequential order, each of the node system current equilibrium equation matrices, CN, GN, and LN is read from NTAPE1, temporarily stored in XS, and postmultiplied by the final tree to node system change of basis matrix, TF. The result of each multiplication is temporarily stored in XF and then written onto NTAPE2. This performs the first half of the node to tree system change of basis operation on the CN, GN, and LN matrices. The operation is XF = XS * TF (subroutine MULTS plus others).

y. Transfer IN, the node system current source equilibrium equations, directly from NTAPE1 to NTAPE2. Rewind NTAPE1 and NTAPE2. Erase XS.

z. Transpose the matrix TF to TFT (subroutine TRANS).

aa. Write TF onto NTAPE3. Erase TF.

bb. In sequential order, each of the partially transformed matrices, CN * TF, GN * TF, LN * TF, and IN is read from NTAPE2 into XF and premultiplied by TFT. The result of each multiplication is temporarily stored in XS and written onto NTAPE1. Thus the following tree basis equilibrium current equation matrices are formed: CT = TFT * CN * TF, GT = TFT * GN * TF, LT = TFT * LN * TF, IT = TFT * IN (subroutine MULTS and others). Erase XS and XF.

cc. Write END OF FILE on NTAPE1 and rewind NTAPE1 and NTAPE2. Erase TFT and XZ.

dd. VC, a two-dimensional type D list, is transformed into a four-dimensional type E representation of a node pair voltage vector, XZ. XZ has the following structure XZ(I, 1, NN, NP) = Data. I is the index of the particular branch descriptor in VC whose terminating nodes are NN and NP. NN and NP are forced to conform to the order displayed by the corresponding descriptor in WLIST rather than VC. If the node order in VC is reversed to that in WLIST, Data is set to -1. Otherwise Data is set to +1. If a descriptor if found in VC which does not correspond to a descriptor in WLIST, a dump is called and control is returned to the FORTRAN Monitor System.

ee. WLIST, a two-dimensional type D descriptor list, is transformed into a four-dimensional type E list representation of a descriptor list called INSTP. INSTP is stripped of all transformer winding descriptors, and voltage source descriptors are added to correspond to the initial condition voltage across each capacitor. The data entry for each filled position of INSTP is made according to INSTP(NE, 1, NN, NP) = 1.

ff. Erase WLIST.

gg. Write INSTP onto NTAPE3 immediately following TF.

hh. Write XZ onto NTAPE3 immediately following INSTP.

ii. Write END OF FILE on NTAPE3 and rewind.

jj. Erase INSTP and XZ.

kk. Write on output tape the status of AVS, COUNT, and MAX (subroutine STATUS).

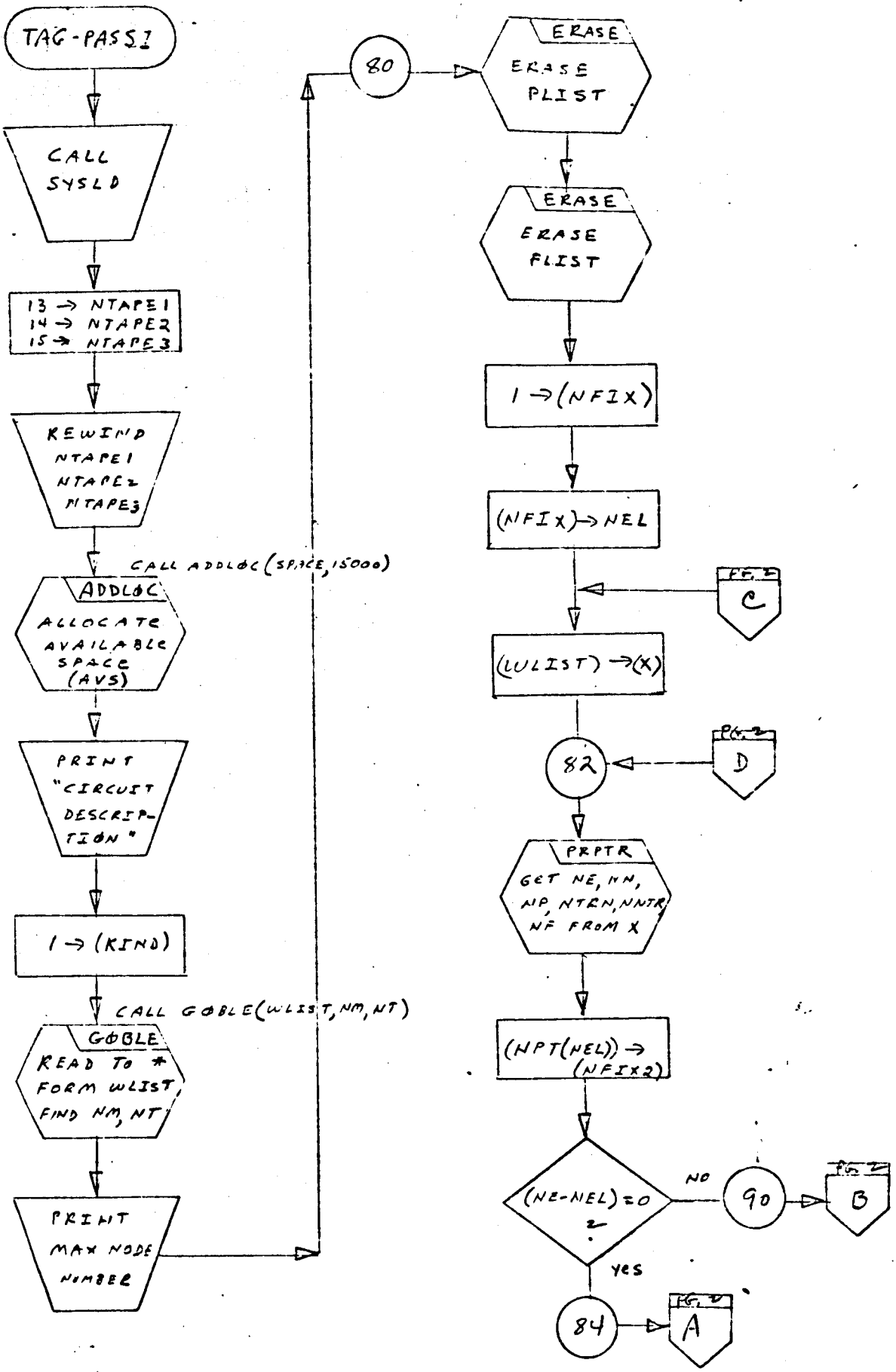
ll. Call in the second pass Main routine from tape (subroutine CHIN).

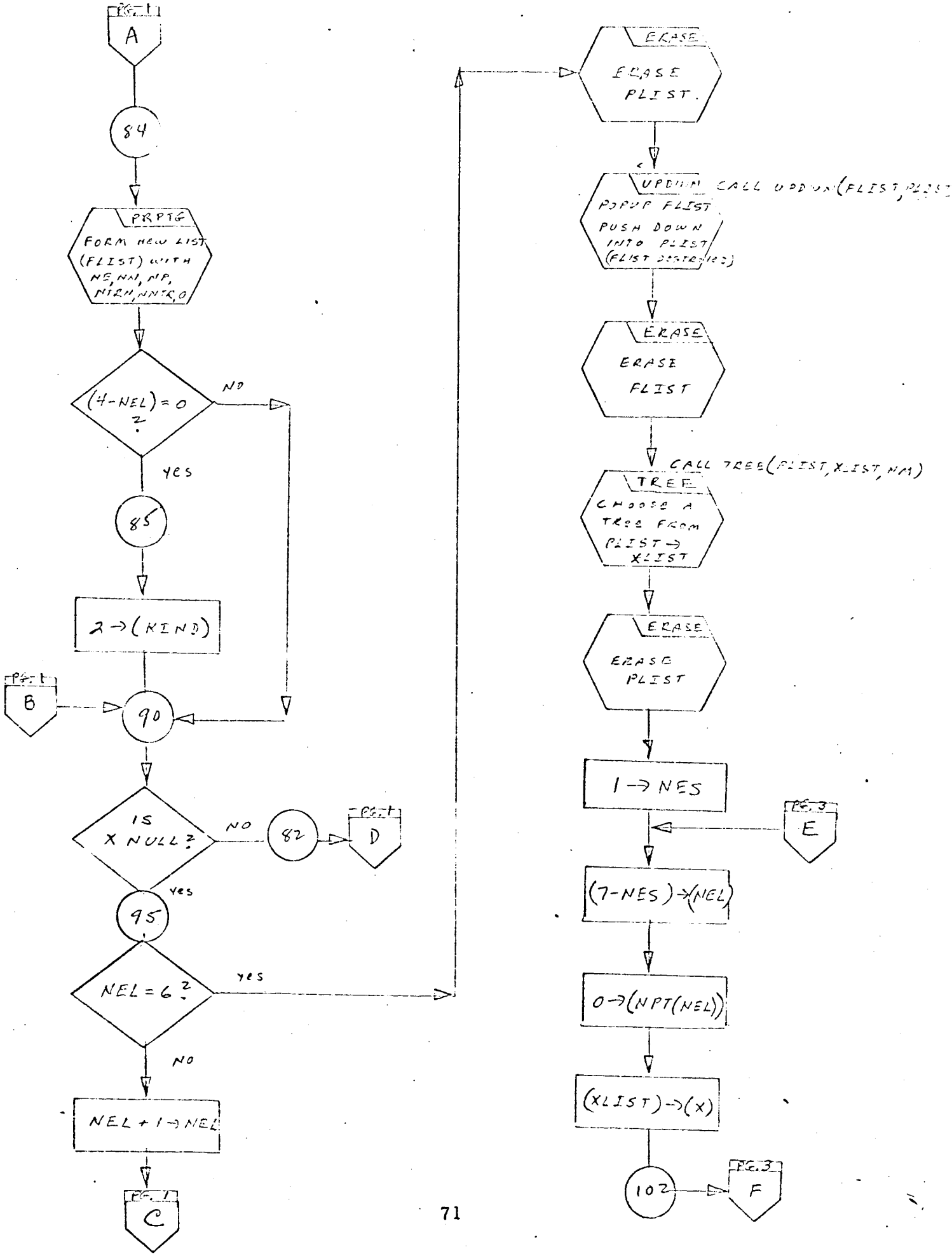
8. Other Subroutines Used

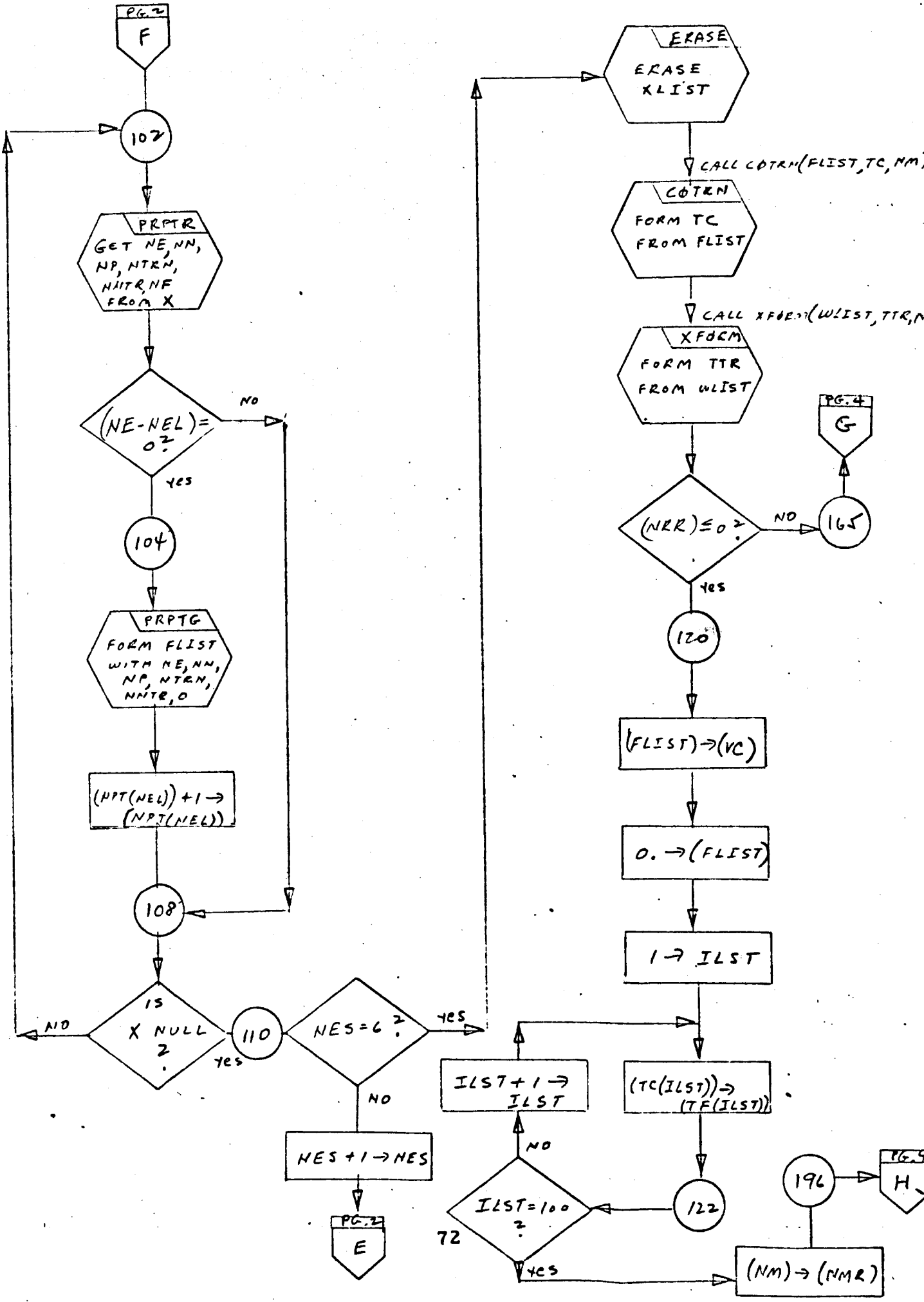
ADDLOC, BAKELM, CHIN, COTRN, DUMP, ERASEA, ERASE, EXIT, FREWND, GOBLE, MATFT, MATOT, MULTS, PARAM, PRPTG, PRPTR, STASH, STATUS, STRIK, TRANS, TREE, UPDWN, XFORM.

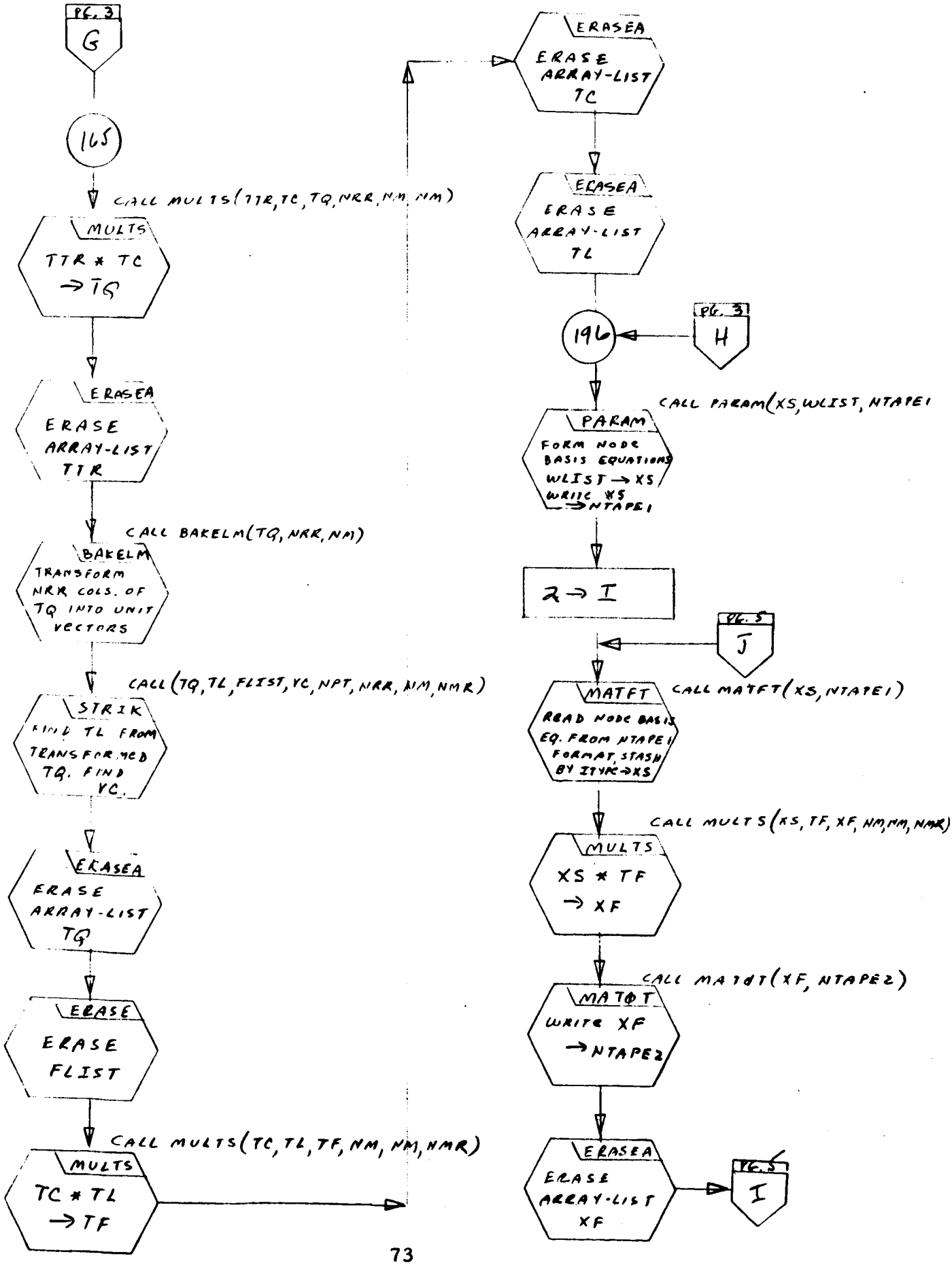
9. Using Subroutines

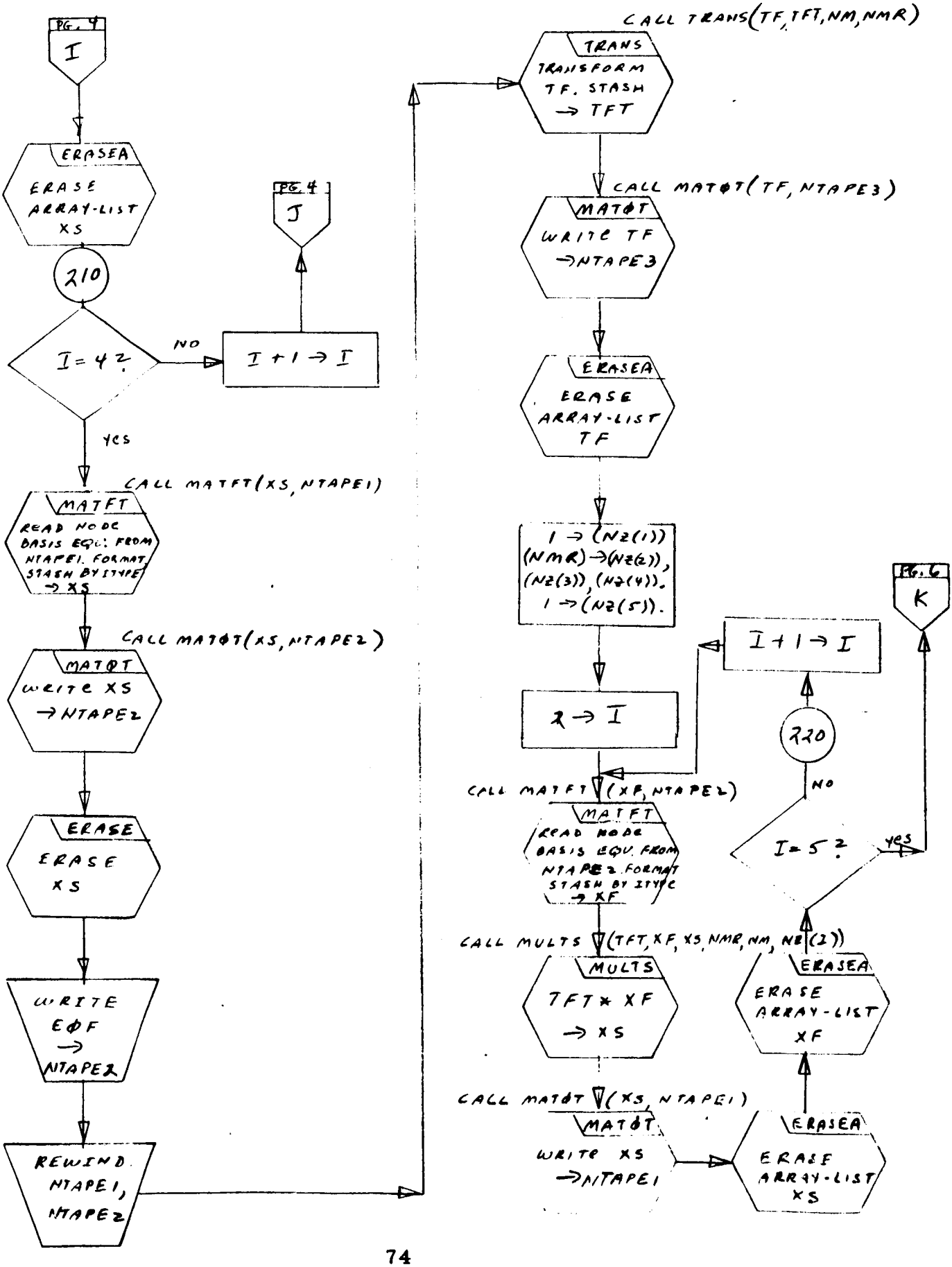
None.

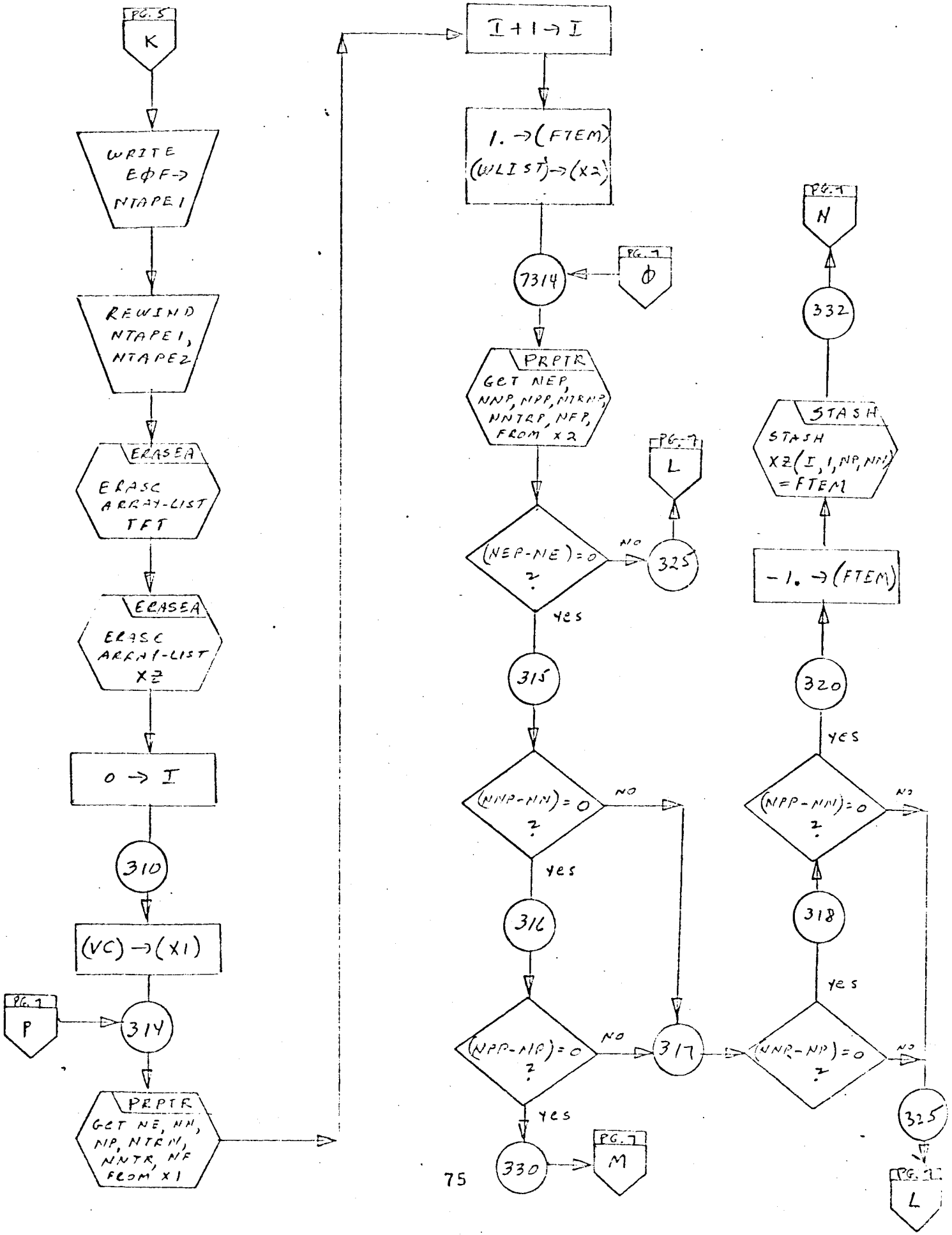

TAG-PASS1
CALL SYSLD
13 → NTAPE1
14 → NTAPE2
15 → NTAPE3
REWIND NTAPE1 NTAPE2 NTAPE3
CALL ADDLOC(SPACE,15000)
ADDLOC
ALLOCATE AVAILABLE SPACE (AVS)
PRINT "CIRCUIT DESCRIPTION"
1 → (KIND)
CALL GOBLE(WLIST,NM,NT)
GOBLE
READ TO * FORM WLIST, FIND NM, NT
PRINT MAX NODE NUMBER
80
ERASE
ERASE PLIST
ERASE
ERASE FLIST
1 → (NFIX)
(NFIX) → NEL
C
(LULIST) → (X)
82
D
PRPTR
GET NE, NN, NP, NTRN, NNTR, NF FROM X
(NPT(NEL)) → (NFIX2)
(NE-NEL) = 0 ?
NO
90
B
yes
84
A

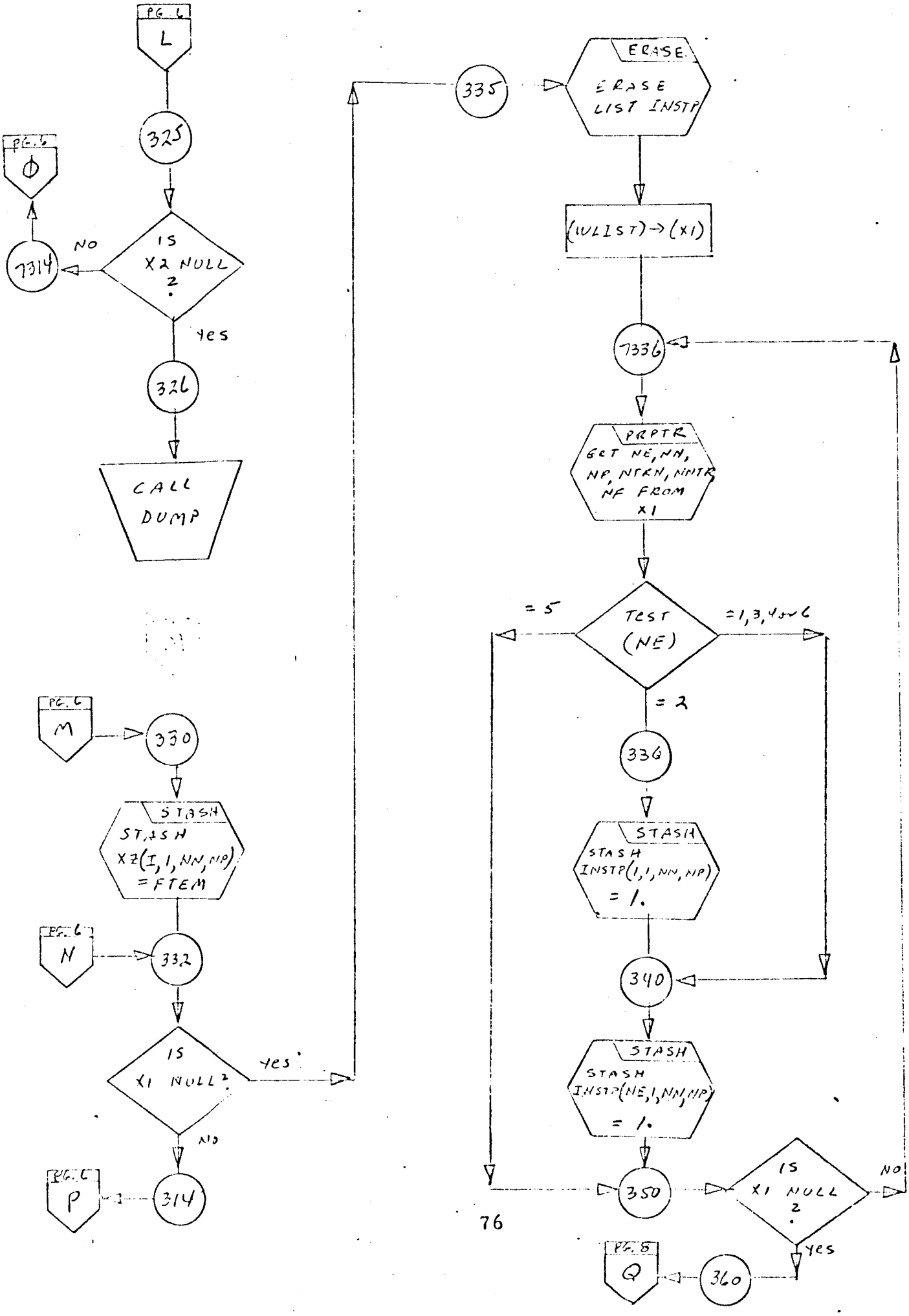

Pg. 1
A
84
PRPTG
FORM NEW LIST (FLIST) WITH NE, NN, NP, NTRM, NNTR, 0
(4-NEL) = 0 ?
NO
yes
85
2 → (KIND)
Pg. 1
B
90
IS X NULL ?
NO
82
Pg. 1
D
yes
95
NEL = 6 ?
yes
NO
NEL + 1 → NEL
Pg. 1
C
ERASE
ERASE PLIST.
UPDUM
CALL UPDUM(FLIST, PLIST)
POPUP FLIST
PUSH DOWN INTO PLIST
(FLIST DESTROYED)
ERASE
ERASE FLIST
CALL TREE(PLIST, XLIST, NM)
TREE
CHOOSE A TREE FROM PLIST → XLIST
ERASE
ERASE PLIST
1 → NES
Pg. 3
E
(7-NES) → (NEL)
0 → (NPT(NEL))
(XLIST) → (X)
102
Pg. 3
F

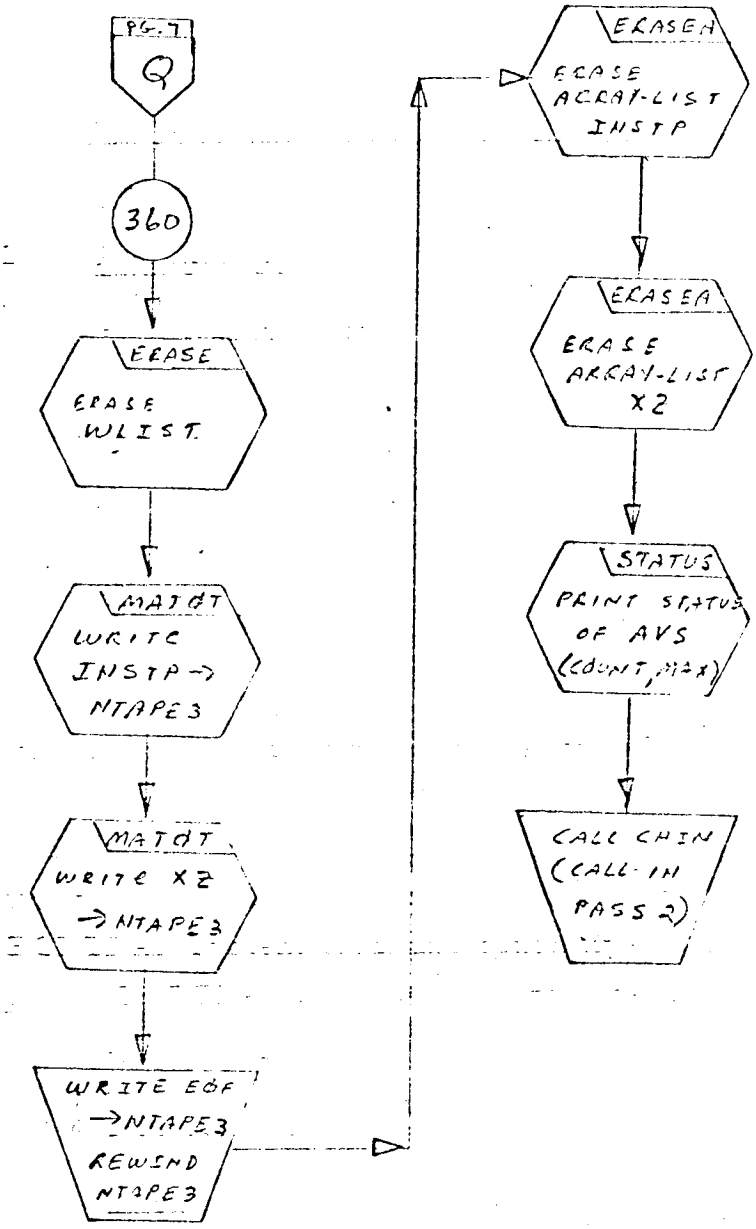

PG. 2
F
102
PRPTR
GET NE, NN, NP, NTRN, NNTR, NF FROM X
(NE-NEL)=0 ?
NO
yes
104
PRPTG
FORM FLIST WITH NE, NN, NP, NTRN, NNTR, 0
(NPT(NEL))+1→(NPT(NEL))
108
IS X NULL ?
NO
yes
110
NES=6 ?
yes
NO
NES+1→NES
PG. 2
E
ERASE
ERASE XLIST
CALL COTRN(FLIST, TC, NM)
COTRN
FORM TC FROM FLIST
CALL XFORM(WLIST, TTR, N
XFORM
FORM TTR FROM WLIST
PG. 4
G
(NRR)≤0 ?
NO
165
yes
120
(FLIST)→(VC)
0.→(FLIST)
1→ILST
(TC(ILST))→(TF(ILST))
122
ILST+1→ILST
NO
ILST=100 ?
yes
196
PG. 9
H
(NM)→(NMR)


pg. 3
G
165
CALL MULTS(TTR,TC,TQ,NRR,NM,NM)
MULTS
TTR * TC
→ TQ
ERASEA
ERASE
ARRAY-LIST
TTR
CALL BAKELM(TQ,NRR,NM)
BAKELM
TRANSFORM
NRR COLS. OF
TQ INTO UNIT
VECTORS
CALL (TQ,TL,FLIST,VC,NPT,NRR,NM,NMR)
STRIK
FIND TL FROM
TRANSFORMED
TQ. FIND
VC.
ERASEA
ERASE
ARRAY-LIST
TQ
ERASE
ERASE
FLIST
CALL MULTS(TC,TL,TF,NM,NM,NMR)
MULTS
TC * TL
→ TF
ERASEA
ERASE
ARRAY-LIST
TC
ERASEA
ERASE
ARRAY-LIST
TL
196
pg. 3
H
CALL PARAM(XS,WLIST,NTAPE1
PARAM
FORM NODE
BASIS EQUATIONS
WLIST → XS
WRITE XS
→ NTAPE1
2 → I
pg. 5
J
MATFT
CALL MATFT(XS,NTAPE1)
READ NODE BASIS
EQ. FROM NTAPE1
FORMAT, STASH
BY ITYPE → XS
CALL MULTS(XS,TF,XF,NM,NM,NMR)
MULTS
XS * TF
→ XF
CALL MATOT(XF,NTAPE2)
MATOT
WRITE XF
→ NTAPE2
ERASEA
ERASE
ARRAY-LIST
XF
pg. 5
I


PG. 9
I
ERASEA
ERASE ARRAY-LIST XS
210
I = 4 ?
NO
I + 1 → I
PG. 4
J
YES
CALL MATFT(XS, NTAPE1)
MATFT
READ NODE BASIS EQU. FROM NTAPE1. FORMAT STASH BY ITYPE → XS
CALL MATØT(XS, NTAPE2)
MATØT
WRITE XS → NTAPE2
ERASE
ERASE XS
WRITE EØF → NTAPE2
REWIND NTAPE1, NTAPE2
CALL TRANS(TF, TFT, NM, NMR)
TRANS
TRANSFORM TF. STASH → TFT
CALL MATØT(TF, NTAPE3)
MATØT
WRITE TF → NTAPE3
ERASEA
ERASE ARRAY-LIST TF
1 → (N2(1))
(NMR) → (N2(2)), (N2(3)), (N2(4)).
1 → (N2(5)).
2 → I
CALL MATFT (XF, NTAPE2)
MATFT
READ NODE BASIS EQU. FROM NTAPE2. FORMAT STASH BY ITYPE → XF
CALL MULTS (TFT, XF, XS, NMR, NM, NB(2))
MULTS
TFT * XF → XS
CALL MATØT (XS, NTAPE1)
MATØT
WRITE XS → NTAPE1
ERASEA
ERASE ARRAY-LIST XS
ERASEA
ERASE ARRAY-LIST XF
I + 1 → I
220
NO
I = 5 ?
YES
PG. 6
K


PG. 5
K
WRITE EΦF→ NTAPE1
REWIND NTAPE1, NTAPE2
ERASEA
ERASE ARRAY-LIST TFT
ERASEA
ERASE ARRAY-LIST X2
0 → I
310
(VC) → (X1)
PG. 1
P
314
PRPTR
GET NE, NN, NP, NTRN, NNTR, NF FROM X1
I+1→I
1. → (FTEM)
(WLIST) → (X2)
7314
PG. 1
Φ
PRPTR
GET NEP, NNP, NPP, NTRNP, NNTRP, NFP, FROM X2
PG. 1
L
(NEP-NE)=0 ?
NO
325
yes
315
(NNP-NN)=0 ?
NO
yes
316
(NPP-NP)=0 ?
NO
317
yes
330
PG. 1
M
PG. 1
N
332
STASH
STASH X2(I, 1, NP, NN) = FTEM
-1. → (FTEM)
320
yes
(NPP-NN)=0 ?
NO
318
yes
(NNP-NP)=0 ?
NO
325
PG. 2
L


PG. 6
L
325
PG. 6
Ø
7314
NO
IS X2 NULL ?
yes
326
CALL DUMP
335
ERASE
ERASE LIST INSTP
(WLIST) → (X1)
7336
PRPTR
GET NE, NN, NP, NTRN, NINTR, NF FROM X1
Test (NE)
= 5
= 1, 3, 4 or 6
= 2
336
STASH
STASH INSTP(1, 1, NN, NP) = 1.
340
STASH
STASH INSTP(NE, 1, NN, NP) = 1.
350
IS X1 NULL ?
NO
yes
PG. 6
Q
360
PG. 6
M
330
STASH
STASH X2(I, 1, NN, NP) = FTEM
PG. 6
N
332
IS X1 NULL ?
yes
NO
PG. 6
P
314


PG.T
Q
360
ERASE
ERASE WLIST
MAIOT
WRITE INSTP → NTAPE3
MAIOT
WRITE X2 → NTAPE3
WRITE EOF → NTAPE3
REWIND NTAPE3
ERASEA
ERASE ARRAY-LIST INSTP
ERASEA
ERASE ARRAY-LIST X2
STATUS
PRINT STATUS OF AVS (COUNT, MAX)
CALL CHIN (CALL-IN PASS 2)

C. Second Pass Writeup and Flow Chart

1. Program Description

2. Identification

a. Routine Label

TAG

b. Name

Preprocessor, Main program, Pass 2.

3. Function

The second pass of the Preprocessor takes the equations generated in the first pass, classifies them as linear or nonlinear, and imbeds them in the proper places in a FORTRAN simulation program. To this program it adds all nonlinear relationships and the output and control sequence specified in the part of the TAG Description Deck which follows the connection list. This program is written out on the FORTRAN System print and punch tapes. It is immediately compiled by the FORTRAN Compiler and may be immediately executed if desired.

4. Programming System

FORTRAN II

5. Usage

a. Calling Sequence

The precompiled binary form of the second pass main program is loaded from tape under control of the FORTRAN Monitor System through a chaining operation.

b. Entry Conditions

Variables preset in COMMON by Pass 1.

KIND = The inductor flag; equals 1 if there are no inductors in the circuit and 2 if there are inductors in the circuit

NM = The total number of circuit nodes less 1.

NMR = The total number of simulation equations and therefore the total number of independent voltage variables

NPT = The tree voltage vector partition array which stores the number of each type of element that appears in the tree. NPT(NE) is equal to the number of NE type elements in the tree.

SPACE = The memory space available for list structures

Matrices generated in the first pass are stored on scratch tape.

Scratch tape, NTAPE1, contains the final cut-set equation matrices for the capacitive, resistive, inductive, and current source topologies stored in that order in a list-type format.

Scratch tape, NTAPE2, contains the partially transformed node basis equation matrices for the capacitive, resistive, inductive, and current source topologies stored in that order in a list-type format. Scratch tape, NTAPE3, contains, in order, the final tree voltage to node voltage basis transformation matrix, TF; the input symbol table for Pass 1 with the names of each branch element (except transformer elements) plus the initial conditions voltages for all capacitors; and the final tree voltage vector corresponding to VC.

SWSW is a sense switch which may be set to delete the second pass.

c. Exit Conditions

The FORTRAN code for the TAG simulation program which was generated by the second pass is output on the FORTRAN System print tape and punch tape. Control is then returned to the FORTRAN Monitor System in such a way as to allow immediate compilation and execution of the generated program.

d. Error Exits

(1) If sense switch, SWSW, is less than or equal to 0, exit is made at the beginning of the second pass.

(2) If dependent variable stop functions are included in a DC steady-state problem, the message "DEPENDENT VARIABLE STATEMENTS HAVE NO MEANING IN A DC PROBLEM" is printed and an exit is made.

6. Definition of Identifiers

NTAPE1 NTAPE2 NTAPE3	Three scratch tapes holding circuit matrices generated by Pass 1.
INTAPE NFRM	Holds tape number of input tape. Holds next number to be assigned to FORMAT statements
IEQ	BCD equals sign
EQST EQST1 EQST2	Temp. lists used to hold parts of equation statements
IBLNK	BCD blank
ICHC	BCD "C"
FRM	BCD list "FORMAT (IH"
ICMA	BCD comma
WRT	BCD list "WRITE OUTPUT TAPE 6,"
IXFR	BCD "X"
IHFR	BCD "H"
IFXED	BCD list "=,I5)"
IFLED	BCD list "=,E16.8)"
REST2	BCD list "FT, FSTEP, FEPSL, FEPSL1, FEPSL2, FEPSL3, LDBG01, LMAX, LTYPE, FOUT, FEBL4"
REST	List used for holding symbols
INLST	List used for holding symbols
TC	Matrix list used to hold matrices input from tape
KELST(8)	Holds for each I, the number of statements of class KE = 1.
IC1	Flag to indicate comment card
ISTOP	Statement number of first executable input statement.
INLIN	Nonlinear flag
RSUT	Used to hold the contents of RIN for examination
KE	Statement-type indicator
IPR	Print switch
RIN	Holds list input as a card image

IS	Holds statement number of card input
IC	Continuation-card indicator
AA	Dummy variable
KDUMY	Dummy variable
LSTMK	List holding dep. stop function names
WRT	Holds the contents of RIN for output
ITEMX	Holds next link of RIN
ITEM	Holds next item of RIN
NHC	Used for counting number of characters in a statement
ELE2	Temp. list
NCHL	Holds number of characters in a FORMAT line.
IFX	BCD temporary storage
KINDSV	Holds saved value of KIND
KDIF	Used to sum the matrix dimensions LNV, LNC, etc.
IL, KEL	Temporary index for DO-loops

7. Method

(Paragraph numbers indicate approximate statement locations)

Initialize

400. Read statements from input program. Each statement is sent to SUBST for variable substitutions. SUBST will examine for nonlinearities, and if any exist will set INLIN = 1. If any dependent-stop-function variables exist, they will be collected in list LSTMK. The statement will be classified as to type, and KE set to this type number. Each revised statement is written out on INTAPE along with its associated KE-value. If KE = 2 and the statement is nonlinear or if KE> 2 and the statement is not nonlinear, then a "WRITE OUTPUT TAPE" statement is set up incorporating the variable on the left side of the statement.

603. Read NTAPE1 into XS and mark the items with flags from XFG, as set by subroutine SUBST. (The flag indicates in which one of four sets the variable is to be output.). After XS has been marked, it is written onto NTAPE2.

604. Output "DIMENSION" statements

Output: LNV = NPT(1)

LNC = NPT(2)

LNG = NPT(3)

LNL = NPT(4)

Read NTAPE3 into INSTP.

605. If this is a transient problem, output:

FSTEP = 1.E-11

FEPSL2 = 5.E-6

FEPSL3 = 5.E-4

FEPSL4 = 1.E-16

LTYPE = 4

608. If this is a nonlinear DC problem, output:

FEPSL = 5.E-6

FEPSL1 = 5.E-6

LDBG01 = 51

LMAX = 50

609. Output: GO TO 6000

610. Read the statements from INTAPE and write out those which have KE = 0. If the statement is "INPUT," output the "CALL INPUT" statement. If the statement is "ZERO," output the "CALL ZEROX" statements.

650. Output the "CALL ZEROX" statements with statement number 6000.

Output the "CALL INPUT" statement.

output: 6100 CONTINUE

LALGFT = 1

If transient problem, output:

LINT = 0

LCNT = 1

FTL = FT

FTO = FT

FHC = FSTEP

658. Read NTAPE3 into XS and output all statements of the form

FV··=X*SV··

660. Read INTAPE, and write out all statements which have KE = 3.

680. Read NTAPE2 into XS, and for each variable flagged as = 0, output:

FC..= X.SC.. file 1

Read successive files of NTAPE2 and similarly output:

FG..=X*SG.. file 2

FL..=X*SL.. file 3

FI ..=X*SI.. file 4

685. Output "CALL INV" statements through INVST.

Output:

CALL RSTOP (FSTOP, FT, FHC)

If there are dependent stop variables, output:

6200 IF (LCNT-3) 6202, 6201, 6395 and go to 788.

If this is a transient problem, output:

6200 IF (LCNT-3) 6202, 6201, 6201 and go to 788.

Else output:

IF(LALGFT-1) 6202, 6202, 6200

6200 CONTINUE

6201 CALL STOP (FOUT, LINT)

6202 CONTINUE

and go to 789.

788. Output:

6201 CALL STOP (FOUT, LINT)

6202 CONTINUE

789. Read the statements on INTAPE and output all statements which have KE = 4. Then output all which have KE = 5.

722. Read NTAPE2 into XS, and for flag = 1, and each file, output statements of form:

FC..= X*SC.. file 1

FG..= X*SG.. file 2

FL..= X*SL.. file 3

FI.. = X*SI.. file 4

Output "CALL INV" statements through INVST. If this is a DC problem and there exist any dependent-STOP-variables, write out

"DEPENDENT VARIABLE STATEMENTS HAVE NO MEANING IN A DC PROBLEM," and then exit. Else if there are no dependent stop functions and this is a DC problem, go to 840.

838. Output:

```
      GO TO (6395,6390), LALGFT
```

If there are no dependent stop functions, go to 7860, else output:

```
6390  IF (LINT) 6391,6391,6393
6391  FSTOP = FSTOP - FTL
      CALL ROUT(0)
6393  FSTOP = FSTOP - FT GO TO 6425
6395  LEOS = 1
6400  CONTINUE
      LCNT = LCNT - 3
```

7840. Read statements from INTAPE and output those with KE = 8
Read NTAPE2, and for variables with flag = 2, output the statements:

```
FC..=X*SC..    for file 1
FG..=X*SG..        file 2
FL..=X*SL..        file 3
FI ..=X*SI..       file 4
```

7852. Output the "CALL INV" statements.
Output:

```
      GO TO (6422, 6420), LALGFT
6420  IF (LEOS) 6421,6421,6422
6421  CALL ROUT(0)
6422  FSTOP = 0.
6425  FT1 = 0.
      FT2 = 0.
```

```
            FHB(1) = FSTEP
            FHB(2) = 1.E-5
            FHB(3) = FEPSL4
            FHB(4) = .5
            FHB(5) = FEPSL2
            FHB(6) = FEPSL3
            LNH(1) = contents of KDIF
            LNH(2) = LNH(1)
            LNH(5) = 5
            CALL FMARK(····
            FT = FTL + FT1
            LEOS = 0
            IF (LCNT -2 ) 6300, 6400, c(ISTOP)
      6300  CONTINUE
      Go to 840.
7860. Output:
      6390  IF (LINT) 6392,6392,6395
      6392  FSTOP = FSTOP - FTL
            CALL ROUT(0)
      6395  FSTOP = FSTOP - FT
            FT1 = 0.
            FT2 = 0.
            FTL = FT
            FHB(1) = FSTEP
            FHB(2) = 1.E-5
            FHB(3) = FEPSL4
            FHB(4) = .5
            FHB(5) = FEPSL2
            FHB(6) = FEPSL3
            LNH(1) = contents of KDIF
            LNH(2) = LNH(2)
            LNH(5) = 5
            CALL FMARK (···)
            FT = FTL + FT1
```

IF (LCNT-2) 6300, 6300, c(ISTOP)

6300 CONTINUE

840. If nonlinear solution, output:

DIMENSION FVR, FVP, FPT

LCNV = 0

6090 CALL SHFTO (FVP, FV31, FV41, LNG, LNL)

6310 CONTINUE

CALL SHFIN (FVP, FV31, FV41, LNG, LNL)

Read INTAPE and output all statements which have KE = 6. Read NTAPE2 into XS and output statements for variables with flag = 3.

FC··= X*SC·· file 1

FG··=X*SG·· file 2

FL··=X*SL·· file 3

FI··=X*SI·· file 4

Output "CALL INV" statements.

Output matrix manipulations through EQFS41. If LNL = 0, go to 827.

823. Output:

FDLT = FT - FTO

IF (FDLT) 7007, 7007, 7005

If LNL > 1, go to 825.

824. Output:

7005 FV41 = (FS41 - FSO41)/FDLT

7007 FSO41 = FS41

Go to 827.

825. Output:

7005 DO 7006 L = 1, LNL

7006 FV41(L) = (FS41(L) - FSO41(L))/FDLT

7007 DO 7008 L = 1, LNL

7008 FSO41(L) = FS41(L)

827. Output "CAL INV" statements.

Output matrix manipulations through EQFV31.

If <u>not</u> nonlinear, output:

LALGFT = 2

Then go to 835.

821. Else output:

```
      Go to (7005, 7010), LALGFT
7005  LALGFT = 2
      Go to 6090
7010  CALL SHFDI (FVR,FV31,FV41,FVP,LNG,LNL)
      IF (LCNV-1) 7015,7020,6000
7015  CALL ROOT (FVR,FVP,FPT,FEPSL,FEPSL1,
      LMAX,LNG + LNL, LCNV,LDBGO1)
      Go to 6310
```

If this is not a transient problem, go to 850. If <u>not</u> non-linear, go to 835, else continue.

Output:

```
      7020 CONTINUE
```

835. If LNV = 0 or LNC = 0, Go to 837. Else continue.

```
      If LNL = 0, output:
      FDLT = FT - FTO
```

Output:

```
      IF (FDLT) 7030,7030,7025
```

If LNV ≤ 1, output:

```
      7025 FVD11 = (FV11 - FVO11)/FDLT
      7030 FVQ11 = FV11
```

and go to 7837.

Else output:

```
      7025 DO 7026 L = 1,LNV
      7026 FVD11(L) = (FV11(L) - FV011(L))/FDLT
      7030 DO 7031 L = 1, LNV
      7031 FVO11(L) = FV11(L)
```

7837. Output:

```
      FTO = FT
```

837. Output "CALL INV" statements.

Output matrix equations through EQFV21.

If (ISIS) ≤ 1 and KIND = 1, go to 886, else continue.

If LNV ≤ 1, output:

```
      FSD11 = FV11
```

Else output:

DO 7040 L = 1, LNV

FSD11(L) = FV11(L)

882. If LNC $\leq$ 1, output:

FSD21 = FV21

Else output:

DO 7042 L = 1, LNC

7042 FSD21(L) = FV21(L)

883. If LNG $\leq$ 1, output:

FSD31 = FV31

Else output:

DO 7014 L = 1, LNG

7044 FSD31(L) = FV31(L)

886. Output:

CALL ROUT(0)

and go to 870.

850. Output:

7020 FT = FSTOP

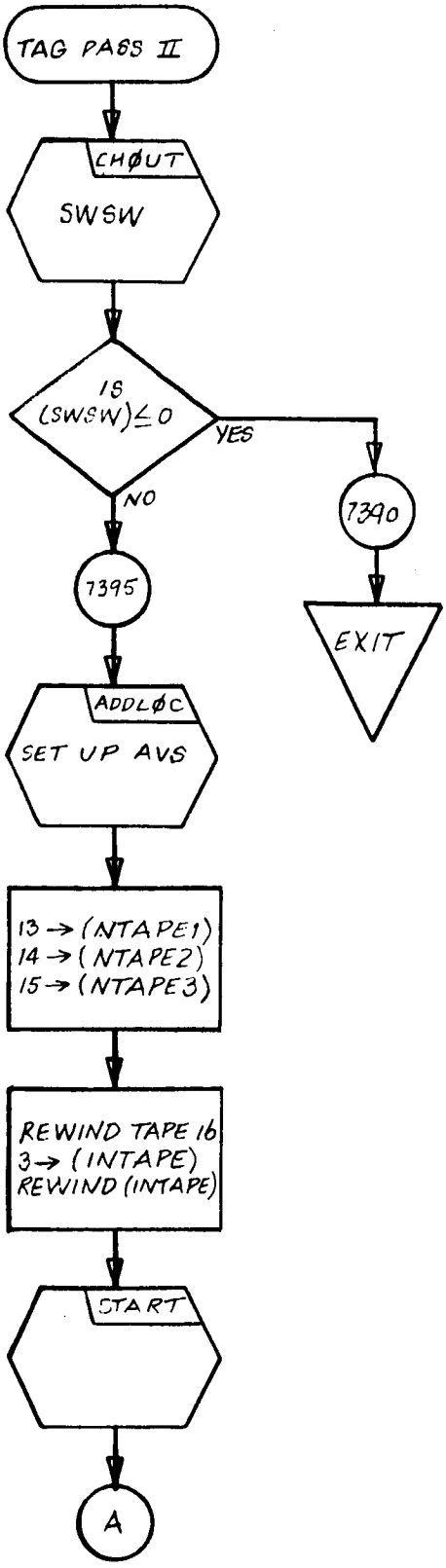
GO TO I

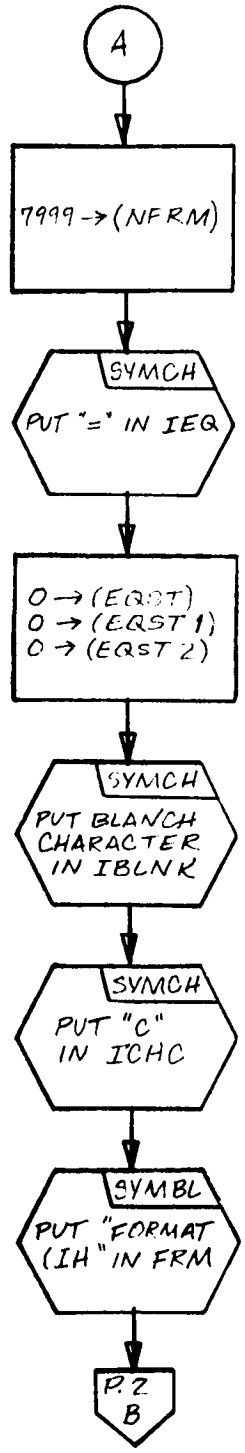
where I is the contents of ISTOP.

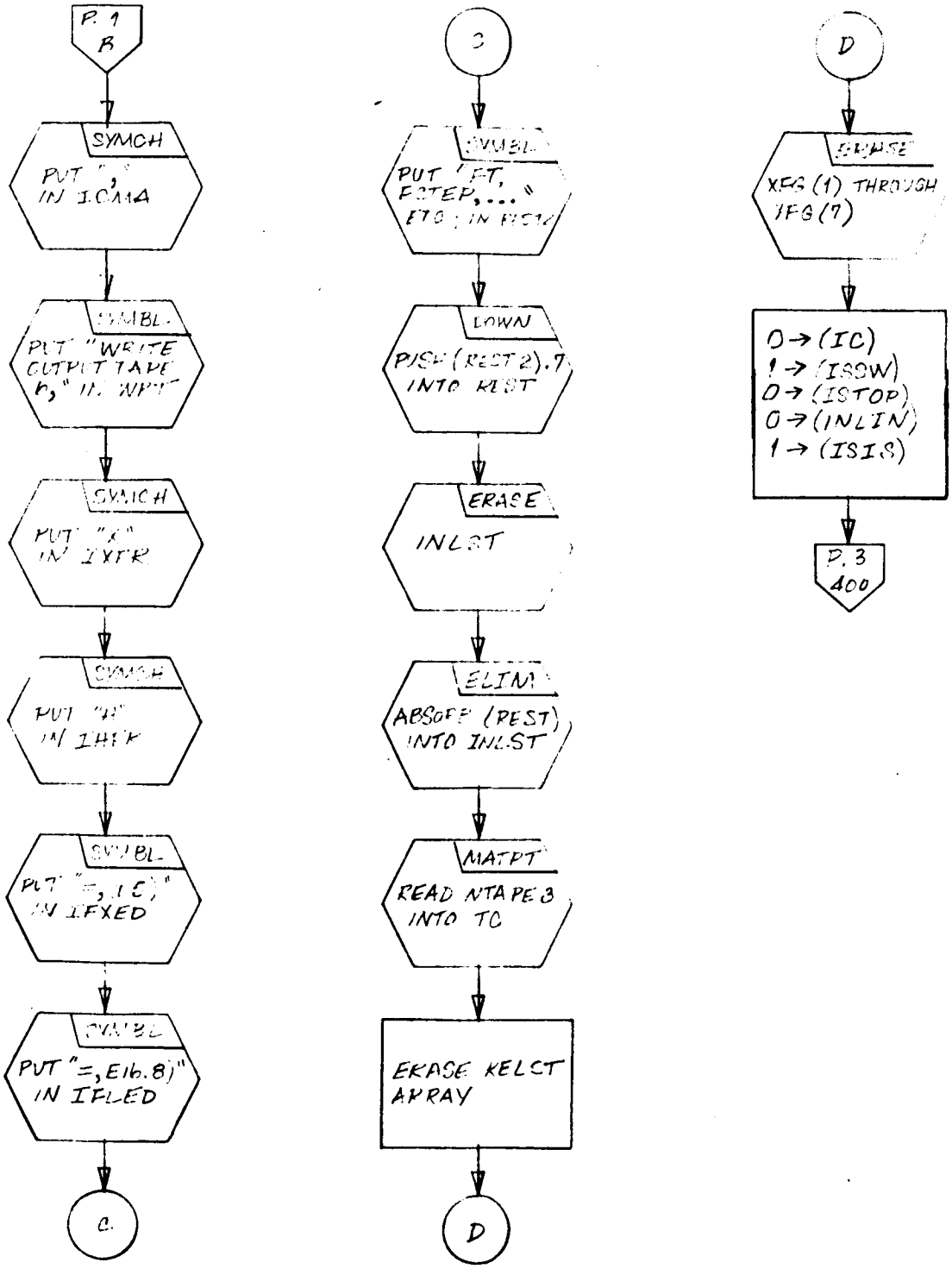
870. Output:

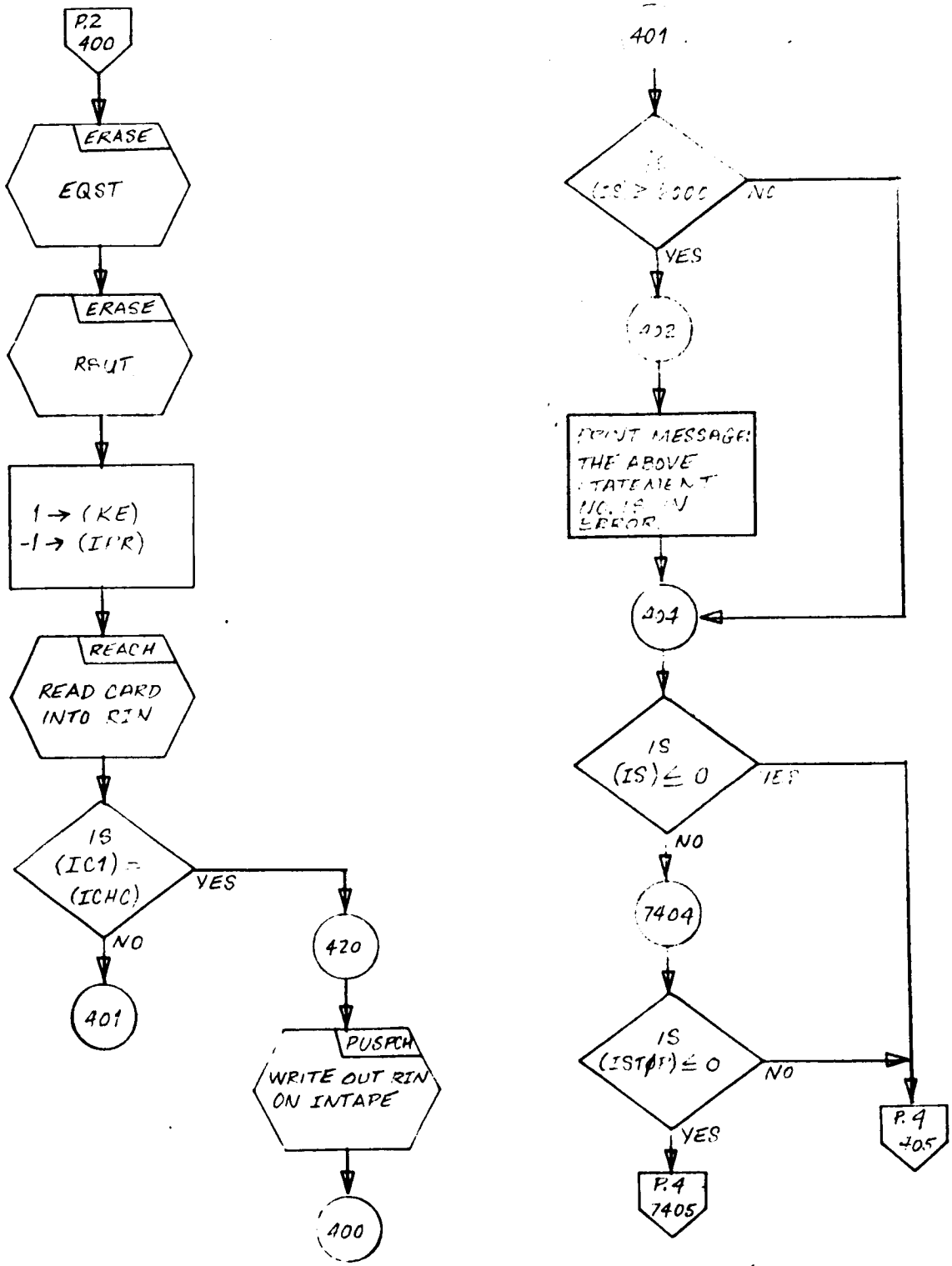
END(1,0,0,0,0,0,1,0,0,0,0,0,0,0,0)

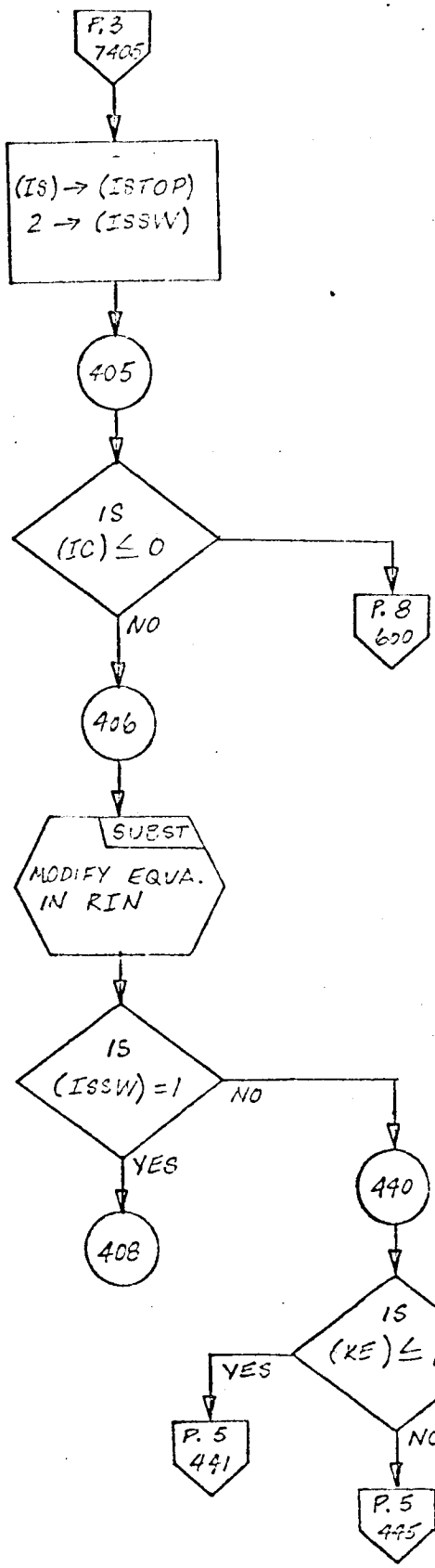
Erase all lists, write end-of-files on output tape, and terminate.

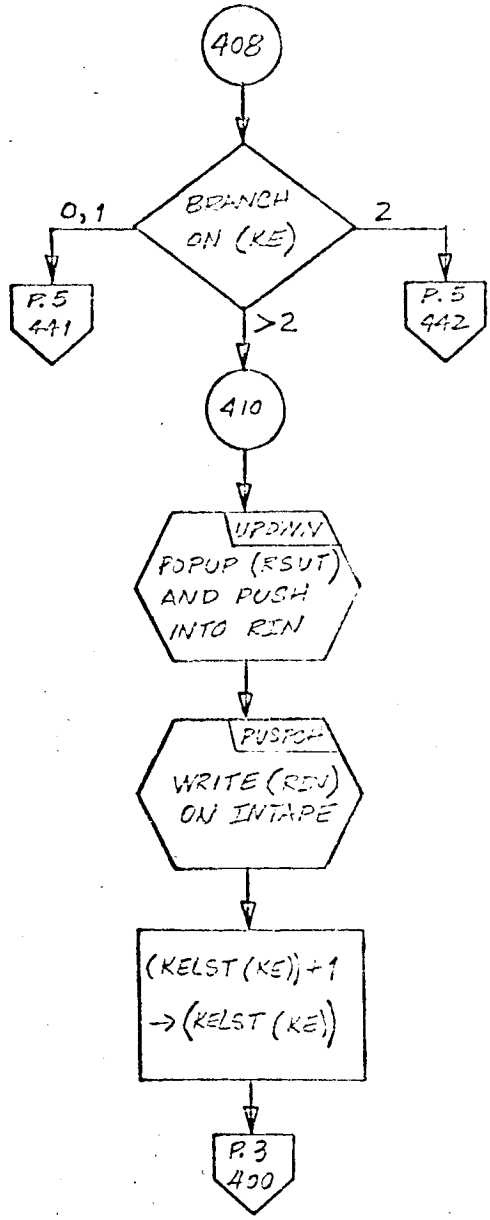

TAG PASS II
CHØUT
SWSW
IS (SWSW) ≤ 0
YES
NO
7390
EXIT
7395
ADDLØC
SET UP AVS
13 → (NTAPE1)
14 → (NTAPE2)
15 → (NTAPE3)
REWIND TAPE 16
3 → (INTAPE)
REWIND (INTAPE)
START
A

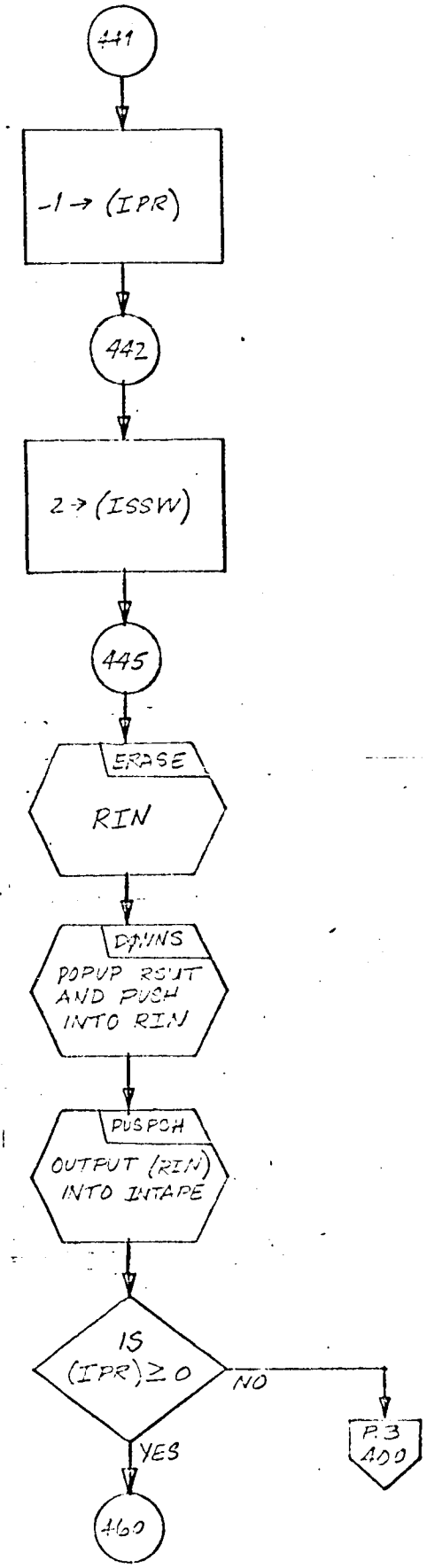

A
7999 → (NFRM)
SYMCH
PUT "=" IN IEQ
0 → (EQST)
0 → (EQST 1)
0 → (EQST 2)
SYMCH
PUT BLANCH CHARACTER IN IBLNK
SYMCH
PUT "C" IN ICHC
SYMBL
PUT "FORMAT (IH " IN FRM
P. 2 B

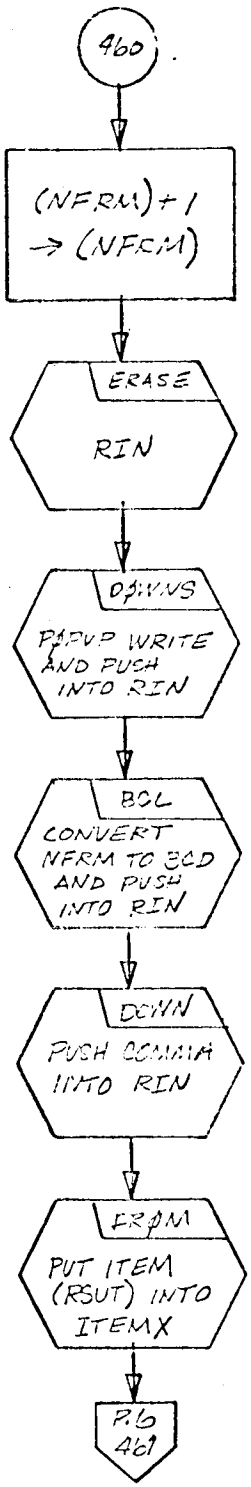

P. 1
B
SYMCH
PUT ',' IN ICMA
SYMBL
PUT "WRITE OUTPUT TAPE 6," IL. WRIT
SYMCH
PUT "X" IN IXTR
SYMCH
PUT "H" IN IHTR
SYMBL
PUT "=,1E)" IN IFXED
SYMBL
PUT "=,E16.8)" IN IFLED
C
C
SYMBL
PUT 'FT, FSTEP,...' ETC ; IN REST
LOWN
PUSH (REST2).7 INTO REST
ERASE
INLST
ELIM
ABSOFF (REST) INTO INLST
MATPT
READ NTAPE 3 INTO TC
ERASE KELCT ARRAY
D
D
SYMBOL
XFG (1) THROUGH XFG (7)
0 → (IC)
1 → (ISSW)
0 → (ISTOP)
0 → (INLIN)
1 → (ISIS)
P. 3
400

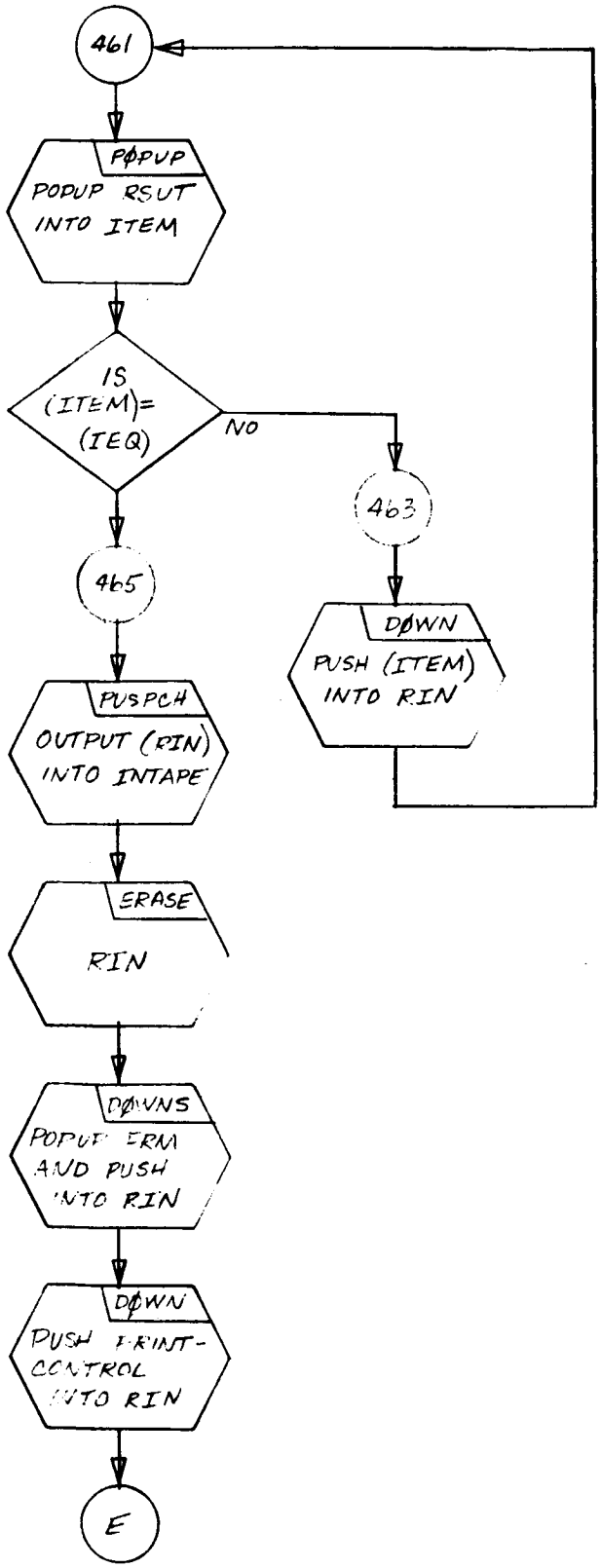

P.2
400
ERASE
EQST
ERASE
RBUT
1 → (KE)
-1 → (IPR)
READ
READ CARD
INTO RIN
IS
(IC1) =
(ICHC)
YES
NO
401
420
PUNCH
WRITE OUT RIN
ON INTAPE
400
401
IS
(IS) > 5000
NO
YES
402
PRINT MESSAGE:
THE ABOVE
STATEMENT
NO. IS AN
ERROR
404
IS
(IS) ≤ 0
NO
7404
IS
(IST(I)) ≤ 0
NO
YES
P.4
7405
P.4
405

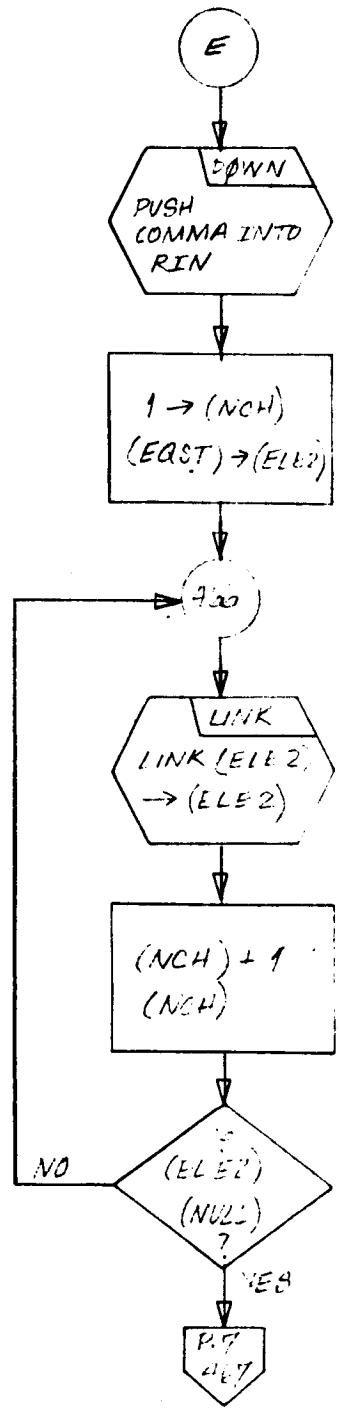

P. 3
7405
(IS) → (ISTOP)
2 → (ISSW)
405
IS
(IC) ≤ 0
NO
P. 8
600
406
SUBST
MODIFY EQUA.
IN RIN
IS
(ISSW) = 1
NO
YES
408
440
IS
(KE) ≤ 1
YES
P. 5
441
NO
P. 5
445

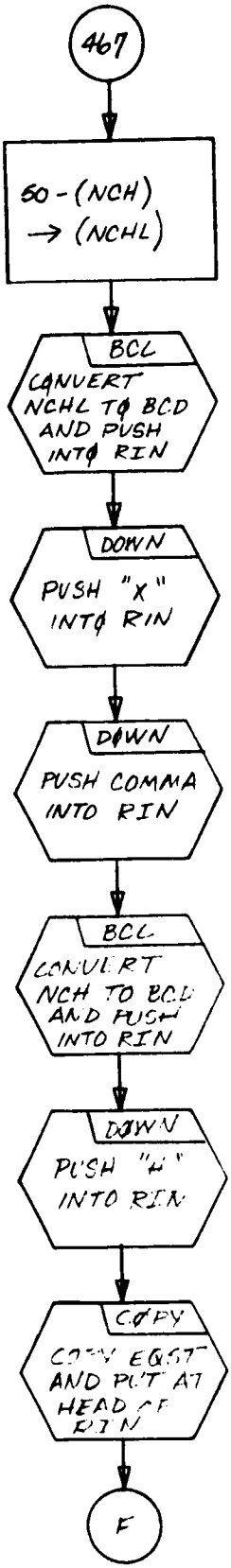

408
BRANCH
ON (KE)
0, 1
2
>2
P. 5
441
P. 5
442
410
UPDNV
POPUP (RSUT)
AND PUSH
INTO RIN
PUSPCH
WRITE (RIN)
ON INTAPE
(KELST (KE)) + 1
→ (KELST (KE))
P. 3
400

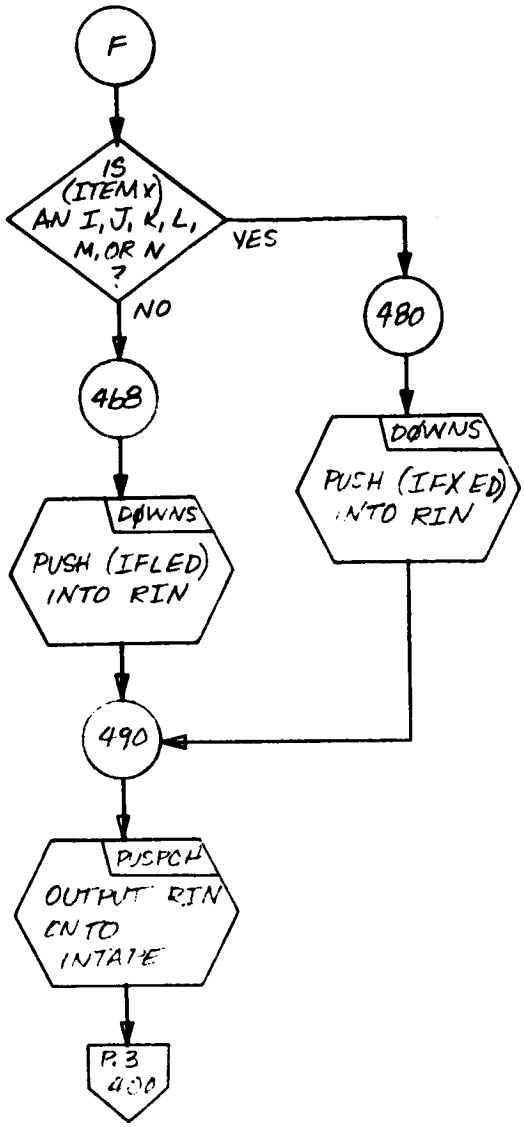

441
-1 → (IPR)
442
2 → (ISSW)
445
ERASE
RIN
DOWNS
POPUP RSUT AND PUSH INTO RIN
PUSPCH
OUTPUT (RIN) INTO INTAPE
IS (IPR) ≥ 0
NO
P.3 400
YES
460

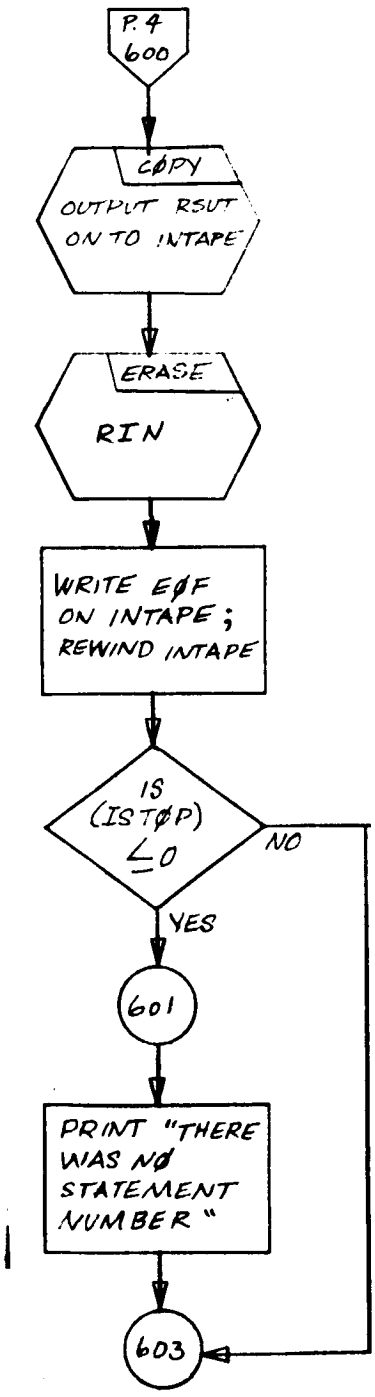

460
(NFRM) + 1 → (NFCM)
ERASE
RIN
DOWNS
POPUP WRITE AND PUSH INTO RIN
BCL
CONVERT NFRM TO BCD AND PUSH INTO RIN
DOWN
PUSH COMMA INTO RIN
FROM
PUT ITEM (RSUT) INTO ITEMX
P.6 461

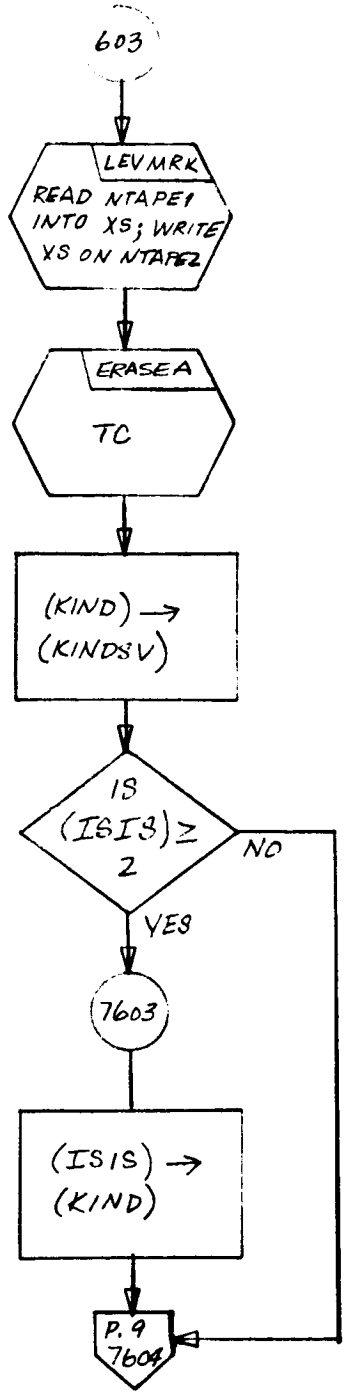

461
PØPUP
PODUP RSUT
INTO ITEM
IS
(ITEM)=
(IEQ)
NO
463
DØWN
PUSH (ITEM)
INTO RIN
465
PUSPCH
OUTPUT (RIN)
INTO INTAPE
ERASE
RIN
DØVNS
POPUP TERM
AND PUSH
INTO RIN
DØWN
PUSH PRINT-
CONTROL
INTO RIN
E

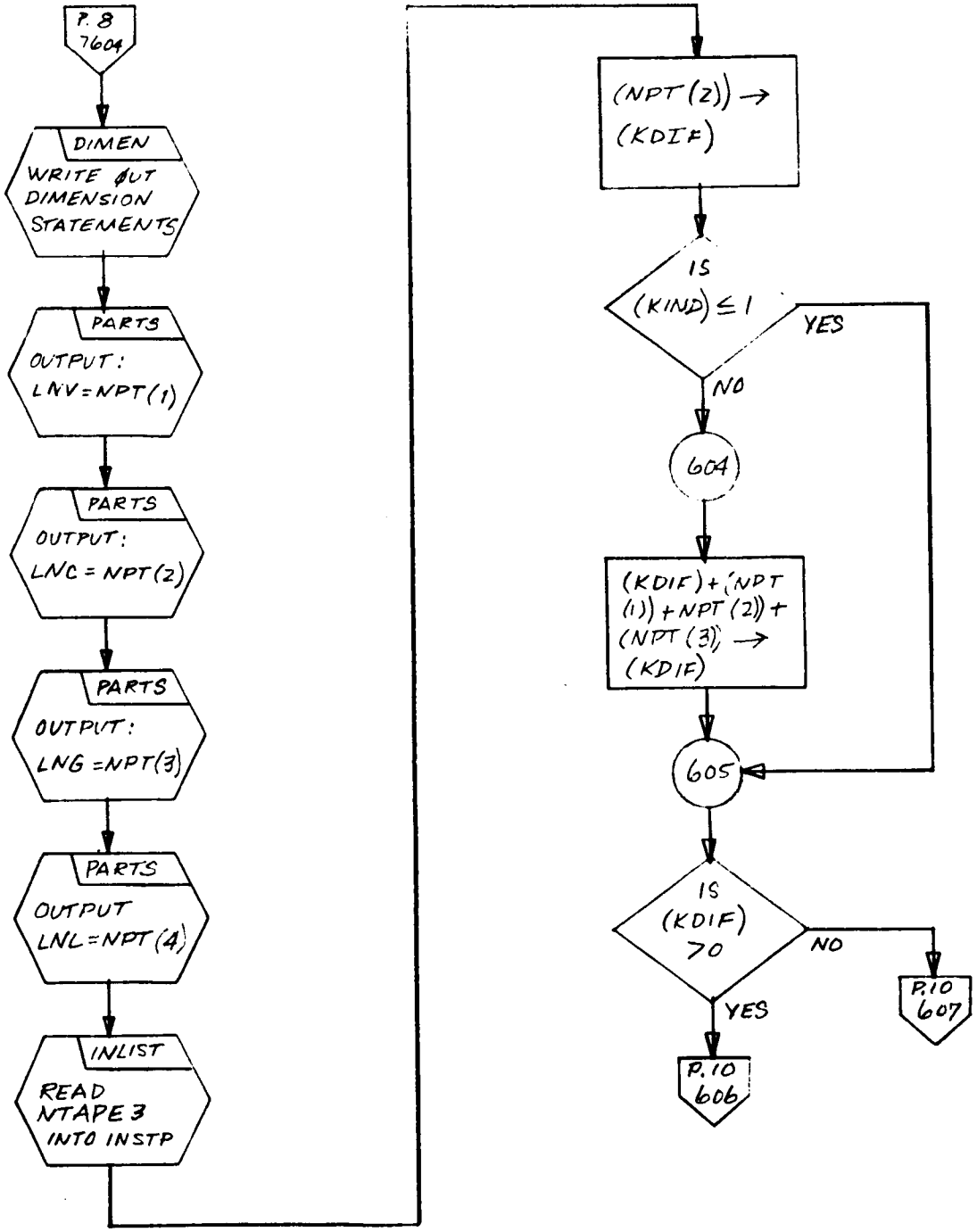

E
DØWN
PUSH
COMMA INTO
RIN
1 → (NCH)
(EQST) → (ELE2)
LINK
LINK (ELE2)
→ (ELE2)
(NCH) + 1
(NCH)
(ELE2)
(NULL)
?
NO
YES

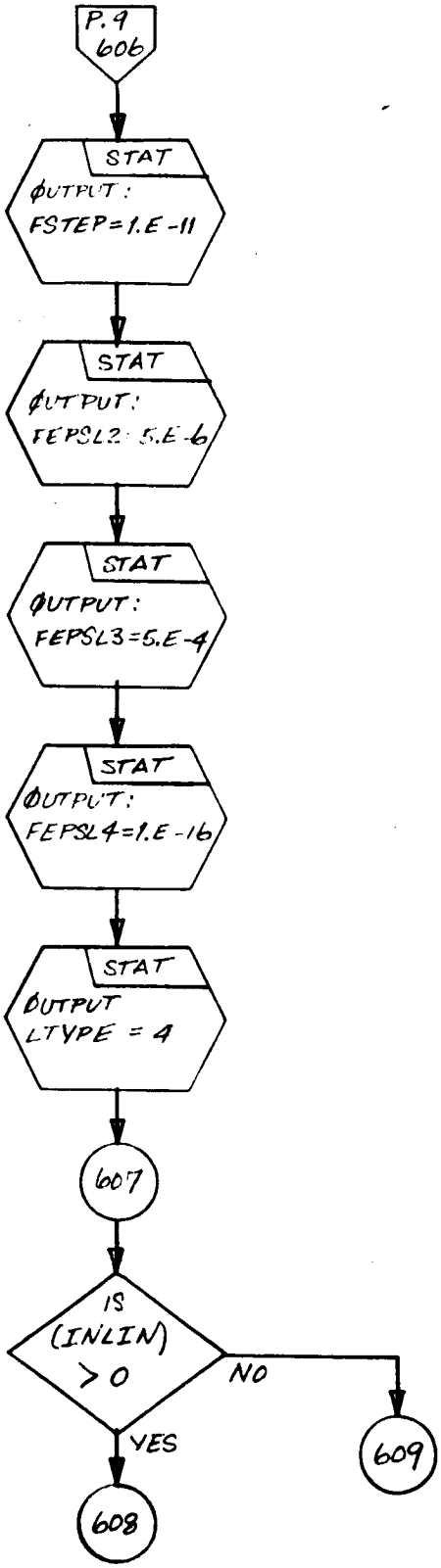

467
50 - (NCH) → (NCHL)
BCL
CONVERT NCHL TO BCD AND PUSH INTO RIN
DOWN
PUSH "X" INTO RIN
DOWN
PUSH COMMA INTO RIN
BCL
CONVERT NCH TO BCD AND PUSH INTO RIN
DOWN
PUSH "Y" INTO RIN
COPY
COPY EQST AND PUT AT HEAD OF RIN
F

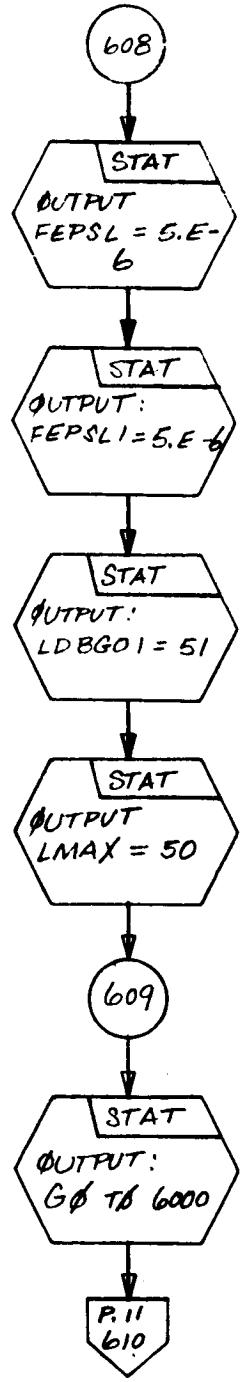

F
IS (ITEM X) AN I, J, K, L, M, OR N ?
YES
NO
480
468
DOWNS
PUSH (IFXED) INTO RIN
DOWNS
PUSH (IFLED) INTO RIN
490
PUSPCH
OUTPUT RIN ONTO INTAPE
P.3 400

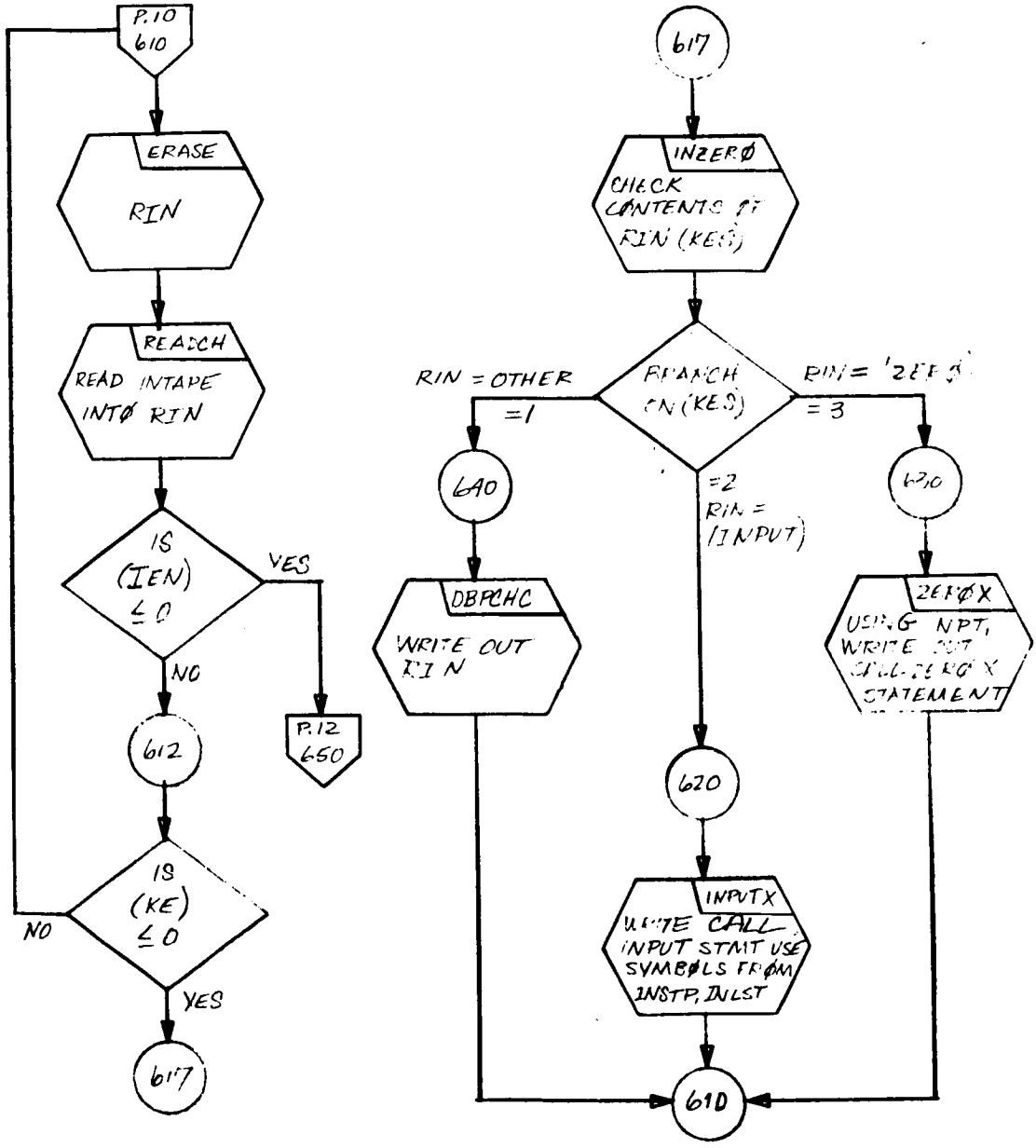

P.4
600
COPY
OUTPUT RSUT
ON TO INTAPE
ERASE
RIN
WRITE EØF
ON INTAPE;
REWIND INTAPE
IS
(ISTØP)
≤0
NO
YES
601
PRINT "THERE
WAS NØ
STATEMENT
NUMBER"
603

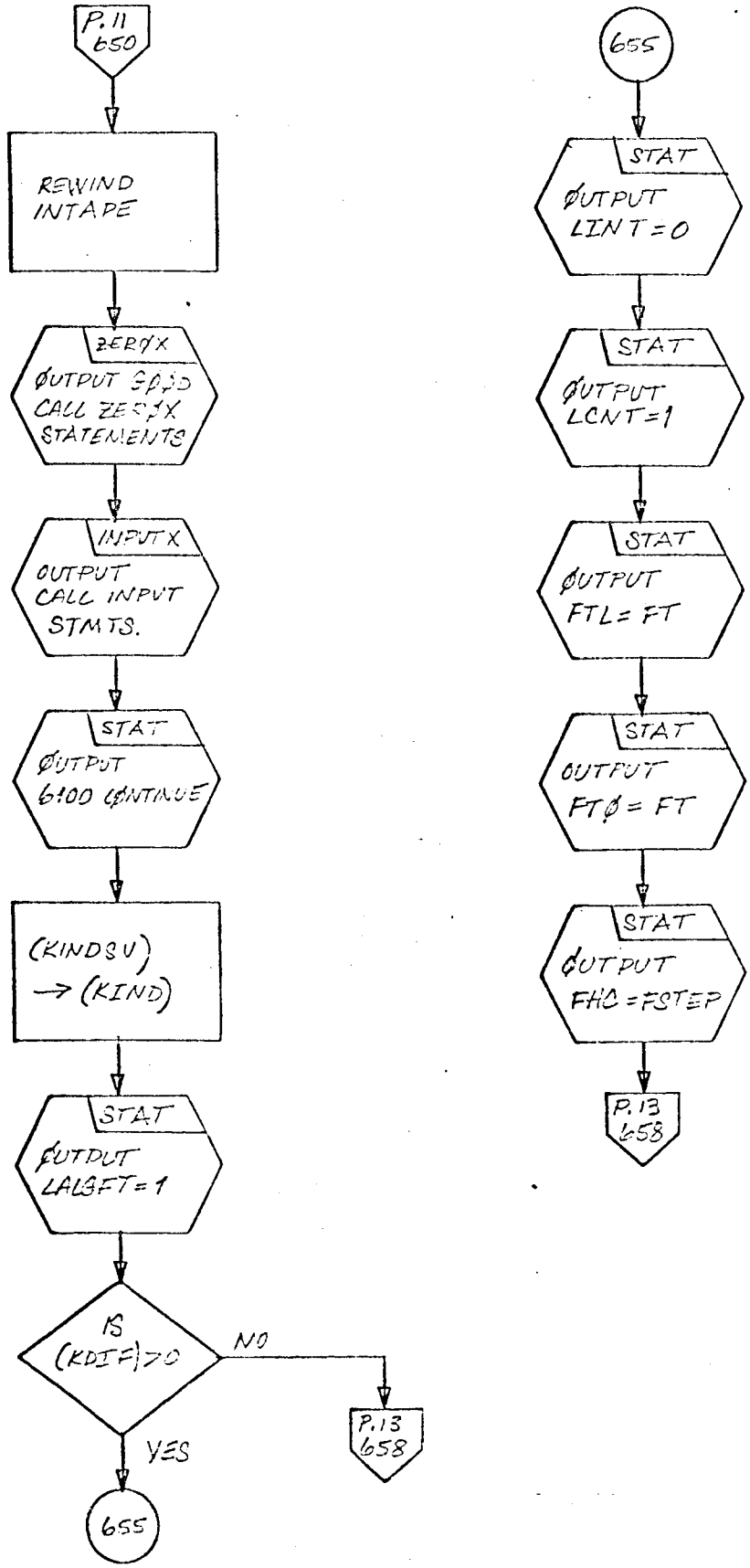

603
LEVMRK
READ NTAPE1
INTO XS; WRITE
XS ON NTAPE2
ERASEA
TC
(KIND) →
(KINDSV)
IS
(ISI3) ≥
2
NO
YES
7603
(ISIS) →
(KIND)
P.9
7604

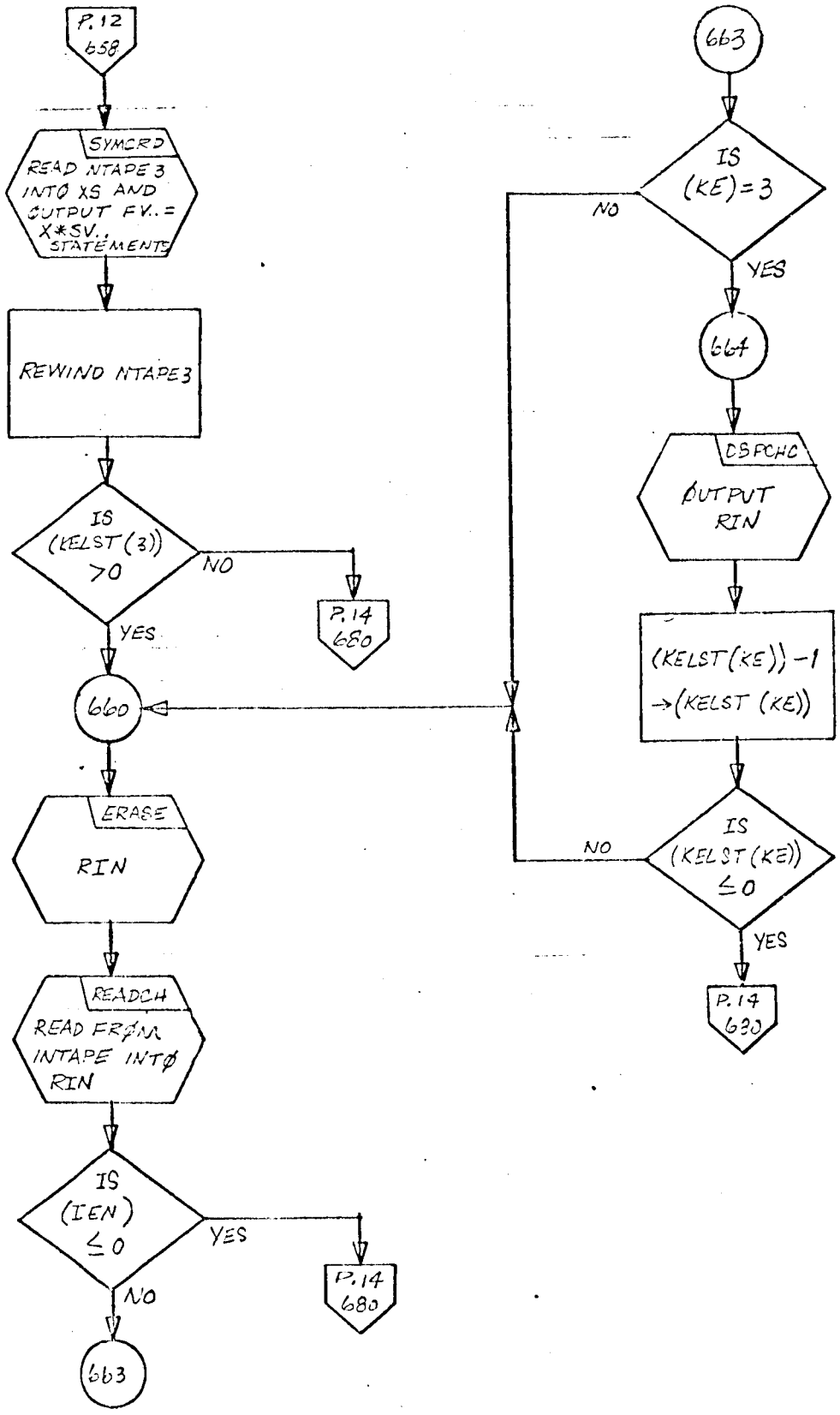

P. 8
7604
DIMEN
WRITE OUT
DIMENSION
STATEMENTS
PARTS
OUTPUT:
LNV = NPT(1)
PARTS
OUTPUT:
LNC = NPT(2)
PARTS
OUTPUT:
LNG = NPT(3)
PARTS
OUTPUT
LNL = NPT(4)
INLIST
READ
NTAPE 3
INTO INSTP
(NPT(2)) →
(KDIF)
IS
(KIND) ≤ 1
YES
NO
604
(KDIF) + (NPT
(1)) + NPT(2)) +
(NPT(3)) →
(KDIF)
605
IS
(KDIF)
70
NO
YES
P. 10
607
P. 10
606

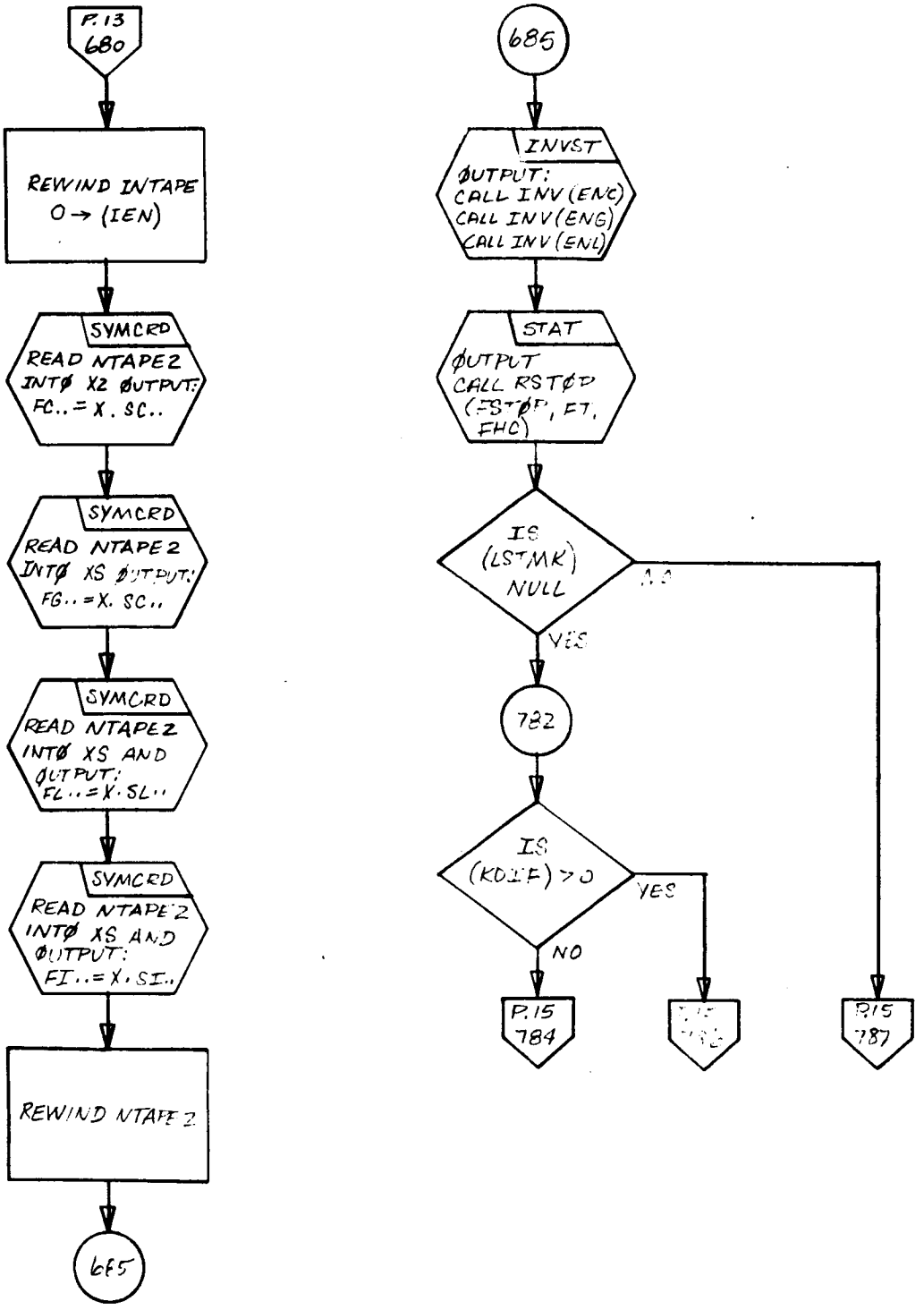

P.9
606
STAT
ØUTPUT:
FSTEP=1.E-11
STAT
ØUTPUT:
FEPSL2=5.E-6
STAT
ØUTPUT:
FEPSL3=5.E-4
STAT
ØUTPUT:
FEPSL4=1.E-16
STAT
ØUTPUT
LTYPE = 4
607
IS
(INLIN)
>0
NO
YES
609
608

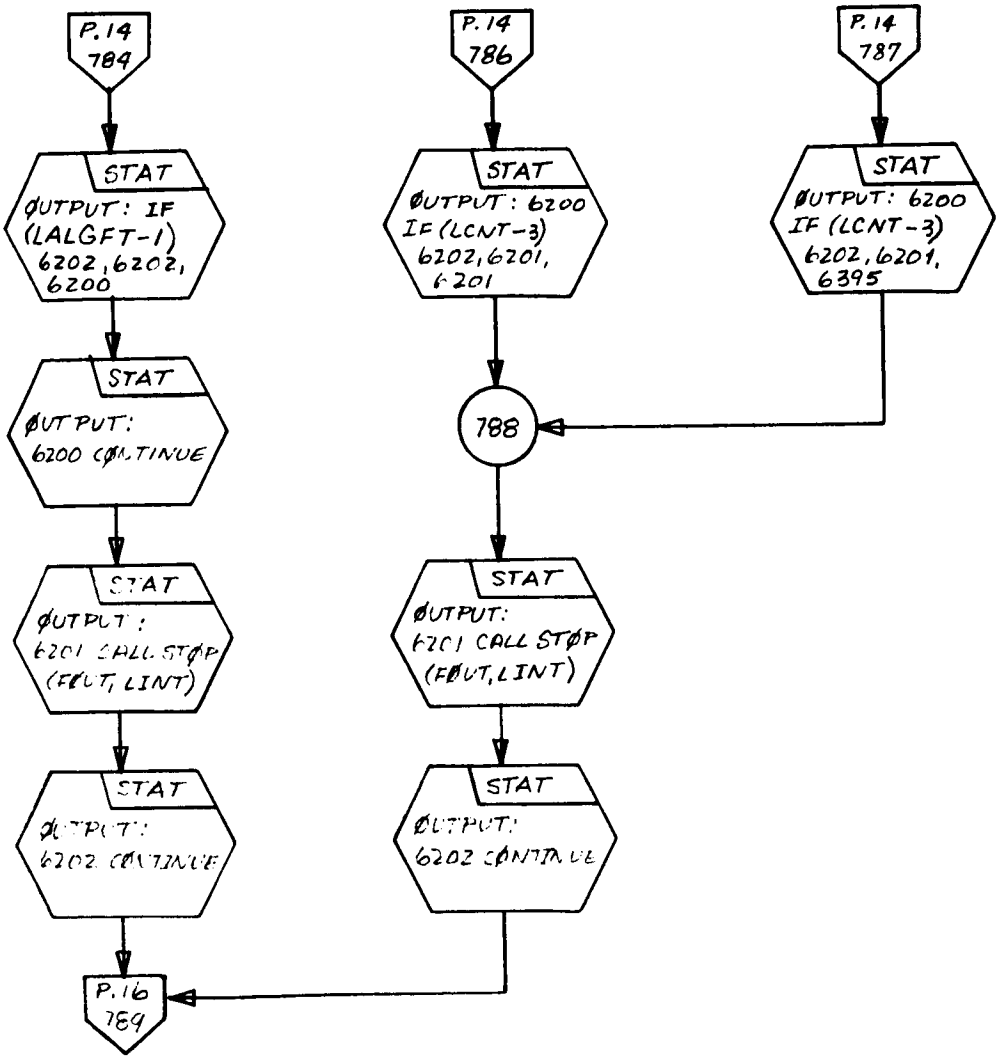

608
STAT
ØUTPUT
FEPSL = 5.E-6
STAT
ØUTPUT:
FEPSL1=5.E-6
STAT
ØUTPUT:
LDBGØ1 = 51
STAT
ØUTPUT
LMAX = 50
609
STAT
ØUTPUT:
GØ TØ 6000
P.11
610

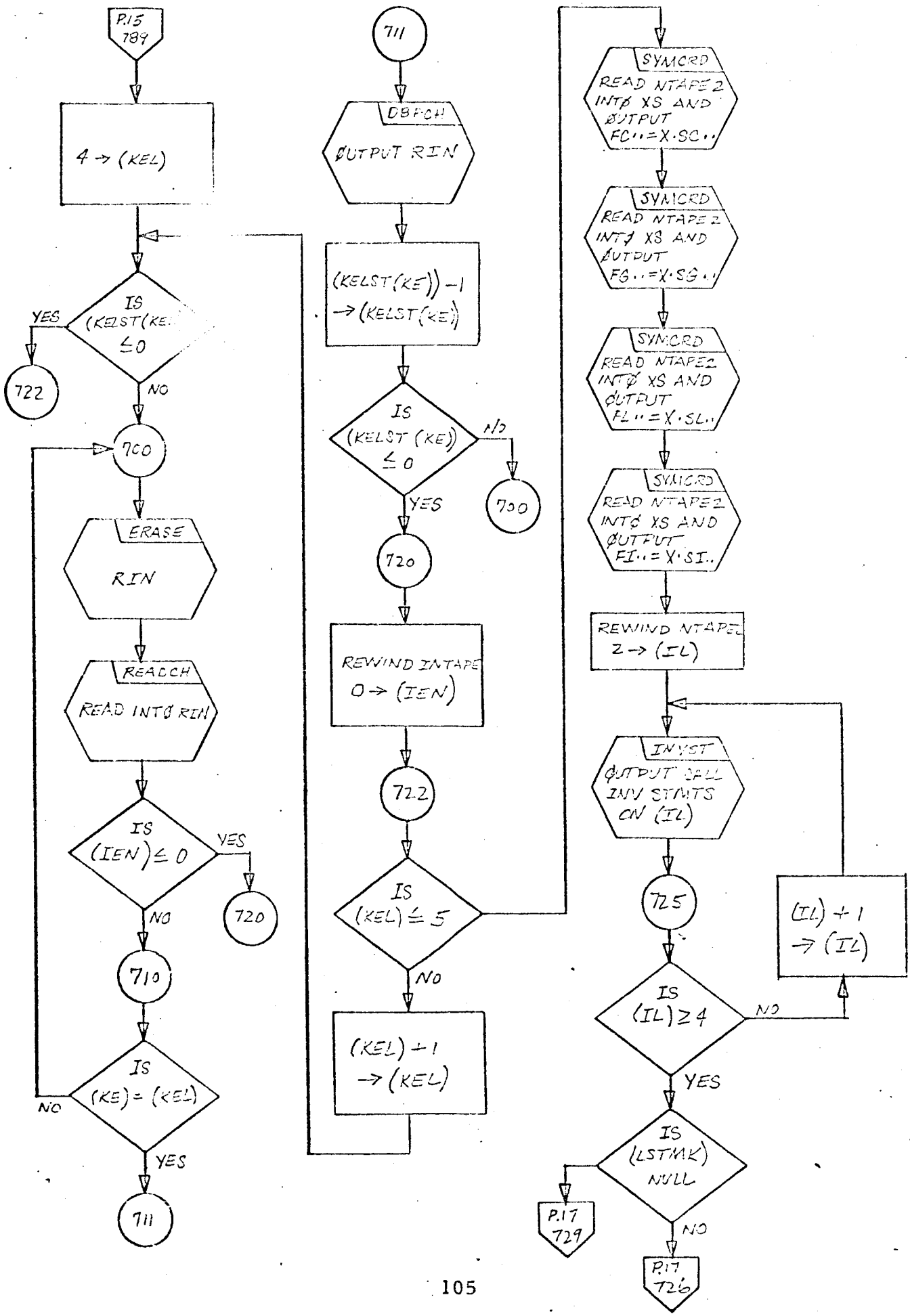

P.10
610
ERASE
RIN
READCH
READ INTAPE
INTØ RIN
IS
(IEN)
≤ 0
YES
NO
P.12
650
612
IS
(KE)
≤ 0
NO
YES
617
617
INZERØ
CHECK
CONTENTS OF
RIN (KEY)
BRANCH
ON (KEY)
RIN = OTHER
=1
=2
RIN =
'INPUT'
RIN = 'ZERØX'
=3
640
620
610
DBPCHC
WRITE OUT
RIN
INPUTX
WRITE CALL
INPUT STMT USE
SYMBØLS FRØM
INSTP, INLST
ZERØX
USING NPT,
WRITE OUT
CALL-PURGE X
STATEMENT
610

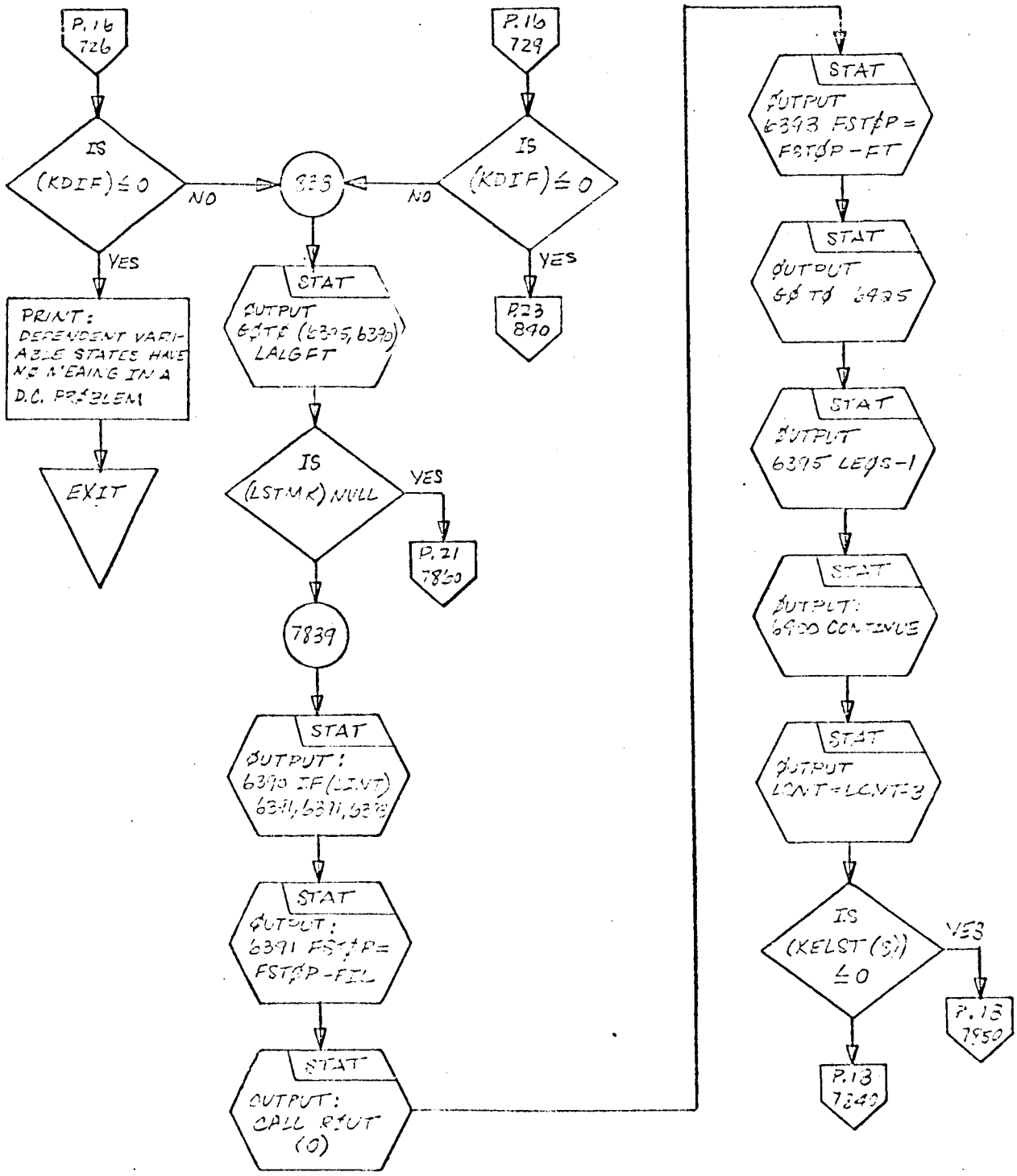

P. 11
650
REWIND
INTAPE
ZERØX
ØUTPUT GØTØ
CALL ZERØX
STATEMENTS
INPUTX
OUTPUT
CALL INPUT
STMTS.
STAT
ØUTPUT
6100 CØNTINUE
(KINDSV)
→ (KIND)
STAT
ØUTPUT
LALGFT=1
IS
(KDIF)>0
NO
YES
P. 13
658
655
655
STAT
ØUTPUT
LINT=0
STAT
ØUTPUT
LCNT=1
STAT
ØUTPUT
FTL=FT
STAT
OUTPUT
FTØ=FT
STAT
ØUTPUT
FHC=FSTEP
P. 13
658

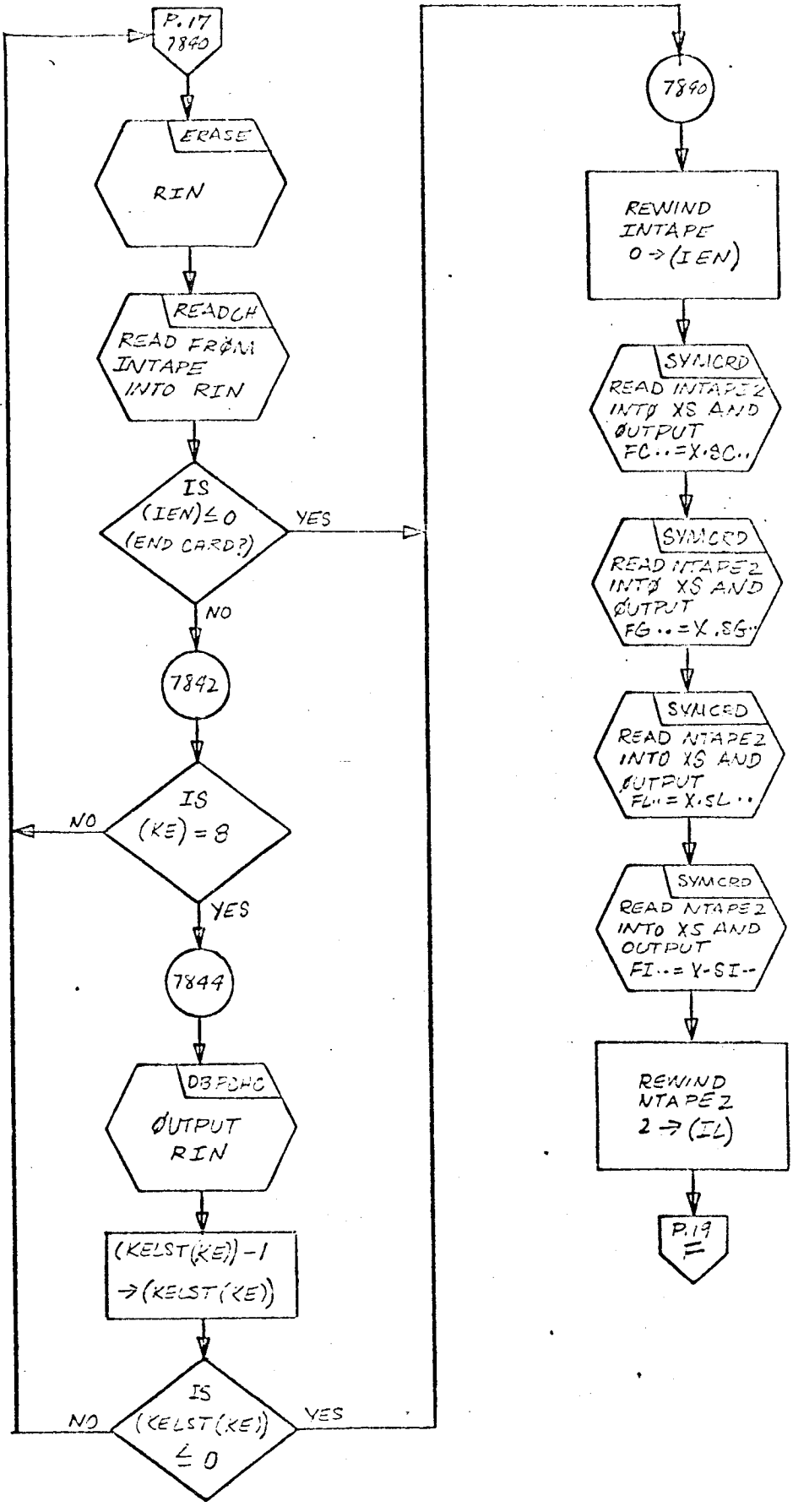

P.12
658
SYMCRD
READ NTAPE3 INTØ XS AND OUTPUT FV.. = X*SV.. STATEMENTS
REWIND NTAPE3
IS (KELST(3)) >0
NO
YES
P.14
680
660
ERASE
RIN
READCH
READ FRØM INTAPE INTØ RIN
IS (IEN) ≤0
YES
NO
P.14
680
663
663
IS (KE)=3
NO
YES
664
DSPCHC
OUTPUT RIN
(KELST(KE))-1 →(KELST(KE))
IS (KELST(KE)) ≤0
NO
YES
P.14
630

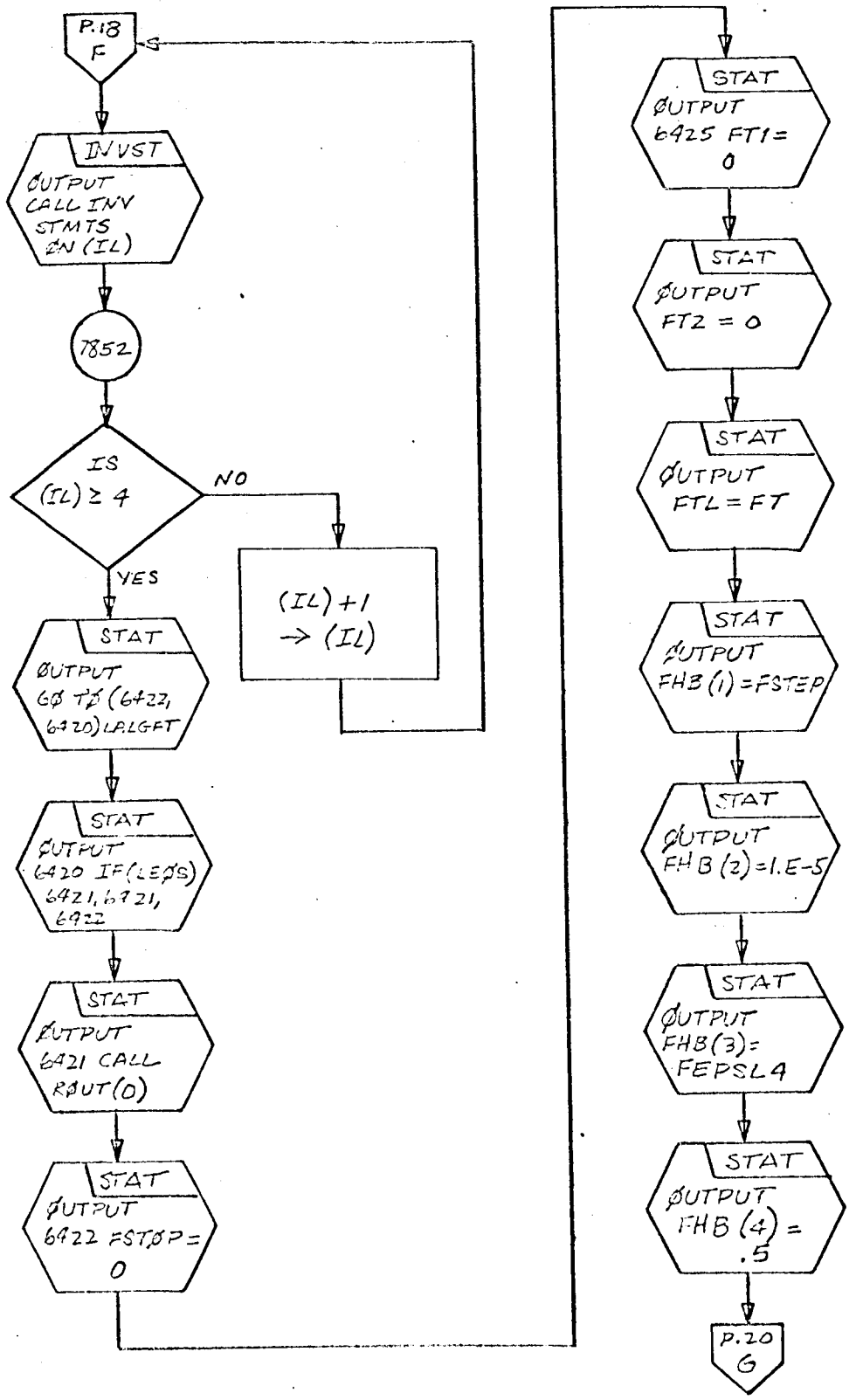

P.13
680
REWIND INTAPE
0 → (IEN)
SYMCRD
READ NTAPE2
INTØ X2 ØUTPUT:
FC.. = X. SC..
SYMCRD
READ NTAPE2
INTØ XS ØUTPUT:
FG.. = X. SC..
SYMCRD
READ NTAPE2
INTØ XS AND
ØUTPUT:
FL.. = X. SL..
SYMCRD
READ NTAPE2
INTØ XS AND
ØUTPUT:
FI.. = X. SI..
REWIND NTAPE 2
685
685
INVST
ØUTPUT:
CALL INV(ENC)
CALL INV(ENG)
CALL INV(ENL)
STAT
ØUTPUT
CALL RSTØP
(FSTØP, FT,
FHC)
IS
(LSTMK)
NULL
YES
782
IS
(KDIF) > 0
YES
NO
P.15
784
P.15
787

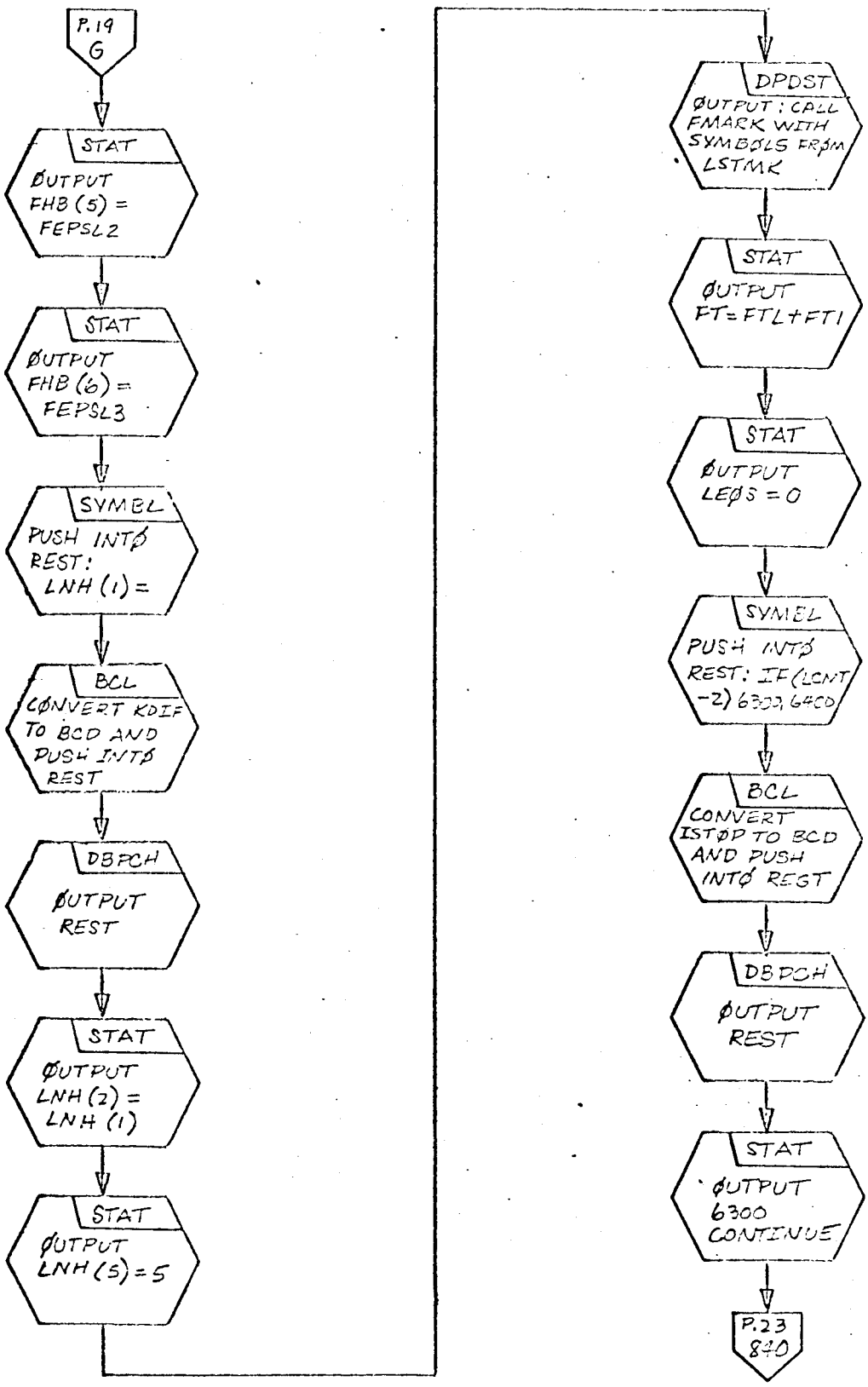

P.14
784
P.14
786
P.14
787
STAT
ØUTPUT: IF
(LALGFT-1)
6202,6202,
6200
STAT
ØUTPUT: 6200
IF (LCNT-3)
6202,6201,
6201
STAT
ØUTPUT: 6200
IF (LCNT-3)
6202,6201,
6395
STAT
ØUTPUT:
6200 CONTINUE
788
STAT
ØUTPUT:
6201 CALL STØP
(FØUT, LINT)
STAT
ØUTPUT:
6201 CALL STØP
(FØUT,LINT)
STAT
ØUTPUT:
6202 CONTINUE
STAT
ØUTPUT:
6202 CONTINUE
P.16
789

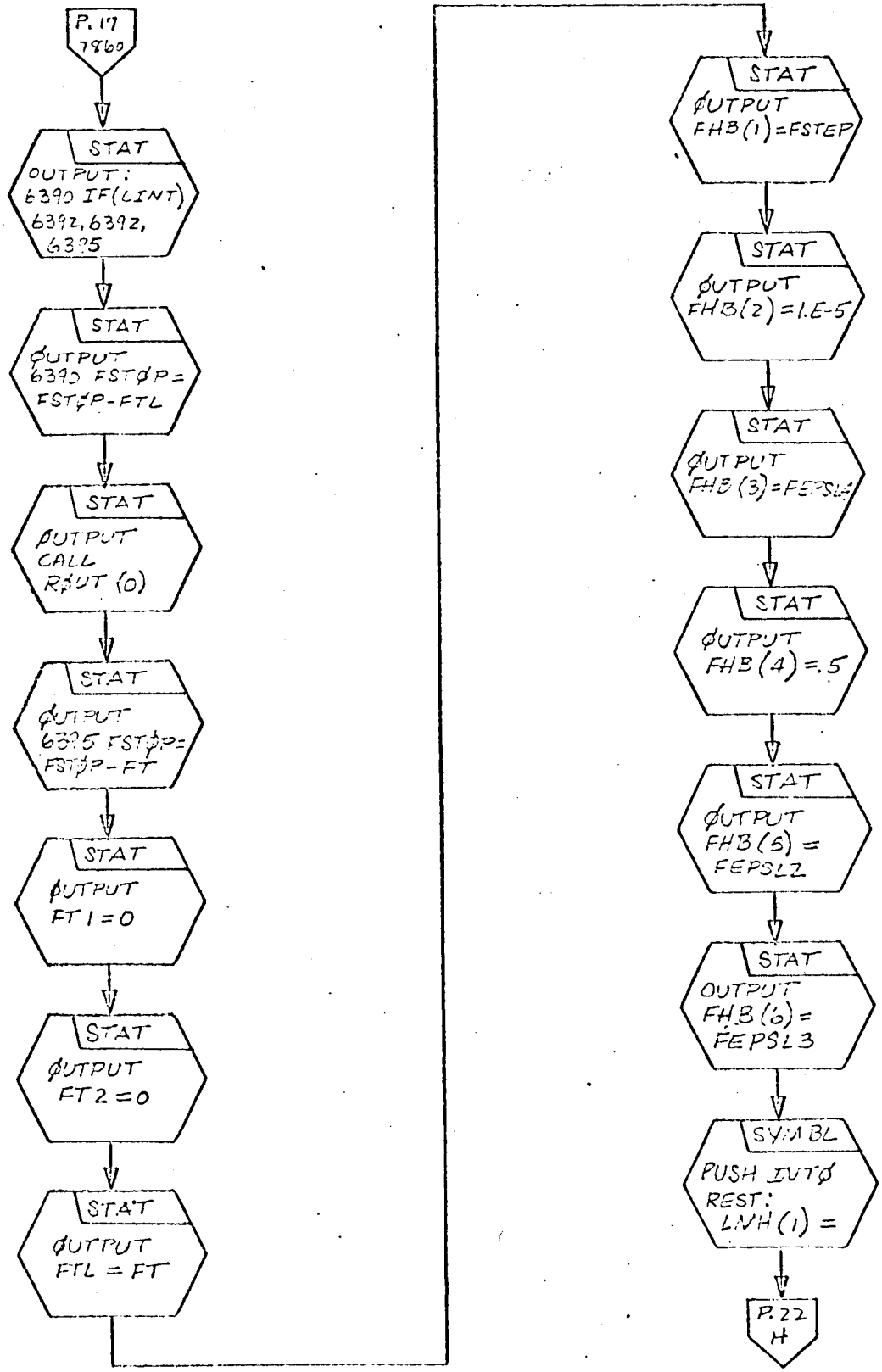

P.15
789
4 → (KEL)
IS
(KELST(KE
≤ 0
YES
NO
722
700
ERASE
RIN
READCH
READ INTØ RIN
IS
(IEN) ≤ 0
YES
NO
720
710
IS
(KE) = (KEL)
NO
YES
711
711
DBPCH
ØUTPUT RIN
(KELST(KE)) − 1
→ (KELST(KE))
IS
(KELST (KE))
≤ 0
NO
YES
700
720
REWIND INTAPE
0 → (IEN)
722
IS
(KEL) ≤ 5
NO
(KEL) + 1
→ (KEL)
SYMCRD
READ NTAPE 2
INTØ XS AND
ØUTPUT
FC..=X·SC..
SYMCRD
READ NTAPE 2
INTØ XS AND
ØUTPUT
FG..=X·SG..
SYMCRD
READ NTAPE 2
INTØ XS AND
ØUTPUT
FL..=X·SL..
SYMCRD
READ NTAPE 2
INTØ XS AND
ØUTPUT
FI..=X·SI..
REWIND NTAPE
2 → (IL)
INVST
ØUTPUT ALL
INV STMTS
ON (IL)
725
(IL) + 1
→ (IL)
IS
(IL) ≥ 4
NO
YES
IS
(LSTMK)
NULL
P.17
729
NO
P.17
726

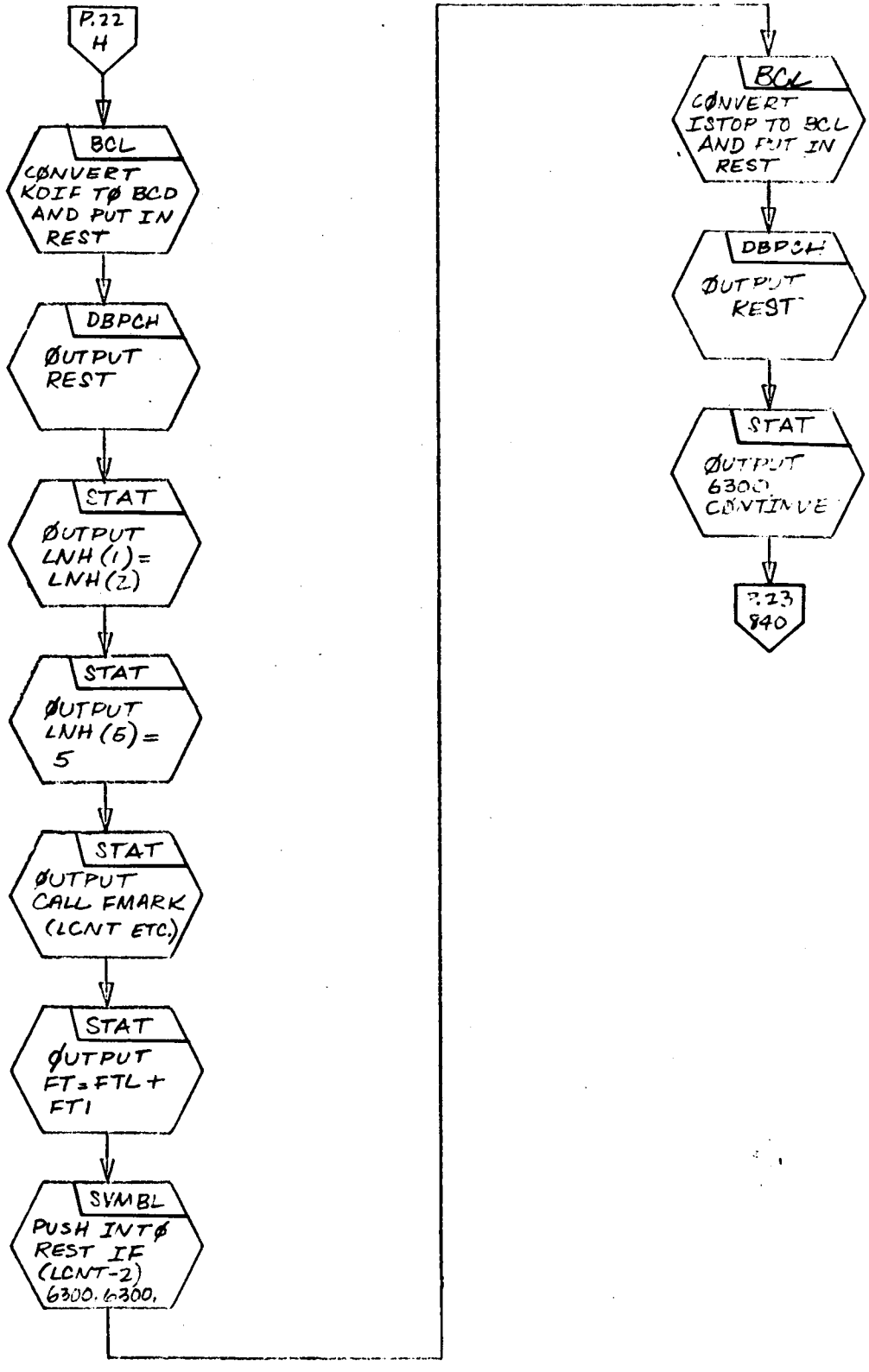

P.16
726
IS
(KDIF) ≤ 0
NO
YES
PRINT:
DEPENDENT VARIABLE STATES HAVE NO MEANING IN A D.C. PROBLEM
EXIT
333
P.16
729
IS
(KDIF) ≤ 0
NO
YES
P.23
890
STAT
OUTPUT
6ØTØ (6395, 6390)
LALGET
IS
(LSTMK) NULL
YES
P.21
7860
7839
STAT
ØUTPUT:
6390 IF(LINT)
6391, 6391, 6392
STAT
ØUTPUT:
6391 FSTØP=
FSTØP-FIL
STAT
OUTPUT:
CALL ØUT
(0)
STAT
ØUTPUT
6393 FSTØP=
FSTØP-FT
STAT
ØUTPUT
6Ø TØ 6925
STAT
ØUTPUT
6395 LEQS-1
STAT
ØUTPUT:
6900 CONTINUE
STAT
ØUTPUT
LCNT=LCNT+3
IS
(KELST(S))
≤ 0
YES
P.13
7950
P.13
7840

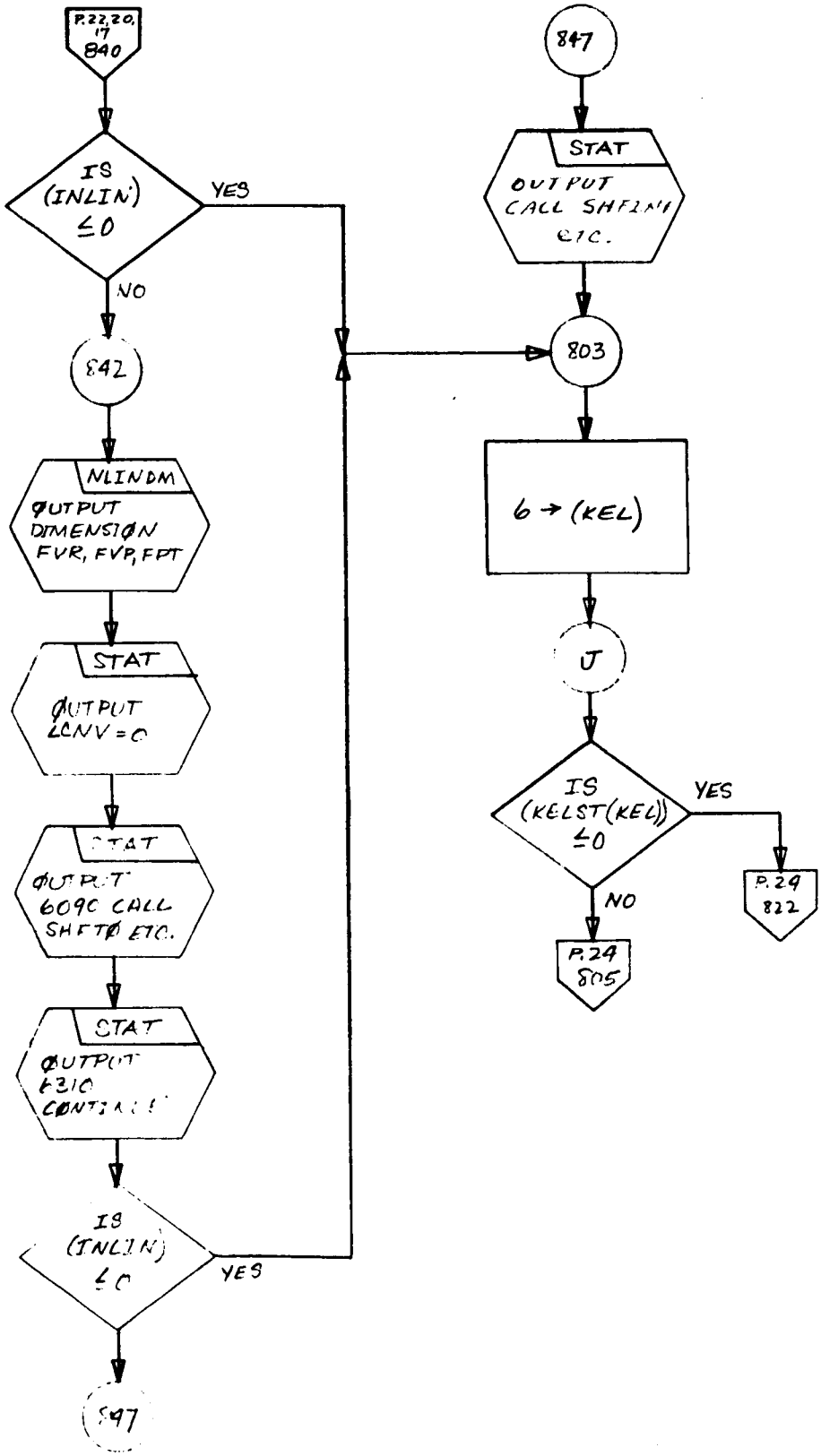

P.17
7840
ERASE
RIN
READCH
READ FROM
INTAPE
INTO RIN
IS
(IEN) ≤ 0
(END CARD?)
YES
NO
7842
IS
(KE) = 8
NO
YES
7844
DBFORC
OUTPUT
RIN
(KELST(KE)) - 1
→ (KELST(KE))
IS
(KELST(KE))
≤ 0
NO
YES
7840
REWIND
INTAPE
0 → (IEN)
SYMCRD
READ INTAPE2
INTO XS AND
OUTPUT
FC..=X.8C..
SYMCRD
READ NTAPE2
INTO XS AND
OUTPUT
FG..=X.8G..
SYMCRD
READ NTAPE2
INTO XS AND
OUTPUT
FL..=X.SL..
SYMCRD
READ NTAPE2
INTO XS AND
OUTPUT
FI..=X-SI--
REWIND
NTAPE2
2 → (IL)
P.19

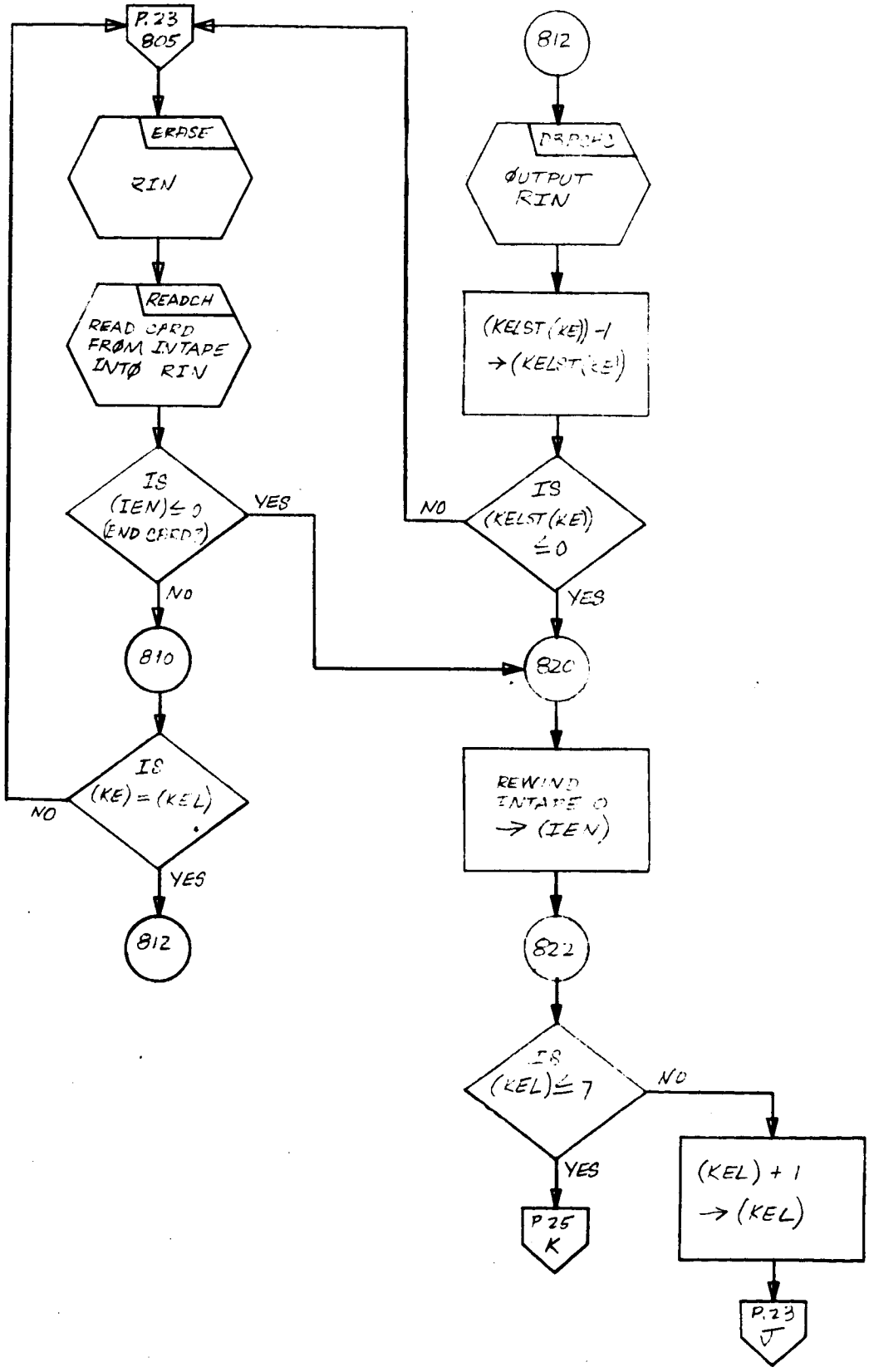

P.18
F
INVST
ØUTPUT
CALL INV
STMTS
ØN (IL)
7852
IS
(IL) ≥ 4
NO
YES
(IL) + 1
→ (IL)
STAT
ØUTPUT
GØ TØ (6422,
6420) LALGFT
STAT
ØUTPUT
6420 IF(LEØS)
6421, 6421,
6422
STAT
ØUTPUT
6421 CALL
RØUT(0)
STAT
ØUTPUT
6422 FSTØP=
0
STAT
ØUTPUT
6425 FT1=
0
STAT
ØUTPUT
FT2 = 0
STAT
ØUTPUT
FTL = FT
STAT
ØUTPUT
FHB(1) = FSTEP
STAT
ØUTPUT
FHB(2) = 1.E-5
STAT
ØUTPUT
FHB(3) =
FEPSL4
STAT
ØUTPUT
FHB(4) =
.5
P.20
G

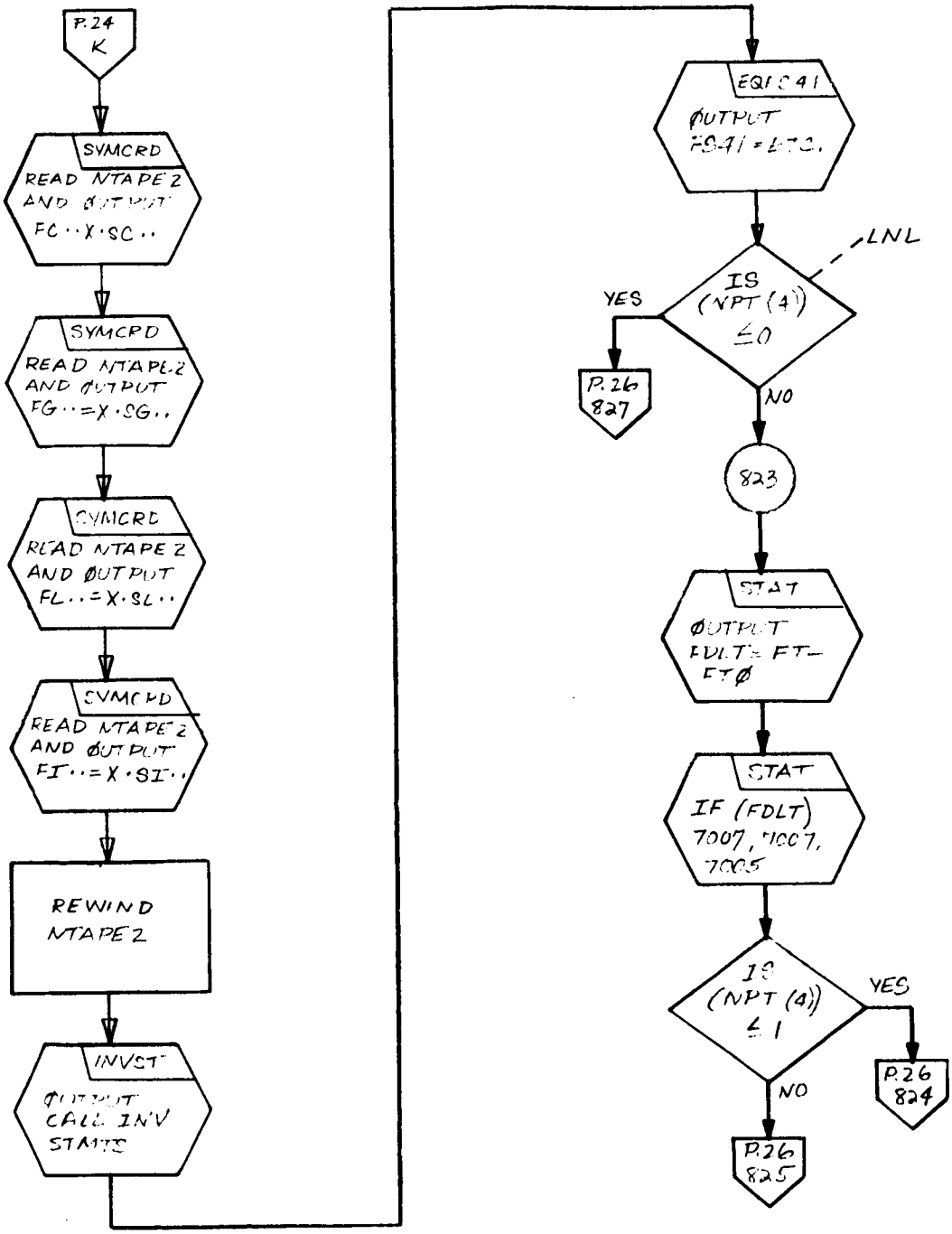

P.19
G
STAT
ØUTPUT
FHB(5) =
FEPSL2
STAT
ØUTPUT
FHB(6) =
FEPSL3
SYMBL
PUSH INTØ
REST:
LNH(1) =
BCL
CØNVERT KDIF
TO BCD AND
PUSH INTØ
REST
DBPCH
ØUTPUT
REST
STAT
ØUTPUT
LNH(2) =
LNH(1)
STAT
ØUTPUT
LNH(5) = 5
DPDST
ØUTPUT: CALL
FMARK WITH
SYMBØLS FRØM
LSTMK
STAT
ØUTPUT
FT = FTL + FTI
STAT
ØUTPUT
LEØS = 0
SYMBL
PUSH INTØ
REST: IF(LCNT
-2) 6300, 6400
BCL
CONVERT
ISTØP TO BCD
AND PUSH
INTØ REGT
DBPCH
ØUTPUT
REST
STAT
ØUTPUT
6300
CONTINUE
P.23
840

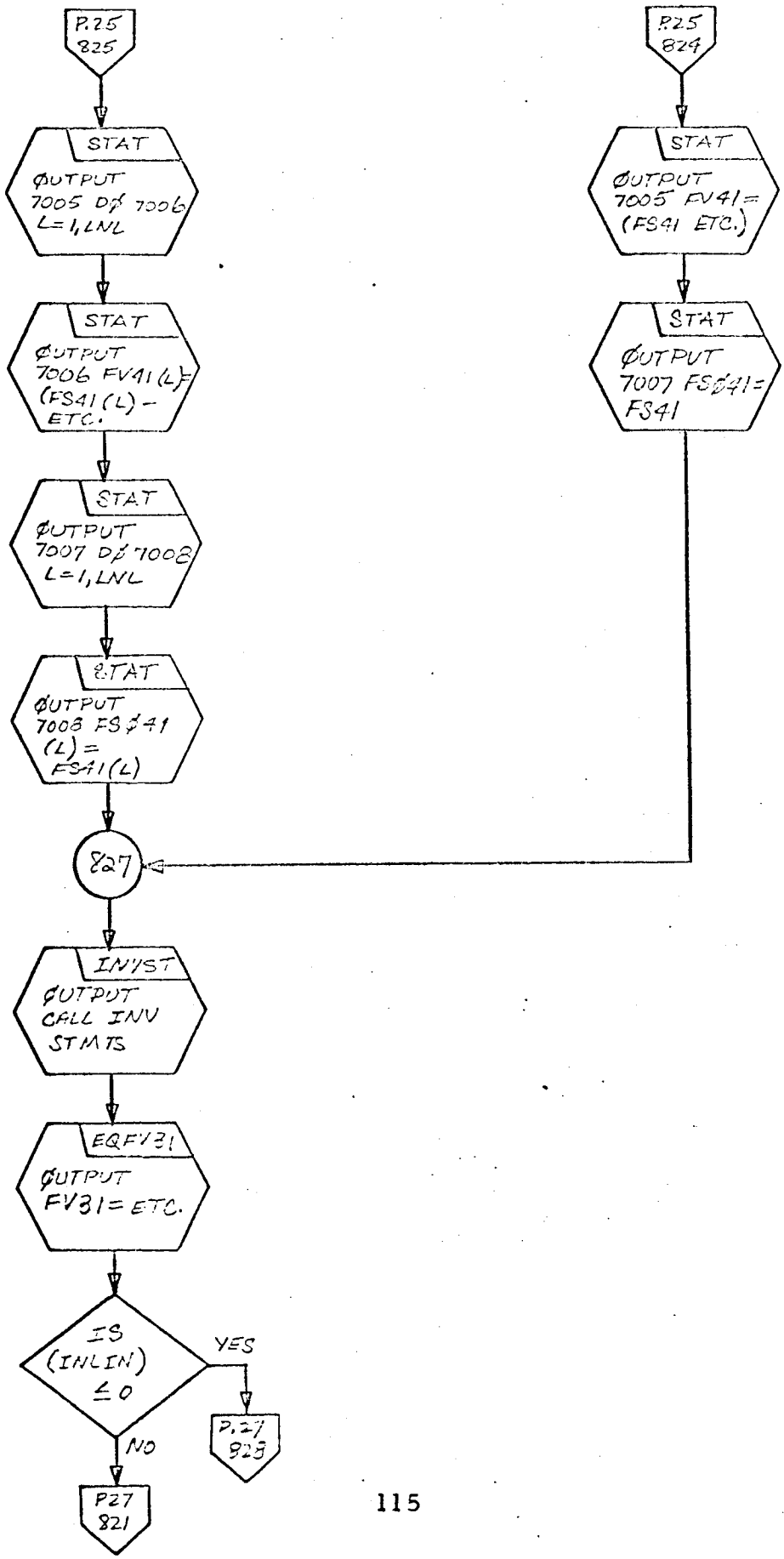

P. 17
7860
STAT
OUTPUT:
6390 IF(LINT)
6392, 6392,
6395
STAT
ØUTPUT
6390 FSTØP=
FSTØP-FTL
STAT
ØUTPUT
CALL
RAUT (0)
STAT
ØUTPUT
6395 FSTØP=
FSTØP-FT
STAT
ØUTPUT
FT1=0
STAT
ØUTPUT
FT2=0
STAT
ØUTPUT
FTL = FT
STAT
ØUTPUT
FHB(1)=FSTEP
STAT
ØUTPUT
FHB(2)=1.E-5
STAT
ØUTPUT
FHB(3)=FEPSL4
STAT
ØUTPUT
FHB(4)=.5
STAT
ØUTPUT
FHB(5)=
FEPSL2
STAT
OUTPUT
FHB(6)=
FEPSL3
SYMBL
PUSH IUTØ
REST:
LNH(1) =
P. 22
H

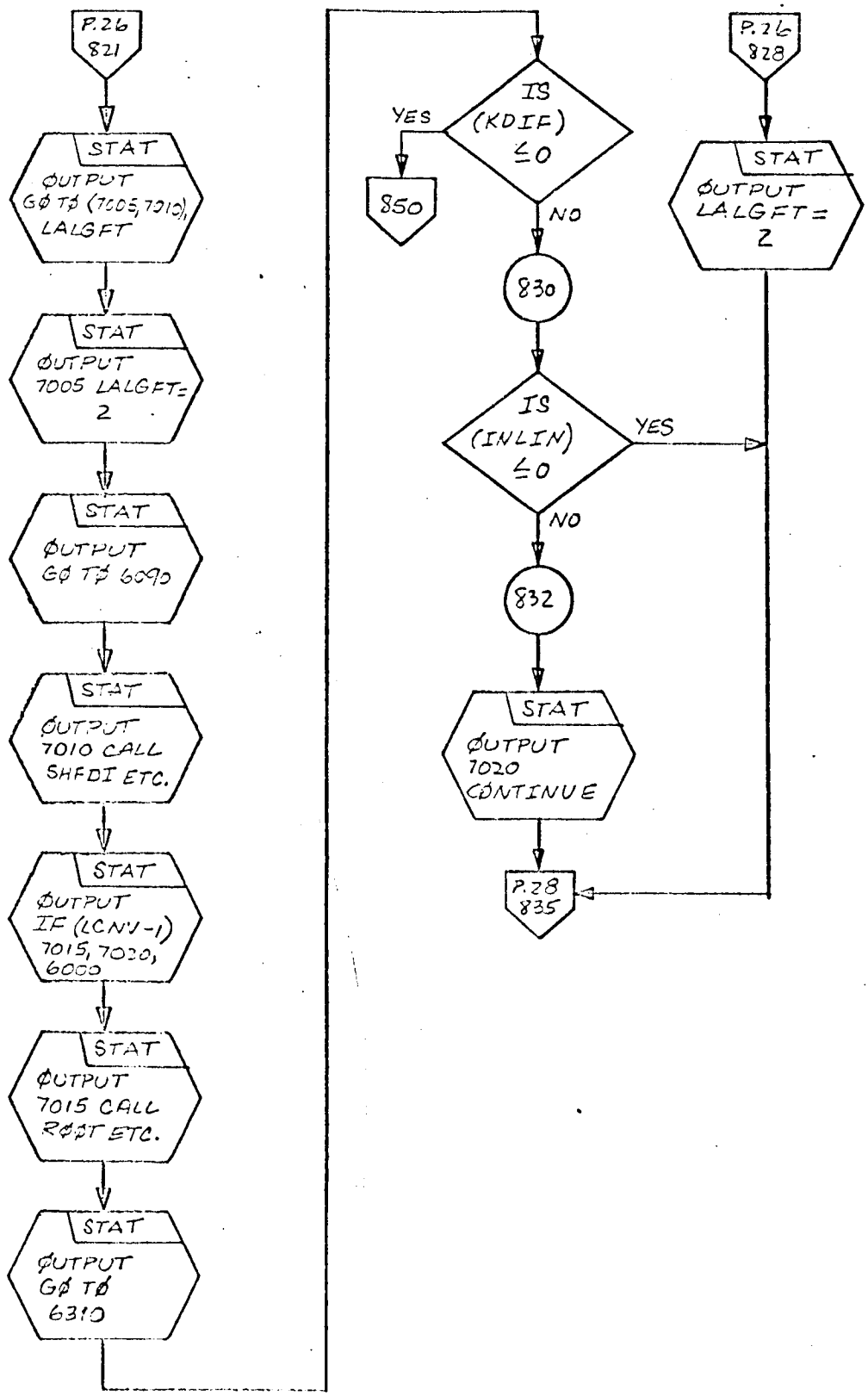

P.22
H
BCL
CONVERT KDIF TO BCD AND PUT IN REST
DBPCH
OUTPUT REST
STAT
OUTPUT LNH(1)= LNH(2)
STAT
OUTPUT LNH(6)= 5
STAT
OUTPUT CALL FMARK (LCNT ETC.)
STAT
OUTPUT FT=FTL+ FTI
SYMBL
PUSH INTO REST IF (LCNT-2) 6300, 6300,
BCL
CONVERT ISTOP TO BCL AND PUT IN REST
DBPCH
OUTPUT REST
STAT
OUTPUT 6300 CONTINUE
P.23
840

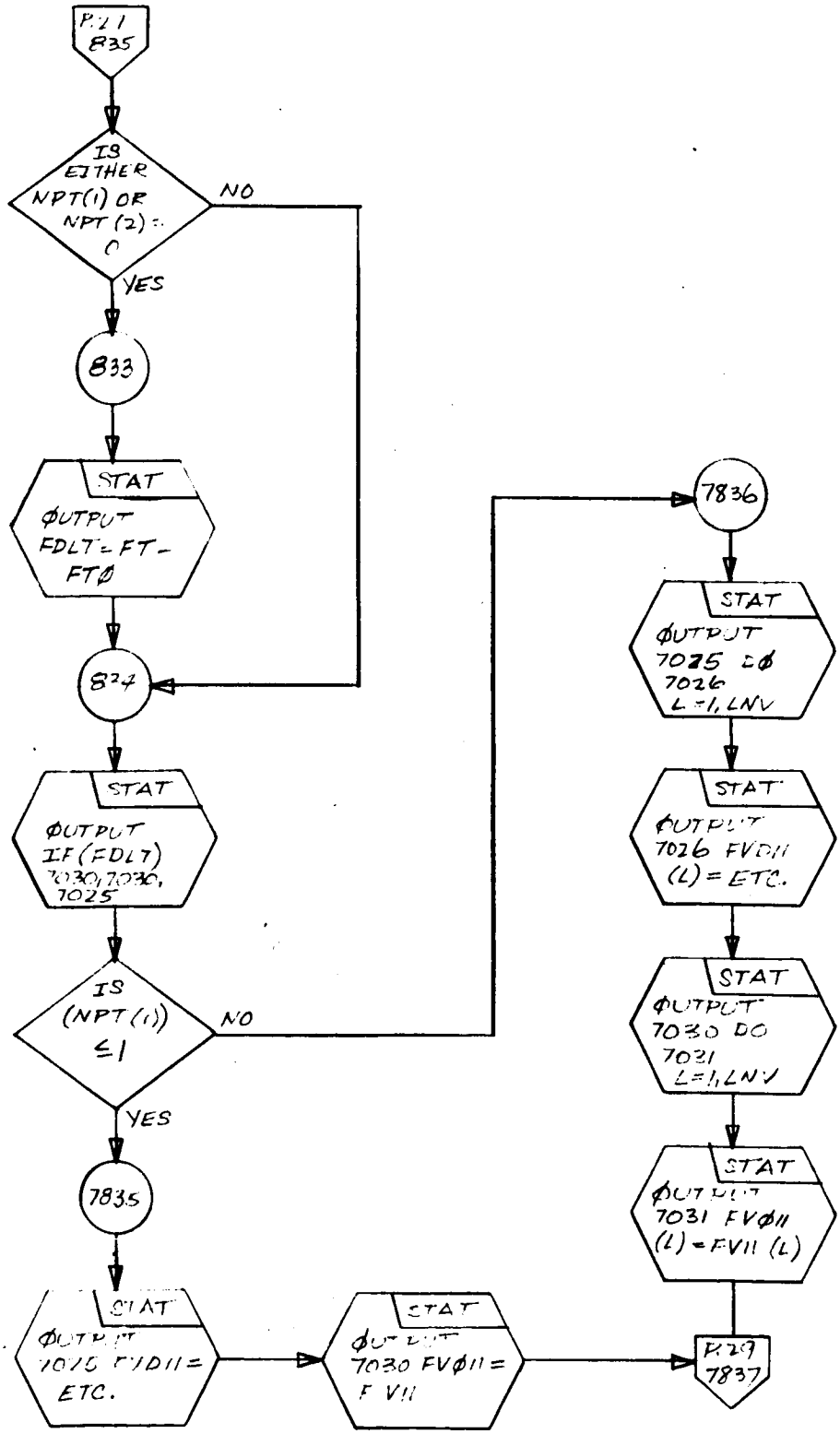

P.22,20, 17 840
IS (INLIN) ≤0
YES
NO
842
NLINDM
OUTPUT DIMENSION FVR, FVP, FPT
STAT
OUTPUT LCNV = C
STAT
OUTPUT 609C CALL SHFTØ ETC.
STAT
OUTPUT 630 CONTINUE
IS (INLIN) ≤ C
YES
897
847
STAT
OUTPUT CALL SHFINI ETC.
803
6 → (KEL)
J
IS (KELST(KEL)) ≤0
YES
NO
P.24 822
P.24 805

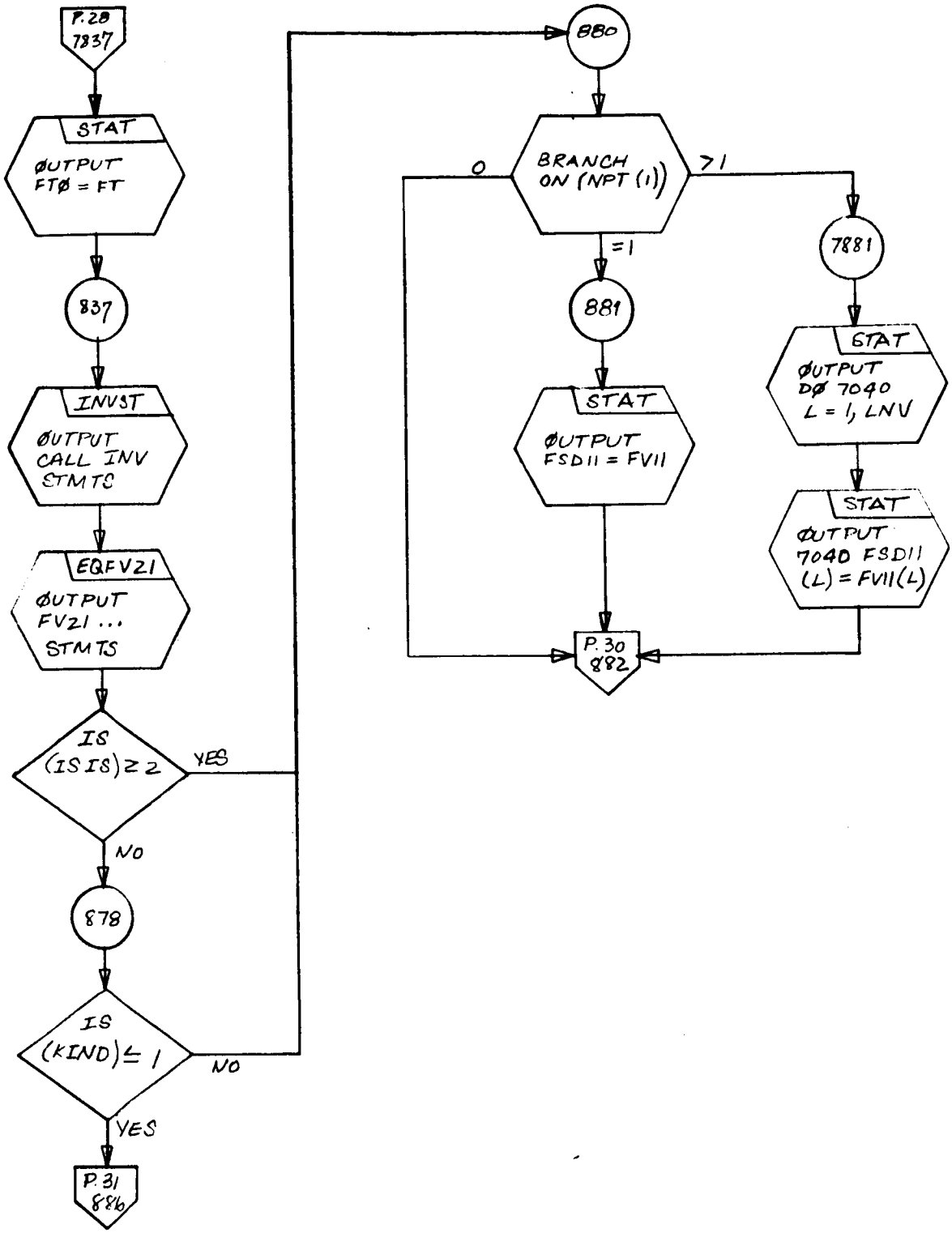

P.23
805
ERASE
RIN
READCH
READ CARD
FROM INTAPE
INTO RIN
IS
(IEN) ≤ 0
(END CARD?)
YES
NO
810
IS
(KE) = (KEL)
NO
YES
812
812
OUTPUT
RIN
(KELST(KE)) - 1
→ (KELST(KE))
IS
(KELST(KE))
≤ 0
NO
YES
820
REWIND
INTAPE 0
→ (IEN)
822
IS
(KEL) ≤ 7
NO
YES
P.25
K
(KEL) + 1
→ (KEL)
P.23
J

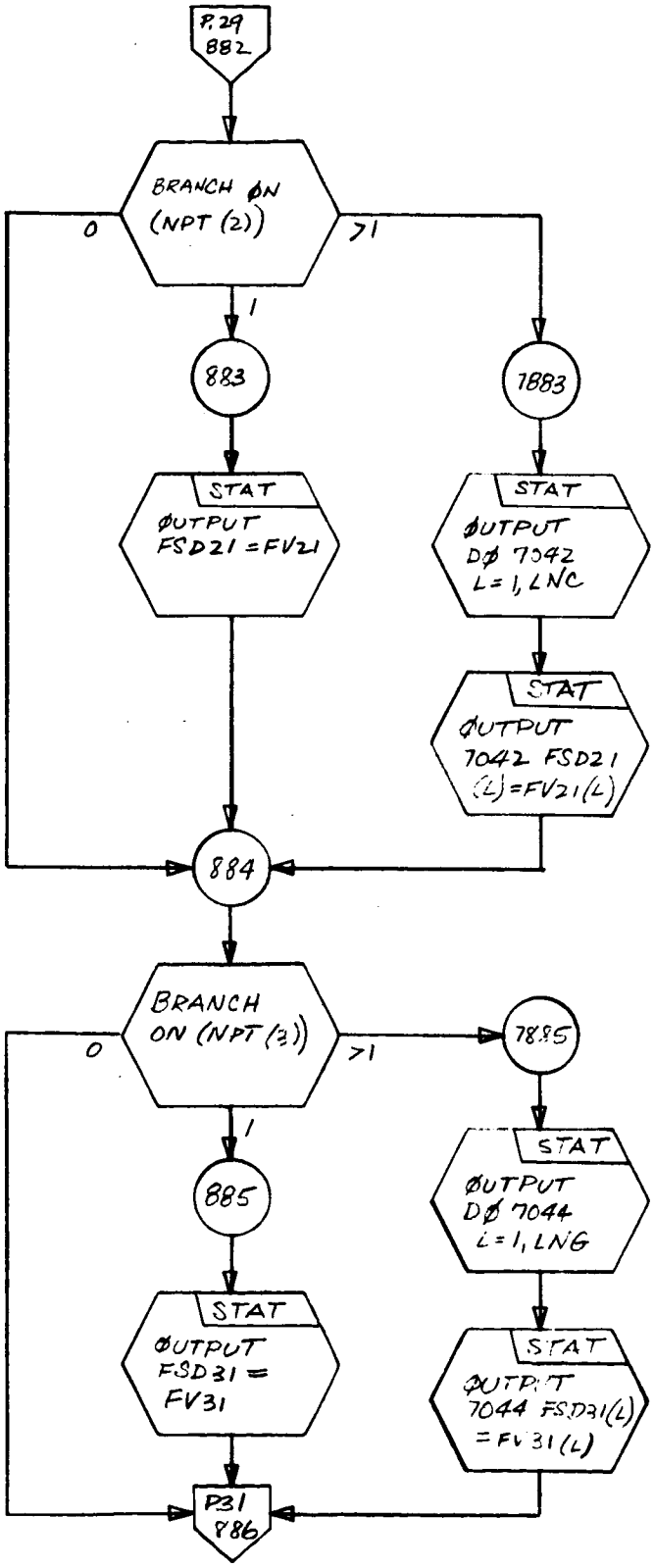

P.24
K
SYMCRD
READ NTAPE 2
AND OUTPUT
FC··X·SC··
SYMCRD
READ NTAPE 2
AND OUTPUT
FG··=X·SG··
SYMCRD
READ NTAPE 2
AND OUTPUT
FL··=X·SL··
SYMCRD
READ NTAPE 2
AND OUTPUT
FI··=X·SI··
REWIND
NTAPE 2
INVST
OUTPUT
CALL INV
STATS
EQUIS 41
OUTPUT
FS41 = ETC.
LNL
IS
(NPT (4))
≤0
YES
P.26
827
NO
823
STAT
OUTPUT
FDLT = FT−
FTØ
STAT
IF (FDLT)
7007, 7007,
7005
IS
(NPT (4))
≤1
YES
P26
824
NO
P26
825

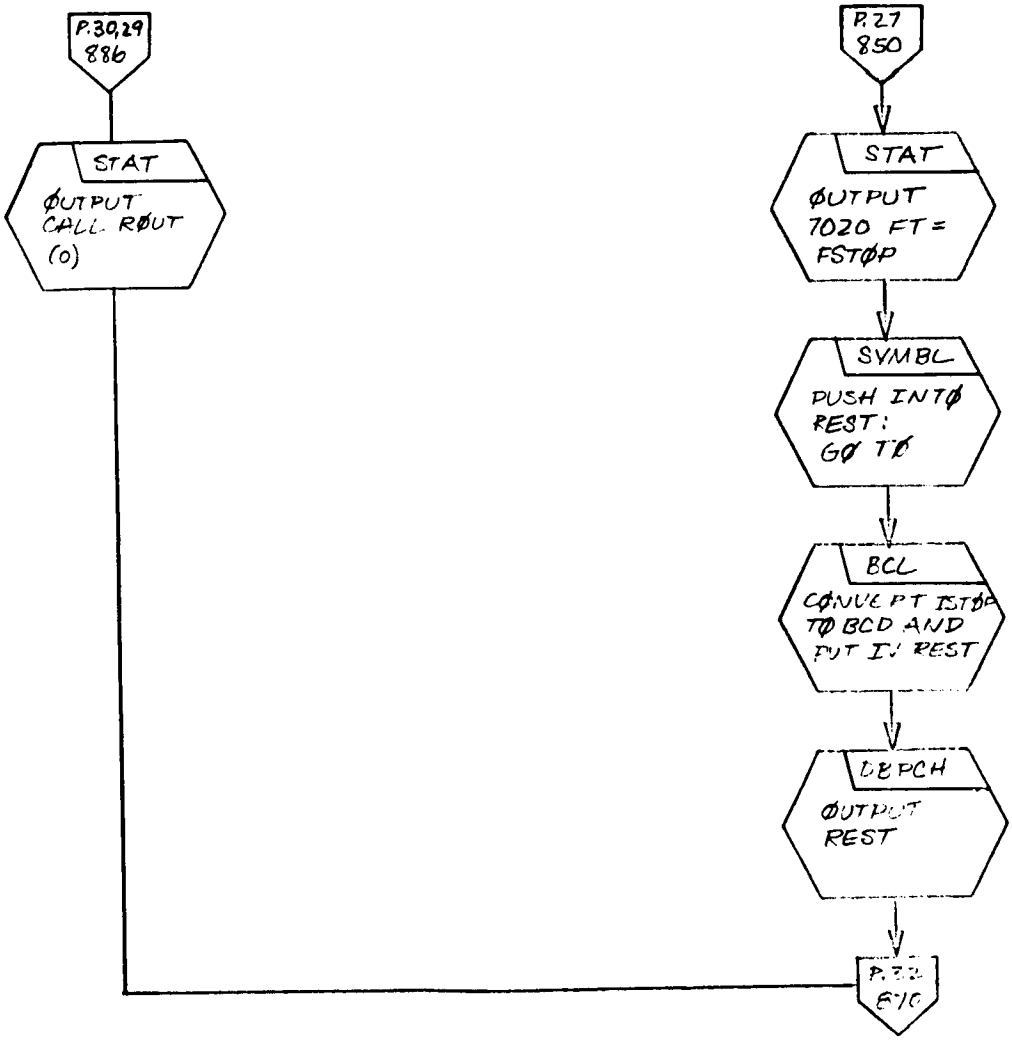

P.25
825
STAT
ØUTPUT
7005 DØ 7006
L=1,LNL
STAT
ØUTPUT
7006 FV41(L)=
(FS41(L)-
ETC.
STAT
ØUTPUT
7007 DØ 7008
L=1,LNL
STAT
ØUTPUT
7008 FSØ41
(L)=
FS41(L)
827
P.25
824
STAT
ØUTPUT
7005 FV41=
(FS41 ETC.)
STAT
ØUTPUT
7007 FSØ41=
FS41
INVST
ØUTPUT
CALL INV
STMTS
EQFV31
ØUTPUT
FV31= ETC.
IS
(INLIN)
≤0
YES
NO
P.27
923
P27
821

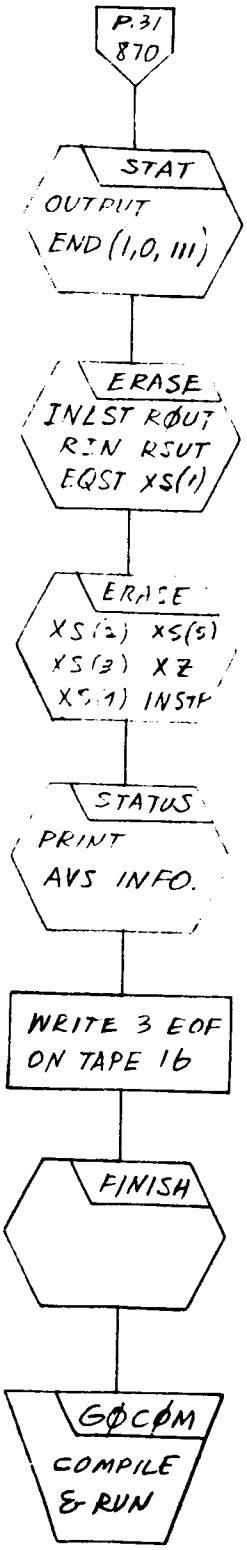

P.26
821
STAT
ØUTPUT
GØ TØ (7005, 7010),
LALGFT
STAT
ØUTPUT
7005 LALGFT=
2
STAT
ØUTPUT
GØ TØ 6090
STAT
ØUTPUT
7010 CALL
SHFDI ETC.
STAT
ØUTPUT
IF (LCNV-1)
7015, 7020,
6000
STAT
ØUTPUT
7015 CALL
RØØT ETC.
STAT
ØUTPUT
GØ TØ
6310
IS
(KDIF)
≤0
YES
850
NO
830
IS
(INLIN)
≤0
YES
NO
832
STAT
ØUTPUT
7020
CONTINUE
P.28
835
P.26
828
STAT
ØUTPUT
LALGFT=
2


P.21
835
IS EITHER NPT(1) OR NPT(2) = 0
NO
YES
833
STAT
ØUTPUT FDLT=FT-FTØ
824
STAT
ØUTPUT IF(FDLT) 7030,7030, 7025
IS (NPT(1)) ≤1
NO
YES
7835
STAT
ØUTPUT 7025 FVDII= ETC.
STAT
ØUTPUT 7030 FVØII= FVII
7836
STAT
ØUTPUT 7025 DØ 7026 L=1,LNV
STAT
ØUTPUT 7026 FVDII (L)=ETC.
STAT
ØUTPUT 7030 DO 7031 L=1,LNV
STAT
ØUTPUT 7031 FVØII (L)=FVII (L)
P.29
7837


P.28
7837
STAT
ØUTPUT
FTØ = FT
837
INVST
ØUTPUT
CALL INV
STMTS
EQFV21
ØUTPUT
FV21 ...
STMTS
IS
(ISIS) ≥ 2
YES
NO
878
IS
(KIND) ≤ 1
NO
YES
P.31
886
880
BRANCH
ON (NPT (1))
0
>1
=1
881
STAT
ØUTPUT
FSD11 = FV11
7881
STAT
ØUTPUT
DØ 7040
L = 1, LNV
STAT
ØUTPUT
7040 FSD11
(L) = FV11(L)
P.30
882


P.29
882
BRANCH ON
(NPT (2))
0
>1
1
883
1883
STAT
ØUTPUT
FSD21 = FV21
STAT
ØUTPUT
DØ 7042
L = 1, LNC
STAT
ØUTPUT
7042 FSD21
(L) = FV21(L)
884
BRANCH
ON (NPT (3))
0
>1
1885
STAT
ØUTPUT
DØ 7044
L = 1, LNG
885
STAT
ØUTPUT
FSD31 =
FV31
STAT
ØUTPUT
7044 FSD31(L)
= FV31(L)
P31
886


P.30,29
886
STAT
ØUTPUT
CALL RØUT
(0)
P.27
850
STAT
ØUTPUT
7020 FT =
FSTØP
SYMBL
PUSH INTØ
REST:
GØ TØ
BCL
CØNVERT ISTØP
TØ BCD AND
PUT IN REST
DSPCH
ØUTPUT
REST


P.31
870
STAT
OUTPUT
END (1,0, 111)
ERASE
INLST RØUT
RIN RSUT
EQST XS(1)
ERASE
XS(2) XS(5)
XS(3) XZ
XS(4) INSTP
STATUS
PRINT
AVS INFO.
WRITE 3 EOF
ON TAPE 16
FINISH
GØCØM
COMPILE
& RUN

V. TAG EXECUTION PROGRAM

A. General Description

1. Function

The function of the Execution Program is to compute the network simulation and generate the performance data as specified by the user in the TAG Description Deck. The circuit simulation program generated in FORTRAN by the Preprocessor constitutes the main instruction sequence of the Execution Program. The remainder of the program is in the form of precoded and precompiled subroutines that provide the actual computational algorithms used.

The Execution Program reads in the parameter data listed in the Data Deck that pertains to the particular network under investigation. These parameter values are then substituted into the proper matrix coefficient expressions and special function statements. The matrices and special functions are used in evaluating the network simulation equations. As the simulation progresses, the network variables specified for output by the user are automatically printed and/or plotted at the specified intervals. The process continues until the specified simulation for each set of input data has been generated. The job then terminates, but the Execution Program may be saved for the purpose of creating further simulations of the particular topology for which the solution program was generated.

2. Organization

The general organization of the Execution Program is illustrated in Exhibit 17.

B. Detailed Description

The following is a detailed description of the steps performed by the Execution Program.

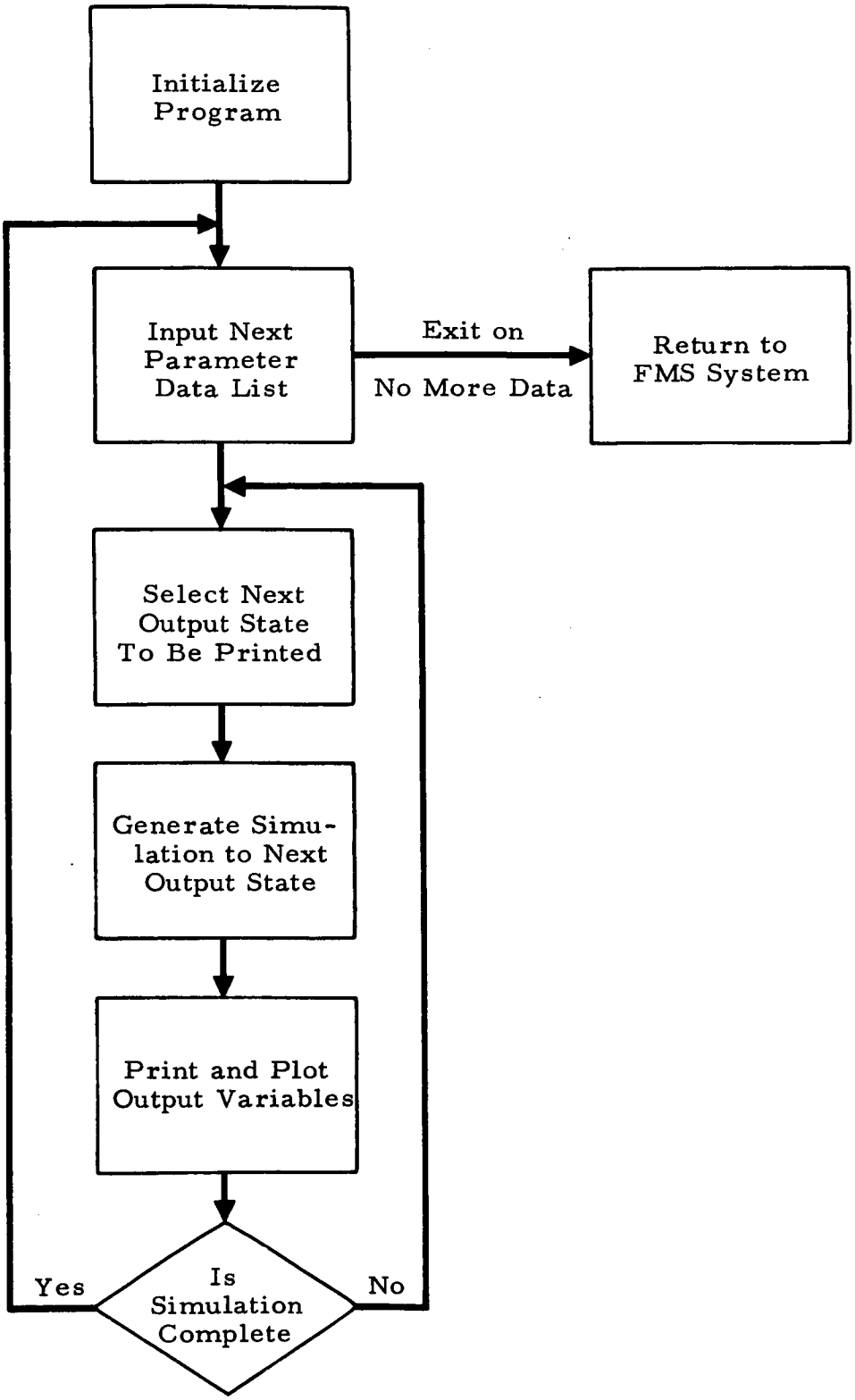


EXHIBIT 17 - GENERAL ORGANIZATION OF THE TAG EXECUTION PROGRAM

1. Program Initialization

   a. DIMENSION all computational matrices.
   b. DIMENSION and COMMON variables as required by FMARK, the transient solution integration routine.
   c. Assign values to the matrix partition constants.
   d. If required, initialize all transient analysis control constants.
   e. If required, initialize all nonlinear DC analysis control constants.
   f. Transfer to step 3.a.
   g. DIMENSION all arrays defined by the user and COMMON all variables as specified by the user.

2. Output Sequence and Termination Control

   a. Print out all variables as specified by the user and perform all FORTRAN instructions included by the user as part of the output sequence. Plot output variables as specified by the user.
   b. Determine whether the current simulation is complete. If it is complete, transfer to step 3.a. If it is not complete, transfer to step 4.b.

3. Establishing Initial Values for Computational Arrays

   a. Set all computational arrays to zero.
   b. Input all required parameter values and control constants from the next data list. If there is no next data list, the job is terminated.
   c. Initialize certain integration control variables.
   d. Initialize the computational voltage vector. This will set capacitor voltages to initial condition values.
   e. Evaluate all special function statements whose values are constant.
   f. Evaluate all constant valued matrix coefficients.
   g. Where required, invert the matrices completed by step 3.f.

4. Select Next Point for Output Sequence Execution

   a. For first pass, set output time stop to zero to ensure the output of initial condition state.

   b. For the initial pass and all subsequent dependent variable stops, transfer is immediately made to step 5.a. For all time stops, the next time stop is selected and the present time stop is examined to determine whether the integrator will require resetting before the simulation is continued.

5. Evaluation of All Equation Coefficients That Are Discontinuous in Time

   a. Evaluate all special function statements whose values change at time stops only (UTF and ULF).

   b. Evaluate all matrix coefficients whose values are dependent only on the functions evaluated in step 5.a.

   c. Where required, invert the matrices completed by step 5.b.

6. Control the Reentry Into FMARK From Print Stops, Time Stops, and Dependent Stops (Transient Only)

   a. Update the local time-stop available for FMARK.

   b. Reentry into FMARK from a print stop is accomplished without resetting the integrator.

   c. Reentry into FMARK from a time stop is accomplished by resetting the integrator from the top without evaluating the end-of-step box.

   d. Reentry into FMARK from a dependent stop is accomplished by resetting the integrator from the top after evaluating the dependent stop variables in the end-of-step box.

7. End-of-Step Box for FMARK Integration Routine (Transient Only)

   a. Evaluate all special dependent stop functions.

   b. Evaluate all matrix coefficients that change value at dependent variable stops only.

c. Where required, invert the matrices completed by step 7.b.

8. Set Up Initial and Reset Entry for FMARK (Transient Only)

a. Reset local time and stop time variable to zero.

b. Reset FMARK control variables to zero.

c. CALL FMARK. The integration routine, FMARK, is entered through the top only for initialization or resetting after a time or dependent stop.

9. FMARK Integration and Exit Control (Transient Only)

The rest of the program is primarily concerned with calculating the derivatives required by FMARK to determine all the voltages in the network at each step in simulated time. This part of the program is therefore called the derivative box. For the Adams-Moulton integration mode, the derivatives must be evaluated twice for each time step; for the Runge-Kutta mode the derivatives must be evaluated four times per step. After each evaluation, FMARK is entered through a special input that does not reset the integrator. At the end of each integration step, the end-of-step box, step 7, is completely evaluated and FMARK is reentered again without being reset.

When a print stop, a time stop, or a dependent stop occurs, exit is made from the internal integration loop to the beginning of the output sequence, step 2.a. If the simulation is continued, FMARK is reset upon reentry from a time stop or a dependent stop and is not reset upon reentry from a print stop.

10. Evaluation of Algebraic Portion of Simulation Equations

a. If the problem contains a nonlinear DC steady-state portion, the extra computational arrays required by the Newton-Raphson process are dimensioned and loaded with the required data.

b. Evaluate all special functions whose values change continuously with time or network state.

c. Evaluate all matrix coefficients whose values depend on the functions evaluated in step 10.b.

d. Where required, invert the matrices completed by step 10.c.

e. Evaluate the terminal flux of all all-inductance cut-sets.

f. Evaluate the voltage of all all-inductance cut-sets by numerically differentiating their terminal flux.

g. Evaluate terminal voltage of all all-conductance cut-sets and all-conductance-inductance cut-sets.

h. If the algebraic equations are nonlinear, setup Newton-Raphson subroutines for evaluating the residual vector and the next solution estimate. Establish the iteration logic to return to step 10.a, if the solution is not complete, and to continue on if the solution is complete.

i. If the circuit is purely DC, a correct solution will route the process to the beginning of the Output Sequence, step 2.a; otherwise the process will be routed to step 11.a.

11. Evaluation of First-Order Derivatives for Capacitive Cut-Set Voltages and All Cut-Set Fluxes If Required

a. Evaluate the first derivative of every voltage source by numerical differentiation.

b. Evaluate the first derivatives of every capacitive tree branch voltage.

c. Evaluate the first derivative of every tree branch flux. These are the tree branch voltages.

d. Return to FMARK through the entry that does not reset the integration process.

12. END Program

C. Example Execution Program

```
                                                    TRANSIENT ANALYSIS GENERATOR (W.J.THOMAS-JPL)
      DIMENSION FHB(6),LNH(6),FMIS(104)
      COMMON FMIS,FV021,FS011,FS021,FS031,FV21,FS11,FS21,FS31,FT2,FT1,LN
     1H,FHB
      DIMENSION FCMDMY(2874)
      COMMON FCMDMY
      LNV=1
      LNC=1
      LNG=1
      LNL=0
      FSTEP=1.E-11
      FEPSL2=5.E-6
      FEPSL3=5.E-4
      FEPSL4=1.E-16
      LTYPE=4
      GO TO 6000
      DIMENSION BF1(100),BF2(100),BF3(100),BF4(100),BF5(100),BF6(100)
   30 TIME=F1
      WRITE OUTPUT TAPE 6,8000,TIME
 8000 FORMAT (1H0,45X,5HTIME=,E16.8)
      VIN=+FV11
      WRITE OUTPUT TAPE 6,8001,VIN
 8001 FORMAT (1H ,46X,4HVIN=,E16.8)
      VOUT=+FV21
      WRITE OUTPUT TAPE 6,8002,VOUT
 8002 FORMAT (1H ,45X,5HVOUT=,E16.8)
      CIN=(-FV31)*SG0102
      WRITE OUTPUT TAPE 6,8003,CIN
 8003 FORMAT (1H ,46X,4HCIN=,E16.8)
      K=1
      IF(FT-DONE)61,60,60
   60 K=2
   61 CALL SCOPE(FT,VIN,BF1,100,K,9HVIN VOLTS,4HTIME)
      CALL SCOPE(FT,VOUT,BF2,100,K,10HVOUT VOLTS,4HTIME)
      CALL SCOPE(FT,CIN,BF3,100,K,8HCIN AMPS,4HTIME)
      IF(M)65,65,64
   65 IF(FT-DONE+TMAG)63,62,62
   62 CONTINUE
      CALL INPUT (6HSI0203,SI0203,6HSL0203,SL0203,6HSG0003,SG0003,6HSG01
     102,SG0102,6HSG0003,SG0003,6HSV0003,SV0003,6HSV0001,SV0001,2HFT,FT,
     25HFSTEP,FSTEP,5HEPSL,EPSL,6HEPSL1,EPSL1,6HEPSL2,EPSL2,6HEPS
     3L3,EPSL3,6HEPSL4,EPSL4,4HFOUT,FOUT,6HLDB001,LDB001,4HLMAX,LMAX,
     45HLTYP,LTYPE,5HLLCNT,LLCNT,6HLALGFT,LALGFT,5HSTOP1,STOP1,5HVSTOP,
     5VSTOP,4HVHIGH,VHIGH,4HVLOW,VLOW,3HVIH,VIH,3HVIL,VIL,3HVIN,VIN,4HVO
     6UT,VOUT,3HBF2,BF2,3HBF3,BF3,3HBF4,BF4,3HBF5,BF5,3HBF6,BF6,3HBFT,BF
     71,4HDONE,DONE,1HK,K,3HCIN,CIN,1HM,M,4HTMAG,TMAG,6H1$$END)
   64 CALL SCOPE(FT,VIN,BF4,100,K,9HVIN VOLTS,4HTIME)
      CALL SCOPE(FT,VOUT,BF5,100,K,10HVOUT VOLTS,4HTIME)
      CALL SCOPE(FT,CIN,BF6,100,K,8HCIN AMPS,4HTIME)
   63 IF(K-2)6200,6000,6000
 6000 CALL ZEROX(FS11,1)
      CALL ZEROX(FS21,1)
      CALL ZEROX(FS31,1)
      CALL ZEROX(FV31,1)
      CALL ZEROX(FG31,1)
      CALL ZEROX(FT31,1)
      CALL ZEROX(FG31,1)
```

```
      CALL ZEROX(FV11,1)
      CALL ZEROX(FG32,1)
      CALL ZEROX(FV21,1)
      CALL ZEROX(FL31,1)
      CALL ZEROX(FL32,1)
      CALL ZEROX(FL33,1)
      CALL ZEROX(FC11,1)
      CALL ZEROX(FI11,1)
      CALL ZEROX(FC12,1)
      CALL ZEROX(FG11,1)
      CALL ZEROX(FG12,1)
      CALL ZEROX(FG13,1)
      CALL ZEROX(FL11,1)
      CALL ZEROX(FL12,1)
      CALL ZEROX(FL13,1)
      CALL ZEROX(FVD21,1)
      CALL ZEROX(FC22,1)
      CALL ZEROX(FI21,1)
      CALL ZEROX(FG21,1)
      CALL ZEROX(FG22,1)
      CALL ZEROX(FG23,1)
      CALL ZEROX(FL21,1)
      CALL ZEROX(FL22,1)
      CALL ZEROX(FL23,1)
      CALL ZEROX(FSD11,1)
      CALL ZEROX(FSD21,1)
      CALL ZEROX(FSD31,1)
      CALL ZEROX(FC21,1)
      CALL ZEROX(FVD11,1)
      CALL ZEROX(FVD11,1)
      FT=0.
      CALL INPUT (6HSI0203,SI0203,6HSL0203,SL0203,6HSG0003,SG0003,6HSG01
     102,SG0102,6HSC0003,SC0003,6HSV0003,SV0003,6HSV0001,SV0001,2HFT,FT,
     29HFSTEP,FSTEP,5HFEPSL,FEPSL,6HFEPSL1,FEPSL1,6HFEPSL2,FEPSL2,6HFEPS
     3L3,FEPSL3,6HFEPSL4,FEPSL4,4HFOUT,FOUT,6HLDBG01,LDBG01,4HLMAX,LMAX,
     45HLTYPE,LTYPE,5HLLCNT,LLCNT,6HLALGFT,LALGFT,5HSTOP1,STOP1,5HVSTOP,
     5VSTOP,5HVHIGH,VHIGH,4HVLOW,VLOW,3HVTH,VTH,3HVTL,VTL,3HVIN,VIN,4HVO
     6UT,VOUT,3HBF2,BF2,3HBF3,BF3,3HBF4,BF4,3HBF5,BF5,3HBF6,BF6,3HBF1,BF
     71,4HDONE,DONE,1HK,K,3HCIN,CIN,1HM,M,4HTMAG,TMAG,6H$$$END)
6100  CONTINUE
      LALGFT=1
      LINT=0
      LCNT=1
      FTL=FT
      FTU=FT
      FHC=FSTEP
      FV11=+SV0001
      FV21=+SV0003
      FV31=+SV0102
      FC22=+SC0003
      FG22=+SG0003
      FG33=+SG0102
      FL11=+SL0203
      FL12=-SL0203
      FL13=+SL0203
      FL21=-SL0203
```

```
      FL22=+SL0203
      FL23=-SL0203
      FL31=+SL0203
      FL32=-SL0203
      FL33=+SL0203
      FI11=-SI0203
      FI21=+SI0203
      FI31=-SI0203
      FCI22=1./FC22
      FGI33=1./FG33
      CALL RSTOP (FSTOP,FT,FHC)
 6200 IF(LCNT-3)6202,6201,6395
 6201 CALL STOP(FOUT,LINT)
 6202 CONTINUE
      GO TO (6395,6390),LALGFT
 6390 IF(LINT)6391,6391,6393
 6391 FSTOP=FSTOP-FTL
      CALL ROUT(0)
 6393 FSTOP=FSTOP-FT
      GO TO 6425
 6395 LEOS=1
 6400 CONTINUE
      LLCNT=LCNT-3
      CALL VMOD((+FV11),(+FV21),STOP1,VSTOP,VHIGH,VLOW,VTH,VTL,LLCNT,LAL
     1GFT,1)
      GO TO(6422,6420),LALGFT
 6420 IF(LEOS)6421,6421,6422
 6421 CALL ROUT(0)
 6422 FSTOP=0.
 6425 FT1=0.
      FT2=0.
      FTL=FT
      FHB(1)=FSTEP
      FHB(2)=1.E-5
      FHB(3)=FEPSL4
      FHB(4)=.5
      FHB(5)=FEPSL2
      FHB(6)=FEPSL3
      LNH(1)=4
      LNH(2)=LNH(1)
      LNH(5)=5
      CALL FMARK(LCNT,FHB,0,LTYPE,1,1,2,FHC,3,FT1,FSTOP,4,STOP1,0,0)
      FT=FTL+FT1
      LEOS=0
      IF(LCNT-2)6300,6400,30
 6300 CONTINUE
CFV31=FGI33*(FI31-FG31*FV11-FG32*FV21-FL31*FS11-FL32*FS21-FL33*FS31)
      CALL MULT(FL33,FS31,FTEM1,1,1,1)
      CALL MULT(FL32,FS21,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FL31,FS11,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FG32,FV21,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FG31,FV11,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
```

```
      CALL MSUM(FI31,FTEM1,FTEM1,1,1)
      CALL MULT(FGI33,FTEM1,FV31,1,1,1)
      LALGFT=2
      FDLT=FT-FTO
      IF(FDLT)7030,7030,7025
 7025 FVD11=(FV11-FVO11)/FDLT
 7030 FVO11=FV11
      FTO=FT
CFVD21=FCI22*(FI21-FG21*FV11-FG22*FV21-FG23*FV31-FL21*FS11-FL22*FS21-FL2
C3*FS31-FC21*FVD11)
      CALL MULT(FC21,FVD11,FTEM1,1,1,1)
      CALL MULT(FL23,FS31,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FL22,FS21,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FL21,FS11,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FG23,FV31,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FG22,FV21,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MULT(FG21,FV11,FTEM2,1,1,1)
      CALL PSUM(FTEM2,FTEM1,FTEM1,1,1)
      CALL MSUM(FI21,FTEM1,FTEM1,1,1)
      CALL MULT(FCI22,FTEM1,FVD21,1,1,1)
      FSD11=FV11
      FSD21=FV21
      FSD31=FV31
      CALL RQUT(0)
      END(1,0,0,0,0,0,1,0,0,0,0,0,0,0,0)
  2SC 35141 MAX  1747 COUNT   931
```

VI. TAG SUBROUTINE WRITEUPS

A. Program Hierarchy

Hierarchy Level	Subroutine Name	Pass		Hierarchy Level	Subroutine Name	Pass	
2	ADDLOC	1	2	3	ELIM		2
7	AFTER	1	2	2	EQFS41		2
6	AFTLK	1	2	2	EQFV21		2
8	BACK	1	2	2	EQFV31		2
2	BAKELM	1		6	ERASE	1	2
4	BCB		2	3	ERASEA	1	2
6	BCD		2	4	EXCPT		2
4	BCL		2	4	EXTRX		2
6	BND2		2	3	FISH	1	
5	CHLNE	1	2	3	FLAG	1	
3	COMBN		2	4	FLTCON		2
5	COPY		2	5	FOUTPT		2
2	COTRN	1		5	FRACT		2
4	DBPCHC		2	5	FRFL	1	2
4	DBPCH		2	8	FROM	1	2
4	DBPFH		2	2	GOBLE	1	
4	DIFA	1	2	5	HEAD		2
2	DIMEN		2	5	HEADC		2
5	DONBD		2	3	HOLBK		2
7	DOWN	1	2	5	IDNTP	1	2
4	DOWNS	1	2	5	INFL	1	2
2	DPDST		2	2	INPUTX		2

Hierarchy Level	Subroutine Name	Pass		Hierarchy Level	Subroutine Name	Pass	
5	INSRT	1	2	3	PRPTR	1	2
2	INTLST		2	2	PUSPCH		2
8	INTO	1	2	2	READCH		2
2	INVST		2	3	RECOVR		2
2	INZERO		2	4	SEGMNT	1	2
2	LEVMRK		2	6	SET	1	2
8	LINK	1	2	4	SNATCH	1	2
4	LNECH	1	2	4	STASH	1	2
7	LNKT	1	2	3	STAT		2
5	LOCAT	1	2	3	STATC		2
5	LOCATA	1	2	2	STATUS	1	2
3	MATFT	1	2	2	STRIK	1	
3	MATOT	1	2	2	SUBST		2
3	MRKLST		2	4	SYMBL	1	2
2	MULTS	1		6	SYMCH	1	2
9	NEWLOC	1	2	2	SYMCRD		2
2	NLINDM		2	3	SYMTP		2
3	NUMB	1		2	TRANS	1	
6	PAGEHD		2	2	TREE	1	
2	PARAM	1		4	UPDWN	1	2
2	PARTS		2	3	WRTEQ		2
7	POPUP	1	2	2	XFORM	1	
3	PRPTG	1	2	2	ZEROX		2

B. Subroutine Writeups and Flow Charts for TAG Preprocessor Program Description

1. Identification

a. Routine Label

ADDLOC

b. Name

Allocate available space (AVS).

2. Function

Space is reserved in core as a list of elements linked to each other by a pointer in the link position of the word.

3. Programming System

FAP

4. Usage

a. Calling Sequence

CALL ADDLOC (A, I)

b. Entry Conditions

A = Head of AVS

I = Number of cells to be allocated as elements in AVS

c. Exit Conditions

a(AD90) = Head of AVS with pointer to current available element

a(AD91) = Pointer to last element in AVS

d. Error Exits

None.

5. Definition of Identifiers

AVS = Available space list from which elements for all list structures are taken

IN90 = A location containing the address AD93, a pointer to the last available element in AVS

6. Method

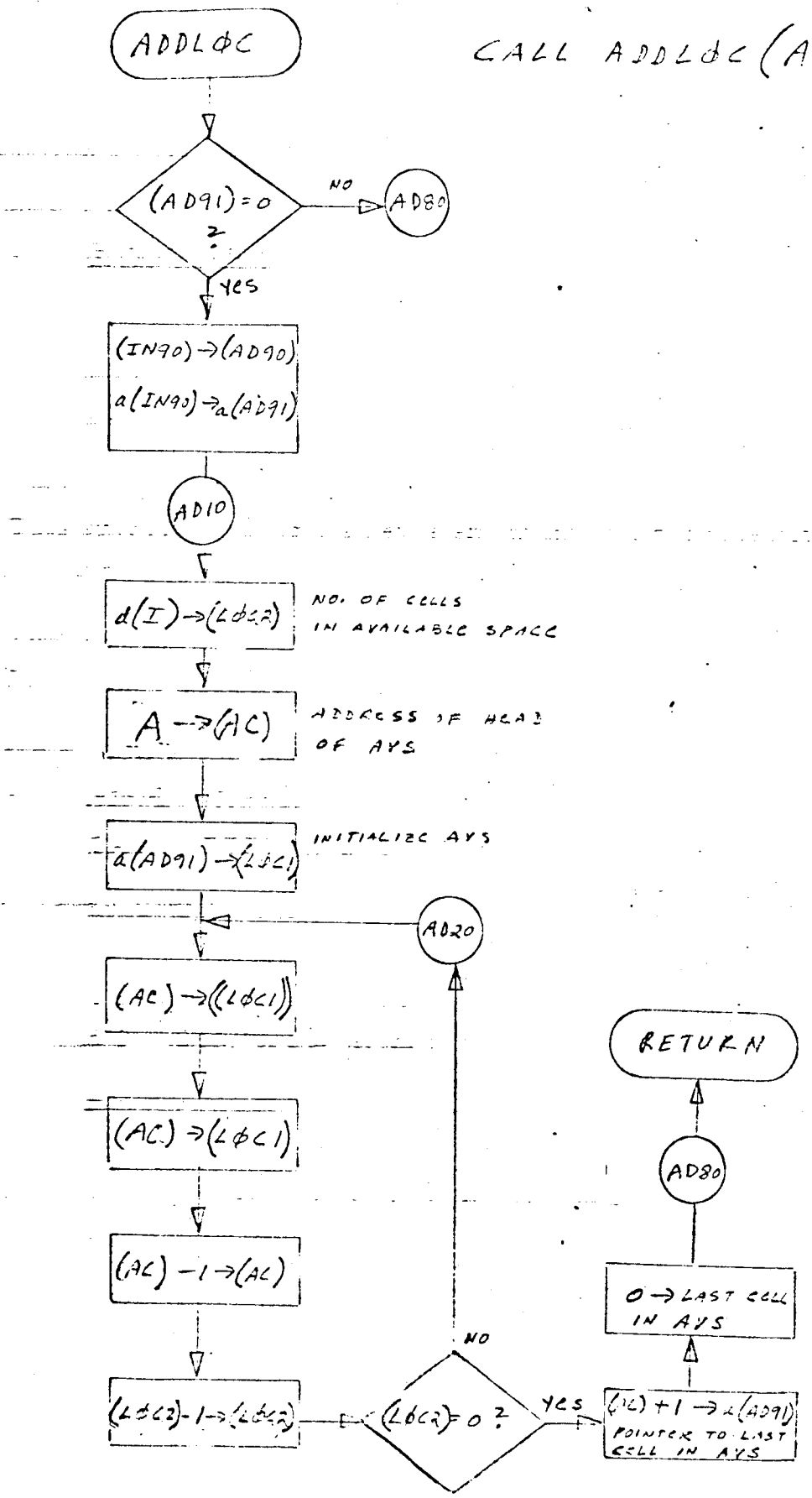
A list, headed by A, is formed of I elements, which occupy sequentially descending positions in core. Each element is linked to its successor by a pointer in the link portion (bits 21-35) of the word. The list is terminated with an element whose link contains zero.

7. Other Subroutines Used

None.

8. Using Subroutines

Main Program for Pass 1 of TAG Preprocessor.
Main Program for Pass 2 of TAG Preprocessor.


ADDLØC
CALL ADDLØC(A, I)
(AD91) = 0 ?
NO
AD80
YES
(IN90) → (AD90)
a(IN90) → a(AD91)
AD10
d(I) → (LØC2)
NO. OF CELLS IN AVAILABLE SPACE
A → (AC)
ADDRESS OF HEAD OF AVS
a(AD91) → (LØC1)
INITIALIZE AVS
AD20
(AC) → ((LØC1))
(AC) → (LØC1)
(AC) - 1 → (AC)
(LØC2) - 1 → (LØC2)
(LØC2) = 0 ?
NO
YES
(AC) + 1 → a(AD91)
POINTER TO LAST CELL IN AVS
0 → LAST CELL IN AVS
AD80
RETURN

Program Description

1. Identification

   a. Routine Label

      AFTER

   b. Name

      Insert an element AFTER the first element.

2. Function

This subroutine inserts a new element into a list immediately following the first element of that list.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL AFTER (A, B, C)

   b. Entry Conditions

      A = Head of a list
      d(B) = Item of new element
      d(C) = Flag of new element

   c. Exit Conditions

      List A contains a new element with d(B) as its item and d(C) as its flag. The first element in list A is linked to this new element.

   d. Error Exits

      None.

5. Definition of Identifiers

   d(AF90) = Pointer to new element to be inserted

6. Method

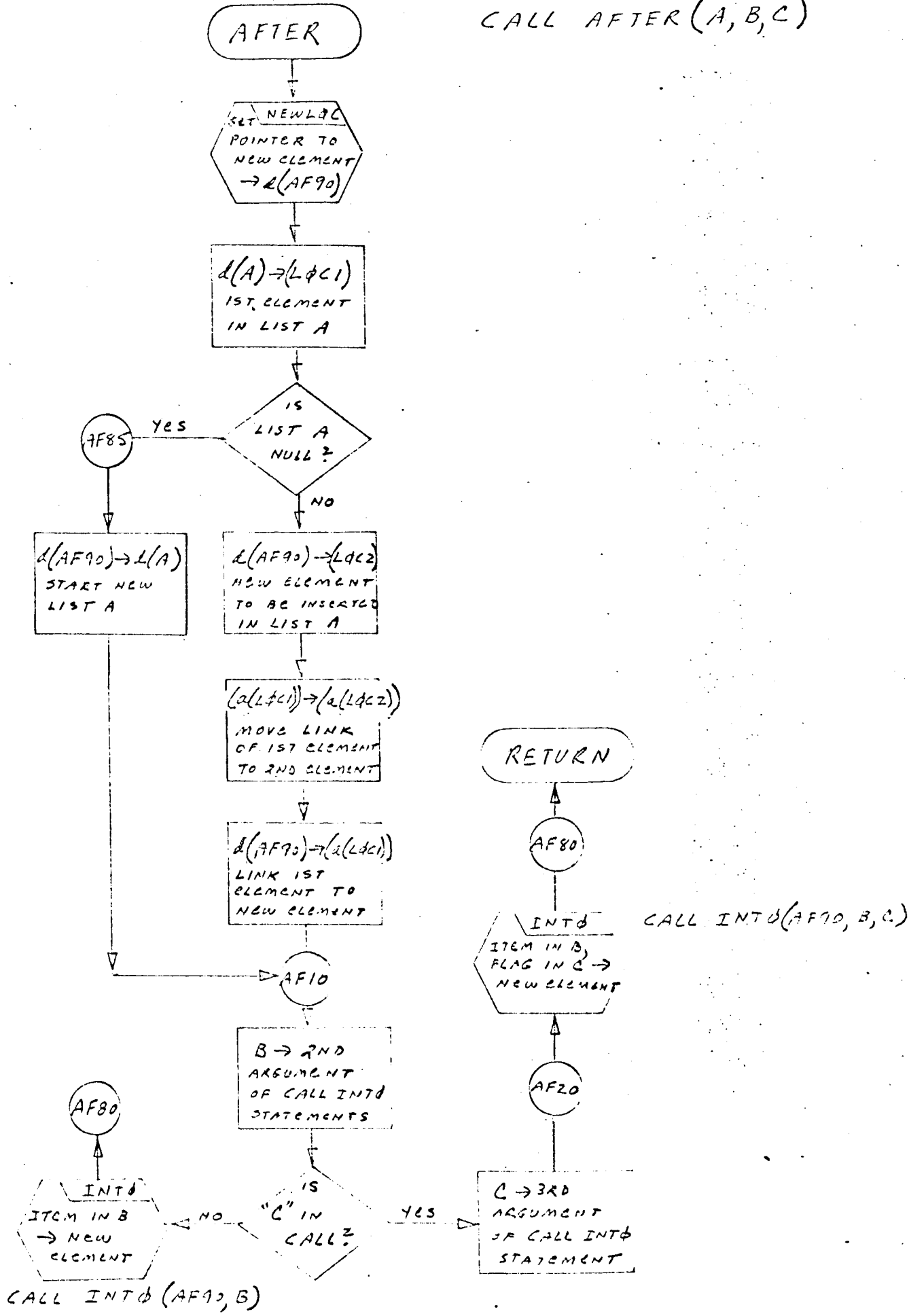
A new element is acquired using NEWLOC. If list A is null, a new list is started that contains this one element, with a link of zero. If list A is not null, its first element is linked to the new element. The link formerly in the first element becomes the link in the new element, thus maintaining the list connections. The item and flag (if any) given in d(B) and d(C) is pushed down into the element by INTO.

7. Other Subroutines Used

INTO, NEWLOC

8. Using Subroutines

AFTLK, COPY, DONBD, MRKLST, SUBST, SYMBL.


AFTER
CALL AFTER (A, B, C)
SET NEWLOC POINTER TO NEW ELEMENT → L(AF90)
L(A) → (L+C1)
1ST ELEMENT IN LIST A
IS LIST A NULL ?
Yes
NO
AF85
L(AF90) → L(A)
START NEW LIST A
L(AF90) → (L+C2)
NEW ELEMENT TO BE INSERTED IN LIST A
(L(L+C1)) → (L(L+C2))
MOVE LINK OF 1ST ELEMENT TO 2ND ELEMENT
L(AF90) → (L(L+C1))
LINK 1ST ELEMENT TO NEW ELEMENT
AF10
B → 2ND ARGUMENT OF CALL INTΦ STATEMENTS
IS "C" IN CALL ?
NO
Yes
AF80
INTΦ
ITEM IN B → NEW ELEMENT
CALL INTΦ (AF90, B)
C → 3RD ARGUMENT OF CALL INTΦ STATEMENT
AF20
AF80
INTΦ
ITEM IN B, FLAG IN C → NEW ELEMENT
CALL INTΦ (AF90, B, C)
RETURN

Program Description

1. Identification

   Routine Label

   AFTLK

2. Function

This subroutine inserts a new element immediately after the first element of a list and resets the head of the list to point to this new element. If the inserted element is the only entry in the list, its location is saved in a flag word.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL AFTLK (A, B, G, F)

   b. Entry Conditions

      A = Head of a list
      d(B) = Item of the new element
      d(G) = Flag of the new element
      F = Flag word

   c. Exit Conditions

If list A was null, a first element is inserted into A with d(B) as its item and d(G) as its flag. If A was not null, a new element, with d(B) as its item and d(G) as its flag, is inserted between the first and second elements of A. In either case, A is reset to point to this new element. If the inserted element is the first and only entry in the list, its location is saved in F.

   d. Error Exits

      None.

5. Definition of Identifiers

None.

6. Method

a. If list A was not null:

(1) A new element with d(B) as its item and d(G) as its flag is inserted immediately after the first element.

(2) The first element is linked to the new element.

(3) The head cell of list A is set to point to the new element.

(4) The new element is linked to the element which was formerly the second element.

b. If list A was null:

(1) A new element is inserted with d(B) as its item, d(G) as its flag, and a link of zero, creating a list of just one element.

(2) The head cell of list A is set to point to this element.

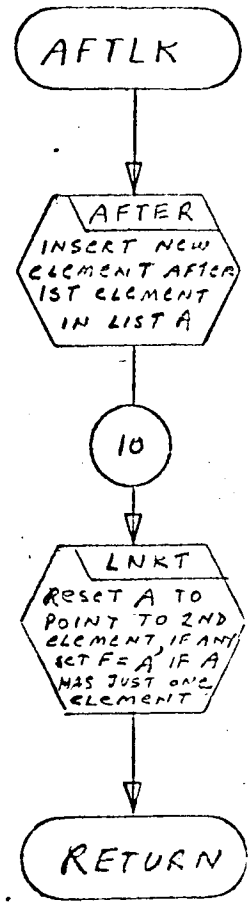
(3) Location of the single element in A is saved in the flag word F.

7. Other Subroutines Used

AFTLK, LNKT

8. Using Subroutines

ELIM, INSRT, MATFT, MRKLST, PRPTG, STASH.



CALL AFTLK(A, B, G, F)

CALL AFTER (A, B, G)

CALL LNKT(A, A, F)

Program Description

1. Identification

   a. Routine Label

      BACK

   b. Name

      Restore an element BACK to AVS.

2. Function

Restores an element back to available space (AVS), then decreases number of elements used by one.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL BACK (I)

   b. Entry Conditions

      d(I) = The decrement of location I holds pointer to element which is to be restored.

   c. Exit Conditions

      (1) Element pointed to by d(I) restored to AVS.
      (2) Pointer to next available element in AVS reset to restored pointer.
      (3) COUNT decreased by one.

   d. Error Exits

If pointer to last element in AVS = 0, or if location I = 0, CALL DUMP is executed and return is made to the FORTRAN monitor system.

5. Definition of Identifiers

a(AD91) = Pointer to last element in AVS

COUNT = Number of elements used from AVS

MAX = Max count of elements used from AVS, maintained for printout by subroutine STATUS

6. Method

a. If (AD91) = 0, the subroutine ADDLOC has not yet been executed to reserve space in AVS, and an error return is made.

b. If d(I) = 0, there is no element to be restored and an error return is made.

c. If (AD91) $\neq$ 0, and d(I) $\neq$ 0, COUNT and MAX are updated and the element pointed to by d(I) is restored to AVS.

d. Pointer to next available element in AVS is reset to the element just restored.

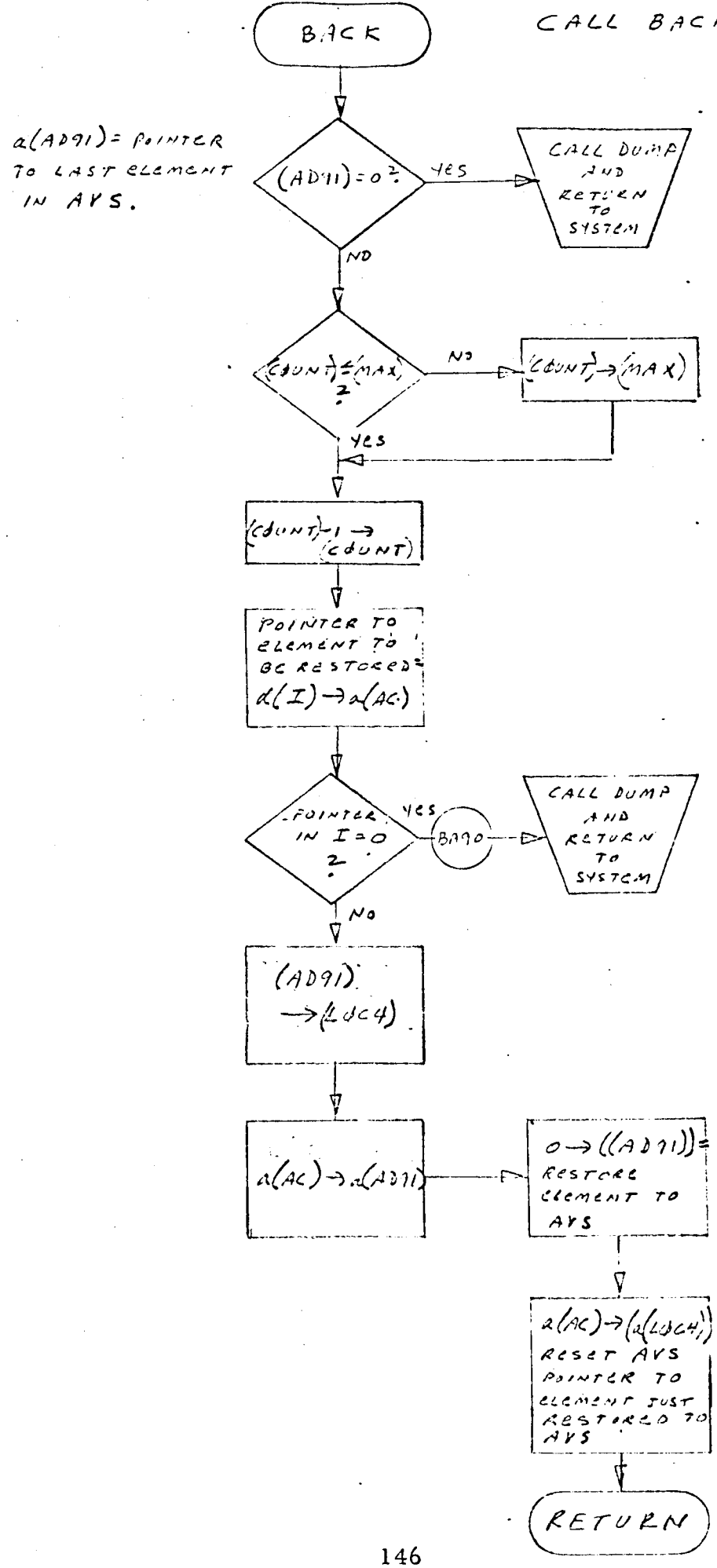
7. Other Subroutines Used

None.

8. Using Subroutines

ERASE, POPUP.

CALL BACK(I)


BACK
a(AD91) = POINTER TO LAST ELEMENT IN AVS.
(AD11) = 0 ?
YES
CALL DUMP AND RETURN TO SYSTEM
NO
(COUNT) ≤ (MAX) ?
NO
(COUNT) → (MAX)
YES
(COUNT) - 1 → (COUNT)
POINTER TO ELEMENT TO BE RESTORED a(I) → a(AC)
POINTER IN I = 0 ?
YES
BA90
CALL DUMP AND RETURN TO SYSTEM
NO
(AD91) → (LOC4)
a(AC) → a(AD11)
0 → ((AD11)) RESTORE ELEMENT TO AVS
a(AC) → (a(LOCH)) RESET AVS POINTER TO ELEMENT JUST RESTORED TO AVS
RETURN

Program Description

1. Identification

   a. Routine Label

      BAKELM

   b. Name

      Perform Jordan elimination on each row of a matrix.

2. Function

BAKELM transforms TQ, the NRR x NM tree voltage based transformer constraint matrix, to a form in which each row, I , intersects a column whose Ith element is unity and whose other elements are all zero. Each row of the transformed TQ matrix is a solution for one of the NRR coordinate variables in the unconstrained tree voltage vector, VT , as a linear combination of the NM - NRR remaining coordinate variables of VT. By this process VT is divided into NM - NRR independent coordinate variables, VTI, and NRR dependent coordinate variables, VTD. The transformation is applied to TQ so that priority for membership in the dependent variable subvector, VTD, is given to the coordinate variables at the bottom of VT.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL BAKELM (TQ, NRR, NM)

   b. Entry Conditions

      TQ = The NRR x NM tree voltage based transformer constraint matrix calculated by the product TRR * TC such that TQ * VT = 0.

      NRR = The number of transformer constraint equations that is equal to the total number of transformer windings less the total number of transformers.

NM = The number of coordinate variables in the unconstrained tree voltage vector VT that is equal to the number of nodes less one.

c. Exit Conditions

TQ = An equivalent NRR x NM transformer constraint matrix in which NRR of the columns have been transformed into the NRR members of the NRR dimensioned identity matrix. Thus NRR of the tree voltage variables are essentially solved in terms of the NM - NRR remaining variables. The transformed columns are chosen whenever possible from the right-hand side of TQ.

d. Error Exits

If NRR of the tree voltage variables cannot be solved for, a dump is called and control is returned to the FORTRAN Monitor.

5. Definition of Identifiers

J = The row search index

I = The column search index

FTEM, FTEM1, FTEM2, FTEM3 are used as temporary variable names for the elements of TQ being operated upon.

L and K are used as search indices for various operations. NR is used to keep track of the number of pivot columns already found.

6. Method

BAKELM uses Jordan elimination to transform the NRR x NM TQ matrix to a form that solves for NRR coordinate variables of the unconstrained tree voltage vector, VT, in terms of the remaining NM - NRR variables. In this process, priority for membership in the set of NRR dependent coordinate variables is given to the bottom of the tree voltage vector, VT, by choosing pivot columns starting from the right-hand side of TQ, and pivot rows from the bottom. The

usual result of this process is to produce a transformed TQ matrix of the form [TQ' : U'] , where TQ is an NRR x NM - NRR submatrix, which expresses the dependent coordinate voltages, as a linear function of the reduced set of independent coordinate variables; and U' is a permuted NRR x NRR identity matrix which expresses the fact that the rows of TQ' will not solve for the dependent variables in the same order as they appear in VT. For some nondegenerate transformer connections, it will not be possible to select the bottom NRR coordinate variables to be dependent. In such cases the columns of TQ' and U' may be interwoven. The following algorithm defines the process as mechanized in BAKELM.

a. Set NR, the elimination control index, to 0 (NR = NRR terminates the process).

b. Set I, the pivot column search index, to NM.

c. Set J, the pivot row search index, to NRR.

d. Starting at row J, search up column I for the first non-zero element. When found, stop the search so that J is the row number and I the column number at which this possible pivot element is located. Store value of element JI in FTEM and go to step e. If the search reaches the top of column I (J = 0) without finding a non-zero element in an unpivoted row, I is decreased by one; and if I > 0 return is made to step c. If I = 0 a dump is taken and control is returned to the FORTRAN Monitor.

e. If I = NM, I is the first column searched and element JI has to be a proper pivot element. The next step taken is g.

f. If I $\neq$ NM, search all elements of row J to the right of column I. If they are all zero, element JI is a proper pivot element and the next step is g. If they are not all zero, row J has already been pivoted on and element JI may not be used as a pivot. In this case, the next step is to add one to J and return to step d.

g. Add one to NR and set K, the pivot row normalization index, to 1.

h. Starting at column 1, replace every element of row J by its initial value divided by the value of the pivot element, JI, stored in FTEM. Thus E_{JK}(NEW) = E_{JK}(OLD)/EJI)OLD) for K = 1, 2, ---NM.

i. Set L, the row elimination index, to 1. Starting at the top of the column I, search down until the first non-zero element is located at LI, and store its value in FTEM. Starting at the left, replace each element of row L by its value decreased by the product of the corresponding element in row J, and the element initially located at LI, whose value is stored in FTEM.

Thus $E_{LK}(NEW) = E_{LK}(OLD) - E_{JK}(NEW) * E_{LI}(OLD)$ for K = 1, 2, ---NM. This process is continued until L = J - 1, at which time all the elements of column I will be reduced to zero except element JI which will be +1.

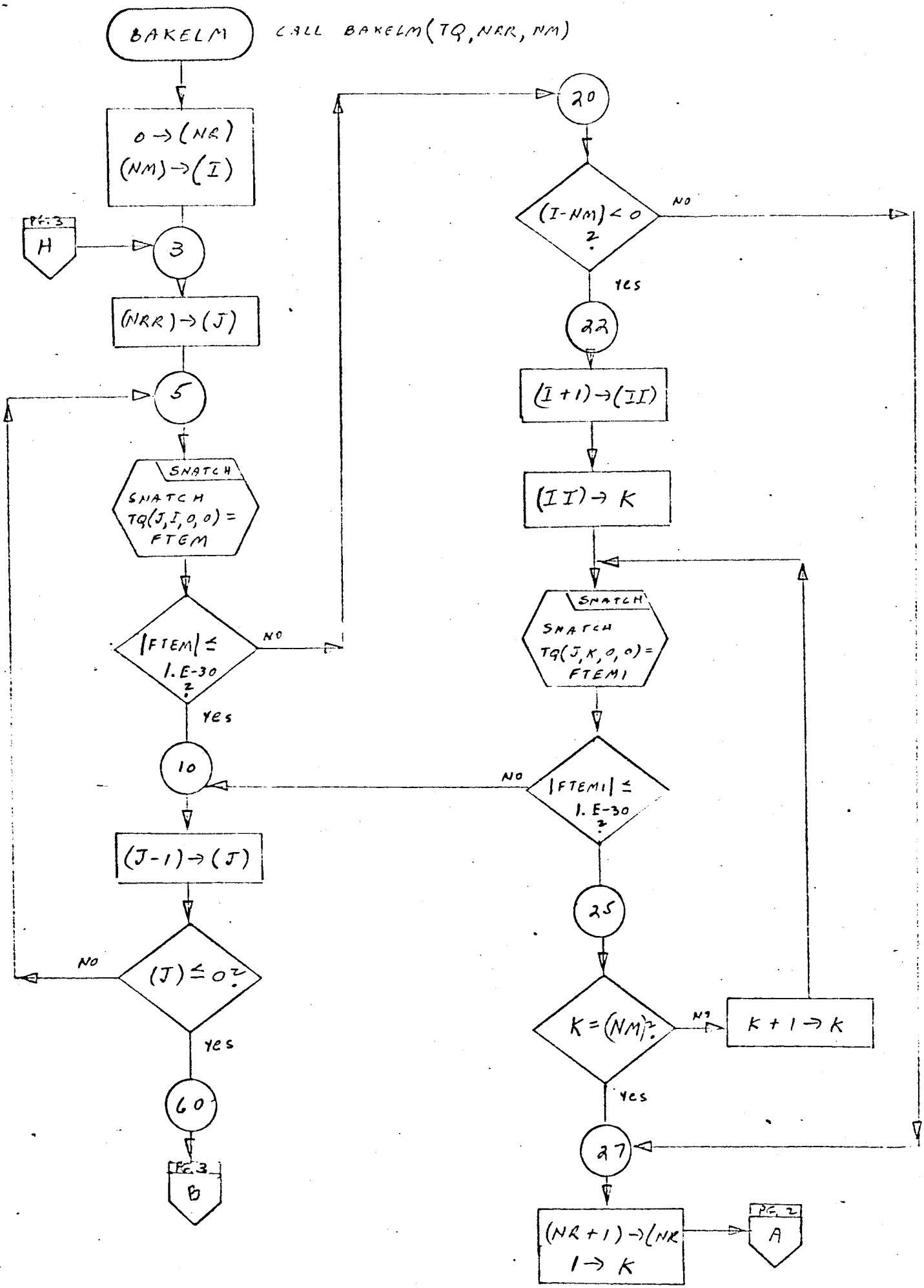
j. If NR is less than NRR, I is decreased by one. If I equals zero, a dump is taken and control is returned to the FORTRAN monitor. If I is greater than zero, the process returns to step c. If NR = NRR, a return is made to the main program.

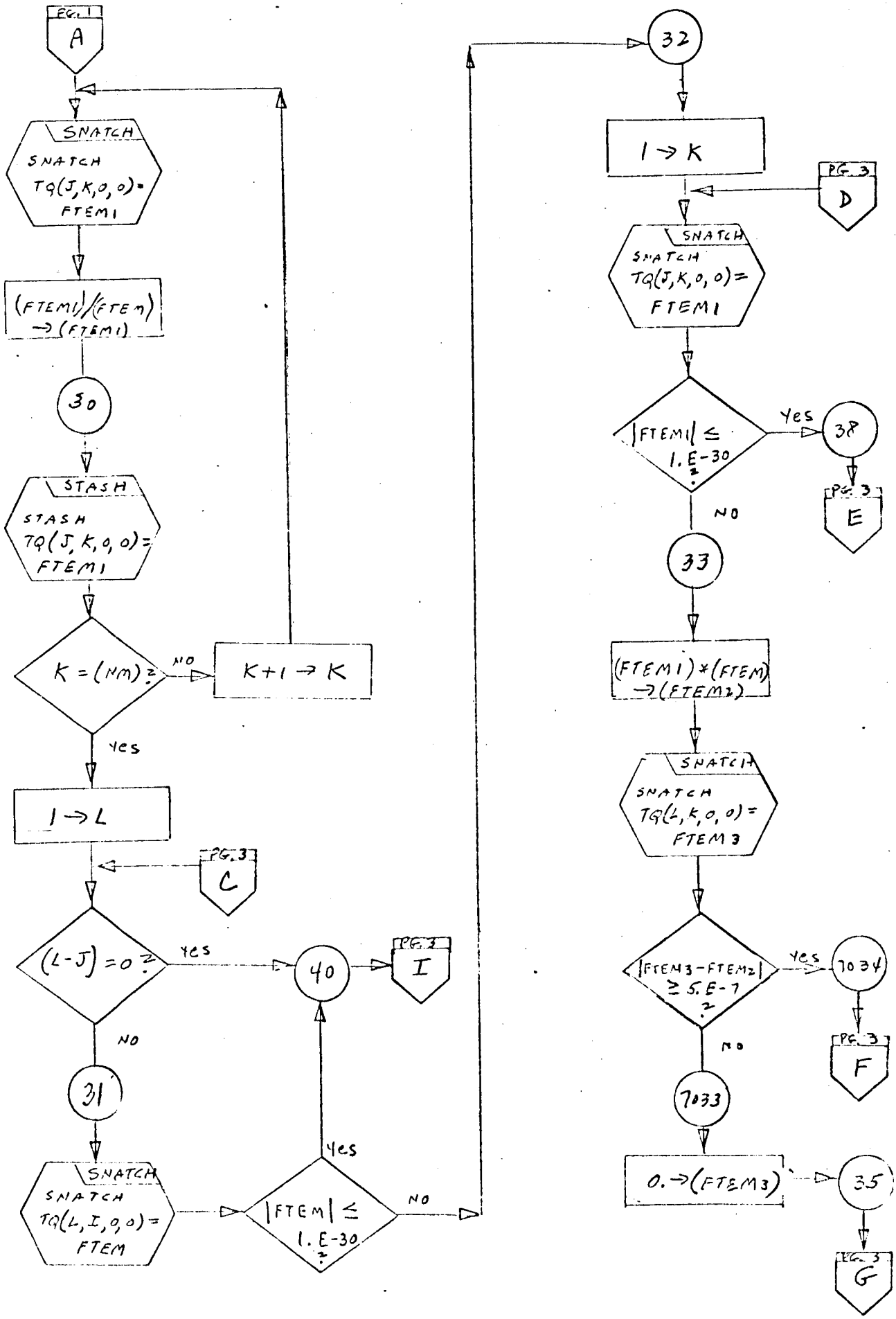
7. Other Subroutines Used

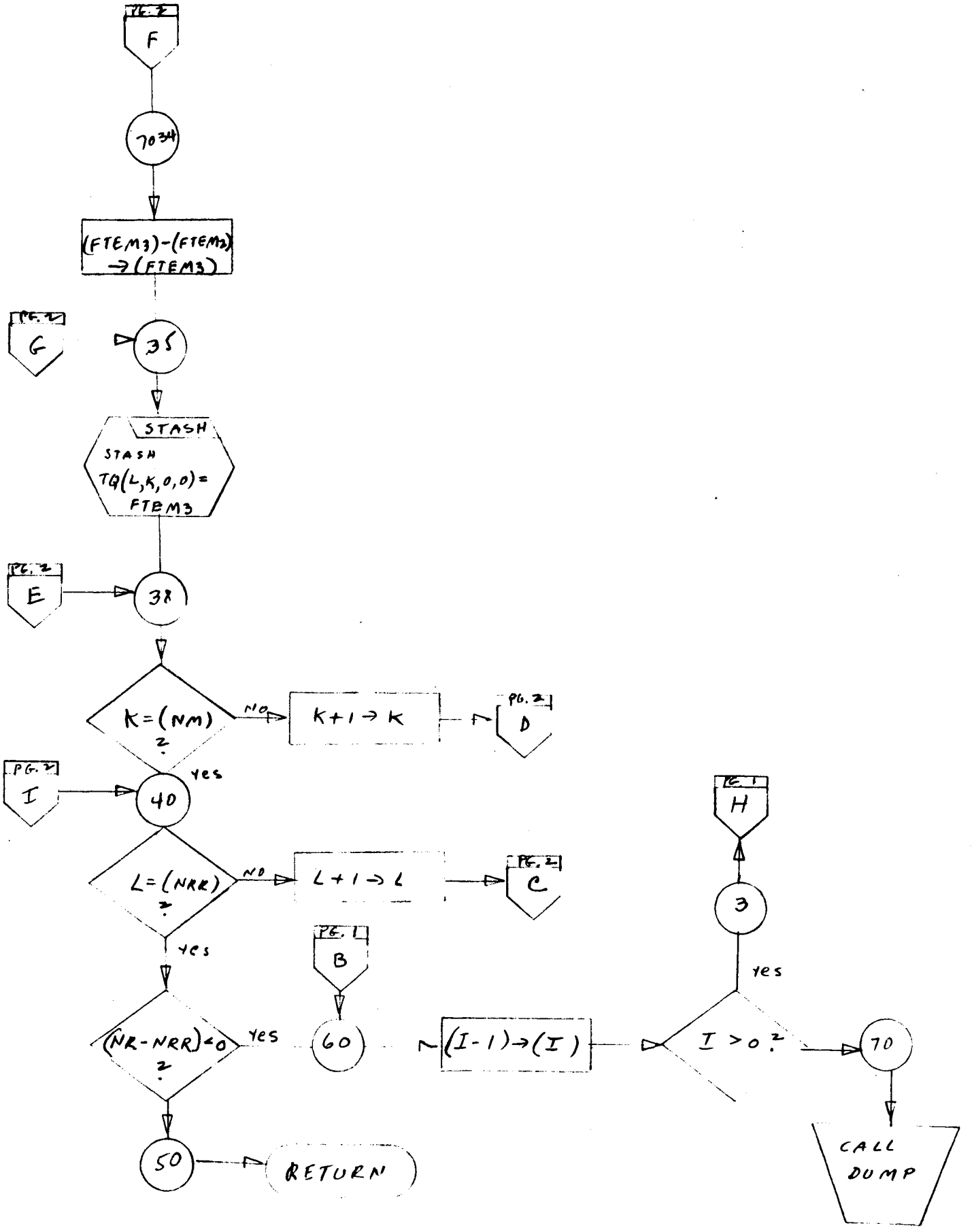
DIFA, DUMP, SNATCH, STASH.

8. Using Subroutines

Main Program for Pass 1 of TAG Preprocessor.


BAKELM
CALL BAKELM(TQ, NRR, NM)
0 → (NR)
(NM) → (I)
PG. 3
H
3
(NRR) → (J)
5
SNATCH
SNATCH TQ(J,I,0,0) = FTEM
|FTEM| ≤ 1.E-30 ?
NO
Yes
10
(J-1) → (J)
(J) ≤ 0 ?
NO
Yes
60
PG. 3
B
20
(I-NM) < 0 ?
NO
Yes
22
(I+1) → (II)
(II) → K
SNATCH
SNATCH TQ(J,K,0,0) = FTEM1
|FTEM1| ≤ 1.E-30 ?
NO
25
K = (NM) ?
NO
K + 1 → K
Yes
27
(NR+1) → (NR
1 → K
PG. 2
A


PG. 1
A
SNATCH
SNATCH
TQ(J,K,0,0)=
FTEM1
(FTEM1)/(FTEM)
→(FTEM1)
30
STASH
STASH
TQ(J,K,0,0)=
FTEM1
K=(NM)?
NO
K+1→K
yes
1→L
PG. 3
C
(L-J)=0?
yes
40
PG. 3
I
NO
31
SNATCH
SNATCH
TQ(L,I,0,0)=
FTEM
|FTEM|≤
1.E-30?
yes
NO
32
1→K
PG. 3
D
SNATCH
SNATCH
TQ(J,K,0,0)=
FTEM1
|FTEM1|≤
1.E-30?
yes
38
PG. 3
E
NO
33
(FTEM1)*(FTEM)
→(FTEM2)
SNATCH
SNATCH
TQ(L,K,0,0)=
FTEM3
|FTEM3-FTEM2|
≥5.E-7?
yes
7034
PG. 3
F
NO
7033
0.→(FTEM3)
35
PG. 3
G


PG. 2
F
7034
(FTEM3)-(FTEM2) → (FTEM3)
PG. 2
G
35
STASH
STASH TQ(L,K,0,0) = FTEM3
PG. 2
E
38
K=(NM)?
NO
K+1 → K
PG. 2
D
yes
PG. 2
I
40
L=(NRR)?
NO
L+1 → L
PG. 2
C
yes
PG. 1
B
(NR-NRR)<0?
yes
60
(I-1) → (I)
I > 0?
yes
PG. 1
H
3
70
CALL DUMP
50
RETURN

Program Description

1. Identification

   a. Routine Label

      BCB

   b. Name

      Convert a BCD number to an integer.

2. Function

This subroutine performs a BCD-to-binary conversion. The converted number is placed in the lower part of the accumulator.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL BCB (ERROR)

   b. Entry Conditions

      BCD number in the accumulator.

   c. Exit Conditions

      Converted number in lower accumulator as a binary integer. Contents of the MQ are not destroyed.

   d. Error Exits

      Error exit is taken if any illegal character is encountered during the conversion.

5. Definition of Identifiers

   ERROR = Contains location of the character in error
   BC85 = A table of binary equivalents for a BCD number. Used with convert instruction CAQ.

6. Method

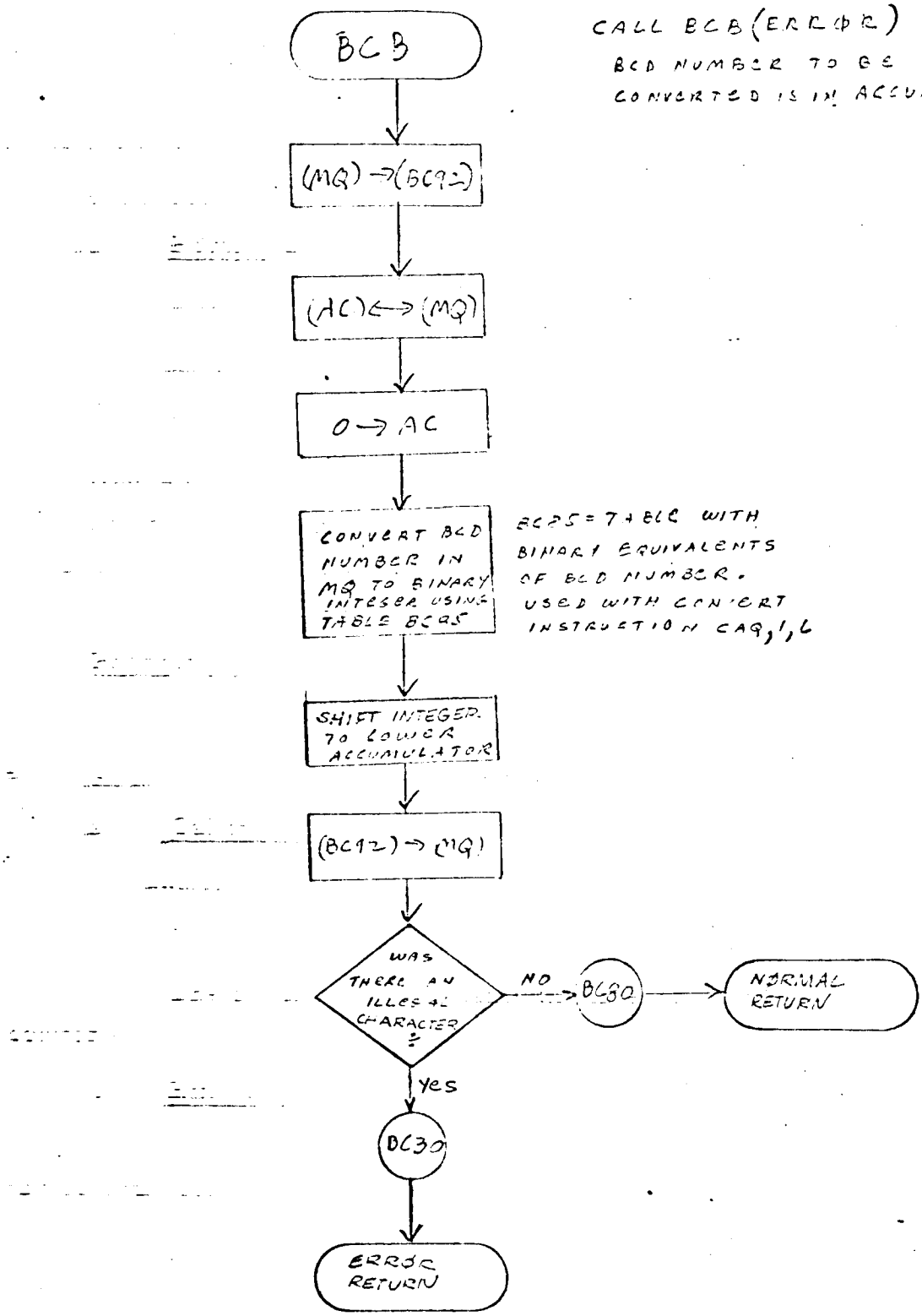
The contents of the MQ (on entry to the routine) are saved in BC92. The BCD number in the accumulator is placed in the MQ, then the accumulator is cleared to zero, in preparation for use of these registers with a CAQ instruction. A table lookup in BC85 is performed for each character to be converted. If an illegal character is encountered, an error exit is taken with the location of the character in ERROR. If all characters were legal, low accumulator will contain the converted number.

7. Other Subroutines Used

None.

8. Using Subroutines

STATC


BCB
CALL BCB(ERROR)
BCD NUMBER TO BE CONVERTED IS IN ACCUMULATOR
(MQ) → (BC92)
(AC) ↔ (MQ)
0 → AC
CONVERT BCD NUMBER IN MQ TO BINARY INTEGER USING TABLE BC85
BC85 = 7 + BCD WITH BINARY EQUIVALENTS OF BCD NUMBER. USED WITH CONVERT INSTRUCTION CAQ,1,6
SHIFT INTEGER TO LOWER ACCUMULATOR
(BC92) → (MQ)
WAS THERE AN ILLEGAL CHARACTER ?
NO
BC80
NORMAL RETURN
YES
BC30
ERROR RETURN

Program Description

1. Identification

   a. Routine Label

      BCD

   b. Name

      Convert an integer to a BCD character.

2. Function

An integer, right adjusted in the accumulator, is converted to a six-character BCD number (right adjusted with leading blanks). The least significant digit is at the top of the list.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL BCD

   b. Entry Conditions

The accumulator contains the integer, right adjusted, to be converted.

   c. Exit Conditions

Six-character BCD number in the accumulator, right adjusted, with leading blanks.

   d. Error Exits

      None.

5. Definition of Identifiers

BC90 = Local variable used to accumulate BCD characters during the conversion.

BC92 = Integer in accumulator, right adjusted, to be converted.

BC93 = A table of six entries, with BCD blank characters (0 to 5 blanks).

BC94 = A table of six entries, each a shift operation, used to right-adjust the converted number in the accumulator.

6. Method

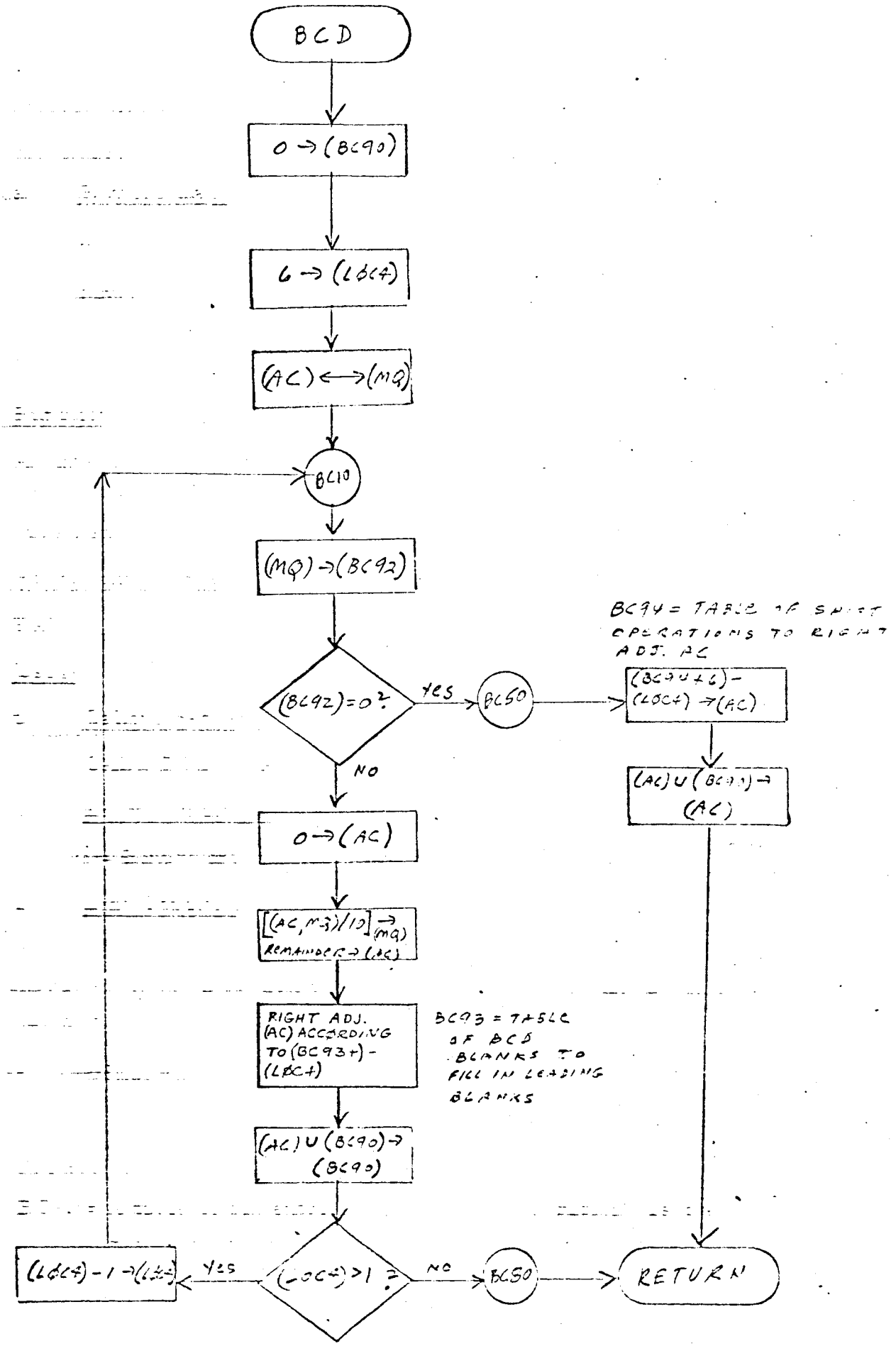
Each of the BCD characters in the accumulator is converted to a binary integer with leading blanks. The converted number is right adjusted in the accumulator.

7. Other Subroutines Used

None.

8. Using Subroutines

BCL, HEAD, HEADC.


BCD
0 → (BC90)
6 → (LOC4)
(AC) ⟷ (MQ)
BC10
(MQ) → (BC92)
(BC92) = 0 ?
YES
BC50
NO
BC94 = TABLE OF SHIFT OPERATIONS TO RIGHT ADJ. AC
(BC94+LC) - (LOC4) → (AC)
(AC) U (BC93) → (AC)
0 → (AC)
[(AC, MQ)/10] → (MQ)
REMAINDER → (AC)
RIGHT ADJ. (AC) ACCORDING TO (BC93+) - (LOC4)
BC93 = TABLE OF BCD BLANKS TO FILL IN LEADING BLANKS
(AC) U (BC90) → (BC90)
(LOC4) - 1 → (LOC4)
YES
(LOC4) > 1 ?
NO
BC50
RETURN

Program Description

1. Identification

a. Routine Label

BCL

b. Name

Convert an integer to BCD and push characters down into a list.

2. Function

An integer is converted to BCD and the characters are split up and pushed down into a list as individual elements. Blanks (leading zeros) are removed.

3. Programming System

FAP

4. Usage

a. Calling Sequence

CALL BCL (I, A)

b. Entry Conditions

I = Binary integer

c. Exit Conditions

The integer in I is converted to BCD. List A is created with each character of the BCD number pushed down into the list as an individual element.

d. Error Exits

None.

5. Definition of Identifiers

BC92 = A table of six entries, each a shift operation, used to right-adjust each character in the accumulator, after conversion from binary to BCD

6. Method

The binary to BCD conversion is performed as follows:

a. The integer in I is placed in the accumulator and converted to a six-character BCD number by the subroutine BCD. The BCD characters will be right adjusted (with leading blanks) in the accumulator.

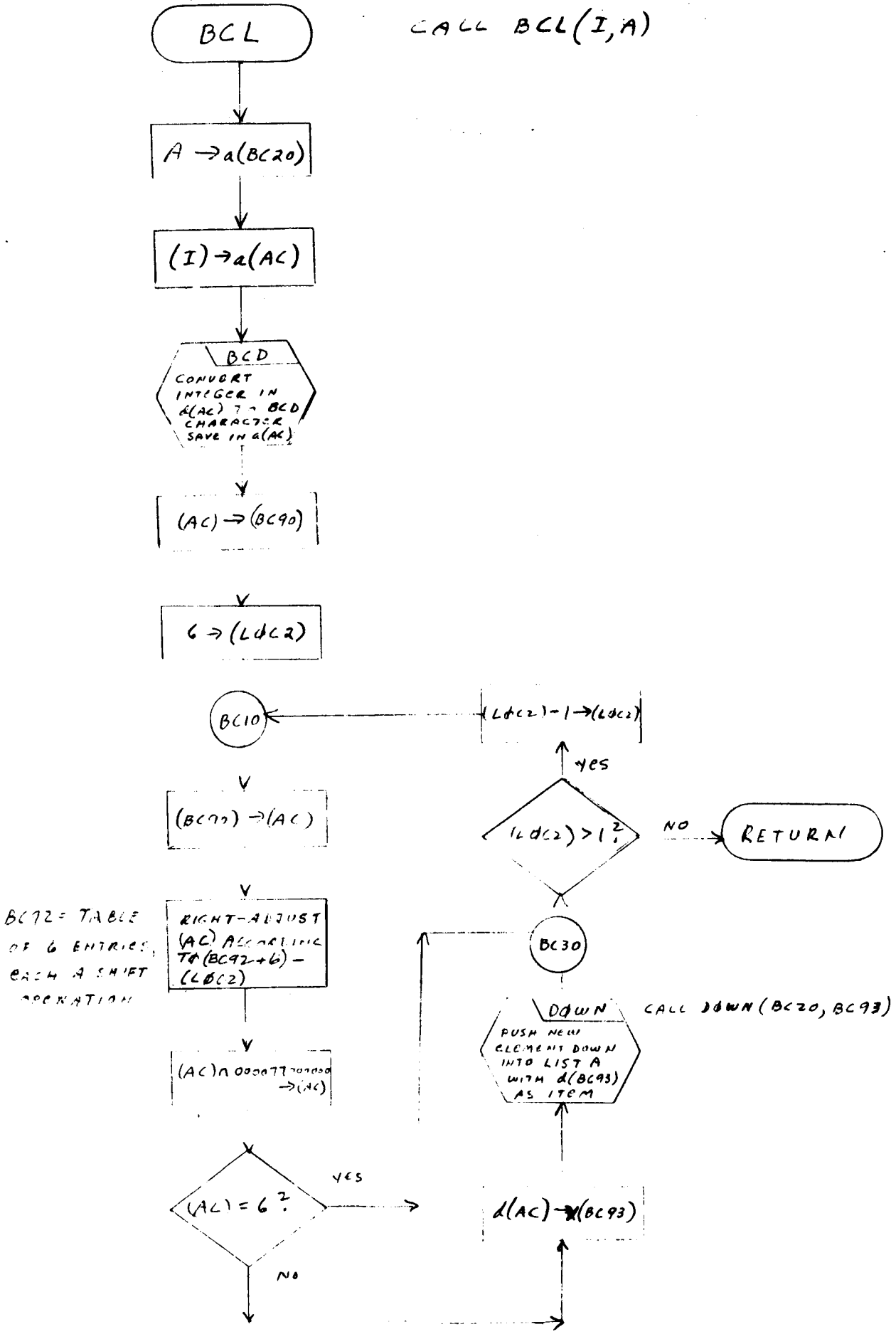
b. Each of the six characters is placed in the item of an element and pushed down into list A by the subroutine DOWN.

7. Other Subroutines Used

BCD

8. Using Subroutines

COMBN, DIMEN, DPDST, INPUTX, NLINDM, PARTS, SYMCRD, WRTEQ, XEROX, Main Program for Pass 2 of TAG Preprocessor.


BCL
CALL BCL(I,A)
A → a(BC20)
(I) → a(AC)
BCD
CONVERT INTEGER IN a(AC) TO BCD CHARACTER SAVE IN a(AC)
(AC) → (BC90)
6 → (LØC2)
BC10
(LØC2) - 1 → (LØC2)
yes
(BC77) → (AC)
(LØC2) > 1 ?
NO
RETURN
BC72 = TABLE OF 6 ENTRIES, EACH A SHIFT OPERATION
RIGHT-ADJUST (AC) ACCORDING TO (BC92+6) - (LØC2)
BC30
DOWN
CALL DOWN(BC20, BC93)
PUSH NEW ELEMENT DOWN INTO LIST A WITH a(BC93) AS ITEM
(AC) ∧ 000077777000 → (AC)
(AC) = 6 ?
YES
a(AC) → a(BC93)
NO

Program Description

1. Identification

   a. Routine Label

      BND2

2. Function

   This subroutine decomposes an integer into a units digit and a tens digit.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL BND2 (I, J, K)

   b. Entry Conditions

      I = Binary integer

   c. Exit Conditions

      J = Units digit of integer I
      K = Tens digit of integer I

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

   The units and tens digits of integer I are extracted as follows:

   a. I is placed in the MQ.
   b. AC is cleared to zero.
   c. I is shifted into the address portion of AC.

d. AC, MQ are divided by 10.

(1) The remainder is shifted into the decrement of AC and then stored into the decrement of J, as the units digit of I.

e. The AC is cleared to zero.

f. AC and integral part of quotient in MQ (from division in step d) are divided by 10.

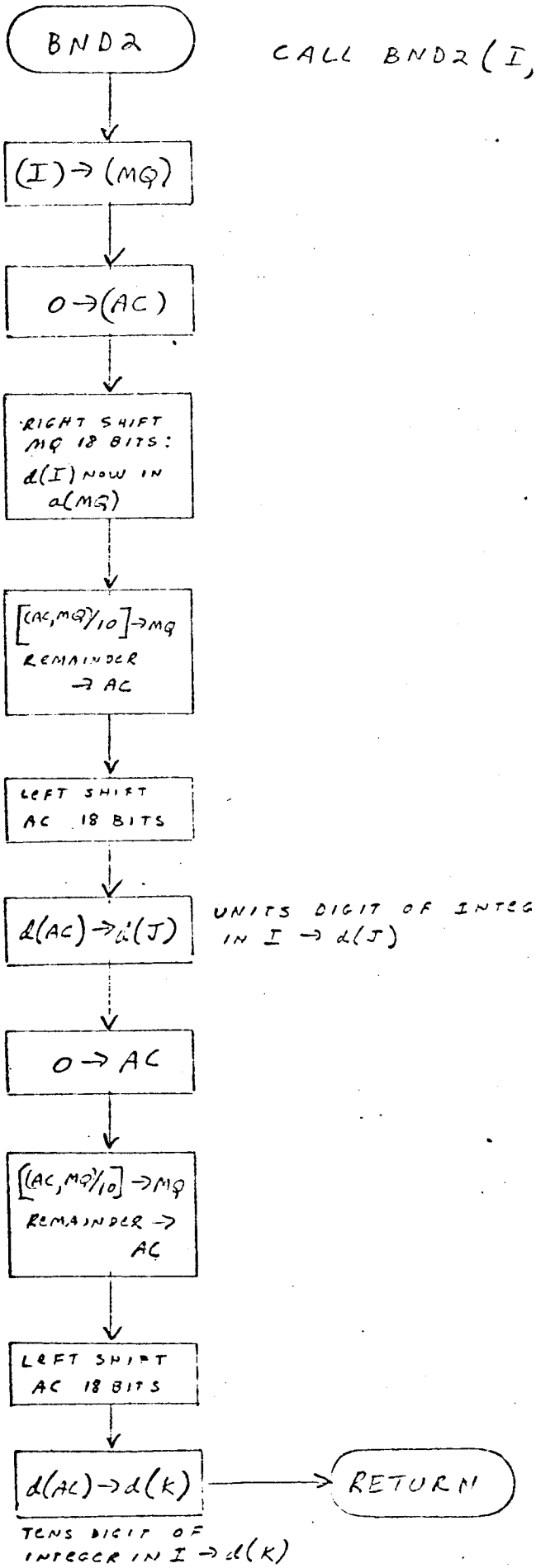
(1) The remainder is shifted into the decrement of AC and then stored into the decrement of K, as the tens digit of I.

7. Other Subroutines Used

None.

8. Using Subroutines

DONBD


BND2
CALL BND2 (I, J, K)
(I) → (MQ)
0 → (AC)
RIGHT SHIFT MQ 18 BITS: d(I) NOW IN a(MQ)
[(AC, MQ)/10] → MQ
REMAINDER → AC
LEFT SHIFT AC 18 BITS
d(AC) → d(J)
UNITS DIGIT OF INTEGER IN I → d(J)
0 → AC
[(AC, MQ)/10] → MQ
REMAINDER → AC
LEFT SHIFT AC 18 BITS
d(AC) → d(K)
RETURN
TENS DIGIT OF INTEGER IN I → d(K)

Program Description

1. Identification

a. Routine Label

CHLNE

b. Name

Insert a character into an array.

2. Function

This subroutine inserts a BCD character into a given character position of a Hollerith array.

3. Programming System

FAP

4. Usage

a. Calling Sequence

CALL CHLNE (A, I, J)

b. Entry Conditions

A = An array of Hollerith characters
I = A character position in the array
J = A word that contains a BCD character

c. Exit Conditions

A BCD character, as an integer in J, is inserted into character position I of the array A. The rest of A is unchanged.

d. Error Exits

None.

5. Definition of Identifiers

CH90 = This cell contains the BCD character in J as an integer (right-adjusted in the decrement portion).

CH92 = A table with six entries, each a shift operation.

6. Method

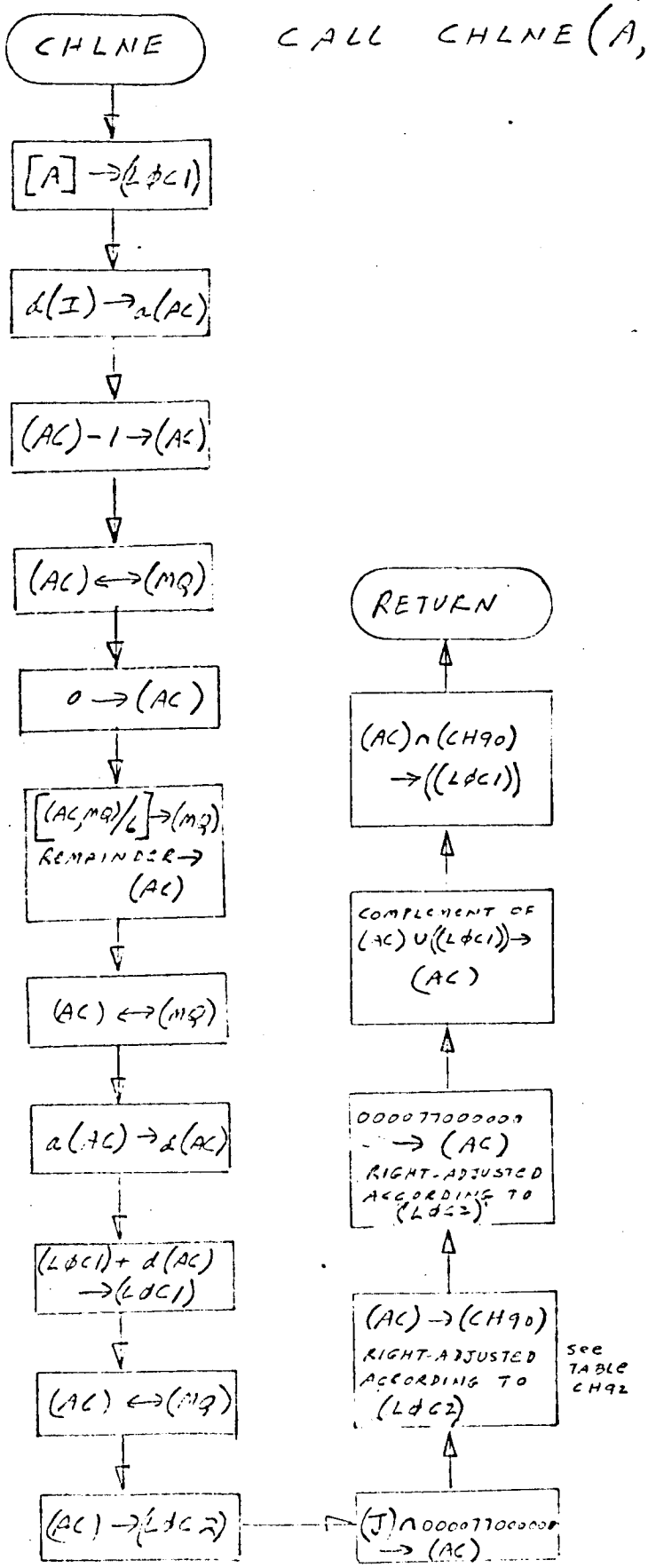
Each time the subroutine is entered, the BCD character in J is shifted into the decrement of CH90 as an integer, right-adjusted (CH92 determines the appropriate shift). The contents of CH90 are then "OR-ED" into the Ith position of the Hollerith array A.

7. Other Subroutines Used

None.

8. Using Subroutines

DBPCHC, DBPCH, DBPFH, GOBLE, INPUTX, MATOT, PUSPCH.


CHLNE
CALL CHLNE(A, I, J)
[A] → (LØC1)
d(I) → d(AC)
(AC) - 1 → (AC)
(AC) ↔ (MQ)
0 → (AC)
[(AC,MQ)/d] → (MQ)
REMAINDER → (AC)
(AC) ↔ (MQ)
d(AC) → d(AC)
(LØC1) + d(AC) → (LØC1)
(AC) ↔ (MQ)
(AC) → (LØC2)
(J) ∩ 000077000000 → (AC)
(AC) → (CH90)
RIGHT-ADJUSTED ACCORDING TO (LØC2)
see TABLE CH92
000077000000 → (AC)
RIGHT-ADJUSTED ACCORDING TO (LØC2)
COMPLEMENT OF (AC) ∪ ((LØC1)) → (AC)
(AC) ∩ (CH90) → ((LØC1))
RETURN

Program Description

1. Identification

   a. Routine Label

      COMBN

2. Function

The abstract quantity descriptors SYN1N2 are transformed into a linear combination of existing data items which will compute the quantities in terms of existing transforms.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL COMBN (RIN, T, N, ROUT, NPT, NCB, IV34, ISSW, ISIS)

   b. Entry Conditions

RIN	=	A simple list containing, in order, the characters representing the sequence of descriptors
T	=	A four-dimensional matrix represented in list-structure form
N	=	The number of nodes
NPT	=	The array containing npt dimension information
ISSW	=	A switch
ISSW	=	1 if processing a nonlinear statement
ISSW	=	2 if processing a standard statement

c. Exit Conditions

ROUT	=	A simple list which contains the character string representing the appropriate transformation
NCB	=	The number of combined data values in ROUT
IV34	=	An indicator computed from ISSW and RIN
ISIS	=	An indicator computed from RIN

d. Error Exits

None.

5. Definition of Identifiers

Constants: BCD character strings

Type	Name	Value
Integer	IBRKL	(
Integer	IBRKR	)
Integer	ISS	S
Integer	IPLUS	+
Integer	F	F
Integer	IMINUS	-
Integer	ITYPE(1)	V
Integer	ITYPE(2)	S
List	Zero	[+;0;.;0]
List	PNTST	[.;*]

Variables:

KN	=	Character counter used to maintain position when popping up RIN
KS	=	Indicator of sign of abstract variable
LTYPE	=	Holds second character of abstract variable
IO	=	Holds first node number
IT	=	Holds second node number
NSP	=	Computed index

NSIJ = Computed index

TEM = List used for construction of the output string

TEMX = List used to hold second node numbers temporarily

ITEM = Used to hold character most recently popped out of RIN

SI(2,100) = Holds numerical values as they are computed from the abstract variable

6. Method

a. RIN contains a linear combination of abstract variables, each of which has six characters:

Ch1 = S

Ch2 = (LTYPE)

Ch3, Ch4 } First node-number

Ch5, Ch6 } Second node-number

Example:

SV0304

RIN may contain any linear combination of these.

Example:

SV0304 + SV0302 - SV0102

b. For each variable ±SYN1N2, the list T is searched, using SNATCH, for any non-zero data. This is done for all $0<i\leq N$, and the data is computed this way:

$$X(i) = \pm (T(N2, i, 0, 0) - T(N1, i, 0, 0))$$

c. For any X(i) that is non-zero, an entry is pushed into TEM which has the following format:

$$\pm Z * Fyj1(k)$$

where Z = the BCD representation of X(i), as a floating-point number

y = the character <V> if the second character of the abstract variable was a V; otherwise y is the character <S>

j and k = BCD representations of integers computed in the following manner:

given $X(i) \neq 0$, J is such that

$$\sum_{m=0}^{j-1} NPT(m) < i \leq \sum_{m=0}^{j} NPT(m) \qquad NPT(o) = 0$$

and $$K = i - \sum_{m=0}^{j} NPT(m)$$

This means that j will indicate the submatrix (V, C, G...) where i occurs, and k will be the index within that submatrix.

d. TEM is popped up and pushed down into ROUT; then exit. ROUT will then have the transformed combination, with the sign and high-order digits of the floating-point number at the top of the list.

e. If TEM is null, then either RIN was null or no non-zero X(i) could be found. A comment "ZERO HAS BEEN PLACED IN THE STATEMENT BELOW" will be printed, and the contents of list ZERO pushed into ROUT.

f. NCB will contain the number of entries

$$\pm Z * F_{yj1}(K)$$

inserted in ROUT.

ISIS will be set = 2 if y was not a V; unchanged otherwise.

IV34 will be set = 1 under these conditions:

(1) c(ISSW) ≤ 1
j = 4

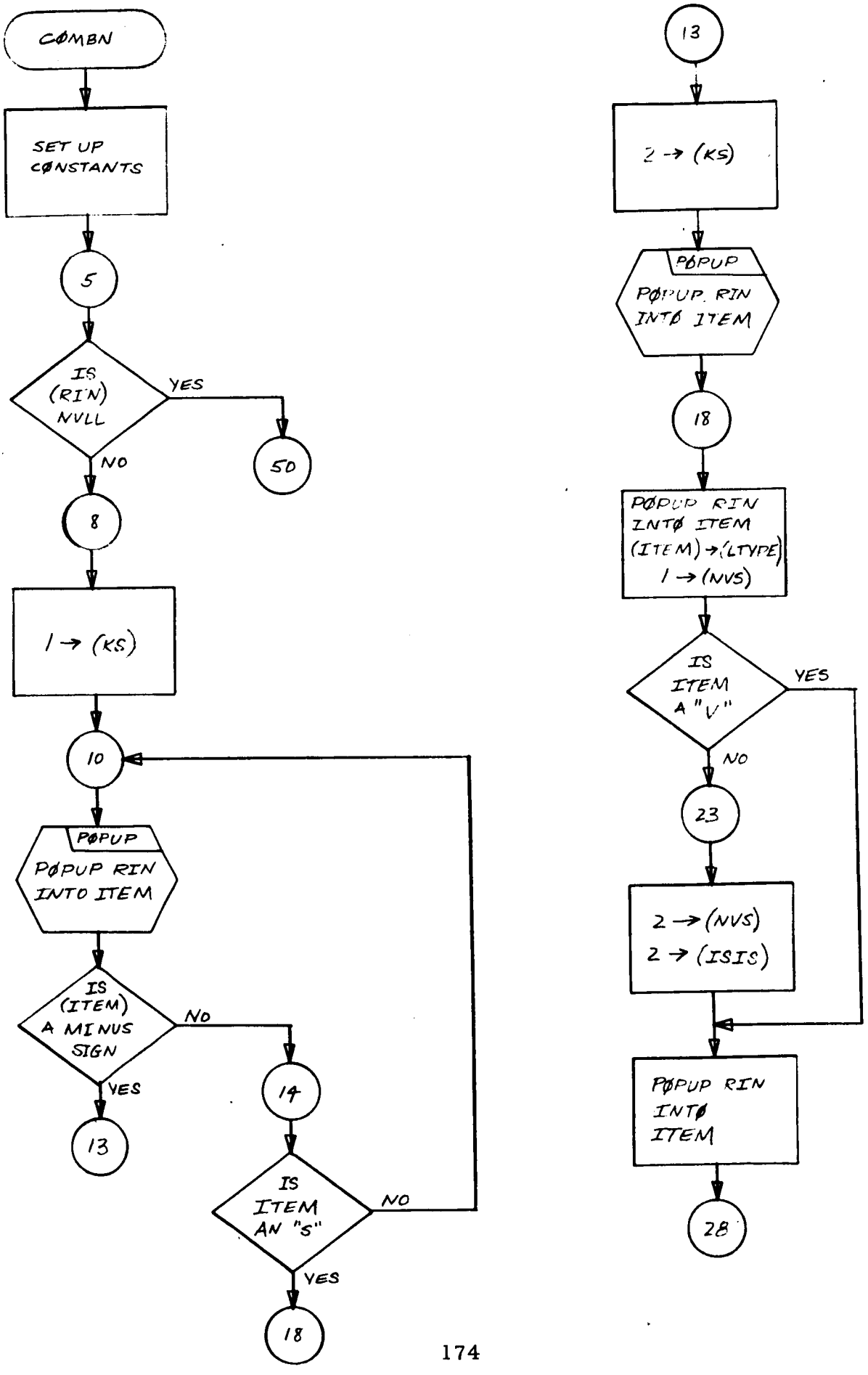
or (2) c(ISSW) ≤ 1
j = 3
y is a V

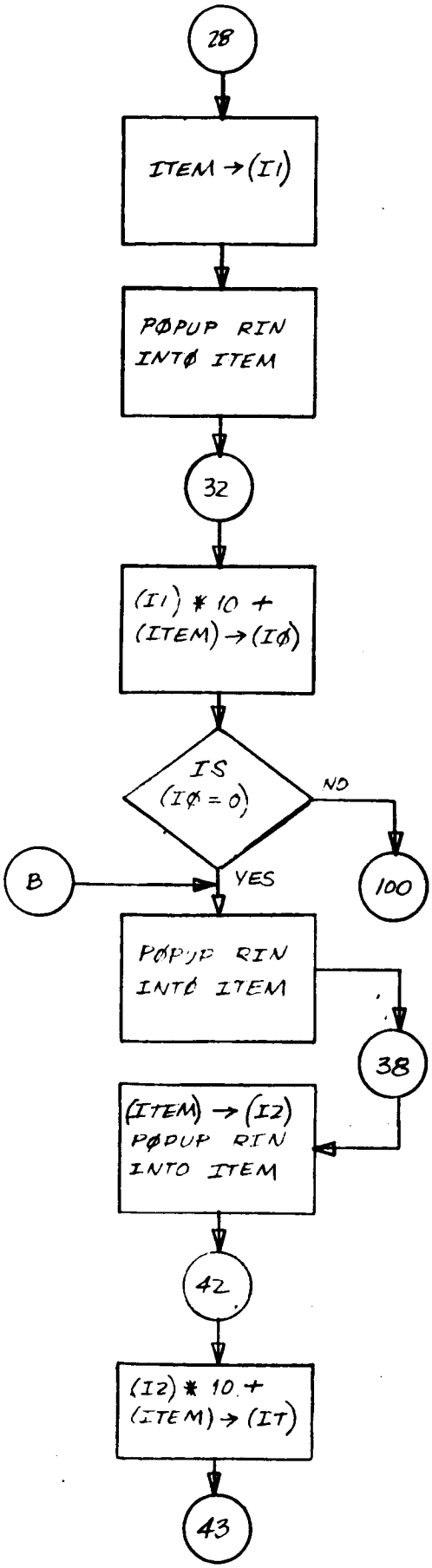
7. Other Subroutines Used

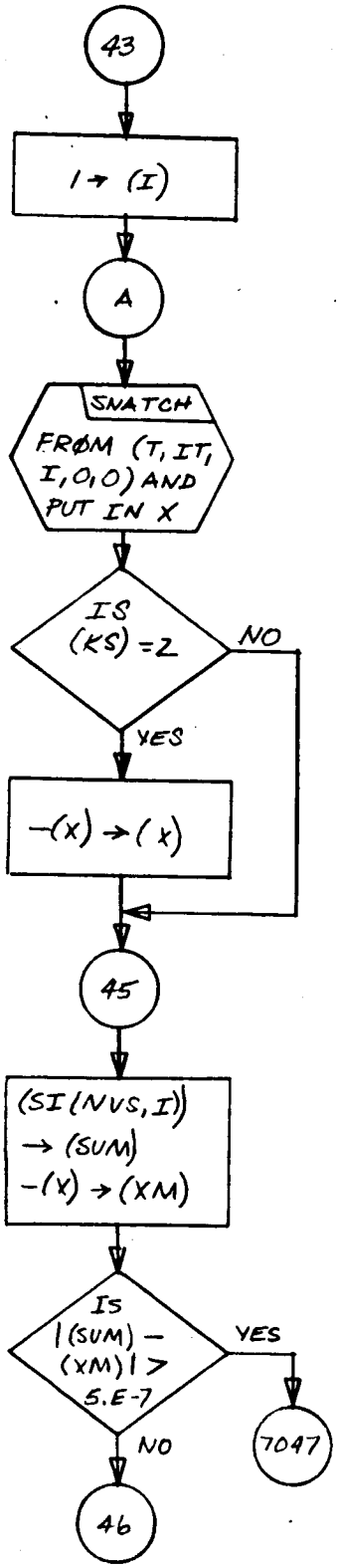
BCL, DIFA, DOWN, DOWNS, DUMP, ERASE, FLTCON, POPUP, SNATCH, SYMBL, UPDWN.

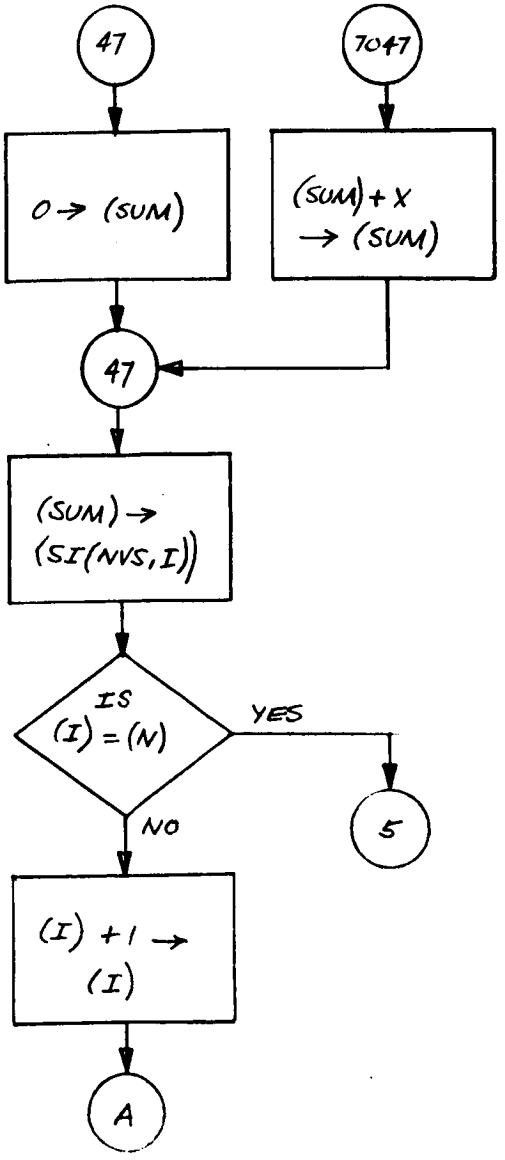
8. Using Subroutines

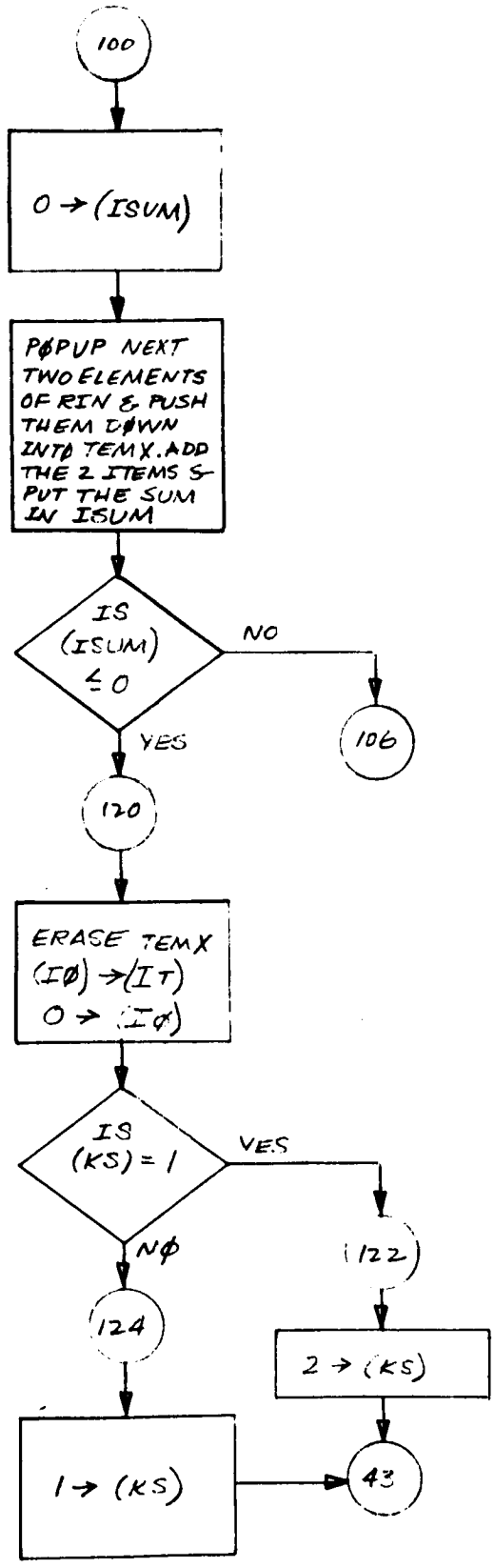
SUBST

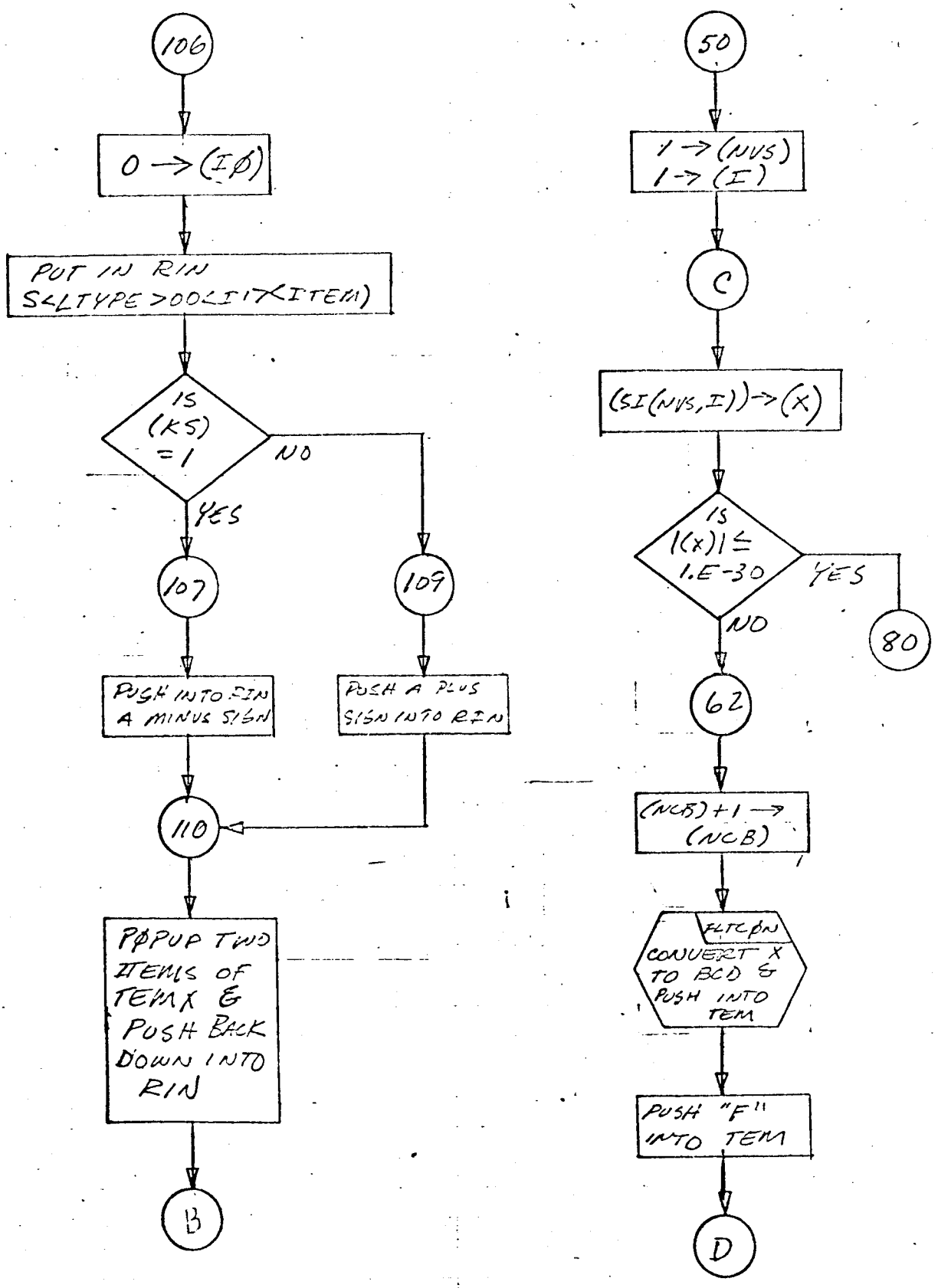

CØMBN
SET UP CØNSTANTS
5
IS (RIN) NVLL
YES
50
NO
8
1 → (KS)
10
PØPUP
PØPUP RIN INTO ITEM
IS (ITEM) A MINUS SIGN
NO
YES
13
14
IS ITEM AN "S"
NO
YES
18
13
2 → (KS)
PØPUP
PØPUP RIN INTØ ITEM
18
PØPUP RIN INTØ ITEM
(ITEM) → (LTYPE)
1 → (NVS)
IS ITEM A "V"
YES
NO
23
2 → (NVS)
2 → (ISIS)
PØPUP RIN INTØ ITEM
28

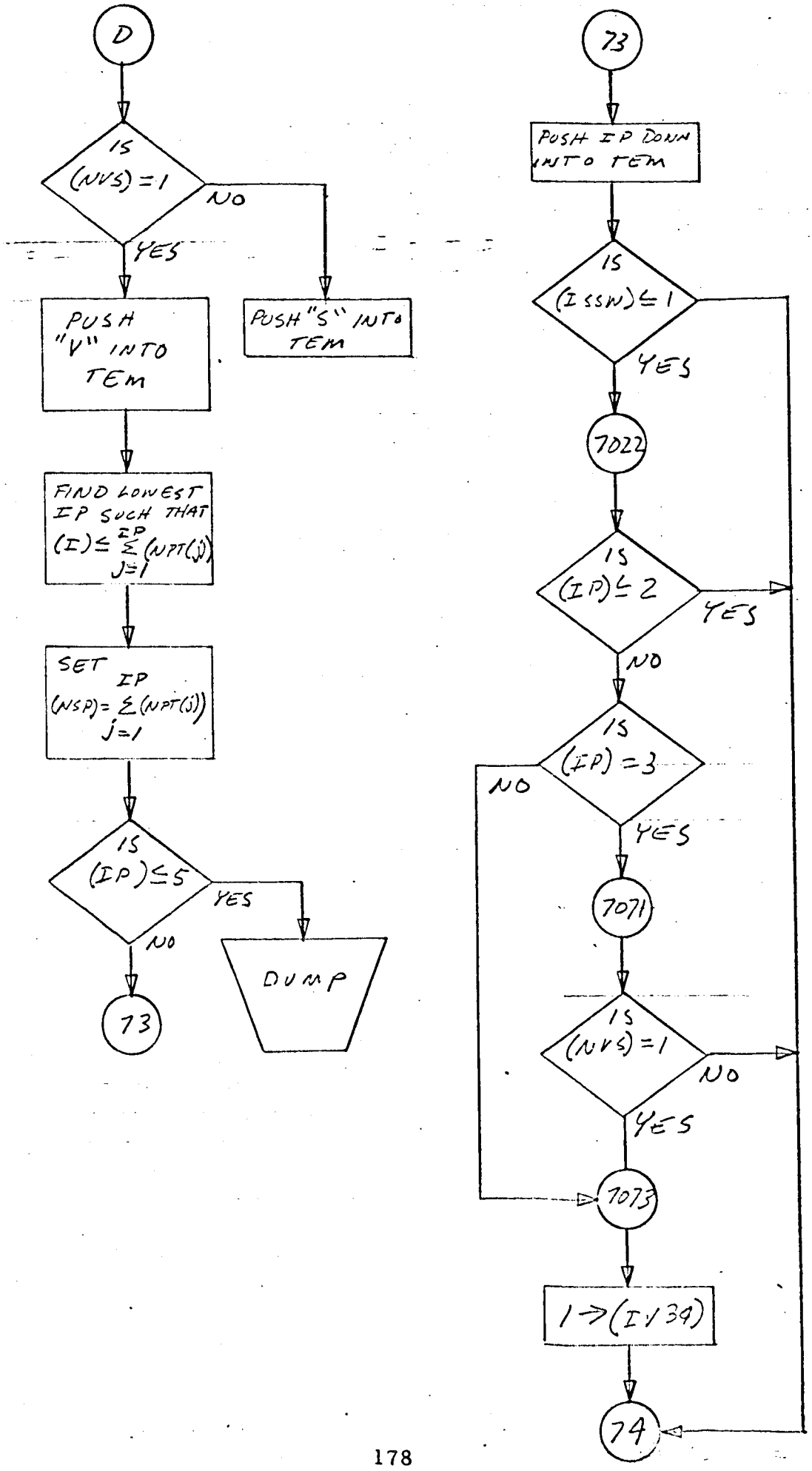

28
ITEM → (I1)
PØPUP RIN INTØ ITEM
32
(I1) * 10 + (ITEM) → (IØ)
IS (IØ = 0)
NO
YES
B
100
PØPUP RIN INTØ ITEM
38
(ITEM) → (I2) PØPUP RIN INTO ITEM
42
(I2) * 10 + (ITEM) → (IT)
43

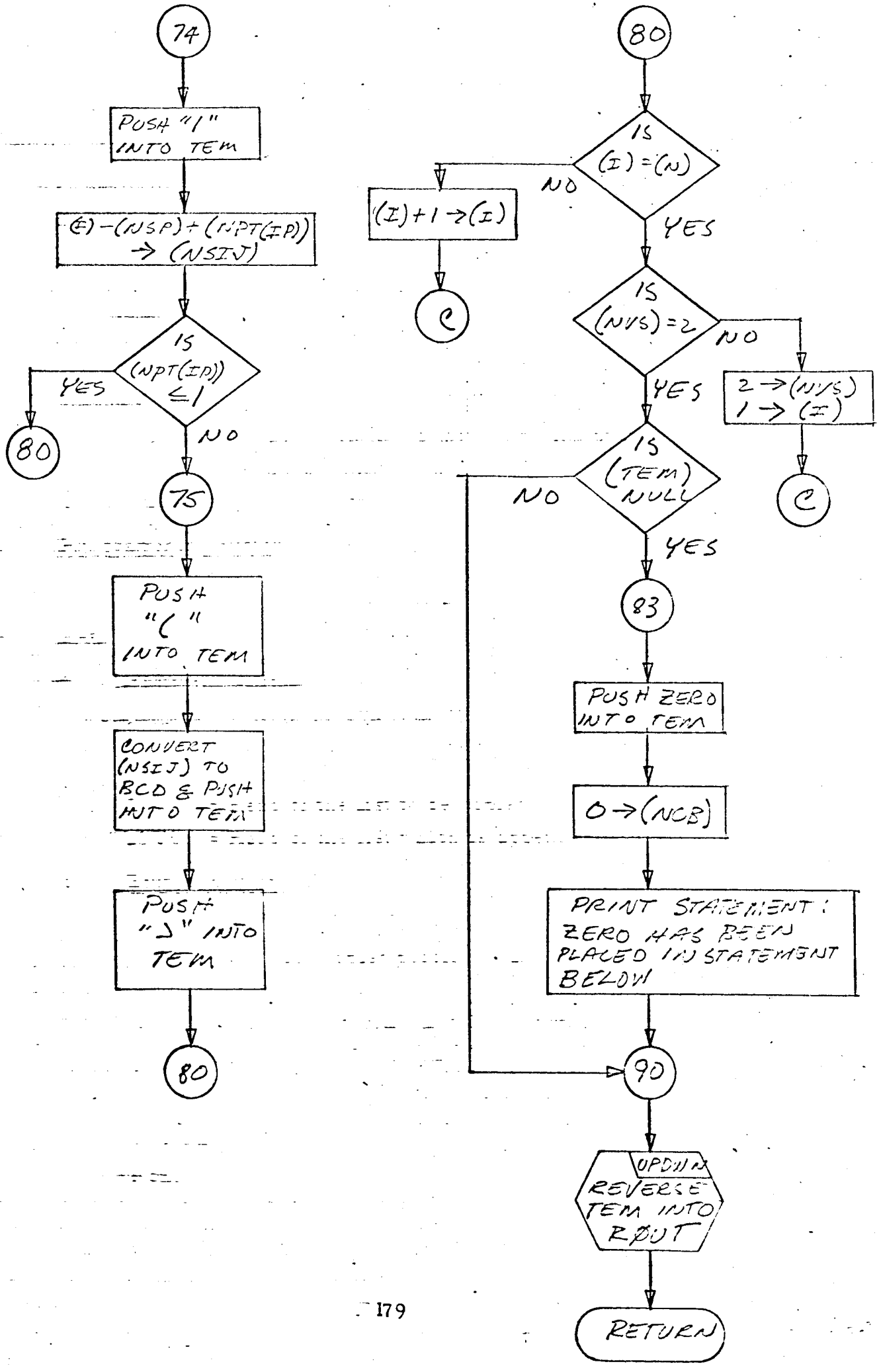

43
1 → (I)
A
SNATCH FRØM (T, IT, I, 0, 0) AND PUT IN X
IS (KS) = 2
NO
YES
-(X) → (X)
45
(SI(NVS, I)) → (SUM) -(X) → (XM)
Is |(SUM) - (XM)| > 5.E-7
YES
NO
7047
46


47
7047
0 → (SUM)
(SUM) + X → (SUM)
47
(SUM) → (SI(NVS,I))
IS (I) = (N)
YES
NO
5
(I) + 1 → (I)
A


100
0 → (ISUM)
PØPUP NEXT TWO ELEMENTS OF RIN & PUSH THEM DØWN INTØ TEMX. ADD THE 2 ITEMS & PUT THE SUM IN ISUM
IS (ISUM) ≤ 0
NO
YES
106
120
ERASE TEMX
(IØ) → (IT)
0 → (IØ)
IS (KS) = 1
YES
NØ
122
124
2 → (KS)
1 → (KS)
43


106
0 → (IØ)
PUT IN RIN
S<LTYPE>00LIT<ITEM)
IS (KS) = 1
NO
YES
107
109
PUSH INTO RIN A MINUS SIGN
PUSH A PLUS SIGN INTO RIN
110
PØPUP TWO ITEMS OF TEMX & PUSH BACK DOWN INTO RIN
B
50
1 → (NVS)
1 → (I)
C
(SI(NVS,I)) → (X)
IS |(X)| ≤ 1.E-30
YES
NO
80
62
(NCB) + 1 → (NCB)
FLTCØN
CONVERT X TO BCD & PUSH INTO TEM
PUSH "F" INTO TEM
D


D
IS (NVS) = 1
NO
YES
PUSH "V" INTO TEM
PUSH "S" INTO TEM
FIND LOWEST IP SUCH THAT (I) ≤ Σ (NPT(J)) J=1 IP
SET (NSP) = Σ (NPT(J)) j=1 IP
IS (IP) ≤ 5
YES
NO
DUMP
73
73
PUSH IP DOWN INTO TEM
IS (ISSN) ≤ 1
YES
7022
IS (IP) ≤ 2
YES
NO
IS (IP) = 3
NO
YES
7071
IS (NVS) = 1
NO
YES
7073
1 → (IV 39)
74


74
PUSH "/" INTO TEM
(E)-(NSP)+(NPT(IP)) → (NSIJ)
IS (NPT(IP)) ≤1
YES
NO
80
75
PUSH "(" INTO TEM
CONVERT (NSIJ) TO BCD & PUSH INTO TEM
PUSH ")" INTO TEM
80
80
IS (I)=(N)
NO
YES
(I)+1→(I)
C
IS (NVS)=2
NO
YES
2→(NVS)
1→(I)
C
IS (TEM) NULL
NO
YES
83
PUSH ZERO INTO TEM
0→(NCB)
PRINT STATEMENT: ZERO HAS BEEN PLACED IN STATEMENT BELOW
90
UPDWN
REVERSE TEM INTO RØUT
RETURN

Program Description

1. Identification

   a. Routine Label

      COPY

   b. Name

      Copy a list and append a second list to the copied list.

2. Function

A copy is made of the list IN and to this copy is appended the list IOUTF. IOUTF is reset as the head of this new list, and IOUTL points to the last element of the copied portion of the new list.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL COPY (IN, IOUTF, IOUTL)

   b. Entry Conditions

      IN = Head of the list to be copied
      IOUTF = Head of the list which is appended to the copy of IN

   c. Exit Conditions

      (1) If IOUTF is null, IOUTF is set to point to the first element of the copied portion of IN, and IOUTL is set to point to to the last element of the copy of IH.
      (2) If IOUTF is not null, IOUTF is appended to the copy of IN, and IOUTF and IOUTL are set as described in step (1) above.

   d. Error Exits

      None.

5. Definition of Identifiers

IOUT = A local variable which is the head cell of the copied portion of list IN

6. Method

The list IN is copied, one element at a time, into a new list IOUT. When all of the elements of IN have been copied, a test is made to determine if IOUTF (the list to be appended to IOUT) is null.

a. If IOUTF is null, set IOUTF and IOUTL as described in step c.

b. If IOUTF is not null, append list IOUTF to list IOUT and proceed at step c.

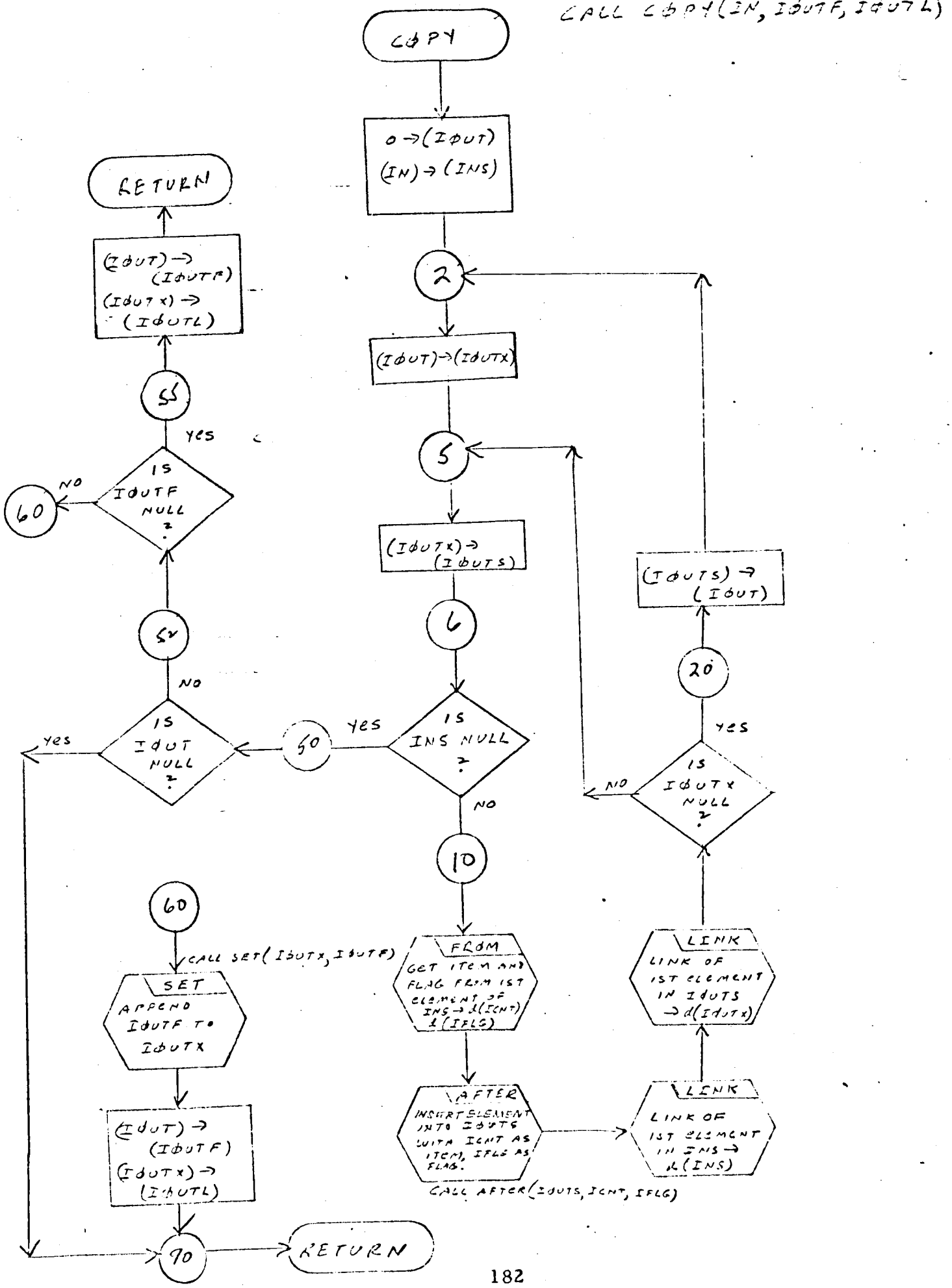
c. Set IOUTF to point to the first element of IOUT and IOUTL to point to the last element of IOUT. Then exit from the routine.

7. Other Subroutines Used

AFTER, FROM, LINK, SET.

8. Using Subroutines

DIMEN, EXCPT, INPUTX, POWER, RAISE, RECOVR, SUBST, WRTEQ, ZEROX, Main Program for Pass 2 of TAG Preprocessor.


CALL COPY(IN, IOUTF, IOUTL)
COPY
0 → (IOUT)
(IN) → (INS)
2
(IOUT) → (IOUTX)
5
(IOUTX) → (IOUTS)
6
IS INS NULL ?
YES
50
NO
10
FROM
GET ITEM AND FLAG FROM 1ST ELEMENT OF INS → I(ICNT), I(IFLG)
AFTER
INSERT ELEMENT INTO IOUTS WITH ICNT AS ITEM, IFLG AS FLAG.
CALL AFTER(IOUTS, ICNT, IFLG)
LINK
LINK OF 1ST ELEMENT IN IOUTS → I(IOUTX)
LINK
LINK OF 1ST ELEMENT IN INS → I(INS)
IS IOUTX NULL ?
NO
YES
20
(IOUTS) → (IOUT)
IS IOUT NULL ?
YES
NO
55
IS IOUTF NULL ?
NO
YES
60
CALL SET(IOUTX, IOUTF)
SET
APPEND IOUTF TO IOUTX
(IOUT) → (IOUTF)
(IOUTX) → (IOUTL)
90
RETURN
RETURN

Program Description

1. Identification

   a. Routine Label

      COTRN

   b. Name

      Form the tree voltage to node voltage transformation matrix.

2. Function

This subroutine uses the sorted tree branch list, FLIST, to form the voltage coordinate transformation matrix TC, as a two-dimensional array-list, such that VN] = [TL] * VT] .

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL COTRN (FLIST, TL, NM)

      TL = Local variable for the TC matrix used in the main routine of Pass 1.

   b. Entry Conditions

      FLIST = A type D list representation of the tree branch descriptions, reordered to VCGLNI sequence.

      NM = The maximum node numbers in the TAG connection list.

   c. Exit Conditions

TL contains a two-dimensional array-list representation of the voltage coordinate transformation matrix TC. TC has NM rows and NM columns. Entries in TC are either +1 or 0. Each row in TC represents a path in the tree, from a particular node to the ground (zero) node. Rows and row numbers correspond directly to nodes and node numbers.

d. Error Exits

When FLIST has been traced through without finding a tree branch whose positive node number equals I, CALL DUMP is executed and control returns to the FORTRAN monitor system.

5. Definition of Identifiers

I = Node numbers

L = Position of the tree branch descriptor in FLIST, counting from the top down

K = Number of some node

Xl = Local variable that corresponds to FLIST

6. Method

Starting at node I = 1 and proceeding to I = NM, a path is traced in the FLIST tree between each node I and node 0. Because of the nature of the FLIST tree (every node has a single tree branch positively incident on it except node zero, which has none), the tree branch voltages in each path will add with a coefficient of +1 to equal the respective node voltages.

For each node number (I = 1, 2, ---NM) corresponding to a row of TL, the column entries of TL are generated as follows. The columns of TL (L = 1, 2, ---NM) are ordered to correspond to the branches of the tree whose descriptors are stored in VCGLNI order in FLIST.

a. K is set to node number (I).

b. Trace through FLIST for a tree branch whose positive node number (NP) equals K. Record its position in FLIST as L.

c. When such a branch descriptor is found, a +1 is placed in row I, column L of the TL matrix.

d. K is reset to equal the negative node number (NN) of the tree branch found in step c.

(1) If K = 0, the path from node I to ground has been completely traced in row I of TL. If the row number I (corresponding to node I) is the same as the maximum node number, NM, the TL matrix is complete, and return is made from the routine. If the maximum node

number has not been processed, the node number I is increased by 1, K is reset to the new node number I, and the search is started again at step b.

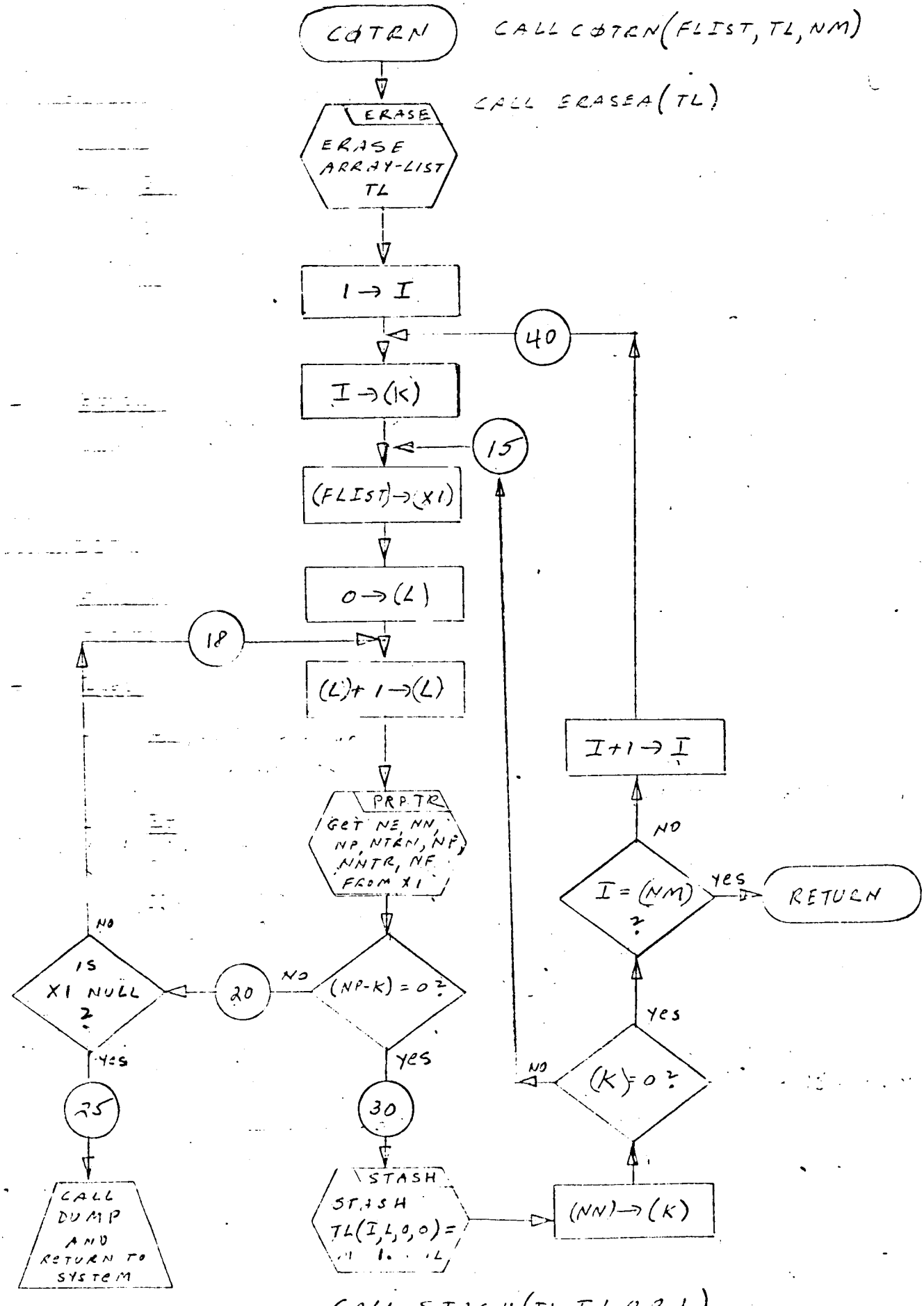
(2) If $K \neq 0$, the path from node I to ground is not complete, and the search through FLIST continues (at step b) until the ground node has been reached for the node number (I) in process.

7. Other Subroutines Used

ERASEA, PRPTR, STASH.

8. Using Subroutines

Main Program for Pass 1 of TAG preprocessor.


CØTRN
CALL CØTRN(FLIST, TL, NM)
CALL ERASEA(TL)
ERASE
ERASE ARRAY-LIST TL
1 → I
40
I → (K)
15
(FLIST) → (X1)
0 → (L)
18
(L) + 1 → (L)
PRPTR
Get NE, NN, NP, NTRN, NF, NNTR, NF FROM X1
I + 1 → I
NO
I = (NM) ?
yes
RETURN
NO
IS X1 NULL ?
20
NO
(NP-K) = 0 ?
yes
yes
(K) = 0 ?
NO
yes
25
30
CALL DUMP AND RETURN TO SYSTEM
STASH
STASH TL(I,L,0,0) = 1.
(NN) → (K)
CALL STASH(TL, I, L, 0, 0, 1.)

Program Description

1. Identification

a. Routine Label

DBPCHC

b. Name

Output a series of card images, including comment cards, from a list of characters in reverse order.

2. Function

This subroutine is exactly like DBPCH, except for comment cards. If IC1 is a C, then IZ is ignored and card text starts in column 2. If IC1 is not a C, IC1 is put in column 1 and the card image is constructed as in DBPCH.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL DBPCHC (P, IZ, IC1)

b. Entry Conditions

P = A list of BCD characters in reverse order
IZ = Statement number
IC1 = Comment card flag word

c. Exit Conditions

(1) For non-comment cards (IC1 $\neq$ C), the BCD string in P is inverted and output as a series of card images, with the statement number in columns 2-5, the continuation card number (if any) in column 6, and the character in IC1 in column 1. The card text is in columns 7-72.

(2) For comment cards (IC1 = C), the BCD string in P is inverted and output as a series of card images with "C" in column 1 and card text in columns 2-72.

(3) Output is on a print tape, a punch tape, and a save tape.

d. Error Exits

None.

5. Definition of Identifiers

Z = A temporary head cell for the list of BCD characters output in A

A = Output buffer

6. Method

The list of BCD characters in list P is output as follows:

a. List P is tested:

(1) If P is null, step (2) is skipped. Continue at step b.

(2) If P is not null, the elements in P are popped up, one at a time, then pushed down into Z, thereby restoring the order of the characters in P.

b. IC1 is tested:

(1) If IC1 is a C, IZ is ignored. The card image being constructed is a comment card and will start in column 2 of buffer A (at step c).

(2) If IC1 is not a C, the statement number in IZ is placed in columns 2-5 of buffer A and the continuation card number, if any, is placed in column 6 of buffer A. The character in IC1 is placed in column 1 of buffer A.

c. The BCD characters in list Z are popped up, one at a time, and placed in columns 2-72 (if a comment card) or columns 7-72 (if not a comment card) of buffer A.

d. Buffer A is output as a record of 13 BCD words, onto a print tape, a save tape, and a punch tape.

e. Z is tested:

(1) If Z is null, exit is made from the routine.

(2) If Z is not null, and continuation cards have not exceeded 10 (for non-comment cards), processing continues at step b(2).

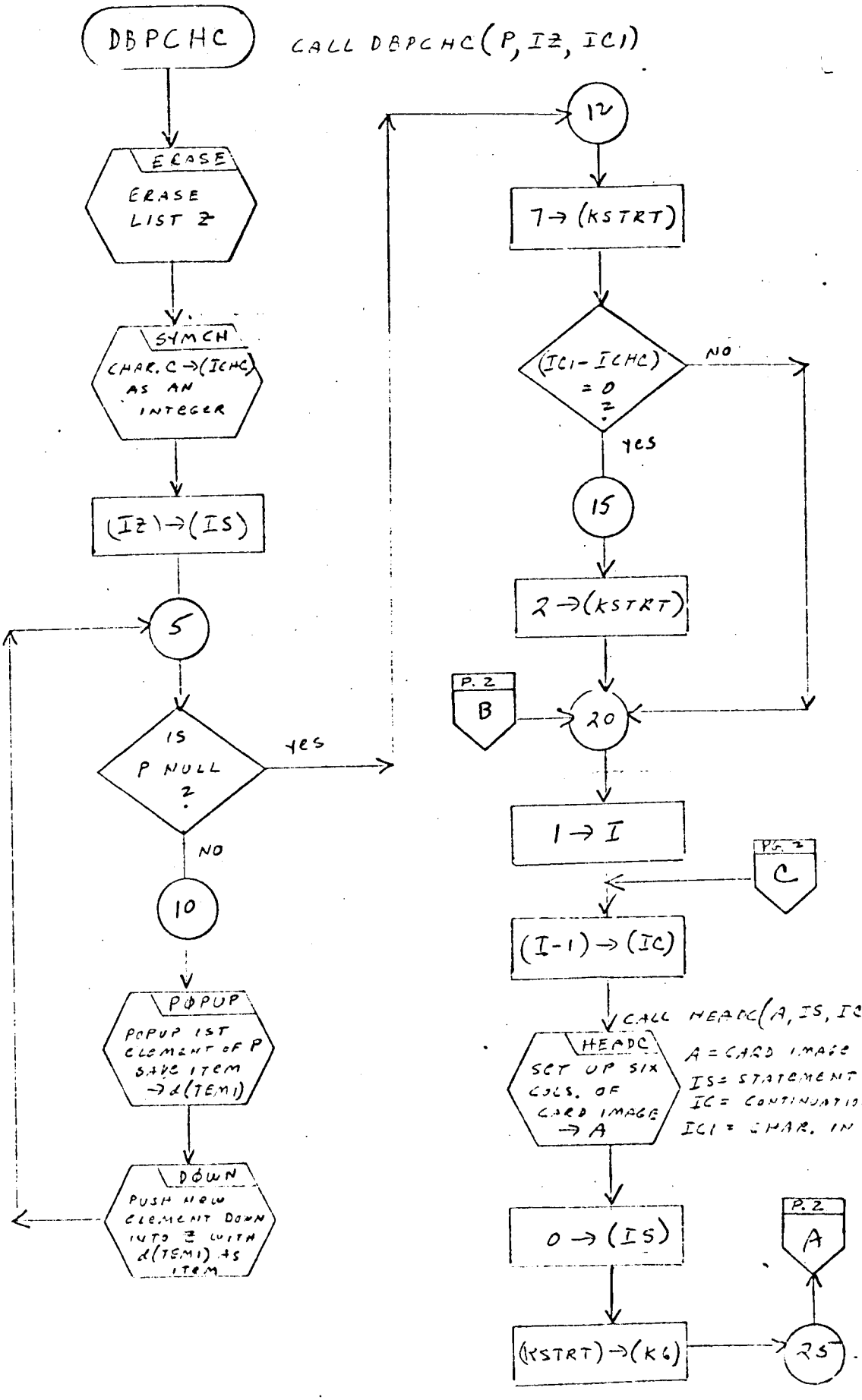
(3) If Z is not null, and 10 continuation cards have been output (for non-comment cards), the continuation card index is re-initialized and the process starts over at step b(2).

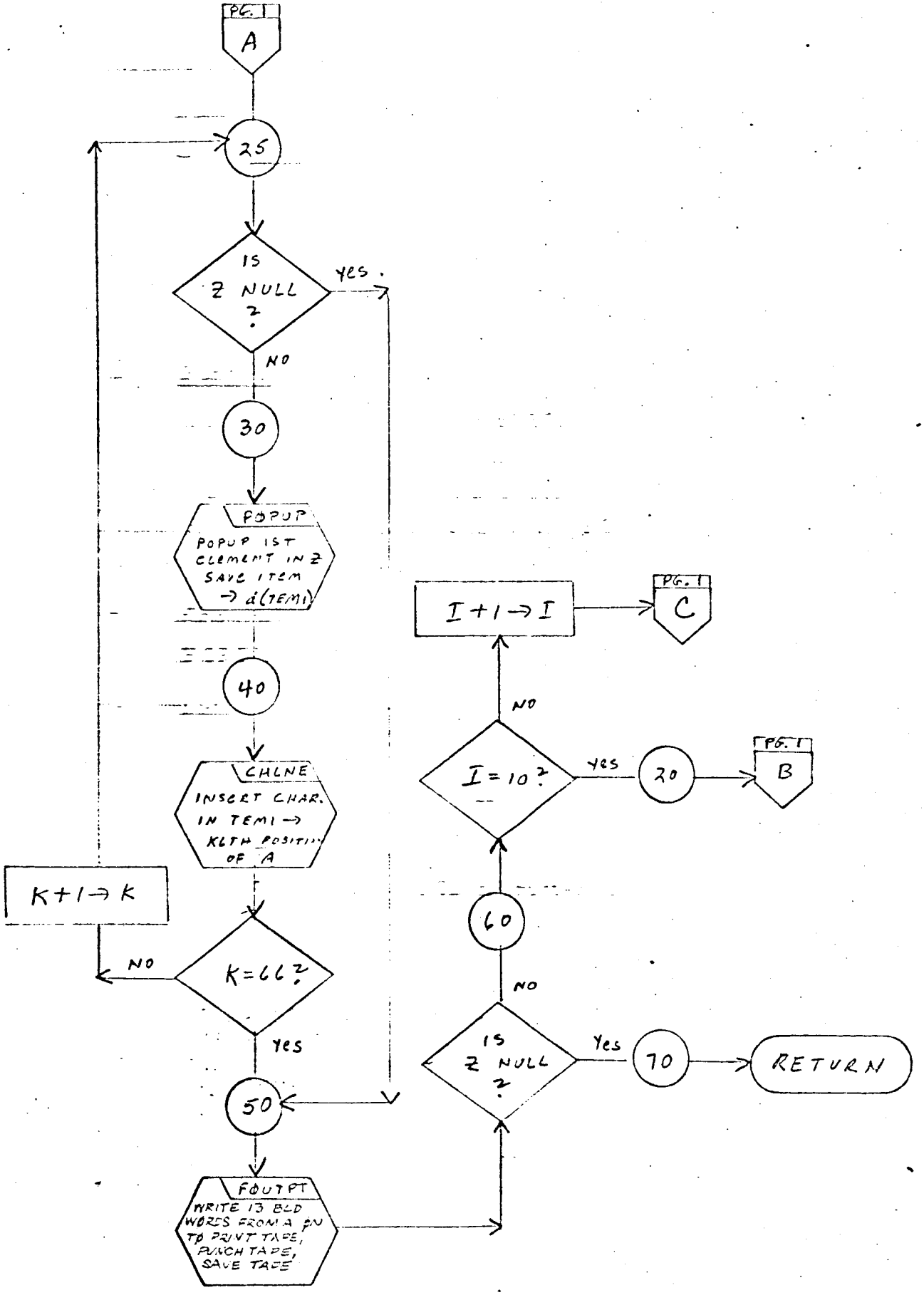
7. Other Subroutines Used

CHLNE, DOWN, ERASE, FOUTPT, HEADC, POPUP, SYMCH.

8. Using Subroutines

WRTEQ, Main Program for Pass 2 of TAG Preprocessor.


DBPCHC
CALL DBPCHC(P, IZ, ICI)
ERASE
ERASE LIST Z
SYMCH
CHAR. C → (ICHC) AS AN INTEGER
(IZ) → (IS)
5
IS P NULL Z.
yes
NO
10
PØPUP
POPUP 1ST ELEMENT OF P SAVE ITEM → A(TEMI)
DØWN
PUSH NEW ELEMENT DOWN INTO Z WITH A(TEMI) AS ITEM
12
7 → (KSTRT)
(ICI - ICHC) = 0 Z.
NO
yes
15
2 → (KSTRT)
P. 2
B
20
1 → I
P. 2
C
(I-1) → (IC)
CALL HEADC(A, IS, IC, IC
HEADC
SET UP SIX COLS. OF CARD IMAGE → A
A = CARD IMAGE SUBS
IS = STATEMENT NO.
IC = CONTINUATION CA
ICI = CHAR. IN COL.
0 → (IS)
P. 2
A
(KSTRT) → (K6)
25


PG. 1
A
25
IS Z NULL ?
yes
NO
30
POPUP
POPUP 1ST ELEMENT IN Z SAVE ITEM → d(TEMI)
40
CHLNE
INSERT CHAR. IN TEMI → KLTH POSITION OF A
K+1→K
NO
K=66 ?
Yes
50
FOUTPT
WRITE 13 BLD WORDS FROM A ON TO PRINT TAPE, PUNCH TAPE, SAVE TAPE
I+1→I
PG. 1
C
NO
I=10 ?
yes
20
PG. 1
B
60
NO
IS Z NULL ?
Yes
70
RETURN

Program Description

1. Identification

a. Routine Label

DBPCH

b. Name

Output a series of card images from a list of characters in reverse order.

2. Function

A list, containing a string of BCD characters in reverse order, is output onto a punch tape, a print tape, and a save tape as a series of card images, with the statement number in columns 1-5. Before the list is placed into the output buffer, the BCD string is inverted to the correct sequence.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL DBPCH (P, IZ)

b. Entry Conditions

P = A list of BCD characters in reverse order

IZ = Statement number

c. Exit Conditions

The BCD string in P is inverted and output as a series of card images, with the statement number IZ in columns 1-5 and the continuation card number (if any) in column 6. The card text is in columns 7-72.

Output is in a punch tape, a print tape, and a save tape.

d. Error Exits

None.

5. Definition of Identifiers

Z = A temporary head cell for the list of BCD characters output into A.

A = Output buffer

6. Method

The list of BCD characters in list P is output as follows:

a. List P is tested:

(1) If P is null, step a(2) is skipped. Continue at step b.

(2) If P is not null, the elements in P are popped up, one at a time, and then pushed down into list Z, thereby restoring the order of the BCD characters in P.

b. The statement number in IZ is converted to BCD and placed in columns 1-5 of buffer A. If the card image being output is a continuation card, the continuation card number is placed in column 6 of buffer A.

c. The BCD characters in list Z are popped up, one at a time, and placed into columns 7-72 of buffer A.

d. Buffer A is output, as a record of 13 BCD words, onto a print tape, a save tape, and a punch tape.

e. Z is tested:

(1) If Z is null, exit is made from the routine.

(2) If Z is not null, and continuation cards have not exceeded 10, processing continues at step b.

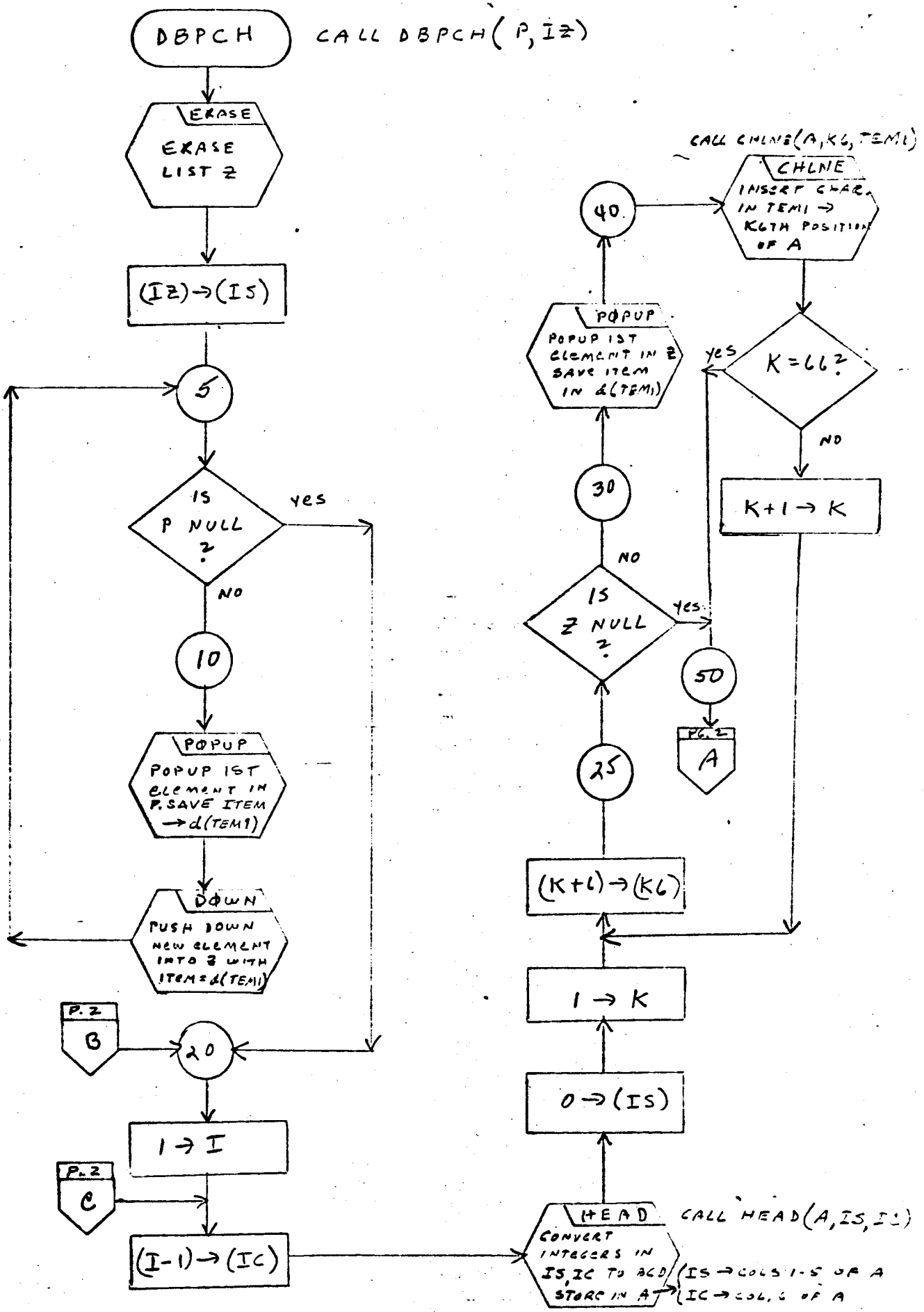
(3) If Z is not null, and 10 continuation cards have been output, the continuation card index is re-initialized and the process starts over at step b.

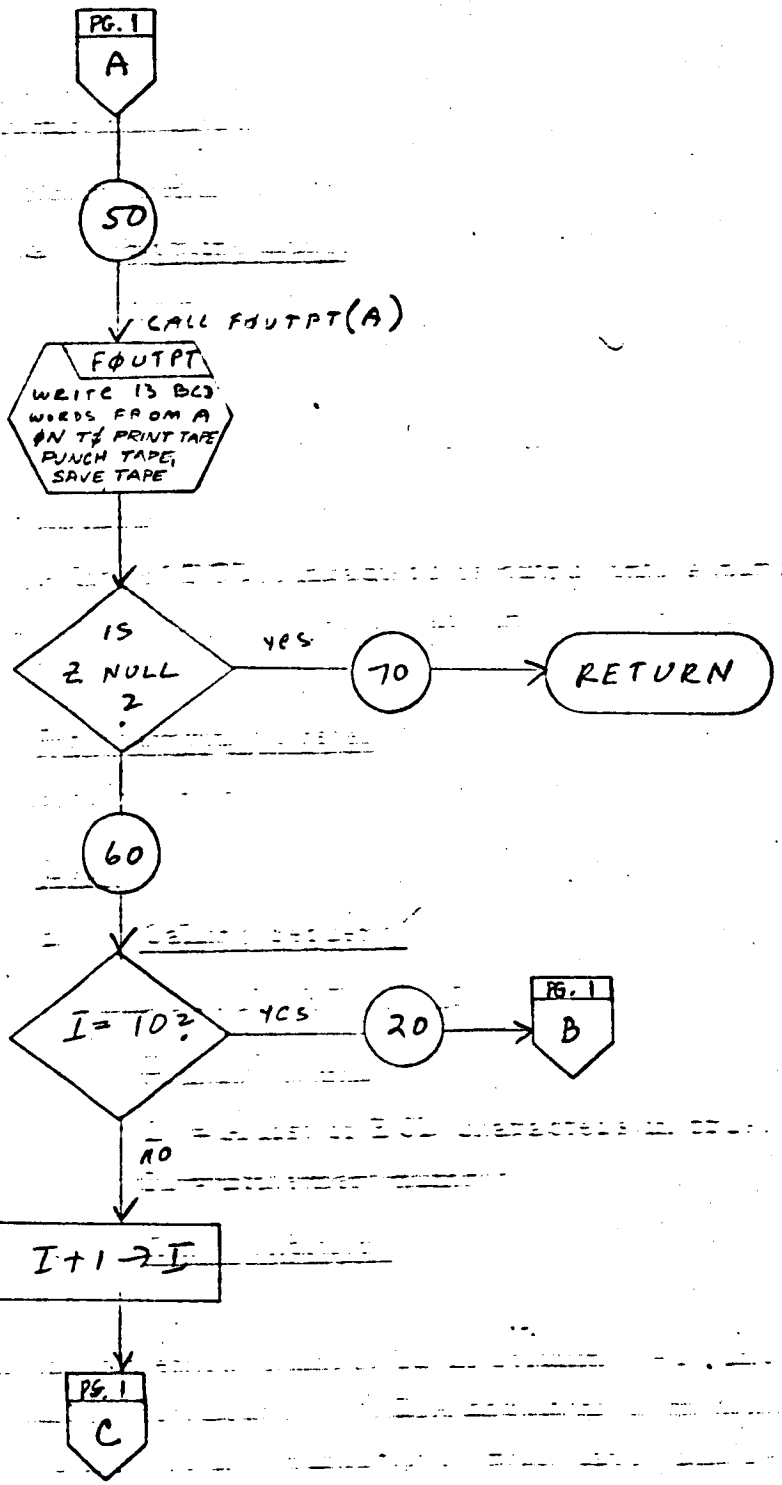
7. Other Subroutines Used

CHLNE, DOWN, ERASE, FOUTPT, HEAD, POPUP.

8. Using Subroutines

DIMEN, DPDST, NLINDM, PARTS, SYMCRD, WRTEQ, ZEROX, Main Program for Pass 2 of TAG Preprocessor.


DBPCH
CALL DBPCH(P, IZ)
ERASE
ERASE LIST Z
(IZ)→(IS)
5
IS P NULL ?
yes
NO
10
POPUP
POPUP 1ST ELEMENT IN P. SAVE ITEM →d(TEM1)
DOWN
PUSH DOWN NEW ELEMENT INTO Z WITH ITEM=d(TEM1)
P. 2
B
20
1→I
P. 2
C
(I-1)→(IC)
CALL CHLNE(A,KL,TEM1)
CHLNE
INSERT CHAR. IN TEM1 → KLTH POSITION OF A
40
POPUP
POPUP 1ST ELEMENT IN Z SAVE ITEM IN d(TEM1)
30
IS Z NULL ?
yes
NO
50
P. 2
A
25
K=LL ?
yes
NO
K+1→K
(K+1)→(KL)
1→K
0→(IS)
HEAD
CONVERT INTEGERS IN IS, IC TO BCD STORE IN A
CALL HEAD(A,IS,IC)
(IS → COLS 1-5 OF A
(IC → COL. 6 OF A


PG. 1
A
50
CALL FOUTPT(A)
FOUTPT
WRITE 13 BCD WORDS FROM A ON T# PRINT TAPE, PUNCH TAPE, SAVE TAPE
IS Z NULL Z
YES
70
RETURN
60
I = 10 ?
YES
20
PG. 1
B
NO
I+1 → I
PG. 1
C

Program Description

1. Identification

   a. Routine Label

      DBPFH

   b. Name

      Output a series of card images.

2. Function

A list of BCD characters is output onto a punch tape, print tape, and save tape as a series of card images, with the statement number in columns 1-5.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL DBPFH (Z, IZ)

   b. Entry Conditions

      Z = A list of BCD characters in order
      IZ = Statement number

   c. Exit Conditions

The BCD string in list Z is output as a series of card images, with the statement number IZ in columns 1-5, and continuation card number, if any, in column 6. The card text is in columns 7-72.

Output is in a punch tape, print tape, and save tape.

   d. Error Exits

      None.

5. Definition of Identifiers

   A = Output buffer

6. Method

The list of BCD characters in list Z is output as follows:

a. The statement number in IZ is converted to BCD and placed in columns 1-5 of buffer A. If the card image being output is a continuation card, the continuation card number is placed in column 6 of buffer A.

b. The BCD characters in Z are popped up, one at a time, and placed into columns 7-72 of buffer A.

d. Buffer A is output, as a record of 13 BCD words, onto a print tape, punch tape, and a save tape.

d. Z is tested:

(1) If Z is null, exit is made from the routine

(2) If Z is not null, and the number of cards has not exceeded 10, processing continues at step a.

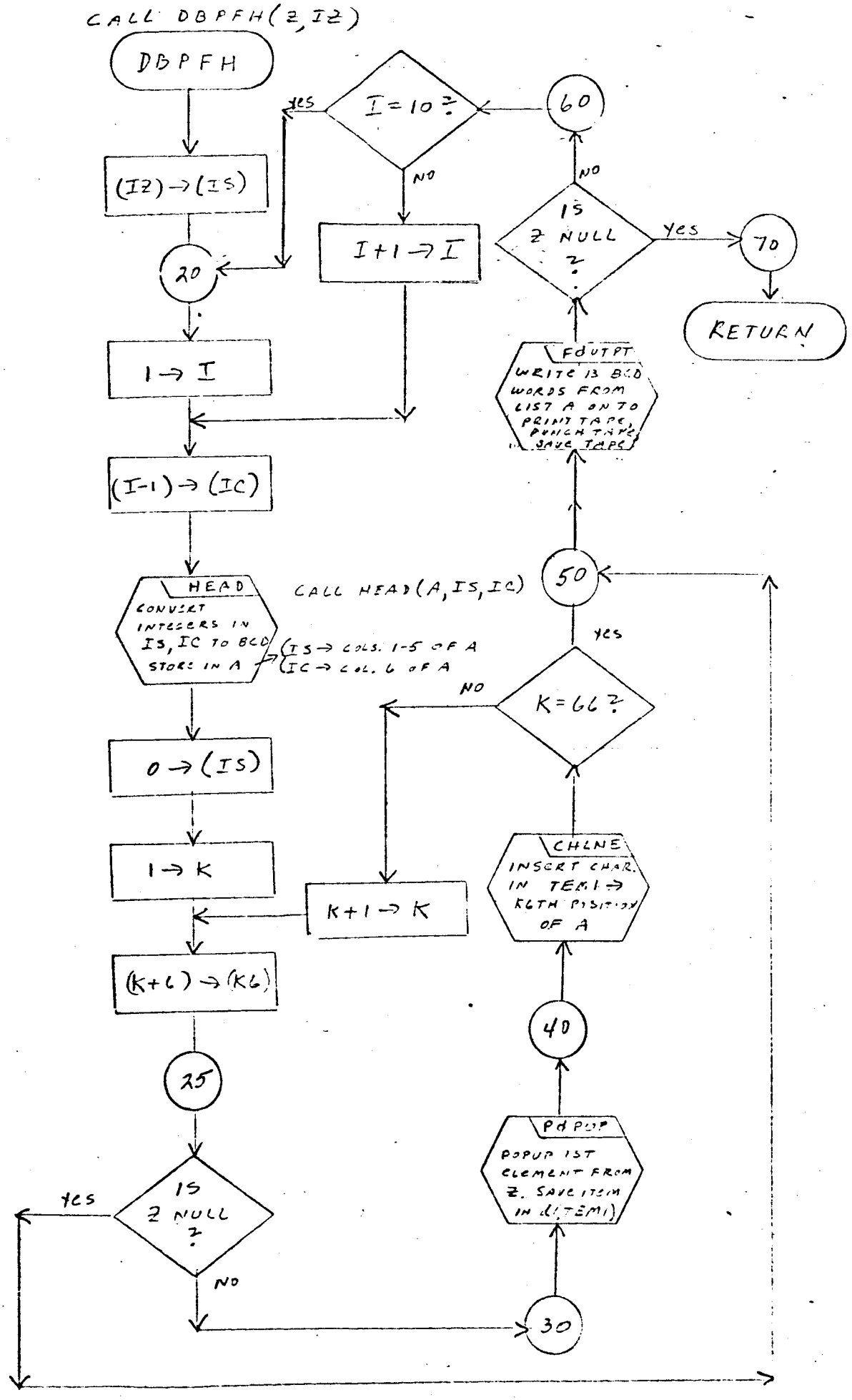
(3) If Z is not null and 10 cards have been output, the continuation card index is re-initialized and processing starts over at step a.

7. Other Subroutines Used

CHLNE, FOUTPT, HEAD, POPUP.

8. Using Subroutines

STAT.


CALL DBPFH(Z,I2)
DBPFH
(I2)→(IS)
20
1→I
(I-1)→(IC)
HEAD
CONVERT INTEGERS IN IS, IC TO BCD STORE IN A
CALL HEAD(A,IS,IC)
IS→ COLS. 1-5 OF A
IC→ COL. 6 OF A
0→(IS)
1→K
(K+6)→(K6)
25
IS Z NULL ?
yes
NO
30
PAPOP
POPUP 1ST ELEMENT FROM Z. SAVE ITEM IN (ITEM1)
40
CHLNE
INSERT CHAR. IN ITEM1 → KTH POSITION OF A
K=66 ?
NO
yes
K+1→K
50
FOUTPT
WRITE 13 BCD WORDS FROM LIST A ONTO PRINT TAPE, PUNCH TAPE, SAVE TAPE
IS Z NULL ?
yes
NO
70
RETURN
I=10 ?
yes
NO
I+1→I
60

Program Description

1. Identification

   Routine Label

   DIFA

2. Function

   This subroutine computes the absolute difference between the characteristics of two floating-point numbers.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL DIFA (A, B, C)

   b. Entry Conditions

      A and B are floating-point numbers.

   c. Exit Conditions

      C = absolute difference between the characteristics of A and B

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

   B is subtracted from A with UFS (un-normalized floating-point subtraction and the result stored in the accumulator. Then,

   a. (AC) ∩ 000777777777 ∪ 177000000000 → (AC)

   b. (AC) + 177000000000 → (AC), where + is a floating-point addition, gives the absolute difference between the characteristics of A and B, which is stored in C.

7. Other Subroutines Used

None.

8. Using Subroutines

BAKELM, COMBN, MULTS, STRIK.

Program Description

1. Identification

   a. Routine Label

      DIMEN

2. Function

   Writes out the "DIMENSION" statements.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL DIMEN (NPT, KIND)

   b. Entry Conditions

      NPT = The NPT array of Main 2
      KIND = The KIND indicator of Main 2

   c. Exit Conditions

      "DIMENSION" and "COMMON" statements written out.

   d. Error Exits

      None.

5. Definition of Identifiers

   IDFSW = An indicator
   DEF = A list of type A holding "DIMENSION"
   DIM = A list of type A holding "DIMENSION FHB(6), LNH(6), FMIS("
   COM = A type A list holding "COMMON FMIS"
   IBLK = A BCD blank
   ICOMA = A BCD comma

IBRKR = A BCD right parenthesis

IBRKL = A BCD left parenthesis

PLC = A list used to hold the output card image

ITEMX = Temporary storage

KYN, N1, N2 — Used to hold output information from EXCPT

MIS, MIS1 — Used to calculate the dimension of FMIS

MIS2 — Holds the dimension of FCMDMY

MAXTM — Holds the dimension of FTEM1, FTEM2

SDEF — A list of the following symbols: FS41, FL44, FI41, FL41, FS11, FL42, FS21, FL43, FS31, FV31, FG33, FI31, FG31, FV11, FG32, FV21, FL31, FL32, FL33, FL34, FC11, FI11, FC12, FG11, FG12, FG13, FL11, FL12, FL13, FL14, FVD21, FC22, FI21, FG21, FG22, FG23, FL21, FL22, FL23, FL24, FSD11, FSD21, FSD31, FC21, FVD11, FVO11, FCI22, FGI33, FLI44, FV41, FSO41

SCOM — A list of the following symbols: FVD21, FSD11, FSD21, FSD31, FV21, FS11, FS21, FS31, FT2, FT1, LNH, FHB

6. Method

a. The symbols of SDEF are sent, one at a time, to EXCPT for testing and transformation.
If KYN = 1, and N1 is non-zero and N2 equals 2 or greater, or if KYN = 1 and N1 is = 2 or greater and N2 = 1, then the symbol is placed in the DIMENSION statement in the form symbol (N1, N2).
The DIMENSION statement is written out.

b. Set MIS = NPT(2). If KIND is greater than zero, then set MIS = MIS + NPT(1) + NPT(2) + NPT(3). If MIS > 0, set MIS1 = 26*MIS and output these statements:

DIMENSION FHB(6), LNH(6), FMIS(mis1)

COMMON FMIS, S1, S2, · · ·

where S1, S2 are symbols from SCOM with this characteristic: each symbol from SCOM is sent to EXCPT for testing. If KYN = 1 and N1 ≠ 0 and N2 ≠ 0, then the symbol is placed in the "COMMON" statement.

c. Set MIS2 = 3000-(MIS*28+14). If MIS2 > 0, output these statements: DIMENSION FCMDMY(mis2) COMMON FCMDMY

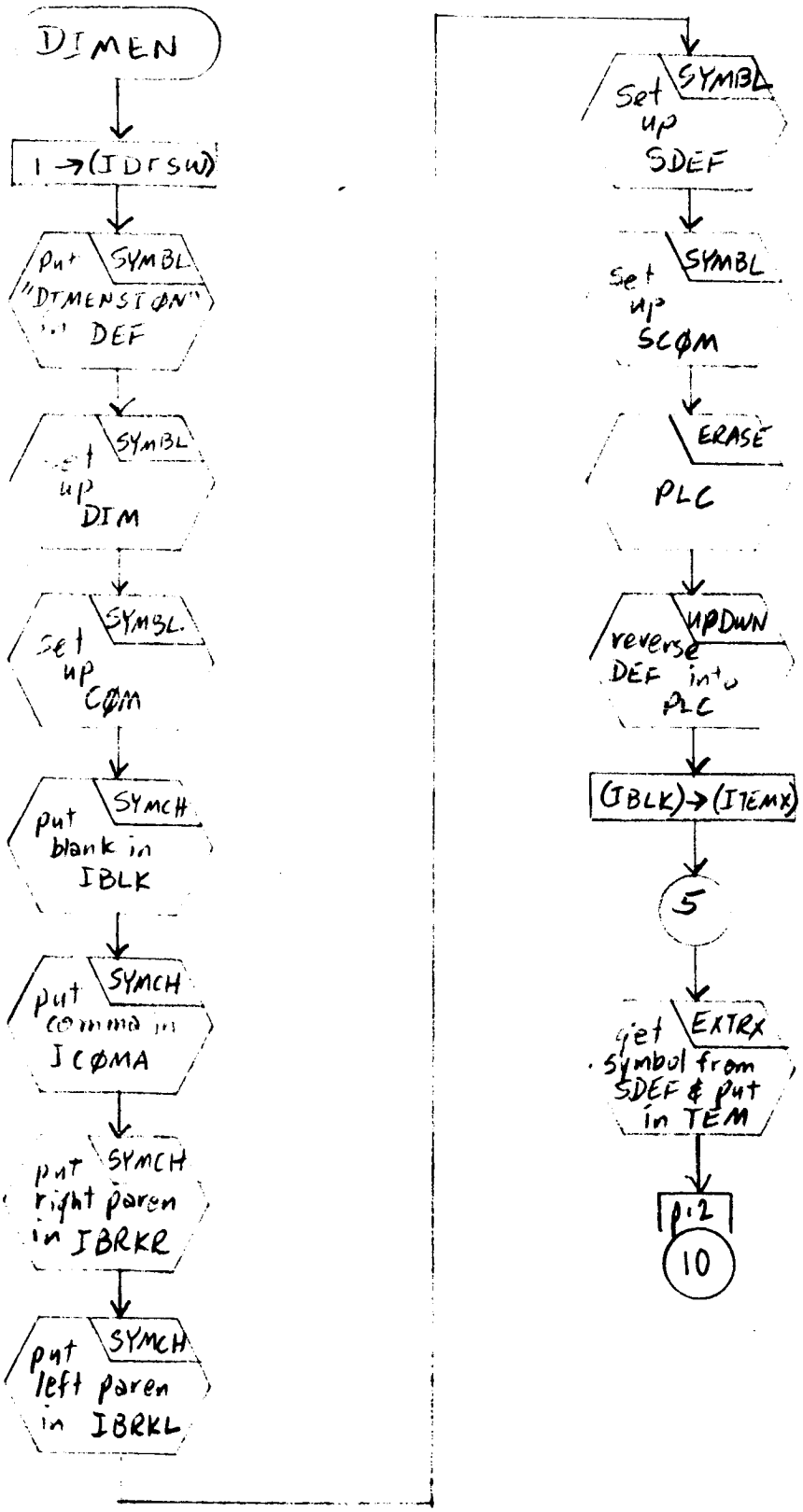
d. Set MAXTM = max(NPT(2), NPT(3), NPT(4)). If MAXTM > 1, output this statement: DIMENSION FTEM1(maxtm), FTEM2 (maxtm)

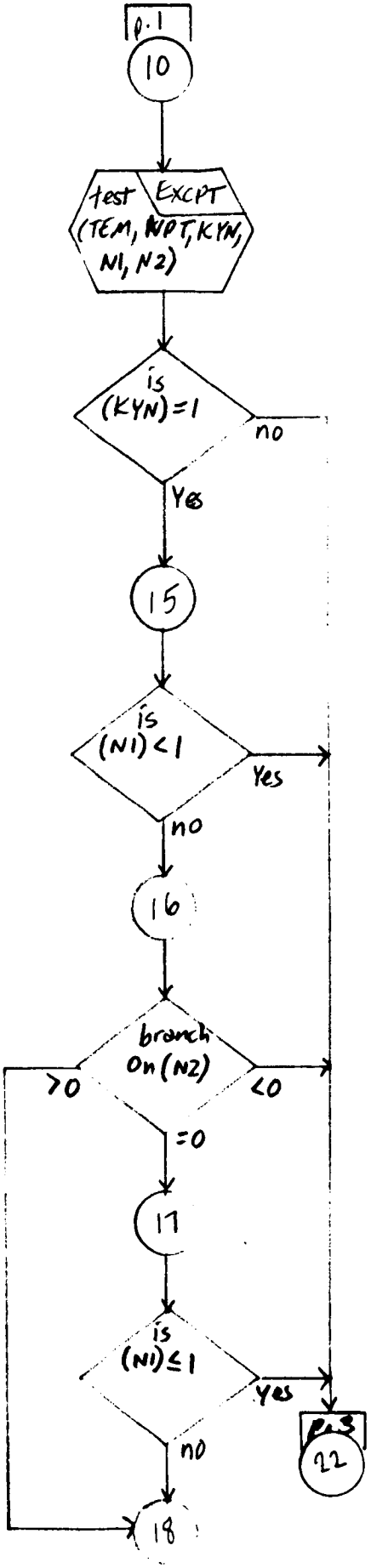
7. Other Subroutines Used

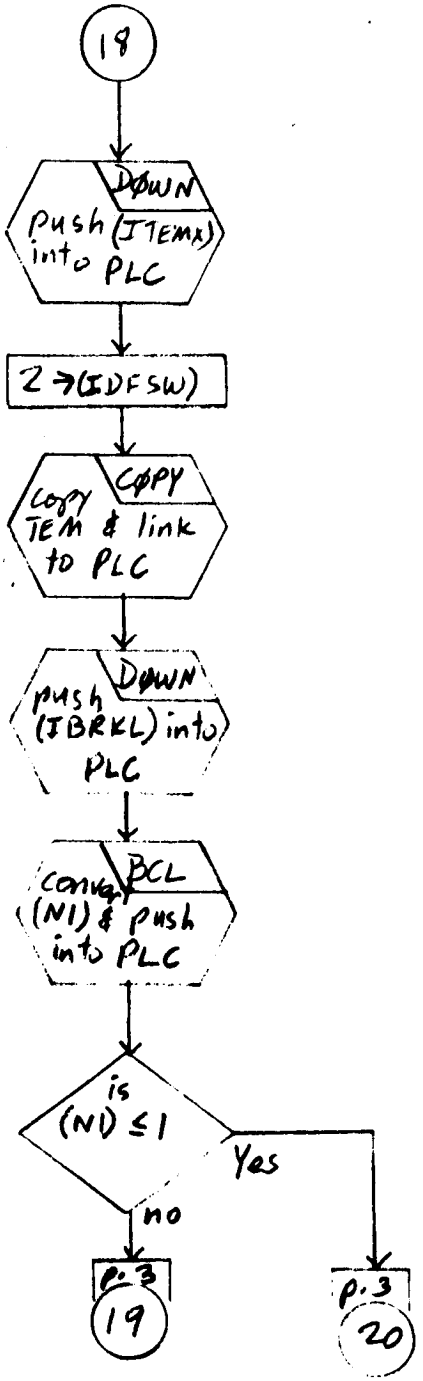
BCL, COPY, DBPCH, DOWN, DUMP, ERASE, EXCPT, EXTRX, STAT, SYMBL, SYMCH, UPDWN.

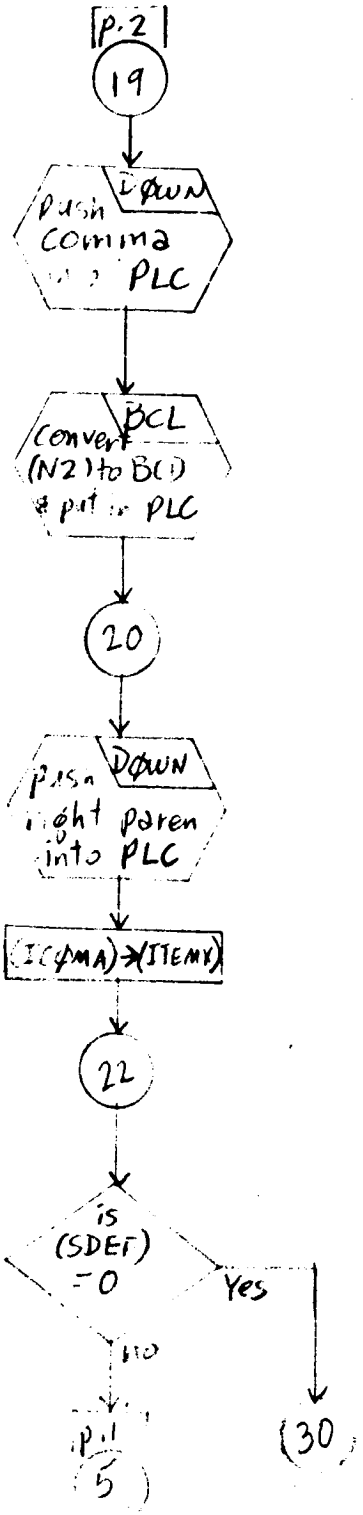
8. Using Subroutines

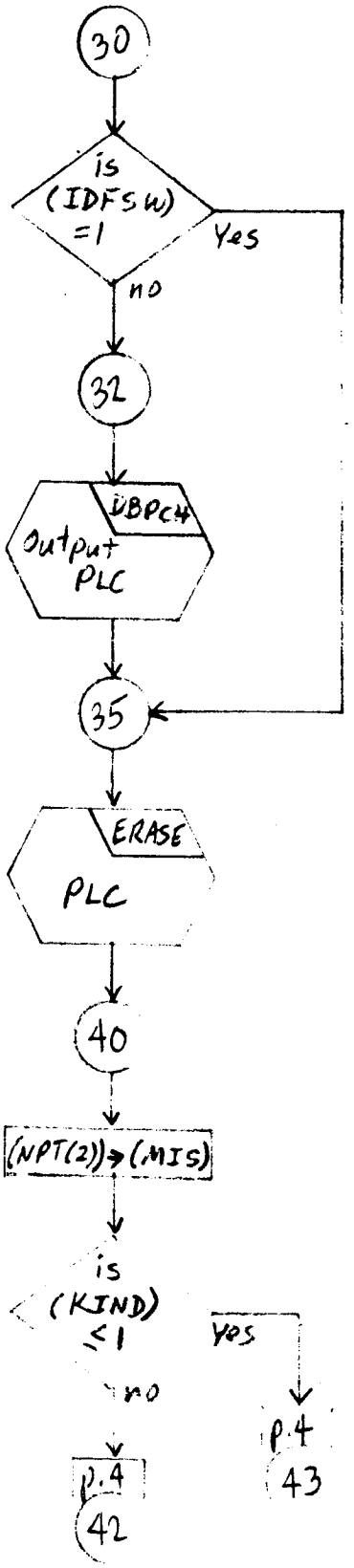
Main Program, Pass 2, TAG Preprocessor.

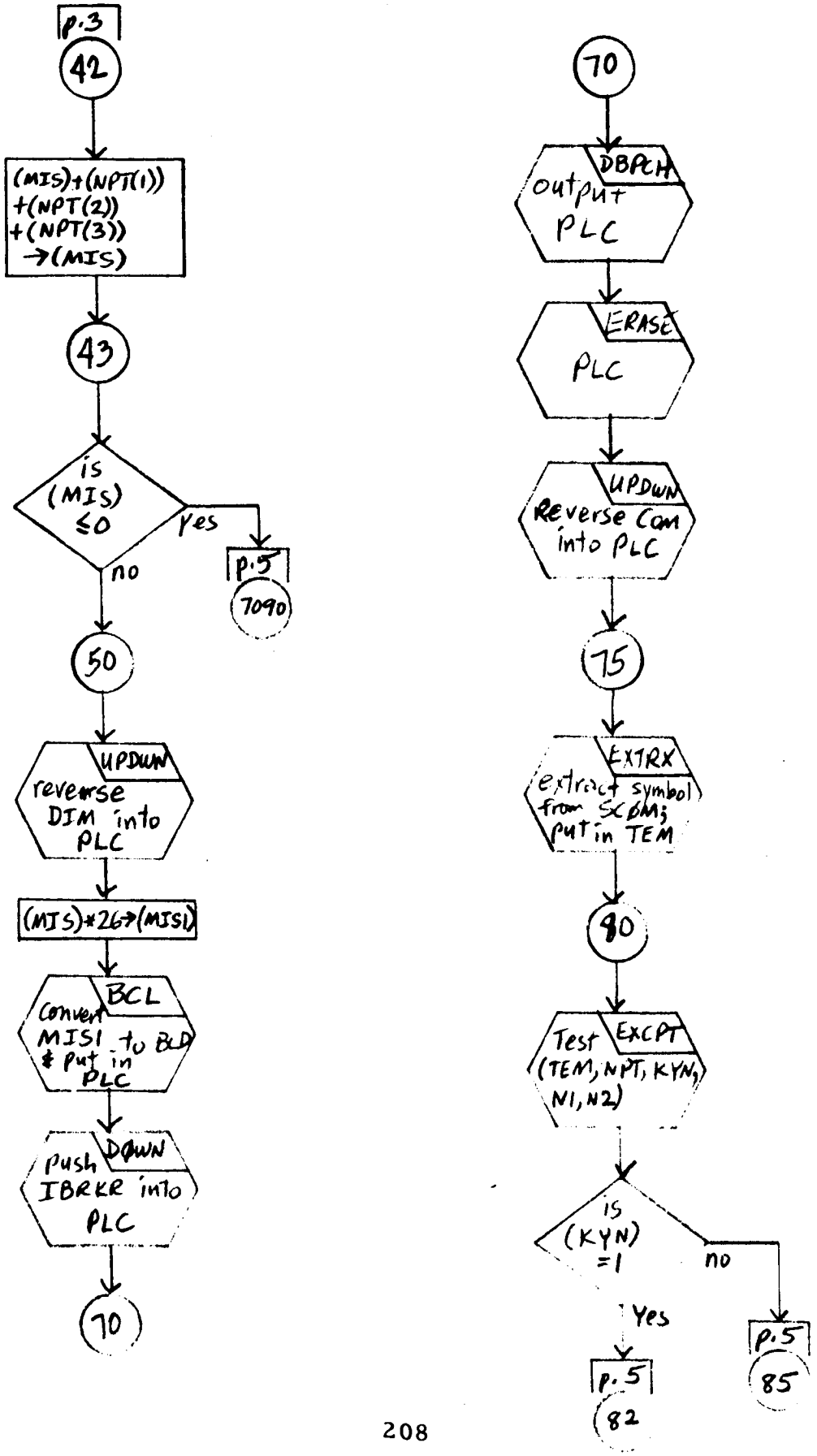

DIMEN
1 → (IDFSW)
Put SYMBL "DIMENSION" DEF
Set up SYMBL DIM
Set up SYMBL COM
put SYMCH blank in IBLK
put SYMCH comma in ICOMA
put SYMCH right paren in IBRKR
put SYMCH left paren in IBRKL
Set up SYMBL SDEF
Set up SYMBL SCOM
ERASE PLC
reverse UPDWN DEF into PLC
(IBLK) → (ITEMX)
5
get EXTRX symbol from SDEF & put in TEM
P.2
10

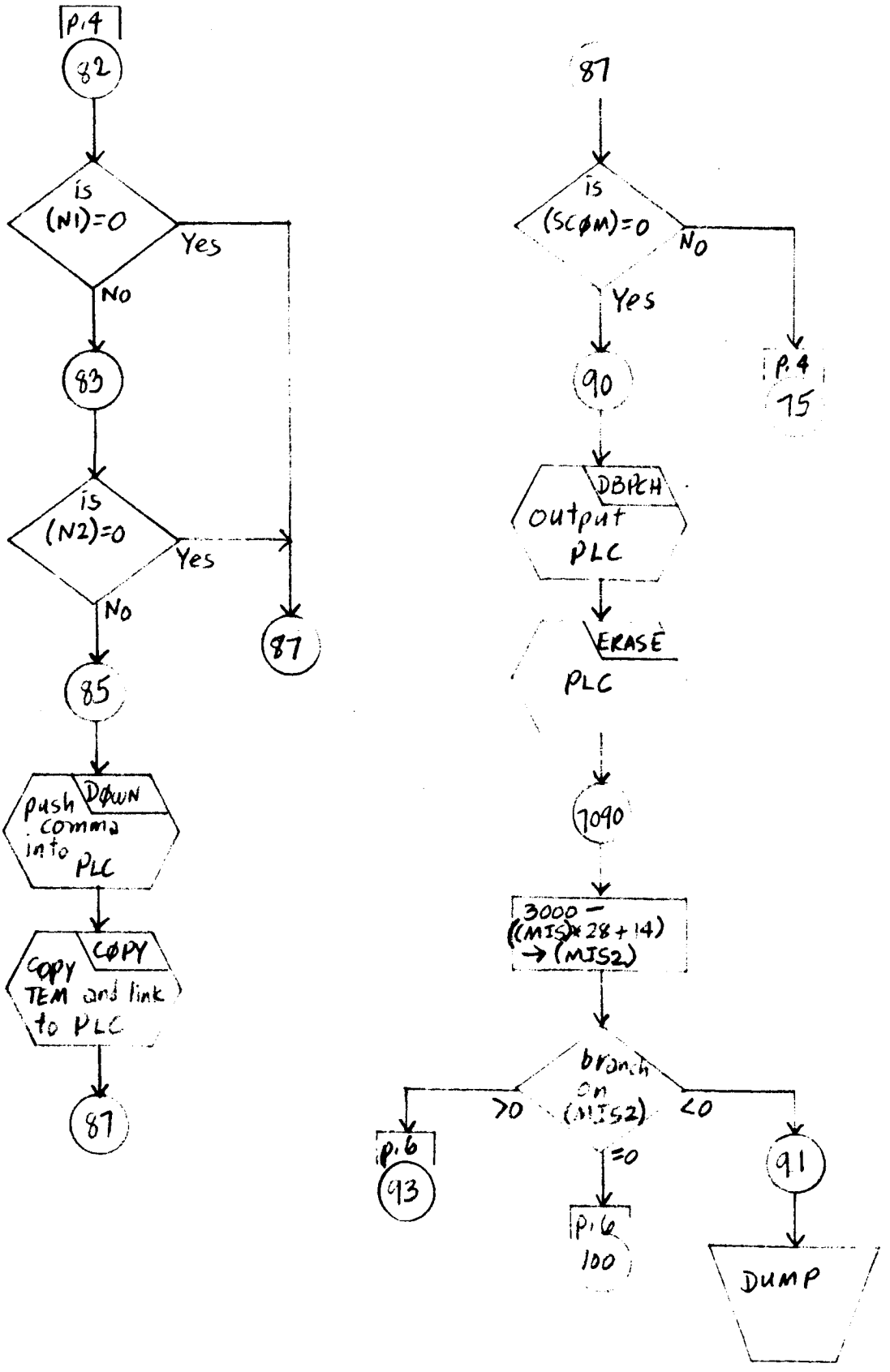

p.1
10
test EXCPT (TEM, NPT, KYN, N1, N2)
is (KYN) = 1
no
Yes
15
is (N1) < 1
Yes
no
16
branch on (N2)
>0
<0
=0
17
is (N1) ≤ 1
yes
no
p.3
22
18

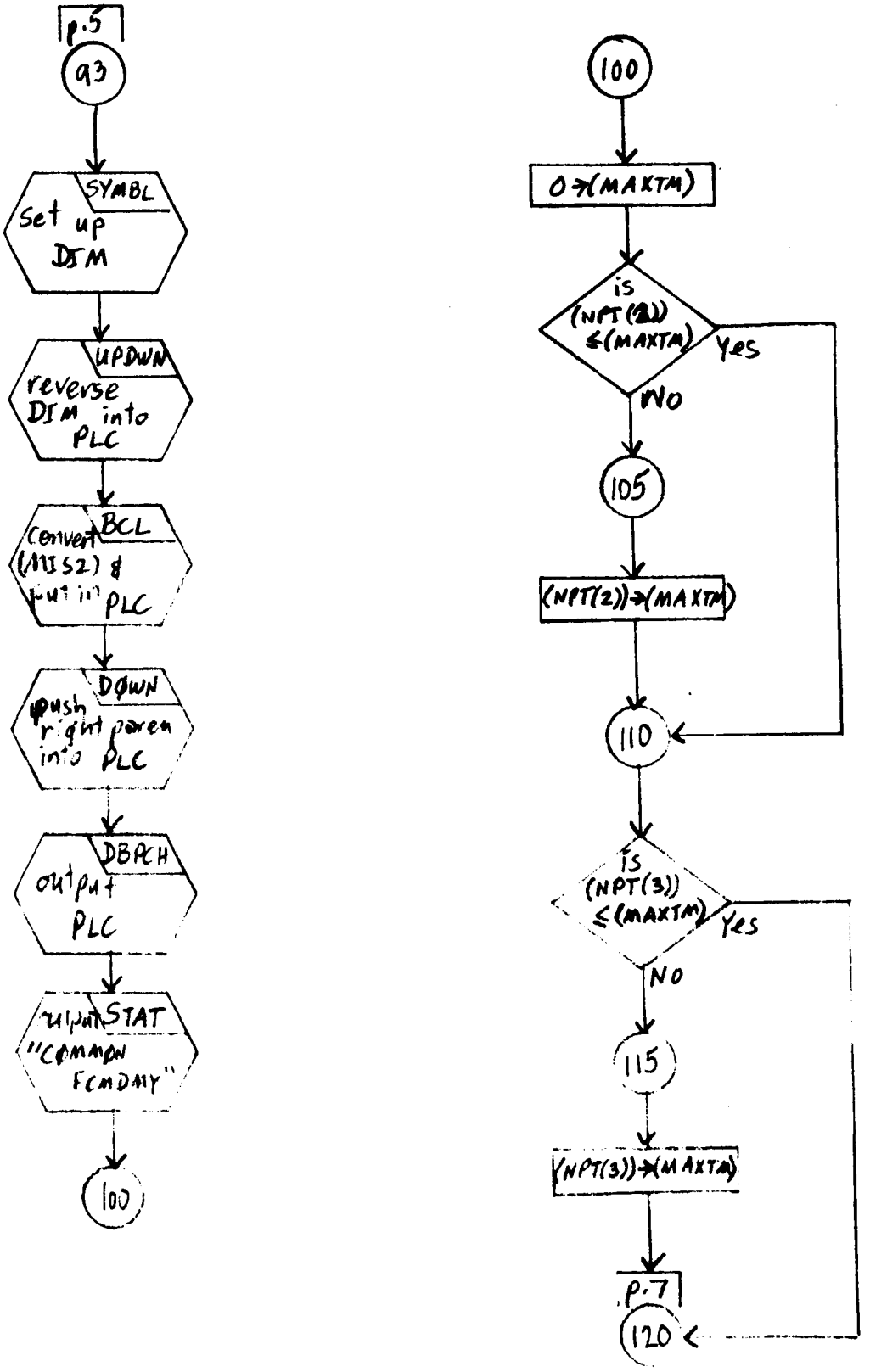

19
DOWN
push (JTEMX) into PLC
2 → (IDFSW)
COPY
copy TEM & link to PLC
DOWN
push (IBRKL) into PLC
BCL
conv (N1) & push into PLC
is (N1) ≤ 1
Yes
no
p.3
19
p.3
20

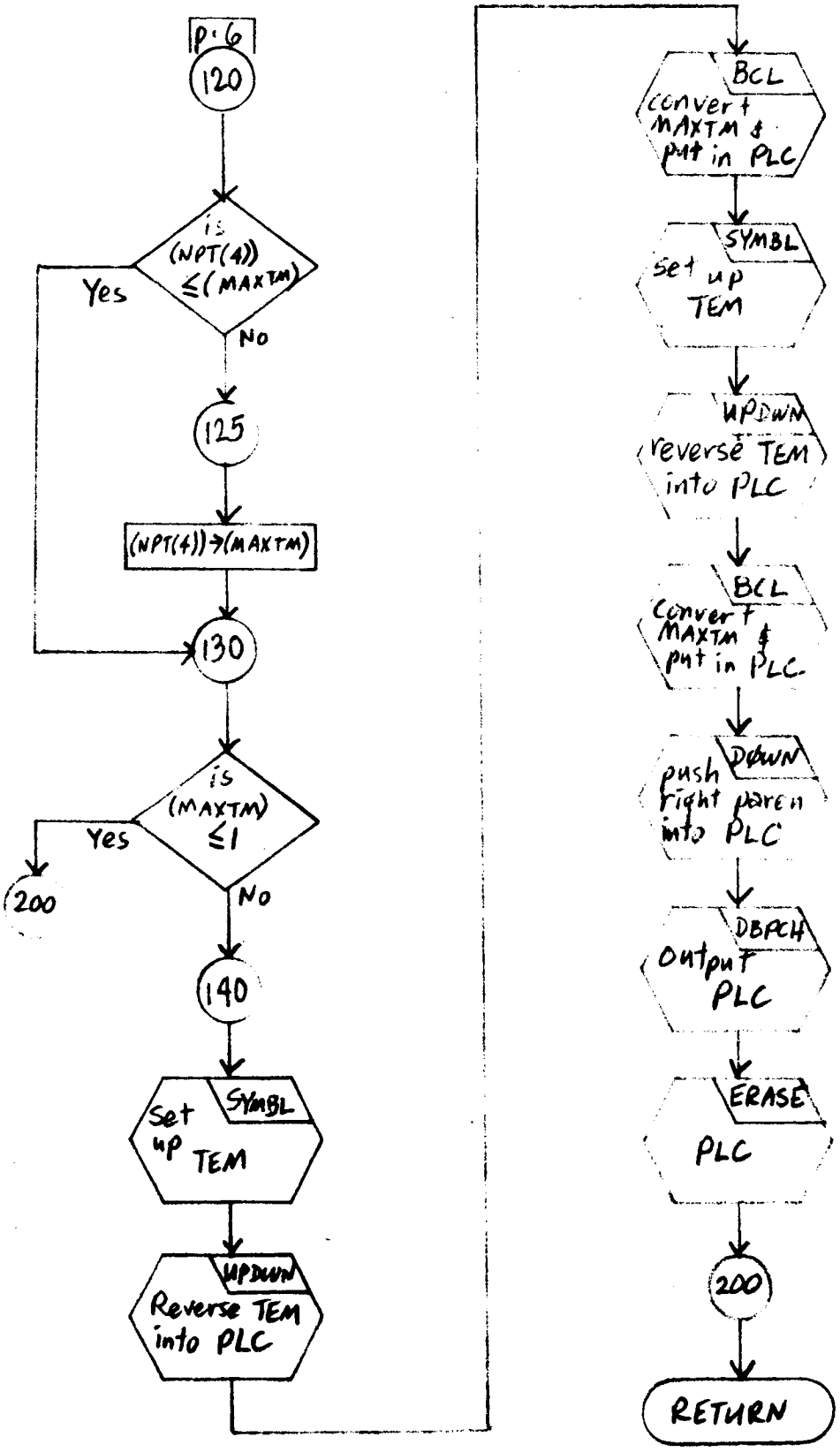

P.2
19
DOWN
Push comma into PLC
DCL
Convert (N2) to BCD & put in PLC
20
DOWN
Push right paren into PLC
(COMA)→(ITEMX)
22
is (SDEF) = 0
Yes
no
P.1
5
30


30
is (IDFSW) = 1
Yes
no
32
DBPCH
Output PLC
35
ERASE
PLC
40
(NPT(2))→(MIS)
is (KIND) ≤ 1
Yes
no
P.4
43
P.4
42


p.3
42
(MIS)+(NPT(1))
+(NPT(2))
+(NPT(3))
→(MIS)
43
is
(MIS)
≤0
Yes
no
p.5
7090
50
UPDWN
reverse
DIM into
PLC
(MIS)*26→(MISI)
BCL
Convert
MISI to BCD
& put in
PLC
DOWN
Push
IBRKR into
PLC
70
70
DBPCH
output
PLC
ERASE
PLC
UPDWN
Reverse Com
into PLC
75
EXTRX
extract symbol
from SCRN;
put in TEM
80
Test
EXCPT
(TEM, NPT, KYN,
N1, N2)
is
(KYN)
=1
no
Yes
p.5
85
p.5
82


P.4
82
is (N1)=0
Yes
No
83
is (N2)=0
Yes
No
87
85
push comma into PLC
DOWN
COPY
COPY TEM and link to PLC
87
87
is (SCOM)=0
No
Yes
90
P.4
75
DBTCH
output PLC
ERASE
PLC
7090
3000 - ((MIS)*28+14) → (MIS2)
branch on (MIS2)
>0
<0
=0
P.6
93
P.6
100
91
DUMP


p.5
93
SYMBL
Set up DIM
UPDWN
reverse DIM into PLC
BCL
convert (MISS2) & put in PLC
DOWN
push right paren into PLC
DBACH
output PLC
STAT
"COMMON FCNDMY"
100
100
0→(MAXTM)
is (NPT(2)) ≤ (MAXTM)
Yes
NO
105
(NPT(2))→(MAXTM)
110
is (NPT(3)) ≤ (MAXTM)
Yes
NO
115
(NPT(3))→(MAXTM)
P.7
120


P·6
120
is (NPT(4)) ≤ (MAXTM)
Yes
No
125
(NPT(4)) → (MAXTM)
130
is (MAXTM) ≤ 1
Yes
No
200
140
SYMBL
Set up TEM
UPDWN
Reverse TEM into PLC
BCL
convert MAXTM & put in PLC
SYMBL
set up TEM
UPDWN
reverse TEM into PLC
BCL
convert MAXTM & put in PLC
DOWN
push right paren into PLC
DBPCH
output PLC
ERASE
PLC
200
RETURN

Program Description

1. Identification

   a. Routine Label

      DONBD

2. Function

The tens and ones digits of integer I are split out and stored as individual BCD characters. They are pushed down into list P such that the ones digit is the first element of P.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL DONBD (P, I)

   b. Entry Conditions

      I = A binary integer
      P = Head of a list

   c. Exit Conditions

List P contains the ones digit of integer I in its first element and the tens digit of integer I in its second element.

   d. Error Exits

      None.

5. Definition of Identifiers

   IX(1) = Tens digit of integer I
   IX(2) = Units digit of integer I

6. Method

The tens and units digits of integer I are split out and pushed down into list P as follows:

a. J is cleared to zero. List L is erased.

b. The units digit is split out from I and placed in IX(2). The tens digit is placed in IX(1).

c. Two elements containing IX(2), IX(1) are inserted into list J, with IX(1) at the top of the list. L is set to point to the first element in J.

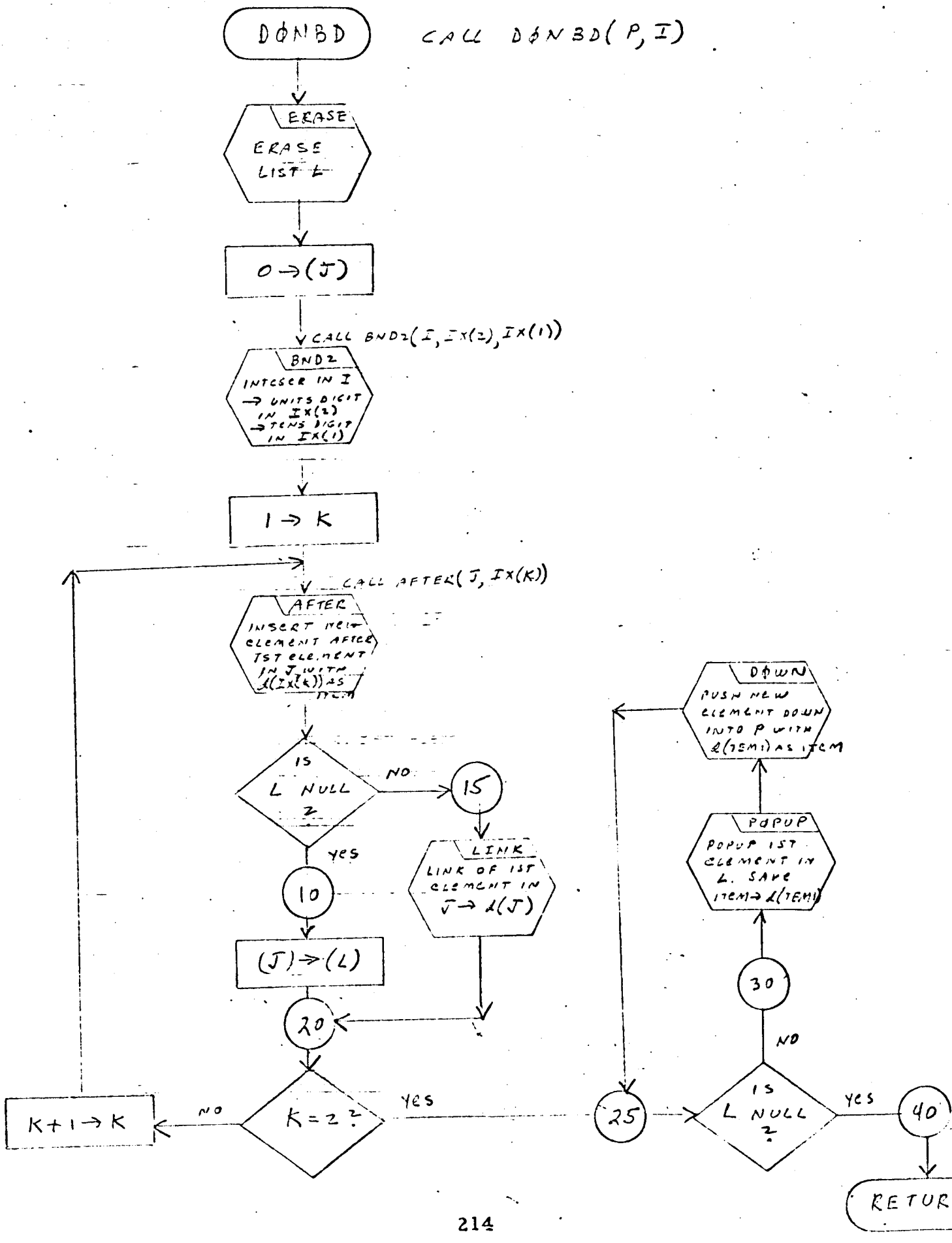
d. The new elements are popped up from L and pushed down into list P, such that the first element contains the units digit in IX(2).

7. Other Subroutines Used

AFTER, BND2, DOWN, ERASE, LINK, POPUP.

8. Using Subroutines

FLTCON, INPUTX, SYMCRD.


DØNBD
CALL DØNBD(P, I)
ERASE
ERASE LIST L
0 → (J)
CALL BND2(I, IX(2), IX(1))
BND2
INTEGER IN I
→ UNITS DIGIT IN IX(2)
→ TENS DIGIT IN IX(1)
1 → K
CALL AFTER(J, IX(K))
AFTER
INSERT NEW ELEMENT AFTER 1ST ELEMENT IN J WITH L((IX(K))) AS ITEM
IS L NULL ?
NO
15
LINK
LINK OF 1ST ELEMENT IN V → L(J)
yes
10
(J) → (L)
20
K = 2 ?
yes
NO
K+1 → K
25
IS L NULL ?
yes
40
RETUR
NO
30
POPUP
POPUP 1ST ELEMENT IN L. SAVE ITEM → L(ITEM)
DØWN
PUSH NEW ELEMENT DOWN INTO P WITH L(ITEM) AS ITEM

Program Description

1. Identification

   a. Routine Label

      DOWN

   b. Name

      Push an element DOWN into a list.

2. Function

This subroutine pushes a new element down into a list. If the list is null, a new list is started which points to this new element.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL DOWN (Z, C, IF)

   b. Entry Conditions

      Z = Head of a list
      d(C) = Item of new element
      d(IF) = Flag of new element

   c. Exit Conditions

New element with d(C) and d(IF), if any, is first element pointed to by Z. The pointer formerly in Z becomes the link portion of the new element. If list Z was null, the new element will contain a link of zero.

   d. Error Exits

      None.

5. Definition of Identifiers

d(DW90) = Pointer to new element to be pushed down into list Z

6. Method

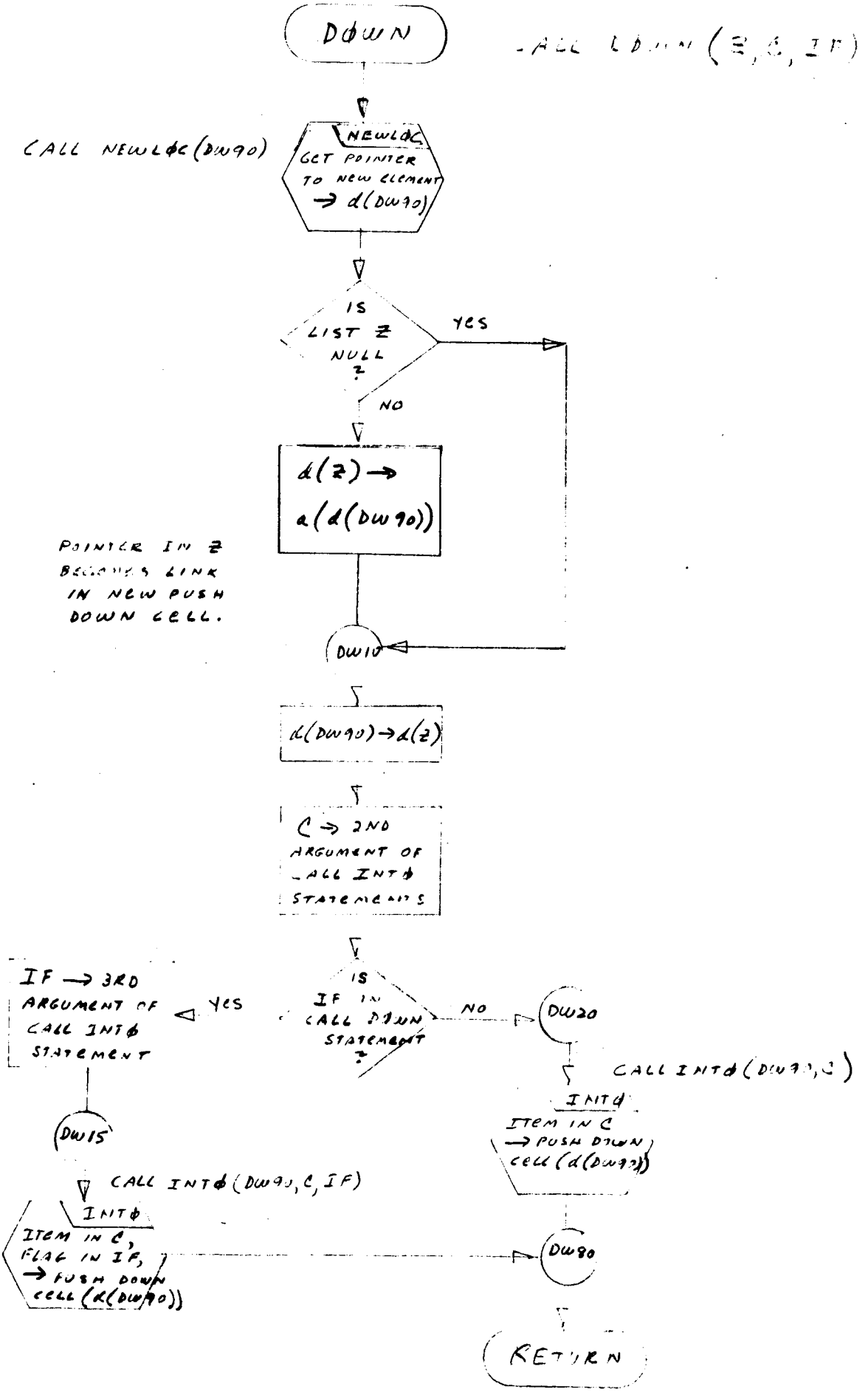
A new element is obtained from AVS and pushed down into Z with the item portion and flag portion of the element as specified by d(C) and d(IF).

7. Other Subroutines Used

INTO, NEWLOC.

8. Using Subroutines

BCL, BLNOUT, COMBN, DBPCHC, DBPCH, DIMEN, DONBD, DOWNS, DPDST, ERASE, EXTRX, FISH, FLTCON, FRACT, GOBLE, HOLBK, INPUTX, LTRACE, MRKLST, NUMB, PARTS, PUSPCH, READCH, RECOVR, STRIK, SUBST, SYMCRD, SYMTP, UPDWN, WRTEQ, ZEROX, Main Program for Pass 2 of Tag Preprocessor.


DOWN
CALL DOWN (Z, C, IF)
CALL NEWLOC(DW90)
NEWLOC
GET POINTER TO NEW ELEMENT → d(DW90)
IS LIST Z NULL ?
YES
NO
d(Z) → a(d(DW90))
POINTER IN Z BECOMES LINK IN NEW PUSH DOWN CELL.
DW10
d(DW90) → d(Z)
C → 2ND ARGUMENT OF CALL INIT∅ STATEMENTS
IS IF IN CALL DOWN STATEMENT ?
NO
YES
IF → 3RD ARGUMENT OF CALL INIT∅ STATEMENT
DW20
CALL INIT∅(DW90, C)
INIT∅
ITEM IN C → PUSH DOWN CELL (d(DW90))
DW15
CALL INIT∅(DW90, C, IF)
INIT∅
ITEM IN C, FLAG IN IF, → PUSH DOWN CELL (d(DW90))
DW80
RETURN

Program Description

1. Identification

   a. Routine Label

      DOWNS

2. Function

The elements of a list are popped up, one at a time, and pushed down into another list, in reverse order.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL DOWNS (P, I)

   b. Entry Conditions

      P = Head of the push down list

      I = Head of the pop-up list

   c. Exit Conditions

List P will contain all elements that were in list I, except that their order will be reversed. List I is not destroyed.

   d. Error Exits

      None.

5. Definition of Identifiers

TEM1 = A temporary cell whose decrement holds the item of the element popped up from list I.

TEM2 = A temporary cell whose decrement holds the flag, if any, of the element popped up from list I.

J = A temporary head cell whose decrement points to the next element to be popped up from list I.

6. Method

a. J is initialized with the contents of the head cell I, thereby pointing to the first element of list I.

b. J is tested:

(1) If J is null, there are no more elements in the list, and exit is made from the routine.

(2) If J is not null, the first (or next) element in the list is popped up, and its item and flag are saved in TEM1 and TEM2.

(3) A new element is pushed down into list P, whose item is TEM1 and whose flag is TEM2.

(4) The link of the element popped up from J is moved to the decrement of J, thus becoming the pointer to the next element to be popped up.

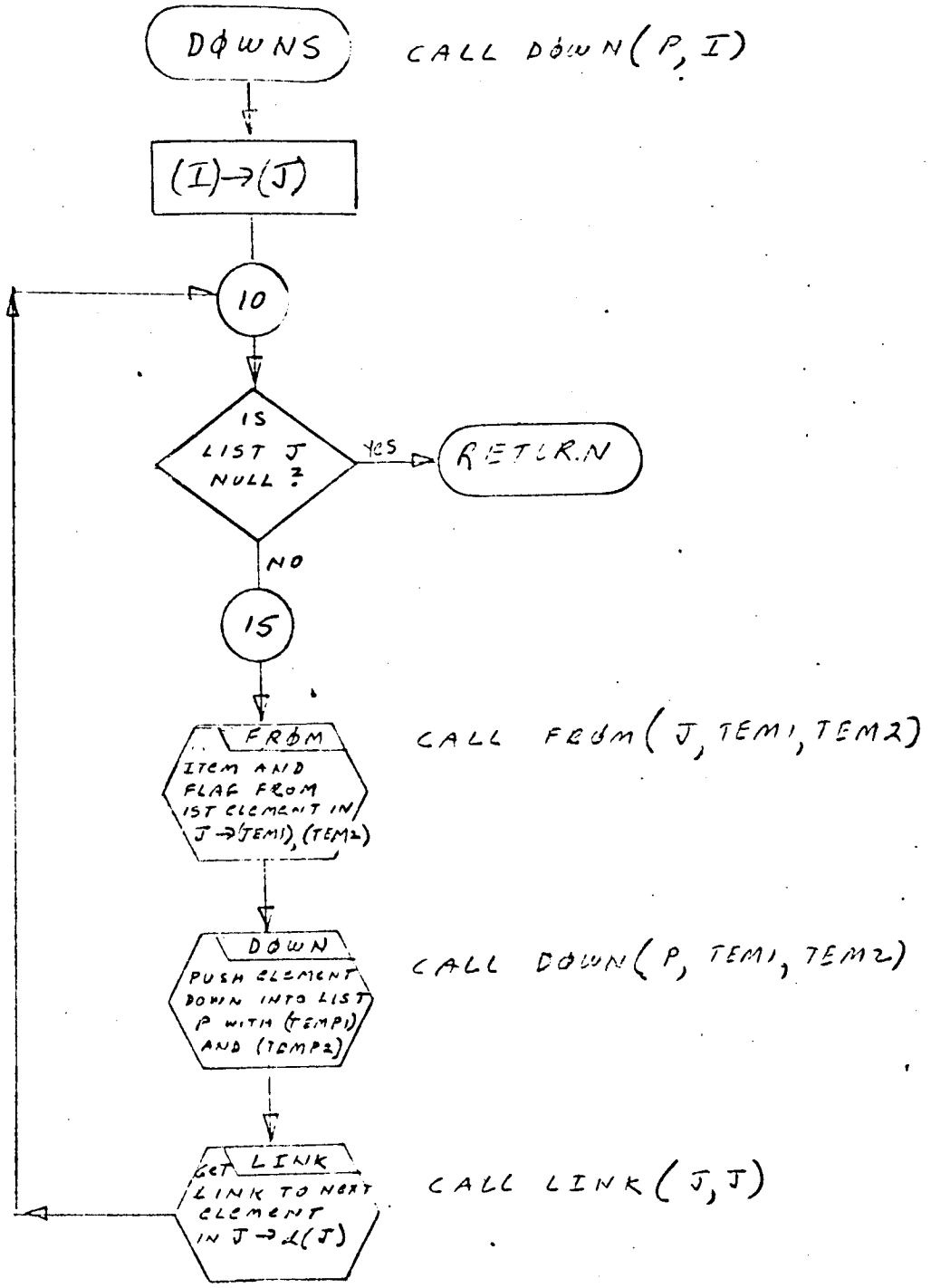
(5) Steps b(1), b(2), b(3), and b(4) are repeated until list I (pointed to by J) is terminated.

7. Other Subroutines Used

DOWN, FROM, LINK.

8. Using Subroutines

COMBN, GOBLE, INZERO, SUBST, SYMCRD, SYMTP, ZEROX, Main Program for Pass 2 of TAG Preprocessor.


DOWNS
CALL DOWN(P, I)
(I)→(J)
10
IS LIST J NULL ?
YES
RETURN
NO
15
FROM
ITEM AND FLAG FROM 1ST ELEMENT IN J→(TEMI), (TEM2)
CALL FROM(J, TEMI, TEM2)
DOWN
PUSH ELEMENT DOWN INTO LIST P WITH (TEMP1) AND (TEMP2)
CALL DOWN(P, TEMI, TEM2)
LINK
GET LINK TO NEXT ELEMENT IN J→L(J)
CALL LINK(J, J)

Program Description

1. Identification

   a. Routine Label

      DPDST

2. Function

   Writes out the CALL FMARK statement using dependent stop variable names.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL DPDST (LSTMK)

   b. Entry Conditions

      LSTMK is a list in list format C which contains the dependent stop function variable names.

   c. Exit Conditions

      The statement CALL FMARK (.... is output.
      LSTMK is unchanged.

   d. Error Exits

      None.

5. Definition of Identifiers

P1	List used to hold LSTMK symbols. P1 is in list format B.
P	Holds P1 in reverse order
ICM	BCD comma
IBLK	BCD blank
NUM	List (format A) used to hold the number portion of the dependent stop variable

ITEM	Holds characters as they are popped out of P
LCNT ITEM ITEM1	Used to convert the BCD number to binary
REST	List (format A) used to hold statement to be output
REST1	List (format A) used to temporarily hold character strings to be put in REST

6. Method

RECOVR is used to transform LSTMK, in format C, to P1, in format B.

P1 is then reversed into P.

P now holds symbols of the following form: na , na , --- where n is an integer and a is a string of alphanumerics. These are the dependent stop function names in the form used in input; e.g.,

$1FLXST1
$2FLXST2

but with the $ stripped off.

REST (in format A) is initialized to hold the first part of the statement to be output:

CALL FMARK(LCNT, FHB, 0, LTYPE, 1, 1, 2, FHC, 3, FT, FSTOP,

The variable names in P are now processed, one at a time. The integer portion of the variable is incremented by 3 and pushed into REST, followed by a comma, followed by the alphanumeric portion, followed by a comma, a zero, and a comma. At termination (when P is null), a final zero and right parenthesis are pushed into REST, and the statement is output. For the two example symbols given above, the output statement would look like this:

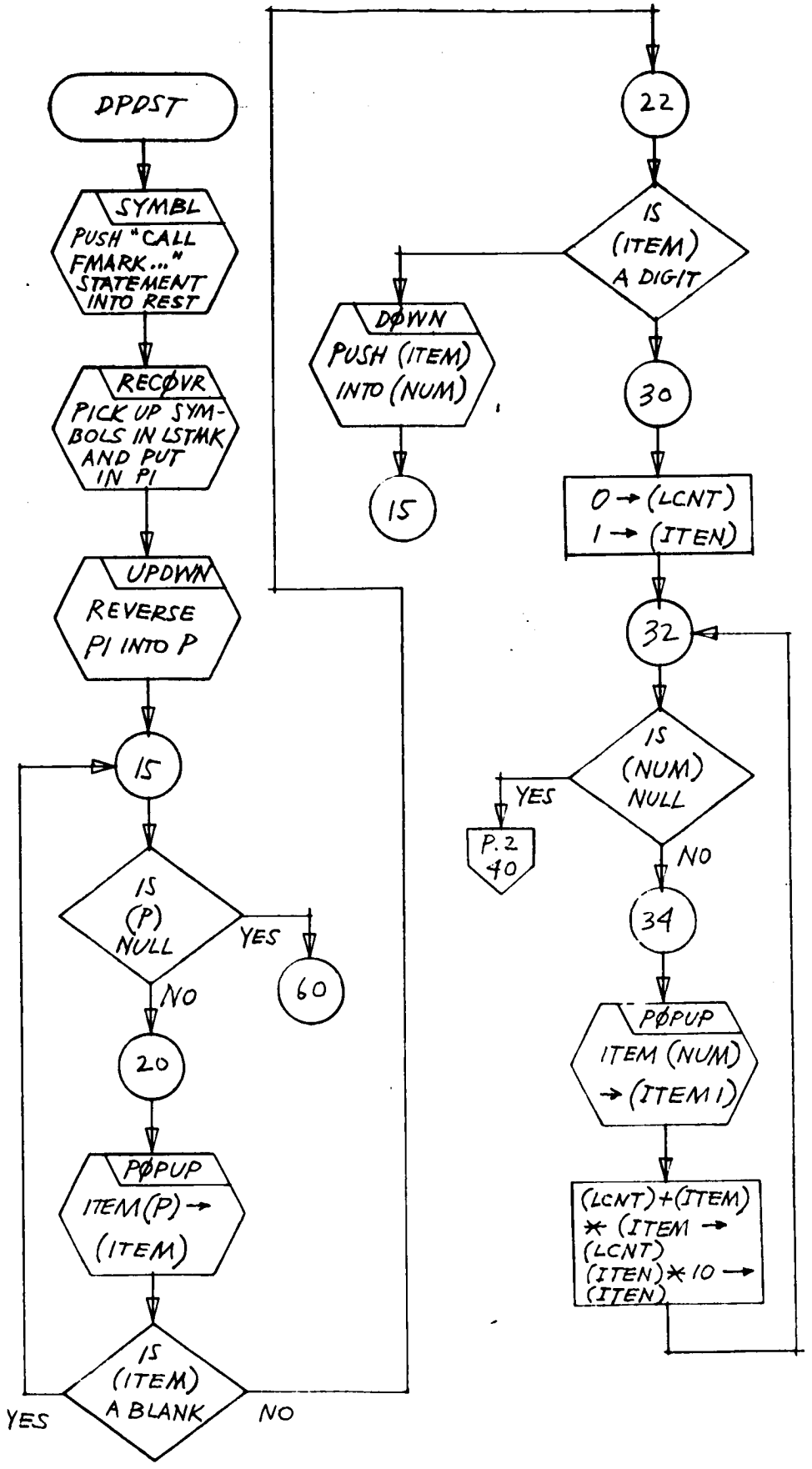
CALL FMARK(LCNT, FHB, 0, LTYPE, 1, 1, 2, FHC, 3, FT, FSTOP, 1, FLXST1, 0, 2, FLXST2, 0, 0)

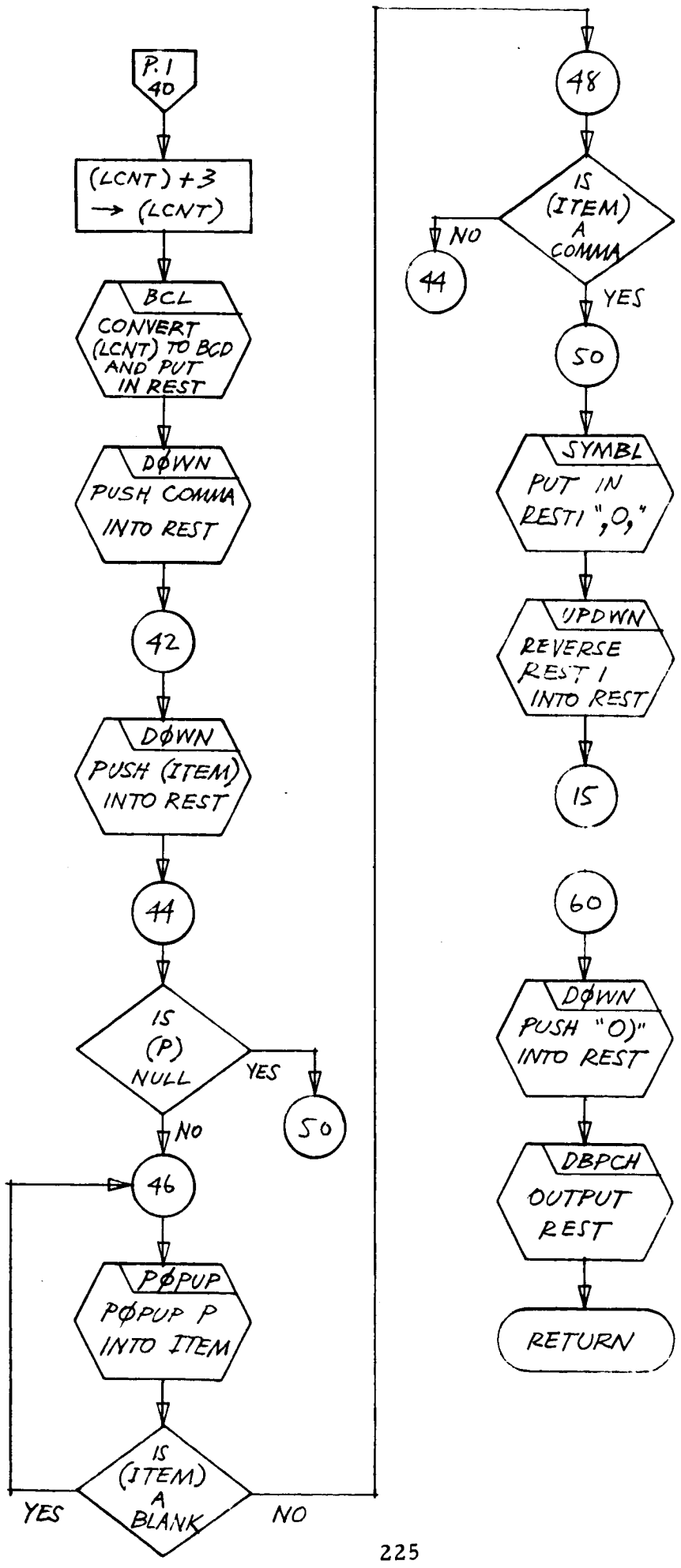
7. Other Subroutines Used

BCL, DBPCH, DOWN, ERASE, POPUP, RECOVR, SYMBL, SYMCH, UPDWN.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


DPDST
SYMBL
PUSH "CALL FMARK..." STATEMENT INTO REST
RECØVR
PICK UP SYMBOLS IN LSTMK AND PUT IN PI
UPDWN
REVERSE PI INTO P
15
IS (P) NULL
YES
60
NO
20
PØPUP
ITEM(P) → (ITEM)
IS (ITEM) A BLANK
YES
NO
22
IS (ITEM) A DIGIT
DØWN
PUSH (ITEM) INTO (NUM)
15
30
0 → (LCNT)
1 → (ITEN)
32
IS (NUM) NULL
YES
P.2 40
NO
34
PØPUP
ITEM (NUM) → (ITEM 1)
(LCNT)+(ITEM) * (ITEN → (LCNT)
(ITEN) * 10 → (ITEN)


P.I 40
(LCNT) + 3 → (LCNT)
BCL
CONVERT (LCNT) TO BCD AND PUT IN REST
DØWN
PUSH COMMA INTO REST
42
DØWN
PUSH (ITEM) INTO REST
44
IS (P) NULL
YES
NO
50
46
PØPUP
PØPUP P INTO ITEM
IS (ITEM) A BLANK
YES
NO
48
IS (ITEM) A COMMA
NO
44
YES
50
SYMBL
PUT IN REST1 ",0,"
UPDWN
REVERSE REST1 INTO REST
15
60
DØWN
PUSH "0)" INTO REST
DBPCH
OUTPUT REST
RETURN

Program Description

1. Identification

   a. Routine Label

      ELIM

   b. Name

      Eliminate duplicate symbols.

2. Function

   Merges two symbol-lists, deleting all common names.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL ELIM (IN, IN1)

   b. Entry Conditions

      IN is a list of characters in list format C.

      IN1 is a list of characters in list format D. It contains a sequence of variable names separated by commas.

   c. Exit Conditions

      The symbols in IN1 are merged into IN. Duplicate names are absorbed, and new names from IN1 are constructed in IN in format C. IN1 is erased.

   d. Error Exits

      None.

5. Definition of Identifiers

ICM	BCD comma
LO	List; used to link down through IN1

L1	Points to the current sublist of LO
L2	Temporarily used to hold new list constructions
L3	List; points at a subportion of L5
L4	Temporarily used to hold the item portion of L3
L5	List; points at current portion of IN1 being examined
ITEM1	Holds the current character in IN
ITEM2	Holds the current character in IN1

6. Method

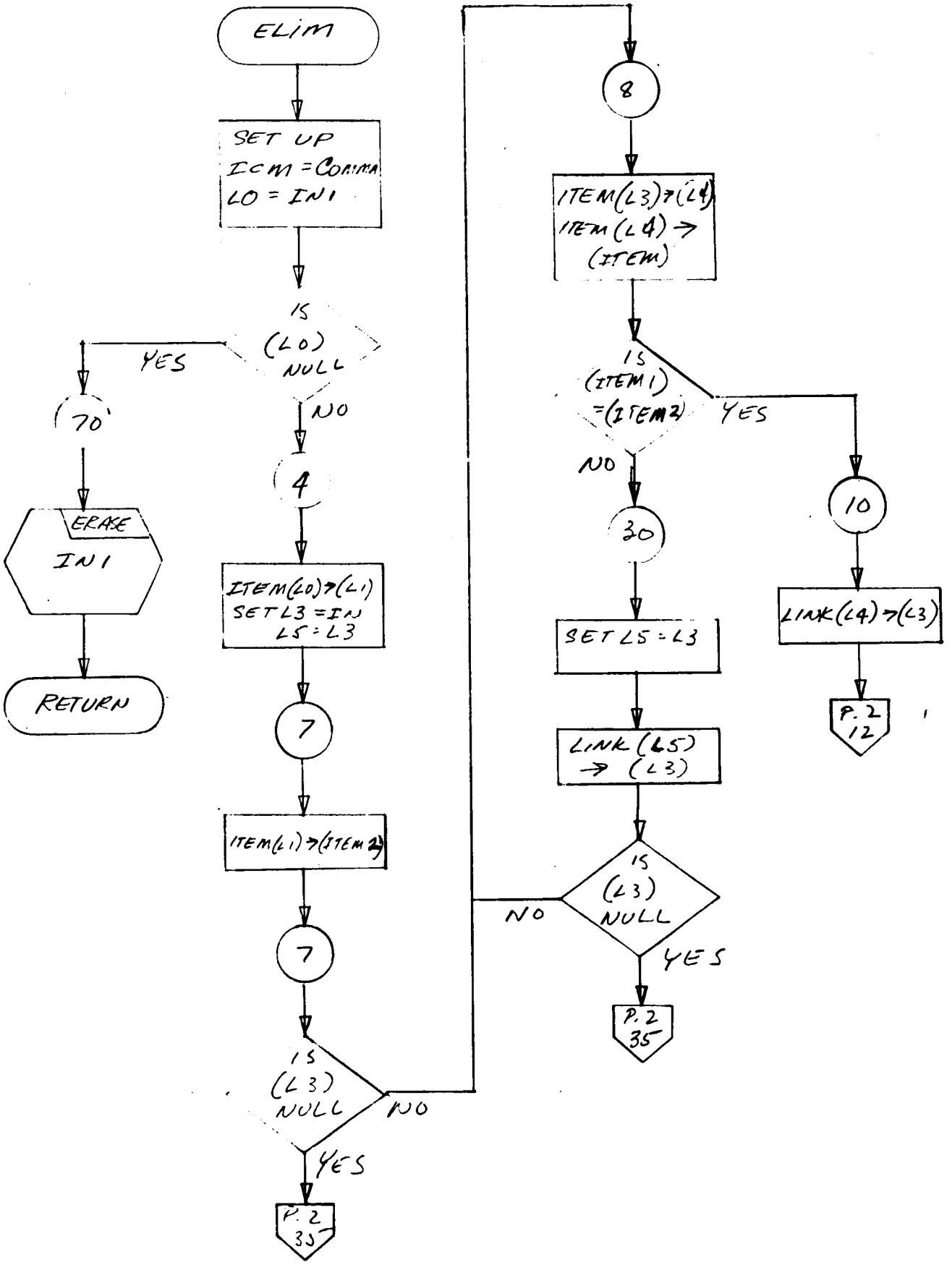
The list IN1 is examined, one sublist (symbol string) at a time. The variable name in this string is placed in L1 and is used to examine IN. L5 and L3 are set to start with IN. ITEM1 is set equal to the item of the item of L3. ITEM2 is set equal to the item of L1. If ITEM1 and ITEM2 are identical, L3 is set to point to the link of the item of L3, L1 is set to the link of L1, and this process is repreated. If L1 becomes null, then the next symbol on L0 is acquired for processing. If L3 becomes null, then the remainder of L1 is appended to L3. If, on comparison, ITEM1 and ITEM2 are not identical, then L5 is set = L3, and L3 is set = link(L3), and the process continues. Subroutine terminates when L0 is null.

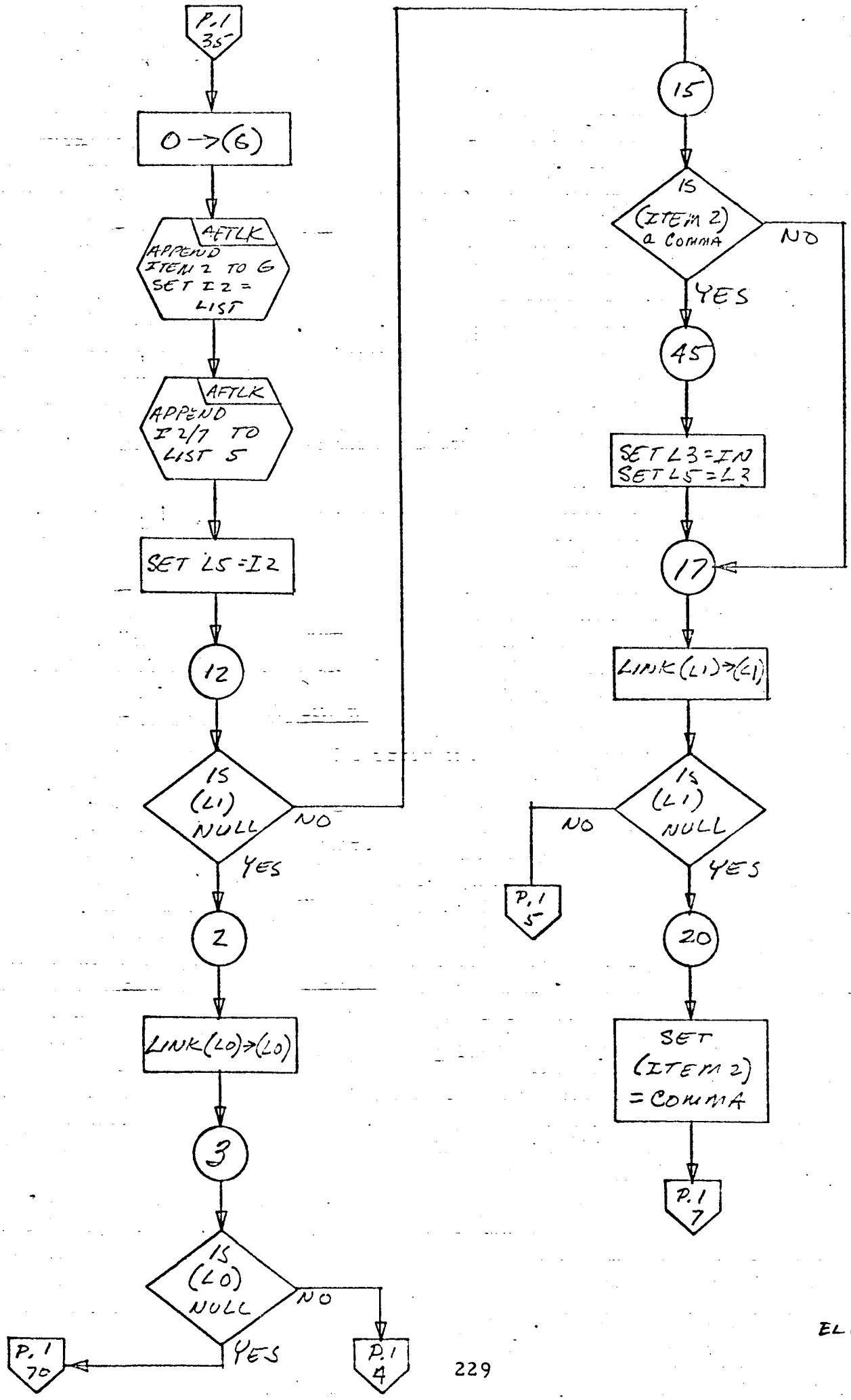
7. Other Subroutines Used

AFTLK, ERASE, FROM, LINK, SYMCH.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor SUBST.


ELIM
SET UP
ICM = COMMA
LO = IN1
IS (L0) NULL
YES
NO
70
ERASE IN1
RETURN
4
ITEM(L0)→(L1)
SET L3 = IN
L5 = L3
7
ITEM(L1)→(ITEM2)
7
IS (L3) NULL
NO
YES
P. 2 35
8
ITEM(L3)→(L4)
ITEM(L4)→(ITEM)
IS (ITEM1) =(ITEM2)
YES
NO
30
SET L5 = L3
LINK(L5) → (L3)
IS (L3) NULL
NO
YES
P. 2 35
10
LINK(L4)→(L3)
P. 2 12


P.1 35-
0 -> (6)
AFTLK
APPEND ITEM 2 TO 6 SET I2 = LIST
AFTLK
APPEND I2/7 TO LIST 5
SET L5 = I2
12
IS (L1) NULL
NO
YES
2
LINK (L0) -> (L0)
3
IS (L0) NULL
NO
YES
P.1 70
P.1 4
15
IS (ITEM 2) A COMMA
NO
YES
45
SET L3 = IN
SET L5 = L3
17
LINK (L1) -> (L1)
IS (L1) NULL
NO
YES
P.1 5-
20
SET (ITEM 2) = COMMA
P.1 7

ELIM 2

Program Description

1. Identification

   a. Routine Label

      EQFS41

   b. Name

      Write equation using FS41.

2. Function

   Outputs a matrix-multiply equation by using WRTEQ.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL EQFS41(NPT)

   b. Entry Conditions

      NPT is the NPT array of Main number 2.

   c. Exit Conditions

      Statements output.

   d. Error Exits

      None.

5. Definition of Identifiers

HFS41	Holds left part of equation: "FS41 = FLI44*(FI41
SFS41	Holds right part of euqation: " -FL41*FS11$ - FL42*FS21$ - FL43*FS31$

6. Method

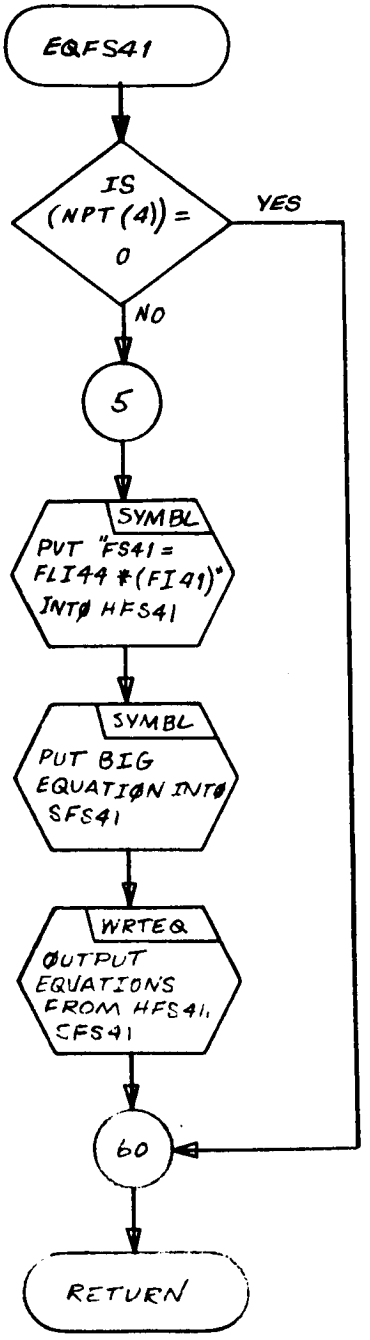
If NPT(4) = 0, exit. Else call WRTEQ: CALL WRTEQ(HFS41, SFS41, 0, 4)

7. Other Subroutines Used

SYMBL, WRTEQ.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


EQFS41
IS (NPT(4)) = 0
YES
NO
5
SYMBL
PUT "FS41 = FLI44 *(FI41)" INTO HFS41
SYMBL
PUT BIG EQUATION INTO SFS41
WRTEQ
OUTPUT EQUATIONS FROM HFS41, CFS41
60
RETURN

Program Description

1. Identification

   a. Routine Label

      EQFV21

   b. Name

      Write equation using FV21.

2. Function

   Outputs a matrix-multiply equation by using WRTEQ.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL EQFV21(NPT)

   b. Entry Conditions

      NPT is the NPT array of Main number 2.

   c. Exit Conditions

      Statements output.

   d. Error Exits

      None.

5. Definition of Identifiers

HFV21	Holds left part of equation: "FVD21 = FCI22*(FI21
SFV21	Holds right part of equation: " -FG21*FV11$ - FG22*FV21$ - FG23*FV31$ -FL21*FS11$ - FL22*FS21$ - FL23*FS31$ -FL24*FS41$ - FC21*FVD11$"

6. Method

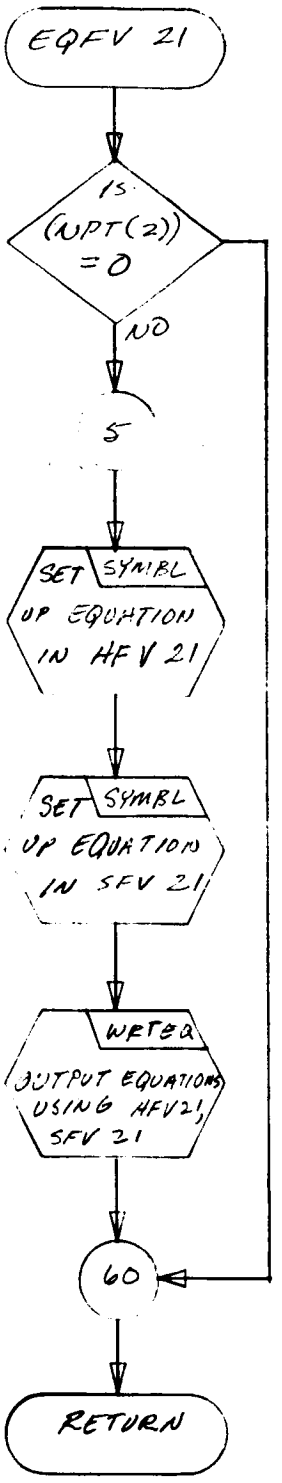
If NPT(2) = 0, exit. ELSE call WRTEQ: CALL WRTEQ(HFV21, SFV21, NPT, 0, 2).

7. Other Subroutines Used

SYMBL, WRTEQ.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


EQFV 21
IS (NPT(2)) =0
NO
5
SET SYMBL UP EQUATION IN HFV 21
SET SYMBL UP EQUATION IN SFV 21
WRTEQ OUTPUT EQUATIONS USING HFV21, SFV 21
60
RETURN

Program Description

1. Identification

   a. Routine Label

      EQFV31

   b. Name

      Write equation using FV21.

2. Function

   Outputs a matrix-multiply equation by using WRTEQ.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL EQFV31(NPT)

   b. Entry Conditions

      NPT is the NPT array of Main number 2.

   c. Exit Conditions

      Statements output.

   d. Error Exits

      None.

5. Definition of Identifiers

HFV31	Holds left part of equation: "FV31 = FGI33*(FI31
SFV31	Holds right part of equation: " -FG31*FV11$ - FG32*FV21$ - FL31*FS11$ -FL32*FS21$ - FL33*FS31$ - FL34*FS41$"

6. Method

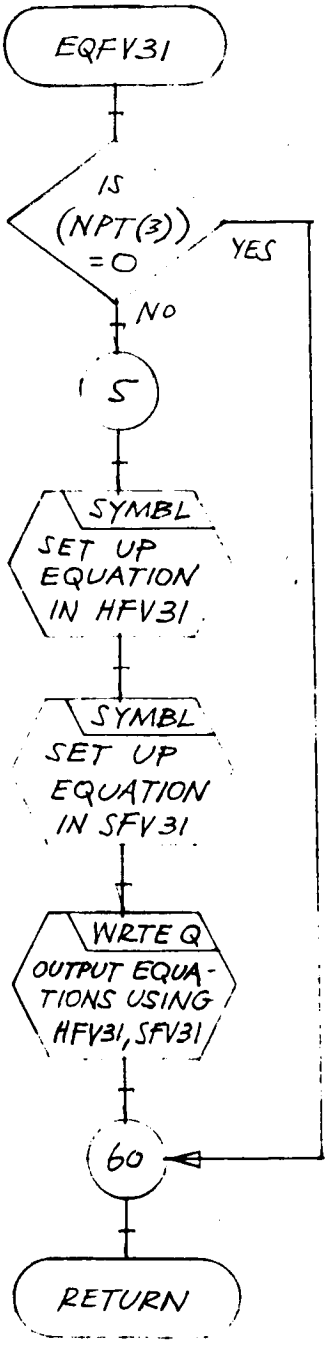
If NPT(3) = 0, exit. Else call WRTEQ: CALL WRTEQ (HFV31, SFV31, NPT, 0, 3).

7. Other Subroutines Used

SYMBL, WRTEQ.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


EQFV31
IS
(NPT(3))
=0
YES
NO
15
SYMBL
SET UP
EQUATION
IN HFV31
SYMBL
SET UP
EQUATION
IN SFV31
WRTE Q
OUTPUT EQUA-
TIONS USING
HFV31, SFV31
60
RETURN

Program Description

1. Identification

   a. Routine Label

      ERASE

   b. Name

      ERASE a list.

2. Function

   This subroutine restores all elements of a specified list, or list structure, to AVS.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL ERASE (C)

   b. Entry Conditions

      C = Head of a list, or list structure

   c. Exit Conditions

      All elements of the list or list structure, pointed to by C, are restored to AVS. Head cell C is set to zero.

   d. Error Exits

      None.

5. Definition of Identifiers

   Z = Temporary cell used as the head of a list each of whose elements point to type 7 elements of a list structure.

   C1, C2 = Temporary head cell used to point to the next element to be restored to AVS.

   E1 = The first element of a list pointed to by C1.

S2 = A cell in whose decrement the item portion of E1 is saved.

K2 = A cell in whose decrement the flag portion of E1 is saved.

IP = A cell whose decrement contains the location of the second element of a list pointed to by C1. This element contains a full word of data and is the end of a list. The element linked to this last element is a type 6.

6. Method

a. Z is initially set = 0.

b. If list C is null, there is no list to be erased, and return is made to the routine that called ERASE.

c. If list C is not null, the following steps are executed:

(1) The pointer in C is saved in C2 and C is set = 0. (C, the head cell of the list to be erased, is made "null.")

(2) The loop for erasing a list is initialized by moving the pointer in C2 to C1, which makes C1 the new head of the remainder of the list.

(3) The item and flag of the first element in C1 is saved in S2 and K2, respectively.

(4) The flag in K2 is tested:

(a) If K2 = 7, an element is pushed down into list Z whose item contains the decrement of C1 (pointer to the type 7 element). S2, whose decrement points to the next sublist to be erased, is stored in C2, and erasure proceeds again at step c(2).

(b) If K2 = 6, there are only two more elements to be erased in the list pointed to by C1, the last one being a full word of data (e.g., a floating point number). After these two elements are restored to AVS, execution continues at step c(5) with a test to determine if list Z is null.

(c) If K2 ≠ 6 or 7, the element pointed to by C1 is restored to AVS and its link is saved in C2. When C2 = 0, the end of a list or sublist has been reached and execution continues at step c(5). If C2 ≠ 0, execution continues at step c(2) where the erasure loop is re-initialized.

(5) The head cell Z is tested:

(a) If Z is null, the end of the list has been reached; therefore exit from the routine.

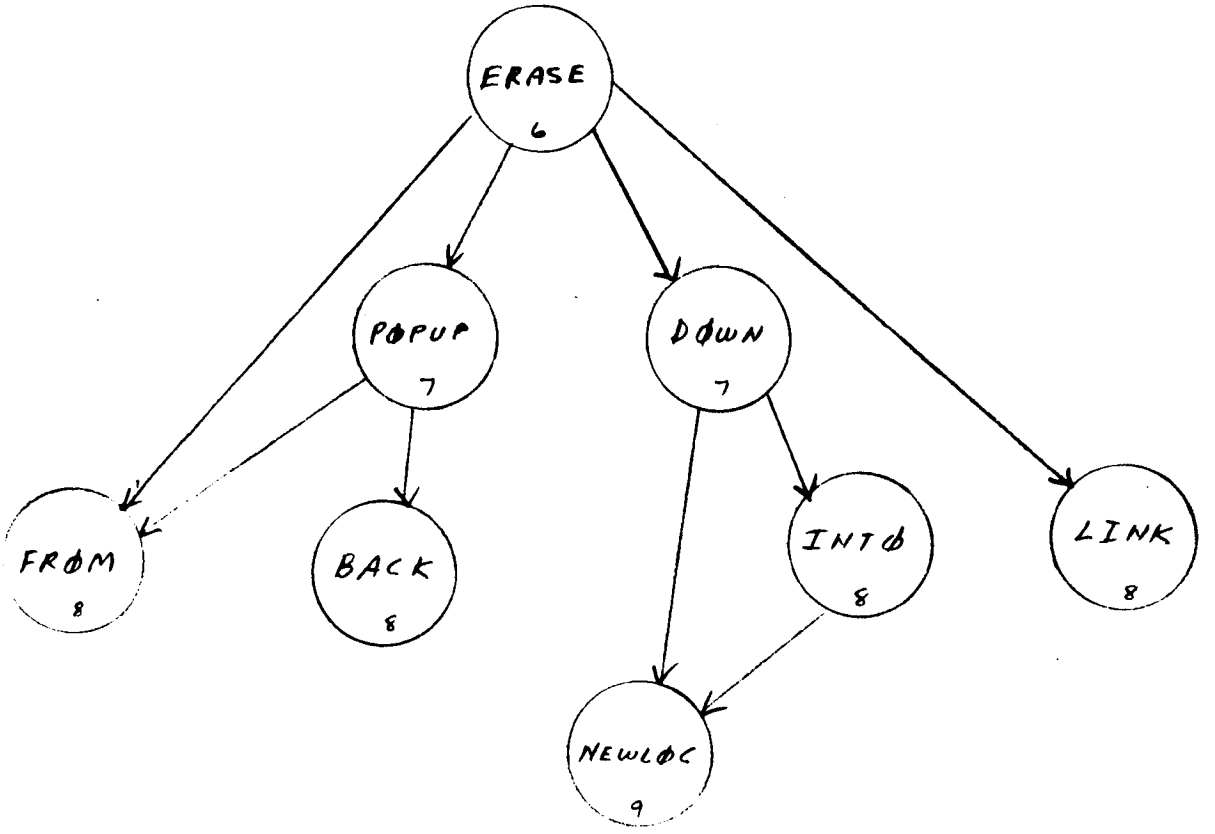
(b) If Z is not null, one more element is popped up from Z, giving the pointer to the next type 7 element to be erased. This pointer is stored in C1, which becomes the temporary head cell of the list whose first element is the type 7 element to which it points. The flag of this type 7 element is ignored (as if it were less than 6) and the process proceeds to step c(4) (c) to erase the type 7 element and all elements appended to it.

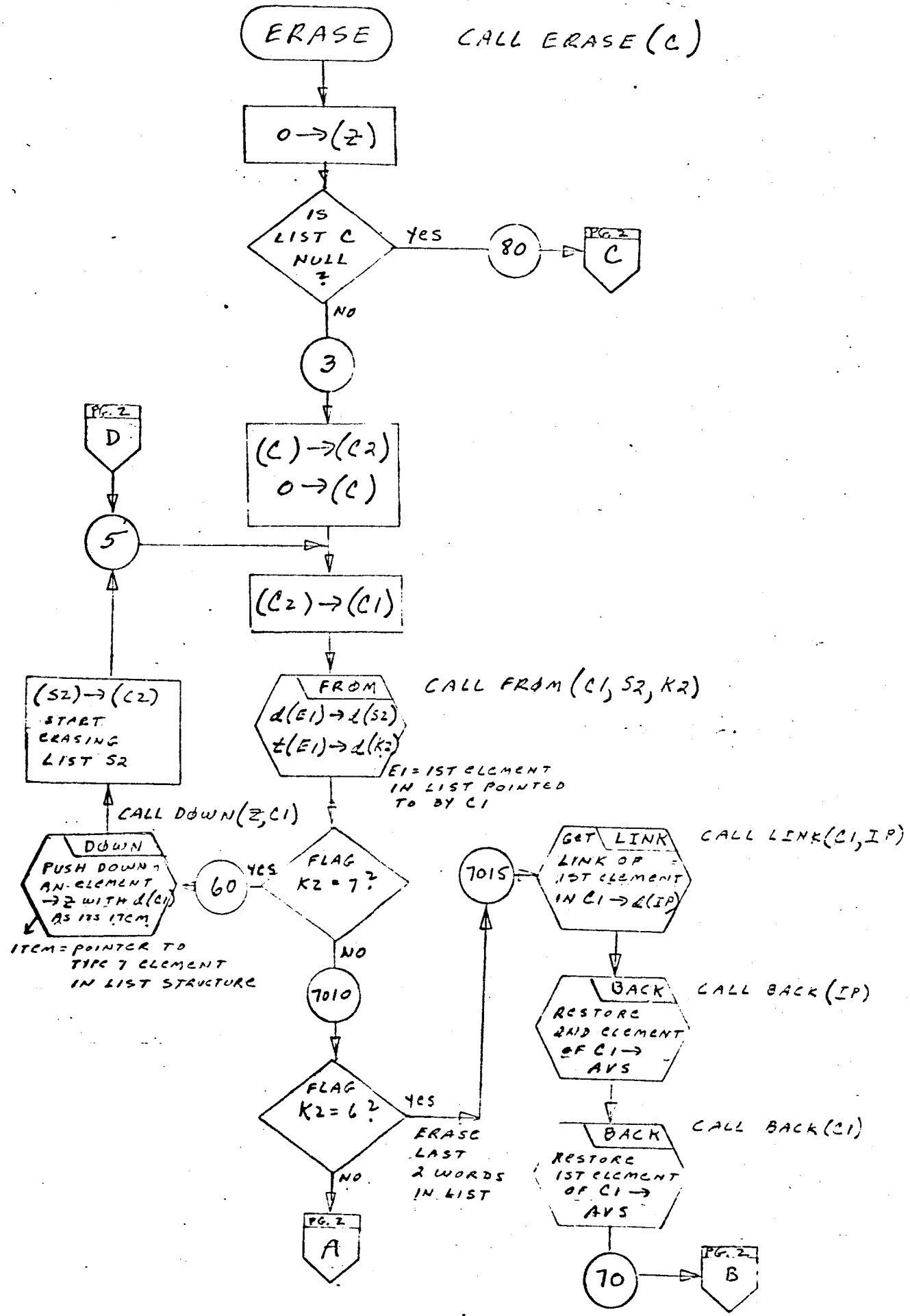
7. Other Subroutines Used

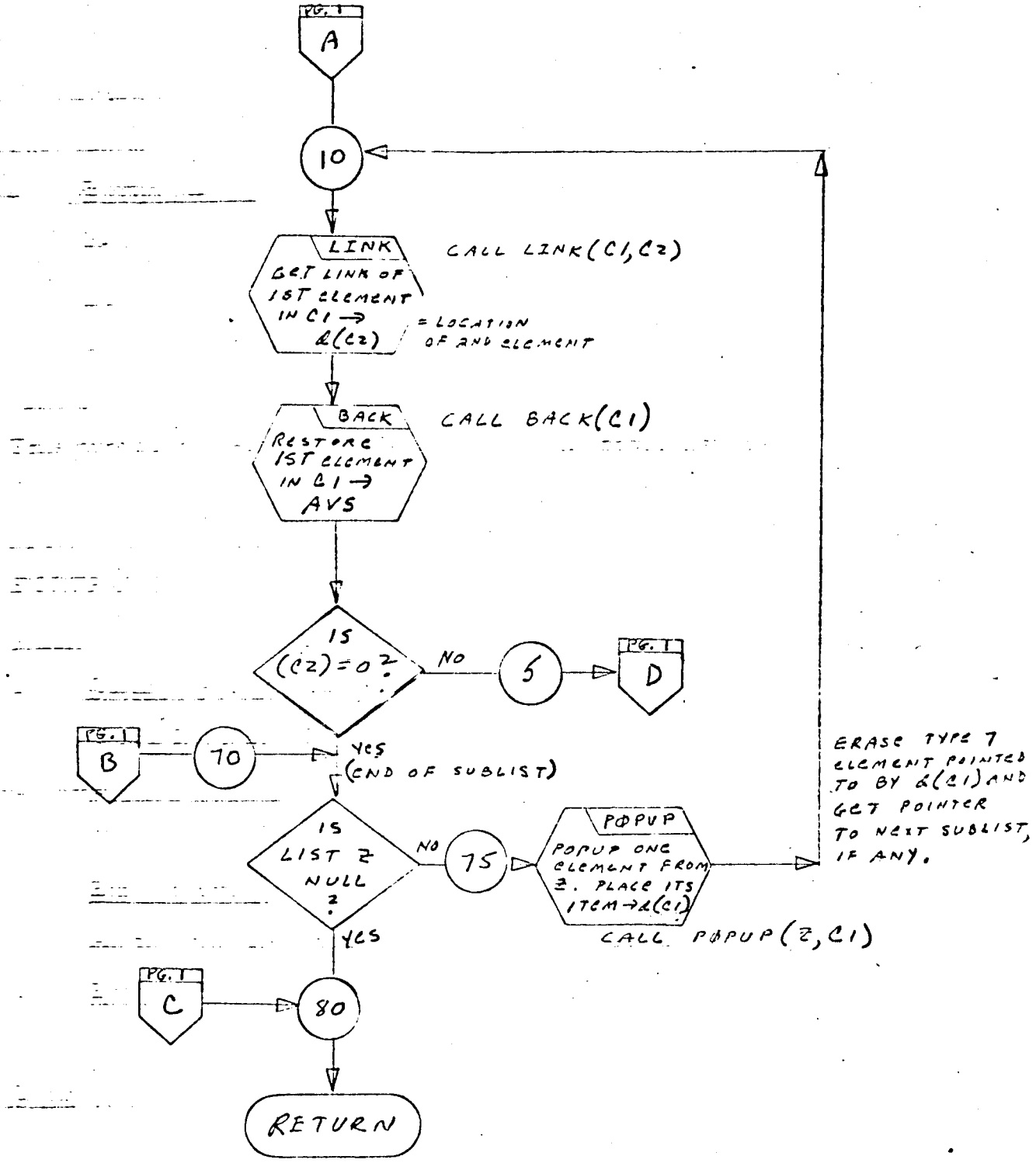
BACK, DOWN, FROM, LINK, POPUP.

8. Using Subroutines

BLNOUT, COMBN, DBPCC, DBPCH, DIMEN, DONBD, DPDST, ELIM, ERASEA, EXCPT, EXTRX, GOBLE, HOLBK, INPUTX, INTLST, INZERO, MULTS, NLINDM, PUSPCH, READCH, RECOVR, SEGMNT, STRIK, SUBST, SYMBL, SYMCRD, SYMTP, TREE, WRTEQ, Main Routines for TAG Pass 1 and Pass 2.


ERASE
6
POPUP
7
DOWN
7
FROM
8
BACK
8
INTO
8
LINK
8
NEWLOC
9


ERASE
CALL ERASE (C)
0 → (Z)
IS LIST C NULL ?
Yes
80
PG. 2 C
NO
3
PG. 2 D
5
(C) → (C2)
0 → (C)
(C2) → (C1)
FROM
CALL FROM (C1, S2, K2)
d(E1) → d(S2)
t(E1) → d(K2)
E1 = 1ST ELEMENT IN LIST POINTED TO BY C1
(S2) → (C2)
START ERASING LIST S2
CALL DOWN(Z, C1)
DOWN
PUSH DOWN AN ELEMENT → Z WITH d(C1) AS ITS ITEM
60
Yes
FLAG K2 = 7 ?
ITEM = POINTER TO TYPE 7 ELEMENT IN LIST STRUCTURE
NO
7010
7015
GET LINK
LINK OF 1ST ELEMENT IN C1 → d(IP)
CALL LINK(C1, IP)
BACK
RESTORE 2ND ELEMENT OF C1 → AVS
CALL BACK (IP)
BACK
RESTORE 1ST ELEMENT OF C1 → AVS
CALL BACK (C1)
FLAG K2 = 6 ?
Yes
ERASE LAST 2 WORDS IN LIST
NO
PG. 2 A
70
PG. 2 B


PG. 1
A
10
LINK
GET LINK OF 1ST ELEMENT IN C1 → &(C2)
= LOCATION OF AND ELEMENT
CALL LINK(C1,C2)
BACK
RESTORE 1ST ELEMENT IN C1 → AVS
CALL BACK(C1)
IS (C2) = 0 ?
NO
5
PG. 1
D
PG. 1
B
70
YES (END OF SUBLIST)
IS LIST Z NULL ?
NO
75
POPUP
POPUP ONE ELEMENT FROM Z. PLACE ITS ITEM → &(C1)
CALL POPUP(Z,C1)
ERASE TYPE 7 ELEMENT POINTED TO BY &(C1) AND GET POINTER TO NEXT SUBLIST, IF ANY.
YES
PG. 1
C
80
RETURN

Program Description

1. Identification

   a. Routine Label

      ERASEA

   b. Name

      ERASE an array of lists.

2. Function

   This subroutine restores to AVS the elements of the lists associated with a vector of 100 head cells.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL ERASEA (C)

   b. Entry Conditions

      C(I) = A vector of 100 head cells

   c. Exit Conditions

      All non-null lists pointed to by C(I) are restored to AVS.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

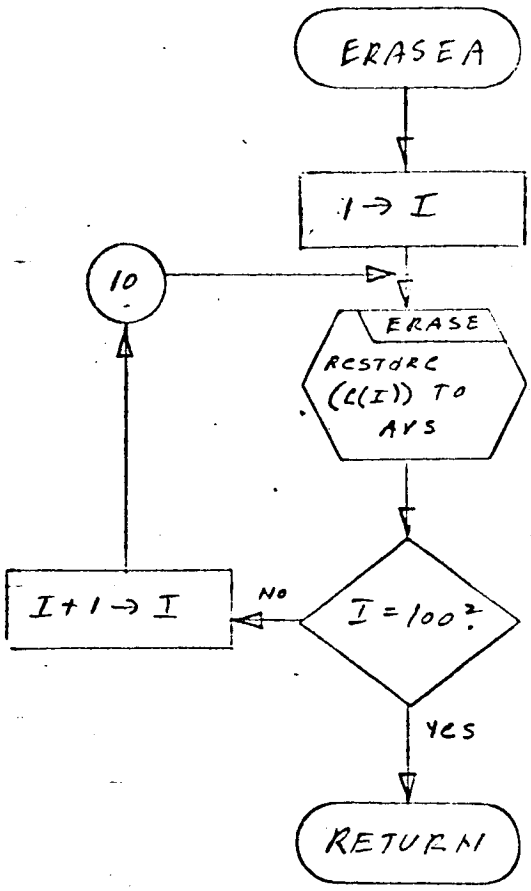
   CALL ERASE (C(I)) is executed 100 times, (I) being initialized to 1 and increased by 1 each time ERASE is called. The 100 head cells, C(I), are a FORTRAN array.

7. Other Subroutines Used

ERASE

8. Using Subroutines

COTRN, LEVMRK, MULTS, PARAM, STRIK, SYMCRD, TRANS, XFORM, Main Programs for Pass 1 and Pass 2 of TAG Preprocessor.



CALL ERASEA(C)

Program Description

1. Identification

   a. Routine Label

      EXCPT

2. Function

   To determine the matrix indices from examination of a circuit element name.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call EXCPT (TEM, NPT, KYN, N1, N2)

   b. Entry Conditions

      TEM A list (type A) containing the characters of the circuit element name in reverse order.

      NPT The NPT array of Main number 2.

      KIND (in COMMON) is an indicator.

   c. Exit Conditions

      N1 holds the first matrix index.

      N2 holds the second matrix index.

      KYN indicates the status of N1, N2.

   d. Error Exits

      None.

5. Definition of Identifiers

   TEM1 A list that is a copy of TEM

   ITEM Used to hold the first item of TEM

   ITMN2 Used to hold the previous contents of ITEM

   ICK Used to hold the characters C, L, G, in succession, for examination

6. Method

TEM contains a variable of the form

w x y z

where w and x are letters, and y and z are digits.

N1 is set = NPT(y).

N2 is set = NPT(z)

If either y or z is not a digit, set KYN = 2 and return

If x is not a "C," "L," or "G," and z ≤ 1,

then set N2 = 1.

If W is an "S" and KIND ≤ 1, then set N2 = 0.

Set KYN = 1 and return.

Example:

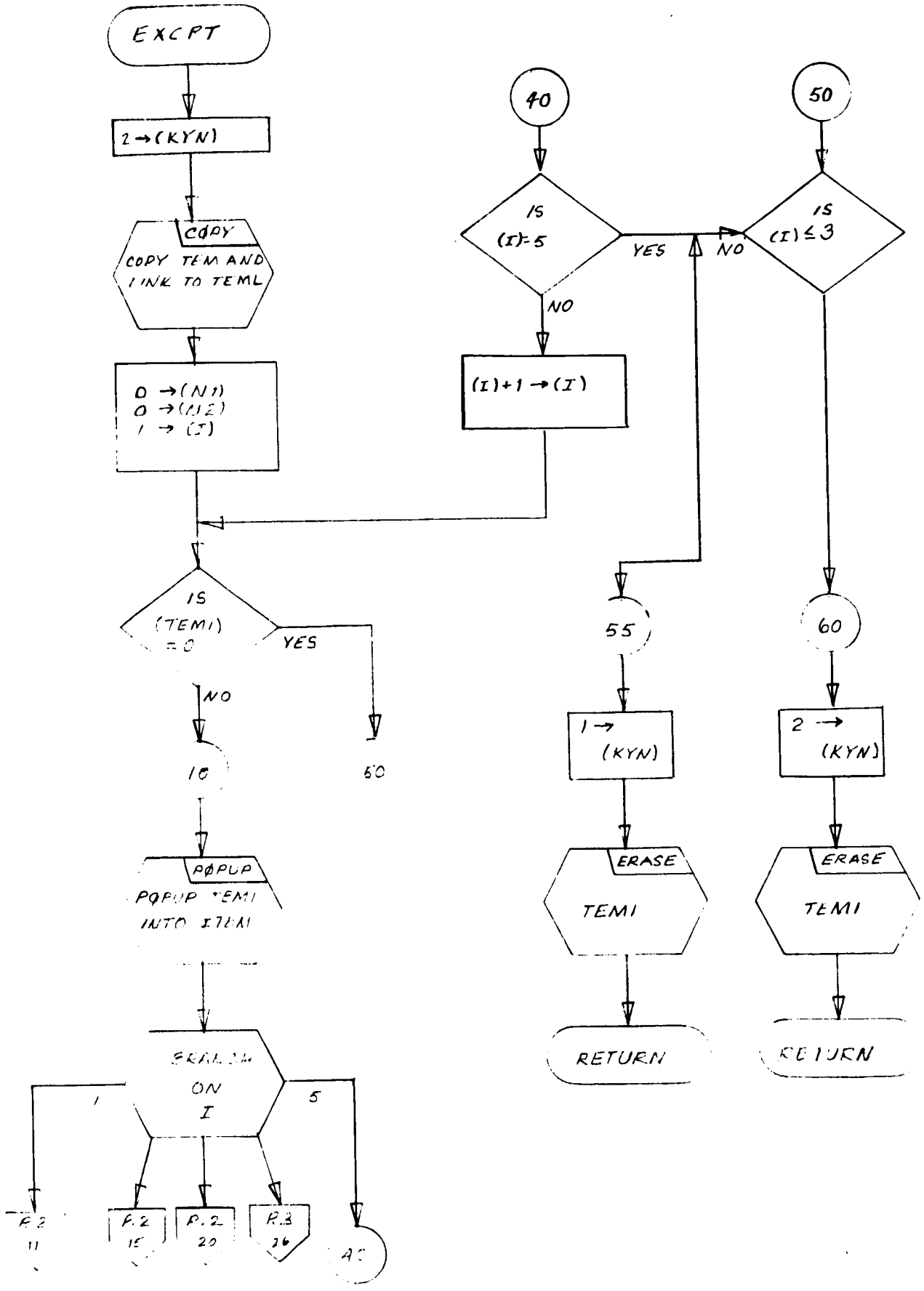
TEM	KYN	N1	N2
SG 3 4	1	NPT(3)	NPT(4)
FPT	2		

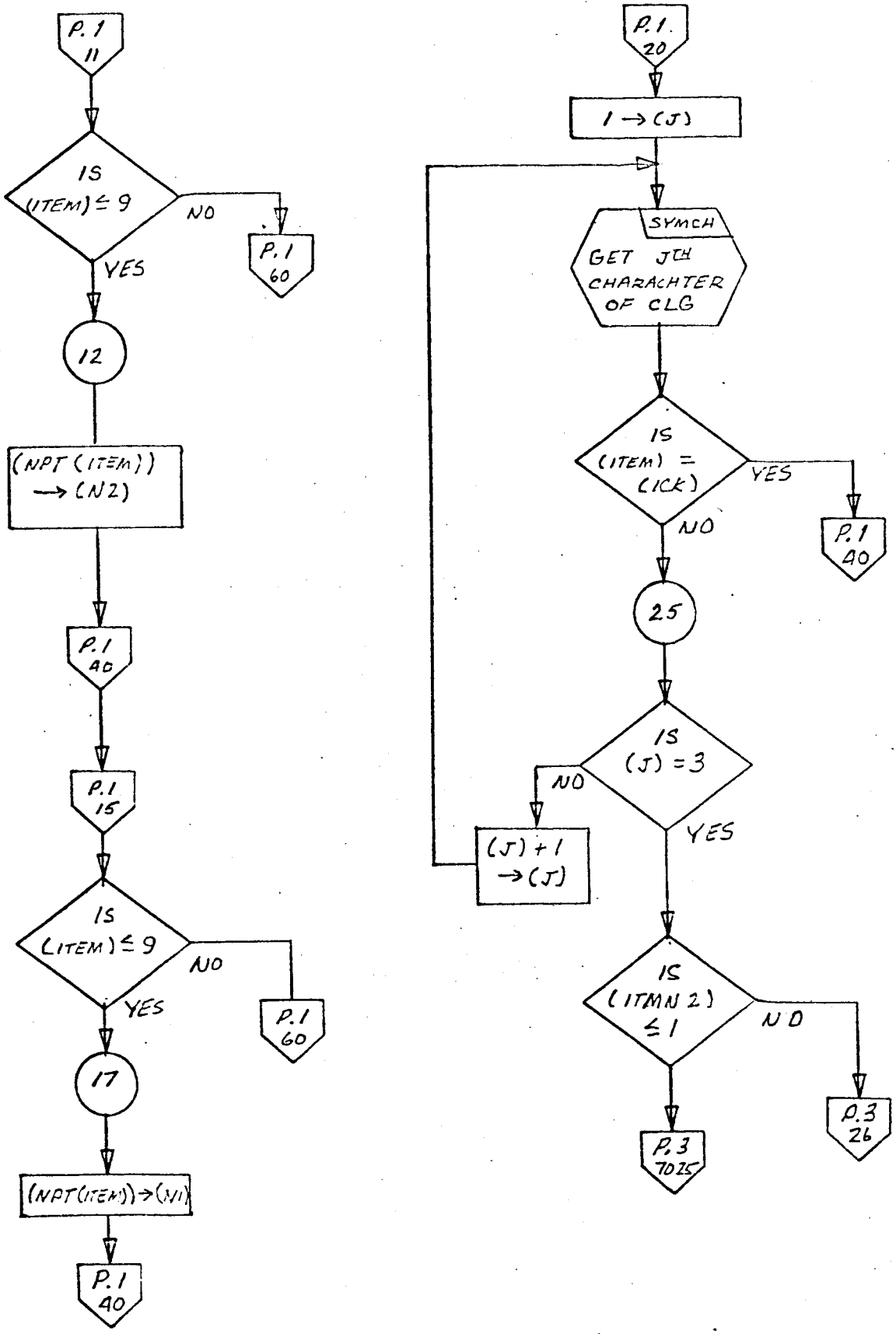
7. Other Subroutines Used

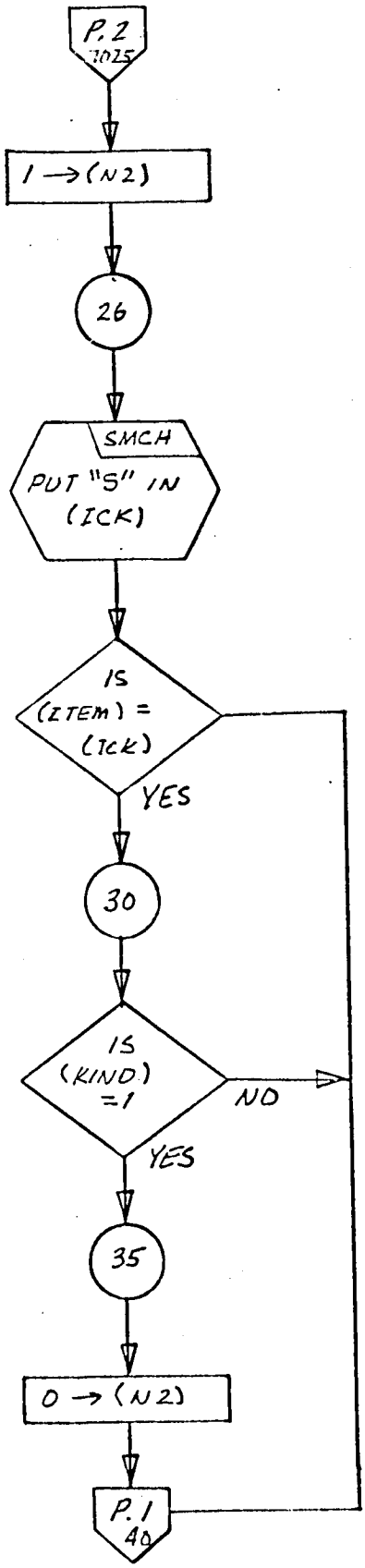
COPY, ERASE, POPUP, SYMCH.

8. Using Subroutines

DIMEN, SYMCRD, WRTEQ, ZEROX .


EXCPT
2→(KYN)
COPY
COPY TEM AND
LINK TO TEML
0→(N1)
0→(N2)
1→(I)
IS
(TEMI)
=0
YES
NO
10
50
PØPUP
PØPUP TEMI
INTO ITEMI
ON
I
1
5
P.2
11
P.2
15
P.2
20
P.3
26
40
IS
(I)=5
YES
NO
(I)+1→(I)
50
IS
(I)≤3
55
60
1→
(KYN)
2→
(KYN)
ERASE
TEMI
ERASE
TEMI
RETURN
RETURN


P.1 11
IS (ITEM) ≤ 9
NO
YES
P.1 60
12
(NPT (ITEM)) → (N2)
P.1 40
P.1 15
IS (ITEM) ≤ 9
NO
YES
P.1 60
17
(NPT(ITEM)) → (N1)
P.1 40
P.1 20
1 → (J)
SYMCH
GET JTH CHARACHTER OF CLG
IS (ITEM) = (ICK)
YES
NO
P.1 40
25
IS (J) = 3
NO
YES
(J) + 1 → (J)
IS (ITMN 2) ≤ 1
NO
P.3 7025
P.3 26


P.2
1025
1 → (N2)
26
SMCH
PUT "5" IN
(ICK)
IS
(ITEM) =
(ICK)
YES
30
IS
(KIND)
=1
NO
YES
35
0 → (N2)
P.1
40

Program Description

1. Identification

   a. Routine Label

      EXTRX

   b. Name

      Extract from a list all elements preceding a dollar sign.

2. Function

The list SRC is scanned for a dollar sign, then split into two parts. The elements preceding the dollar sign are removed from SRC and become the list TEM. SRC and TEM are in reverse order, with the dollar sign removed.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL EXTRX (SRC, TEM)

   b. Entry Conditions

      SRC = Head of a list

   c. Exit Conditions

All elements preceding the dollar sign in SLC are popped up from SRC and pushed down into the new list TEM. The dollar sign is discarded. List TEM will be in reverse order from list SRC.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

List SRC is split as outlined below.

a. SRC is tested.

(1) If SRC is not null, an element is popped up and its item tested:

(a) If the item is not a dollar sign, the element containing the item is pushed down into list TEM. The procedure continues at step a.

(b) If the item is a dollar sign, list TEM is complete and exit is made from the routine.

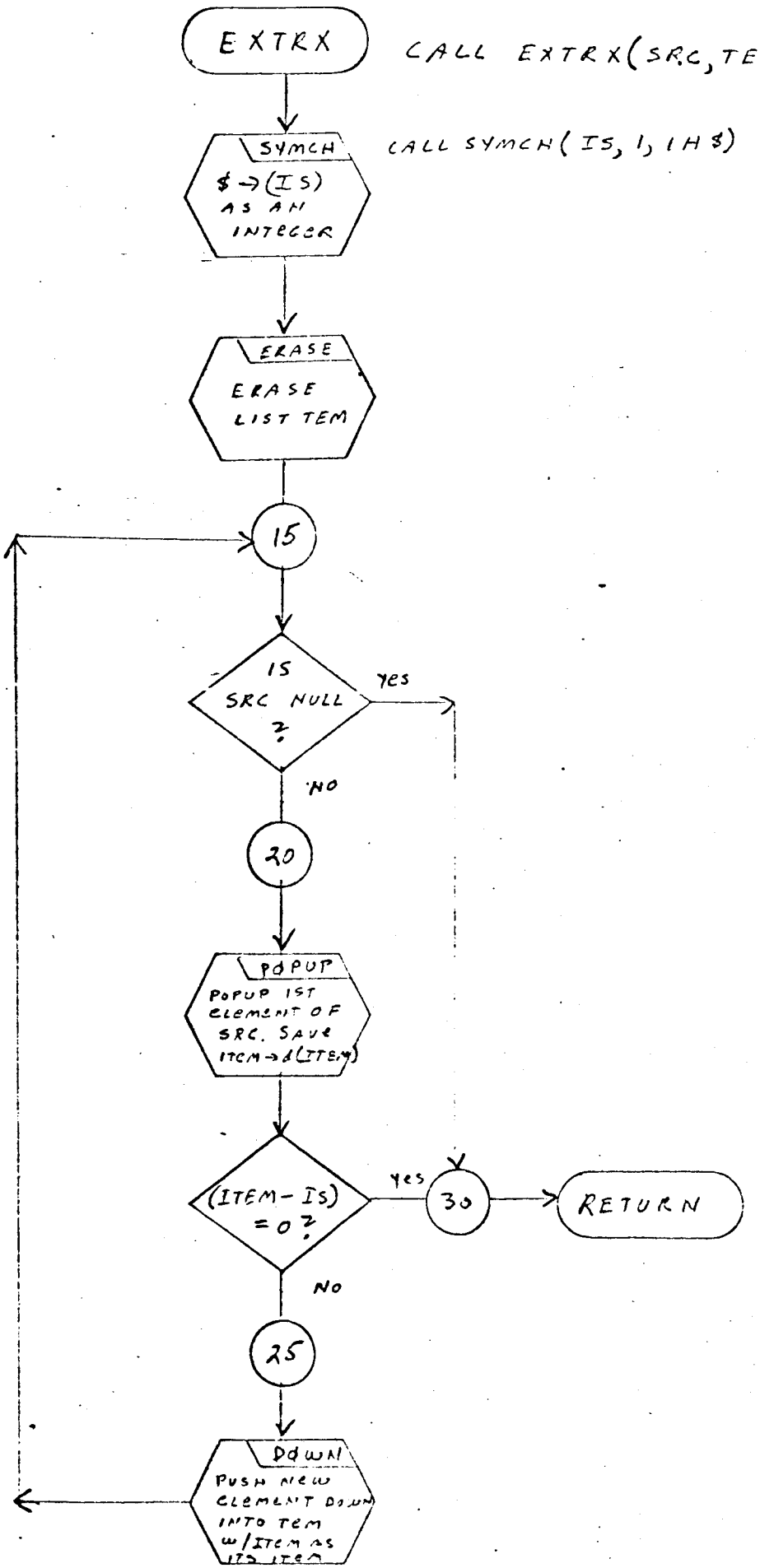
(2) If SRC is null, exit is made from the routine.

7. Other Subroutines Used

DOWN, ERASE, POPUP, SYMCH.

8. Using Subroutines

DIMEN, WRTEQ, ZEROX.


EXTRX
CALL EXTRX(SRC, TEM)
SYMCH
$ → (IS) AS AN INTEGER
CALL SYMCH(IS, 1, 1H$)
ERASE
ERASE LIST TEM
15
IS SRC NULL ?
yes
NO
20
PDPUP
POPUP 1ST ELEMENT OF SRC. SAVE ITEM → A(ITEM)
(ITEM - IS) = 0 ?
yes
30
RETURN
NO
25
PDWN
PUSH NEW ELEMENT DOWN INTO TEM W/ITEM AS ITS ITEM

Program Description

1. Identification

a. Routine Label

FISH

b. Name

Extract all the data associated with the I, J position of a matrix in array list format.

2. Function

FISH provides a process for extracting all the coefficient data, one piece at a time, from the I, J position in matrix, M, represented by a type E array list. For a two-dimensional array list, FISH is identical to SNATCH, and places in FTEM the data word associated with I, J in matrix M. For a four-dimensional array list, FISH finds the submatrix position, NN, NP, of the first unprocessed, non-zero data entry in the I, J position of matrix M, and returns the values of NN, NP and their associated data word, FTEM. In addition, ICNT is returned containing a set of pointers to the next non-zero entry in the I, J submatrix of M. Continuous applications of FISH to M(I, J) will extract on a one-at-a-time basis all the non-zero data entries in the I, J submatrix of M as well as all their corresponding submatrix indices, NN and NP.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL FISH (M, I, J, NN, NP, FTEM, ICNT)

b. Entry Conditions

M = A two-dimensional or four-dimensional matrix represented in an array list of type E format

I = Index of the row lists

J = Index of the column lists

c. Exit Conditions

NN = Negative node number.

NP = Positive node number.

ICNT = A special list whose elements keep track of the manning position of the NN, NP sublists.

When M is a two-dimensional matrix:

(1) If J was found with a flag = 6, the data word in the element linked to J is placed in FTEM. ICNT is made null and NN, NP = 0.

(2) If J was not found, ICNT will be null, NN, NP, FTEM = 0.

When M is a four-dimensional matrix:

(1) If J was found with a flag $\neq$ 6, and this was a first entry (ICNT null), return is made with FTEM along with its identifying NN, NP.

(2) If J was found with a flag $\neq$ 6 and ICNT was not null, return is made with the next non-zero FTEM along with its NN, NP.

d. Error Exits

None.

5. Definition of Identifiers

M(I)	=	Ith list structure of array-list M.
ICNT1	=	An element pushed down into ICNT whose decrement contains the location of the next NN sublist of J in M(I).
ICNT2	=	An element pushed down into ICNT whose decrement contains the location of the next NP sublist of J, NN in M(I).
IC	=	An element popped up from ICNT whose decrement gives the location of the next NN or NP sublist of J in M(I).

6. Method

The search through M(I) for FTEM and its identifying NN, NP is performed as outlined below.

a. FTEM is cleared to zero.

b. ICNT is tested. If ICNT is null, continue at the next step. If ICNT is not null, proceed to step d to continue.

c. ICNT is null. M(I) is scanned to locate J.

(1) If J cannot be found, NN, NP are cleared to zero and exit is made from the routine with no data word in FTEM.

(2) If J was found, its flag is tested to determine if a data word is appended to the element containing J.

(a) If flag (J) = 6, there is a data word. ICNT is made null, the data is placed in FTEM, NN, NP and cleared to zero, and exit is made from the routine.

(b) If flag (J) $\neq$ 6, no data word is appended to J. The search continues to the element whose item points to an NN sublist.

(i) The link of this element is saved in d (ICNT1). d(ICNT1) is pushed down into the ICNT list to keep track of the location of the next NN sublist.

(ii) NN is extracted and saved in d (NN).

(iii) The search continues to the element whose item points to an NP sublist.

- The link of this element is saved in d (ICNT2). d(ICNT2) is pushed down into the ICNT list to keep track of the location of the next NP sublists.
- NP is extracted and saved in d (NP).
- The data word in the element appended to NP is placed in FTEM.

(c) FTEM is tested.

(i) If FTEM $\neq$ 0, exit is made from the routine with FTEM and its identifying NN, NP. The first element in ICNT will contain the location of the next NN or NP sublist.

(ii) If FTEM = 0, ICNT is tested.

- o If ICNT is null, NN, NP are cleared to zero and exit is made from the routine.
- o If ICNT is not null, continue at step d.

d. ICNT is not null.

(1) The first element in ICNT is popped up into the d(IC).

(2) ICNT is tested.

(a) If ICNT is null, the d(IC) is used to initialize the location of the next NN sublist.

(i) IC is tested.

- o If IC is null, NN, NP are cleared to zero and exit is made from the routine with FTEM = 0.
- o If IC is not null, proceed to step c(2)(b)(i) to continue the scan to the element whose item points to the next NN sublist.

(b) If ICNT is null, the d(IC) is used to initialize the location of the next NP sublist.

(i) IC is tested.

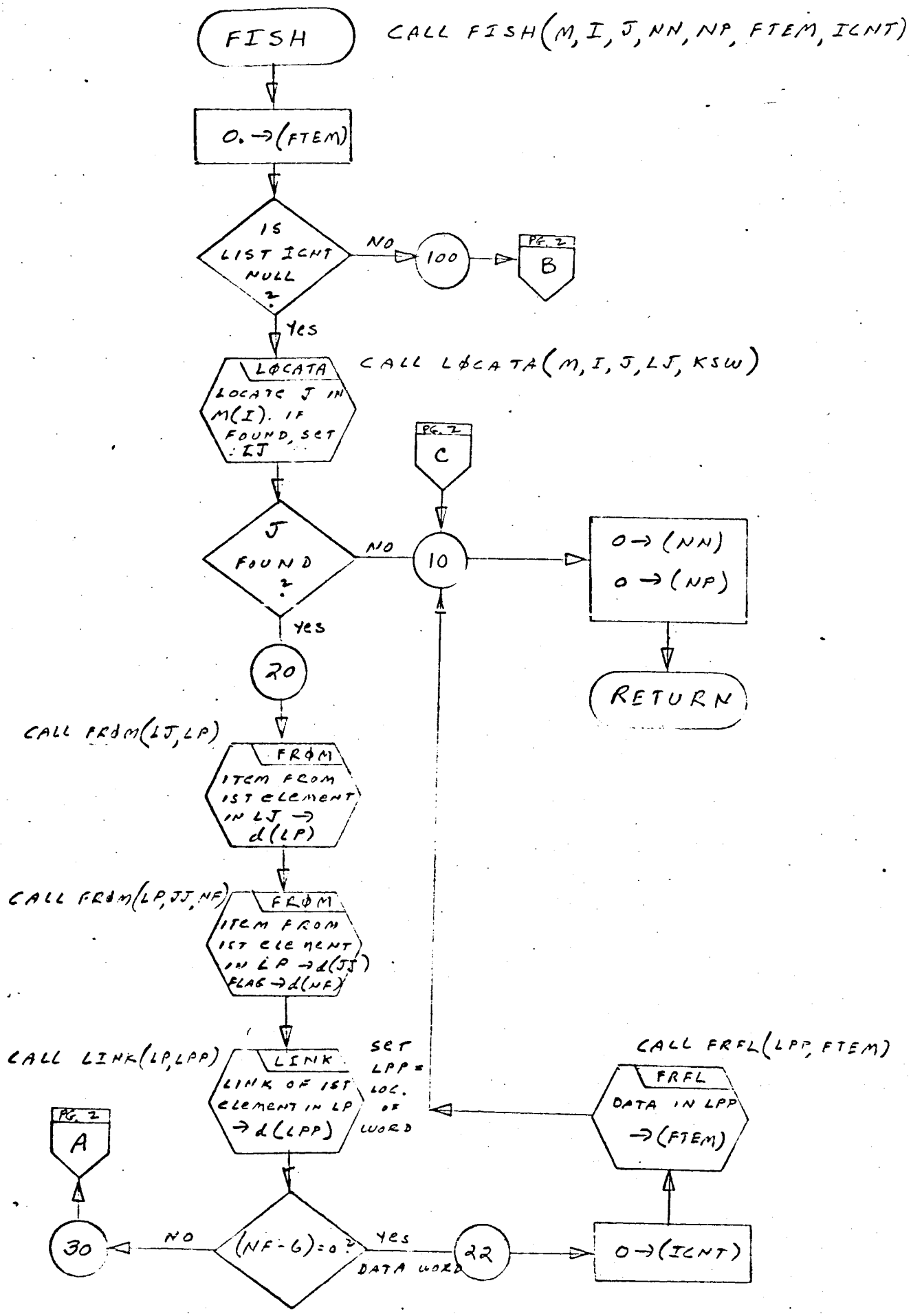
- o If IC is null, return to step d(1) to continue.
- o If IC is not null, return to step c(2)(b)(iii) to continue.

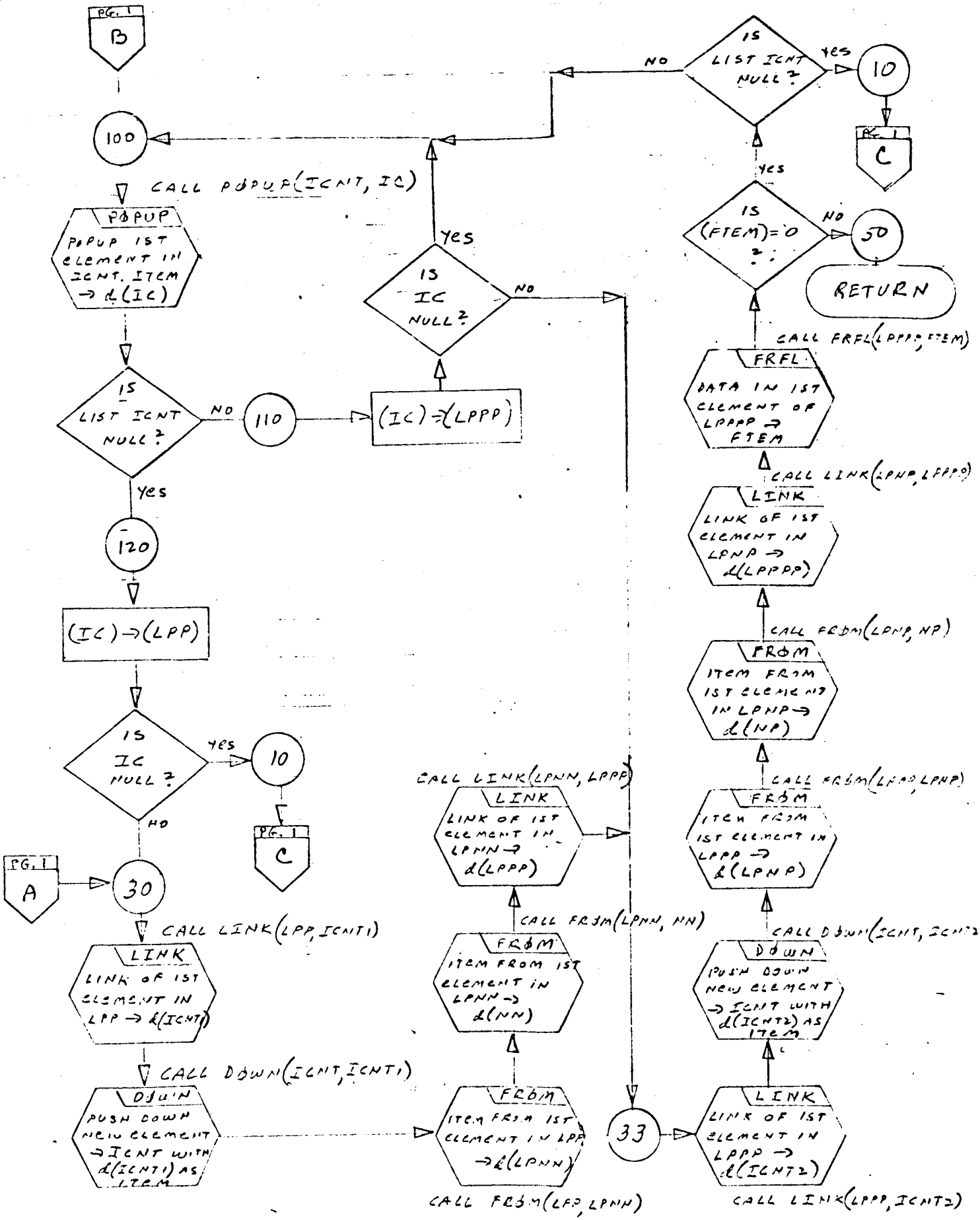
7. Other Subroutines Used

DOWN, FRFL, FROM, LINK, LOCATA, POPUP.

8. Using Subroutines

MULTS


FISH
CALL FISH(M, I, J, NN, NP, FTEM, ICNT)
0. → (FTEM)
IS LIST ICNT NULL ?
NO
100
PG. 2
B
Yes
LOCATA
LOCATE J IN M(I). IF FOUND, SET IJ
CALL LOCATA(M, I, J, LJ, KSW)
PG. 2
C
J FOUND ?
NO
10
0 → (NN)
0 → (NP)
RETURN
Yes
20
CALL FROM(LJ, LP)
FROM
ITEM FROM 1ST ELEMENT IN LJ → d(LP)
CALL FROM(LP, JJ, NF)
FROM
ITEM FROM 1ST ELEMENT IN LP → d(JJ) FLAG → d(NF)
CALL LINK(LP, LPP)
LINK
LINK OF 1ST ELEMENT IN LP → d(LPP)
set LPP = LOC. OF WORD
CALL FRFL(LPP, FTEM)
FRFL
DATA IN LPP → (FTEM)
PG. 2
A
30
NO
(NF-6)=0 ?
Yes
DATA WORD
22
0 → (ICNT)


PG. 1
B
100
CALL POPUP(ICNT, IC)
POPUP
POPUP IST ELEMENT IN ICNT, ITEM → ℒ(IC)
IS LIST ICNT NULL?
NO
110
(IC) → (LPPP)
IS IC NULL?
Yes
NO
Yes
120
(IC) → (LPP)
IS IC NULL?
yes
10
PG. 1
C
NO
PG. 1
A
30
CALL LINK(LPP, ICNT1)
LINK
LINK OF IST ELEMENT IN LPP → ℒ(ICNT1)
CALL DOWN(ICNT, ICNT1)
DOWN
PUSH DOWN NEW ELEMENT → ICNT WITH ℒ(ICNT1) AS ITEM
CALL LINK(LPNN, LPPP)
LINK
LINK OF IST ELEMENT IN LPNN → ℒ(LPPP)
CALL FROM(LPNN, NN)
FROM
ITEM FROM IST ELEMENT IN LPNN → ℒ(NN)
FROM
ITEM FROM IST ELEMENT IN LPP → ℒ(LPNN)
CALL FROM(LFP, LPNN)
33
IS LIST ICNT NULL?
yes
10
PG. 1
C
NO
IS (ITEM) = 0 ?
yes
NO
50
RETURN
CALL FREL(LPPP, ITEM)
FREL
DATA IN IST ELEMENT OF LPPPP → ITEM
CALL LINK(LPNP, LPPP)
LINK
LINK OF IST ELEMENT IN LPNP → ℒ(LPPPP)
CALL FROM(LPNP, NP)
FROM
ITEM FROM IST ELEMENT IN LPNP → ℒ(NP)
CALL FROM(LPP, LPNP)
FROM
ITEM FROM IST ELEMENT IN LPPP → ℒ(LPNP)
CALL DOWN(ICNT, ICNT2)
DOWN
PUSH DOWN NEW ELEMENT → ICNT WITH ℒ(ICNT2) AS ITEM
LINK
LINK OF IST ELEMENT IN LPPP → ℒ(ICNT2)
CALL LINK(LPPP, ICNT2)

Program Description

1. Identification

   a. Routine Label

      FLAG

   b. Name

      Insert a FLAG into an element.

2. Function

This subroutine inserts a flag into the tag portion of the first element of the first sublist pointed to by a given head cell.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL FLAG (X, IFLG)

   b. Entry Conditions

      X = Head of a list
      IFLG = Flag to be inserted into the element of a sublist

   c. Exit Conditions

The first element of the first sublist pointed to by X will have IFLG as its tag.

   d. Error Exits

      None.

5. Definition of Identifiers

Z	=	A temporary head cell
ITEM	=	A temporary cell used for storage of the item in the first element of a sublist

6. Method

a. The item portion of the first element in list X is saved in Z. This item is a pointer to some sublist.

b. The item portion of the first element in Z is saved in ITEM.

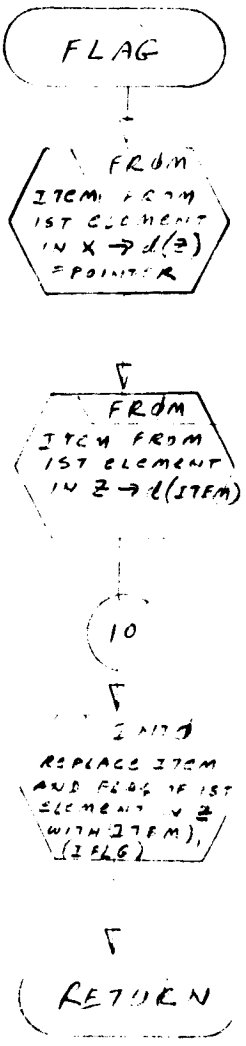
c. ITEM is returned to the first element of Z along with IFLG as its tag.

7. Other Subroutines Used

FROM, INTO.

8. Using Subroutines

STRIK, TREE, XFORM.



CALL FLAG (X, IFLG)

CALL FRDM (X, Z)

CALL FRDM (Z, ITEM)

CALL INTD (Z, ITEM, IFLG)

Program Description

1. Identification

   a. Routine Label

      FLTCON

   b. Name

      Floating-point conversion

2. Function

   To convert a floating-point number to BCD

3. Program System

   FORTRAN

4. Usage

   a. Calling Sequence

      CALL FLTCON (X, LST)

   b. Entry Conditions

      X contains floating-point number.
      LST is a list.

   c. Exit conditions

      X is converted to BCD and the characters pushed into LST; an asterisk is pushed into LST after X.

5. Definition of Identifiers

MULTS	BCD "*"
FLOG	Used to compute the exponent
ICH	Used to compute the exponent
FICH	Used to compute the exponent
FRAC	Fractional portion of X
SIGN	Used to hold the sign portion
EXP	BCD "E"

6. Method

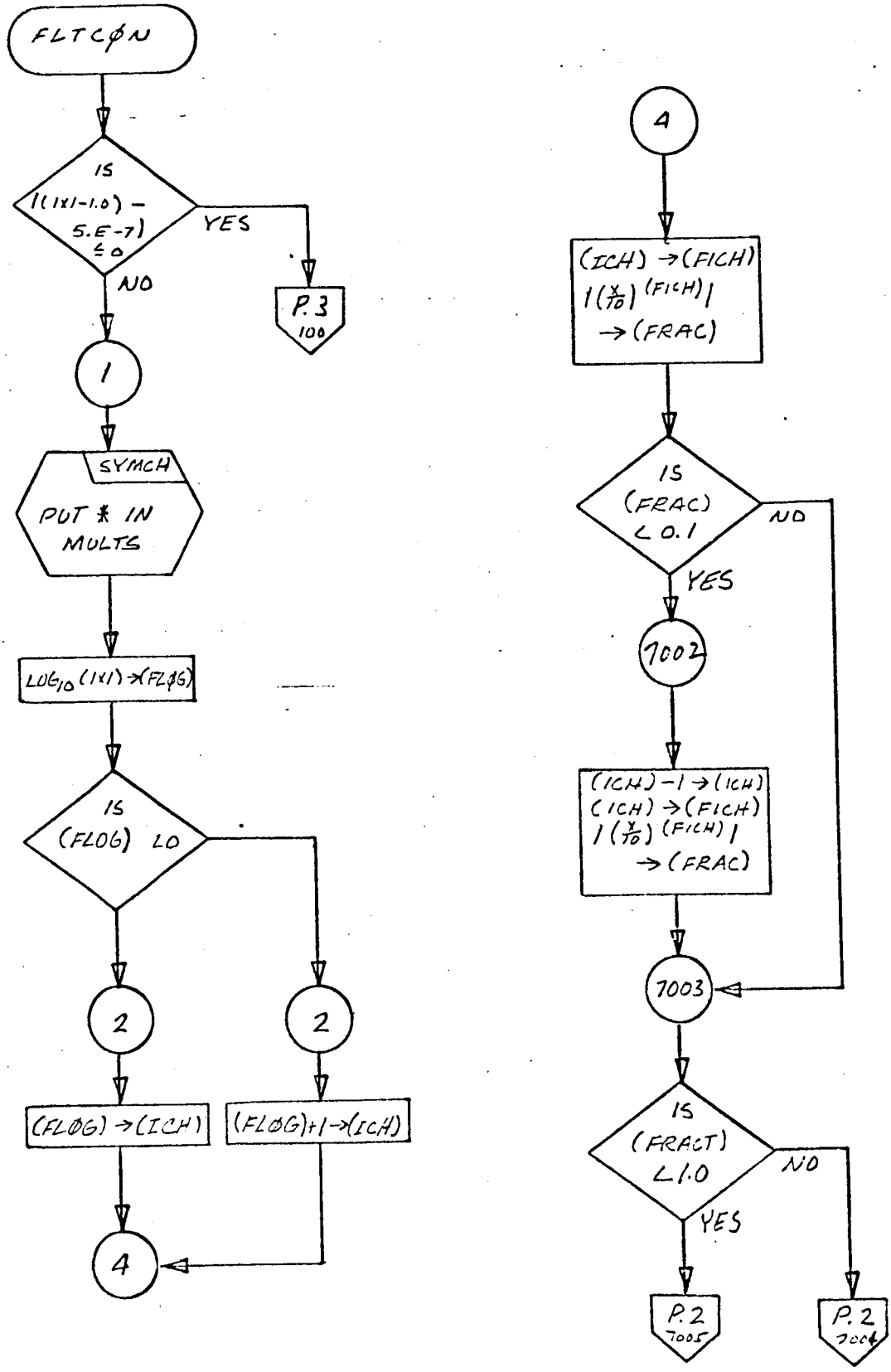
The exponent of X is placed in ICH.
The fractional part is placed in FRAC.
The algebraic signs of ICH and FRAC are determined.
The sign (FRAC) is pushed into LST.
Decimal point is pushed into LST.
(FRAC) is converted to BCD and pushed into LST.
"E" is pushed into LST.
Sign (ICH) is pushed into LST.
ICH is converted to BCD and pushed into LST.
An " * " is pushed into LST.

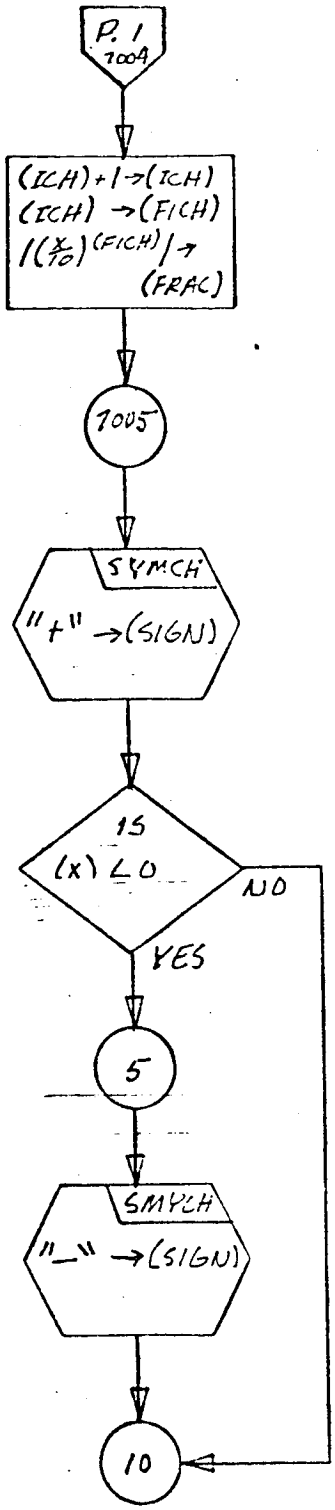
7. Other Subroutines Used

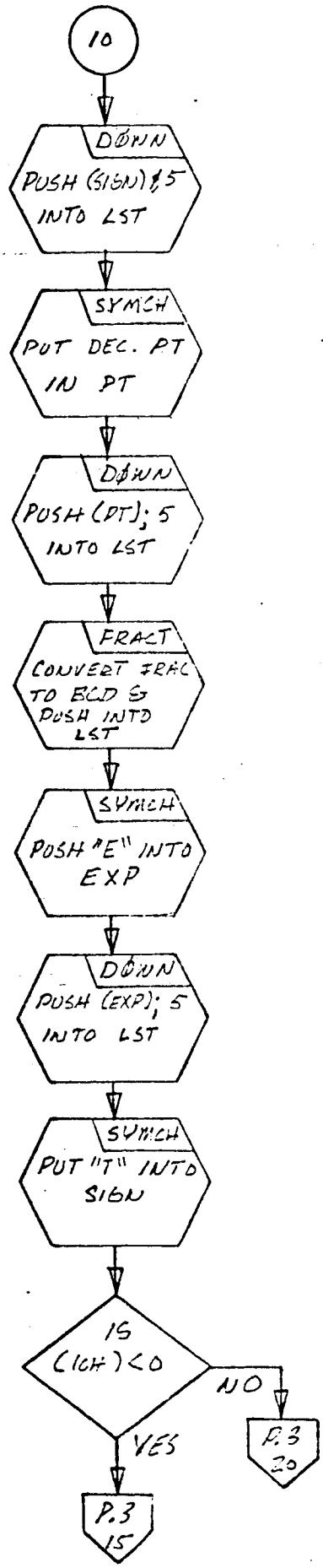
DONBD, DOWN, FRACT, SYMCH.

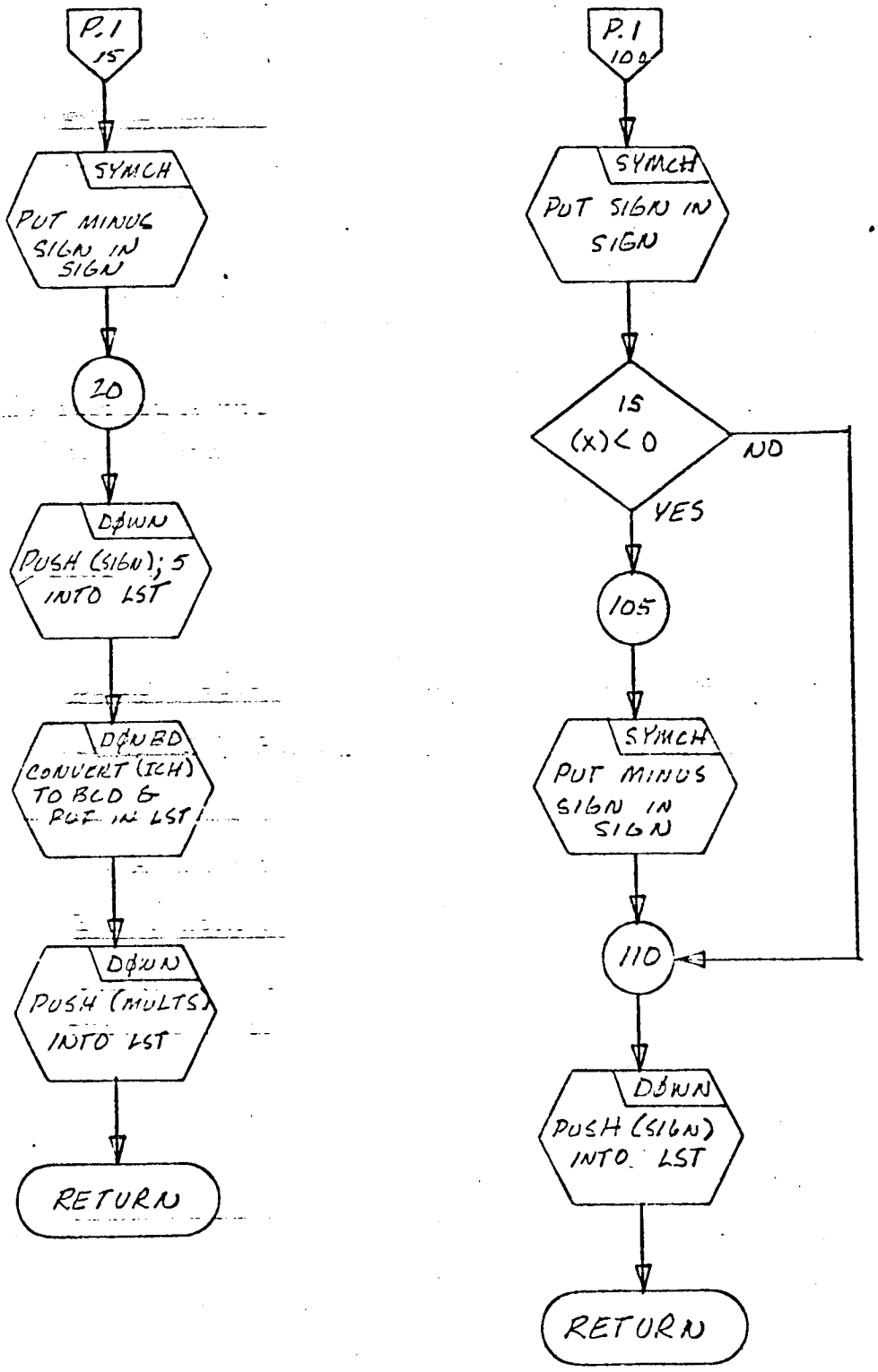
8. Using Subroutines

COMBN, SYMCRD.


FLTCØN
IS
|(|X|-1.0) -
5.E-7|
≤ 0
YES
NO
P.3
100
1
SYMCH
PUT * IN
MULTS
LOG10 (|X|) → (FLØG)
IS
(FLOG) LO
2
2
(FLØG) → (ICH)
(FLØG)+1 → (ICH)
4
4
(ICH) → (FICH)
|(X/10) (FICH)|
→ (FRAC)
IS
(FRAC)
< 0.1
NO
YES
7002
(ICH) - 1 → (ICH)
(ICH) → (FICH)
|(X/10) (FICH)|
→ (FRAC)
7003
IS
(FRACT)
< 1.0
NO
YES
P.2
7005
P.2
7004


P. 1
7004
(ICH)+1→(ICH)
(ICH) →(FICH)
1(X/10)(FICH)1→
(FRAC)
7005
SYMCH
"+" →(SIGN)
IS
(X) < 0
NO
YES
5
SMYCH
"_" →(SIGN)
10


10
DOWN
PUSH (SIGN); 5
INTO LST
SYMCH
PUT DEC. PT
IN PT
DOWN
PUSH (DT); 5
INTO LST
FRACT
CONVERT FRAC
TO BCD &
PUSH INTO
LST
SYMCH
PUSH "E" INTO
EXP
DOWN
PUSH (EXP); 5
INTO LST
SYMCH
PUT "T" INTO
SIGN
IS
(ICH) < 0
NO
YES
P.3
20
P.3
15


P.I
15
SYMCH
PUT MINUS SIGN IN SIGN
20
DOWN
PUSH (SIGN); 5 INTO LST
DOUBD
CONVERT (ICH) TO BCD & PUT IN LST
DOWN
PUSH (MULTS) INTO LST
RETURN
P.I
100
SYMCH
PUT SIGN IN SIGN
IS (X) < 0
NO
YES
105
SYMCH
PUT MINUS SIGN IN SIGN
110
DOWN
PUSH (SIGN) INTO LST
RETURN

Program Description

1. Identification

   a. Routine Label

      FOUTPT

   b. Name

      Write one record on FORTRAN Monitor tapes.

2. Function

   This subroutine writes 13 BCD words of a buffer, as one record, onto a print tape, a punch tape, and a save tape.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL FOUTPT (A)

   b. Entry Conditions

      A = A buffer that holds 13 BCD words.

   c. Exit Conditions

      Contents of buffer A are written on a FORTRAN Monitor print tape, punch tape, and save tape.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

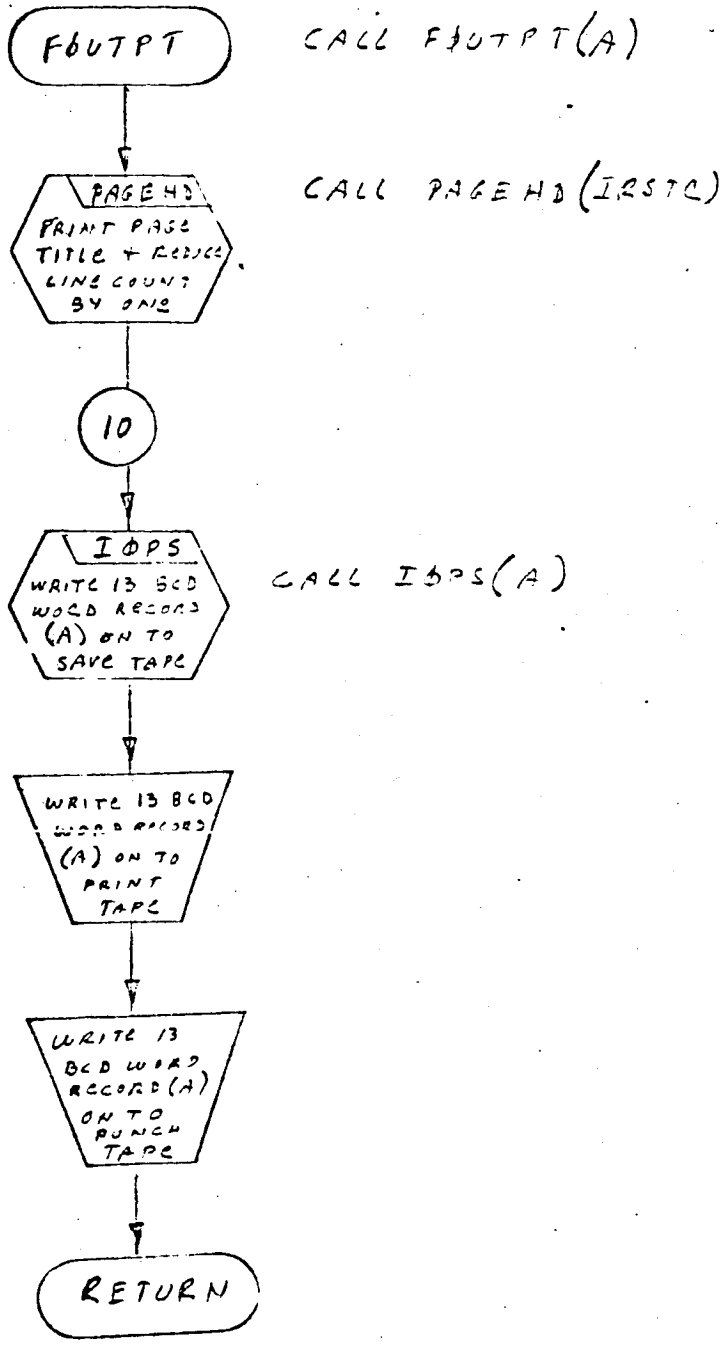
   The contents of buffer A are written as one record onto a save tape, a print tape, and a punch tape. The subroutine PAGEHD is used to control the page eject and title printing for each new page.

7. Other Subroutines Used

IOPS, PAGEHD.

8. Using Subroutines

DBPCHC, DBPCH, DBPFH, INPUTX.


FOUTPT
CALL FOUTPT(A)
PAGEHD
PRINT PAGE TITLE + REDUCE LINE COUNT BY ONE
CALL PAGEHD(IRSTC)
10
IOPS
WRITE 13 BCD WORD RECORD (A) ON TO SAVE TAPE
CALL IOPS(A)
WRITE 13 BCD WORD RECORD (A) ON TO PRINT TAPE
WRITE 13 BCD WORD RECORD (A) ON TO PUNCH TAPE
RETURN

Program Description

1. Identification

a. Routine Label

FRACT

b. Name

Convert a floating-point number to BCD and push the BCD characters down into a list.

2. Function

The floating-point number A is converted to decimal and stored as a sequence of BCD characters in list L. Storage is in reverse order, with the least significant digit at the top of the list.

3. Programming System

FAP

4. Usage

a. Calling Sequence

CALL FRACT (L, A)

b. Entry Conditions

A = Floating point-number

c. Exit Conditions

List L will contain a sequence of BCD characters, in reverse order, of the converted number in A. Each element in L will have a flag of 5.

d. Error Exits

None.

5. Definition of Identifiers

None.

6. Method

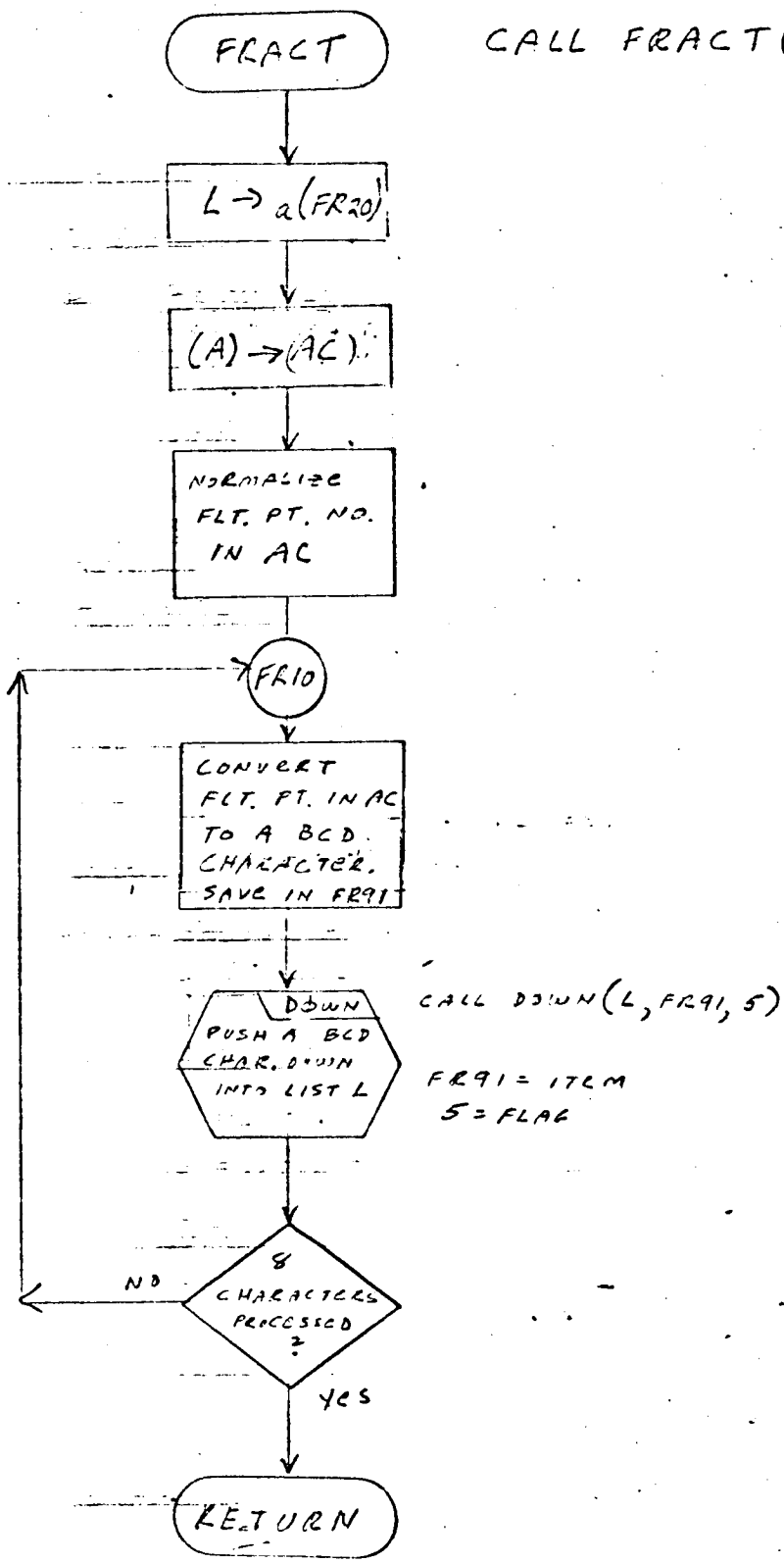
The floating-point number in A is normalized and saved in the accumulator. The number in the accumulator is converted to 8 BCD characters and pushed down into list L, each element containing one BCD character as its item and a flag of 5.

7. Other Subroutines Used

DOWN

8. Using Subroutines

FLTCON


FRACT
CALL FRACT(L,A)
L → a(FR20)
(A) → (AC)
NORMALIZE FLT. PT. NO. IN AC
FR10
CONVERT FLT. PT. IN AC TO A BCD. CHARACTER. SAVE IN FR91
DOWN
PUSH A BCD CHAR. DOWN INTO LIST L
CALL DOWN(L, FR91, 5)
FR91 = ITEM
5 = FLAG
8 CHARACTERS PROCESSED ?
NO
YES
RETURN

Program Description

1. Identification

   a. Routine Label

      FRFL

   b. Name

      Replace the contents of a word.

2. Function

   This subroutine replaces the contents of a given word with the contents of the first element of a list.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL FRFL (I, A)

   b. Entry Conditions

      I = Head of a list
      A = A full word of information

   c. Exit Conditions

      The first element of list I replaces the contents of A.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

   1. d(I) → (LOC2)
   2. ((LOC2)) → (A)

7. Other Subroutines Used

None.

8. Using Subroutines

FISH, LTRACE, MATOT, SNATCH, SYMCRD.

Program Description

1. Identification

   a. Routine Label

      FROM

   b. Name

      Extract item and flag FROM an element.

2. Function

This subroutine extracts the item portion and flag portion, if any, from the first element of a list.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL FROM (Z, C, IF)

   b. Entry Conditions

      Z = Head of a list

   c. Exit Conditions

      d(C) = Item of the first element in Z
      d(IF) = Flag, if any, of the first element in Z

   d. Error Exits

      If list Z is null, CALL DUMP is executed and return is made to the FORTRAN monitor system.

5. Definition of Identifiers

   None.

6. Method

The item portion and flag portion, if any, of the first element in list Z is extracted from the element and placed in the decrement of C

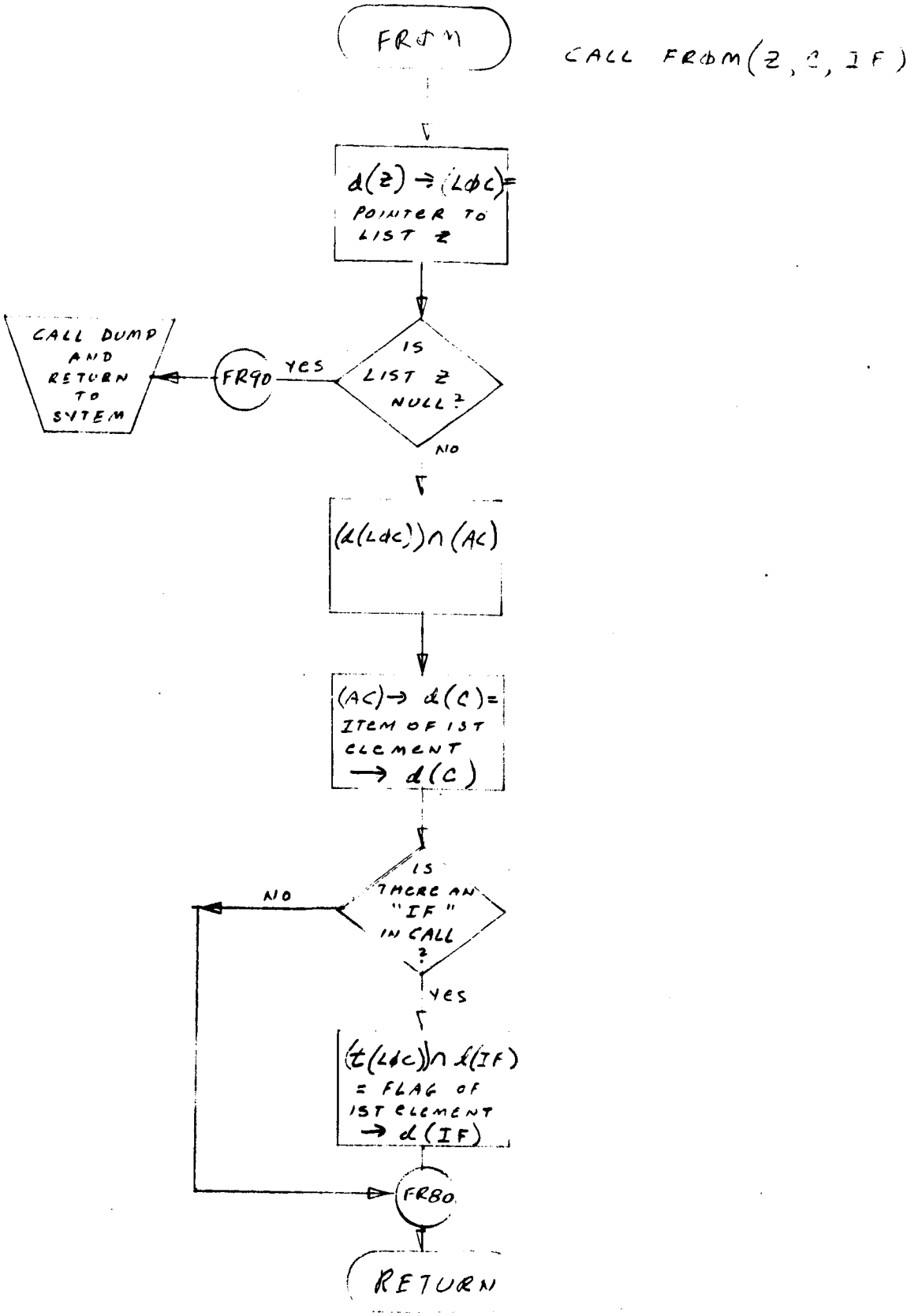
and the decrement of IF, respectively. If Z was null upon entry to this routine, CALL DUMP is executed and return is made to the FORTRAN Monitor System.

7. Other Subroutines Used

None.

8. Using Subroutines

COPY, DOWNS, ELIM, ERASE, FISH, FLAG, IDNTP, LEVMRK, LOCAT, LOCATA, MATFT, MATOT, MRKLST, INVST, POPUP, PRPTR, READCH, RECOVR, SNATCH, STASH, STRIK, SYMCRD, SYMTP, WRTEQ, Main Programs for Pass 2 of TAG Preprocessor.


FROM
CALL FROM(Z, C, IF)
d(Z) → (LOC)= POINTER TO LIST Z
IS LIST Z NULL ?
YES
FR9D
CALL DUMP AND RETURN TO SYTEM
NO
(d(LOC)) ∩ (AC)
(AC) → d(C)= ITEM OF IST ELEMENT → d(C)
IS THERE AN "IF" IN CALL ?
NO
YES
(t(LOC)) ∩ f(IF) = FLAG OF IST ELEMENT → d(IF)
FR8O
RETURN

Program Description

1. Identification

   a. Routine Label

      GOBLE

   b. Name

      Read TAG connection lists from tape and from WLIST.

2. Function

This subroutine reads the TAG connection list, removes punctuation and excess characters, converts BCD characters to binary, and stores a descriptor at a time in WLIST. NM is set to the maximum node number and NT will hold the total number of branch descriptors.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL GOBLE (WLIST, NM, NT)

   b. Entry Conditions

      The TAG connection list will be on a FORTRAN input tape in BCD.

   c. Exit Conditions

      WLIST is formed as a two-dimensional list representation of the TAG connection list. Each branch descriptor is represented by a sublist with 3 or 5 elements. Branch ordering is not changed. All components of the branch description are represented by binary integers. The element type characters are replaced by the following integers:

      V = 1
      C = 2
      G = 3

L = 4

N = 5

I = 6

Node numbers, transformer turns, and transformer numbers are converted directly to binary. NM will contain the maximum node number in the connection list. NT will contain the total number of branch descriptors.

d. Error Exits

None.

5. Definition of Identifiers

P = A temporary list used in processing the branch descriptors. P contains an inverted copy of the TAG branch connection list. All blanks are removed and the terminating * is not included. P is a one-dimensional list with one character stored in each element.

A = A local variable, dimensioned at 14. Used for input, output of BCD characters.

I = Index for BCD words being processed.

IEXM(I) = A local variable dimensioned at 6. Contains the 6 allowed terminating characters for a branch descriptor.

ITEM = Contains the current character being processed.

ERROR1, ERROR2 = Temporary lists that hold the characters for error comments to be printed.

6. Method

The TAG connection list is read in to form WLIST as follows:

a. Clear KTRN, KIND, NM, NT to zero.

b. ERASE P list and WLIST.

c. Store six characters blank *, S / - in IEXM(I), I = 1, 6.

d. Read 13 BCD words (branch descriptors) from tape and store in A(I), I = 1, 13.

e. Write 13 BCD words from A(I), I 1, 13 onto a print tape.

f. The Ith character of A is placed in ITEM and compared with IEXM(2) to determine if the character is an asterisk (*).

(1) If ITEM contains an *, discontinue reading the input tape and proceed to step i to continue.

(2) If the character in ITEM is not an *, proceed to the next step.

g. The Ith character of A in ITEM is compared with IEXM(1) to determine if the character is a blank.

(1) If ITEM contains a blank, ITEM is not pushed down into P list. Go to step h to continue.

(2) If ITEM does not contain a blank, a new element is pushed down into P list with the d(ITEM) as its item.

h. I is tested:

(1) If I $\neq$ 72, store I + 1 in I. Return to step f and continue.

(2) If I = 72, return to step d and continue processing.

i. An asterisk has been read (step f(1)):

(1) Clear NTRN, NNTR to zero. Initialize flag word KSW to 1.

(2) Segment list P at the first comma encountered (after the first descriptor in list P). Form a new list Q containing the segmented portion of P (first descriptor followed by a comma). P will contain the rest of the descriptors. Set flag word K.

(3) ERASE list ERROR1.

(4) Popup list Q and push elements down into ERROR1. List Q is not destroyed.

(5) K is tested:

(a) If K = 3, list P was null. Exit from the routine.

(b) If K = 1 or 2, segment list Q at the first / or - encountered in the descriptor. Form new list R of segmented portion of Q. Set K.

(i) For K = 1 or 4, a / was found or Q was null. Go to step j to print error comment.

(ii) If K = 2, a - was found. Proceed to step i(12) and continue.

(iii) If K = 3, a / or - was not found within the descriptor. List Q is now null and list R contains the elements formerly in Q. Continue at next step.

(6) The first two elements in list R are popped up and their characters converted from BCD to binary, then saved in NP. K is set.

(a) If (K-2) $\neq$ 0, either R was null or one of the digits popped up from R was non-numeric. Go to step j to print error comment.

(b) If (K-2) = 0, the digits in the two elements (popped up from R) were converted.

(7) Popup the next two elements in R, convert them to binary, then save the converted number in NN. Set K.

(a) If (K-2) $\neq$ 0, go to step j to print error comment (same conditions as in (6)(a) above).

(b) If (K-2) = 0, the digits popped up from R were converted.

(8) Match the item of the next element in R with a number of the array, VCGLNI, to determine the type of branch descriptor being processed. The number corresponding to the position of the matched character in the above array is saved in NE.

(a) Popup the next element from R (whose item was just matched) and store in the decrement of ITEM. Test NE.

(i) If NE = 1, 2, 3, or 6, go to step i(10) to continue.

(ii) If NE = 4, go to step i(9) to continue.

(iii) If NE M 5, go to step i(11) to continue.

(9) NE = 4 (branch descriptor of type L) set KIND to 2.

(10) NE = 1, 2, 3, 4 or 6 (branch descriptor of type V, C, G, L, or I)

(a) Test KSW.

(i) If KSW = 2, go to step j to print error comment.

(ii) If KSW = 1, continue.

(b) Test if R is null.

(i) If R is null, go to step j to print error comment.

(ii) If R is not null, continue.

(c) Popup the next element in R. Save its item in d(ITEM).

(i) If the character in ITEM is not an S, go to step j to print error comment.

(ii) If ITEM does contain an S, continue.

(d) Test if R is null.

(i) If R is null, go to step i(10)(f) to continue.

(ii) If R is not null, proceed at next step.

(e) Popup the next element in R. Save its item in d(ITEM).

(i) If ITEM does not contain comma (,), go to step j to print error comment.

(ii) If the character in ITEM is a (,), continue at the next step.

(f) ERASE list R.

(g) Push down a sublist into WLIST containing elements with item NE, NN, NP, NTRN, NNTR for the descriptor just processed.

(h) NN is compared with NM.

(i) If NN > NM, NN replaces NM as the maximum node number. Proceed at step (i) below.

(ii) If NN < NM, NM is unchanged.

(i) The NP of the descriptor is compared with NM.

(i) If NP > NM, NP replaces NM as the maximum node number.

(ii) If NP < NM, NM is unchanged.

(j) NT, which holds the number of branch descriptors, is increased by 1.

(k) Return to step i(1) to continue processing of the next descriptor in list P.

(11) NE = 5 (transformer branch descriptor SN)

(a) Test KSW.

(i) If KSW = 1, go to step j to print error comment.

(ii) If KSW = 2, continue at step i(10)(b) above.

(12) A - was found within the descriptor. (transformer)

(a) Set KSW to 2.

(b) All characters in R up to 10 are connected from BCD to binary and saved in NNTR. K is set to number of actual characters converted.

(c) Set KTRN to 2.

(d) Segment list Q at the first / encountered within the descriptor. Form new list R from segmented portion of Q (up to and including /). Set K.

(e) Test K.

(i) If K = 2 or 3, a / could not be found or Q was null. Go to step j to print error comment.

(ii) If K = 1, continue at next step.

(f) All characters in list R up to 10 are converted from BCD to binary and saved in NTRN. K is set to number of characters actually converted.

(g) Test R.

(i) If R is null, go to step j to print error comment.

(ii) If R is not null, continue.

(h) Pop up the next element in R. Save its item in d(ITEM).

(i) Test R

(i) If R is not null, go to step j to print erase comment.

(ii) If R is null, move the pointer in head cell Q to head cell R. List R is now pointing to the remaining elements which were in Q. Q is made null. Return to step i(6) to continue processing.

j. An error has been detected during the descriptor processing.

(1) Place the characters "/ IS IN ERROR" into list ERROR2.

(2) Insert a blank into the first position of A.

(3) Set I to 2.

(4) Test list ERROR1.

(a) If ERROR1 is null, continue at step j(8).

(b) If ERROR1 is not null, proceed to next step.

(5) Popup the next element in ERROR1. Save its item in d(ITEM).

(6) Place character in ITEM into the Ith position of A.

(7) Test I

(a) If I $\neq$ 72, increase I by 1 and return to step j(4) to continue processing for error comment.

(b) If I = 72, print error comment from A(I), I = 1, 13, CALL EXIT and return to FORTRAN monitor system.

(8) ERROR1 was null.

(a) Test list ERROR2

(i) If ERROR2 is null, store a blank character in ITEM and go to step j(6) to continue.

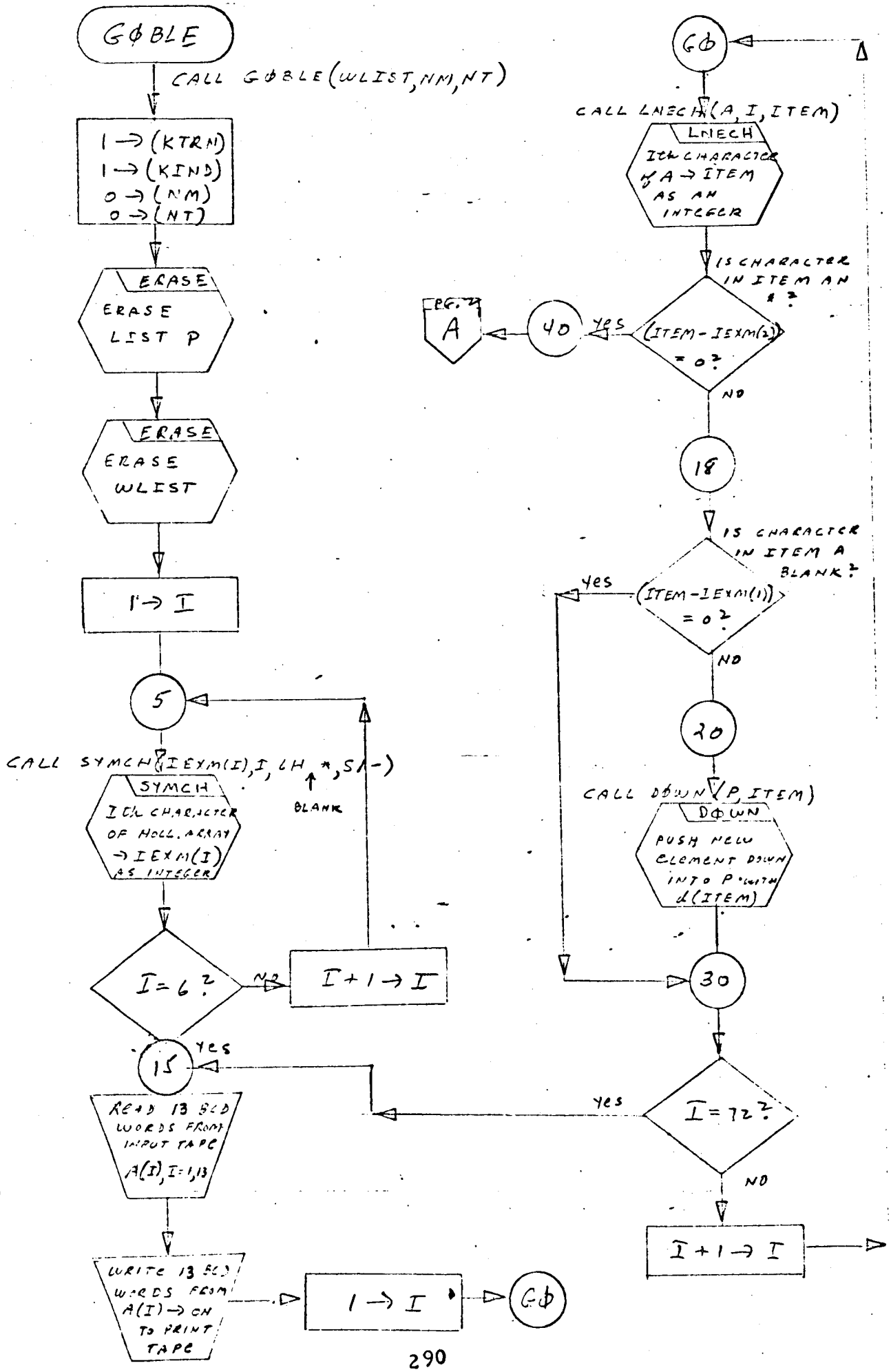
(ii) If ERROR2 is not null, popup the next element from ERROR2 and save its item in d(ITEM). Go to step j(6) to continue.

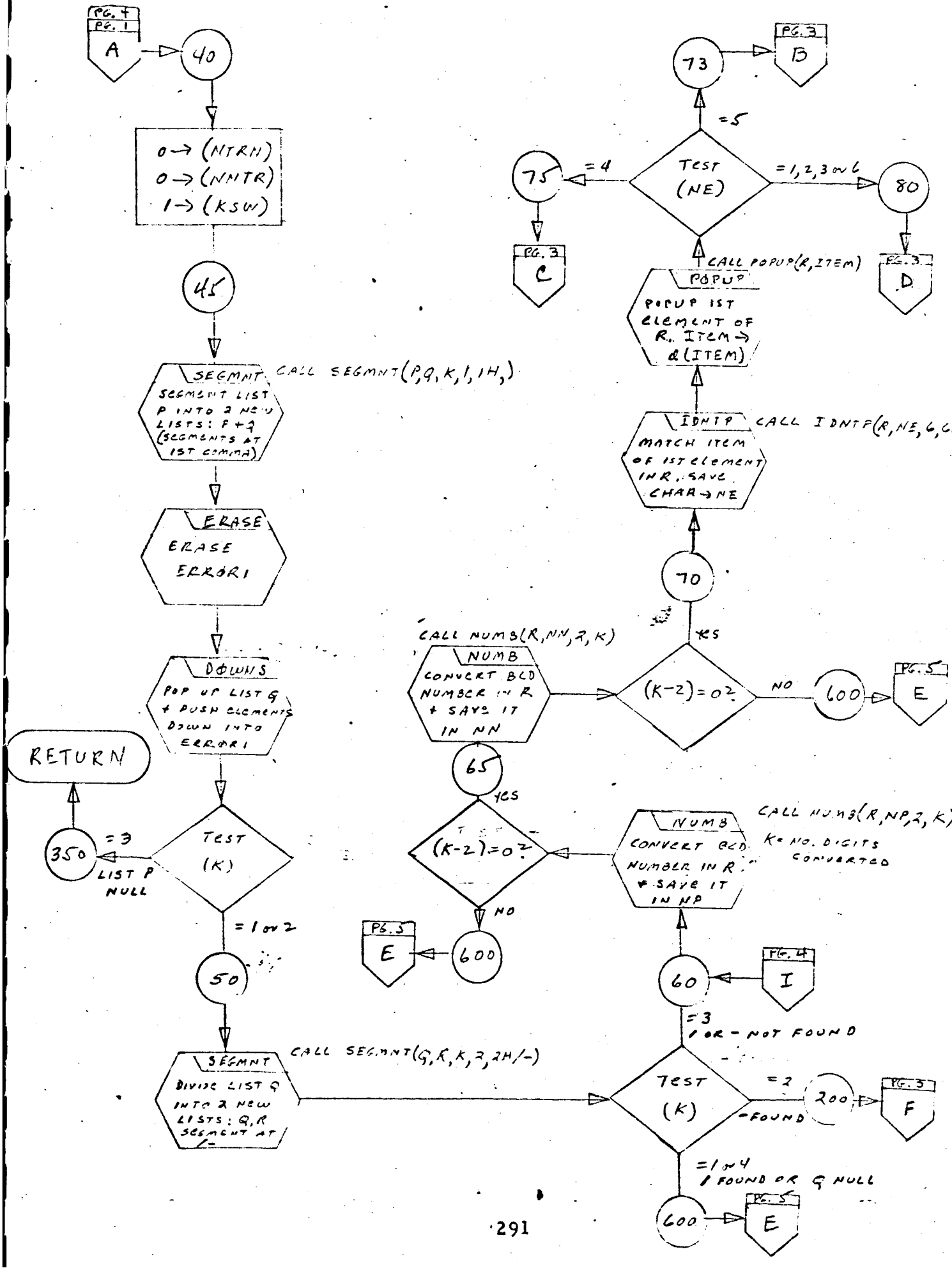
7. Other Subroutines Used

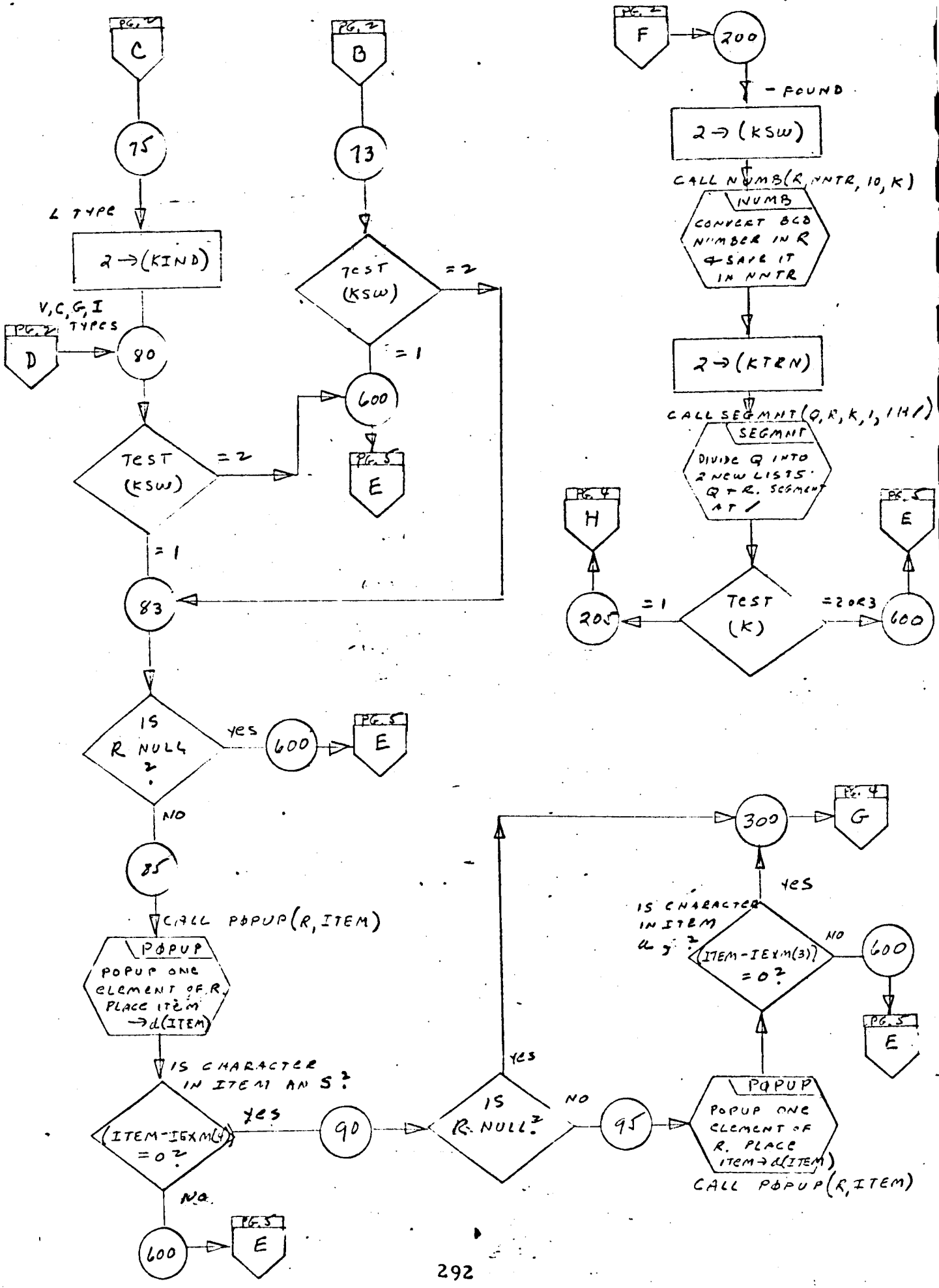
CHLNE, DOWN, DOWNS, ERASE, IDNTP, LNECH, NUMB, POPUP, PRPTG, SEGMNT, SYMBL, SYMCH, FORTRAN SYSTEM ROUTINE "EXIT."

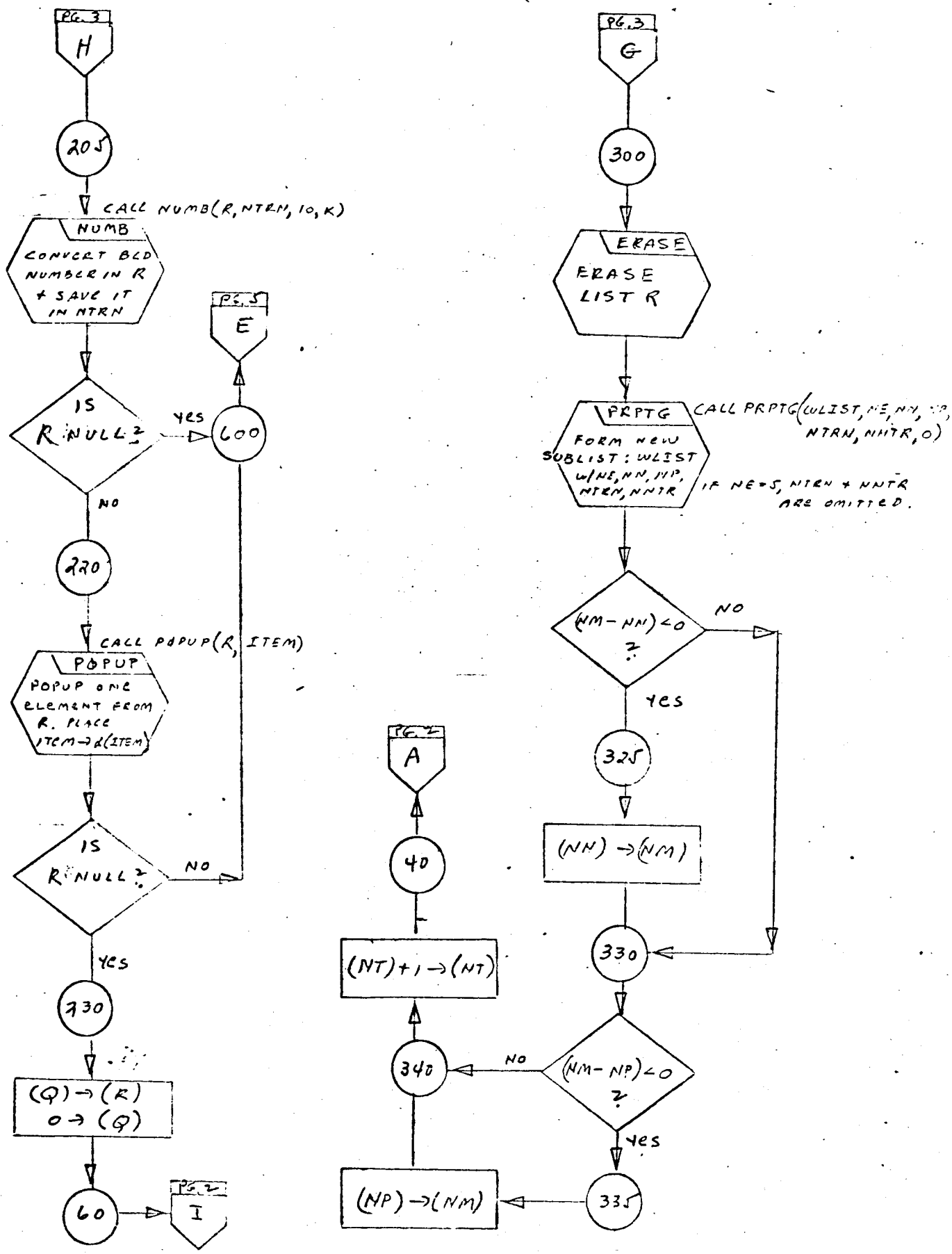
8. Using Subroutines

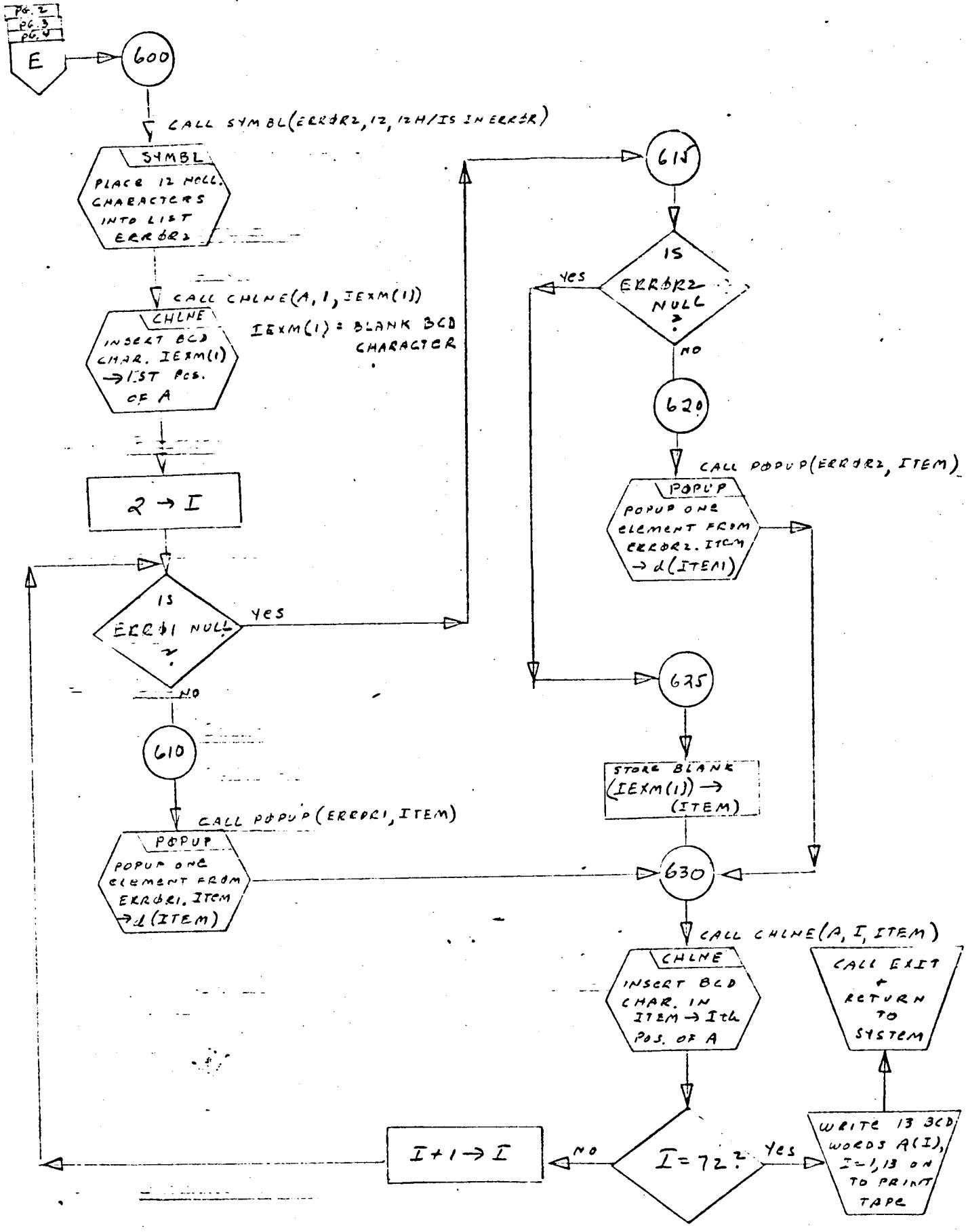
Main program for PASS 1 of TAG preprocessor.


GOBLE
CALL GOBLE(WLIST,NM,NT)
1 → (KTRM)
1 → (KIND)
0 → (NM)
0 → (NT)
ERASE
ERASE LIST P
ERASE
ERASE WLIST
1 → I
5
CALL SYMCH(IEXM(I),I,6H *,S1-)
SYMCH
Ith CHARACTER OF HOLL. ARRAY → IEXM(I) AS INTEGER
BLANK
I = 6 ?
NO
I + 1 → I
yes
15
READ 13 BCD WORDS FROM INPUT TAPE A(I), I=1,13
WRITE 13 BCD WORDS FROM A(I) → ON TO PRINT TAPE
1 → I
GO
GO
CALL LNECH(A, I, ITEM)
LNECH
Ith CHARACTER OF A → ITEM AS AN INTEGER
IS CHARACTER IN ITEM AN #?
(ITEM - IEXM(2)) = 0 ?
yes
40
PG. 7
A
NO
18
IS CHARACTER IN ITEM A BLANK?
yes
(ITEM - IEXM(1)) = 0 ?
NO
20
CALL DOWN(P, ITEM)
DOWN
PUSH NEW ELEMENT DOWN INTO P WITH d(ITEM)
30
I = 72 ?
yes
NO
I + 1 → I


PG. 4 PG. 1
A
40
0 → (NTRN)
0 → (NNTR)
1 → (KSW)
45
SEGMNT
SEGMENT LIST P INTO 2 NEW LISTS: P + Q (SEGMENTS AT 1ST COMMA)
CALL SEGMNT(P,Q,K,1,1H,)
ERASE
ERASE ERROR1
DOWNS
POP UP LIST Q & PUSH ELEMENTS DOWN INTO ERROR1
RETURN
TEST (K)
=3
350
LIST P NULL
=1 or 2
50
SEGMNT
DIVIDE LIST Q INTO 2 NEW LISTS: Q,R SEGMENT AT /-
CALL SEGMNT(Q,R,K,2,2H/-)
NUMB
CONVERT BCD NUMBER IN R & SAVE IT IN NN
CALL NUMB(R,NN,2,K)
65
YES
(K-2)=0?
NO
PG. 5
E
600
NUMB
CONVERT BCD NUMBER IN R & SAVE IT IN NP
CALL NUMB(R,NP,2,K)
K = NO. DIGITS CONVERTED
60
PG. 4
I
=3
1 OR - NOT FOUND
TEST (K)
=2
-FOUND
200
PG. 3
F
=1 or 4
1 FOUND OR Q NULL
600
PG. 5
E
70
YES
(K-2)=0?
NO
600
PG. 5
E
IDNTF
MATCH ITEM OF 1ST ELEMENT IN R. SAVE CHAR → NE
CALL IDNTF(R,NE,6,CHVCGLN)
POPUP
POPUP 1ST ELEMENT OF R. ITEM → Q(ITEM)
CALL POPUP(R,ITEM)
TEST (NE)
=4
75
PG. 3
C
=5
73
PG. 3
B
=1,2,3 or 6
80
PG. 3
D


PG. 4
C
75
L TYPE
2 → (KIND)
V, C, G, I TYPES
PG. 2
D
80
TEST (KSW)
= 2
= 1
83
IS R NULL?
yes
600
PG. 5
E
NO
85
CALL POPUP(R, ITEM)
POPUP
POPUP ONE ELEMENT OF R, PLACE ITEM → d(ITEM)
IS CHARACTER IN ITEM AN S?
(ITEM-IEXM(4)) = 0?
yes
90
No
600
PG. 5
E
PG. 2
B
13
TEST (KSW)
= 2
= 1
600
PG. 5
E
PG. 2
F
200
- FOUND
2 → (KSW)
CALL NUMB(R, NNTR, 10, K)
NUMB
CONVERT BCD NUMBER IN R & SAVE IT IN NNTR
2 → (KTRN)
CALL SEGMNT(Q, R, K, 1, IHF)
SEGMNT
DIVIDE Q INTO 2 NEW LISTS: Q + R. SEGMENT AT /
TEST (K)
= 1
205
PG. 4
H
= 2 OR 3
600
PG. 5
E
IS R NULL?
yes
NO
95
POPUP
POPUP ONE ELEMENT OF R. PLACE ITEM → d(ITEM)
CALL POPUP(R, ITEM)
300
PG. 4
G
yes
IS CHARACTER IN ITEM A S?
(ITEM-IEXM(3)) = 0?
NO
600
PG. 5
E


PG. 3
H
205
CALL NUMB(R, NTRN, 10, K)
NUMB
CONVERT BCD NUMBER IN R & SAVE IT IN NTRN
PG. 5
E
IS R NULL ?
YES
600
NO
220
CALL POPUP(R, ITEM)
POPUP
POPUP ONE ELEMENT FROM R. PLACE ITEM → R(ITEM)
IS R NULL ?
NO
YES
230
(Q) → (R)
0 → (Q)
60
PG. 2
I
PG. 3
G
300
ERASE
ERASE LIST R
PRPTG
CALL PRPTG(WLIST, NE, NN, NP, NTRN, NNTR, 0)
FORM NEW SUBLIST: WLIST W/NE, NN, NP, NTEN, NNTR
IF NE=5, NTRN & NNTR ARE OMITTED.
(NM - NN) < 0 ?
NO
YES
325
(NN) → (NM)
330
(NM - NP) < 0 ?
NO
340
YES
335
(NP) → (NM)
(NT) + 1 → (NT)
40
PG. 2
A


PG. 2
PG. 3
PG. 4
E
600
CALL SYMBL(ERROR2, 12, 12H/IS IN ERROR)
SYMBL
PLACE 12 HOLL. CHARACTERS INTO LIST ERROR2
CALL CHLNE(A, 1, IEXM(1))
CHLNE
INSERT BCD CHAR. IEXM(1) → 1ST POS. OF A
IEXM(1) = BLANK BCD CHARACTER
2 → I
IS ERROR1 NULL ?
yes
NO
610
CALL POPUP(ERROR1, ITEM)
POPUP
POPUP ONE ELEMENT FROM ERROR1. ITEM → d(ITEM)
615
IS ERROR2 NULL ?
yes
NO
620
CALL POPUP(ERROR2, ITEM)
POPUP
POPUP ONE ELEMENT FROM ERROR2. ITEM → d(ITEM)
625
STORE BLANK (IEXM(1)) → (ITEM)
630
CALL CHLNE(A, I, ITEM)
CHLNE
INSERT BCD CHAR. IN ITEM → Ith POS. OF A
I = 72 ?
yes
NO
I + 1 → I
WRITE 13 BCD WORDS A(I), I=1,13 ON TO PRINT TAPE
CALL EXIT + RETURN TO SYSTEM

Program Description

1. Identification

   a. Routine Label

      HEAD

   b. Name

      Convert statement number and continuation card number to BCD and store in an output buffer.

2. Function

   The integers in NS and NC are converted to BCD and stored in BUFF. NS goes into columns 1-5, right justified, and the low digit of NC goes into column 6.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL HEAD (BUFF, NS, NC)

   b. Entry Conditions

      NS = Statement number
      NC = Continuation card number

   c. Exit Conditions

      NS and NC are converted to BCD and stored in the output buffer BUFF, NS to column 1-5, right justified, and low digit of NC to column 6.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

a. The output buffer BUFF is cleared to blanks.

b. The statement number in NS is placed in the accumulator, right justified, then converted to BCD and stored in columns 1-5 of BUFF.

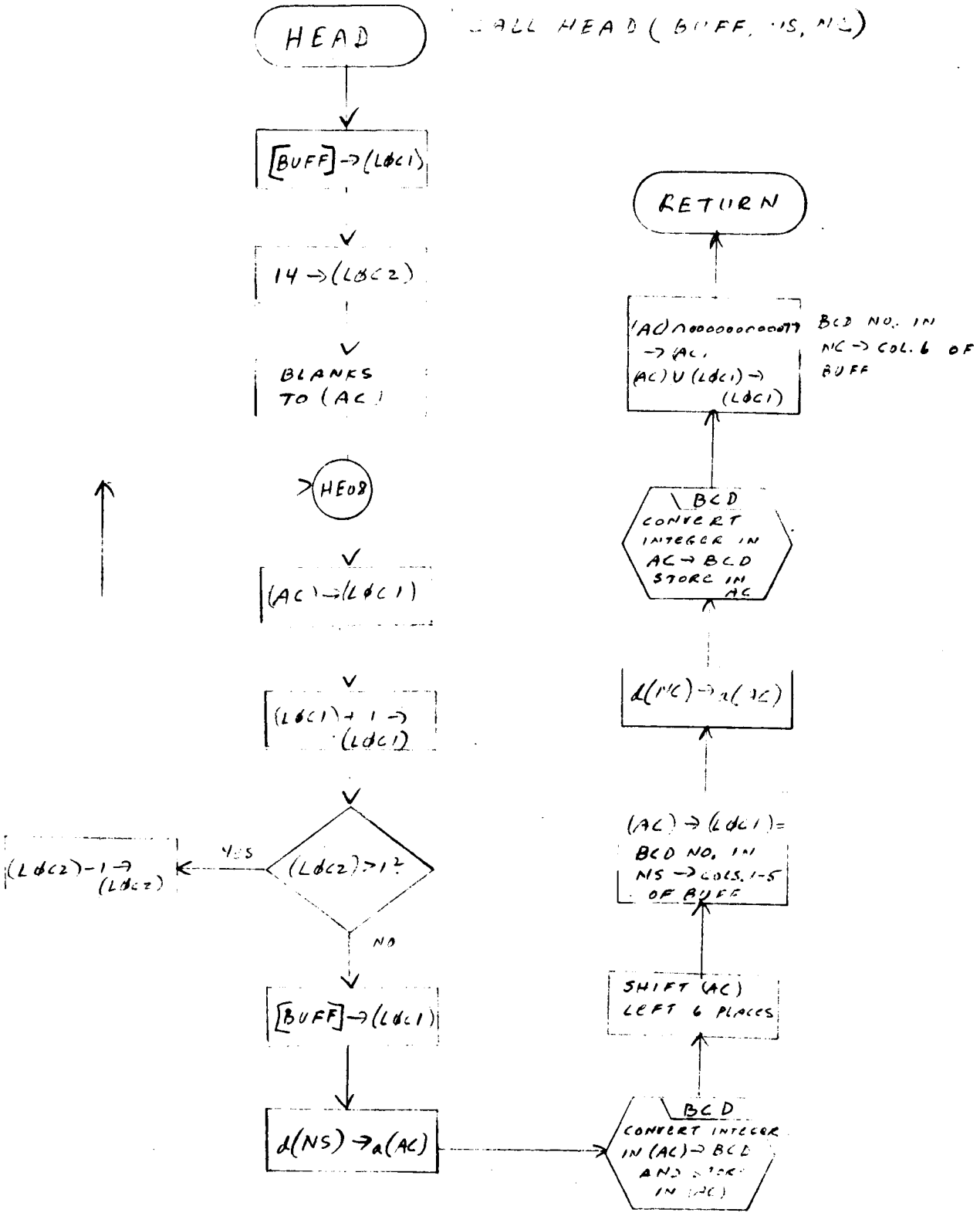
c. The continuation card number in NC is placed in the accumulator, right justified, then converted to BCD and stored in column 6 of BUFF.

7. Other Subroutines Used

BCD

8. Other Subroutines

DBPCH, DBPFH, INPUTX


HEAD
CALL HEAD (BUFF, NS, NC)
[BUFF] → (LØC1)
14 → (LØC2)
BLANKS TO (AC)
HEØ8
(AC) → (LØC1)
(LØC1) + 1 → (LØC1)
(LØC2) > 1?
YES
(LØC2) - 1 → (LØC2)
NO
[BUFF] → (LØC1)
d(NS) → a(AC)
BCD CONVERT INTEGER IN (AC) → BCD AND STORE IN (AC)
SHIFT (AC) LEFT 6 PLACES
(AC) → (LØC1) = BCD NO. IN NS → COLS. 1-5 OF BUFF
d(NC) → a(AC)
BCD CONVERT INTEGER IN AC → BCD STORE IN AC
(AC) ∧ 000000000077 → (AC), (AC) U (LØC1) → (LØC1)
BCD NO. IN NC → COL. 6 OF BUFF
RETURN

Program Description

1. Identification

   a. Routine Label

      HEADC

   b. Name

      Convert statement number, continuation card number, and special character (for column 1) to BLD and store in output buffer.

2. Function

   The integers in IS and IC are converted to BCD and stored in columns 2-5 and column 6, respectively, of output buffer A. The character in IC1 is stored in column 1 of buffer A.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL HEADC (A, IS, IC, IC1)

   b. Entry Conditions

      IS = Statement number
      IC = Continuation card number
      IC1 = Character for column 1

   c. Exit Conditions

      Output buffer A will contain IC1 in column 1, IS converted to BCD in columns 2-5, and IC converted to BCD in column 6.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

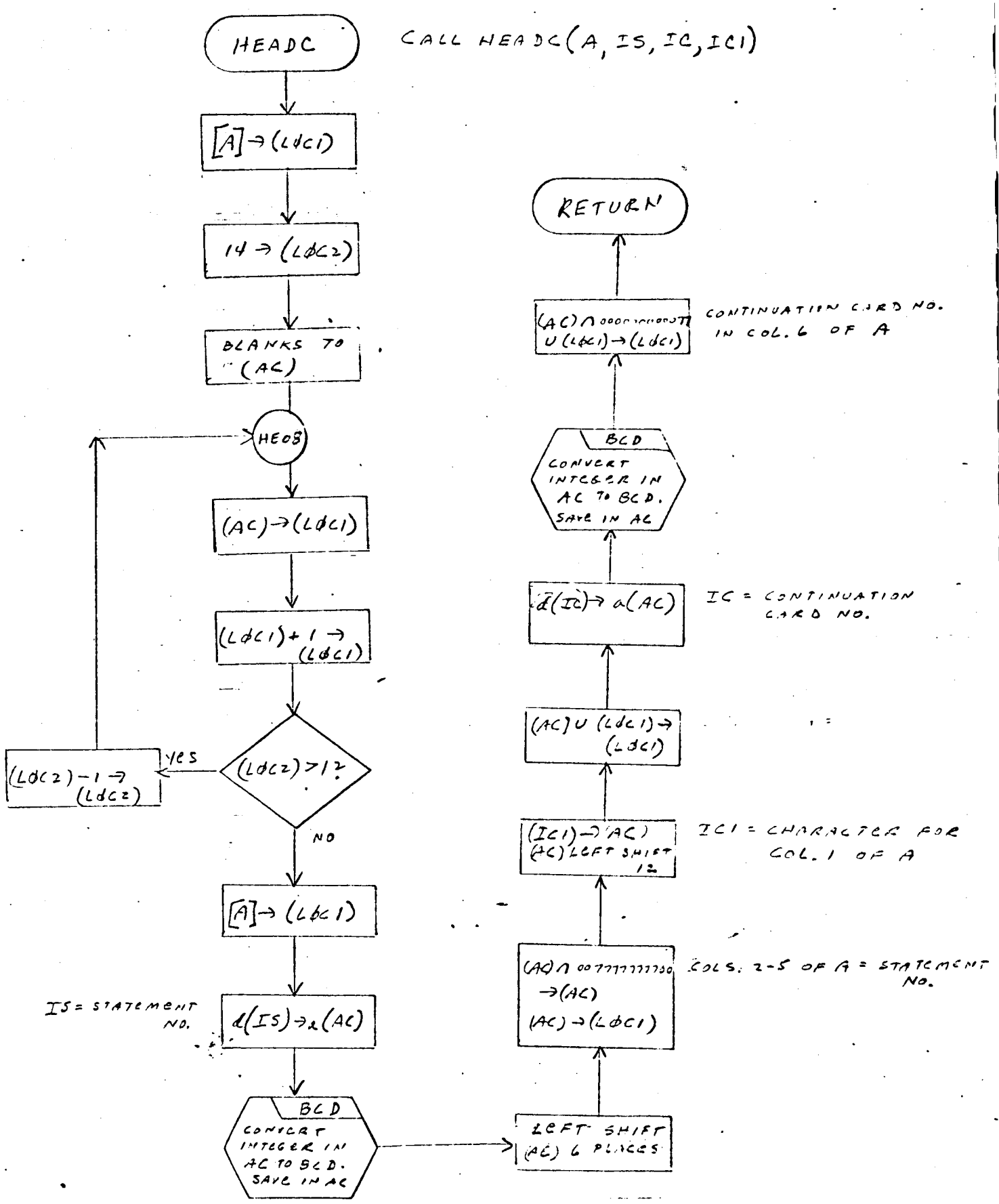
1. The output buffer A is cleared to blanks.
2. The statement number in IS is placed in the accumulator, right justified, and is then converted to BCD and stored in columns 2-5 of buffer A.
3. The character in IC1 is stored in column 6 of buffer A.
4. The continuation card number in IC is placed in the accumulator, converted to BCD, and stored in column 6 of buffer A.

7. Other Subroutines Used

BCD

8. Using Subroutines

DBPCHC, PUSPCH


HEADC
CALL HEADC(A, IS, IC, IC1)
[A] → (LØC1)
14 → (LØC2)
BLANKS TO (AC)
HEØ8
(AC) → (LØC1)
(LØC1) + 1 → (LØC1)
(LØC2) > 12
yes
(LØC2) - 1 → (LØC2)
NO
[A] → (LØC1)
IS = STATEMENT NO.
d(IS) → a(AC)
BCD
CONVERT INTEGER IN AC TO BCD. SAVE IN AC
LEFT SHIFT (AC) 6 PLACES
(AC) ∧ 000777777730 → (AC)
(AC) → (LØC1)
COLS. 2-5 OF A = STATEMENT NO.
(IC1) → (AC)
(AC) LEFT SHIFT 12
IC1 = CHARACTER FOR COL. 1 OF A
(AC) U (LØC1) → (LØC1)
d(IC) → a(AC)
IC = CONTINUATION CARD NO.
BCD
CONVERT INTEGER IN AC TO BCD. SAVE IN AC
(AC) ∧ 000000777777 U (LØC1) → (LØC1)
CONTINUATION CARD NO. IN COL. 6 OF A
RETURN

Program Description

1. Identification

   a. Routine Label

      HOLBK

   b. Name

      Remove blanks from a list of BCD characters.

2. Function

RIN is a list of BCD characters from which all blanks are removed except those which occur imbedded in Hollerith variables of the form--6H· · · ---

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL HOLBK (RIN)

   b. Entry Conditions

      RIN = A list of BCD characters

   c. Exit Conditions

List RIN will contain the original BCD characters with all blanks removed except those in a literal Hollerith string.

   d. Error Exits

      None.

5. Definition of Identifiers

ITEM = Contains a BCD character
RT = A list which contains the same elements as RIN, in reverse order

CNT = A list used to save the integer which precedes the character H in a literal Hollerith string

6. Method

The blanks are removed from list RIN as outlined below.

a. List RIN is popped up, one element at a time, and then pushed down into RT. RIN is destroyed.

b. RT is tested:
   (1) If RT is null, exit is made from the routine.
   (2) If RT is not null, the first element in RT is popped up and the character in its item is saved in ITEM.

c. ITEM is tested:
   (1) If the character in ITEM is a (or a , execution continues at step d.
   (2) If the character in ITEM is a blank, the blank is not pushed down into the new list RIN. Execution is repeated at step b.
   (3) If the character in ITEM is not a (or a , or a blank, it is pushed down into RIN, and then execution is repeated at step b.

d. When the character in ITEM is a (or a , the (or , is pushed down into RIN.

e. RT is tested:
   (1) If RT is null, exit is made from the routine.
   (2) If RT is not null, the next element in RT is popped up and saved in ITEM.

f. ITEM is tested:
   (1) If ITEM holds a blank, step e is repeated.
   (2) If ITEM holds an integer (1 to 9), it is pushed down into RIN and CNT, then step e is repeated.
   (3) If ITEM does not hold a blank or an integer, a test is made to determine if the character is an H or a comma.

(a) If the character is a comma, continue at step d.

(b) If the character is an H, it is pushed down into RIN as a new element. The integer saved in CNT is popped up to determine how many characters are in the literal Hollerith string following H. These characters, including any embedded blanks, are pushed down into RIN with a flag of 1 in each element.

(c) RT is tested:

(i) If RT is null, exit is made from the routine.

(ii) If RT is not null, go back to step b to continue.

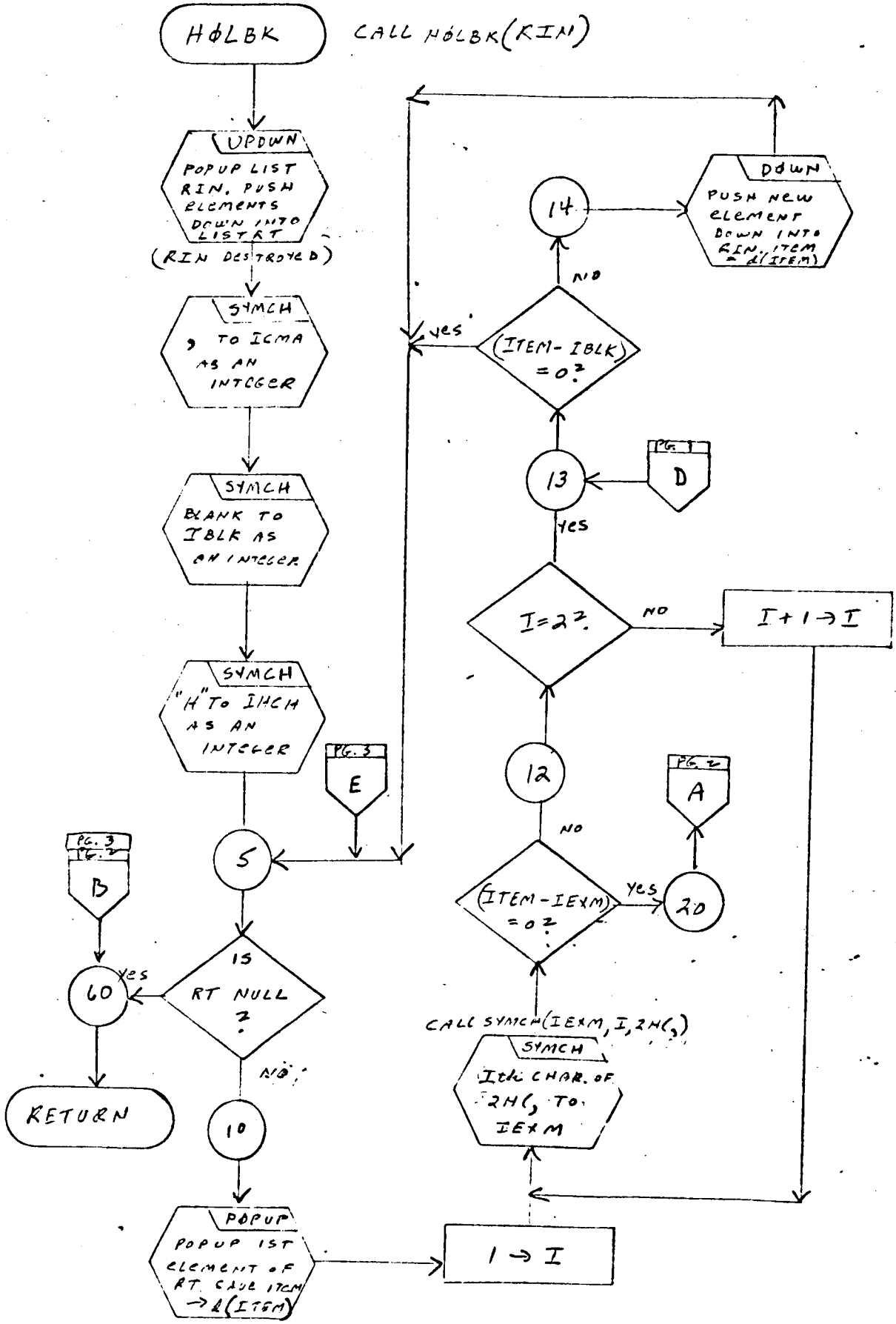
(d) If the character in ITEM is not a comma or an H, continue at step c(2).

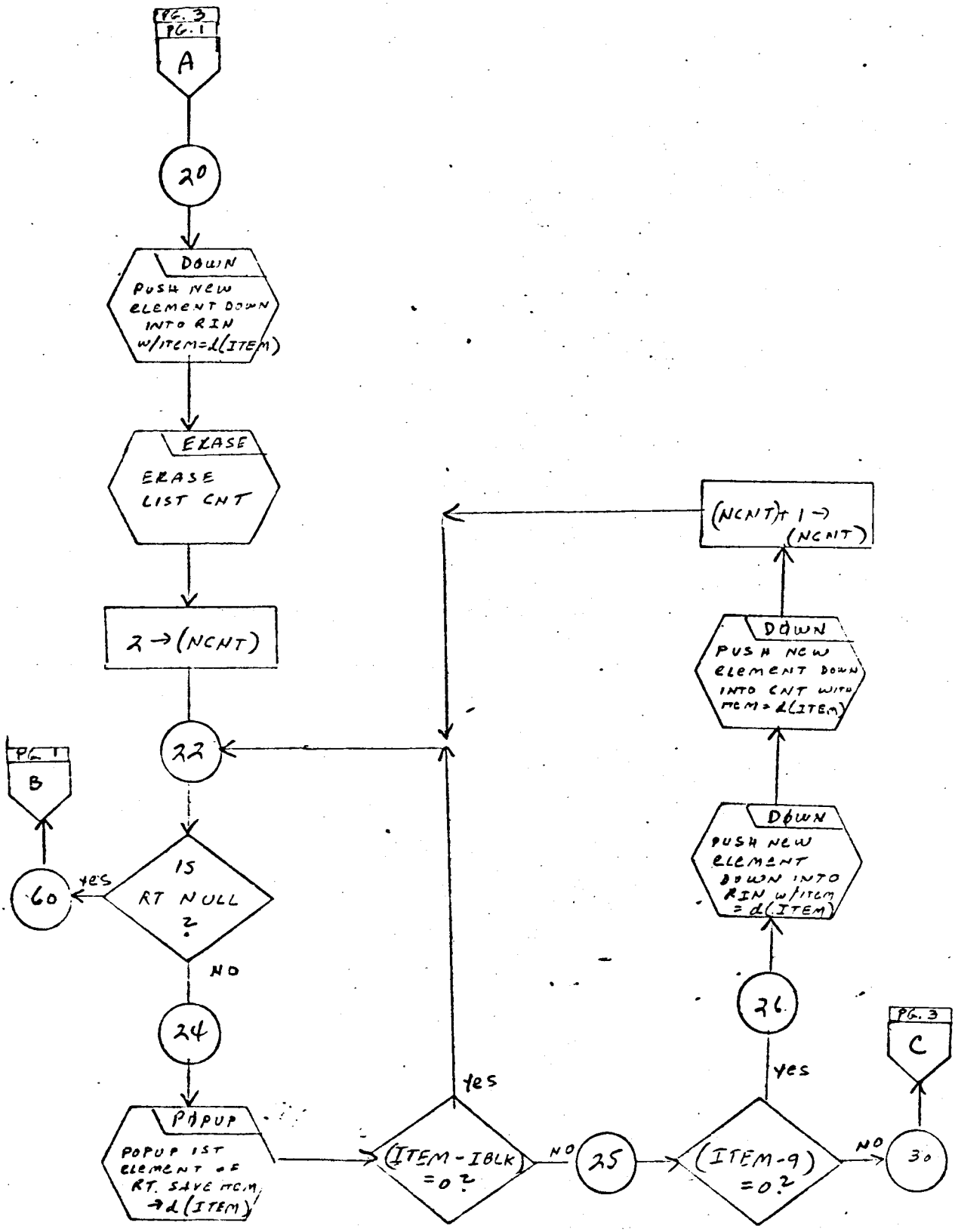
7. Other Subroutines Used

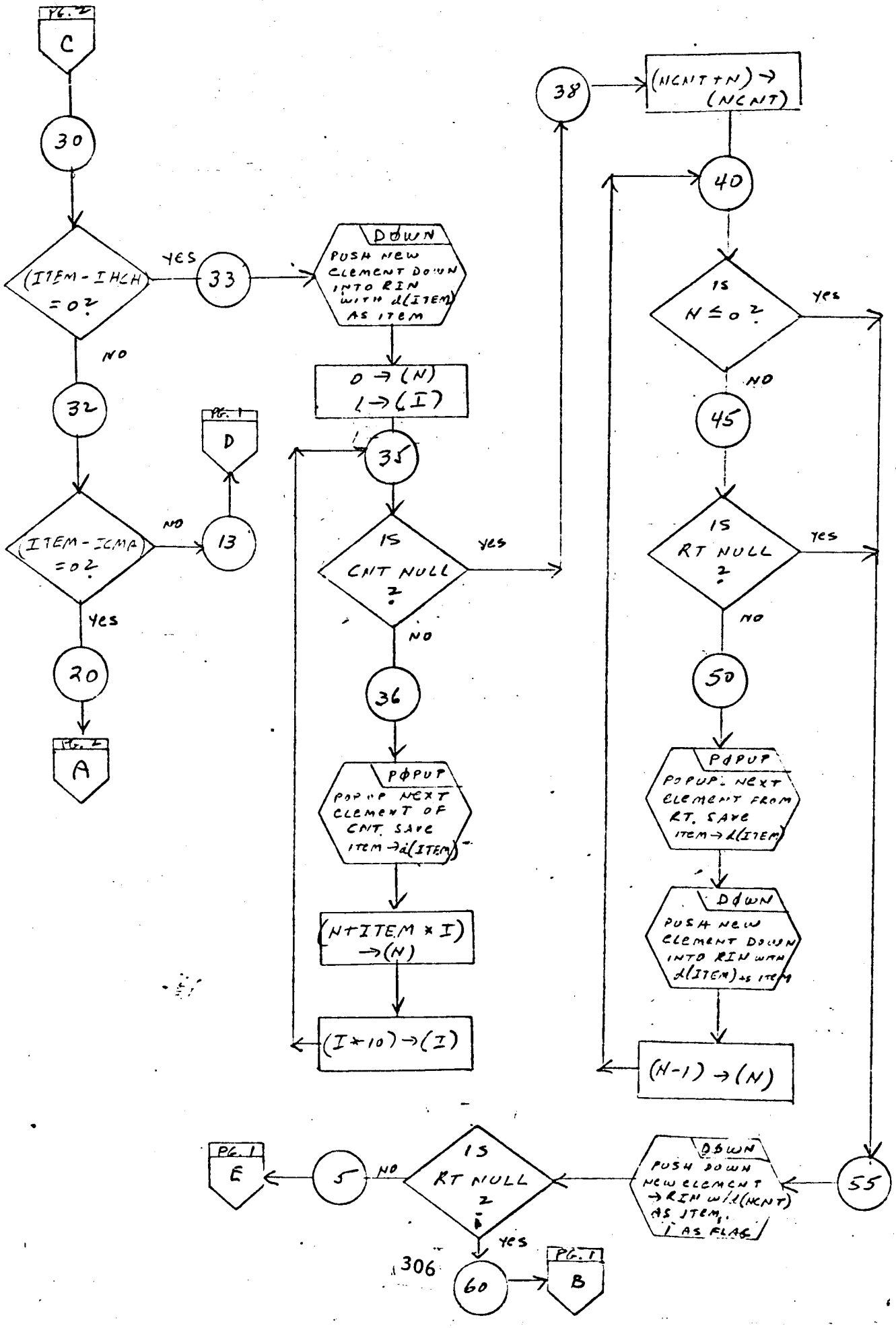
DOWN, ERASE, POPUP, SYMCH, UPDWN.

8. Using Subroutines

INZERO, SUBST.


HØLBK
CALL HØLBK(RIN)
UPDOWN
POPUP LIST RIN. PUSH ELEMENTS DOWN INTO LIST AT
(RIN DESTROYED)
SYMCH
, TO ICMA AS AN INTEGER
SYMCH
BLANK TO IBLK AS AN INTEGER
SYMCH
"H" TO IHCM AS AN INTEGER
PG. 3
E
5
PG. 3
PG. 2
B
60
yes
RETURN
IS RT NULL ?
NO
10
POPUP
POPUP IST ELEMENT OF RT. CALL ITEM → A(ITEM)
1 → I
CALL SYMCH(IEXM, I, 2H( )
SYMCH
Ith CHAR. OF 2H(TO IEXM
(ITEM - IEXM) = 0 ?
yes
20
PG. 2
A
NO
12
I = 2 ?
NO
I + 1 → I
yes
13
PG. 1
D
(ITEM - IBLK) = 0 ?
yes
NO
14
DOWN
PUSH NEW ELEMENT DOWN INTO RIN. ITEM = A(ITEM)


PG. 3
PG. 1
A
20
DOWN
PUSH NEW ELEMENT DOWN INTO RIN W/ITEM=d(ITEM)
ERASE
ERASE LIST CNT
2→(NCNT)
22
PG. 1
B
60
yes
IS RT NULL ?
NO
24
POPUP
POPUP 1ST ELEMENT OF RT. SAVE ITEM →d(ITEM)
(ITEM-IBLK) =0 ?
yes
NO
25
(ITEM-9) =0 ?
yes
NO
26
DOWN
PUSH NEW ELEMENT DOWN INTO RIN w/ITEM =d(ITEM)
DOWN
PUSH NEW ELEMENT DOWN INTO CNT WITH ITEM=d(ITEM)
(NCNT)+1→(NCNT)
PG. 3
C
30


PG. 2
C
30
(ITEM - IHCH) = 0?
YES
33
DOWN
PUSH NEW ELEMENT DOWN INTO RIN WITH d(ITEM) AS ITEM
NO
32
0 → (N)
1 → (I)
PG. 1
D
35
(ITEM - ICMA) = 0?
NO
13
IS CNT NULL ?
YES
38
(NCNT + N) → (NCNT)
Yes
20
NO
36
PG. 2
A
POPUP
POP UP NEXT ELEMENT OF CNT. SAVE ITEM → d(ITEM)
(N + ITEM * I) → (N)
(I * 10) → (I)
40
IS N ≤ 0 ?
YES
NO
45
IS RT NULL ?
YES
NO
50
POPUP
POPUP NEXT ELEMENT FROM RT. SAVE ITEM → d(ITEM)
DOWN
PUSH NEW ELEMENT DOWN INTO RIN WITH d(ITEM) AS ITEM
(N-1) → (N)
55
DOWN
PUSH DOWN NEW ELEMENT → RIN W/ d(NCNT) AS ITEM, 1 AS FLAG
IS RT NULL ?
NO
5
PG. 1
E
YES
60
PG. 1
B

Program Description

1. Identification

   a. Routine Label

      IDNTP

   b. Name

      Search an array of characters to match a given character.

2. Function

This subroutine searches N characters of a Hollerith array until one is found which matches a character in the item portion of the first element of a given list.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL IDNTP (X, I, N, H)

   b. Entry Conditions

      X = Head of a list
      N = Number of characters to be searched
      H = An array of Hollerith characters

   c. Exit Conditions

      If a match is found,
      I = the number of the character (in H) which matches the given character.
      If there is no match, I = N + 1.

   d. Error Exits

      None.

5. Definition of Identifiers

ITEM = The decrement of this cell contains the character to be matched.

ITEM1 = The decrement of this cell contains the Ith character of array H, right-adjusted.

6. Method

a. A character is extracted from the item portion of the first element in list X and saved in the decrement of ITEM.

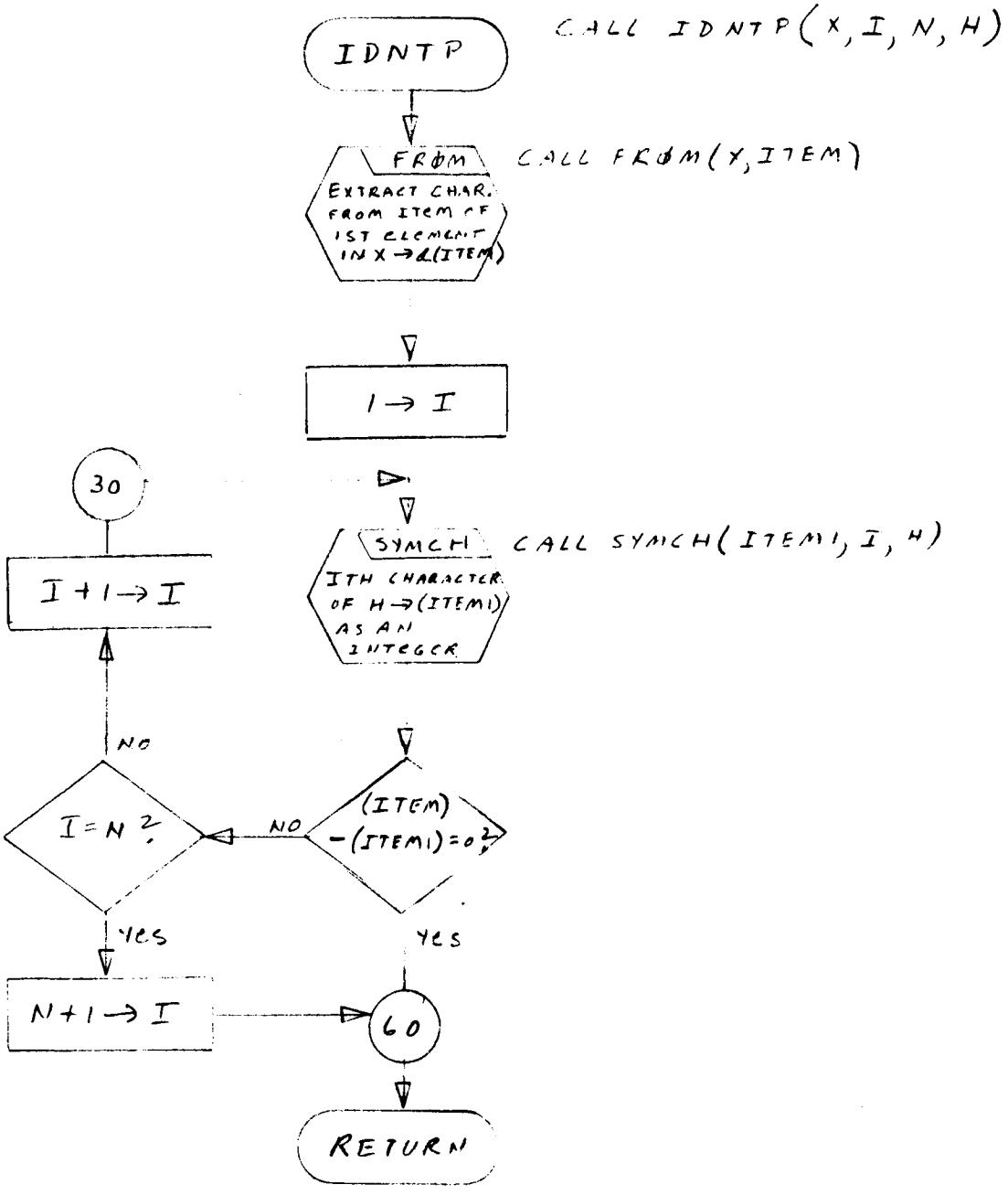
b. Each of the first N characters of the Hollerith array H is placed on ITEM1 and compared with ITEM until a match is found. I is set to the number of the character in H when there is a match. If none of the characters in H match the character in ITEM, I is set = N + 1.

7. Other Subroutines Used

FROM, SYMCH.

8. Using Subroutines

GOBLE, SEGMNT.


IDNTP
CALL IDNTP(X, I, N, H)
FROM
EXTRACT CHAR. FROM ITEM OF IST ELEMENT IN X → L(ITEM)
CALL FROM(X, ITEM)
1 → I
30
SYMCH
ITH CHARACTER OF H → (ITEM1) AS AN INTEGER
CALL SYMCH(ITEM1, I, H)
I + 1 → I
NO
(ITEM) - (ITEM1) = 0 ?
NO
I = N ?
yes
yes
N + 1 → I
60
RETURN

Program Description

1. Identification

   a. Routine Label

      INFL

   b. Name

      Replace the contents of an element.

2. Function

This subroutine replaces the contents of the first element of a list with the contents of a given word.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL INFL (I, A)

   b. Entry Conditions

      I = Head of a list
      A = A full word of information

   c. Exit Conditions

      The contents of A replace the contents of the first element of list I.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

   a. d(I) → (LOC2)
   b. (A) → ((LOC2))

7. Other Subroutines Used

None.

8. Using Subroutines

STASH

Program Description

1. Identification

   a. Routine Label

      INPUTX

2. Function

   To write out the "CALL INPUT" statement

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INPUTX (IS, INSTP, INLST)

   b. Entry Conditions

IS	=	The statement number to be used for the "CALL INPUT" statement
INSTP	=	Contains a list-structure of system parameters (the list INSTP of Main 2)
INLST	=	Contains a list of input variable names (the list INLST of Main 2)

   c. Exit Conditions

      The statement "CALL INPUT ..." is output.

   d. Error Exits

      None.

5. Definition of Identifiers

Z	Type A list initially containing " , 6H$$$END)" (this list is used to construct the output card image)
INPUT	Type A list containing "CALL INPUT ("
ICM	BCD comma

PAR1	BCD "S"
ISP	BCD blank
IBRK	BCD right parenthesis
IBLK	BCD blank
IH	BCD "H"
ISEP	BCD " / "
IN1	Used to link through INSTP
NE, NO, NT, NTRN, NNTR, NF	Used as parameters to examine a symbol in IN1 through the PRPTR subroutine
P	Holds the "input" symbols as they are constructed
ITEM	Used to hold the top element of P
Z1	Used to hold a symbol temporarily before it is placed in Z.
IC	Used to compute the continuation card number
A	Output buffer holding the completed card image

6. Method

a. The symbols in INSTP are extracted, one at a time, and sent to PRPTR for reduction. The output parameters NE, NO, NT are used to construct a four-letter word of this form: SX · Y · Z

X is V, C, G, L, $, I, depending on the value of NE

Y is NO

Z is NT

This symbol is pushed into list P.

b. The list INLST is RECOVR-d, and each symbol is pushed into P.

c. The list P is then examined. Each symbol (e.g., SV12) is popped out and pushed into Z. The number of characters (e.g., 4) is determined, and a special symbol (e.g., 4HSV12,) is constructed and pushed into Z. When P is exhausted, the statement

CALL INPUT ("Z-list" , 6H$$$END)

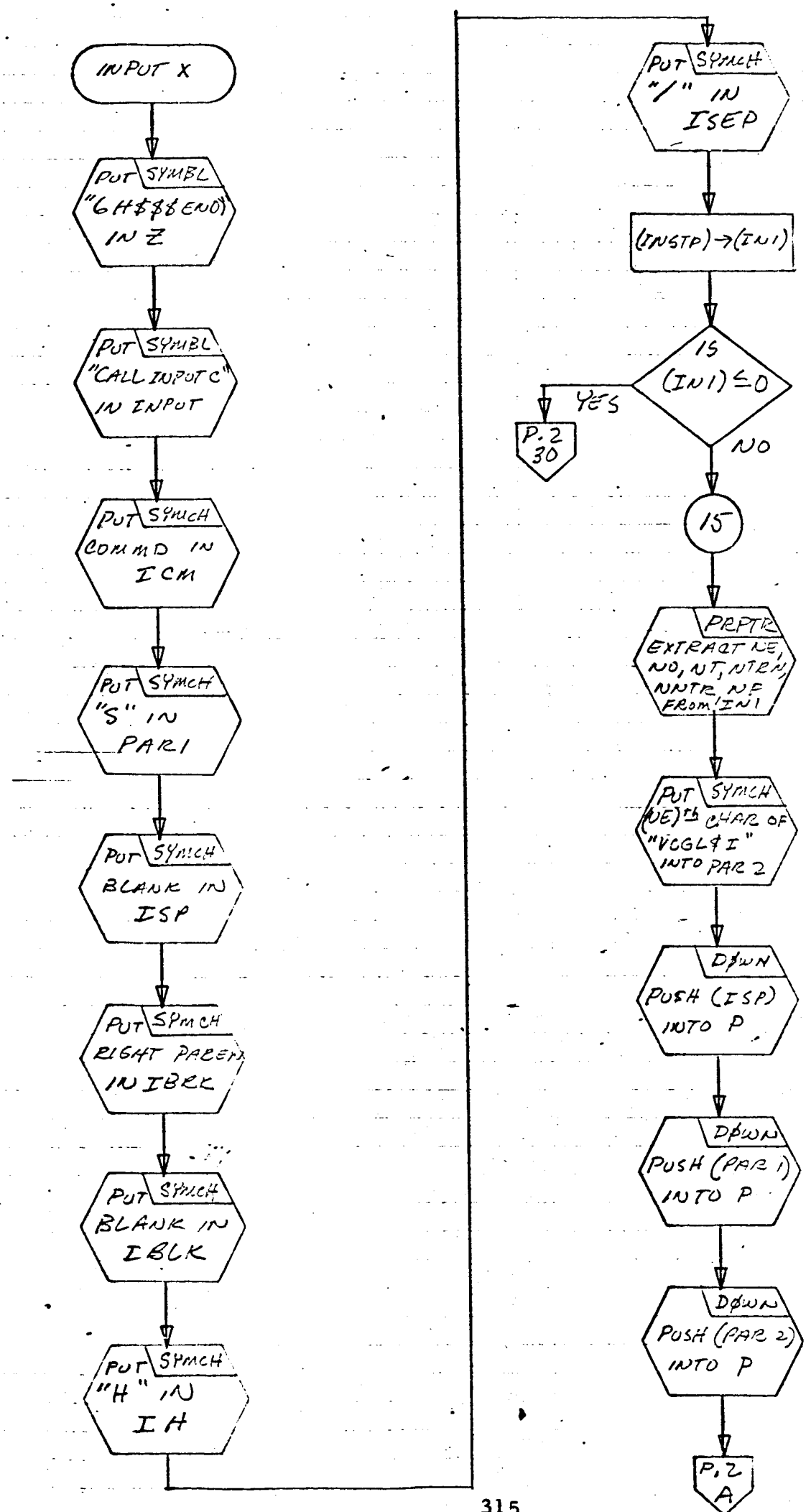
is output with statement number IS.

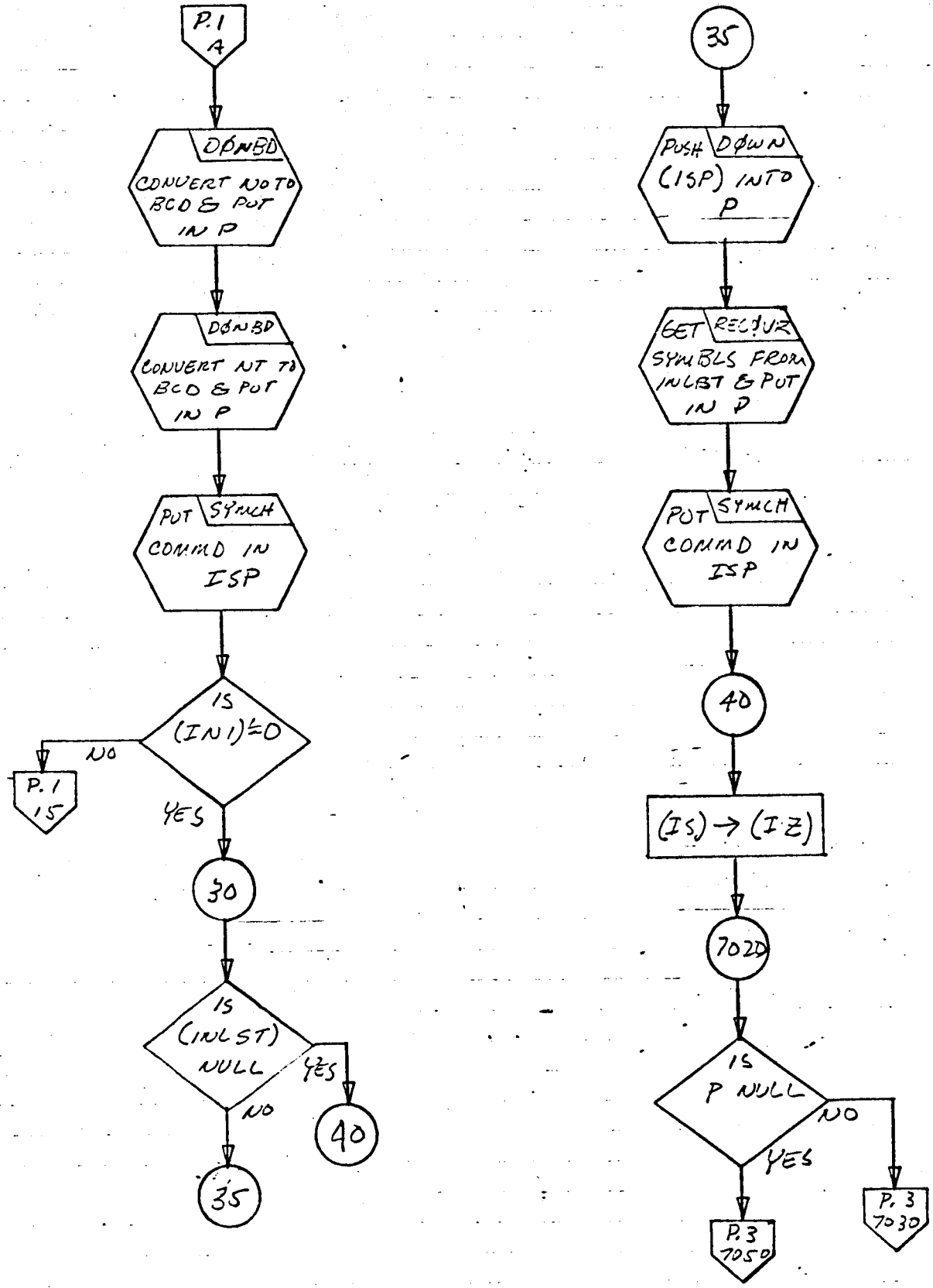
7. Other Subroutines Used

BCL, CHLNE, COPY, DONBD, DOWN, ERASE, FOUTPT, HEAD, POPUP, PRPTR, RECOVR, SYMBL, SYMCH, UPDWN.

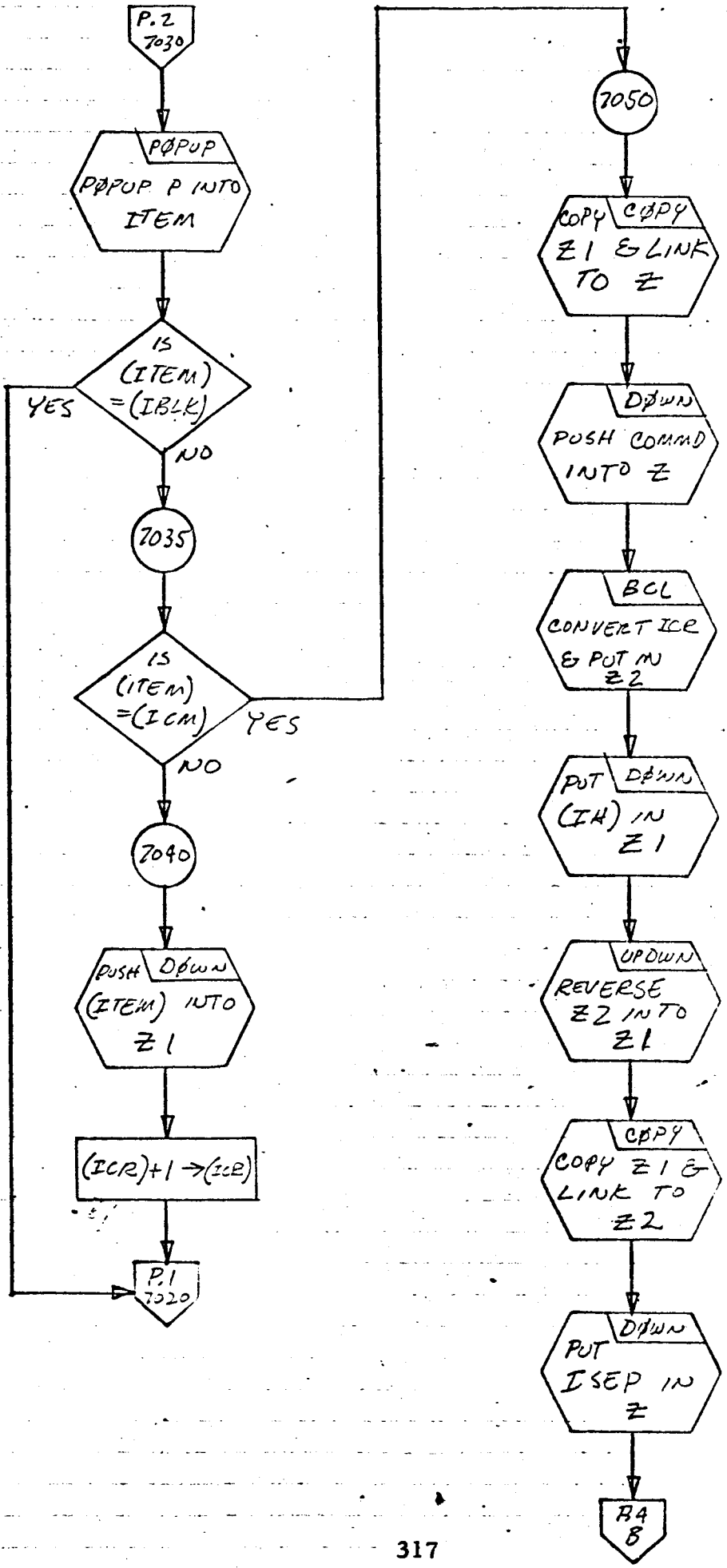
8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.

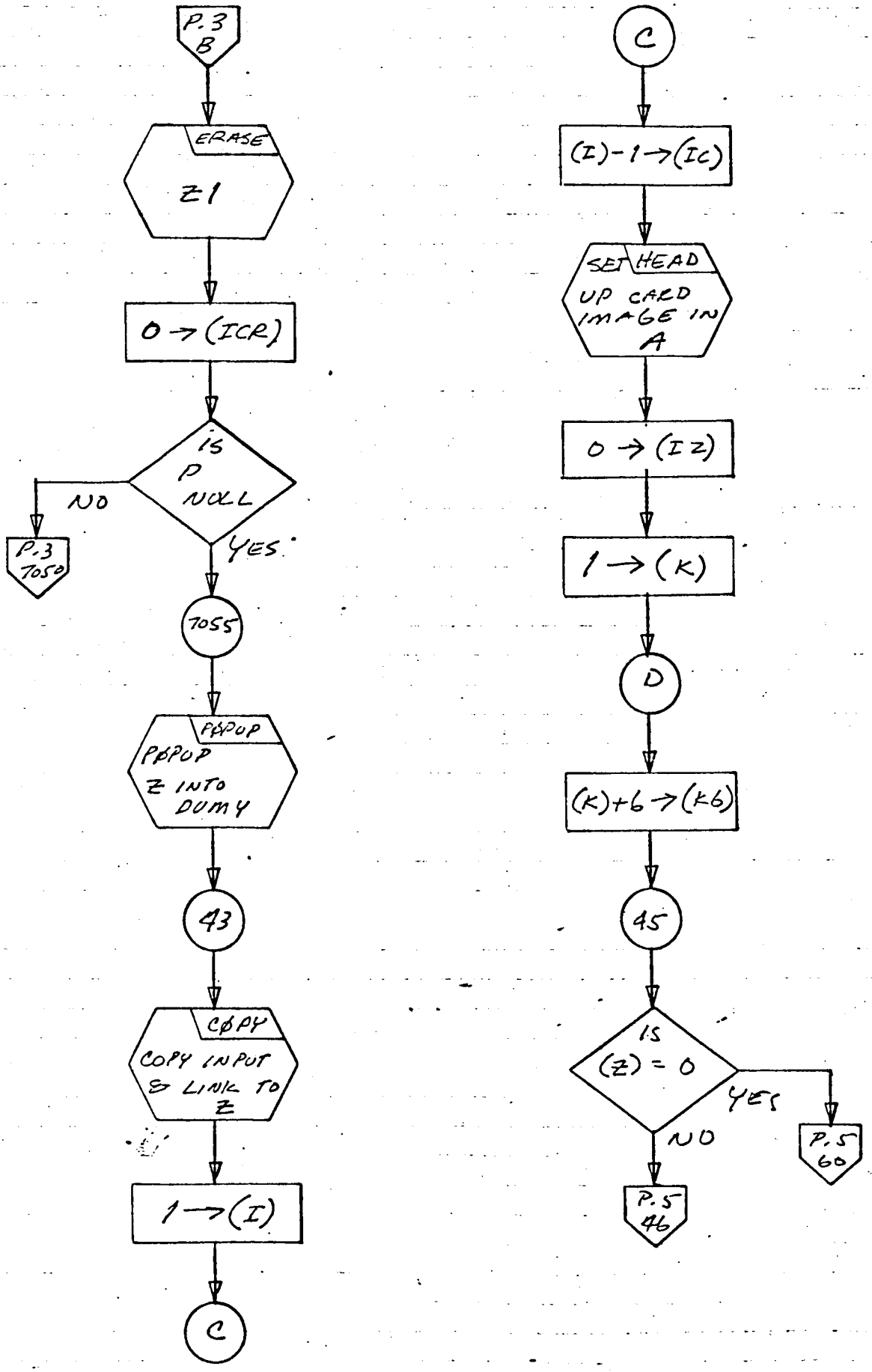

INPUT X
PUT SYMBL "6H$$$END" IN Z
PUT SYMBL "CALL INPUT C" IN INPUT
PUT SYMCH COMMD IN ICM
PUT SYMCH "S" IN PAR1
PUT SYMCH BLANK IN ISP
PUT SYMCH RIGHT PAREN IN IBRK
PUT SYMCH BLANK IN IBLK
PUT SYMCH "H" IN IH
PUT SYMCH "/" IN ISEP
(INSTP)→(IN1)
IS (IN1) ≤ 0
YES
P. 2 30
NO
15
PRPTR EXTRACT NE, NO, NT, NTRM, NNTR, NF FROM IN1
PUT SYMCH (NE)th CHAR OF "VCGL$I" INTO PAR 2
DOWN PUSH (ISP) INTO P
DOWN PUSH (PAR 1) INTO P
DOWN PUSH (PAR 2) INTO P
P, 2 A


P.1
A
DØNBD
CONVERT NO TO BCD & PUT IN P
DØNBD
CONVERT NT TO BCD & PUT IN P
PUT SYNCH
COMMD IN ISP
IS (IN1)≠0
NO
P.1
15
YES
30
IS (INLST) NULL
YES
NO
40
35
35
PUSH DØWN
(ISP) INTO P
GET RECIVZ
SYMBLS FROM INLST & PUT IN P
PUT SYNCH
COMMD IN ISP
40
(IS) → (IZ)
7020
IS P NULL
NO
YES
P.3
7030
P.3
7050

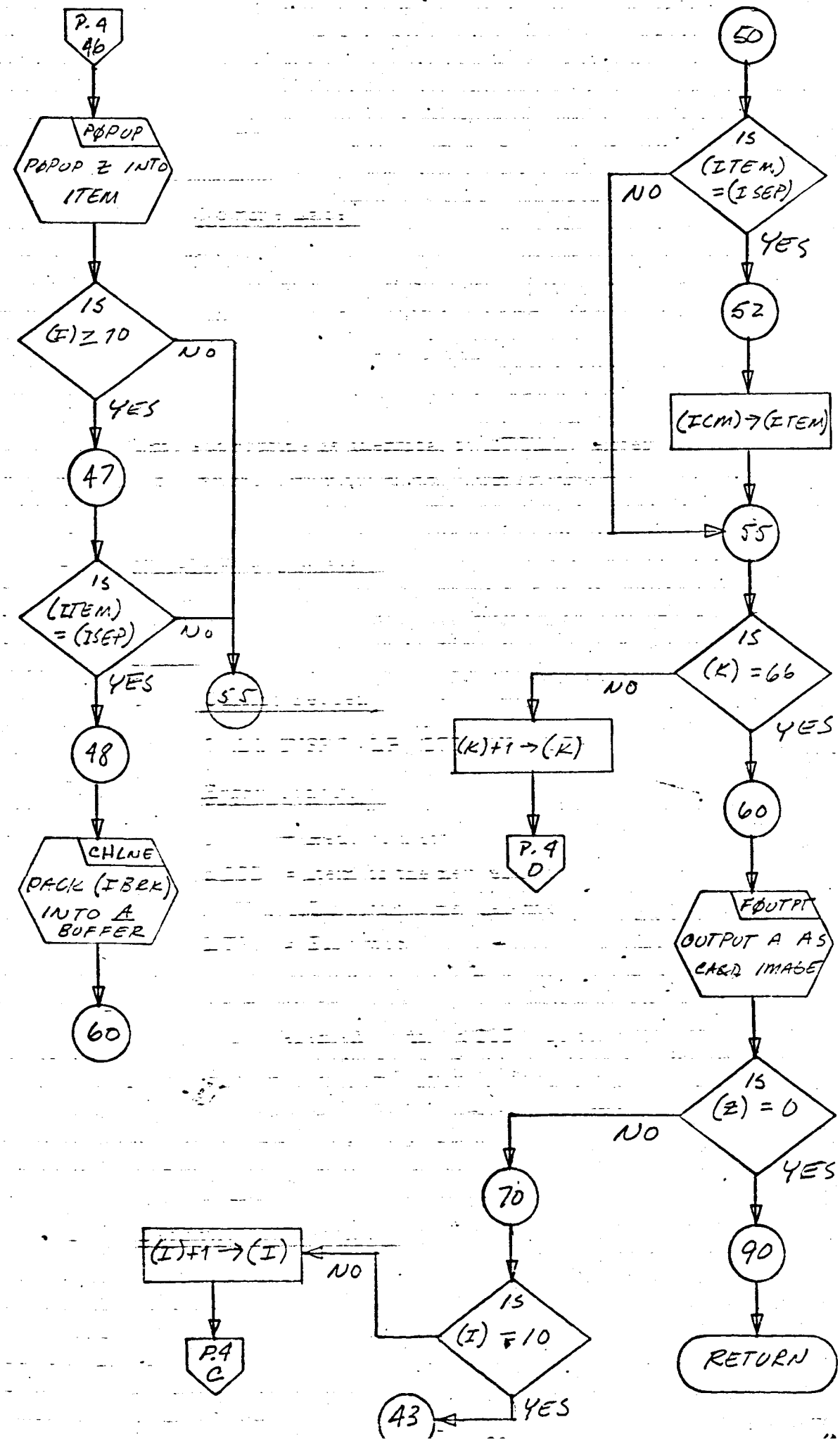
INPUT X 2


P.2
7030
PØPUP
PØPUP P INTO
ITEM
IS
(ITEM)
=(IBLK)
YES
NO
7035
IS
(ITEM)
=(ICM)
YES
NO
7040
PUSH DØWN
(ITEM) INTO
Z1
(ICR)+1 →(ICR)
P.1
7020
7050
COPY CØPY
Z1 & LINK
TO Z
DØWN
PUSH COMMD
INTO Z
BCL
CONVERT ICR
& PUT IN
Z2
PUT DØWN
(IH) IN
Z1
UPDOWN
REVERSE
Z2 INTO
Z1
CØPY
COPY Z1 &
LINK TO
Z2
DØWN
PUT
ISEP IN
Z
B4
B

INPUT X 3


P.3
B
ERASE
Z1
0 → (ICR)
IS
P
NULL
NO
YES
P.3
7050
7055
PØPOP
PØPOP
Z INTO
DUMY
43
CØPY
COPY INPUT
& LINK TO
Z
1 → (I)
C
C
(I) - 1 → (IC)
SET HEAD
UP CARD
IMAGE IN
A
0 → (IZ)
1 → (K)
D
(K) + 6 → (K6)
45
IS
(Z) = 0
YES
NO
P.5
60
P.5
46

INPUT K 4


P. 4 46
POPUP
POPUP Z INTO ITEM
IS (I) ≥ 10
NO
YES
47
IS (ITEM) = (ISEP)
NO
YES
55
48
CHLNE
PACK (IBEK) INTO A BUFFER
60
50
IS (ITEM) = (ISEP)
NO
YES
52
(ICM) → (ITEM)
55
IS (K) = 66
NO
YES
(K)+1 → (K)
P. 4 D
60
FOUTPT
OUTPUT A AS CARD IMAGE
IS (Z) = 0
NO
YES
70
90
RETURN
(I)+1 → (I)
NO
P.4 C
IS (I) = 10
YES
43

Program Description

1. Identification

   a. Routine Label

      INSRT

   b. Name

      INSERT a new element into a list.

2. Function

This subroutine is identical to AFTLK, except that when the list is null, the address portion of the element inserted into the list is cleared to zero.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INSRT (LP, ICD, NT, LTP)

   b. Entry Conditions

      LP = Head of a list
      d(ICD) = Item of the new element
      d(NT) = Flag of the new element
      LTP = Flag word

   c. Exit Conditions

A new element, with d(ICD) as its item and d(NT) as its flag, is inserted into list LP as its second element. The pointer in LP is reset to this new element. If the new element is the only element in the list, its location is saved in LTP.

   d. Error Exits

5. Definition of Identifiers

   None

6. Method

a. If list LP was not null:

(1) A new element is inserted immediately after the first element.

(2) The first element is linked to the new element.

(3) The head cell of list LP is reset to point to the new element.

(4) The new element is linked to the element that was formerly the second one in list LP.

b. If list LP was null:

(1) A new element is inserted with a link of zero, creating a list of just one element.

(2) The head cell of list LP will be set to point to this element.

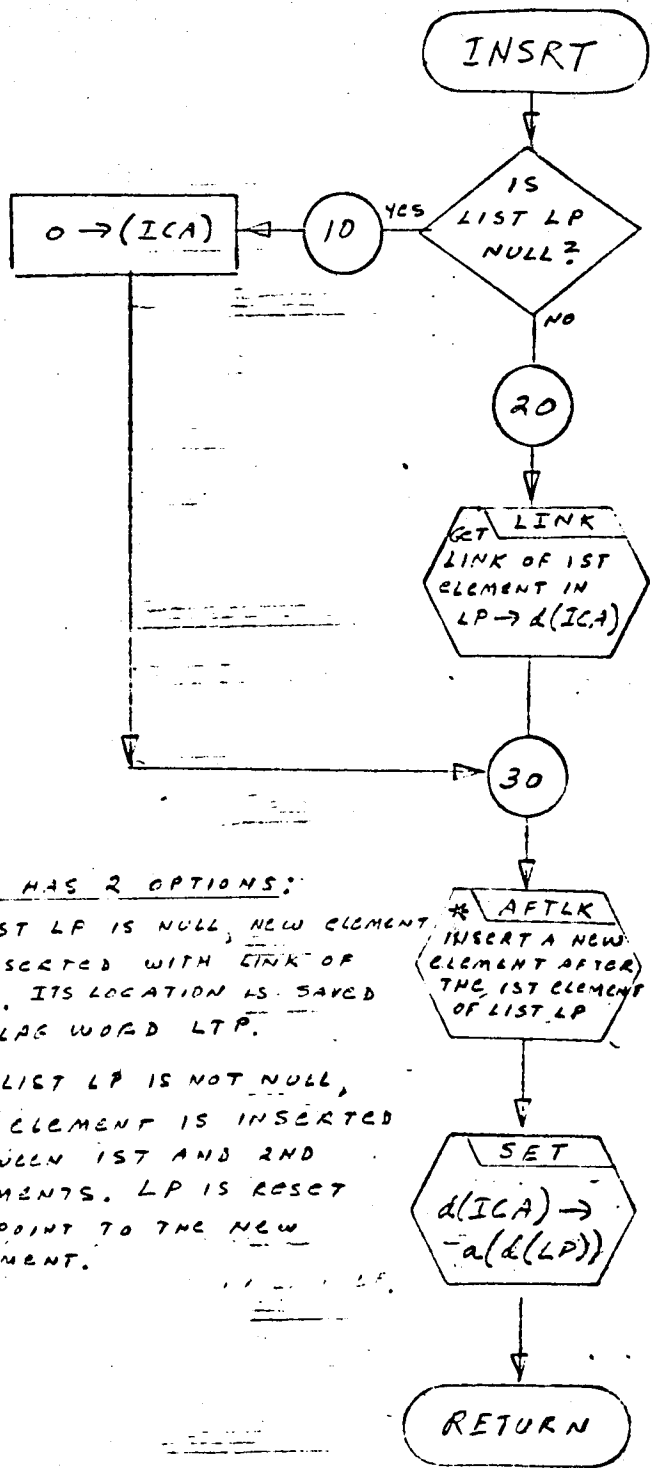
(3) Location of the single element in LP is saved in the flag word LTP.

7. Other Subroutines Used

AFTLK, LINK, SET.

8. Using Subroutines

MATFT, STASH.



CALL INSRT(LP, ICD, NT, LTP)

CALL LINK(LP, ICA)

CALL AFTLK(LP, ICD, NT, LTP)

CALL SET(LP, ICA)

* AFTLK HAS 2 OPTIONS:

1. IF LIST LP IS NULL, NEW ELEMENT IS INSERTED WITH LINK OF ZERO. ITS LOCATION IS SAVED IN FLAG WORD LTP.
2. IF LIST LP IS NOT NULL, NEW ELEMENT IS INSERTED BETWEEN 1ST AND 2ND ELEMENTS. LP IS RESET TO POINT TO THE NEW ELEMENT.

Program Description

1. Identification

   a. Routine Label

      INTLST

2. Function

   Reads type 2 records from tape and stores the information in a list.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INTLST (INLST, NTAPE)

   b. Entry Conditions

      INLST = The list to be updated by the read-in information (It is type- .)

      NTAPE = The tape to be read

   c. Exit Conditions

      INLST is updated.

   d. Error Exits

      None.

5. Definition of Identifiers

ITYPE NE J NF FTEM NN NP HOLTH	parameters read in from NTAPE

6. Method

Read records from NTAPE in this format:

Columns	Format	Destination
2	integer	ITYPE
4-6	integer	NE
8-10	integer	J
13	integer	NF
15-30	real number	FTEM
32-34	integer	NN
36-38	integer	NP
40-75	Hollerith	HOLTH

If ITYPE = 2, enter the six quantities NE, NN, NP, 0, 0, 0 into INLST through PRPTG.

If ITYPE = 5, exit.

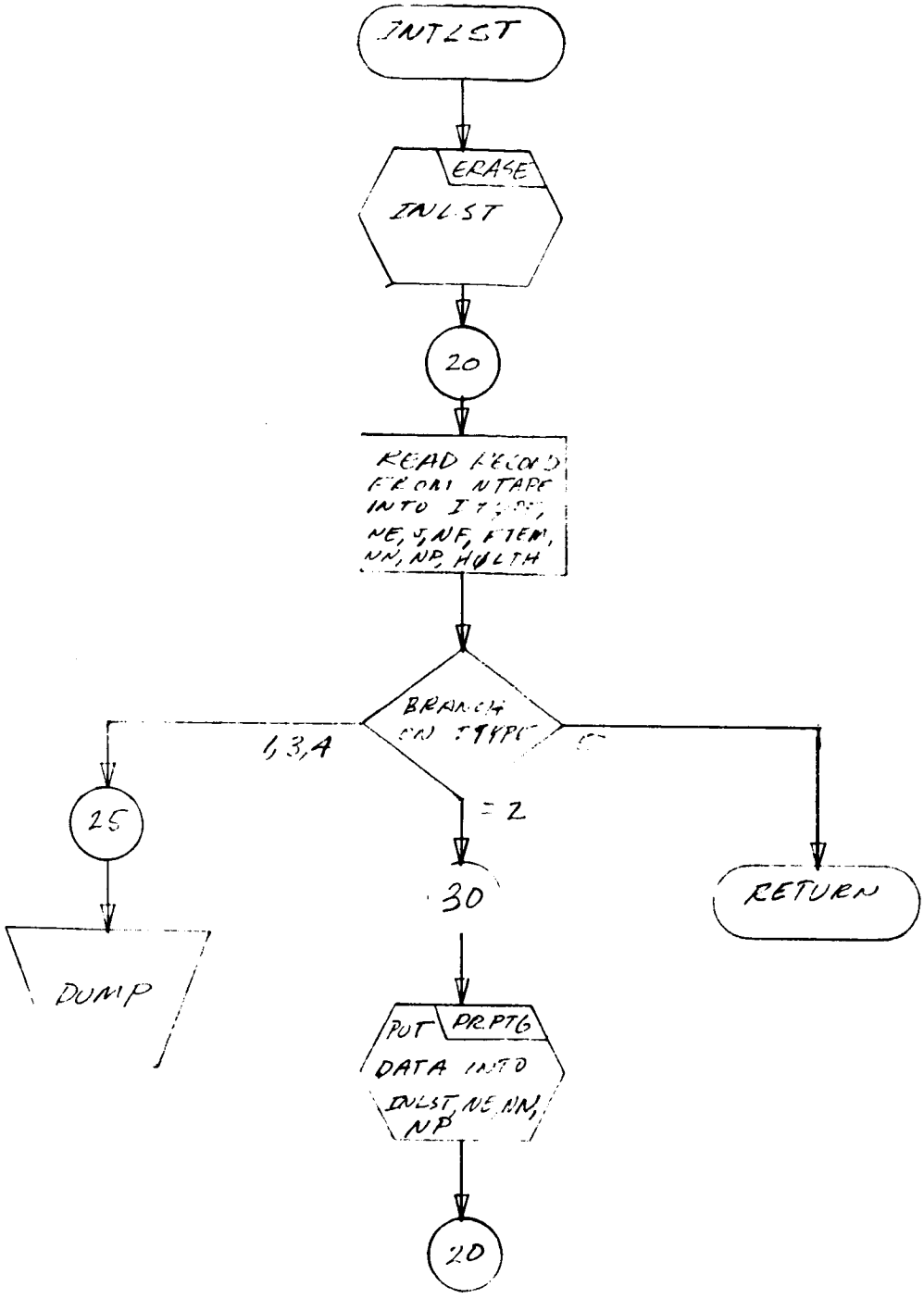
If ITYPE is anything but 2 or 5, then CALL DUMP and exit.

7. Other Subroutines Used

DUMP, ERASE, PRPTG.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


INTLST
ERASE
INLST
20
READ RECORD
FROM NTAPE
INTO I
NE, J, NF, ITEM,
NN, NP, HOLTH
BRANCH
ON TYPE
1,3,4
=2
25
30
RETURN
DUMP
PUT
PRPTG
DATA INTO
INLST, NE, NN,
NP
20

Program Description

1. Identification

   a. Routine Label

      INTO

   b. Name

      Replace item and flag of first element of a list.

2. Function

The item portion and flag portion of the first element of a list is updated. The address (link) is unchanged. If the list is null, a new element with a link of zero is pushed down into a new list.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL INTO (Z, I, IF)

   b. Entry Conditions

      Z = Head of a list
      d(I) = Item
      d(IF) = Flag

   c. Exit Conditions

      d (1st element) = Item
      t (1st element) = Flag

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Definition of Identifiers

None.

7. Method

If the head of list Z = 0, it is considered to be null. A new element is obtained from AVS (using NEWLOC), and this element is pushed down into Z with d(I) as its item, d(IF) as its flag, and a link of zero.

If the head of list Z $\neq$ 0, d(I) and d(IF) replace the item and flag of the first element of Z, the link portion of the element is unchanged.

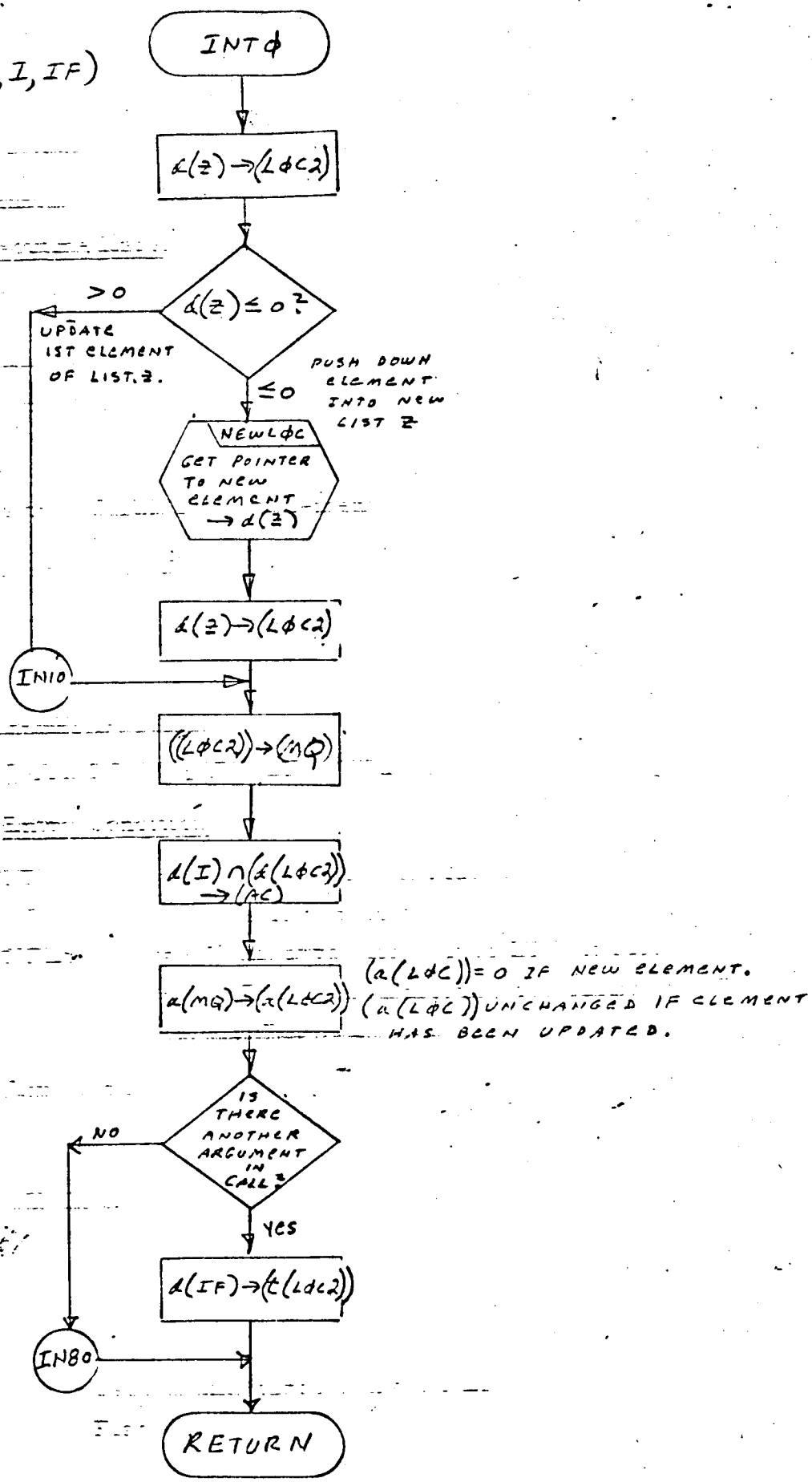
8. Other Subroutines Used

NEWLOC

9. Using Subroutines

AFTER, DOWN, FLAG, LEVMRK, MATFT, MRKLST, SUBST.

CALL INTϕ(Z, I, IF)



Program Description

1. Identification

   a. Routine Label

      INVST

   b. Name

      Write "CALL INV" statement.

2. Function

   Outputs the "CALL INV" statements.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INVST (NPT, LEV, NFLS, I)

   b. Entry Conditions

NPT	=	Array NPT of Main 2
LEV	=	A flag indicator
NFLS	=	An array list. It is XFG of Main 2.
I	=	The index indicating the sublist of NPT, NFLS to be processed

   c. Exit Conditions

      "CALL INV" statements output.

   d. Error Exits

      None.

5. Definition of Identifiers

KSW	Control indicator
N1	Used to link through list NFLS(I)
NF	Flag (N1)

L1	Used to link through the sublist of N1
NT	Item (L1)
NEX	Flag (L1)

6. Method

KSW is set = 1

N1 is set = NFLS(I)

If N1 is null, then if LEV = 0, output statement, else exit.

If N1 is not null, N1 is examined, one sublist at a time. L1 is set equal to the sublist, and the flag portion of each element of the sublist is compared with LEV.

If (NEX) > (LEV), exit.

If (NEX) = (LEV), set KSW = 2 and continue.

If (NEX) < (LEV), continue.

When done, if KSW = 2, exit.

Else, output statement according to the following schedule:

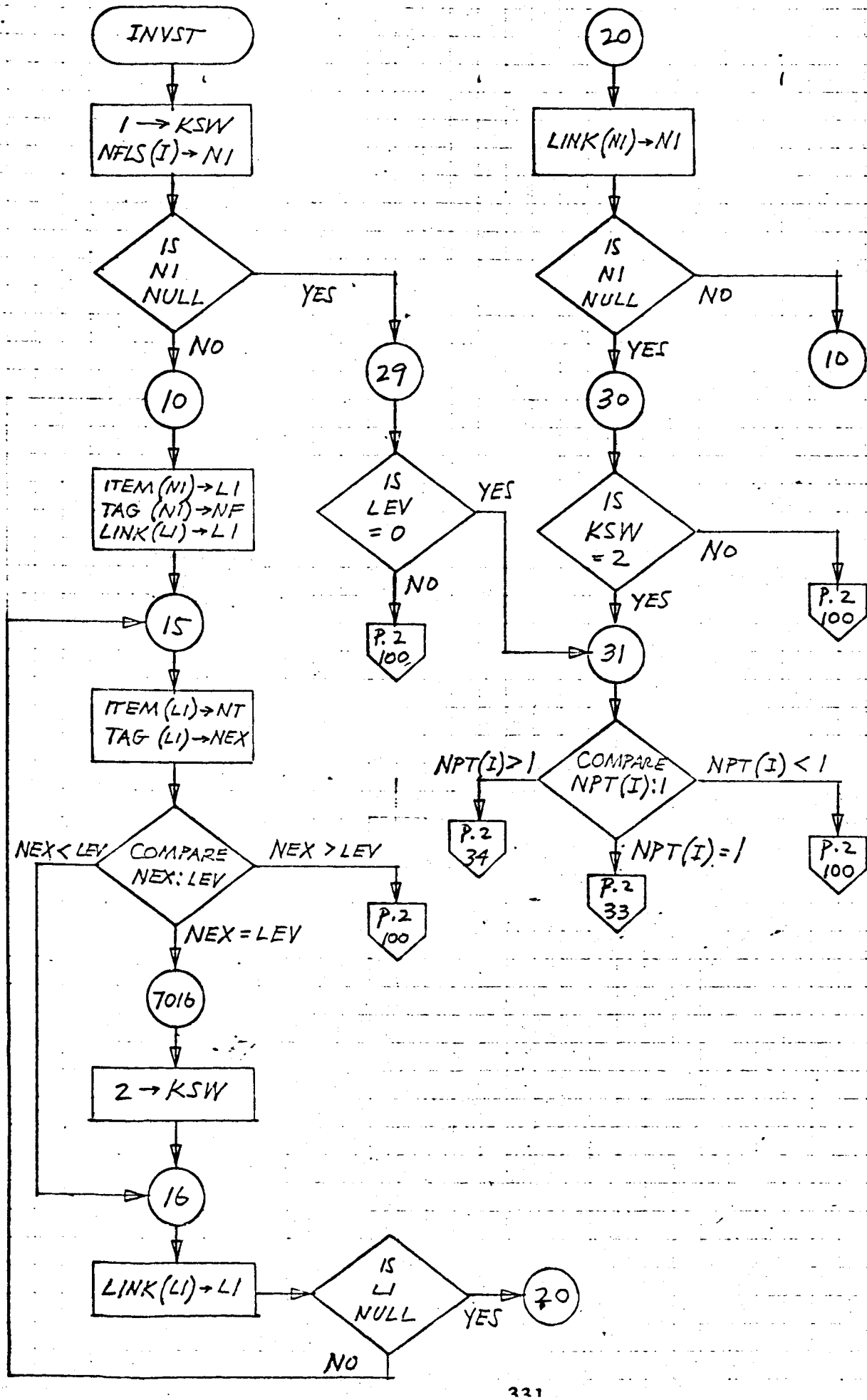
	NPT(I) ≤ 0	NPT(I) = 1	NPT (I) > 1
I = 1	exit	exit	exit
I = 2	exit	FCI22 = 1. / FC22	CALL INV(FC22, FCI22, LNC)
I = 3	exit	FGI22 = 1. / FG33	CALL INV(FG33, FGI33, LNG)
I = 4	exit	FLI44 = 1. / FL44	CALL INV (FL44, FLI44, LNL)

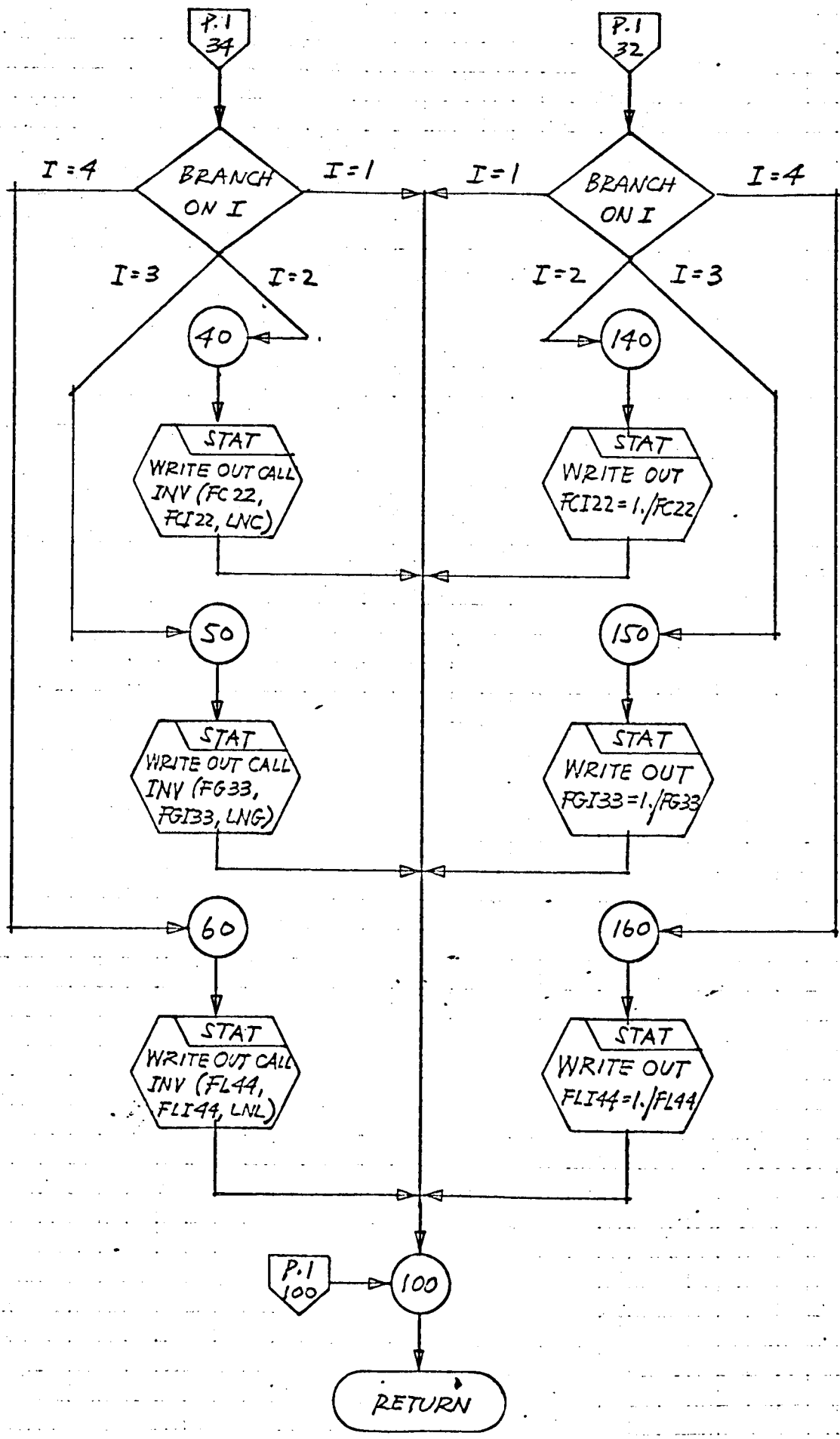
7. Other Subroutines Used

FROM, LINK, STAT.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


INVST
1 → KSW
NFLS(I) → N1
IS N1 NULL
YES
NO
29
10
ITEM(N1) → L1
TAG (N1) → NF
LINK(L1) → L1
15
ITEM(L1) → NT
TAG (L1) → NEX
COMPARE NEX: LEV
NEX < LEV
NEX > LEV
NEX = LEV
7016
2 → KSW
16
LINK(L1) → L1
IS L1 NULL
YES
NO
20
IS LEV = 0
YES
NO
P.2 100
20
LINK(N1) → N1
IS N1 NULL
NO
YES
10
30
IS KSW = 2
NO
YES
P.2 100
31
COMPARE NPT(I): 1
NPT(I) > 1
NPT(I) < 1
NPT(I) = 1
P.2 34
P.2 100
P.2 33


P.1 34
BRANCH ON I
I=4
I=1
I=3
I=2
40
STAT
WRITE OUT CALL INV (FC22, FCI22, LNC)
50
STAT
WRITE OUT CALL INV (FG33, FGI33, LNG)
60
STAT
WRITE OUT CALL INV (FL44, FLI44, LNL)
P.1 32
BRANCH ON I
I=1
I=4
I=2
I=3
140
STAT
WRITE OUT FCI22=1./FC22
150
STAT
WRITE OUT FGI33=1./FG33
160
STAT
WRITE OUT FLI44=1./FL44
P.1 100
100
RETURN

/NVST 2

Program Description

1. Identification

   a. Routine Label

      INZERO

2. Function

   Tests if a BCD string contains "INPUT" or "ZERO."

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INZERO (RIN, KES)

   b. Entry Conditions

      RIN is a simple list (type A) in reverse order.

   c. Exit Conditions

      KES = 1 (No find)
      KES = 2 (RIN contains "INPUT")
      KES = 3 (RIN contains "ZERO")
      RIN is erased for KES = 2, 3

   d. Error Exits

      None.

5. Definition of Identifiers

TEM	Temporary list used to hold RIN in reverse order
IEXM	Used to hold literal BCD chars for comparison with items in TEM
ITEM	Used to hold the current item of TEM

6. Method

KES is set = 1.

RIN is reversed into TEM, and all blank characters are removed. The top item of TEM is then examined. If it is an "I", then the rest of TEM is examined. If the rest of TEM contains "NPUT" and no other characters, KES is set = 2, RIN is erased, and exit is made. If the first character is an I, but the rest of RIN does not match, then exit with RIN as on input. Match, then exit with RIN as on input.

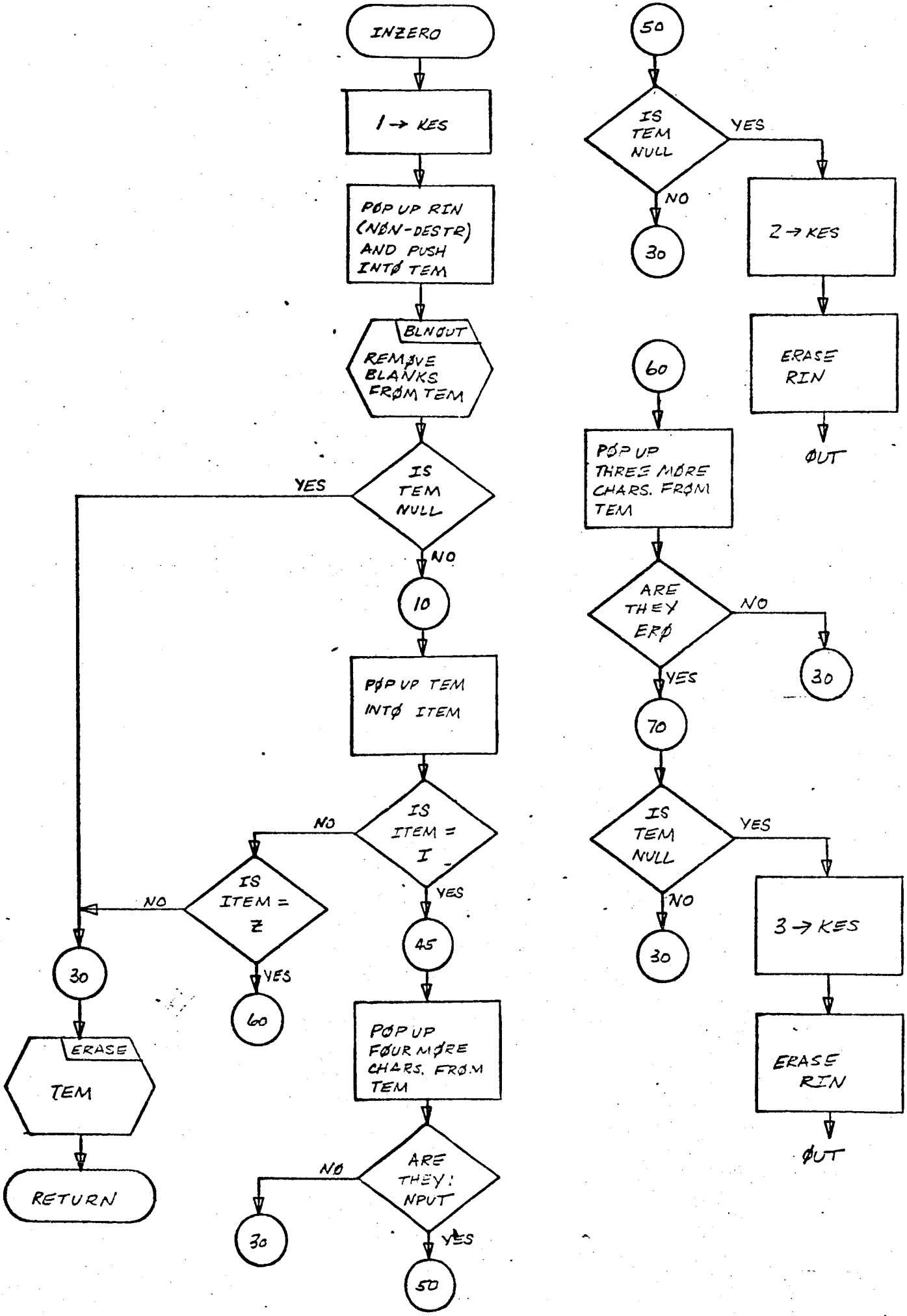
If the first item is "Z," then if the rest of TEM contains "ERO" and no other characters, KES is set = 3, RIN is erased, and exit is made. Else exit with RIN as on input.

7. Other Subroutines Used

BLNOUT, DOWNS, ERASE, POPUP, SYMCH.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


INZERO
1 → KES
POP UP RIN (NON-DESTR) AND PUSH INTO TEM
BLNOUT
REMOVE BLANKS FROM TEM
IS TEM NULL
YES
NO
10
POP UP TEM INTO ITEM
IS ITEM = I
NO
YES
IS ITEM = Z
NO
YES
30
60
45
ERASE TEM
RETURN
POP UP FOUR MORE CHARS. FROM TEM
ARE THEY INPUT
NO
YES
30
50
50
IS TEM NULL
YES
NO
30
2 → KES
ERASE RIN
ØUT
60
POP UP THREE MORE CHARS. FROM TEM
ARE THEY ERØ
NO
YES
30
70
IS TEM NULL
YES
NO
30
3 → KES
ERASE RIN
ØUT

Program Description

1. Identification

   a. Routine Label

      LEVMRK

2. Function

   To update the flag portion of the major XS matrices.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL LEVMRK (X, IFLS, NM, NTAPE1, NTAPE2)

   b. Entry Conditions

      X = An array list used as a base for absorbing the XS matrices from tape
      IFLS = An array list of type
      NM = The array size of X
      NTAPE1 = The input tape
      NTAPE2 = The output tape

   c. Exit Conditions

      Revised lists XS are output to NTAPE2.

   d. Error Exits

      None.

5. Definition of Identifiers

   I Index number of the matrix retrieved from NTAPE1
   N1 IFLS(I)
   L1 Next item of N1
   NO First item of L1
   NT Second item of L1

NEX Second flag of L1

I1, J } Used to index through X

LJ Used to hold position in X(I1, J)

KSW "Found" (indicator set by LOCATA)

LJJ Item (LJ)

JP Link (LJJ)

NEF Flag (LJJ)

6. Method

Each list XS is read from NTAPE1 and absorbed into X through MATFT, and flags found through IFLS are used to update X. Then X is output to NTAPE2 through MATOT. This is done for four matrix lists, using I = 2,5. The procedure is as follows:

a. Read and merge XS into X.

b. For each triplet NO, NT, NEX on IFLS(I), update X:

(1) For each I1, J where I1 and J range from 1 to NM, find

$$JP = LOCATA\ (x(I1, J))$$

and then

$$LJ = LOCAT\ (JP\,(NO, NT))$$

(2) If such a quadruplet exists, then update its flag NEF by setting

$$NEF = \max\ (NEF, NEX)$$

c. Output X to NTAPE2

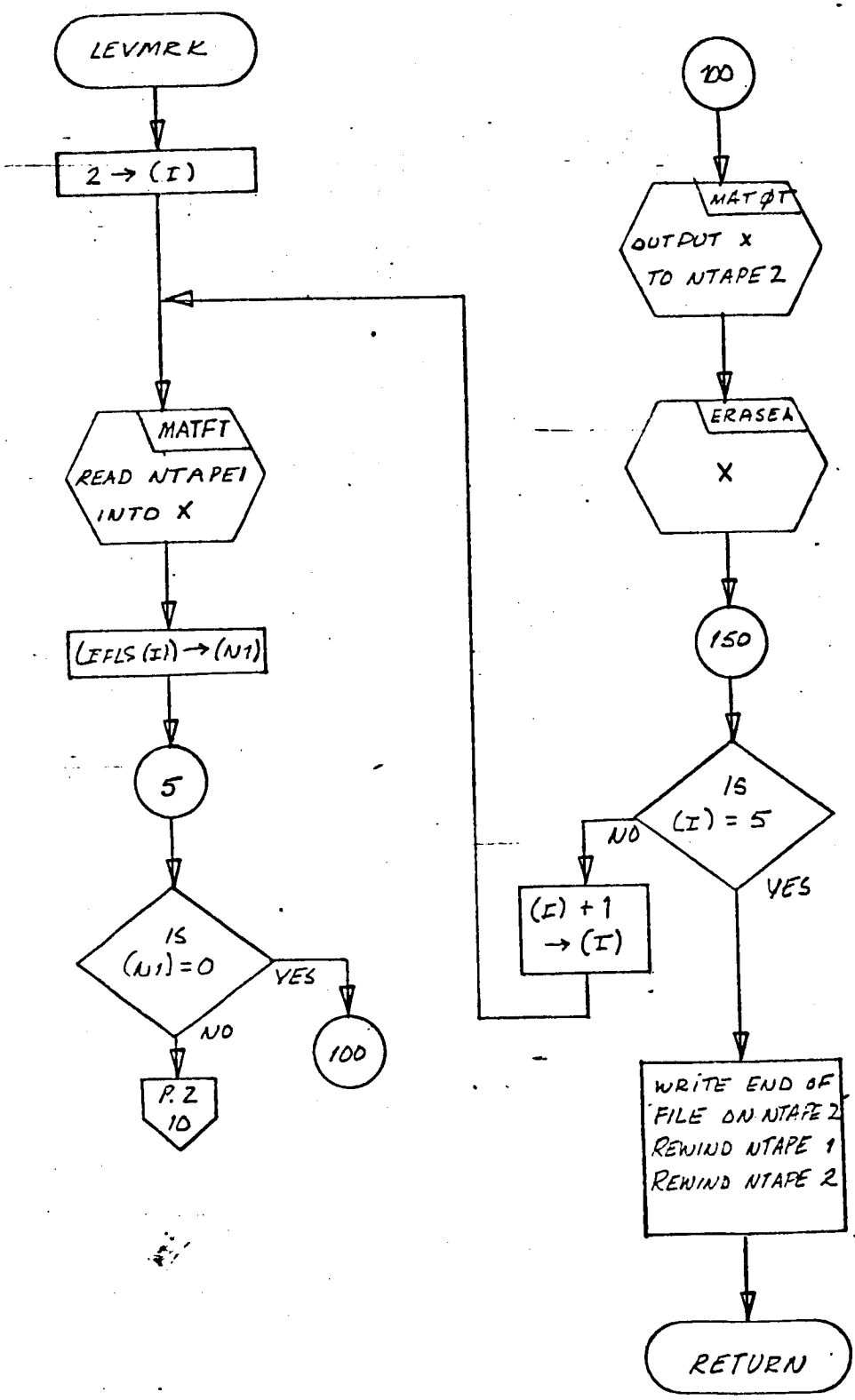
After all four XS matrices have been updated and output, write an EOF on NTAPE2, rewind NTAPE1 and NTAPE2, and exit.

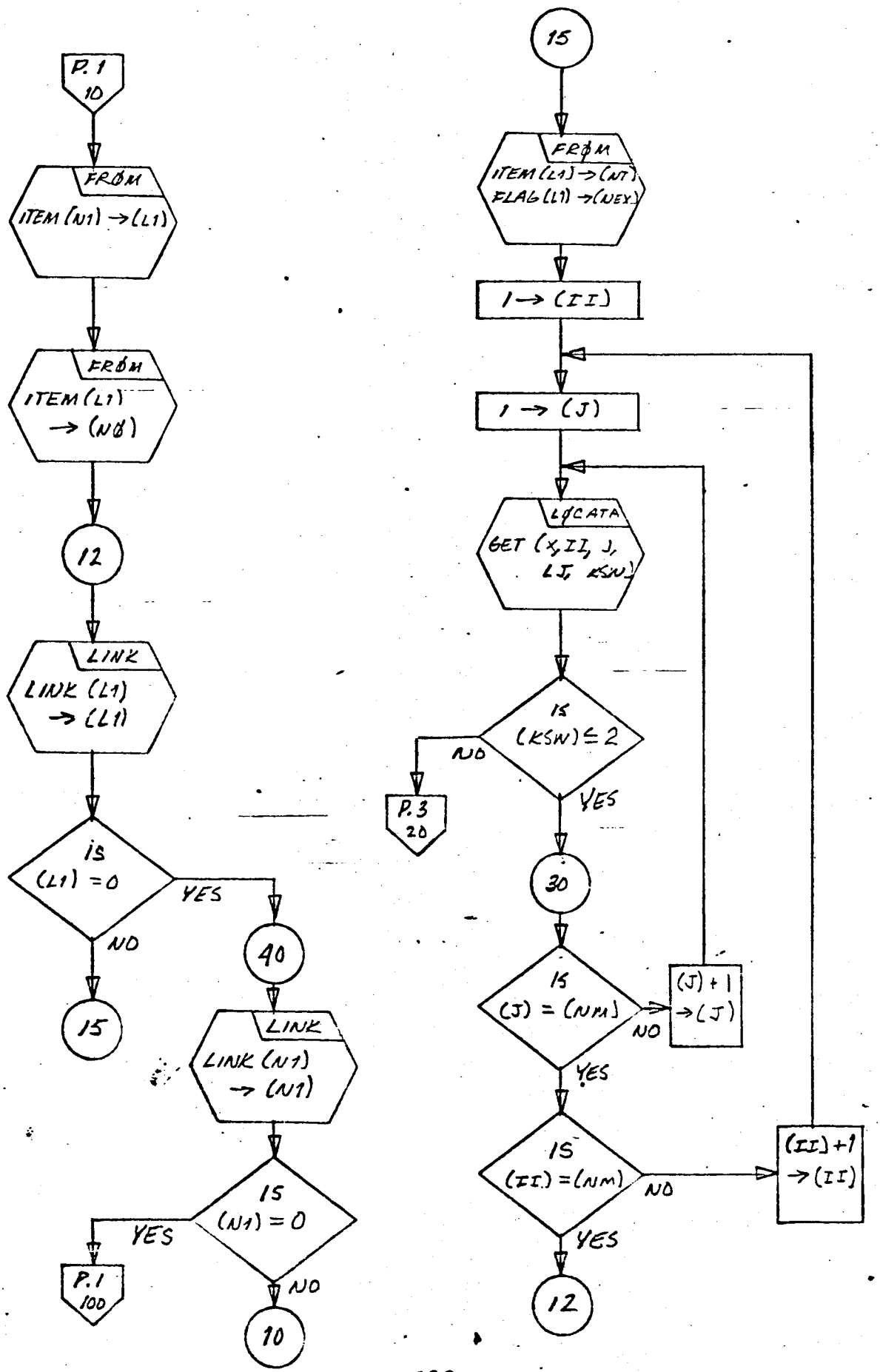
7. Other Subroutines Used

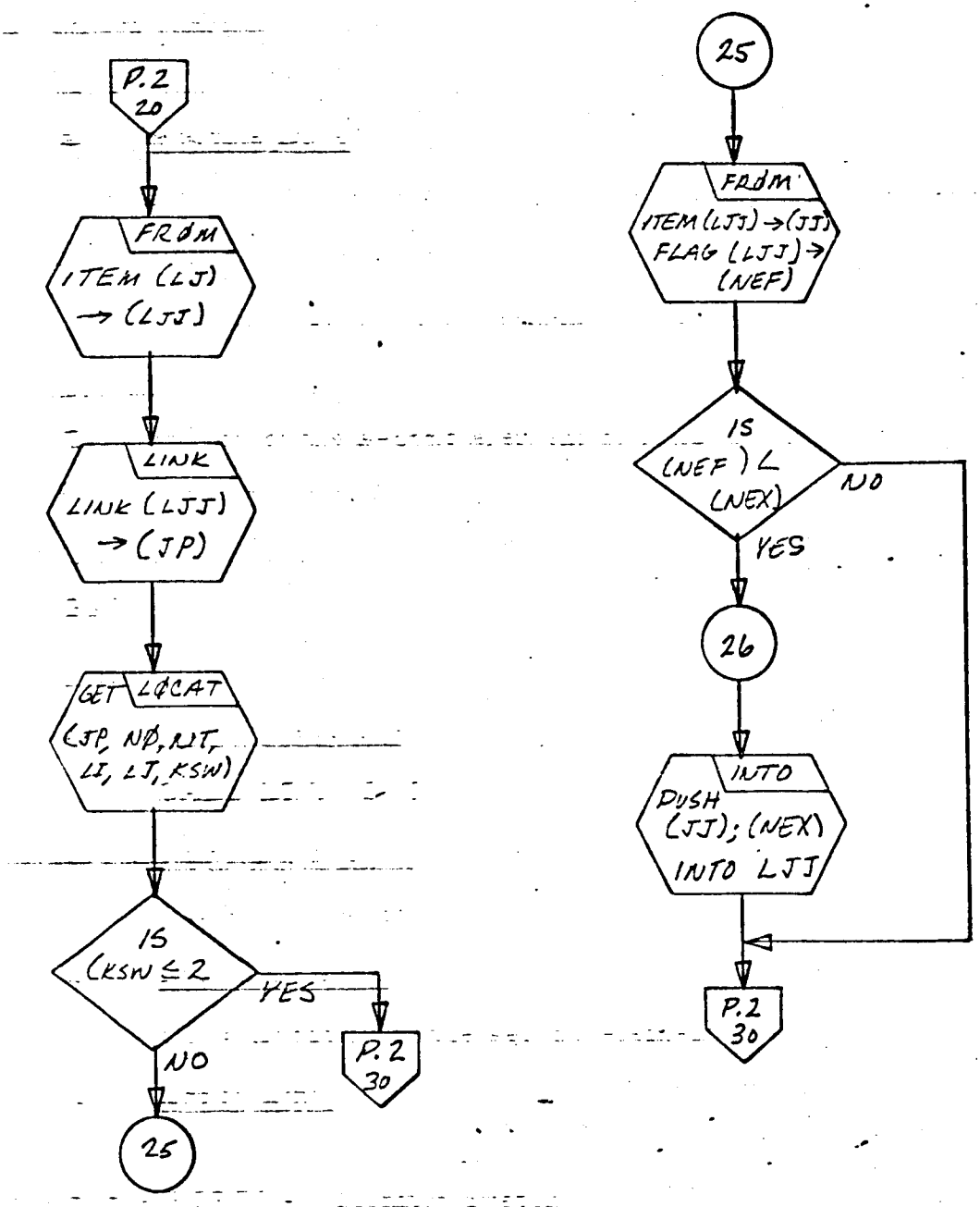
ERASEA, FROM, INTO, LINK, LOCATA, LOCAT, MATFT, MATOT.

8. Using Subroutines

Main program for Pass 2 of TAG Preprocessor.


LEVMRK
2 → (I)
MATFT
READ NTAPE1
INTO X
(IFLS(I)) → (N1)
5
IS
(N1) = 0
YES
NO
100
P. 2
10
100
MATPT
OUTPUT X
TO NTAPE 2
ERASEA
X
150
IS
(I) = 5
NO
YES
(I) + 1
→ (I)
WRITE END OF
FILE ON NTAPE 2
REWIND NTAPE 1
REWIND NTAPE 2
RETURN


P. 1
10
FROM
ITEM (N1) → (L1)
FROM
ITEM (L1)
→ (N8)
12
LINK
LINK (L1)
→ (L1)
IS
(L1) = 0
YES
NO
40
15
LINK
LINK (N1)
→ (N1)
IS
(N1) = 0
YES
NO
P. 1
100
10
15
FROM
ITEM (L1) → (NT)
FLAG (L1) → (NEX)
1 → (II)
1 → (J)
LOCATA
GET (X, II, J,
LI, KSW)
IS
(KSW) ≤ 2
NO
YES
P. 3
20
30
IS
(J) = (NM)
NO
(J) + 1
→ (J)
YES
IS
(II) = (NM)
NO
(II) + 1
→ (II)
YES
12


P.2
20
FROM
ITEM (LJ)
→ (LJJ)
LINK
LINK (LJJ)
→ (JP)
GET LOCAT
(JP, NO, NT,
LI, LJ, KSW)
IS
(KSW ≤ 2
YES
NO
P.2
30
25
25
FROM
ITEM (LJJ) → (JJ)
FLAG (LJJ) →
(NEF)
IS
(NEF) <
(NEX)
NO
YES
26
INTO
PUSH
(JJ); (NEX)
INTO LJJ
P.2
30

Program Description

1. Identification

   a. Routine Label

      LINK

   b. Name

      Get LINK of the first element of a list.

2. Function

The location of the second element of a list is placed in the decrement of a word as an integer.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL LINK (C, I)

   b. Entry Conditions

      C = Head of a list

   c. Exit Conditions

      d(I) = Location of the second element in list C

   d. Error Exits

      If list C is null, CALL DUMP is executed, and return is made to the FORTRAN Monitor System.

5. Definition of Identifiers

   None.

6. Method

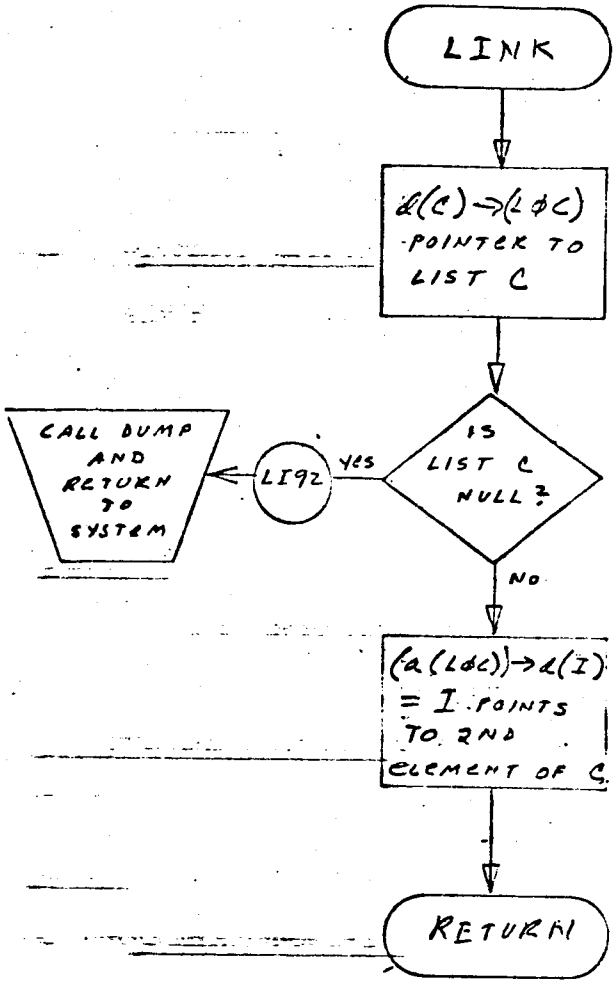
The link (bits 2-35) of the first element in list C is inserted in the decrement portion of I.

7. Other Subroutines Used

None.

8. Using Subroutines

COPY, DONBD, DOWNS, ELIM, ERASE, FISH, INSRT, INVST, LEVMRK, LNKT, LOCAT, LOCATA, LTRACE, MATFT, MATOT, MRKLST, PRPTR, READCH, RECOVR, SEGMNT, SNATCH, STASH, STRIK, SUBST, SYMBL, SYMCRD, SYMTP, Main Program for PASS 2 of TAG Preprocessor.


LINK
CALL LINK (C, I)
@(C) → (L@C)
POINTER TO
LIST C
IS
LIST C
NULL ?
YES
LI92
CALL DUMP
AND
RETURN
TO
SYSTEM
NO
(@(L@C)) → @(I)
= I POINTS
TO 2ND
ELEMENT OF C
RETURN

Program Description

1. Identification

   a. Routine Label

      LNECH

   b. Name

      Place one character of a Hollerith array into the decrement of a word.

2. Function

   The Ith character of an array is placed into a word as an integer (see SYMCH).

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL LNECH (A, I, ILOC)

   b. Entry Conditions

      A = An array of Hollerith characters
      I = Character position in A

   c. Exit Conditions

      ILOC = Ith character of A as an integer

   d. Error Exits

      None.

5. Definition of Identifiers

   LN92 = A table with six entries, each a shift operation.

6. Method

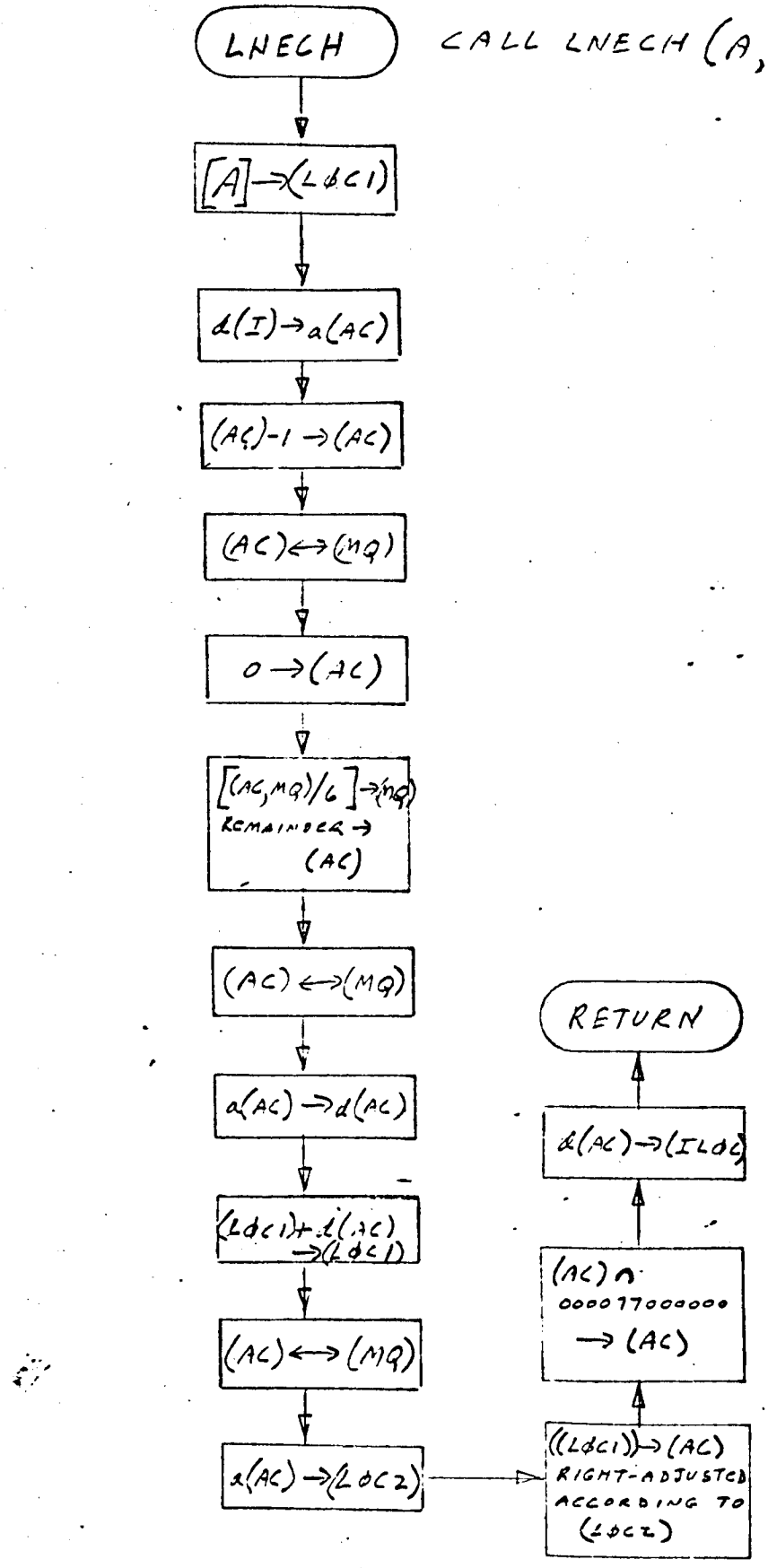
   Each time the subroutine is entered, the character corresponding to the value of I is shifted into the decrement of ILOC as an integer, right-adjusted. A table, located at LN92, determines the appropriate shift.

7. Other Subroutines Used

None.

8. Using Subroutines

GOBLE, MATFT, READCH.


LNECH
CALL LNECH (A, I, ILOC)
[A]→(LOC1)
d(I)→a(AC)
(AC)-1→(AC)
(AC)↔(MQ)
0→(AC)
[(AC,MQ)/6]→(MQ)
REMAINDER→(AC)
(AC)↔(MQ)
a(AC)→d(AC)
(LOC1)+d(AC)→(LOC1)
(AC)↔(MQ)
a(AC)→(LOC2)
((LOC1))→(AC) RIGHT-ADJUSTED ACCORDING TO (LOC2)
(AC)∧ 000077000000 →(AC)
d(AC)→(ILOC)
RETURN

Program Description

1. Identification

   a. Routine Label

      LNKT

   b. Name

      Set a head cell to point to the second element of a list.

2. Function

The location of the second element of a list, if one exists, is placed in a head cell.

3. Usage

   a. Calling Sequence

      CALL LNKT (A, B, F)

   b. Entry Conditions

      A = Head of a list
      B = Head cell
      F = Flag word

   c. Exit Conditions

B points to the second element of A. If A has only one element, B points to this element, and F is also set to point to this element.

   d. Error Exits

      None.

4. Definition of Identifiers

None.

5. Method

The location of the second element in list A is placed in X. If X = 0, A is pointing to its only (or last) element with a terminating link of zero. The pointer in A is saved in B and the flag word F. If

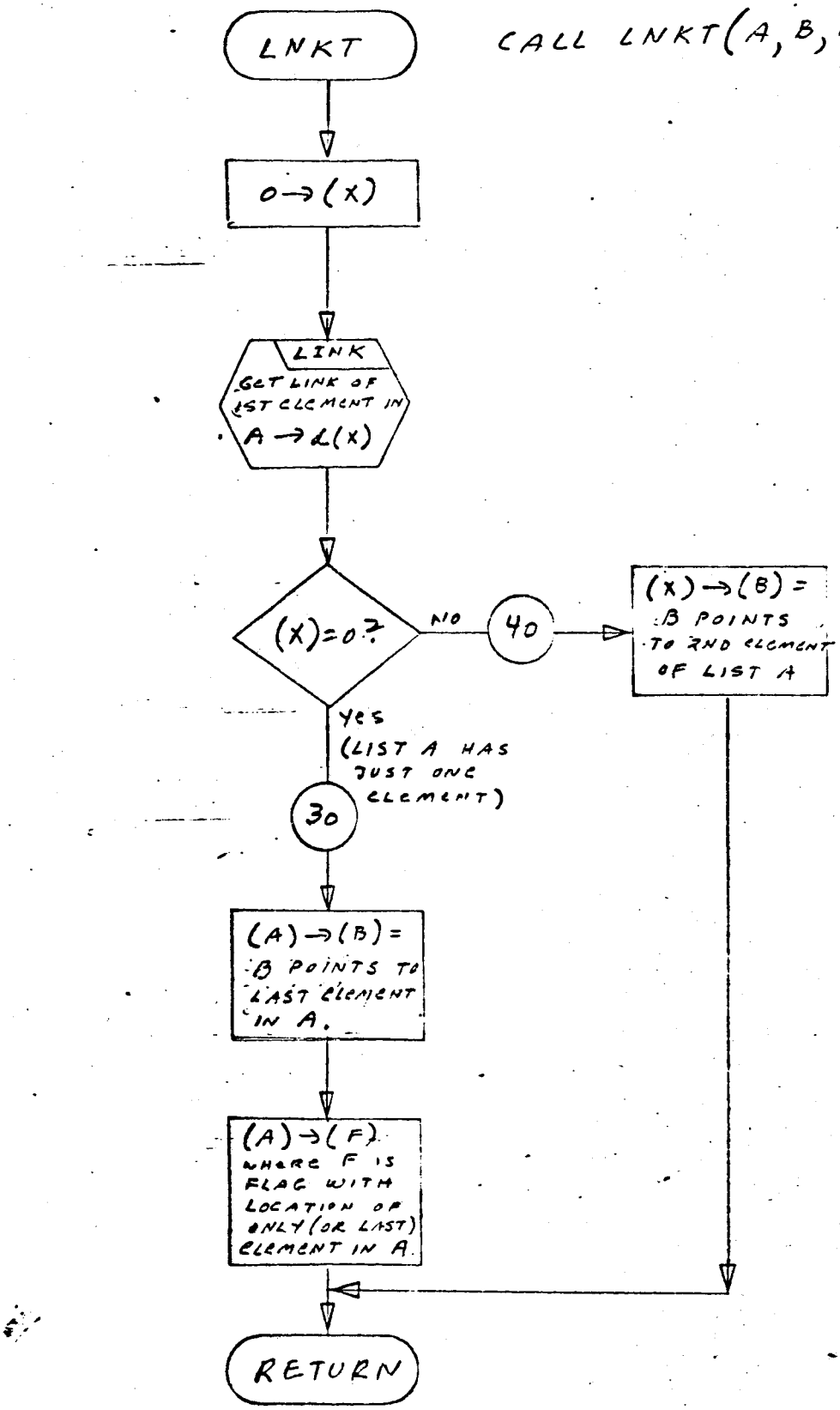
$X \neq 0$, list A has a second element, B is set to point to this element, and F is unchanged.

7. Other Subroutines Used

   LINK

8. Using Subroutines

   AFTLK


LNKT
CALL LNKT(A, B, F)
0 → (X)
LINK
GET LINK OF 1ST ELEMENT IN A → L(X)
(X) = 0 ?
NO
40
(X) → (B) = B POINTS TO 2ND ELEMENT OF LIST A
Yes (LIST A HAS JUST ONE ELEMENT)
30
(A) → (B) = B POINTS TO LAST ELEMENT IN A.
(A) → (F) WHERE F IS FLAG WITH LOCATION OF ONLY (OR LAST) ELEMENT IN A.
RETURN

Program Description

1. Identification

a. Routine Label

LOCAT

b. Name

Scan a list structure representing the four-dimensional array of a matrix.

2. Function

This subroutine scans a list structure for an element whose decrement points to a word containing row index I as its decrement. If found, LI is set to point to this element, which is considered as a sublist. This sublist is then similarly scanned for the column index J, and, if found, LJ is also set.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

Call LOCAT (M, I, J, LI, LJ, KSW)

b. Entry Conditions

M = A four-dimensional matrix represented in list-structure (list type E)

I = Index of the row lists

J = Index of the column lists

c. Exit Conditions

KSW = 1 if neither index, I or J, found.

KSW = 2 if index I found.

KSW = 3 if both indexes I, J were found.

If I is found, LI will contain the location of the element that points to I.

If I is not found, LI = 0.

If J is found, LJ will contain the location of the element that points to J.

If J is not found, LJ = 0.

d. Error Exits

None.

5. Definition of Identifiers

LN = A temporary head cell that is set to point to a sublist in the four-dimensional list headed by M.

LI = A temporary head cell initialized by LN when scanning for I.

LJ = When scanning for J, a temporary head cell initialized by the pointer in LN at the time I was found.

IP = Item ((LI)) for I index, item ((LJ)) for J index.

II = Item ((IP)) for I index.

JJ = Item ((IP)) for J index.

6. Method

a. LI and LJ are cleared to zero.

b. KSW is set = 1.

c. LN is initialized by (M).

d. LN is tested:

(1) If LN = 0, the list structure pointed to by LN is null. Exit is made from the routine with KSW = 1, LI = 0, LJ = 0, indicating that neither I nor J was found.

(2) If LN $\neq$ 0, the list structure pointed to by LN is not null, and the scan proceeds for the sublist that contains I.

e. LI is set to point to the sublist pointed to by LN.

f. (d(LI))→d(IP), then (d(IP))→d(II) gives an element II, which should contain the index I.

g. If (II - I) $\neq$ 0, I has not been found. The scan is repeated at step d. after (link ((LI)) → item (LN)) is executed.

h. If (II - I) = 0, I has been found in M. KSW is set = 2; LI will be pointing to the sublist whose first element contains I.

i. The location of the next element in IP, item ((LI)), is stored in LN to initialize the search for J.

j. LH is again tested:

(1) If LN = 0, the sublist IP pointed to by item ((LI)) is null. Exit is made from the routine with KSW = 2. LI set is described in step h. LJ = 0, for I found but not J.

(2) If LN $\neq$ 0, the sublist IP is not null. The scan proceeds for the sublist which contains J.

(3) LJ is set to point to the sublist pointed to by LN.

(4) (d(LJ)) $\rightarrow$ d(IP), then (d(IP)) $\rightarrow$ d(JJ) gives an element JJ, which should contain the index J.

(5) If (JJ - J) $\neq$ 0, J has not been found. The scan is repeated at step j. after (link ((LJ)) $\rightarrow$ item (LN)) is executed.

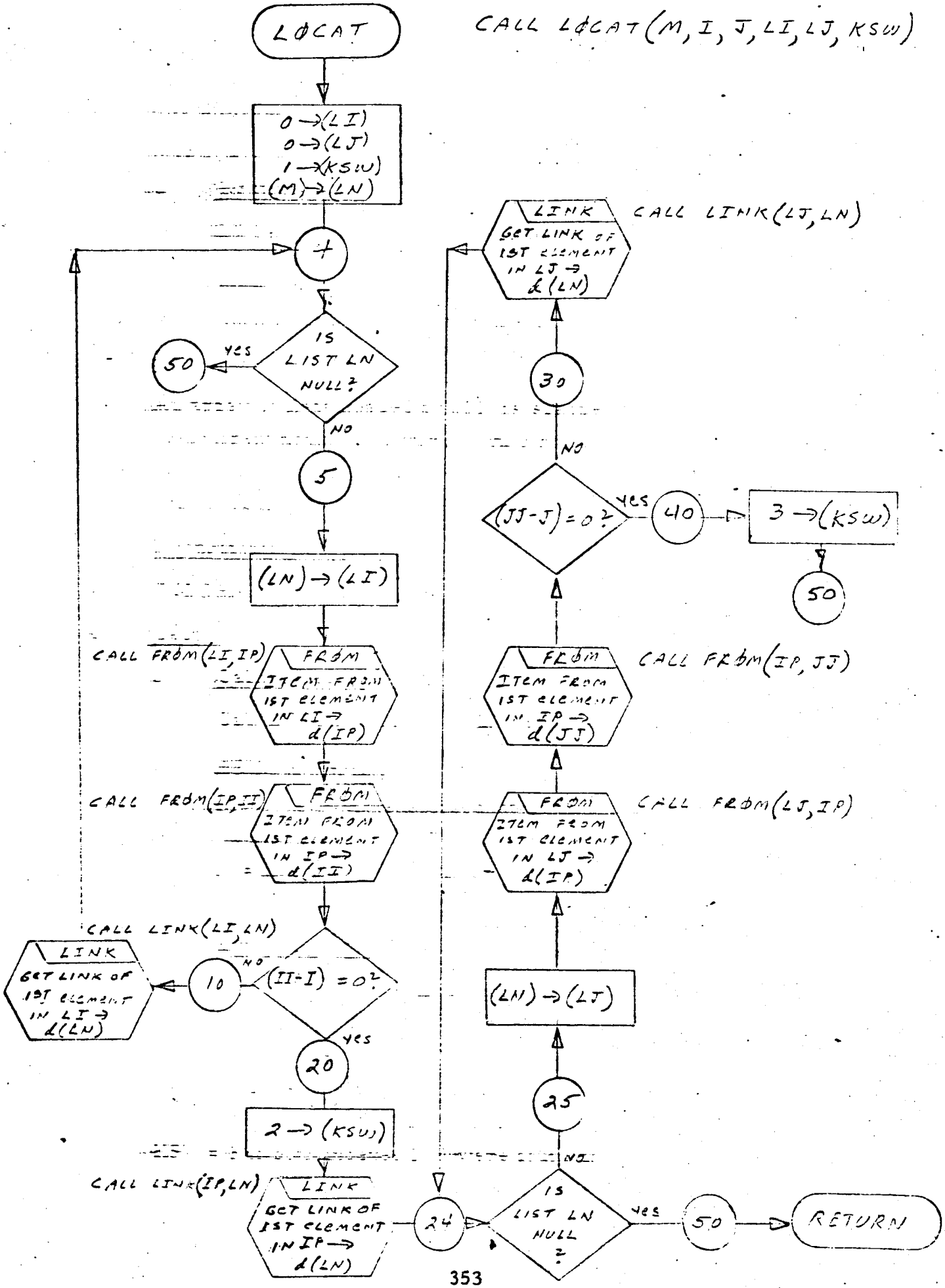
(6) If (JJ - J) = 0, J has been found in M. KSW is set = 3; LJ will be pointing to the sublist whose first element contains J.

7. Other Subroutines Used

FROM, LINK.

8. Using Subroutines

LEVMRK, SNATCH, STASH.


LOCAT
CALL LOCAT(M, I, J, LI, LJ, KSW)
0 → (LI)
0 → (LJ)
1 → (KSW)
(M) → (LN)
IS LIST LN NULL ?
YES
50
NO
5
(LN) → (LI)
CALL FROM(LI, IP)
FROM
ITEM FROM 1ST ELEMENT IN LI → d(IP)
CALL FROM(IP, II)
FROM
ITEM FROM 1ST ELEMENT IN IP → d(II)
CALL LINK(LI, LN)
LINK
GET LINK OF 1ST ELEMENT IN LI → L(LN)
10
NO
(II-I) = 0 ?
YES
20
2 → (KSW)
CALL LINK(IP, LN)
LINK
GET LINK OF 1ST ELEMENT IN IP → L(LN)
24
IS LIST LN NULL ?
YES
50
RETURN
NO
25
(LN) → (LJ)
CALL FROM(LJ, IP)
FROM
ITEM FROM 1ST ELEMENT IN LJ → d(IP)
CALL FROM(IP, JJ)
FROM
ITEM FROM 1ST ELEMENT IN IP → d(JJ)
CALL LINK(LJ, LN)
LINK
GET LINK OF 1ST ELEMENT IN LJ → L(LN)
30
NO
(JJ-J) = 0 ?
YES
40
3 → (KSW)
50

Program Description

1. Identification

a. Routine Label

LOCATA

b. Name

Scan an array of lists for the column index of a matrix.

2. Function

An array of lists headed by M(I) is scanned to locate an element whose decrement points to a word containing the column index J as its decrement. If found, LJ is set to point to this element, which is considered as a sublist.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

Call LOCATA (M, I, J, LJ, KSW)

b. Entry Conditions

M = A four-dimensional matrix represented in array-list form (list type E)

I = Index of the row lists

J = Index of the column lists

c. Exit Conditions

If J is found, LJ will contain the location of the element that points to J.

If J is not found, LJ = 0.

KSW = 1 if neither index found in M(I).

KSW = 2 if I is the only index in M(I).

KSW = 3 if both indexes I, J were found in M(I).

d. Error Exits

None.

5. Definition of Identifiers

M(I) = Ith list structure of the array-list M.
LN = A temporary head cell set to point to M(I).
LJ = A temporary head cell initialized by LN.
IP = Item ((LJ)).
JJ = Item ((IP)).

6. Method

a. LJ is cleared to zero, and KSW is set = 1.

b. If M(I) is null, exit is made from the routine with KSW = 1; LJ = 0.

c. If M(I) is not null,

(1) KSW is set = 2.

(2) LN is initialized by M(I).

d. LN is tested:

(1) If LN = 0, the list pointed to by LH is null. Exit is made from the routine with KSW = 2, LJ = 0, indicating that J could not be found.

(2) If LN $\neq$ 0, M(I) is not null, so the scan proceeds for J:

(a) LJ is set to point to the list pointed to by LN.

(b) (d(LJ)) $\rightarrow$ d(IP), then (d(IP)) $\rightarrow$ d(JJ) gives an element JJ which should contain J.

(c) If (JJ - J) $\neq$ 0, J has not been found. The scan is repeated at step d. after (link ((LJ)) $\rightarrow$ item (LN)) is executed.

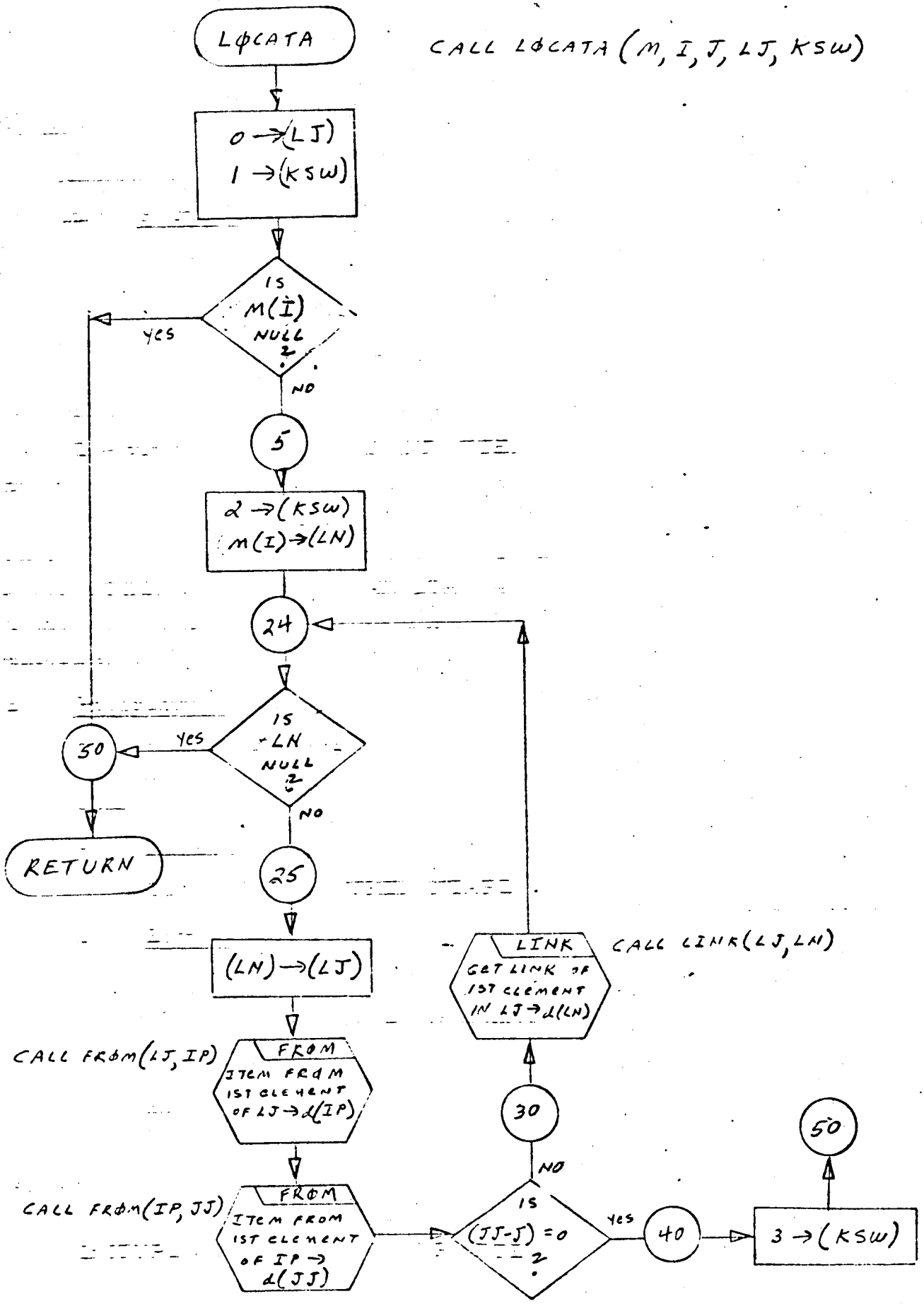
(d) If (JJ - J) = 0, J has been found in M(I). KSW is set = 3, and exit is made from the routine, with LJ pointing to the sublist whose first element contains J.

7. Other Subroutines Used

FROM, LINK.

8. Using Subroutines

FISH, LEVMRK, MATFT, SNATCH, STASH, SYMCRD.


LOCATA
CALL LOCATA (M, I, J, LJ, KSW)
0 → (LJ)
1 → (KSW)
IS M(I) NULL
yes
NO
5
2 → (KSW)
M(I) → (LN)
24
IS LN NULL
yes
30
NO
RETURN
25
(LN) → (LJ)
LINK
GET LINK OF 1ST ELEMENT IN LJ → L(LN)
CALL LINK(LJ, LN)
CALL FROM(LJ, IP)
FROM
ITEM FROM 1ST ELEMENT OF LJ → L(IP)
30
50
CALL FROM(IP, JJ)
FROM
ITEM FROM 1ST ELEMENT OF IP → L(JJ)
IS (JJ-J) = 0
NO
yes
40
3 → (KSW)

Program Description

1. Identification

   a. Routine Label

      MATFT

   b. Name

      Read from tape to reconstruct a matrix in array-list form.

2. Function

The variables ITYPE, I, J, NF, FTEM, NN, NP, and HOLTH are read from NTAPE. The type of element in each record is identified by ITYPE. If ITYPE = 1, 2 then FTEM is "STASHED" in MATRIX by I, J, NN, NP. If ITYPE = 3, HOLTH is "STASHED" instead of FTEM. If ITYPE = 4, HOLTH is stored linked to previous characters. Reading of NTAPE terminates when ITYPE = 5 is encountered.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call MATFT (MATRIX, NTAPE)

   b. Entry Conditions

      NTAPE = An intermediate tape used in Pass 1 whose records contain the elements for reconstructing a matrix in array-list form.

   c. Exit Conditions

The array-list is reconstructed in MATRIX for each type as shown below.

If ITYPE = 1, I, J and FTEM are "STASHED" in MATRIX.

If ITYPE = 2, I, J/NF, NN, NP, FTEM are "STASHED" in MATRIX.

If ITYPE = 3, I, J, HOLTH are "STASHED" in MATRIX.

If ITYPE = 4, successive sets of HOLTH are inserted in MATRIX.

If ITYPE = 5, the last record has been read from NTAPE, and reading terminates.

d. Error Exits

None.

5. Definition of Identifiers

If tag (J) = 6, ITYPE = 1

If tag (J) ≠ 5, ITYPE = 2

If tag (J) = 5, ITYPE = 3 for first 36 characters, and ITYPE = 4 for successive sets of 36.

IBLK = A BCD blank character as an integer.

MATRIX(I) = Ith list structure of array-list MATRIX.

6. Method

The array-list on NTAPE is reconstructed in MATRIX as follows:

a. A blank Hollerith character is stored in IBLK as an integer.

b. Read a record from NTAPE.

c. Test ITYPE.

(1) If ITYPE = 1 or 2, test FTEM.

(a) If FTEM = 0, go to step b.

(b) If FTEM ≠ 0, "STASH" FTEM into MATRIX, identified by I, J, NN, NP.

(i) Test NF. If NF = 0, go to step b to read the next record. If NF ≠ 0, continue.

(ii) Locate J in MATRIX(I). Save flag (J) in NFF.

(iii) Compare NFF with NF.

- If NFF = NF, go to step b to read the next record.
- If NFF ≠ NF, replace the flag in the element containing J with NF, then go to step b to read the next record.

(2) If ITYPE = 3, scan MATRIX (I) for J.

(a) If MATRIX(I) is null, store nonblank characters from HOLTH in a simple list which is linked to MATRIX(I). After 36 characters have been stored, go to step b.

(b) If MATRIX(I) is not null, but J was not found, store nonblank characters from HOLTH in a simple list linked to the J sublist. After 36 characters have been stored, go to step b.

(c) If J has been found in MATRIX(I), replace FTEM identified by I, J, NN, NP with nonblank characters from HOLTH. After 36 characters have been stored, go to step b.

(3) If ITYPE = 4, store nonblank characters from HOLTH in a simple list, linked to previous characters, in MATRIX(I). After 36 characters have been stored, go to step b.

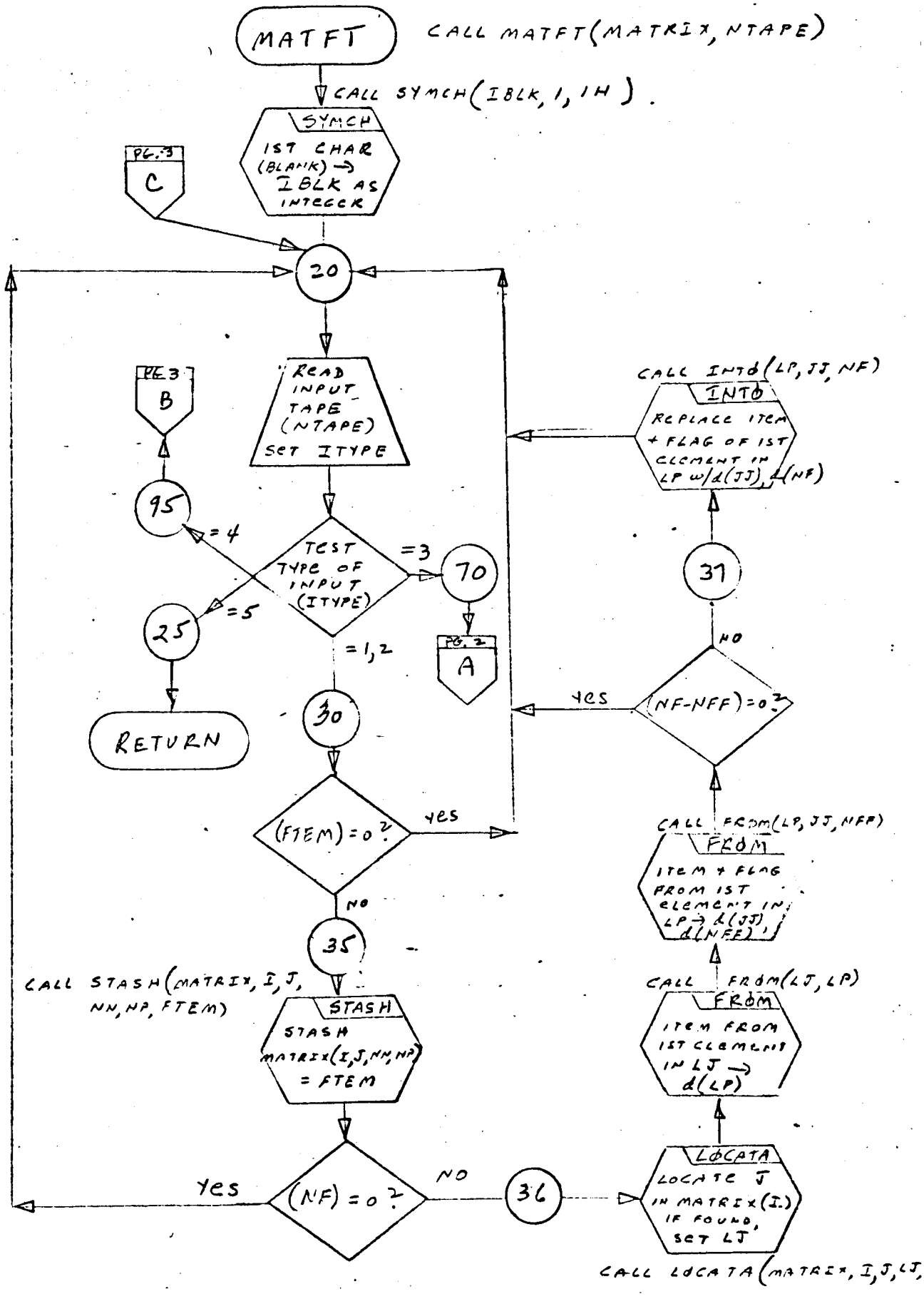
(4) If ITYPE = 5, discontinue reading from NTAPE and exit from the routine.

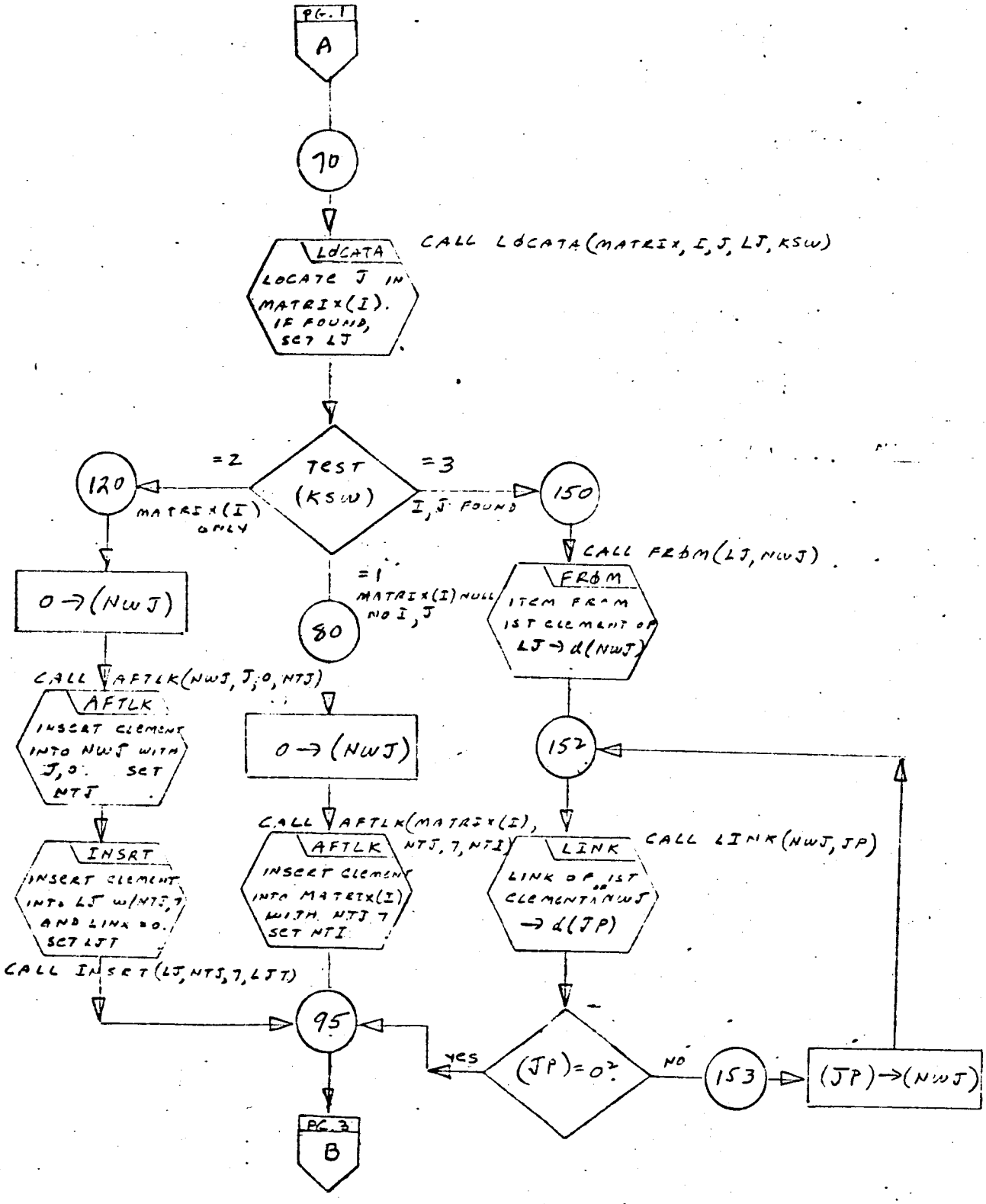
7. Other Subroutines Used

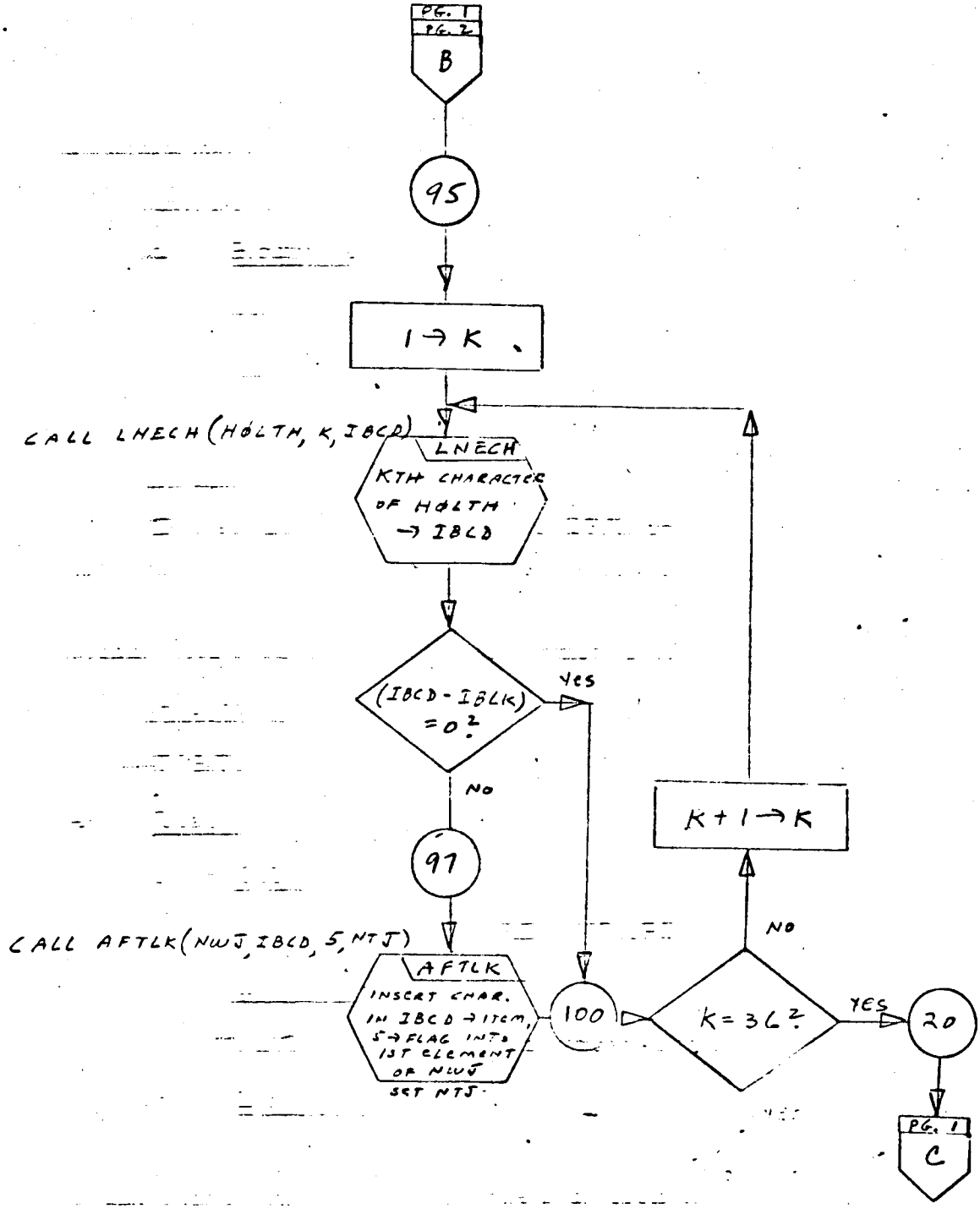
AFTLK, FROM, INSRT, INTO, LINK, LNECH, LOCATA, STASH, SYMCH.

8. Using Subroutines

LEVMRK, SYMCRD, Main Programs for Pass 1 and Pass 2 of TAG Preprocessor.


MATFT
CALL MATFT(MATRIX, NTAPE)
CALL SYMCH(IBLK, 1, IH).
SYMCH
1ST CHAR (BLANK) → IBLK AS INTEGER
PG. 3
C
20
READ INPUT TAPE (NTAPE) SET ITYPE
PG. 3
B
95
TEST TYPE OF INPUT (ITYPE)
=4
=3
=5
=1, 2
70
PG. 2
A
25
RETURN
30
(FTEM) = 0 ?
yes
NO
35
CALL STASH(MATRIX, I, J, NN, NP, FTEM)
STASH
STASH MATRIX(I,J,NN,NP) = FTEM
(NF) = 0 ?
yes
NO
36
CALL LOCATA(MATRIX, I, J, LJ,
LOCATA
LOCATE J IN MATRIX(I.) IF FOUND, SET LJ
CALL FROM(LJ, LP)
FROM
ITEM FROM 1ST CLEMENT IN LJ → d(LP)
CALL FROM(LP, JJ, NFF)
FROM
ITEM + FLAG FROM 1ST ELEMENT IN LP → d(JJ), d(NFF)
(NF-NFF) = 0 ?
yes
NO
37
CALL INTO(LP, JJ, NF)
INTO
REPLACE ITEM + FLAG OF 1ST CLEMENT IN LP w/ d(JJ), d(NF)


PG. 1
A
70
LOCATA
LOCATE J IN MATRIX(I). IF FOUND, SET LJ
CALL LOCATA(MATRIX, I, J, LJ, KSW)
TEST (KSW)
=2
MATRIX(I) ONLY
120
=3
I, J FOUND
150
=1
MATRIX(I) NULL NO I, J
80
0 → (NWJ)
CALL FROM(LJ, NWJ)
FROM
ITEM FROM 1ST ELEMENT OF LJ → d(NWJ)
CALL AFTLK(NWJ, J, 0, NTJ)
AFTLK
INSERT ELEMENT INTO NWJ WITH J, 0. SET NTJ
0 → (NWJ)
152
INSRT
INSERT ELEMENT INTO LJ W/NTJ, J AND LINK = 0. SET LJT
CALL AFTLK(MATRIX(I), NTJ, J, NTI)
AFTLK
INSERT ELEMENT INTO MATRIX(I) WITH NTJ, J SET NTI
LINK
LINK OF 1ST ELEMENT IN NWJ → d(JP)
CALL LINK(NWJ, JP)
CALL INSET(LJ, NTJ, J, LJT)
95
(JP) = 0?
YES
NO
153
(JP) → (NWJ)
PG. 3
B


PG. 1
PG. 2
B
95
1 → K .
CALL LNECH (HØLTH, K, IBCD)
LNECH
KTH CHARACTER OF HØLTH → IBCD
(IBCD - IBLK) = 0 ?
Yes
No
97
CALL AFTLK (NWJ, IBCD, 5, NTJ)
AFTLK
INSERT CHAR. IN IBCD → ITEM, 5 → FLAG INTO 1ST ELEMENT OF NWJ
SET NTJ
100
K = 36 ?
Yes
No
K + 1 → K
20
PG. 1
C

Program Description

1. Identification

   a. Routine Label

      MATOT

   b. Name

      Write an array-list onto tape.

2. Function

The sublists of array-list MATRIX are identified according to list types (a two-dimensional or four-dimensional list containing numerical information, a one-dimensional or two-dimensional list containing Hollerith information) are written onto NTAPE.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call MATOT (MATRIX, NTAPE)

   b. Entry Conditions

      MATRIX = A matrix in array-list form

   c. Exit Conditions

The array-list in MATRIX is written onto NTAPE with a different format for each of the four element types. Each type is identified by ITYPE, which is the first word of each record. The formats for each type are as follows:

ITYPE 1 = 1, I, J, FTEM
ITYPE 2 = 2, I, J, NF, FTEM, NN, NP
ITYPE 3 = 3, I, J, HOLTH
ITYPE 4 = 4, HOLTH
ITYPE 5 = 5, END (written as the terminating record on NTAPE)

d. Error Exits

None.

5. Definition of Identifiers

NTAPE = Local variable that corresponds to a Pass 1 intermediate tapes.
ITYPE = 1, if tag of J = 6
ITYPE = 2, if tag of J ≠ 5
ITYPE = 3, if tag of J = 5 (for first 36 characters)
ITYPE = 4, if tag of J = 5 (for successive sets of 36 characters)
ITYPE = 5, written on NTAPE to signify the end of the array-list
M = MATRIX(I), Ith list structure of array-list MATRIX
M2 = Temporary head cell that points to the next J sublist in M
M3 = Temporary head cell that points to the next NN sublist in M
M4 = Temporary head cell that points to the next NP sublist in M
MMM = Temporary head cell that is changed for each J sublist
I = List structure index for array-list MATRIX
K = BCD character index
IBLK = A BCD blank character as an integer

6. Method

The sublists for each of the element types in MATRIX are extracted and written onto NTAPE as follows:

a. A blank Hollerith character is stored in IBLK as an integer.
b. I + 1 → I sets.
c. Initialize M with MATRIX(I) for the Ith list structure in MATRIX.
d. Clear M2, M3, M4 to zero.
e. Test M. If M is null, I is tested.
   (1) If I ≥ 100, all-array-lists have been extracted and written onto NTAPE. An ITYPE = 5 record is written as the terminating record on NTAPE, and exit is made from the routine.

(2) If I < 100, continue at step b.

f. If M is not null, extract the link of the first element in M and save it in M2 as the pointer to the next sublist in M.

(1) Extract J and its flag, NF. Save link of element holding J, in MMM, as a pointer to the next J sublist.

(2) Test NF.

(a) If NF = 6, extract data word in element pointed to by J and store it in FTEM. Write an ITYPE 1 record on to HTAPE containing 1, I, J, FTEM.

(b) Re-initialize M with the pointer in M2. Continue at step e. above.

(c) If NF ≠ 6, save link of the first element in MMM, in M3, as a pointer to the next NN sublist.

(d) If NF ≠ 5, save pointer to the next NP sublist in M4. Extract NN, NP and FTEM. Write an ITYPE 2 record onto NTAPE containing 2, I, J, NF, FTEM, NN, NP.

(i) If M4 is not null, repeat f(2)(d) for the next NP.

(ii) If M4 is null, save M3 in MMM and test M3.

- If M3 is null, continue at step f(2)(b) for the next J sublist.
- If M3 is not null, continue at step f(2)(c) for the next NN sublist.

(e) If NF = 5, continue at step g.

g. Set ITYPE = 3.

h. Initialize K to 1.

i. Test MMM.

(1) If MMM is null, insert IBLK in Kth position of HOLTH. Continue at next step.

(2) If MMM is not null, extract BCD character from the first element of MMM and place it in the Kth position of HOLTH as an integer.

(3) Test K.

(a) If K ≠ 36, increase K by 1 and continue at step i.

(b) If K = 36, test ITYPE.

(i) If ITYPE = 3, write a record onto NTAPE containing 3, I, J, HOLTH.

- Set ITYPE = 4.
- Test MMM.
- If MMM is not null, continue at step h for the next 36 characters.
- If MMM is null, continue at step f(2)(b) for the next J sublist.

(ii) If ITYPE ≠ 3, write an ITYPE 4 record onto NTAPE containing 4, HOLTH.

(iii) Test MMM.

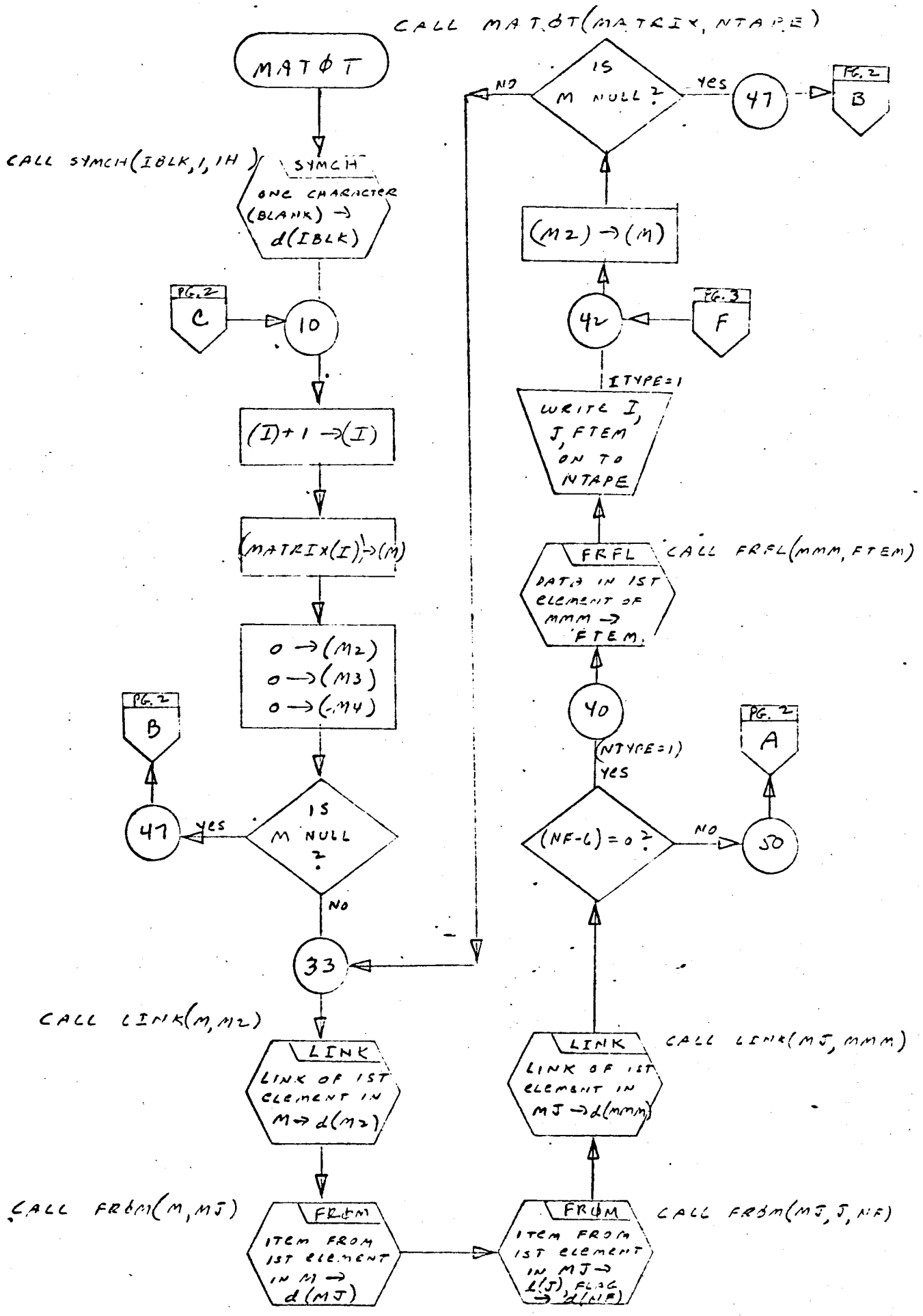
- If MMM is null, continue at step f(2)(b) for the next J sublist.
- If MMM is not null, continue at step h for the next 36 characters.

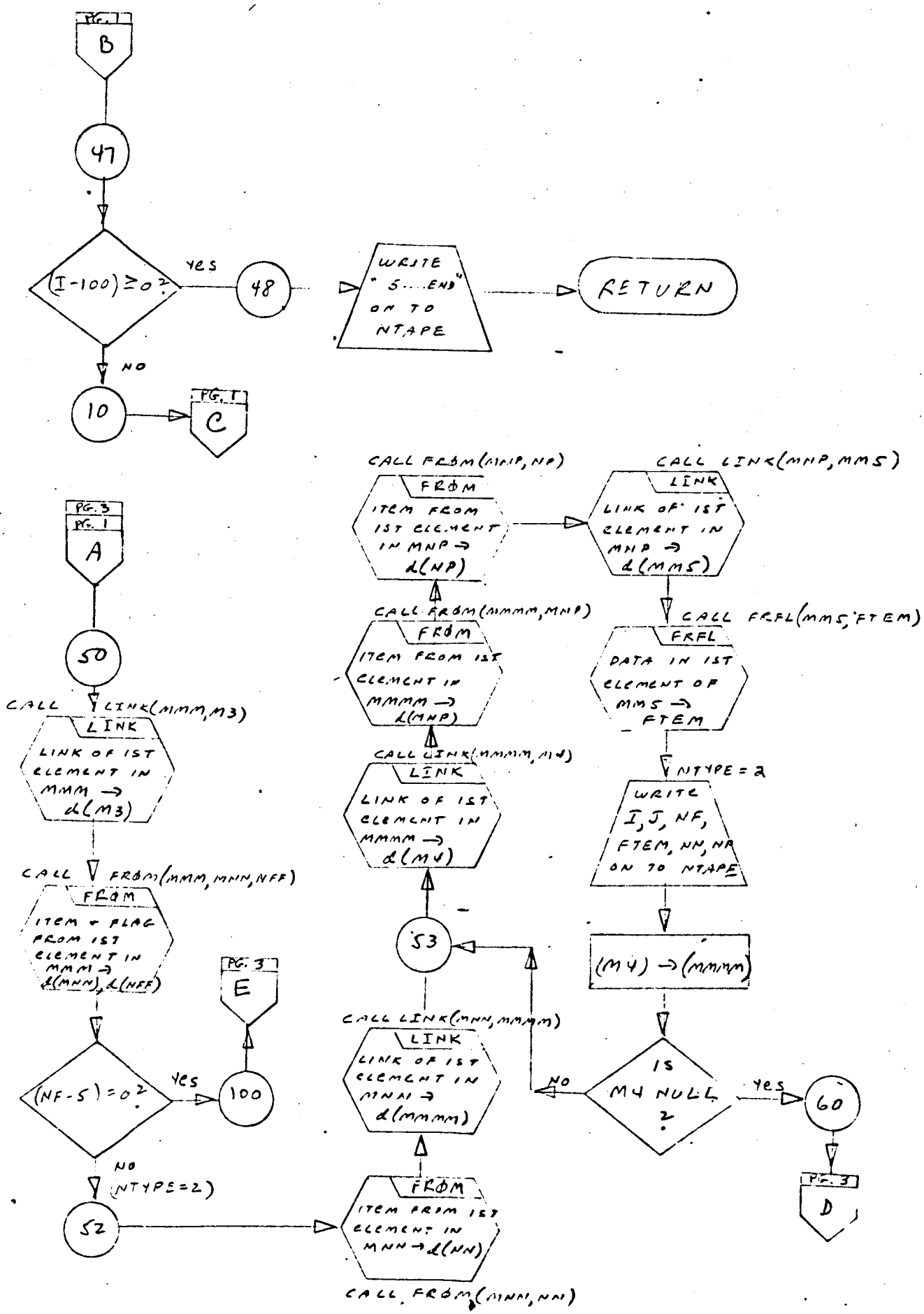
7. Other Subroutines Used

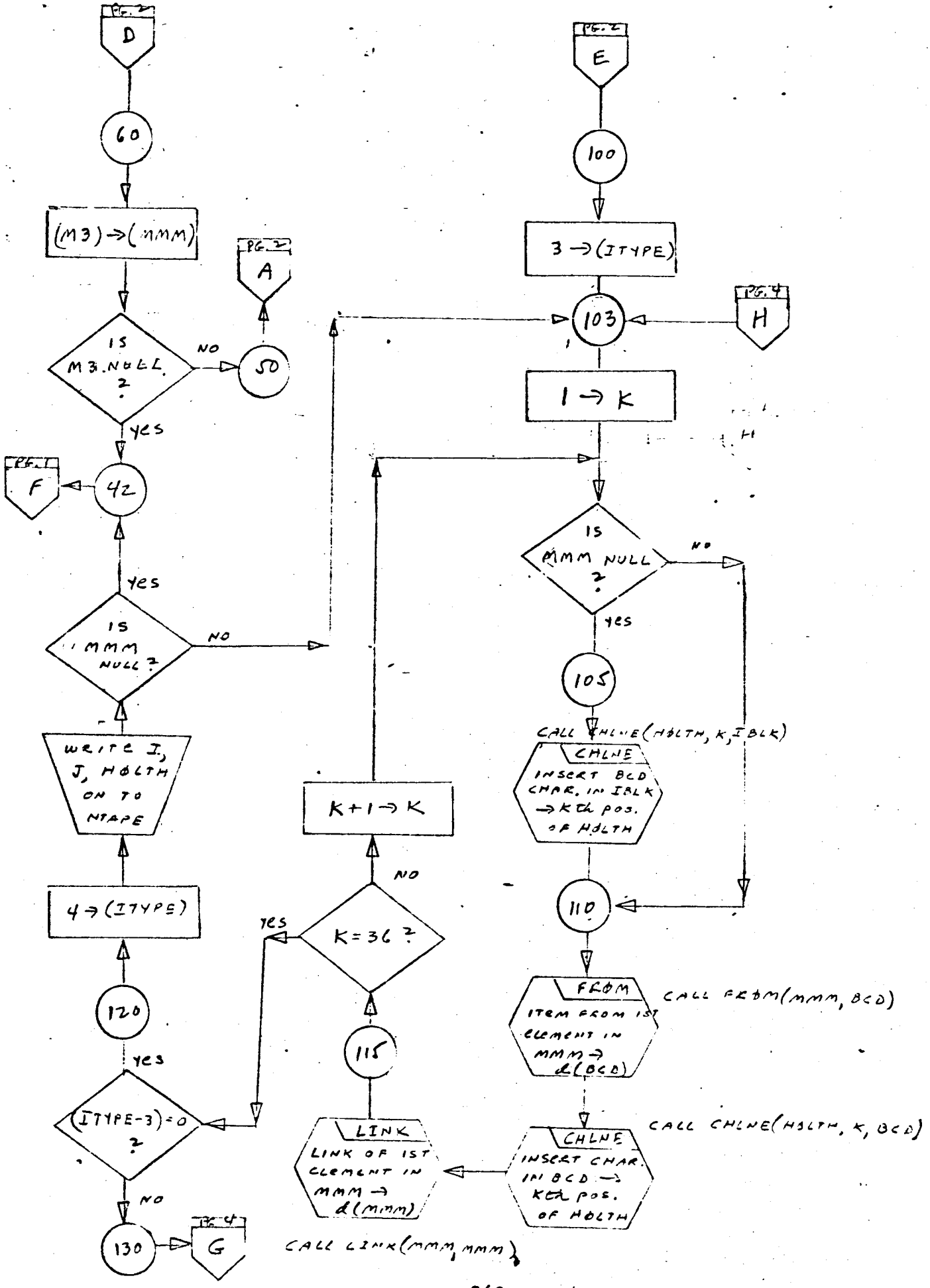
CHLNE, FRFL, FROM, LINK, SYMCH.

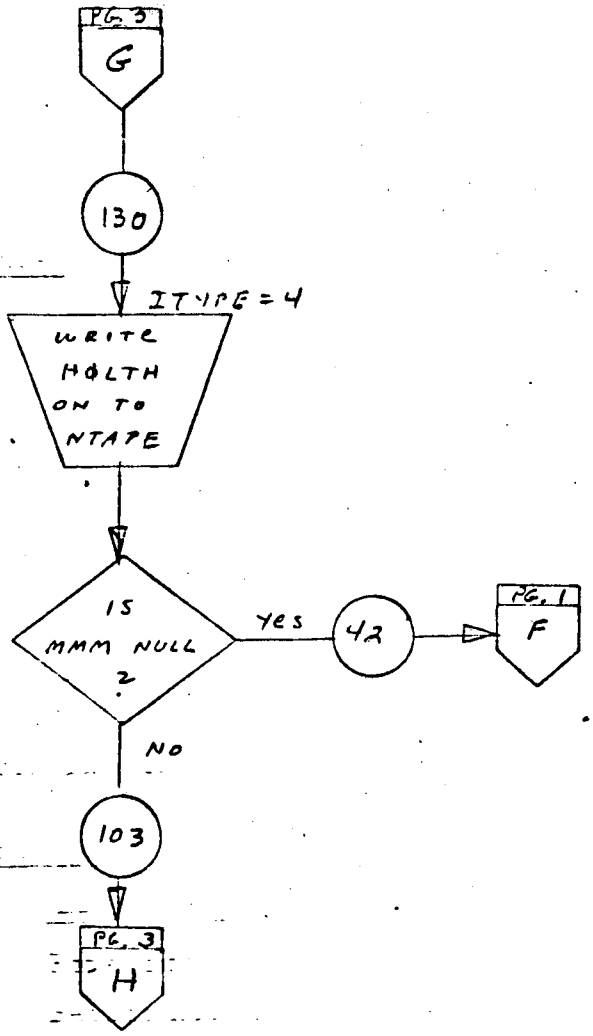
8. Using Subroutines

LEVMRK, PARAM, Main Program for Pass 1 of TAG Preprocessor.


CALL MATOT(MATRIX, NTAPE)
MATOT
CALL SYMCH(IBLK, 1, 1H)
SYMCH
ONE CHARACTER (BLANK) → d(IBLK)
PG. 2
C
10
(I)+1 → (I)
(MATRIX(I)) → (M)
0 → (M2)
0 → (M3)
0 → (M4)
PG. 2
B
47
yes
IS M NULL ?
NO
33
CALL LINK(M, M2)
LINK
LINK OF 1ST ELEMENT IN M → d(M2)
CALL FROM(M, MJ)
FROM
ITEM FROM 1ST ELEMENT IN M → d(MJ)
IS M NULL ?
NO
yes
47
PG. 2
B
(M2) → (M)
42
PG. 3
F
ITYPE=1
WRITE I, J, FTEM ON TO NTAPE
FRFL
DATA IN 1ST ELEMENT OF MMM → FTEM.
CALL FRFL(MMM, FTEM)
40
(NTYPE=1)
yes
(NF-6) = 0 ?
NO
50
PG. 2
A
LINK
LINK OF 1ST ELEMENT IN MJ → d(MMM)
CALL LINK(MJ, MMM)
FROM
ITEM FROM 1ST ELEMENT IN MJ → L(J), FLAG → d(NF)
CALL FROM(MJ, J, NF)


PG. 1
B
47
(I-100) ≥ 0 ?
yes
48
WRITE " 5....END" ON TO NTAPE
RETURN
NO
10
PG. 1
C
PG. 3
PG. 1
A
50
CALL LINK(MMM, M3)
LINK
LINK OF 1ST ELEMENT IN MMM → L(M3)
CALL FROM(MMM, MNN, NFF)
FROM
ITEM + FLAG FROM 1ST ELEMENT IN MMM → L(MNN), L(NFF)
(NF-5) = 0 ?
yes
100
PG. 3
E
NO
(NTYPE=2)
52
CALL FROM(MNNP, NP)
FROM
ITEM FROM 1ST ELEMENT IN MNP → L(NP)
CALL LINK(MNNP, MMS)
LINK
LINK OF 1ST ELEMENT IN MNP → L(MMS)
CALL FROM(MMMM, MNP)
FROM
ITEM FROM 1ST ELEMENT IN MMMM → L(MNP)
CALL FREL(MMS, FTEM)
FREL
DATA IN 1ST ELEMENT OF MMS → FTEM
CALL LINK(MMMM, M4)
LINK
LINK OF 1ST ELEMENT IN MMMM → L(M4)
NTYPE = 2
WRITE I, J, NF, FTEM, NN, NP ON TO NTAPE
53
(M4) → (MMMM)
CALL LINK(MNN, MMMM)
LINK
LINK OF 1ST ELEMENT IN MNN → L(MMMM)
IS M4 NULL ?
NO
yes
60
PG. 3
D
FROM
ITEM FROM 1ST ELEMENT IN MNN → L(NN)
CALL FROM(MNN, NN)


PG. 2
D
60
(M3) → (MMM)
PG. 2
A
IS M3 NULL ?
NO
50
yes
PG. 1
F
42
yes
IS MMM NULL ?
NO
WRITE I, J, HOLTH ON TO NTAPE
4 → (ITYPE)
120
yes
(ITYPE-3) = 0 ?
NO
130
PG. 4
G
K+1 → K
NO
K = 36 ?
yes
115
LINK
LINK OF 1ST ELEMENT IN MMM → d(MMM)
CALL LINK(MMM, MMM)
PG. 2
E
100
3 → (ITYPE)
103
PG. 4
H
1 → K
IS MMM NULL ?
NO
yes
105
CALL CHLNE(HOLTH, K, IBLK)
CHLNE
INSERT BCD CHAR. IN IBLK → Kth POS. OF HOLTH
110
FROM
ITEM FROM 1ST ELEMENT IN MMM → d(BCD)
CALL FROM(MMM, BCD)
CHLNE
INSERT CHAR. IN BCD → Kth POS. OF HOLTH
CALL CHLNE(HOLTH, K, BCD)


PG. 3
G
130
ITYPE = 4
WRITE HOLTH ON TO NTAPE
IS MMM NULL 2
Yes
42
PG. 1
F
No
103
PG. 3
H

Program Description

1. Identification

   a. Routine Label

      MRKLST

2. Function

   To update the flag portion of a node-pair list.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL MRKLST (NE, NO, NT, FLS, KE)

   b. Entry Conditions

NE	Index to FLS array list
NO	First node number
NT	Second node number
FLS	Array list of node-pairs (type list)
KE	Type code (1-8)

   c. Exit Conditions

      FLS list will be updated.

   d. Error Exits

      None.

5. Definition of Identifiers

NEX	=	New flag portion computed from value of KE
L2	=	Sublist FLS(NE)
L3	=	Item (L2)
NO1	=	Item (L3)
L4	=	Link (L3)
NT1	=	Item (14)
NEZ	=	Flag (14)

6. Method

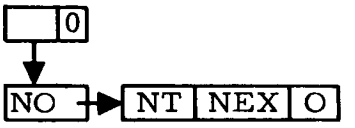
If KE = 1, 2, 3, then exit immediately.

If KE = 4, 5, set NEX = 1.

If KE = 8, set NEX = 2.

If KE = 6, 7, set NEX = 3.

Search L2 for NO1, NT1 such that NO1 = NO and NT1 = NT. If such a pair exists, set NEZ = max (NEZ, NEX) and exit. Else construct a new sublist:



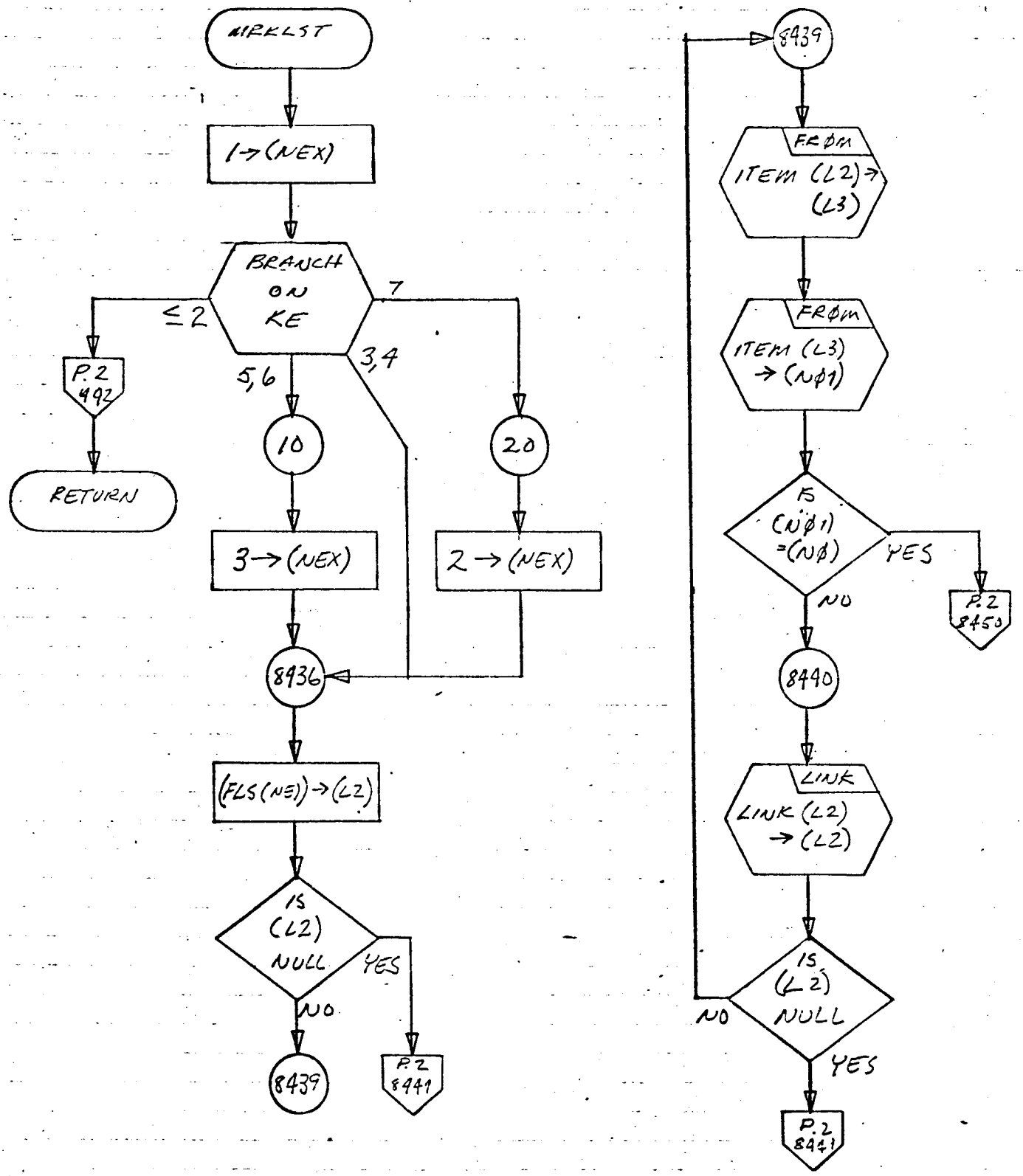
and append it to the end of L2.

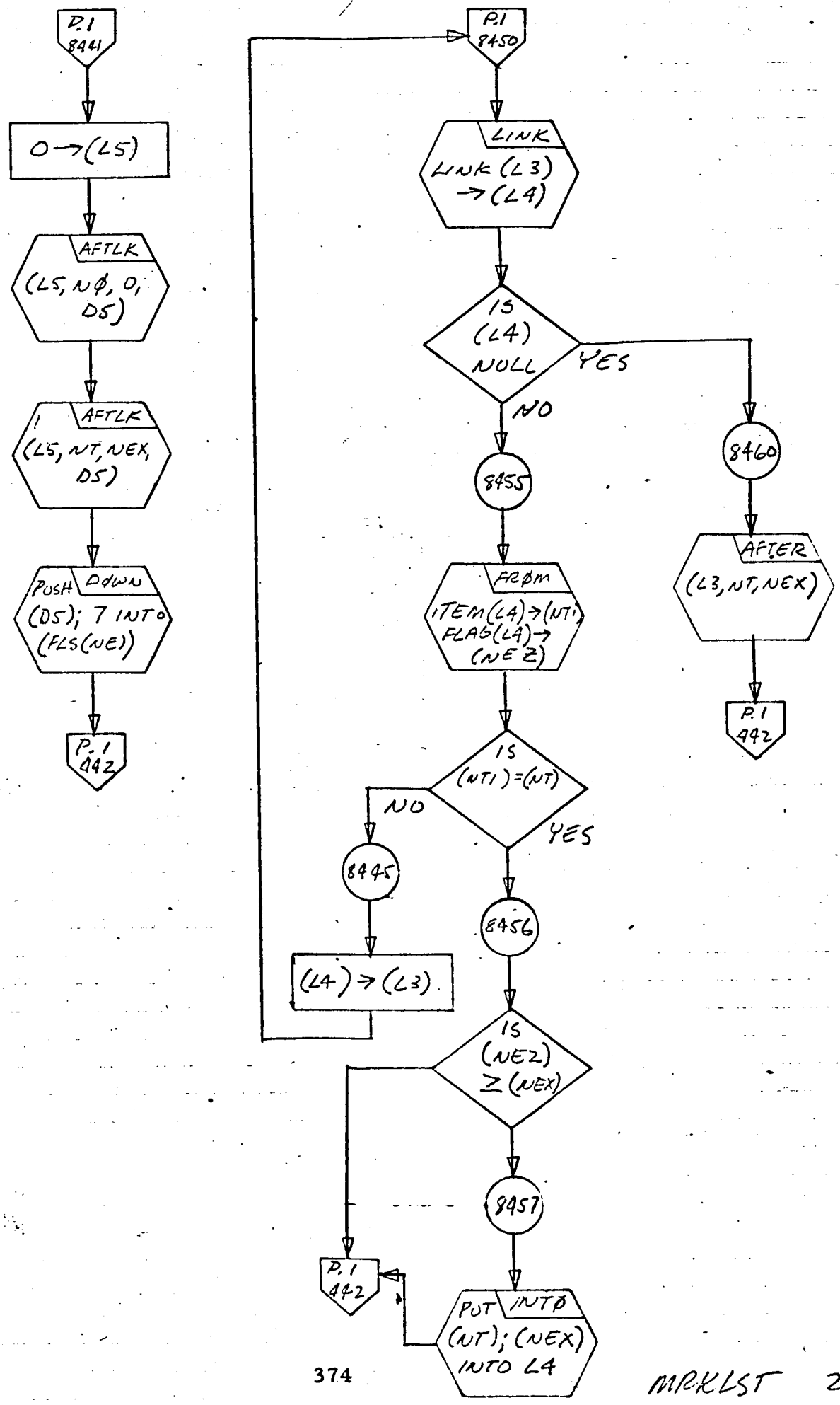
7. Other Subroutines Used

AFTER, AFTLK, DOWN, FROM, INTO, LINK.

8. Using Subroutines

SUBST


MRKLST
1→(NEX)
BRANCH ON KE
≤2
7
5,6
3,4
P.2 492
RETURN
10
20
3→(NEX)
2→(NEX)
8936
(FLS(NE))→(L2)
IS (L2) NULL
YES
NO
8439
P.2 8441
8439
FROM ITEM (L2)→(L3)
FROM ITEM (L3) →(NØ1)
IS (NØ1) =(NØ)
YES
NO
P.2 8450
8440
LINK LINK (L2) →(L2)
IS (L2) NULL
NO
YES
P.2 8441


P.1
8441
0→(L5)
AFTLK
(L5, NØ, 0, DS)
AFTLK
(L5, NT, NEX, DS)
PUSH DOWN
(DS); 7 INTO (FLS(NE))
P.1
442
P.1
8450
LINK
LINK (L3) →(L4)
IS (L4) NULL
YES
NO
8460
AFTER
(L3, NT, NEX)
P.1
442
8455
FRØM
ITEM(L4)→(NT1) FLAG(L4)→(NE2)
IS (NT1)=(NT)
NO
YES
8445
(L4)→(L3)
8456
IS (NE2) ≠ (NEX)
8457
P.1
442
PUT INTØ
(NT); (NEX) INTO L4

MRKLST 2

Program Description

1. Identification

   a. Routine Label

      MULTS

   b. Name

      Multiply two matrices in array list form.

2. Function

MULTS performs a standard matrix multiplication between matrices stored in array list format. Specifically the M x L matrix, TX, is postmultiplied by the L x N matrix, TY, and the result is stored in M x N matrix TZ. If TX and TY are both two dimensional:

$$[TZ](I,J) = \sum_{K=1}^{L} [TX](I,K) * [TY](K,J) .$$

If TX is two dimensional and TY is four dimensional:

$$[TZ](I,J,NN,NP) = \sum_{K=1}^{L} [TX](I,K) * [TY](K,J,NN,NP) .$$

If TX is four dimensional and TY is two dimensional:

$$[TZ](I,J,NN,NP) = \sum_{K=1}^{L} [TX](I,K,NN,NP) * [TY](I,J) .$$

TX and TY may not both be four dimensional.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL MULTS (TX, TY, TZ, M, L, N)

b. Entry Conditions

TX = An M x L matrix represented by either a two- or four-dimensional array list

TY = An L x N matrix represented by either a two- or four-dimensional array list

TX and TY may not both be four-dimensional.

M = The number of rows in TX and TZ

L = The number of columns in TX and rows in TY

N = The number of columns in TY and TZ

c. Exit Conditions

TZ = An M x N matrix product of TX * TY.

TZ is in two-dimensional array list format if both TX and TY are two-dimensional. TZ is in four-dimensional array list format if either TX or TY is two-dimensional.

d. Error Exits

None.

5. Definition of Identifiers

I = The row index of TX and TZ

K = The column index of TX and the row index of TY

J = The column index of TY and TZ

NN = The negative node number of an element descriptor

NP = The positive node number of an element descriptor

AIK = The current coefficient of TX being used to form the IJth product term of TZ

BKJ = The current coefficient of TY being used to form the IJth product term of TZ

TX1 = The list which holds the pointers to the next non-zero coefficient in the NN, NP submatrix [TX](I,J)

TY1 = The list which holds the pointers to the next non-zero coefficient in the NN, NP submatrix [TY](I,J)

PROD = The current product, AIK * BKJ

SUM = The partial summation of

$$\sum_{K=1}^{L} [TX](I,K) * [TY](K,J)$$

for all NN, NP if any

NN1 = The negative node descriptor index for TX

NP1 = The positive node descriptor index for TX

NN2 = The negative node descriptor index for TY

NP2 = The positive node descriptor index for TY

6. Method

A straight matrix multiplication is performed between two two-dimensional coefficient matrices, or between a two-dimensional coefficient matrix and a four-dimensional coefficient matrix in which each element (I, J) is itself a matrix which stores the coefficients (CIJ) of a summation of the form

$$\sum_{NN=1}^{100} \sum_{NP=1}^{100} CIJ(NN,NP) * F(NN,NP)$$

by indices NN and NP. The multiplication algorithm is mechanized twice to conveniently accommodate the cases in which TX is two dimensional and four-dimensional, respectively. When TX is two dimensional and TY is two dimensional, the IJth element of the two-dimensional product matrix, $\lfloor TZ \rfloor(I,J)$, is constructed as

$$\sum_{K=1}^{L} [TX](I,K) * [TY](K,J)$$

for all I from 1 to M and all J from 1 to N. When TX is two dimensional and TY is four-dimensional, the (I, J, NN, NP)th element of the four-dimensional product matrix TZ (I, J, NN, NP) is constructed as

$$\sum_{K=1}^{L} [TX](I,K) * [TY](K,J,NN,NP)$$

for all I from 1 to M, all J from 1 to N, all NN from 1 to 99, and all NP from 1 to 99. When TX is four-dimensional, TY must be two dimensional, and the (I, J, NN, NP)th element of the four-dimensional product matrix TZ(I, J, NN, NP) is constructed as

$$\sum_{K=1}^{L} [TX](I, K, NN, NP) * [TY](K, J)$$

for all I from 1 to M, all J from 1 to N, all NN from 1 to 99, and all NP from 1 to 99.

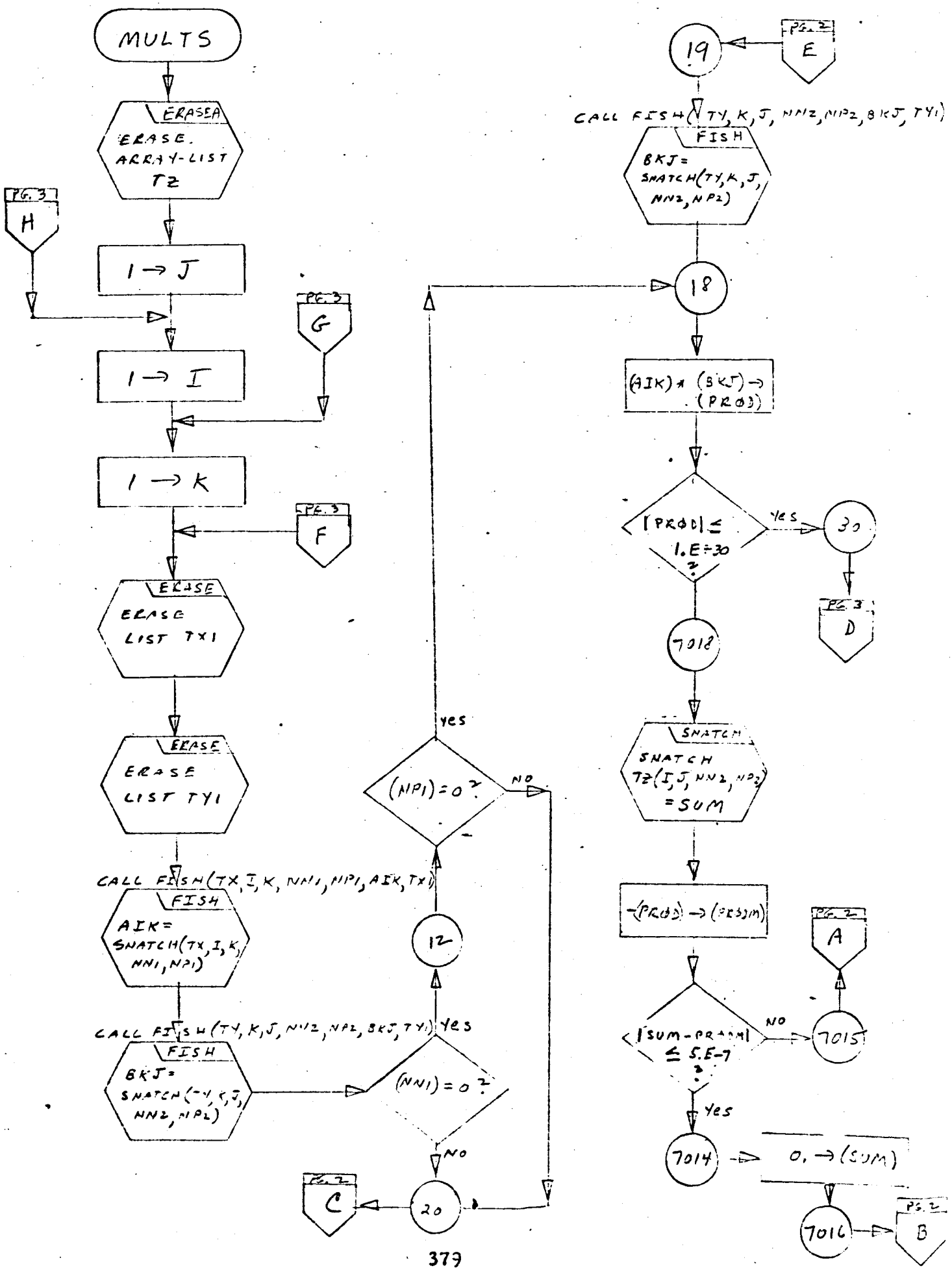
7. Other Subroutines Used

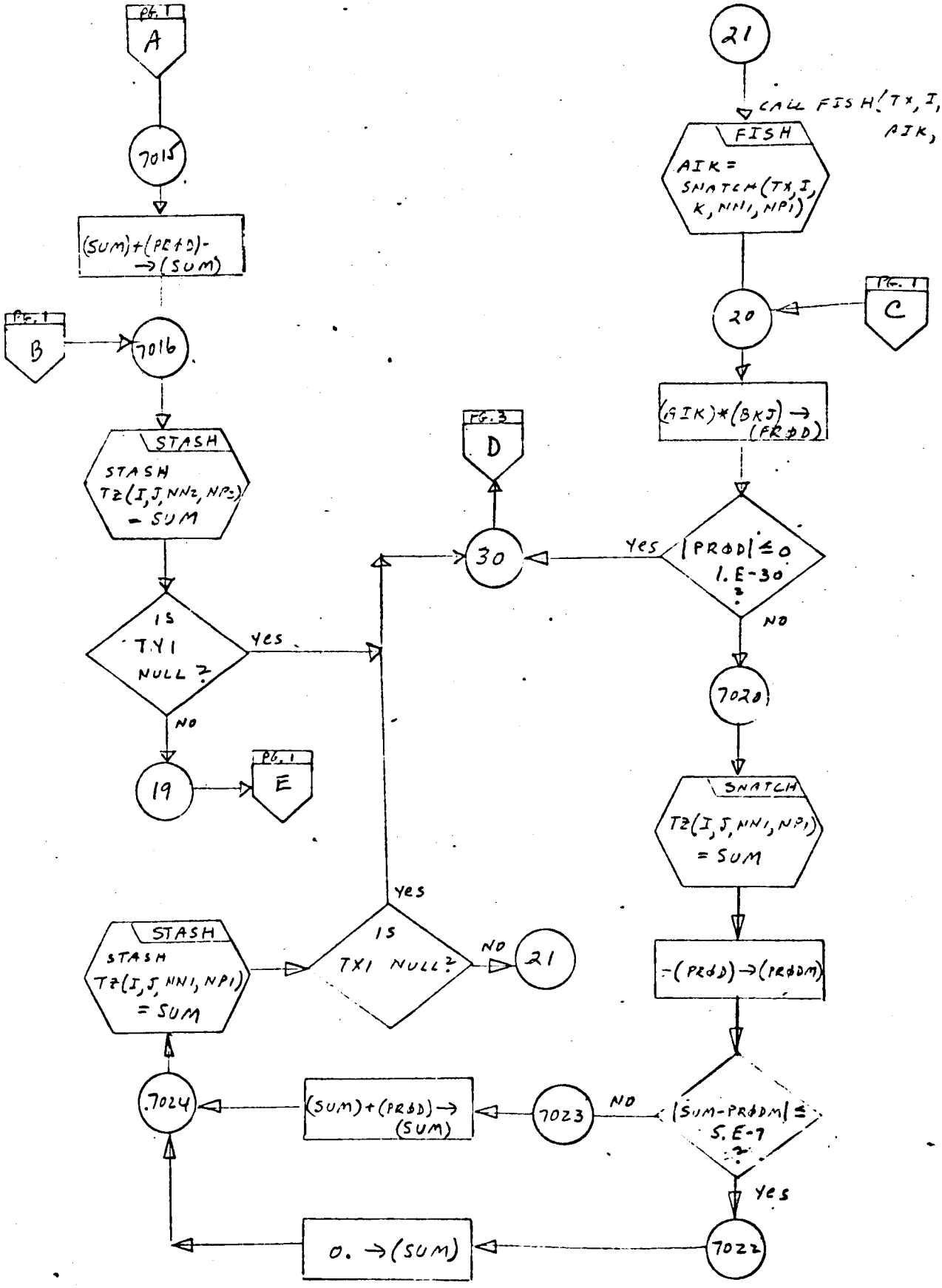
   DIFA, ERASEA, ERASE, FISH, SNATCH, STASH.

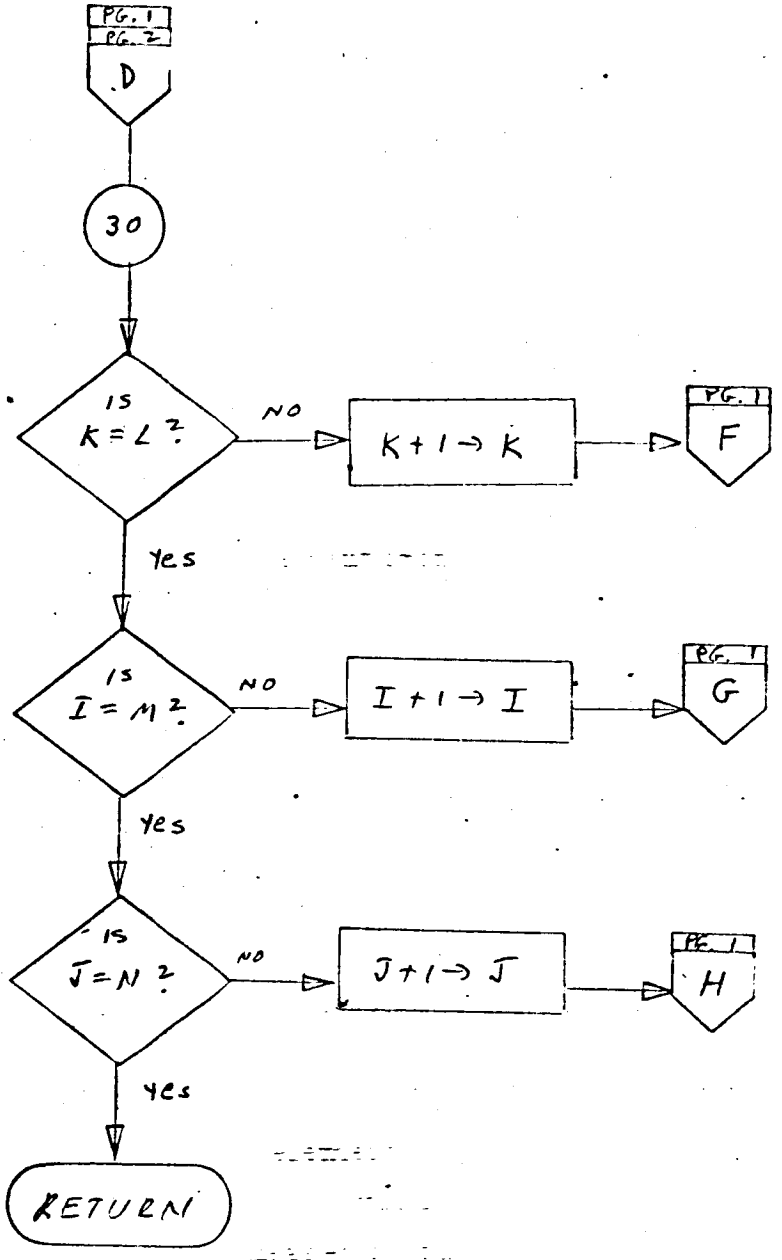
8. Using Subroutines

   Main Program for Pass 1 of TAG Preprocessor.

CALL MULTS(TX, TY, TZ, M, L, N)




Pg. 1
A
7015
(SUM)+(PROD)→(SUM)
Pg. 1
B
7016
STASH
STASH TZ(I,J,NN2,NP2) = SUM
IS TYI NULL ?
yes
NO
19
Pg. 1
E
Pg. 3
D
30
yes
|PROD| ≤ 0 1.E-30
NO
7020
SNATCH
TZ(I,J,NN1,NP1) = SUM
-(PROD)→(PRODM)
|SUM-PRODM| ≤ 5.E-7
yes
7022
0.→(SUM)
NO
7023
(SUM)+(PROD)→(SUM)
7024
STASH
STASH TZ(I,J,NN1,NP1) = SUM
yes
IS TXI NULL ?
NO
21
21
CALL FISH(TX,I,K,NN1,NP1,AIK,TX1)
FISH
AIK = SNATCH(TX,I,K,NN1,NP1)
20
Pg. 1
C
(GIK)*(BKJ)→(PROD)


PG. 1
PG. 2
D
30
IS K = L ?
NO
K + 1 → K
PG. 1
F
yes
IS I = M ?
NO
I + 1 → I
PG. 1
G
yes
IS J = N ?
NO
J + 1 → J
PG. 1
H
yes
RETURN

Program Description

1. Identification

   a. Routine Label

      NEWLOC

   b. Name

      Get a NEW LOCATION from AVS.

2. Function

   This subroutine obtains a new element from available space (AVS).

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL NEWLOC (A).

   b. Entry Conditions

      None.

   c. Exit Conditions

      d(A) = Pointer to new element
      a(AD90) = Pointer to the next available element in AVS
      COUNT is increased by one.

   d. Error Exits

      If either (AD91) or (AD90) = 0, AVS is exhausted and a CALL DUMP is executed. Then return is made to the FORTRAN monitor system.

5. Definition of Identifiers

   AVS = Available space list from which elements for all list structures are taken
   COUNT = Number of elements used from AVS

a(AD90) = Pointer to first element in AVS

a(AD91) = Pointer to last element in AVS

6. Method

The pointer to the first element in AVS is stored in the decrement of A; COUNT is incremented by one, and the pointer to the second element in AVS is placed in a(AD90), thereby becoming the new first element of AVS.

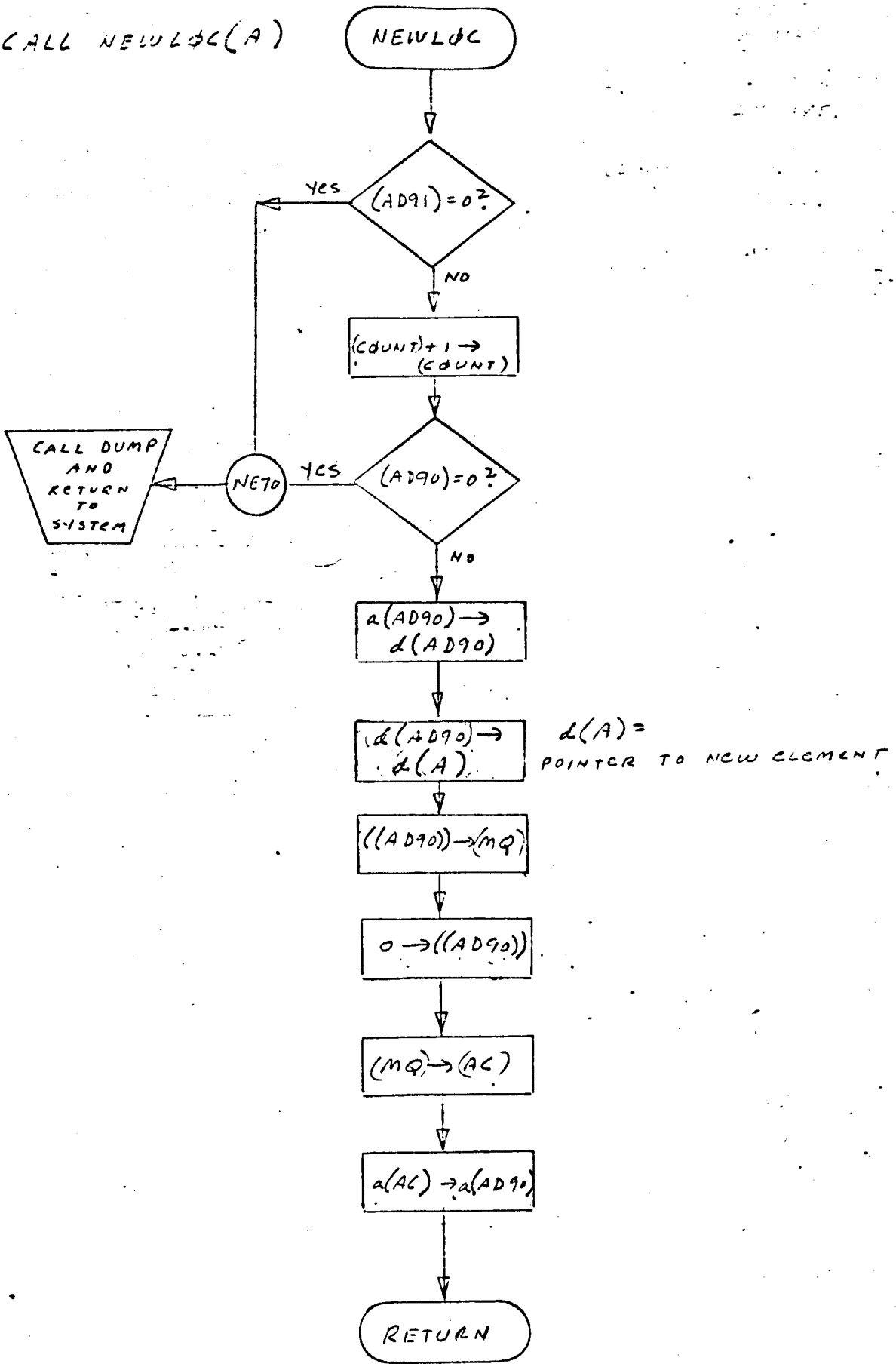
7. Other Subroutines Used

None.

8. Using Subroutines

AFTER, DOWN, INTO, STASH.

CALL NEWLOC(A)



Program Description

1. Identification

   a. Routine Label

      NLINDM

2. Function

   Outputs "DIMENSION FVR· · ·" statement.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL NLINDM (NPT)

   b. Entry Conditions

      NPT is the NPT array of Main 2.

   c. Exit Conditions

      "DIMENSION· · ·" statement output.

   d. Error Exits

      None.

5. Definition of Identifiers

   REST = Used to construct the card image
   REST1 = Used to hold partial contents of REST
   NUM = Used to compute the dimension of the arrays referred to in the DIMENSION statement

6. Method

   Set NUM = NPT(3) + NPT(4)

   If NUM ≤ 0, exit immediately; else convert the contents of NUM to BCD and output this statement:

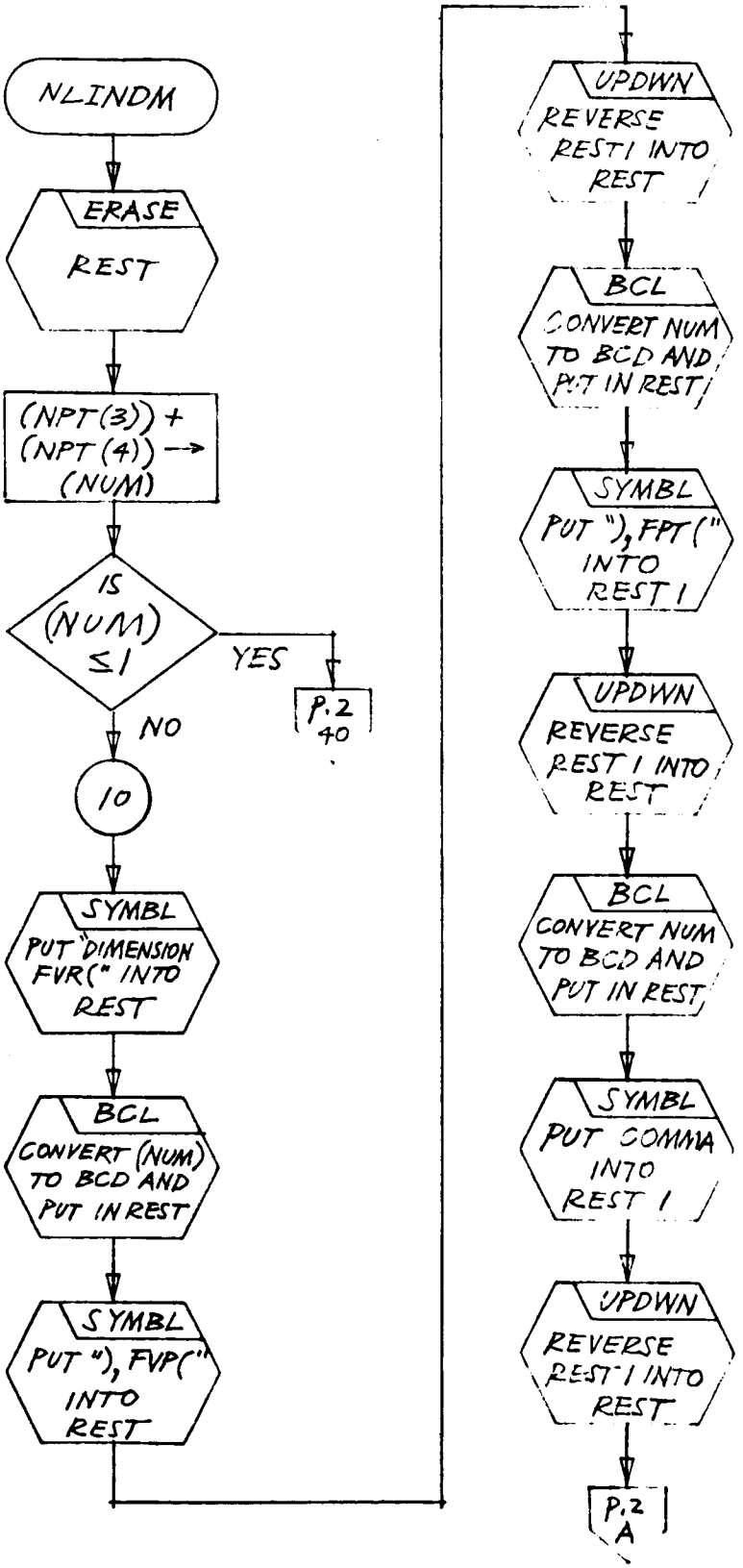
   DIMENSION FVR(num), FVP(num), FPT(num, num)

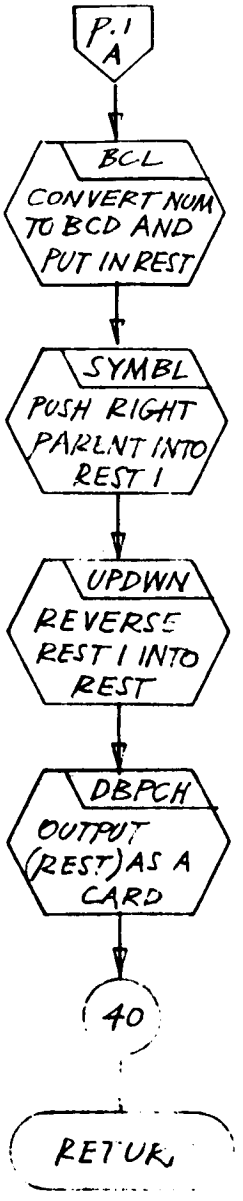
7. Other Subroutines Used

BCL, DBPCH, ERASE, SYMBL, UPDWN.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


NLINDM
ERASE
REST
(NPT(3)) +
(NPT(4)) →
(NUM)
IS
(NUM)
≤1
YES
P.2
40
NO
10
SYMBL
PUT "DIMENSION
FVR(" INTO
REST
BCL
CONVERT (NUM)
TO BCD AND
PUT IN REST
SYMBL
PUT "), FVP("
INTO
REST
UPDWN
REVERSE
REST1 INTO
REST
BCL
CONVERT NUM
TO BCD AND
PUT IN REST
SYMBL
PUT "), FPT("
INTO
REST 1
UPDWN
REVERSE
REST 1 INTO
REST
BCL
CONVERT NUM
TO BCD AND
PUT IN REST
SYMBL
PUT COMMA
INTO
REST 1
UPDWN
REVERSE
REST 1 INTO
REST
P,2
A


P.1
A
BCL
CONVERT NUM TO BCD AND PUT IN REST
SYMBL
PUSH RIGHT PARENT INTO REST 1
UPDWN
REVERSE REST 1 INTO REST
DBPCH
OUTPUT (REST) AS A CARD
40
RETURN

NLINDM

Program Description

1. Identification

   a. Routine Label

      NUMB

   b. Name

      BCD to binary conversion.

2. Function

A BCD number (stored in reverse order in a list) is converted to a binary whole integer, providing all digits representing the number are numeric.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL NUMB (A, K, N, I)

   b. Entry Conditions

      A = Head of a list

      N = The maximum number of elements in list A that represent the BCD number

   c. Exit Conditions

If I = N + 1, no digits were converted (either list A was null on entry to the routine, or the first digit popped up from list A was non-numeric).

Otherwise, I = the number of digits converted and K = the BCD number converted to a binary whole integer.

   d. Error Exits

      None.

5. Definition of Identifiers

ITEM = A temporary cell whose decrement holds one of the digits belonging to the BCD number to be converted.

ITEN = A temporary cell with a conversion factor (integer of 1 or 10) in its decrement.

6. Method

a. If list A is null upon entry to this routine, exit is made with I = N + 1 and K = 0 (no conversion).

b. If the first element popped up from list A has as its item a non-numeric digit, the element is pushed back down into the list and exit is made with I = N + 1 and K = 0 (no conversion).

c. If any other than the first digit in N elements is non-numeric, the element in which it appeared is pushed back down into the list, and exit is made with the conversion in K and I = the number of digits converted before the non-numeric item appeared.

d. If the end of list A is encountered before N elements have been popped up from the list, exit is made with the conversion in K, and I = the number of digits converted before the end of the list was reached.

e. When N elements have been popped up from list A and all of the digits in the elements have been numeric, conversion is considered complete. Exit is made with I = the number of digits converted and K = the BCD number converted to a binary whole integer.

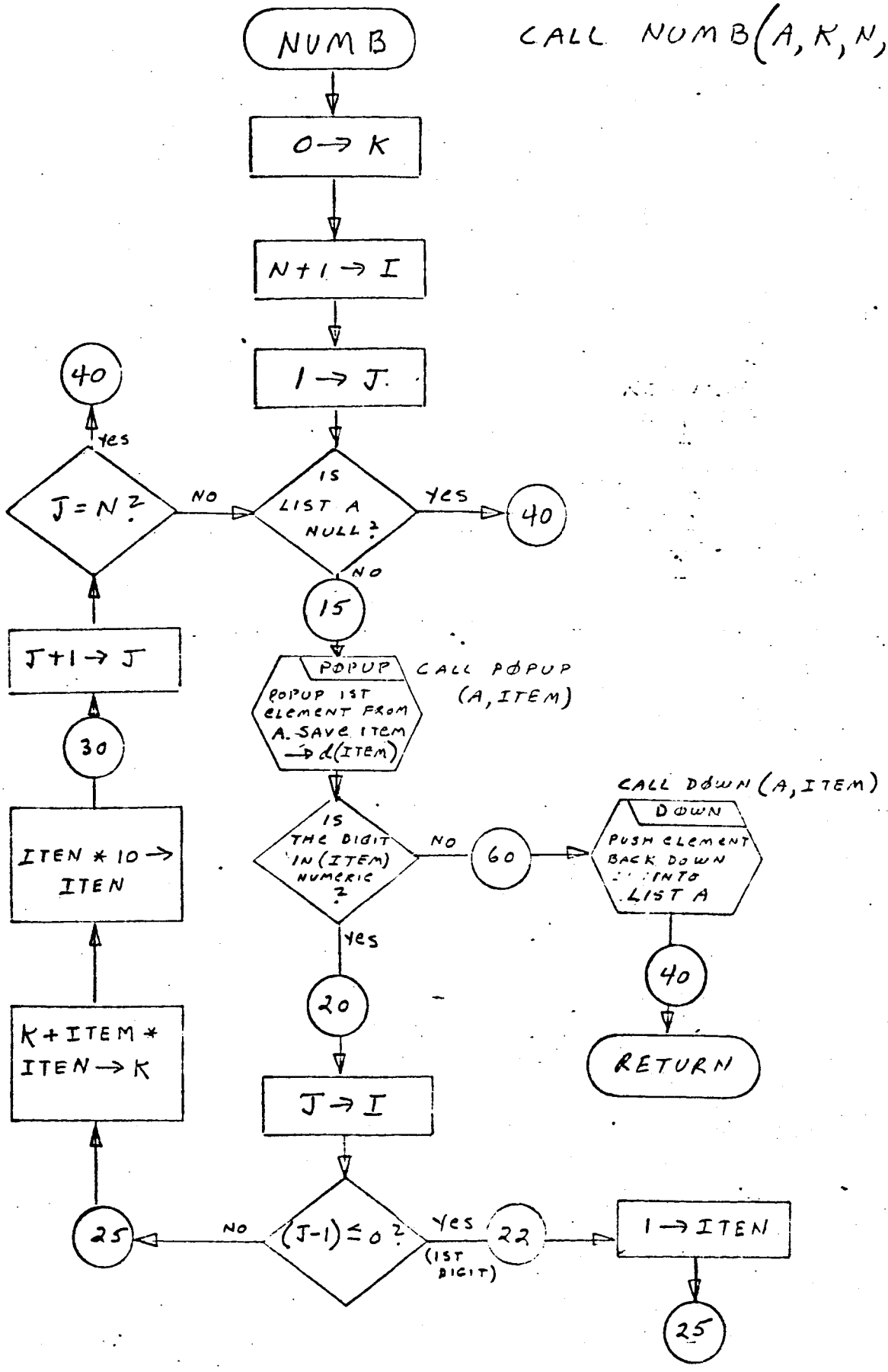
f. Conversion is accomplished as follows:
K + ITEM * ITEN → K, where ITEN = 1 for the first digit and ITEN = ITEN * 10 for each successive digit, until N digits have been obtained.

7. Other Subroutines Used

DOWN, POPUP.

8. Using Subroutines

GOBLE


NUMB
CALL NUMB(A, K, N, I)
0 → K
N+1 → I
1 → J
40
yes
J = N ?
NO
IS LIST A NULL ?
yes
40
NO
15
J+1 → J
POPUP
CALL PØPUP (A, ITEM)
POPUP 1ST ELEMENT FROM A. SAVE ITEM → d(ITEM)
30
ITEN * 10 → ITEN
IS THE DIGIT IN (ITEM) NUMERIC ?
NO
60
CALL DØWN (A, ITEM)
DOWN
PUSH ELEMENT BACK DOWN INTO LIST A
yes
40
K + ITEM * ITEN → K
20
RETURN
J → I
25
NO
(J-1) ≤ 0 ?
yes
(1ST DIGIT)
22
1 → ITEN
25

Program Description

1. Identification

a. Routine Label

PAGEHD

b. Name

Print title at the head of each page.

2. Function

This subroutine prints the title "TRANSIENT ANALYSIS GENERATOR (W.J. THOMAS - JPL)" at the head of each page of FORTRAN code generated by the TAG Preprocessor. Each entry to the routine reduces the line count by one until 56 lines have been printed, at which time a page eject is given and a title is again printed.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL PAGEHD (IRSTC)

b. Entry Conditions

IRSTC = Input argument which controls page eject. The first time the routine is entered, IRSTC = 0.

c. Exit Conditions

If IRSTC = 0 or 1 and IPAGE = 0, the page is ejected, a title is printed, IPAGE is initialized to 56, and IPAGE is reduced by one.

If IRSTC = 1 and IPAGE > 0, IPAGE is reduced by one.

d. Error Exits

None.

5. Definition of Identifiers

IPAGE = Line Count per page

6. Method

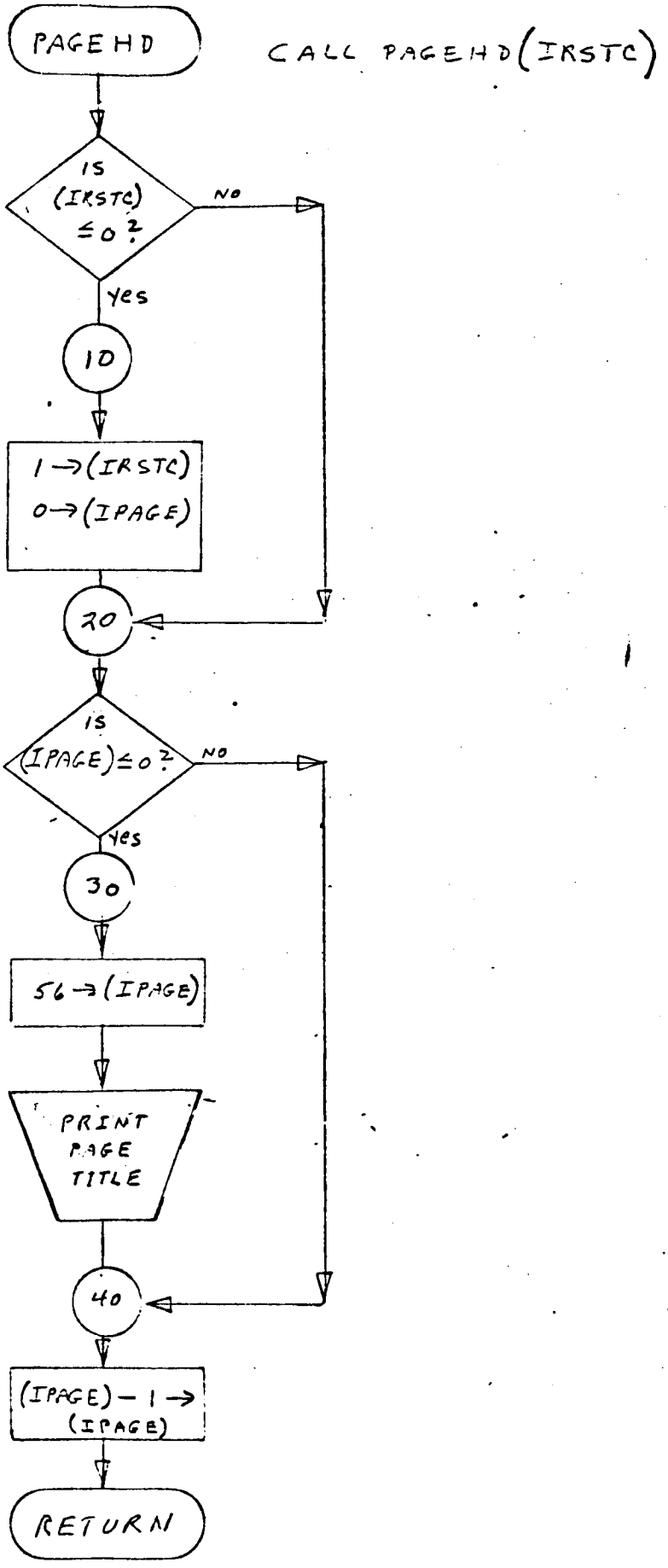
Page eject is controlled by IRSTC and line count by IPAGE. If PAGEHD is being entered for the first time, IRSTC is set = 1, IPAGE = 56, and then the title is printed and the line count in IPAGE is reduced by one. Subsequent entries to the routine reduce the line count by one until 56 lines have been printed, at which point another page eject is given, a title printed, etc., until all the FORTRAN code operated by the TAG program is printed.

7. Other Subroutines Used

None.

8. Using Subroutines

FOUTPT, INPUTX.


PAGEHD
CALL PAGEHD(IRSTC)
IS (IRSTC) ≤ 0 ?
NO
Yes
10
1 → (IRSTC)
0 → (IPAGE)
20
IS (IPAGE) ≤ 0 ?
NO
Yes
30
56 → (IPAGE)
PRINT PAGE TITLE
40
(IPAGE) − 1 → (IPAGE)
RETURN

Program Description

1. Identification

   a. Routine Label

      PARAM

   b. Name

      Form the node basis equations and store them in a four-dimensional list.

2. Function

   This subroutine forms a four-dimensional list representation of the node system current equilibrium equations for each of the element types: capacitors, conductances, reciprocal inductances, and current sources. Each of the four array-lists is constructed in XS and written onto NTAPE.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call PARAM (XS, WLIST, NTAPE)

   b. Entry Conditions

      WLIST = A type D list representation of the TAG connection list

   c. Exit Conditions

      NTAPE has written on it the set of node basic current equilibrium equations pertaining to each element type, in order: capacitors, conductances, reciprocal inductances, and current sources. Each set of equations was formed in XS as a four-dimensional array-list and is therefore stored on NTAPE as ITYPE 2. XS is null on exit from the routine.

d. Error Exits

When a descriptor in WLIST is found with both nodes zero, a comment to that effect is printed, CALL DUMP is executed, and control returns to the FORTRAN monitor system.

5. Definition of Identifiers

I = Element type index.

X1 = Local variable used to search WLIST.

NTAPE = Local variable that corresponds to the PASS 1 intermediate tape NTAPE1.

6. Method

For each type of element, capacitors, conductances, reciprocal inductances, and current sources (these correspond to NE = 2, 3, 4, and 6, respectively), a node current equilibrium equation matrix is established in four-dimensional array-list XS by the following algorithm.

a. Set element type search index I = 2.

b. Test search index. If I = 5 (X former windings), set I = 6 and continue. If $I \neq 5$, continue.

c. Erase XS and set X1 to point to the top of WLIST.

d. Search WLIST starting at X1 for the first descriptor whose element type index, NE, matches search index I. When found, extract NN and NP and set X1 to point to next WLIST descriptor.

e. If the element type index $NE \neq 6$, the matrix XS being formed is not for current sources, and plus or minus ones are stashed in XS according to the following table. For WLIST descriptor variables NE, NN, NP:

Conditions on NN and NP	Locations in XS			
	NN, NN, NN, NP	NN, NP, NN, NP	NP, NN, NN, NP	NP, NP, NN, NP
NP = 0, NN ≠ 0	0	0	0	+1
NP ≠ 0, NN = 0	+1	0	0	0
NP ≠ 0, NN ≠ 0	+1	-1	-1	+1
NP = 0, NN = 0	Call Dump and Return to System			

If the element type index NE = 6, the matrix being formed is for current sources, and plus or minus ones are stacked in XS according to the following table:

Conditions on NN and NP	Locations in XS	
	NN, 1, NN, NP	NP, 1, NN, NP
NP = 0, NN ≠ 0	-1	0
NP ≠ 0, NN = 0	0	+1
NP ≠ 0, NN ≠ 0	-1	+1
NP = 0, NN = 0	No entries are made in XS.	

f. Test to see if entire WLIST has been searched for element Type I.

(1) If the search is incomplete return to step d.

(2) If the search is complete, write XS for element Type I onto NTAPE and erase XS. Test to see whether I is less than 6.

(a) If I is less than 6, add one to I and return to step b to form XS for next element type.

(b) If I is equal to or greater than 6, then all four XS matrices have been formed. An EOF is written on NTAPE; it is rewound, and return is made to the main program.

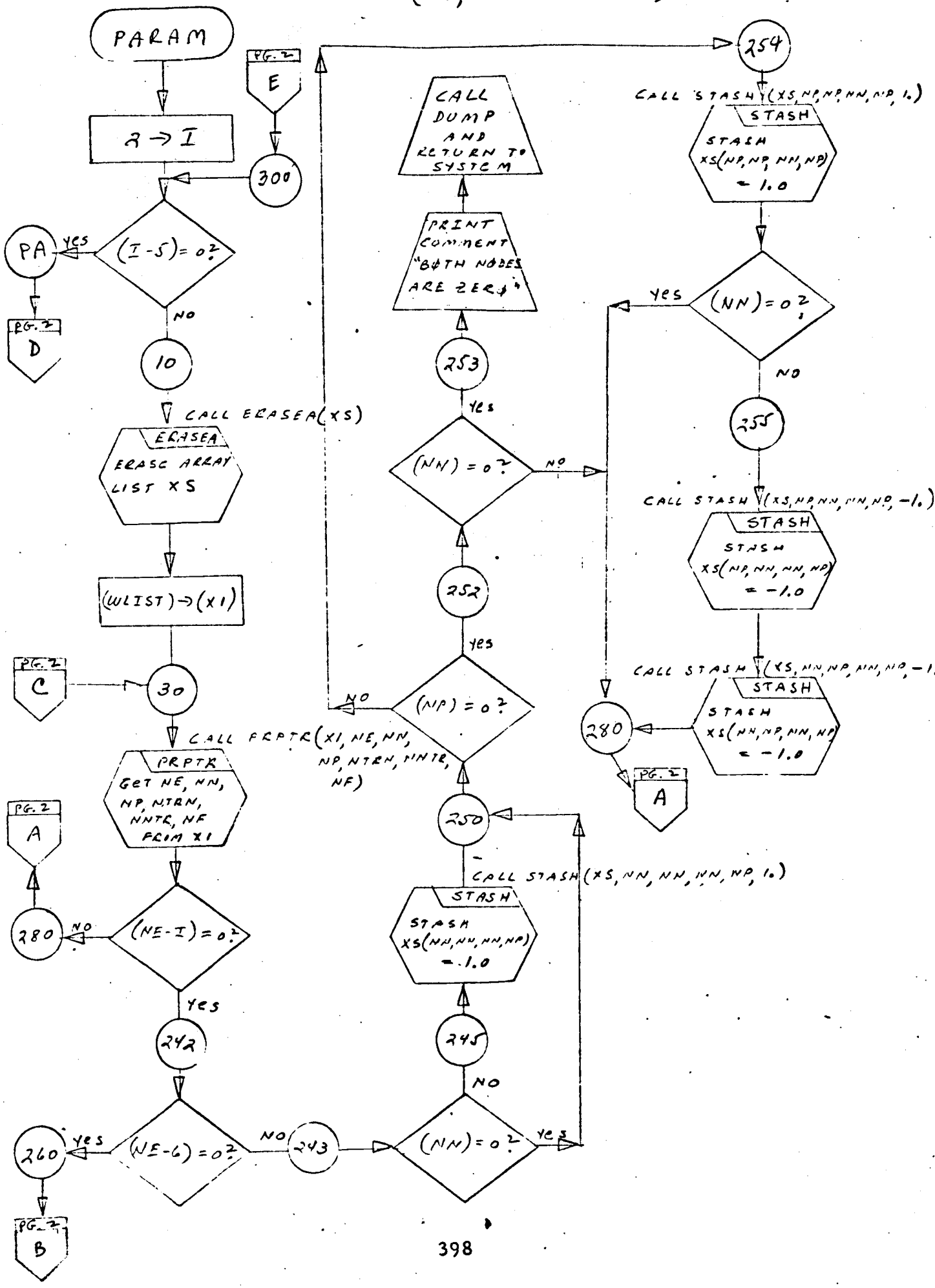
7. Other Subroutines Used

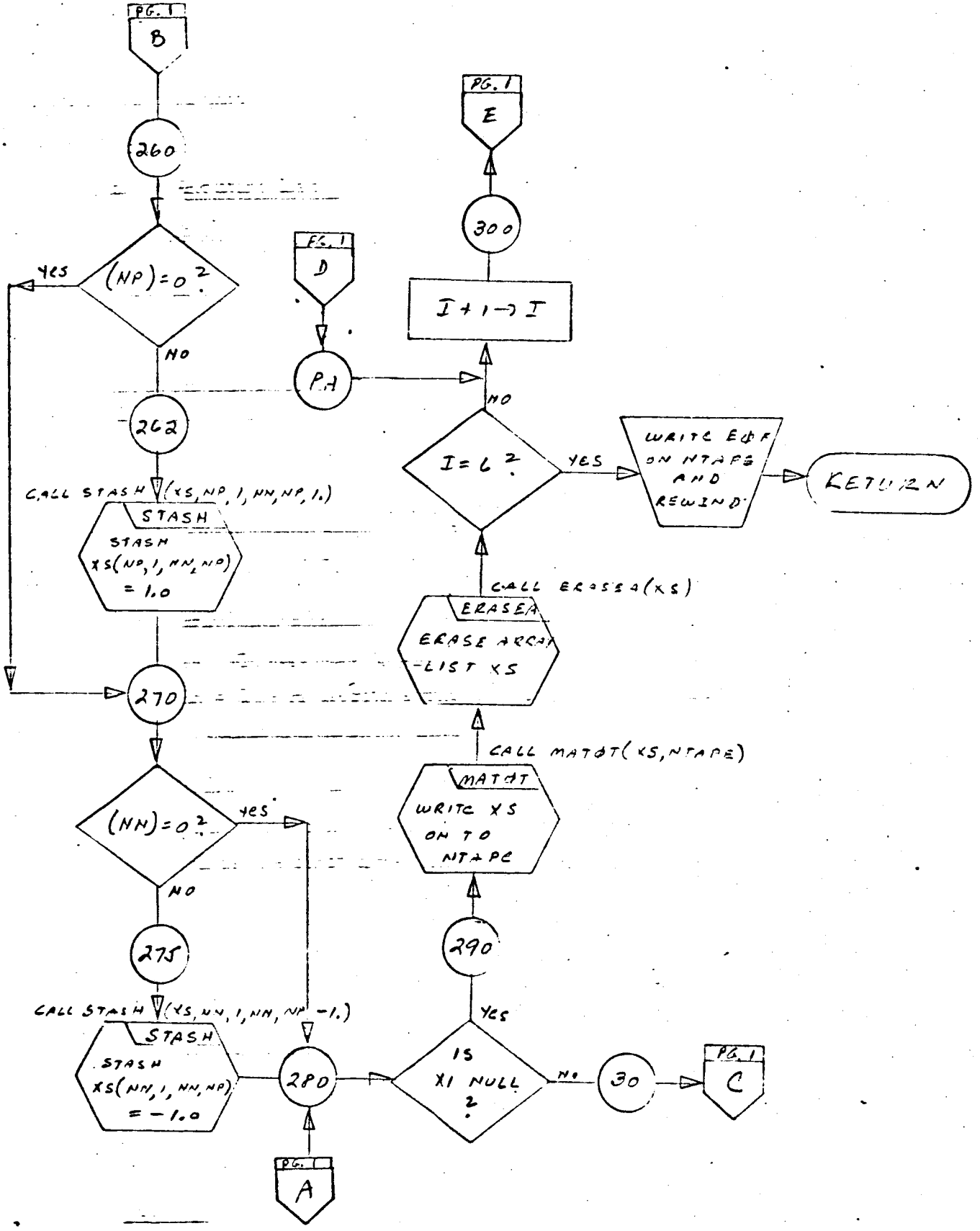
ERASEA, MATOT, PRPTR, STASH.

8. Using Subroutines

Main Program of Pass 1 for TAG Preprocessor.

CALL PARAM(XS, WLIST, NTAPE)




PG. 1
B
260
(NP) = 0 ?
YES
NO
PG. 1
D
P.A
262
CALL STASH (XS, NP, 1, NN, NP, 1.)
STASH
STASH XS(NP, 1, NN, NO) = 1.0
270
(NN) = 0 ?
YES
NO
275
CALL STASH (XS, NN, 1, NN, NP, -1.)
STASH
STASH XS(NN, 1, NN, NP) = -1.0
280
PG. 1
A
PG. 1
E
300
I + 1 → I
NO
I = 6 ?
YES
WRITE EOF ON NTAPE AND REWIND
RETURN
CALL ERASEA(XS)
ERASEA
ERASE ARRAY-LIST XS
CALL MATOT(XS, NTAPE)
MATOT
WRITE XS ON TO NTAPE
290
YES
IS XI NULL ?
NO
30
PG. 1
C

Program Description

1. Identification

   a. Routine Label

      PARTS

2. Function

   Outputs a statement of the form: XXX = N.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL PARTS(N, H)

   b. Entry Conditions

      N = The integer to appear on the right side of the equation
      H = The Hollerith symbol to appear on the left side. H consists of packed BCD characters.

   c. Exit Conditions

      The statement h = n is output.
      (h is the first three characters of H. n is the integer value of N.)

   d. Error Exits

      None.

5. Definition of Identifiers

   IWRK = A type A list containing the first three characters of H
   IOUT = IWRK, but in reverse order
   EQL = BCD equals sign

6. Method

The first three characters of H are extracted and placed in IWRK, then reversed into IOUT. An "=" is pushed into IOUT. The value of

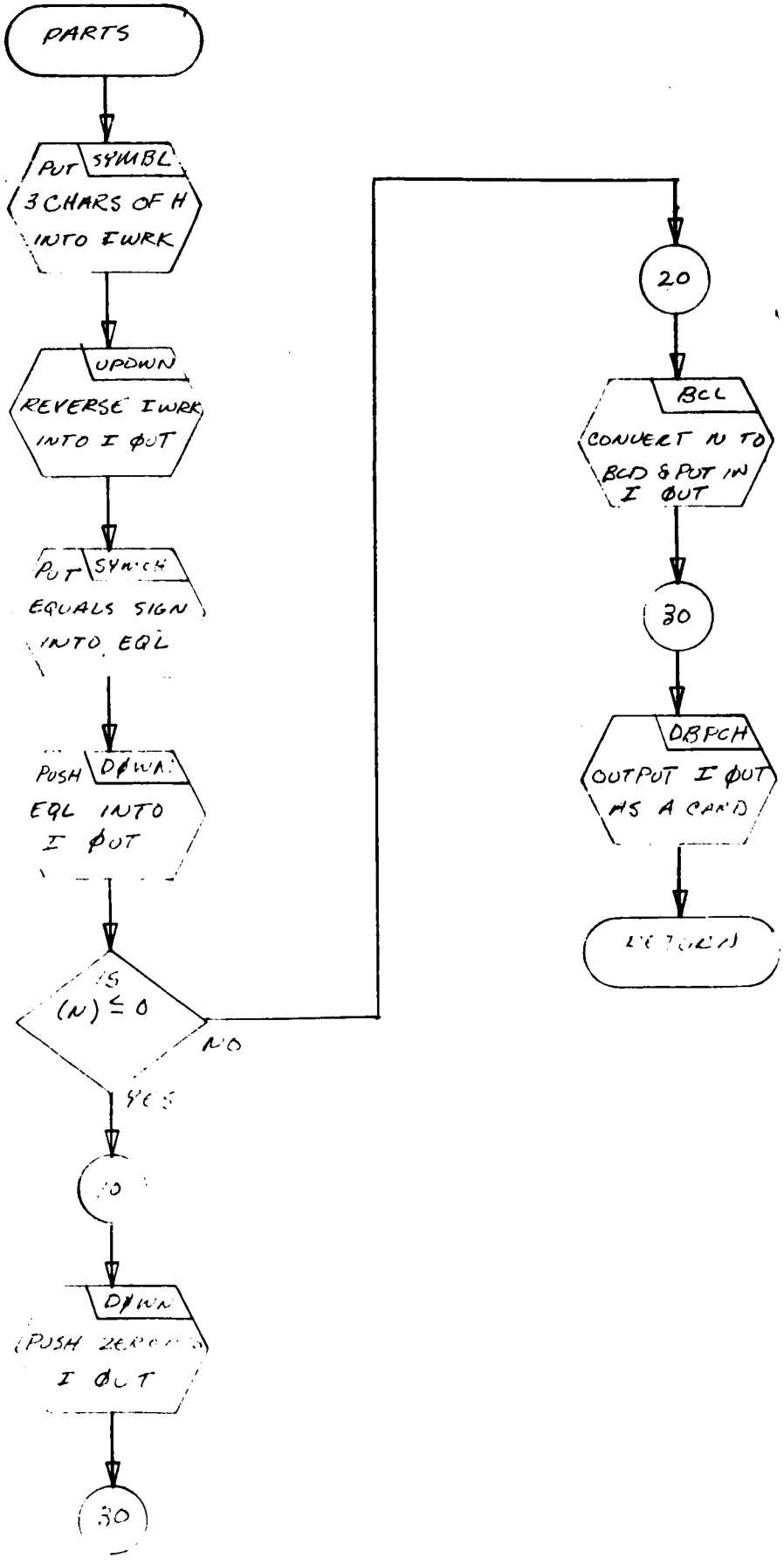
N is converted to BCD and pushed into IOUT, and then IOUT is output as a card image.

7. Other Subroutines Used

   BCL, DBPCH, DOWN, SYMBL, SYMCH, UPDWN.

8. Using Subroutines

   Main Routine, Pass 2, TAG Preprocessor.


PARTS
PUT SYMBL
3 CHARS OF H
INTO IWRK
UPDWN
REVERSE IWRK
INTO I ØUT
PUT SYMBL
EQUALS SIGN
INTO EQL
PUSH DØWN
EQL INTO
I ØUT
IS
(N) ≤ 0
NO
YES
20
BCL
CONVERT N TO
BCD & PUT IN
I ØUT
30
DRPCH
OUTPUT I ØUT
AS A CARD
RETURN
30
DØWN
PUSH ZERO
I ØUT
30

Program Description

1. Identification

   a. Routine Label

      POPUP

   b. Name

      POPUP the first element from a list.

2. Function

The first element of a list is popped up, and its item and flag are saved. The element is then restored to AVS, and the count of elements used is decreased by one.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      Call POPUP (Z, C, IF)

   b. Entry Conditions

      Z = Head of a list

   c. Exit Conditions

      d(C) = Item portion of the element popped up from Z
      d(IF) = Flag portion of the same element
      Head of list Z will point to the next element in the list.

   d. Error Exits

If list Z is null, CALL DUMP is executed, and return is made to the FORTRAN Monitor System.

5. Definition of Identifiers

   (PP90) = Contents of the element popped up from list Z.

6. Method

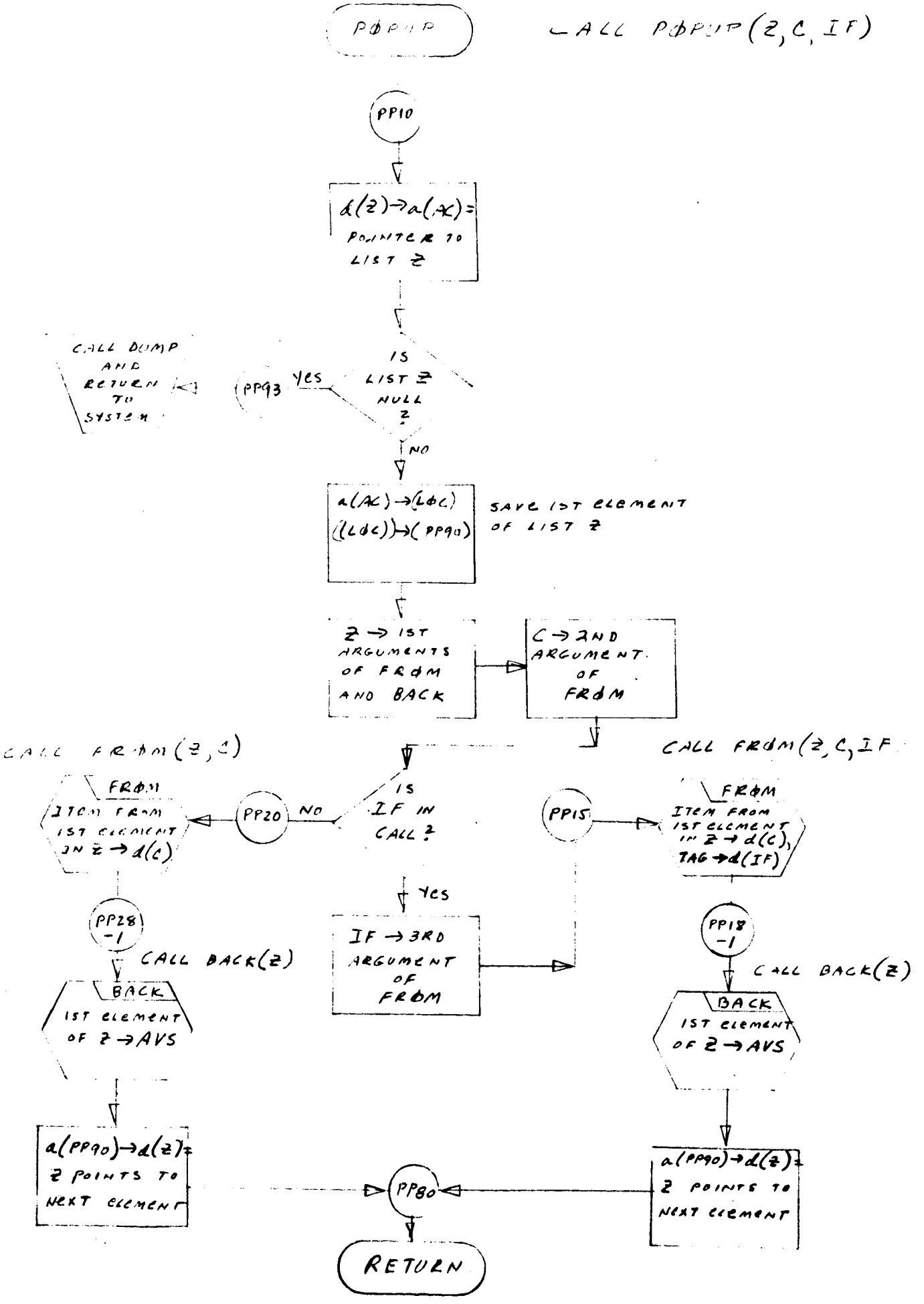
The item portion of the first element in list Z is placed in d(C), and the flag portion, if any, is saved in d(IF) by the subroutine FROM. The link portion (bits 21-35) of the element, saved in a(PP90), is placed in the d(Z), so that Z will now be pointing to the next (second) element of the list. If Z was null upon entry to the routine, CALL DUMP is executed and return is made to the FORTRAN Monitor System.

7. Other Subroutines Used

BACK, FROM

8. Using Subroutines

BLNOUT, COMBN, DBPCHC, DBPFH, DONBD, DPDST, ERASE, EXCPT, EXTRX, FISH, GOBLE, HOLBK, INPUTX, INZERO, NUMB, PUSPCH, RECOVR, SUBST, SYMTP, UPDWN, WRTEQ, Main Program for Pass 2 of TAG Preprocessor.


PØPUP
CALL PØPUP(Z, C, IF)
PP10
d(Z)→a(AC)= POINTER TO LIST Z
IS LIST Z NULL ?
yes
PP93
CALL DUMP AND RETURN TO SYSTEM
NO
a(AC)→(LØC)
((LØC))→(PP90)
SAVE 1ST ELEMENT OF LIST Z
Z → 1ST ARGUMENTS OF FRØM AND BACK
C → 2ND ARGUMENT OF FRØM
CALL FRØM(Z, C)
CALL FRØM(Z, C, IF
IS IF IN CALL ?
NO
PP20
FRØM
ITEM FROM 1ST ELEMENT IN Z → d(C)
PP28 -1
CALL BACK(Z)
BACK
1ST ELEMENT OF Z → AVS
a(PP90)→d(Z)= Z POINTS TO NEXT ELEMENT
yes
IF → 3RD ARGUMENT OF FRØM
PP15
FRØM
ITEM FROM 1ST ELEMENT IN Z → d(C), TAG → d(IF)
PP18 -1
CALL BACK(Z)
BACK
1ST ELEMENT OF Z → AVS
a(PP90)→d(Z)= Z POINTS TO NEXT ELEMENT
PP80
RETURN

Program Description

1. Identification

a. Routine Label

PRPTG

b. NAME

Place the parameters of a single branch descriptor into a compound list.

2. Function

A simple list representation of a single branch descriptor is formed of the quantities, in order: NE/NF, NN, NP, NTRN, NNTR. For branches other than transformer windings, NTRN and NNTR are omitted. A type 7 element whose item points to this simple list is pushed down into the list, LIST, to create a type D list of branch descriptors.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL PRPTG (LIST, NE, NN, NP, NTRN, NNTR, NF)

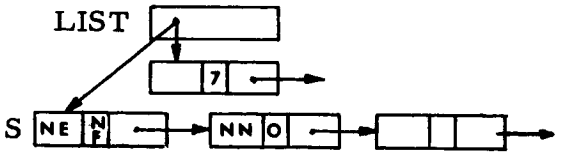
b. Entry Conditions

LIST	= Head cell of the main list being formed
NE	= The integer representing the element type of a particular branch. If NE = 5, NTRN, NNTR contain transformer information
NN	= The number of the first, or negative, node
NP	= The number of the second, or positive, node
NTRN	= The transformer core number
NNTR	= The number of turns on a transformer unwinding
NF	= The flag in the NE element of the list

c. Exit Conditions

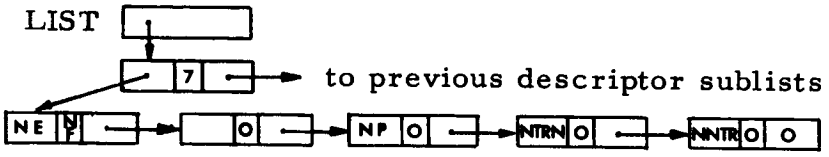
The list pointed to by LIST may be represented as shown below:

1. No transformers (NE ≠ 5)



to previous descriptor sublists

2. Transformers (NE = 5)



d. Error Exits

None.

5. Definition of Identifiers

S = Pointer to a sublist which contains NE/NF, NN, NP, NTRN, NNTR

T = Temporary head cell for generating sublist S. The pointer in T changes after each insertion of a parameter

6. Method

A new parameter list is pushed down into LIST as outlined in the steps below:

a. T is initialized to 0.

b. NE with a tag of NF is inserted into the first element of T. The location of this element is saved in S. T is set to point to the first element.

c. NN with a tag of 0 is inserted after the first element of T. T is reset to point to this new element.

d. NP with a tag of 0 is inserted after the first element of T. T is reset to point to this new element.

e. If NE ≠ 5, S with a tag of 7 is inserted between the head cell and first element of LIST. The new list is complete, and return is made.

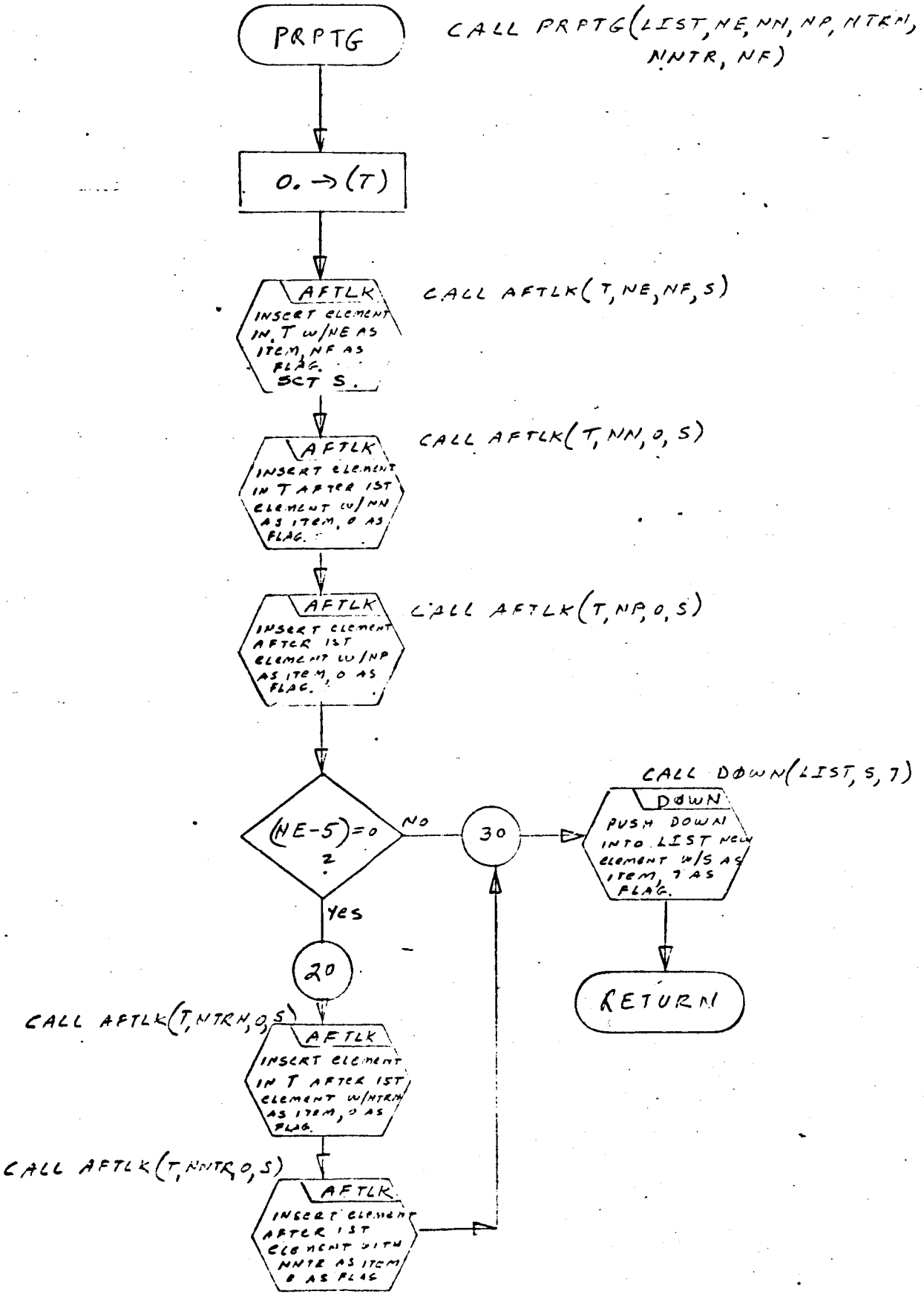
f. If NE = 5, NTRN and NNTR with tags of 0 are inserted as the next two elements in T; then step 3 is executed.

7. Other Subroutines Used

AFTLK, DOWN.

8. Using Subroutines

GOBLE, INTLST, STRIK, TREE, Main Program for Pass 1 of TAG Preprocessor.


PRPTG
CALL PRPTG(LIST, NE, NN, NP, NTRM, NNTR, NF)
0. → (T)
AFTLK
INSERT ELEMENT IN T W/NE AS ITEM, NF AS FLAG. SET S.
CALL AFTLK(T, NE, NF, S)
AFTLK
INSERT ELEMENT IN T AFTER 1ST ELEMENT W/NN AS ITEM, 0 AS FLAG.
CALL AFTLK(T, NN, 0, S)
AFTLK
INSERT ELEMENT AFTER 1ST ELEMENT W/NP AS ITEM, 0 AS FLAG.
CALL AFTLK(T, NP, 0, S)
(NE-5)=0
NO
30
CALL DOWN(LIST, S, 7)
DOWN
PUSH DOWN INTO LIST NEW ELEMENT W/S AS ITEM, 7 AS FLAG.
RETURN
Yes
20
CALL AFTLK(T, NTRM, 0, S)
AFTLK
INSERT ELEMENT IN T AFTER 1ST ELEMENT W/NTRM AS ITEM, 0 AS FLAG.
CALL AFTLK(T, NNTR, 0, S)
AFTLK
INSERT ELEMENT AFTER 1ST ELEMENT WITH NNTR AS ITEM 0 AS FLAG

Program Description

1. Identification

a. Routine Label

PRPTR

b. Name

Extract a single branch descriptor from the next sublist.

2. Function

This subroutine extracts the elements associated with a single branch descriptor from a simple sublist of a type D list. The sublist is not destroyed, and the head of the list is reset to point to the next sublist.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL PRPTR (X, NE, NN, NP, NTRN, NNTR, NF)

b. Entry Conditions

X = Head of a list with a type D format whose first element points to the next sublist of X.

c. Exit Conditions

The elements in the first descriptor sublist of X are extracted and given variable names as shown below:

NE = The integer representing the element type of a particular branch. If NE = 5, NTRN, NNTR contain transformer information.

NN = The number of the first, or negative, node

NP = The number of the second, or positive, node

NTRN = The transformer core number (= 0 if NE $\neq$ 5)

NNTR = The number of turns on a transformer winding (= 0 if NE ≠ 5)

NF = The flag in the NE element of the list

The sublist which contained these elements remains intact. X is reset so that its first sublist is the next descriptor sublist of the main list.

d. Error Exits

None.

5. Definition of Identifiers

Z = Temporary head cell that points to the next sublist of X. The pointer in Z changes after each extraction of an element type.

ITEM = Local variable that holds an element of the branch descriptor.

6. Method

The next descriptor is extracted from X as follows:

a. Item ((X)) → d(Z) initializes Z to point to the next sublist containing a single branch descriptor.

b. NTRN and NNTR are cleared to zero.

c. The branch descriptor element type index I is initialized to 1.

d. Z is tested:

(1) If Z is null, link ((X)) → d(X) sets the first element in X to point to the next descriptor sublist, and exit is made from the routine.

(2) If Z is not null, item ((Z)) → d(ITEM) provides the next element of the branch descriptor in ITEM.

(3) I is tested:

(a) If I = 1, the element type is NE. The item and flag of the first element in Z is stored in d(NE) and d(NF), respectively.

(b) If I = 2, the element type is NN. d (ITEM) is stored in NN.

(c) If I = 3, the element type is NP. d (ITEM) is stored in NP.

(d) If I = 4, the element type is NTRN. d (ITEM) is stored in NTRN.

(e) If I = 5, the element type is NNTR. d(ITEM) is stored in NNTR.

(4) Link ((Z)) → d(Z) gives the location of the next element type in the Z sublist.

(5) I is tested:

(a) If I = 5, all elements have been extracted for the branch descriptor. Step d(1) is repeated; and exit is made from the routine.

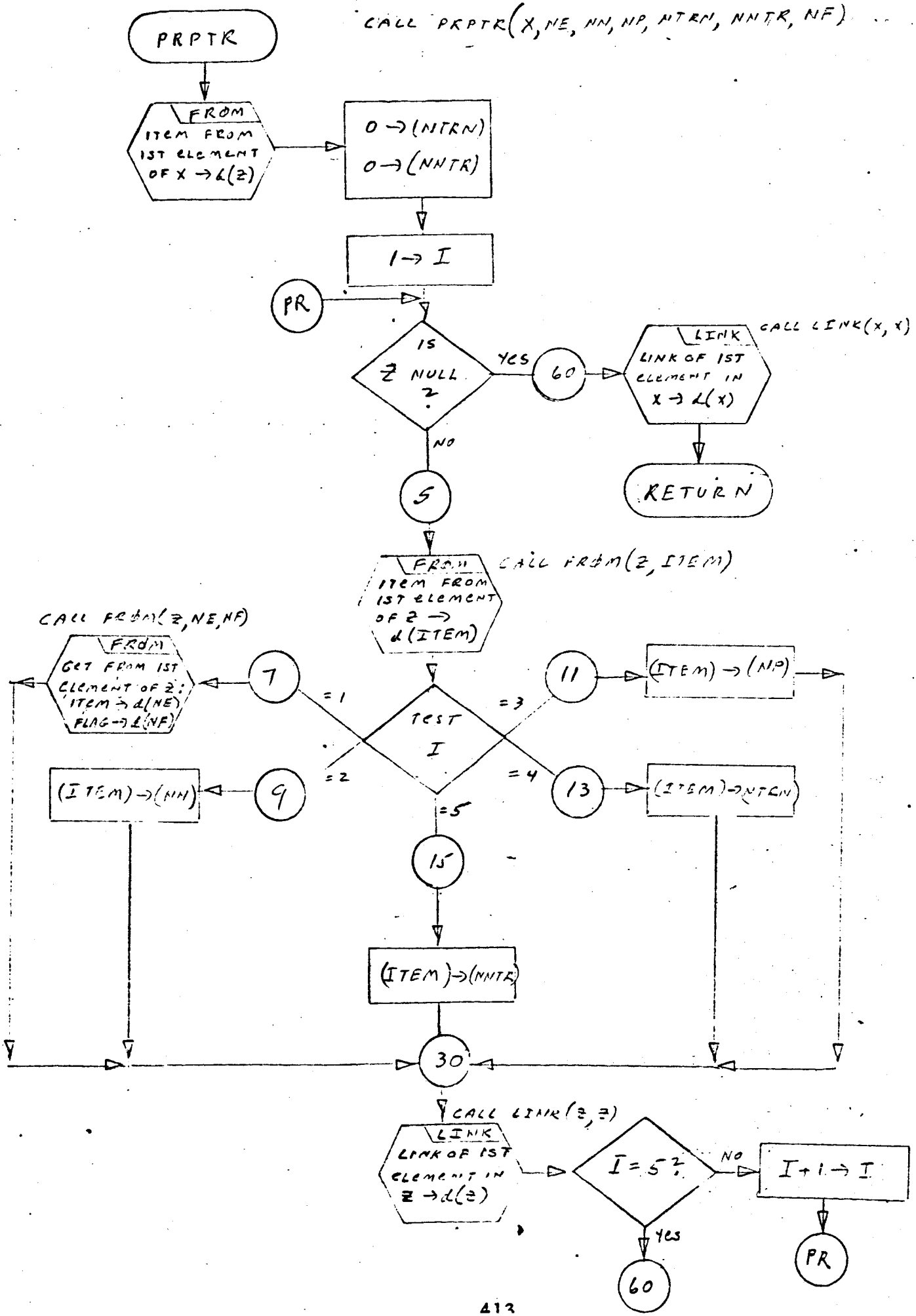
(b) If I ≠ 5, I is increased by 1, and steps d(1) through d(5) are repeated for the next element type.

7. Other Subroutines Used

FROM, LINK.

8. Using Subroutines

COTRN, INPUTX, PARAM, STRIK, TREE, XFORM, Main Program for Pass 1 of TAG Preprocessor.


PRPTR
CALL PRPTR(X, NE, NN, NP, NTRN, NNTR, NF)
FROM
ITEM FROM 1ST ELEMENT OF X → L(Z)
0 → (NTRN)
0 → (NNTR)
1 → I
PR
IS Z NULL ?
YES
60
LINK
LINK OF 1ST ELEMENT IN X → L(X)
CALL LINK(X, X)
RETURN
NO
5
FROM
ITEM FROM 1ST ELEMENT OF Z → L(ITEM)
CALL FROM(Z, ITEM)
CALL FROM(Z, NE, NF)
FROM
GET FROM 1ST ELEMENT OF Z: ITEM → L(NE) FLAG → L(NF)
7
TEST I
=1
=2
=3
=4
=5
11
(ITEM) → (NP)
9
(ITEM) → (NN)
13
(ITEM) → (NTRN)
15
(ITEM) → (NNTR)
30
CALL LINK(Z, Z)
LINK
LINK OF 1ST ELEMENT IN Z → L(Z)
I = 5 ?
NO
I + 1 → I
YES
60
PR

Program Description

1. Identification

   a. Routine Label

      PUSPCH

2. Function

   Constructs and outputs a card image from a character list.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL PUSPCH (P, IZ, NTAPE, KE, IC1)

   b. Entry Conditions

      P = List of characters (type A list) to be output as a card image. P is in reverse order.

      IZ = Statement number to be used for output statement.

      NTAPE = Tape number of tape on which card image is to be written

      KE = Two-digit integer to be placed in columns 73-74 of card image

      IC1 = Comment card indicator

   c. Exit Conditions

      C(P) output as a series of card images on NTAPE

   d. Error Exits

      None.

5. Definition of Identifiers

   A Output buffer

   ICHC BCD comma

Z	Used to hold P in reverse order
IS	Holds statement number IZ
TEM1	Temporary storage
KSTRT	Column number of beginning of image
IC	Continuation card number
K6	Column number used to accomplish character packing
I	Card sequence number

6. Method

C(P) is reversed into Z.

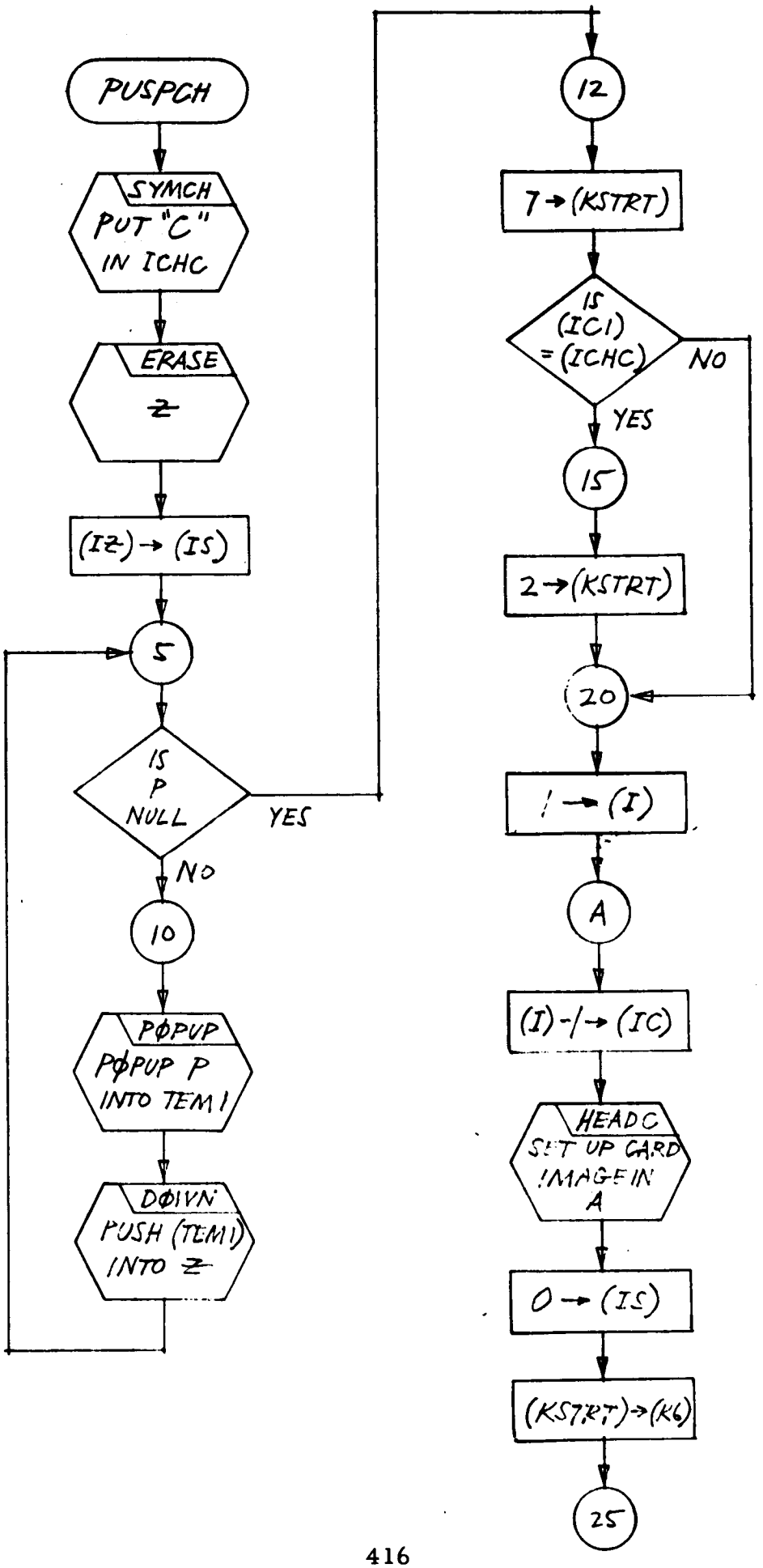
If IC1 is a BCD "C," then card image is set up as a comment card by setting the starting column number = 2. Else the starting column is set as 7. HEADC is used to format the contents of columns 1-6, using IC1, IS, IC. The characters in Z are then inserted, one at a time, into the buffer A. When full, it is output to NTAPE. This may continue for up to 10 continuation cards. If Z is still not empty, the statement "A GENERATED STATEMENT HAS MORE THAN 10 LINES" will be printed on the output listing.

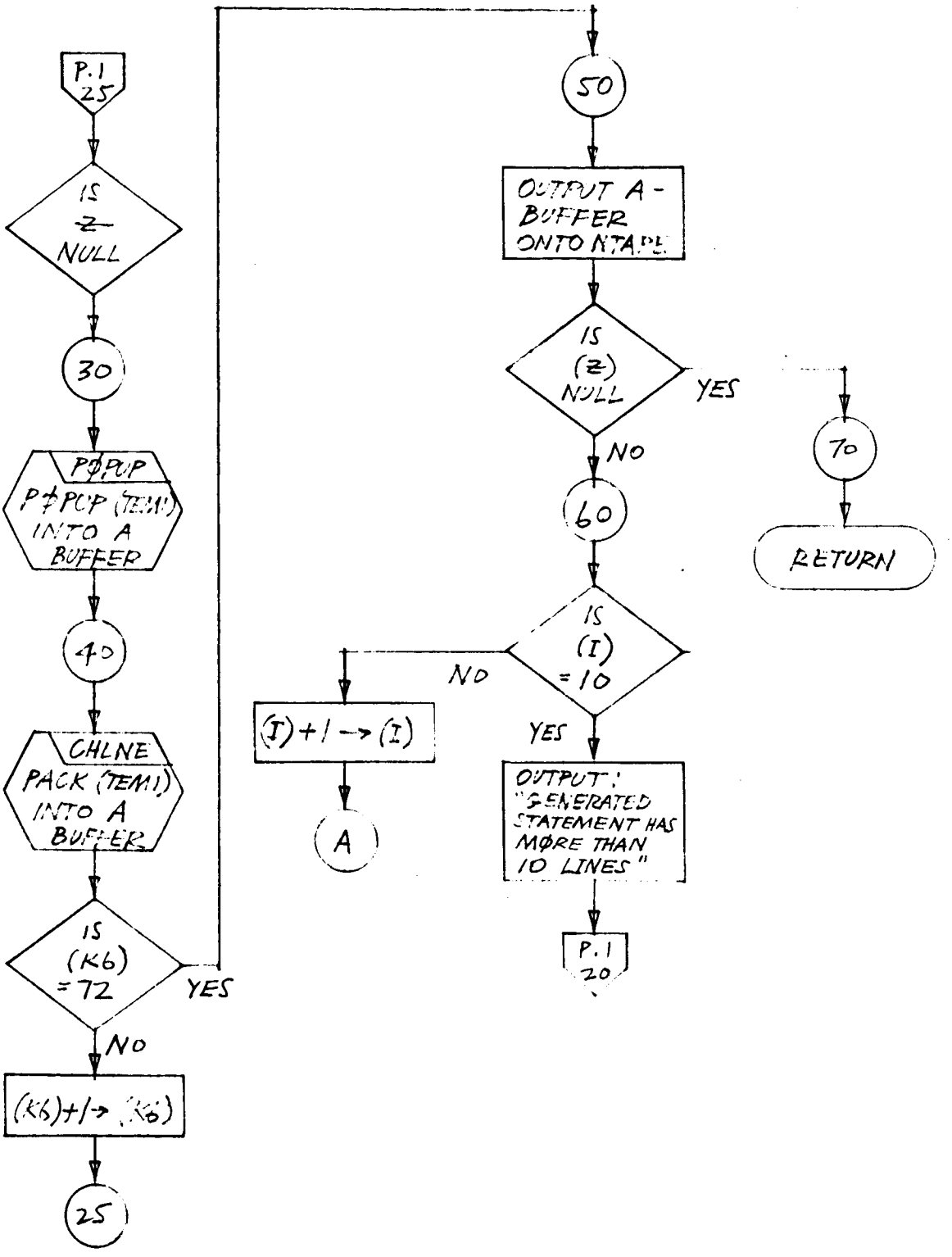
7. Other Subroutines Used

CHLNE, DOWN, ERASE, HEADC, POPUP, SYMCH.

8. Using Subroutines

Main Routine, Pass 2, TAG Preprocessor.


PUSPCH
SYMCH
PUT "C" IN ICHC
ERASE
Z
(IZ) → (IS)
5
IS P NULL
YES
NO
10
PØPUP
PØPUP P INTO TEM1
DØIVN
PUSH (TEM1) INTO Z
12
7 → (KSTRT)
IS (IC1) = (ICHC)
NO
YES
15
2 → (KSTRT)
20
1 → (I)
A
(I) -/→ (IC)
HEADC
SET UP CARD IMAGE IN A
0 → (IS)
(KSTRT) → (K6)
25


P.1
25
IS
Z
NULL
30
POPUP
POPUP (TEM1)
INTO A
BUFFER
40
CHLNE
PACK (TEM1)
INTO A
BUFFER
IS
(K6)
=72
YES
NO
(K6)+1→(K6)
25
50
OUTPUT A-
BUFFER
ONTO NTAPE
IS
(Z)
NULL
YES
NO
60
IS
(I)
=10
NO
YES
(I)+1→(I)
A
OUTPUT:
"GENERATED
STATEMENT HAS
MORE THAN
10 LINES"
P.1
20
70
RETURN

Program Description

1. Identification

   a. Routine Label

      READCH

   b. Name

      Read a card image from tape and push characters down into a list.

2. Function

This subroutine reads a card image from NTAPE when IC = 0; the card image is the first statement, and it is read into buffer A. When IC = 1, NTAPE is not read; the card image to be processed will be contained in A. In either case, the characters in A are processed and pushed down into list P.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL READCH (P, IS, IC, IPR, A, NTAPE, KE, IC1)

   b. Entry Conditions

NTAPE	=	Input tape
IC	=	0 if card image is on NTAPE (first statement)
IC	=	1 if card image is in buffer A
P	=	Location available for a push-down list

   c. Exit Conditions

P contains the card image in a push-down list of characters. IS contains the statement number as an integer. IC = 0 if this was an end card; IC = 1 if otherwise. IPR will contain print control information (1 = EJECT, 0 = DOUBLE SPACE, BLANK = REGULAR SPACING).

Buffer A will contain the next card image. KE is the statement type that flags the type of format required for the card image in P. IC1 contains the character from column 1.

d. Error Exits

None.

5. Definition of Identifiers

T = A temporary cell used as head of a list which points to the first element of list P

KEE = A local variable containing the statement type (read in from NTAPE)

6. Method

The card image on NTAPE (or in buffer A) is processed as follows:

a. IC is tested.

(1) If IC = 0, this is the first statement.

NTAPE is read and the card image is saved in A. The statement type is saved in KEE. IC is set to 1 (card image in A). Processing continues at step b.

(2) If IC = 1, the card image is already contained in A. Continue at step b.

b. Extract from buffer A.

(1) The statement number is saved in IS.

(2) The continuation card number is saved in IX.

(3) Column 1 is placed in IC1.

c. Statement type in KEE is stored in KE.

d. A test is made to determine if this is a comment card.

(1) If it is a comment card, columns 2-72 are extracted from buffer A and pushed down into list P.

(2) If it is not a comment card, columns 7-72 are extracted from buffer A and pushed down into list P.

e. Column 73 is extracted from A, and the appropriate print control character is placed in IPR.

f. NTAPE is tested.

(1) If NTAPE = 5, a title is printed (written onto the print tape) and page count is updated. The next step is then executed.

(2) If NTAPE ≠ 5, buffer A (13 BCD words), is written onto the print tape.

g. IX is tested.

(1) If IX = 1, the card image being processed is not a continuation card.

(a) T is set to point to list P.

(b) T is tested.

(i) If T is null, continue at step g(2)(a).

(ii) If T is not null, continue at next step.

(c) List T is scanned for its first nonblank item. When a nonblank item has been found, the next two items are extracted from T.

(i) If the three items contain the characters "END," IC is set to zero and exit is made from the routine.

(ii) If the characters are not "END," continue at next step.

(2) If IX ≠ 1, the card image being processed is a continuation card.

(a) Read the next card image from NTAPE and save in buffer A.

(b) Extract continuation card code from A and save in IX.

(c) Test IX.

(i) IF IX = 1, the next card to be processed is not a continuation card. Exit is made from the routine.

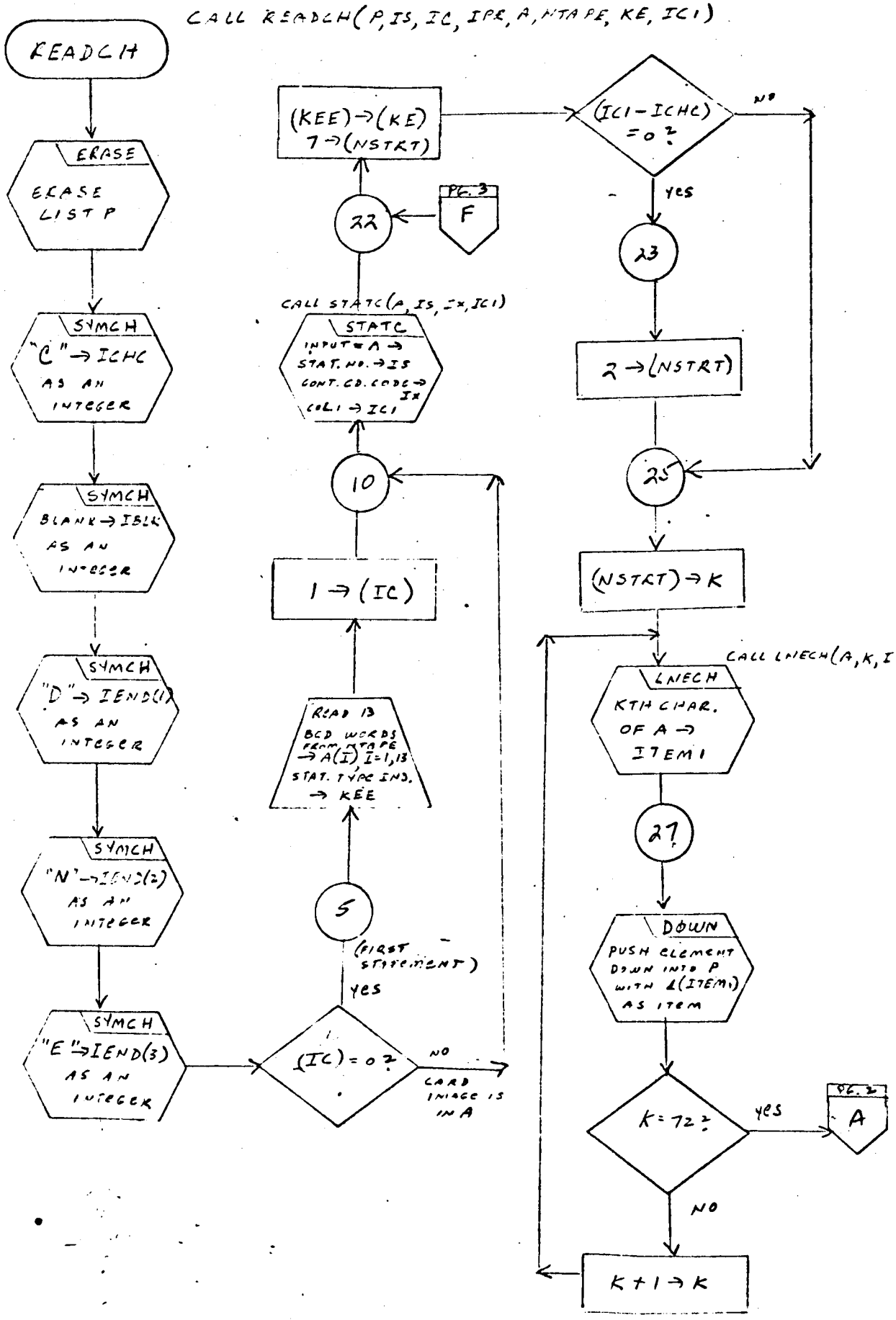
(ii) If IX ≠ 1, the next card to be processed is a continuation card. Continue at step c.

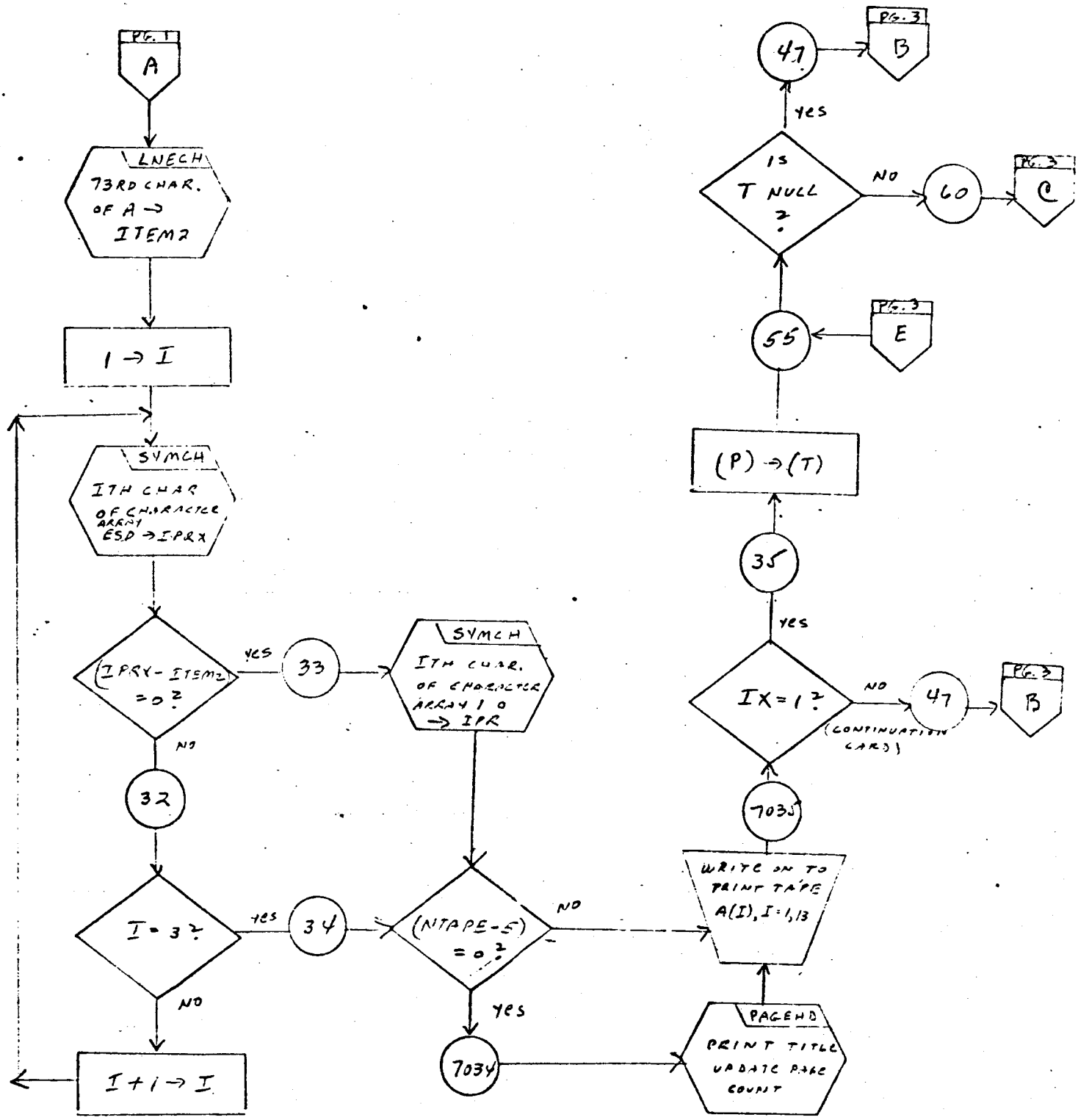
7. Other Subroutines Used

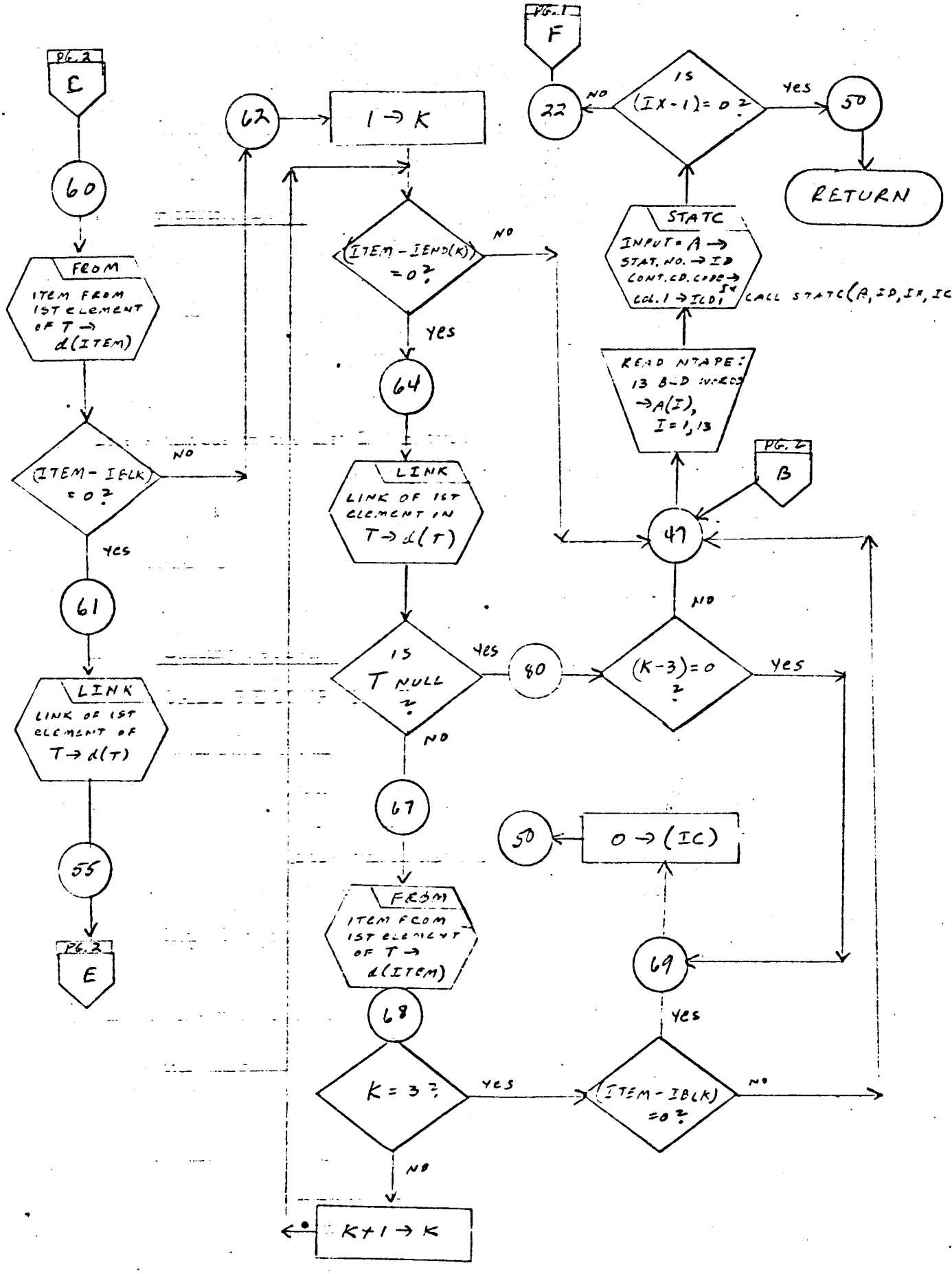
DOWN, ERASE, FROM, LINK, LNECH, PAGEHD, STATC, SYMCH.

8. Using Subroutines

Main Program for Pass 2 of TAG Preprocessor.


CALL READCH(P,IS,IC,IPE,A,NTAPE,KE,IC1)
READCH
ERASE
ERASE LIST P
SYMCH
"C" → ICHC AS AN INTEGER
SYMCH
BLANK → IBLK AS AN INTEGER
SYMCH
"D" → IEND(1) AS AN INTEGER
SYMCH
"N" → IEND(2) AS AN INTEGER
SYMCH
"E" → IEND(3) AS AN INTEGER
(IC) = 0 ?
NO
CARD IMAGE IS IN A
(FIRST STATEMENT)
yes
5
READ 13 BCD WORDS FROM NTAPE → A(I) I=1,13 STAT. TYPE IND. → KEE
1 → (IC)
10
CALL STATC(A,IS,IX,IC1)
STATC
INPUT = A → STAT. NO. → IS CONT. CD. CODE → IX COL1 → IC1
22
PG. 3
F
(KEE) → (KE) 7 → (NSTRT)
(IC1 - ICHC) = 0 ?
NO
yes
23
2 → (NSTRT)
25
(NSTRT) → K
CALL LNECH(A,K,ITEM1)
LNECH
KTH CHAR. OF A → ITEM1
27
DOWN
PUSH ELEMENT DOWN INTO P WITH L(ITEM1) AS ITEM
K = 72 ?
yes
PG. 2
A
NO
K + 1 → K


PG. 1
A
LNECH
73RD CHAR. OF A → ITEM2
1 → I
SYMCH
ITH CHAR OF CHARACTER ARRAY ESD → IPRX
(IPRX - ITEM2) = 0 ?
yes
33
SYMCH
ITH CHAR. OF CHARACTER ARRAY I O → IPR
no
32
I = 3 ?
yes
34
(NTAPE-5) = 0 ?
no
yes
7034
I + 1 → I
no
PAGEHD
PRINT TITLE UPDATE PAGE COUNT
WRITE ON TO PRINT TAPE A(I), I = 1,13
7035
IX = 1 ?
no
47
PG. 3
B
(CONTINUATION CARD)
yes
35
(P) → (T)
55
PG. 3
E
IS T NULL ?
yes
47
PG. 3
B
no
60
PG. 3
C


PG. 1
F
22
NO
IS (IX-1) = 0 ?
yes
50
RETURN
STATC
INPUT = A →
STAT. NO. → ID
CONT. CD. CODE → IC
COL. 1 → ICD
CALL STATC(A, ID, IX, IC
READ NTAPE: 13 B-D WORDS →A(I), I = 1, 13
PG. 2
B
47
NO
(K-3) = 0 ?
yes
80
IS T NULL ?
yes
NO
67
50
0 → (IC)
69
yes
(ITEM - IBLK) = 0 ?
NO
FROM
ITEM FROM 1ST ELEMENT OF T → d(ITEM)
68
K = 3 ?
yes
NO
K+1 → K
LINK
LINK OF 1ST ELEMENT IN T → d(T)
64
(ITEM - IEND(K)) = 0 ?
NO
yes
62
1 → K
PG. 2
E
60
FROM
ITEM FROM 1ST ELEMENT OF T → d(ITEM)
(ITEM - IBLK) = 0 ?
NO
yes
61
LINK
LINK OF 1ST ELEMENT OF T → d(T)
55
PG. 2
E

Program Description

1. Identification

   a. Routine Label

      RECOVR

   b. Name

      Recover symbol string from a symbol list structure.

2. Function

The symbols in the list IN (in type C format) are re-collected as individual symbols and placed in list IN1 in the type B list format.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL RECOVR (IN, IN1)

   b. Entry Conditions

      IN = Head of a list of type C format. List IN contains symbols separated by commas. Those symbols which have a common initial character are stored as one initial character plus two substrings.

   c. Exit Conditions

The type C list IN is re-collected in list IN1 as a simple symbol string of the type B format.

   d. Error Exits

      None.

5. Definition of Identifiers

   IXX = Initially set to a blank, otherwise contains a comma

ICM	=	Contains the Hollerith representation of a comma
IRC	=	A temporary cell that is the head of a list of pointers to the sublists being scanned in IN
L1	=	Head cell that is set to point to the top of list IN
TEM1	=	A temporary list containing the copied portion of TEM
TEM	=	A temporary list used to collect the symbols that will be placed into list IN1
ITEM	=	Contains one character of the symbol being processed

6. Method

List IN is placed into list IN1 as outlined in the steps below.

a. IXX is initialized to a blank, ICM to a comma. Lists IRC and TEM are erased.

b. L1 is set to point to list IN.

c. The pointer to list L1 is pushed down into IRC .

d. TEM1 is made null.

e. TEM is copied into TEM1.

f. The pointer to TEM1 is pushed down into IRC.

g. The item of the first element in L1 (pointer to a sublist) is placed in d(L2).

h. The item from the first element pointed to by d(L2) is saved in ITEM.

i. ITEM is tested

   (1) If the character in ITEM is not a comma:

      (a) The character in ITEM is pushed down into list TEM as the item of a new element.

      (b) L1 is reset with the link of the first element in L2.

      (c) Processing of list L1 continues at step c.

   (2) If the character in ITEM is a comma:

(a) IXX is pushed down into IN1 as the item of a new element.

(b) List TEM is copied into IN1.

(c) IXX is reset with the comma in ICM.

j. IRC is tested.

(1) If IRC is null, exit is made from the routine.

(2) If IRC is not null:

(a) The next (or first) element in IRC is popped up and its item (pointer to a sublist) is saved in d(TEM).

(b) The next element in IRC is popped up and its item (pointer to a sublist) is saved in d(L1).

k. L1 is tested.

(1) If L1 is null, go to step j to continue.

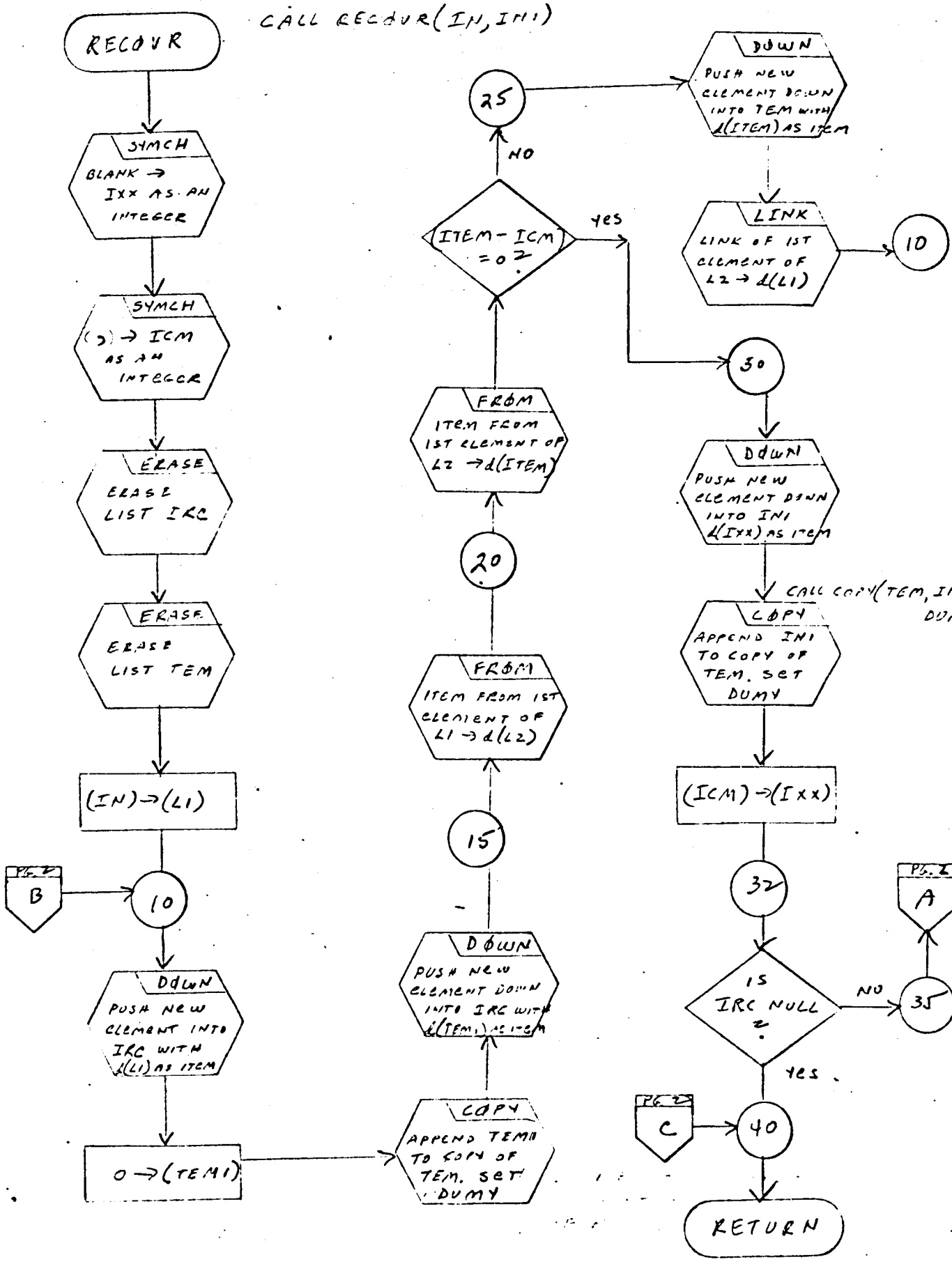
(2) If L1 is not null, go to step c to continue processing.

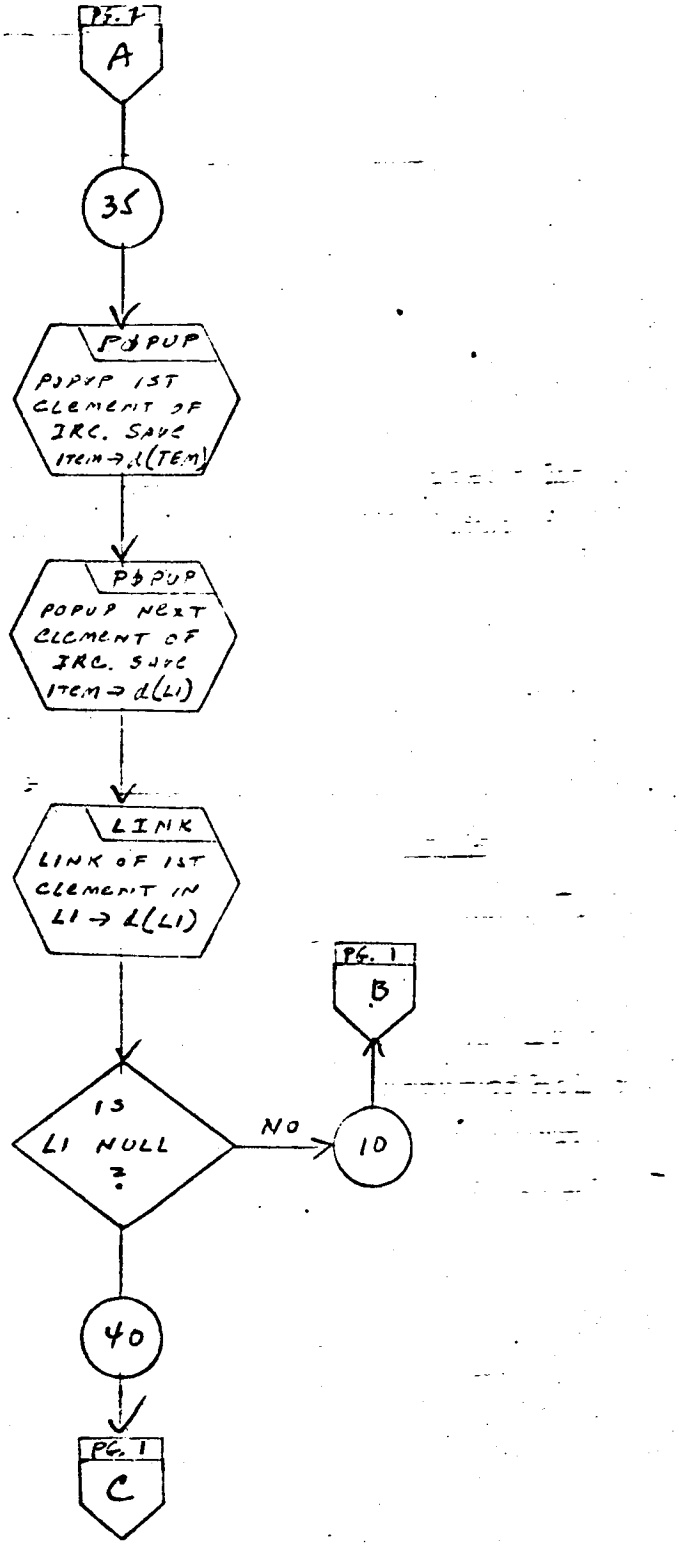
7. Other Subroutines Used

COPY, DOWN, ERASE, FROM, LINK, POPUP, SYMCH.

8. Using Subroutines

DPDST, INPUTX.


CALL RECOVR(IN,IN1)
RECOVR
SYMCH
BLANK → IXX AS AN INTEGER
SYMCH
( ) → ICM AS AN INTEGER
ERASE
ERASE LIST IRC
ERASE
ERASE LIST TEM
(IN) → (L1)
PG. 2
B
10
DOWN
PUSH NEW ELEMENT INTO IRC WITH L(L1) AS ITEM
0 → (TEM1)
COPY
APPEND TEM1 TO COPY OF TEM. SET DUMY
DOWN
PUSH NEW ELEMENT DOWN INTO IRC WITH L(ITEM1) AS ITEM
15
FROM
ITEM FROM IST ELEMENT OF L1 → L(L2)
20
FROM
ITEM FROM IST ELEMENT OF L2 → L(ITEM)
ITEM - ICM = 0 ?
NO
25
DOWN
PUSH NEW ELEMENT DOWN INTO TEM WITH L(ITEM) AS ITEM
yes
LINK
LINK OF IST ELEMENT OF L2 → L(L1)
10
30
DOWN
PUSH NEW ELEMENT DOWN INTO IN1 L(IXX) AS ITEM
CALL COPY(TEM, IN1, DUMY)
COPY
APPEND IN1 TO COPY OF TEM. SET DUMY
(ICM) → (IXX)
32
IS IRC NULL ?
NO
35
PG. 2
A
yes
PG. 2
C
40
RETURN


PG. 1
A
35
POPUP
POPUP 1ST ELEMENT OF IRC. SAVE ITEM → L(ITEM)
POPUP
POPUP NEXT ELEMENT OF IRC. SAVE ITEM → L(LI)
LINK
LINK OF 1ST ELEMENT IN LI → L(LI)
PG. 1
B
IS LI NULL ?
NO
10
40
PG. 1
C

Program Description

1. Identification

   a. Routine Label

      SEGMNT

   b. Name

      Segment a list.

2. Function

This subroutine divides a list into two new lists. The division takes place at the first character in the list that matches any of the first H characters of a specified list of Hollerith characters.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SEGMNT (A, B, I, N, IHL)

   b. Entry Conditions

      A = Head of the list to be segmented
      IHL = An array of Hollerith characters
      N = Number of characters in IHL to be matched with characters in list A

   c. Exit Conditions

At the first character in A that matches one of the N characters in Hollerith array IHL, the original list A is severed. A is the head of the list of all the characters of A following the matched character. B is the head of a list starting at the first character, formerly in list A, and extending to and including the matched character. I is the number of the characters in the Hollerith array IHL for which the match was achieved.

If a match is not found, and the end of list A is reached, I = N + 1, and list A is null. If A is null upon entry, list B is erased and I is set to N + 2.

d. Error Exits

None.

5. Definition of Identifiers

AL = Temporary head cell used to save the current pointer of A during the search for IHL

6. Method

Upon entry to the routine, list B is erased, and B is set to point to the top of list A. Starting at the top of list A, each character in A is sequentially compared to the first N characters in Hollerith array IHL.

If a match occurs, I is set to the character number in array IHL, and the head of list A will point to the element immediately following the matched character. The last element of list B will contain the matched character as an item and a link of zero.

If a match does not occur, and the end of list A is reached, I is set to N + 1, and list A will be null.

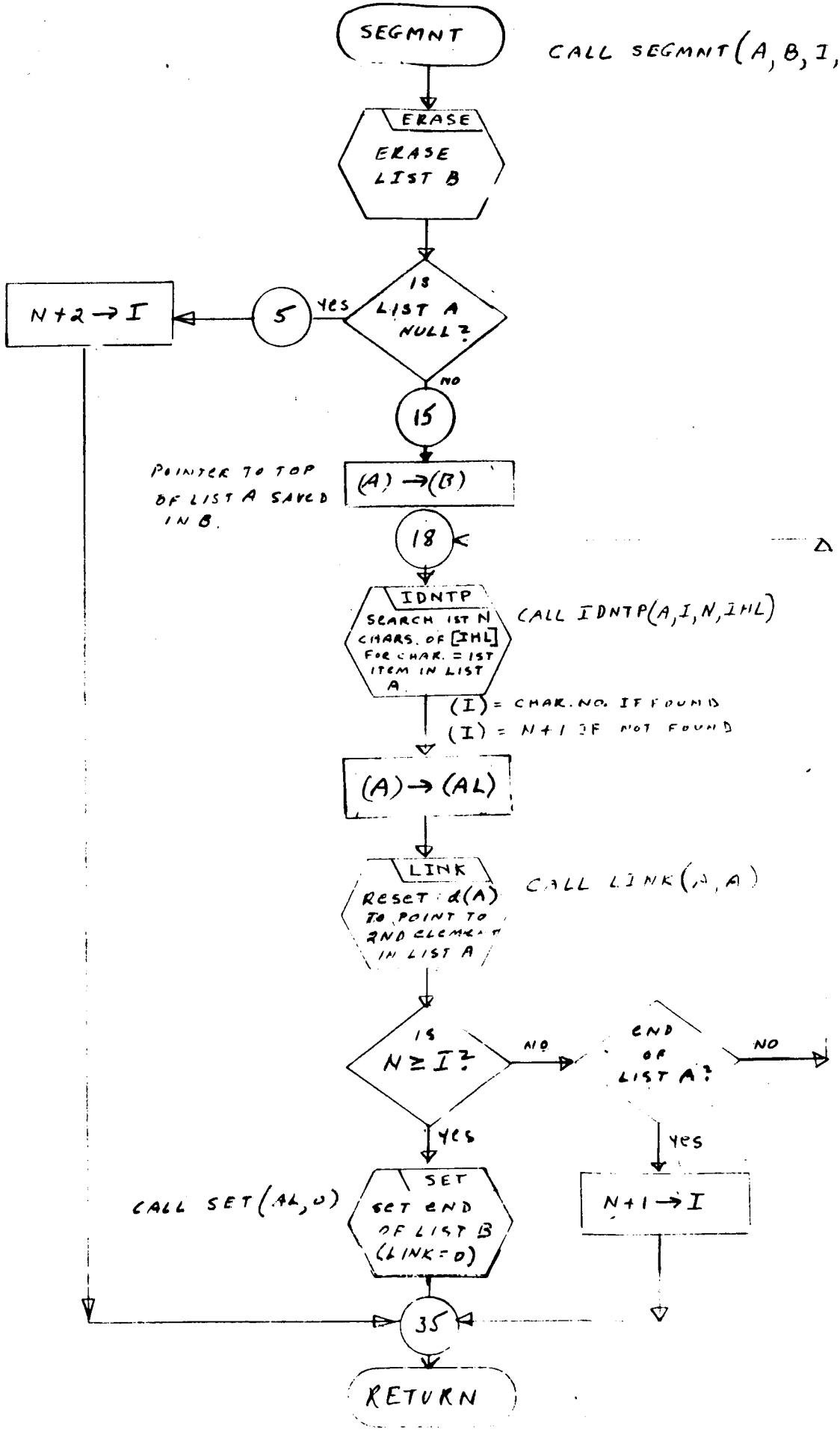
If list A is null upon entry to the routine, list B is erased, and I is set to N + 2.

7. Other Subroutines Used

ERASE, IDNTP, LINK, SET.

8. Using Subroutines

GOBLE, WRTEQ.


SEGMNT
CALL SEGMNT(A, B, I, N, IHL)
ERASE
ERASE LIST B
IS LIST A NULL?
yes
5
N+2 → I
NO
15
POINTER TO TOP OF LIST A SAVED IN B.
(A) → (B)
18
IDNTP
SEARCH 1ST N CHARS. OF [IHL] FOR CHAR. = 1ST ITEM IN LIST A
CALL IDNTP(A, I, N, IHL)
(I) = CHAR. NO. IF FOUND
(I) = N+1 IF NOT FOUND
(A) → (AL)
LINK
Reset d(A) TO POINT TO 2ND ELEMENT IN LIST A
CALL LINK(A, A)
IS N ≥ I?
NO
END OF LIST A?
NO
yes
yes
SET
SET END OF LIST B (LINK = 0)
CALL SET(AL, 0)
N+1 → I
35
RETURN

Program Description

1. Identification

   a. Routine Label

      SET

   b. Name

      Append one list to another list.

2. Function

The list IF is appended to list IX, such that the first element of list IF becomes the second element of list IX.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL SET (IX, IF)

   b. Entry Conditions

      IX = Head of a list
      IF = Head of a list

   c. Exit Conditions

      List IF is appended to the list IX.

   d. Error Exits

      None.

5. Definition of Identifiers

   None.

6. Method

The decrement of IF (pointer to a list) is shifted to the address portion of the accumulator, and then stored in the address of the location

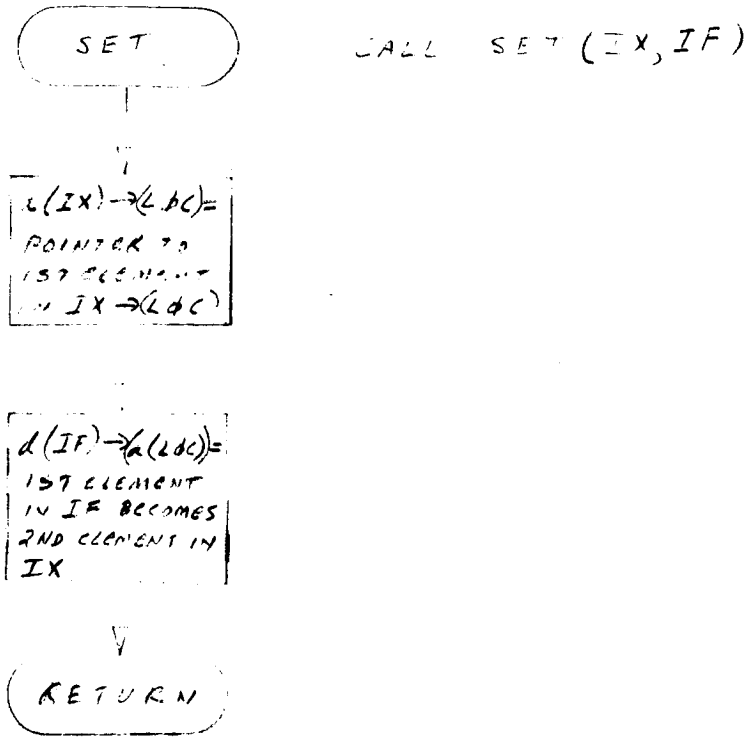
contained in the decrement of IX (pointer to a list with just one element).

7. Other Subroutines Used

None.

8. Using Subroutines

COPY, INSRT, SEGMNT, STASH.


SET
CALL SET (IX, IF)
c(IX)→(c.bc)=
POINTER TO
1ST ELEMENT
IN IX→(c.dc)
d(IF)→(a(c.bc))=
1ST ELEMENT
IN IF BECOMES
2ND ELEMENT IN
IX
RETURN

Program Description

1. Identification

   a. Routine Label

      SNATCH

   b. Name

      SNATCH a data word from an array of list structures.

2. Function

This subroutine searches an array of list structures for a data word identified by successive parameters. When found, this word is placed in FTEM. If the list is two-dimensional, the data is identified by I, J. If the list is four-dimensional, the data is identified by I, J, NN, and NP.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SNATCH (M, I, J, NN, NP, FTEM)

   b. Entry Conditions

      M = A two-dimensional or four-dimensional matrix represented in array-list form (type E)
      I = Index of the row lists
      J = Index of the column lists
      NN = Negative node index
      NP = Positive node index

If NN and NP are not given (0 in calling sequence), list M is a two-dimensional matrix. If NN and NP are given, list M is a four-dimensional matrix.

c. Exit Conditions

When M is a two-dimensional matrix:

(1) If J was found in an element whose flag = 6, the data word in the element linked to J is placed in FTEM.

(2) If J was found in an element whose flag $\neq$ 6, exit is made with FTEM = 0.

Where M is a four-dimensional matrix:

(1) If J, NN, and NP were found and the element containing NP has a flag = 6, the data word in the element linked to NP is placed in FTEM.

(2) If J was found, but not NN and NP, exit is made with FTEM = 0.

If M(I) was null on entry to the routine, or if J could not be found, exit is made with FTEM = 0.

d. Error Exits

None.

5. Definition of Identifiers

M(I) = Ith list structure of the array-list M

LPP = A multilevel list structure of array-list M

LJ = A temporary head cell that points to the sublist containing J or NP

LP = Item ((LJ))

JJ = Item ((LP))

NF = Flag ((LP))

6. Method

The search through array-list M to locate the data word is executed as outlined below.

a. Where M is two-dimensional, M(I) is scanned for J.

(1) If M(I) was null, or if J could not be found, exit is made with FTEM = 0.

(2) If J was found, item (LJ) = location of the element which points to the word containing J (or J sublist).

(3) Item ((LJ)) → d(LP), item ((LP)) → d(JJ), gives J in JJ. Flag ((LP)) → d(NF) gives the flag of the element containing J.

(4) Link ((LP)) → d(LPP) gives the location of the element appended to J.

(5) If the flag in d(NF) = 6, the element pointed to by d(LPP) contains a data word. This word is placed in FTEM and exit is made from the routine.

b. When M is a four-dimensional matrix, the scan for J in M(I) is made as described in steps a(1), a(2), a(3), and a(4). However, in this case, the location in d(LPP) is considered as an element which points to a sublist. This sublist (called LPP) is searched to locate the data word identified by successive parameters NN and NP.

(1) If NN and NP cannot be found, exit is made with FTEM = 0.

(2) If NN and NP were found, item (LJ) = location of the element which points to the word containing NP.

(3) Item ((LJ)) → d(LP), item ((LP)) → d(JJ), gives NP in JJ. Flag ((LP)) → d(NF) gives the flag of the element containing NP.

(4) Link ((LP)) → d(LPP) gives the location of the element appended to NP.

(5) If the flag in d(NF) = 6, the element pointed to by d(LPP) contains the data word identified by NN and NP. This word is placed in FTEM and exit is made from the routine.

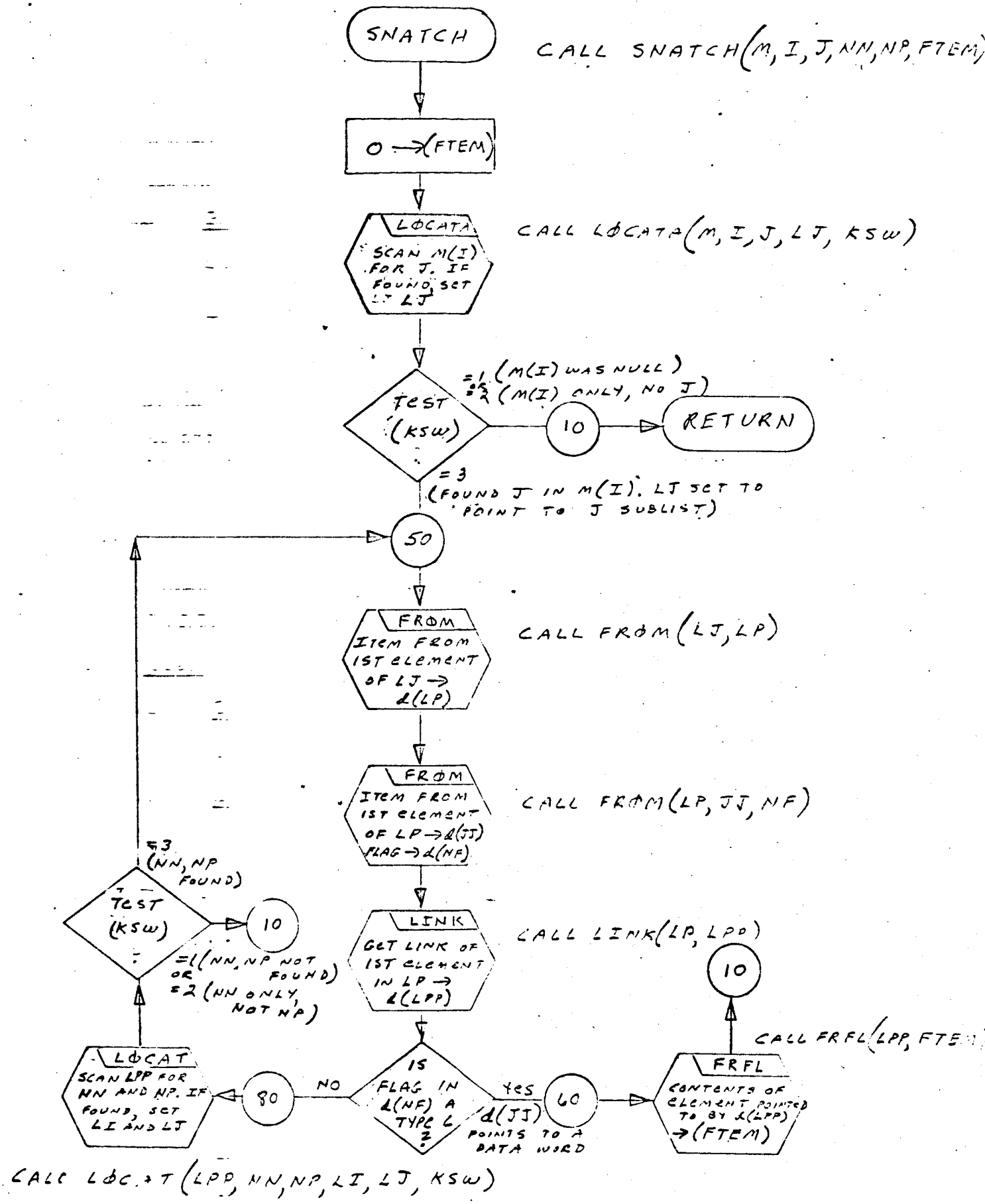
(6) If the flag in d(NF) ≠ 6, the scan is repeated in all sublists of LPP until NN and NP are found or until the end of LPP is reached (see steps b(1) through b(5)).

7. Other Subroutines Used

FRFL, FROM, LINK, LOCAT, LOCATA.

8. Using Subroutines

BAKELIM, COMBN, MULTS, STRIK, TRANS, XFORM.


SNATCH
CALL SNATCH(M, I, J, NN, NP, FTEM)
0 → (FTEM)
LOCATA
SCAN M(I) FOR J. IF FOUND, SET LJ
CALL LOCATA(M, I, J, LJ, KSW)
TEST (KSW)
=1 (M(I) WAS NULL)
=2 (M(I) ONLY, NO J)
10
RETURN
=3 (FOUND J IN M(I). LJ SET TO POINT TO J SUBLIST)
50
FROM
ITEM FROM 1ST ELEMENT OF LJ → L(LP)
CALL FROM(LJ, LP)
FROM
ITEM FROM 1ST ELEMENT OF LP → L(JJ) FLAG → L(NF)
CALL FROM(LP, JJ, NF)
=3 (NN, NP FOUND)
TEST (KSW)
10
=1 (NN, NP NOT FOUND) OR =2 (NN ONLY, NOT NP)
LINK
GET LINK OF 1ST ELEMENT IN LP → L(LPP)
CALL LINK(LP, LPP)
10
CALL FRFL(LPP, FTEM)
LOCAT
SCAN LPP FOR NN AND NP. IF FOUND, SET LI AND LJ
80
NO
IS FLAG IN L(NF) A TYPE 2
YES
L(JJ) POINTS TO A DATA WORD
60
FRFL
CONTENTS OF ELEMENT POINTED TO BY L(LPP) → (FTEM)
CALL LOCAT(LPP, NN, NP, LI, LJ, KSW)

Program Description

1. Identification

a. Routine Label

STASH

b. Name

STASH a data word into an array of list structures.

2. Function

An array of list structures is created to hold a data word identified uniquely by successive parameters. If the list is two-dimensional, the data is identified by I, J. If four-dimensional, the data is identified by I, J, NN, and NP.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

CALL STASH (M, I, J, NN, NP, FTEM)

b. Entry Conditions

M = A two-dimensional or four-dimensional matrix represented in array-list form (list type E)

I = Index of the row lists

J = Index of the column lists

NN = Negative node index

NP = Positive node index

If NN and NP are not given (0 in calling sequence), list M is two dimensional. If NN and NP are given, list M is considered a four-dimensional matrix.

c. Exit Conditions

If M(I) is null, a sublist M(I) is created which holds J, NN, NP, and FTEM. If NN and NP are not given in the call statement, a

data word containing FTEM is appended to an element containing J, forming M(I), a two-dimensional sublist. If NN and NP are given, FTEM is appended to NP, forming a four-dimensional sublist M(I).

The remaining exit conditions are separated into the two categories of matrices as follows:

Two-Dimensional Matrix

(1) If M(I) is not null, J was not found, and NFG has been set = 6 (NN + NP ≤ 0), J is inserted into M(I) with an element containing FTEM appended to it.

(2) If J was found in M(I) and NFG has been set = 6 (NN + NP ≤ 0), an element containing FTEM is appended to that element in which J was found.

Four-Dimensional Matrix

(1) If M(I) is not null, J was not found, and NFG = 0 (NN + NP > 0), J, NN, NP, and FTEM are inserted into M(I).

(2) If J was found in M(I), NFG = 0, and NN, NP were found in the J sublist, a data word containing FTEM is appended to the element containing NP.

(3) If J was found in M(I), NFG = 0, and NN was found but not NP, NP and FTEM are inserted into M(I).

(4) If J was found in M(I), NFG = 0, but NN, NP were not found in the J sublist, NN, NP, and FTEM are inserted into M(I).

d. Error Exits

None.

5. Definition of Identifiers

M(I) = Ith list structure of the array-list M

LJ = Contains location of element whose decrement points to J

LI = Contains location of element whose decrement points to I

JP = J sublist which is searched for NN and NP

NFG = A flag word which is tested to determine whether FTEM is to be appended to J or NP
NFG = 6 where NN + NP ≤ 0 (two dimensions given)
NFG = 0 where NN + NP > 0 (four dimensions given)

NWJ = Temporary head cell

NTJ = Location of element containing J

NTI = Location of element which points to J

NWNN = Temporary head cell of NN sublist

NWNP = Temporary head cell of NP sublist

NTNP = Location of element containing NP

IFTEM = Temporary head cell which points to an element containing FTEM

NTNN = Location of element containing NN

NTJJ = Location of element containing NTNN

LJT = Location of a type 7 element which points to an NP sublist

LJJ = Location of element whose decrement points to a J sublist

NPP = Location of element whose decrement points to an NP sublist

NPPP = Temporary head cell of a sublist whose first element contains FTEM

LIT = Location of type 7 element which points to an NN sublist

6. Method

a. The flag word NFG is initialized to zero.

b. (NN + NP) is tested:

(1) If ≤ 0, NN and NP were not given and NFG is set = 6. This flags the routine that the array-list M has two dimensions.

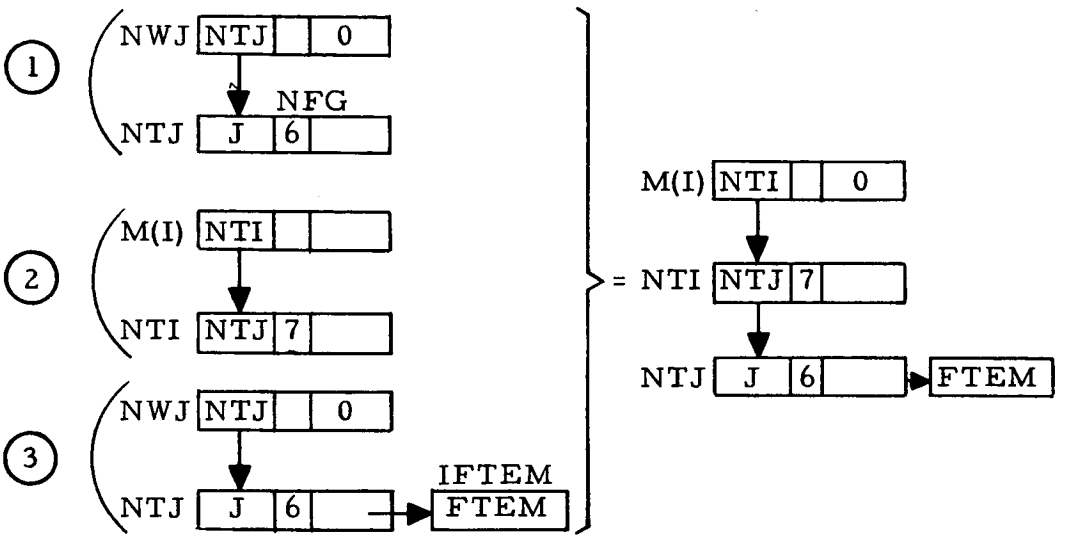
(2) If > 0, NN and NP were given and NFG = 0, flagging the routine that array-list M has four dimensions.

c. The following lists show, in graphic form, how the sublists are constructed for each possible condition. The circled numbers in each item represent, sequentially, the operations

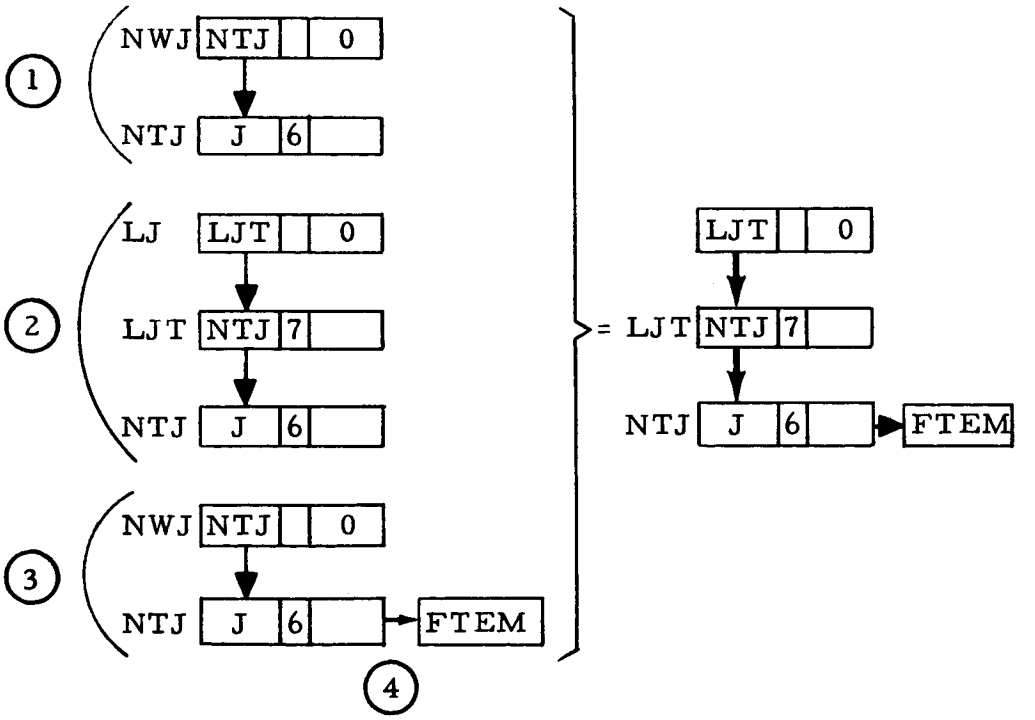
required to construct each sublist. The symbols used for each list correspond to those used in the routine.

d. Two-Dimensional Matrix

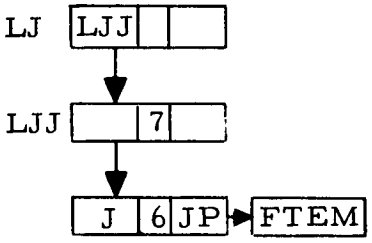
(1) M(I) Null, NFG ≠ 0



(2) No J in M(I), NFG ≠ 0

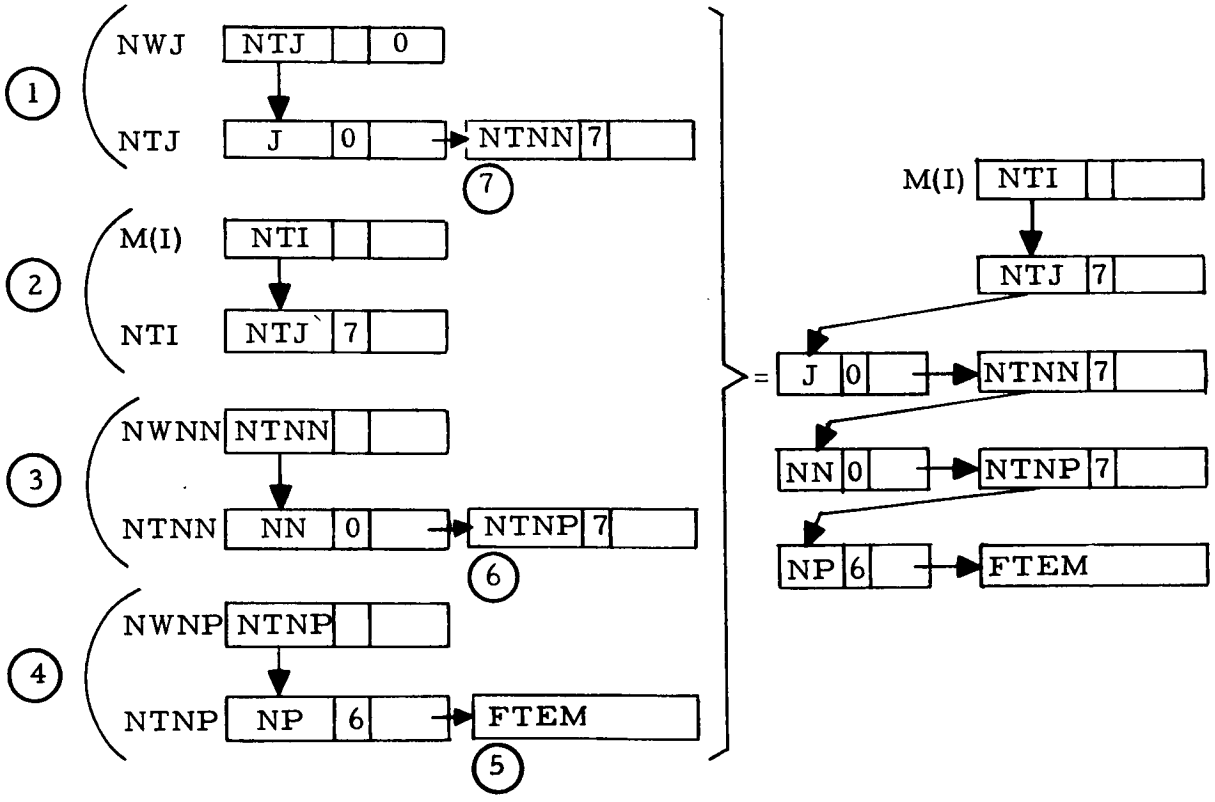


(3) J Found in M(I), NFG ≠ 0

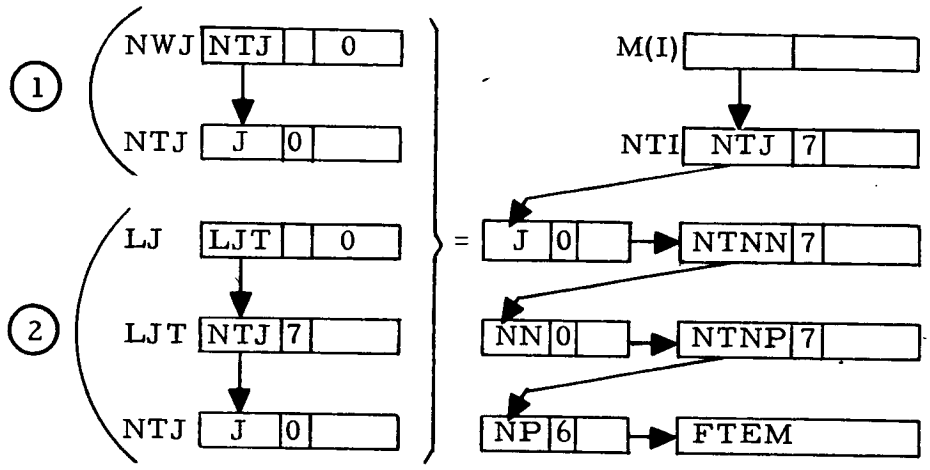


e. Four-Dimensional Matrix

(1) M(I) Null, NFG = 0

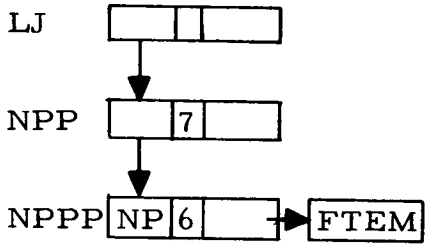


(2) No J in M(I), NFG = 0

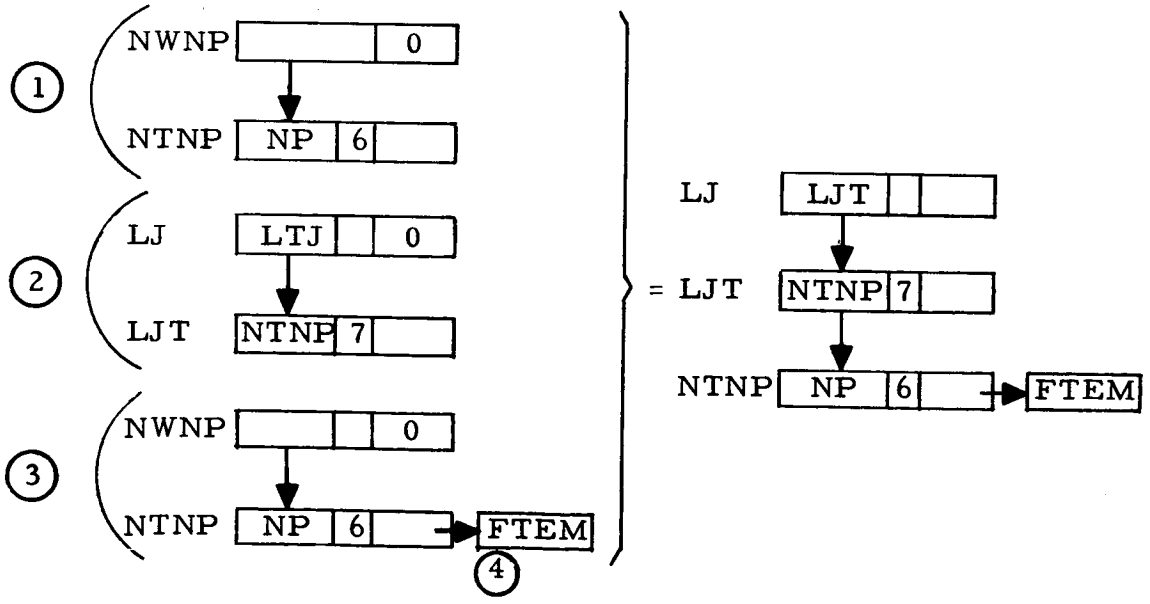


then (3) through (7) as shown in e(1) above.

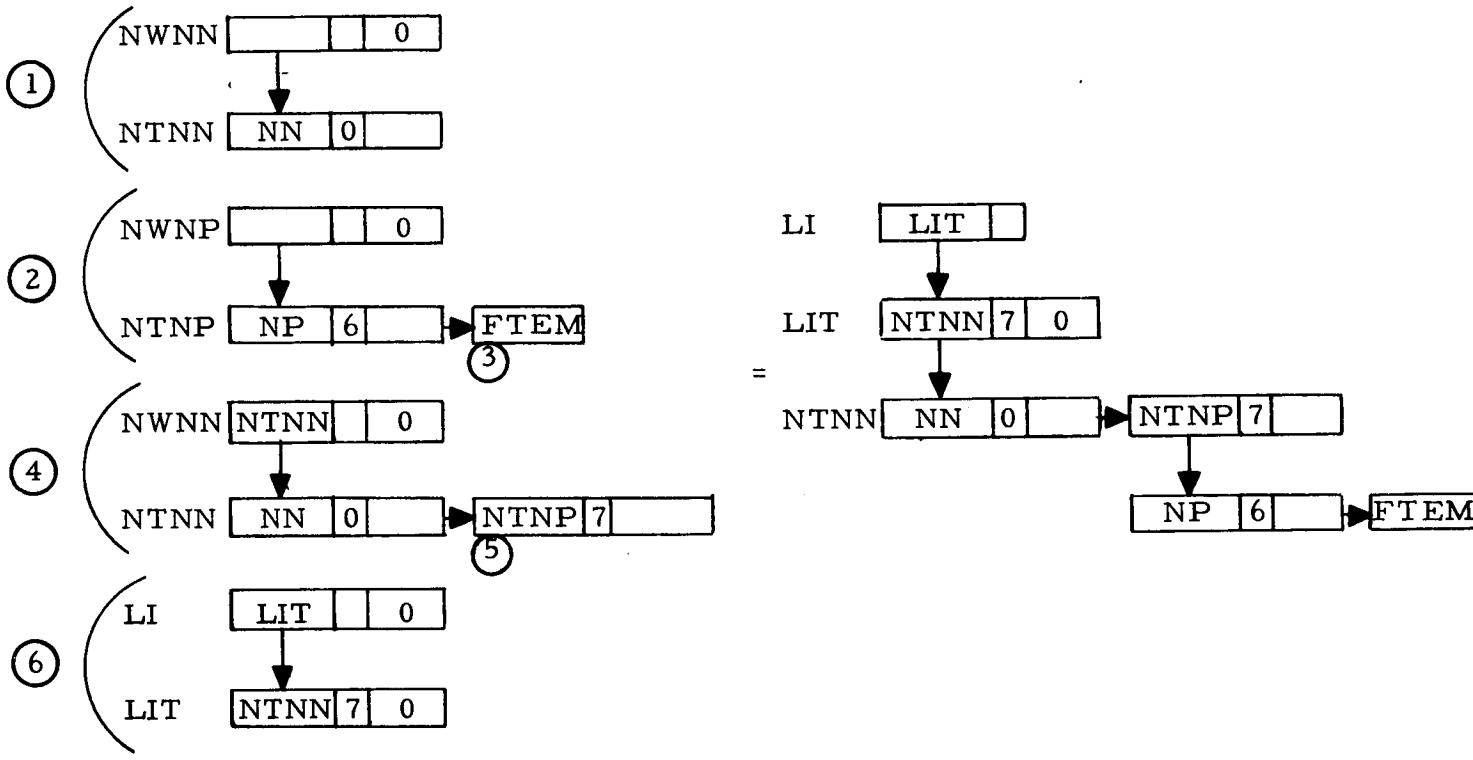
(3) J Found in M(I), NFG = 0, NN, NP Found in JP Sublist



(4) J Found in M(I), NFG = 0, NN Found in JP Sublist But Not NP



(5) J Found in M(I), NFG = 0, NN and NP Not Found in JP Sublist

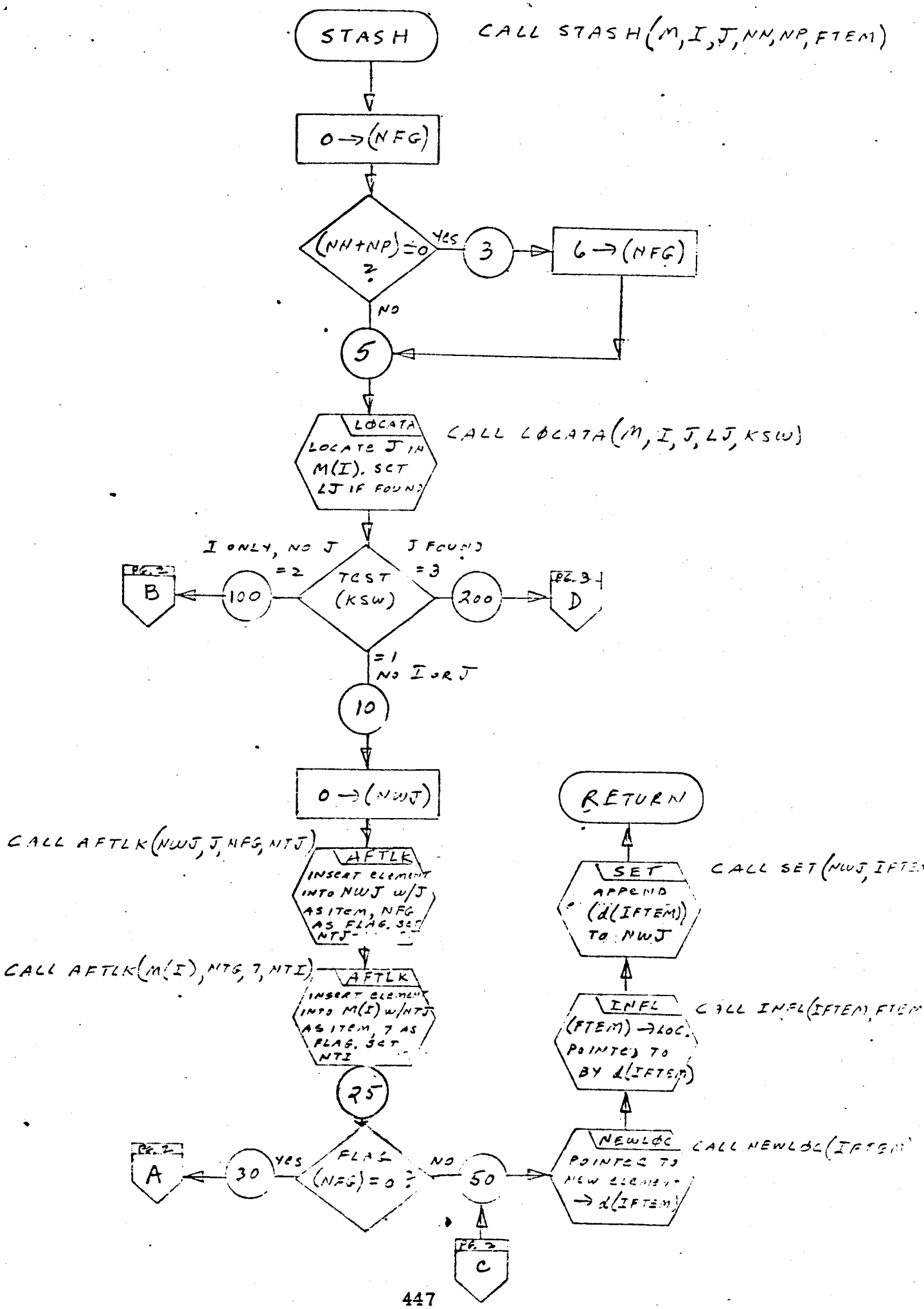


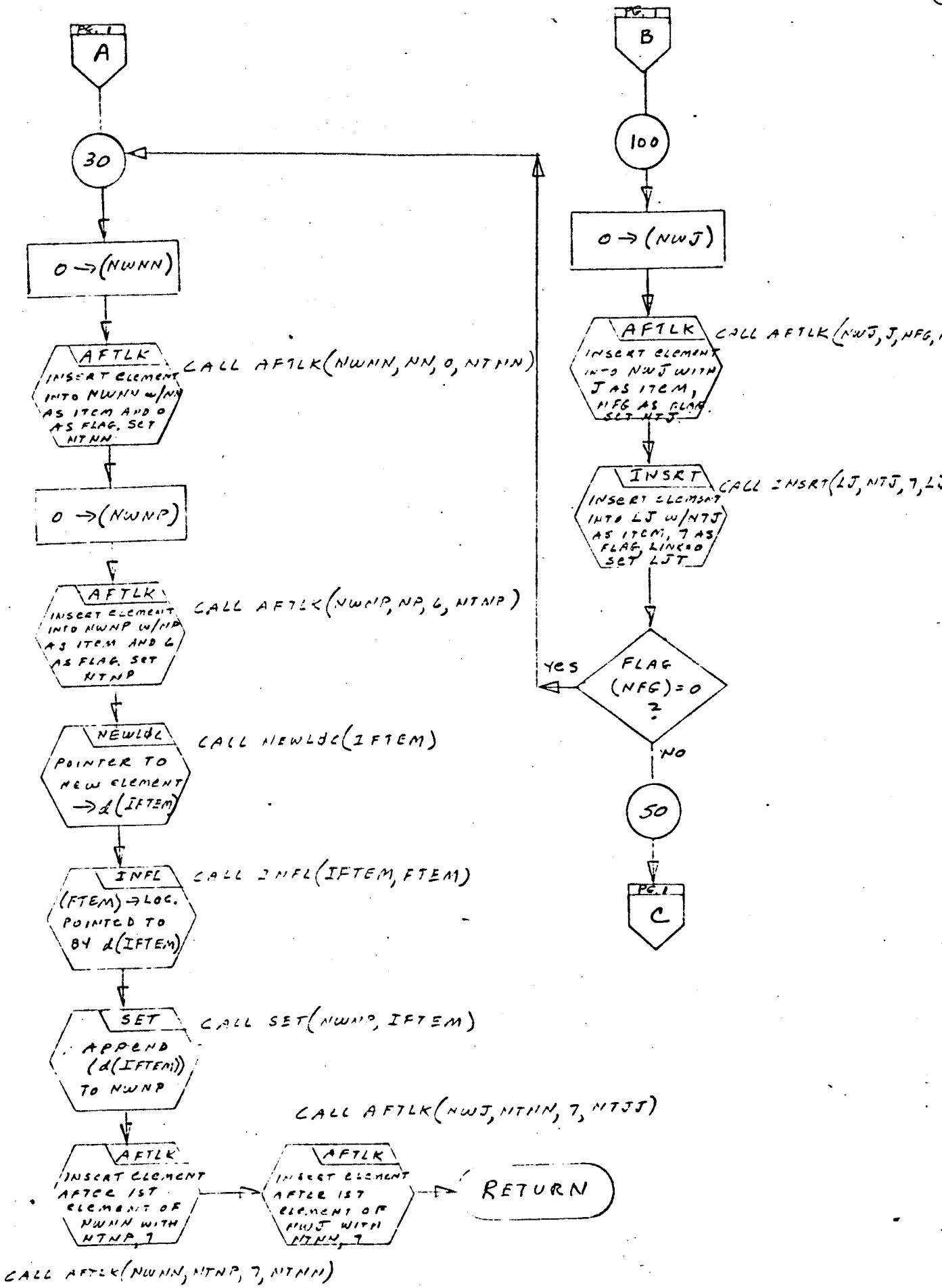
7. Other Subroutines Used

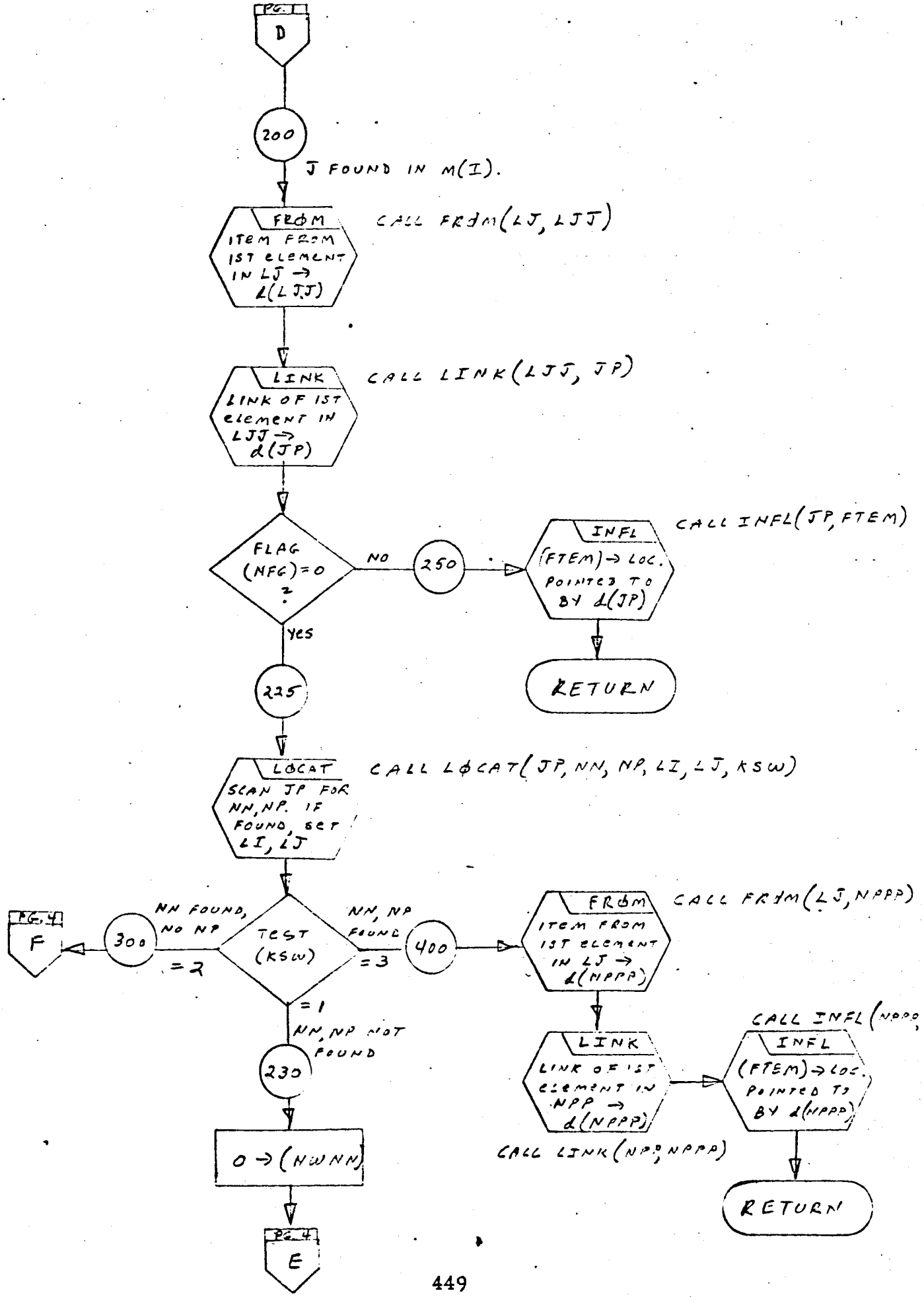
AFTLK, FROM, INFL, INSRT, LINK, LOCATA, LOCAT, NEWLOC, SET.

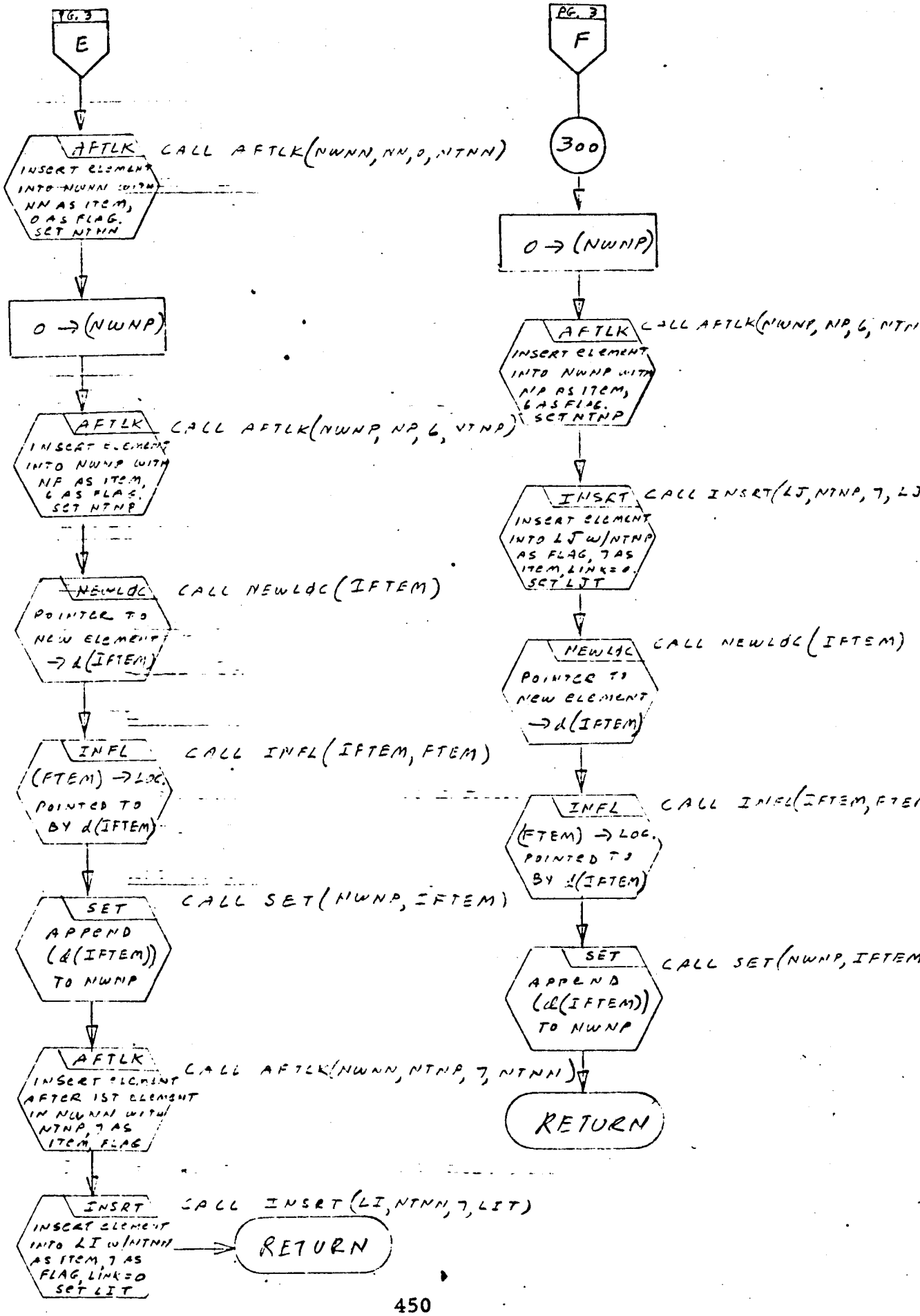
8. Using Subroutines

BAKELM, COTRN, MATFT, MULTS, PARAM, STRIK, TRANS, XFORM, Main Program for Pass 2 of TAG Preprocessor.


STASH
CALL STASH(M,I,J,NN,NP,FTEM)
0→(NFG)
(NN+NP)=0 ?
YES
3
6→(NFG)
NO
5
LOCATA
LOCATE J IN M(I). SET LJ IF FOUND
CALL LOCATA(M,I,J,LJ,KSW)
I ONLY, NO J
=2
TEST (KSW)
=3
J FOUND
PG. 3
B
100
200
PG. 3
D
=1
NO I OR J
10
0→(NWJ)
CALL AFTLK(NWJ,J,NFG,NTJ)
AFTLK
INSERT ELEMENT INTO NWJ W/J AS ITEM, NFG AS FLAG. SET NTJ
CALL AFTLK(M(I),NTG,7,NTI)
AFTLK
INSERT ELEMENT INTO M(I) W/NTJ AS ITEM, 7 AS FLAG. SET NTI
25
PG. 2
A
30
YES
FLAG (NFG)=0 ?
NO
50
PG. 2
C
NEWLOC
POINTER TO NEW ELEMENT → L(IFTEM)
CALL NEWLOC(IFTEM
INFL
(FTEM)→LOC. POINTED TO BY L(IFTEM)
CALL INFL(IFTEM,FTEM
SET
APPEND (L(IFTEM)) TO NWJ
CALL SET(NWJ,IFTEM
RETURN


PG. 1
A
30
0 → (NWNN)
AFTLK
INSERT ELEMENT INTO NWNN W/NN AS ITEM AND 0 AS FLAG. SET NTNN
CALL AFTLK(NWNN, NN, 0, NTNN)
0 → (NWNP)
AFTLK
INSERT ELEMENT INTO NWNP W/NP AS ITEM AND 6 AS FLAG. SET NTNP
CALL AFTLK(NWNP, NP, 6, NTNP)
NEWLOC
POINTER TO NEW ELEMENT → d(IFTEM)
CALL NEWLOC(IFTEM)
INFL
(FTEM) → LOC. POINTED TO BY d(IFTEM)
CALL INFL(IFTEM, FTEM)
SET
APPEND (d(IFTEM)) TO NWNP
CALL SET(NWNP, IFTEM)
CALL AFTLK(NWJ, NTNN, 7, NTJJ)
AFTLK
INSERT ELEMENT AFTER 1ST ELEMENT OF NWNN WITH NTNP, 7
AFTLK
INSERT ELEMENT AFTER 1ST ELEMENT OF NWJ WITH NTNN, 7
RETURN
CALL AFTLK(NWNN, NTNP, 7, NTNN)
PG. 1
B
100
0 → (NWJ)
AFTLK
INSERT ELEMENT INTO NWJ WITH J AS ITEM, NFG AS FLAG. SET NTJ
CALL AFTLK(NWJ, J, NFG,
INSRT
INSERT ELEMENT INTO LJ W/NTJ AS ITEM, 7 AS FLAG, LINK=0 SET LJT
CALL INSRT(LJ, NTJ, 7, LJ
FLAG (NFG) = 0 ?
YES
NO
50
PG. 1
C


PG. D
200
J FOUND IN M(I).
FROM
ITEM FROM 1ST ELEMENT IN LJ → L(LJJ)
CALL FRØM(LJ, LJJ)
LINK
LINK OF 1ST ELEMENT IN LJJ → L(JP)
CALL LINK(LJJ, JP)
FLAG (NFG) = 0 ?
NO
250
INFL
(FTEM) → LOC. POINTED TO BY L(JP)
CALL INFL(JP, FTEM)
RETURN
YES
225
LOCAT
SCAN JP FOR NN, NP. IF FOUND, SET LI, LJ
CALL LØCAT(JP, NN, NP, LI, LJ, KSW)
TEST (KSW)
NN FOUND, NO NP
=2
300
PG. 4 F
NN, NP FOUND
=3
400
FROM
ITEM FROM 1ST ELEMENT IN LJ → L(NPPP)
CALL FRØM(LJ, NPPP)
LINK
LINK OF 1ST ELEMENT IN NPP → L(NPPP)
CALL LINK(NPP, NPPP)
INFL
(FTEM) → LOC. POINTED TO BY L(NPPP)
CALL INFL(NPPP, FTEM)
RETURN
=1
NN, NP NOT FOUND
230
0 → (NWNN)
PG. 4 E


PG. 3
E
AFTLK
INSERT ELEMENT INTO NWNN WITH NN AS ITEM, 0 AS FLAG. SET NTNN
CALL AFTLK(NWNN, NN, 0, NTNN)
0 → (NWNP)
AFTLK
INSERT ELEMENT INTO NWNP WITH NP AS ITEM, 6 AS FLAG. SET NTNP
CALL AFTLK(NWNP, NP, 6, NTNP)
NEWLOC
POINTER TO NEW ELEMENT → d(IFTEM)
CALL NEWLOC(IFTEM)
INFL
(FTEM) → LOC. POINTED TO BY d(IFTEM)
CALL INFL(IFTEM, FTEM)
SET
APPEND (d(IFTEM)) TO NWNP
CALL SET(NWNP, IFTEM)
AFTLK
INSERT ELEMENT AFTER 1ST ELEMENT IN NWNN WITH NTNP, 7 AS ITEM, FLAG
CALL AFTLK(NWNN, NTNP, 7, NTNN)
INSRT
INSERT ELEMENT INTO LI W/NTNN AS ITEM, 7 AS FLAG, LINK=0 SET LIT
CALL INSRT(LI, NTNN, 7, LIT)
RETURN
PG. 3
F
300
0 → (NWNP)
AFTLK
INSERT ELEMENT INTO NWNP WITH NP AS ITEM, 6 AS FLAG. SET NTNP
CALL AFTLK(NWNP, NP, 6, NTN
INSRT
INSERT ELEMENT INTO LJ W/NTNP AS FLAG, 7 AS ITEM, LINK=0. SET LJT
CALL INSRT(LJ, NTNP, 7, LJ
NEWLOC
POINTER TO NEW ELEMENT → d(IFTEM)
CALL NEWLOC(IFTEM)
INFL
(FTEM) → LOC. POINTED TO BY d(IFTEM)
CALL INFL(IFTEM, FTE
SET
APPEND (d(IFTEM)) TO NWNP
CALL SET(NWNP, IFTEM
RETURN

Program Description

1. Identification

   a. Routine Label

      STAT

   b. Name

      Output characters of a Hollerith array as a card image.

2. Function

N characters of H, a Hollerith variable, are output as a card image with the statement number IS.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL STAT (IS, N, H)

   b. Entry Conditions

      IS = Statement number
      N = Number of characters in Hollerith array H
      H = Hollerith array

   c. Exit Conditions

N characters of H are output as a series of card images with the statement number IS in columns 1-5. Output is a punch tape, a print tape and a save tape.

   d. Error Exits

      None.

5. Definition of Identifiers

IWRK = A temporary cell used to head a list containing N characters of H after the first element.

6. Method

a. N characters of H are placed in list IWRK, immediately after element 1.

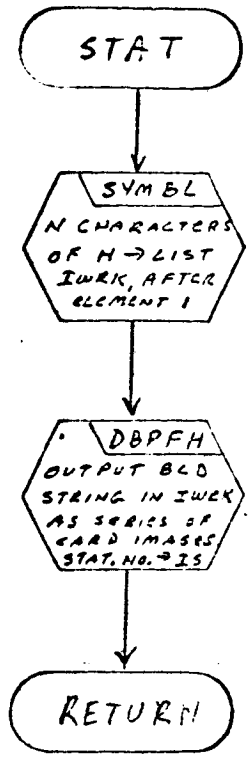
b. The list IWRK is output onto a print tape, a punch tape, and a save tape, as a series of card images, with the statement number IS in columns 1-5. Continuation cards, if any, will have a continuation card number in column 6.

7. Other Subroutines Used

DBPFH, SYMBL.

8. Using Subroutines

DIMEN, INVST, ZEROX, Main Program for Pass 2 of TAG Preprocessor.



CALL STAT(IS, N, H)

CALL SYMBL(IWRK, N, H)

CALL DBPFH(IWRK, IS)

IWRK = LIST OF BCD CHARACTERS IN ORDER.

IS = STATEMENT NO. IN COLS. 1-5

Program Description

1. Identification

   a. Routine Label

      STATC

   b. Name

      Extract the first word from a card image buffer.

2. Function

If the card image in buffer A is not a comment card, the statement number is extracted, converted to an integer and stored in IS. If the card image in A is a continuation card, IX is set to one. If it is not a continuation card, IX is set to two. If the card image in A is a comment card, the statement number is ignored, IX is set to one, and the character C is stored in IC.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL STATC (A, IS, IX, IS)

   b. Entry Conditions

      A = Card image buffer of 13 BCD words

   c. Exit Conditions

      If A does not contain a comment card,
      IS = statement number from A as an integer
      IX = 1 if not a continuation card
      IX = 2 if a continuation card
      IC = character in column 1
      If A does contain a comment card,
      IC = the character "C"

d. Error Exits

If a non-numeric statement number is encountered during the BCD-to-binary conversion of the statement number in A, an error comment "NON-NUMERIC STATEMENT NUMBER" is printed. The accumulator is cleared to zero and exit is made from the routine.

5. Definition of Identifiers

None.

6. Method

The following steps are performed to extract to first word from buffer A:

a. Column 6 is extracted and tested.
   (1) If it is not a zero or a blank, the card image in A is a continuation card and IX is set to 2.
   (2) If it is a zero or blank, the card image in A is not a continuation card, and IX is set to 1.

b. Column 1 is extracted and tested.
   (1) If the character in column 1 is a "C," the card image in A is a comment card. "C" is placed in IC, IX is set to one, the accumulator is cleared to zero, and exit is made from the routine.
   (2) If the character in column 1 is not a "C," the character is stored in IC.

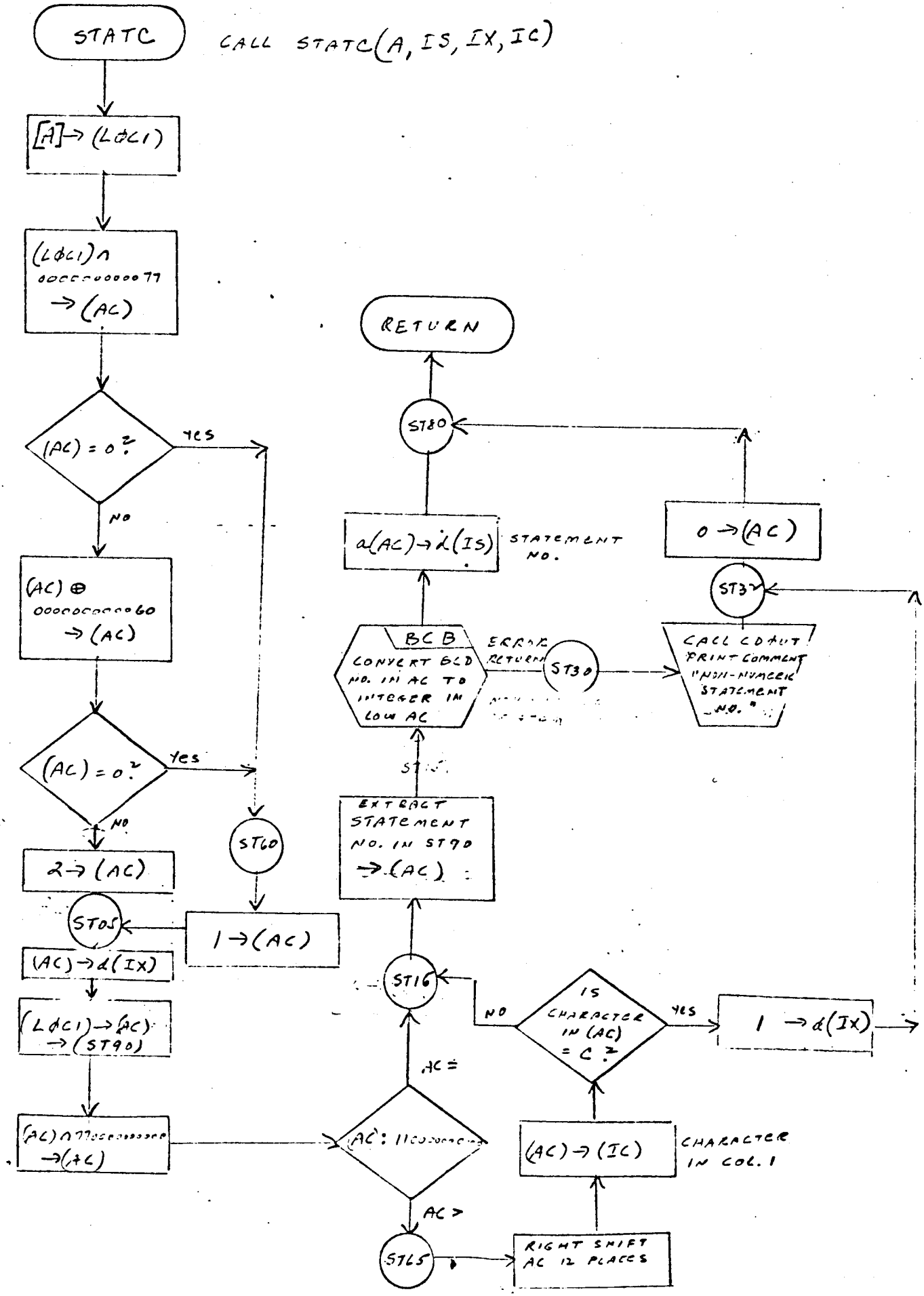
c. The statement number in A is extracted, converted to BCD and stored in IS, then exit is made from the routine.
   (1) If a non-numeric statement number is found, an error comment is printed, the accumulator is cleared to zero, and exit is made from the routine.

7. Other Subroutines Used

BCB

8. Using Subroutines

READCH


STATC
CALL STATC(A, IS, IX, IC)
[A] → (LOC1)
(LOC1) ∧ 000000000077 → (AC)
(AC) = 0 ?
YES
NO
(AC) ⊕ 000000000060 → (AC)
(AC) = 0 ?
YES
NO
2 → (AC)
ST05
ST60
1 → (AC)
(AC) → d(IX)
(LOC1) → (AC) → (ST90)
(AC) ∧ 777000000000 → (AC)
AC:
AC =
AC >
ST16
ST15
EXTRACT STATEMENT NO. IN ST90 → (AC)
BCB
CONVERT BCD NO. IN AC TO INTEGER IN LOW AC
ERROR RETURN
ST30
CALL COMMENT PRINT COMMENT "NON-NUMERIC STATEMENT NO. "
ST37
0 → (AC)
a(AC) → d(IS)
STATEMENT NO.
ST80
RETURN
IS CHARACTER IN (AC) = C ?
NO
YES
1 → d(IX)
(AC) → (IC)
CHARACTER IN COL. 1
RIGHT SHIFT AC 12 PLACES

Program Description

1. Identification

   a. Routine Label

      STATUS

   b. Name

      Print STATUS of AVS.

2. Function

   This subroutine prints the current and maximum count of elements used from AVS.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      CALL STATUS

   b. Entry Conditions

      None.

   c. Exit Conditions

      STATUS of AVS (maintained in COUNT and MAX) is printed.

   d. Error Exits

      None.

5. Definition of Identifiers

   COUNT = Number of elements used from AVS
   MAX = Max count of elements used from AVS

6. Method

   The subroutine CDOUT (a system routine) is called to print the contents of COUNT and MAX.

7. Other Subroutines Used

CDOUT (System Routine).

8. Using Subroutines

Main Programs for Pass 1 and Pass 2 of TAG Preprocessor.

Program Description

1. Identification

   a. Routine Label

      STRIK

2. Function

From the TQ matrix as transformed by BAKELM, STRIK produces an NM x NMR (NMR = NM - NRR) matrix, TL, which relates the unconstrained tree voltage coordinate vector, VT, to the NMR independent coordinate variables, VTI, of VT under the constraint of NRR transformer equations. In addition, STRIK extracts the independent set of coordinate variables, VTI, from FLIST in which the VT vector is stored and inserts them, in order, in the list VC. Finally, STRIK counts the number of each type of element in VC and stores these partition indices in the NPT array such that NPT(NE) = the number of type NE elements in VC. The resulting matrix relationship is VT = TL * VC.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL STRIK (TQ, TL, PLIST, VC, NPT, NRR, NM, NMR)

   b. Entry Conditions

      TQ = The equivalent transformer constraint matrix as output from BAKELM

      PLIST = A local name for FLIST which is a two-dimensional list representation of the unconstrained tree voltage vector

NPT = An array which stores the number of branches of each element type that make up the unconstrained tree in PLIST

NRR = The number of transformer constraint equations; is equal to the total number of transformer windings less the number of transformers

NM = The number of branches in the unconstrained tree

c. Exit Conditions

TL = A two-dimensional list type E which represents the NM x NMR coefficient matrix that expresses the NM unconstrained tree voltage variables in terms of the NMR members of the final constrained tree voltage vector VC

VC = A two-dimensional list containing the NMR members of the final constrained tree voltage vector

NPT = An array which stores the number of branches of each element type that make up the final constrained tree voltage vector listed in VC. NPT(NE) = the number of type NE elements in VC

NMR = The number of variables listed in VC; is equal to NM - NRR

d. Error Exits

(1) If a row of TQ is found in which the first non-zero element on the right end is not unity or in which there is no non-zero element, a dump is called and control is returned to the FORTRAN Monitor system.

(2) If the index of a pivot column in TQ does not correspond to a member of PLIST, CALL DUMP and return control to the FORTRAN Monitor system.

5. Definition of Identifiers

T is an NM x NM matrix used in formation of TL and is in type E array list format.

I, J, and K are used as indices for addressing individual elements of the TQ, T, and TL matrices.

FTEM, FTEM1, FTEM2, and FTEM3 are used as temporary names for the elements of the various matrices being manipulated.

ST is a simple list which stores the pivot column numbers of TQ in right-to-left order.

X1 is used as a temporary head cell for searching PLIST.

X2 is used as a temporary head cell for searching ST.

NMR counts the number of columns generated for TL.

VV is the two-dimensional list type D in which the final NMR members of the constrained tree voltage vector are listed in reverse order.

VC is the final constrained tree voltage vector listed in proper order in type D format.

6. Method

STRIK forms the matrix TL such that VT = TL * VC, where VT is the unconstrained tree voltage coordinate vector stored in PLIST (the local name for FLIST) and VC is the list which stores the independent members of VT under the transformer constraint equations TQ * VT = 0. TL is formed by removing from an auxiliary NM x NM matrix T, which is formed such that T * VT = VT, the NRR zero columns which correspond to the dependent members of VT. The formulation of matrix T implies the expansion and rearrangement of the TQ matrix to an NM x NM matrix TQ' in which the rows of TQ are placed such that their pivot elements fall along the principal diagonal. All other rows of TQ' are zero. This rearrangement does not alter the original set of constraints, and TQ' * VT = 0 where the zero vector is now of dimension NM. T is formed by subtracting TQ' from an NM x NM identity matrix I such that T = I - TQ'. The equality T * VT = VT is obvious from the fact that I * VT = VT and TQ' * VT = 0. The NRR columns of T corresponding to the dependent members of VT will be zero, and T * VT may be reduced to TL * VC by removing the zero columns of T to form TL and

removing the corresponding dependent members of VT to form VC. The portion index array NPT(NE), which previously held the number of each type of element in VT, is reset to reflect the contents of the reduced tree, VC. The following procedure describes the mechanization of STRIK.

a. Erase lists TL, T, ST, VC, and VV.

b. Create T as an identity matrix of dimension NM x NM.

c. Set the row search index, I, for TQ to 1. Set the column search index, J, for TQ to NM.

d. Search row I of TQ starting at J = NM (the right-hand side) for the pivot element. If no pivot element is found or if its value is not unity, call for a dump and return to the FORTRAN Monitor system. For a pivot element found in row I and column J subtract the Ith row of TQ from the Jth row of T. Store the pivot column number, J, in list ST. If I does not equal NRR, add one to it, reset J to NM, and repeat step d. If I equals NRR go on to step e.

e. Set the column transfer index, I, of T to 1 and the column insertion index, NMR, of TL to 0.

f. Test column index I to see if it corresponds to any of the pivot column numbers in list ST. If not, add one to the column index, NMR, of TL and replace column NMR of TL with column I of T. Proceed to step h.

g. If I does correspond to a member of list ST, then column I in T is zero and corresponds to a dependent member in the tree voltage vector stored in PLIST. This member of PLIST is flagged with a 1. If no member of PLIST corresponds to I, CALL DUMP.

h. Test I, If I is not equal to NM, I is increased by one and the process returns to step f. If I is equal to NM, the process continues to step i.

i. Lists ST and T are erased, and array NPT(NE) is set equal to zero for NE = 1, 2, ---7.

j. List VV is formed in reverse order from all the unflagged members of PLIST, and a new NPT(NE) array is computed such that NPT(NE) is equal to the number of elements in VV of type NE.

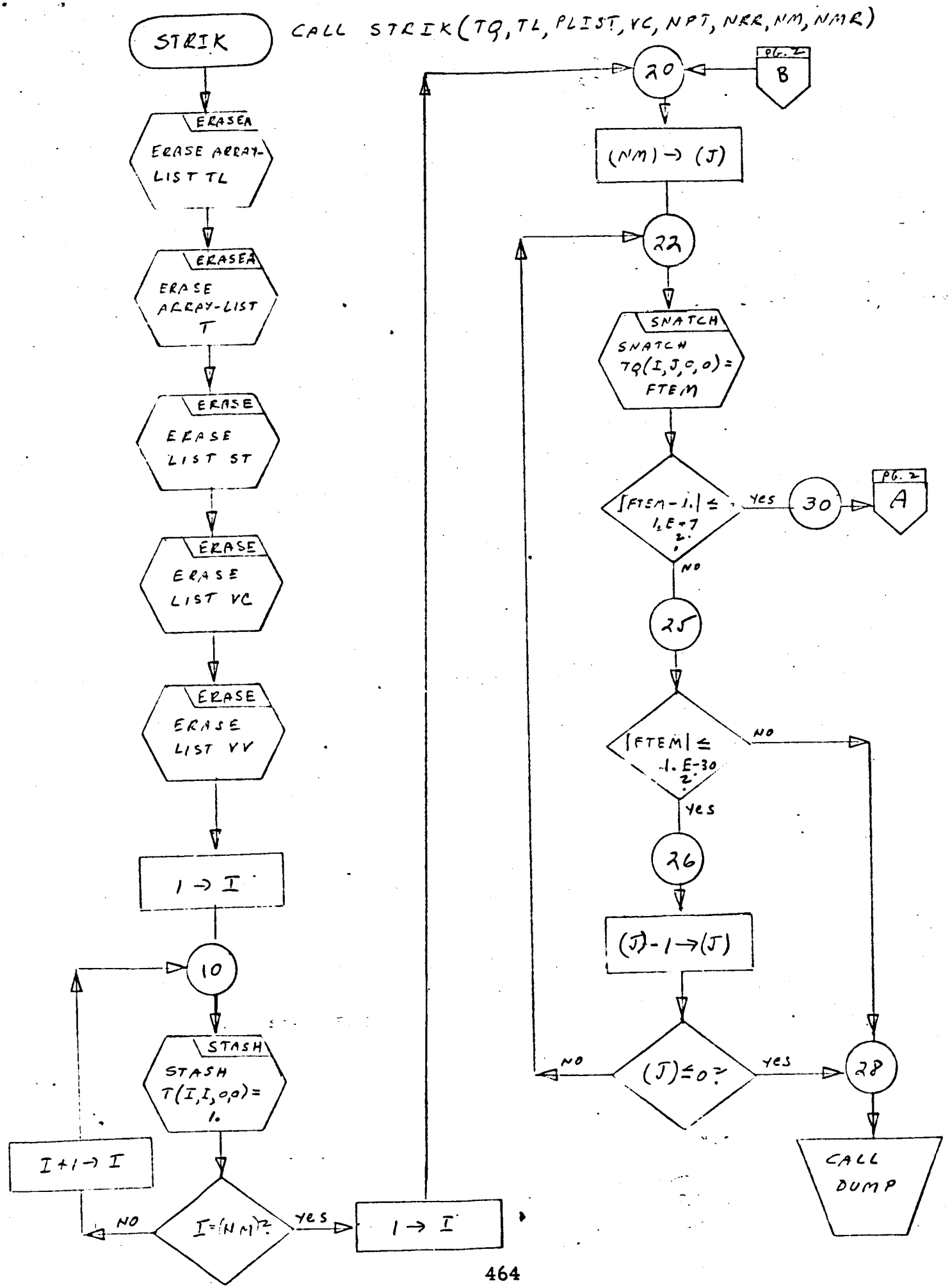
k. VC is formed by reversing the order of VV and return is made to the main program.

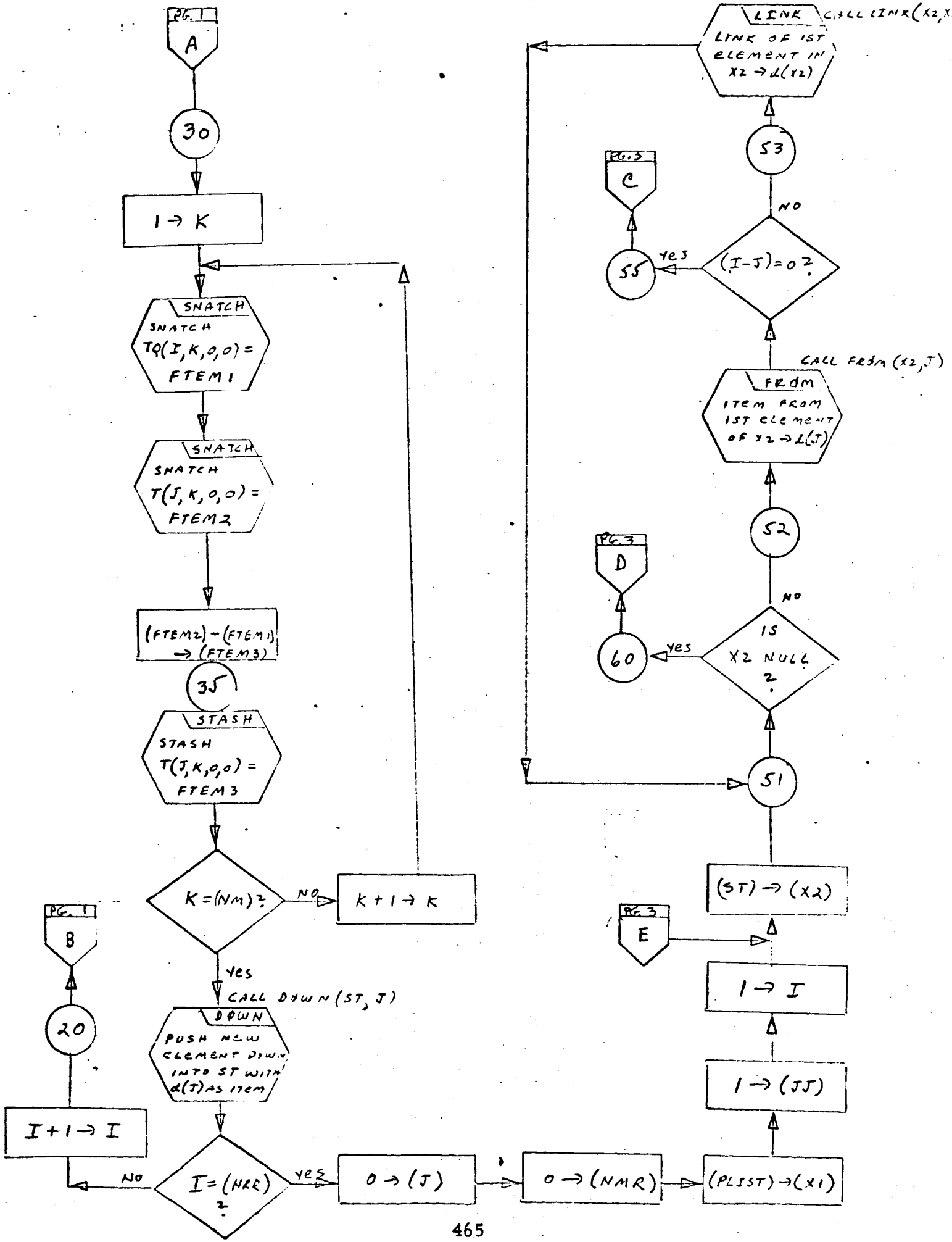
7. Other Subroutines Used

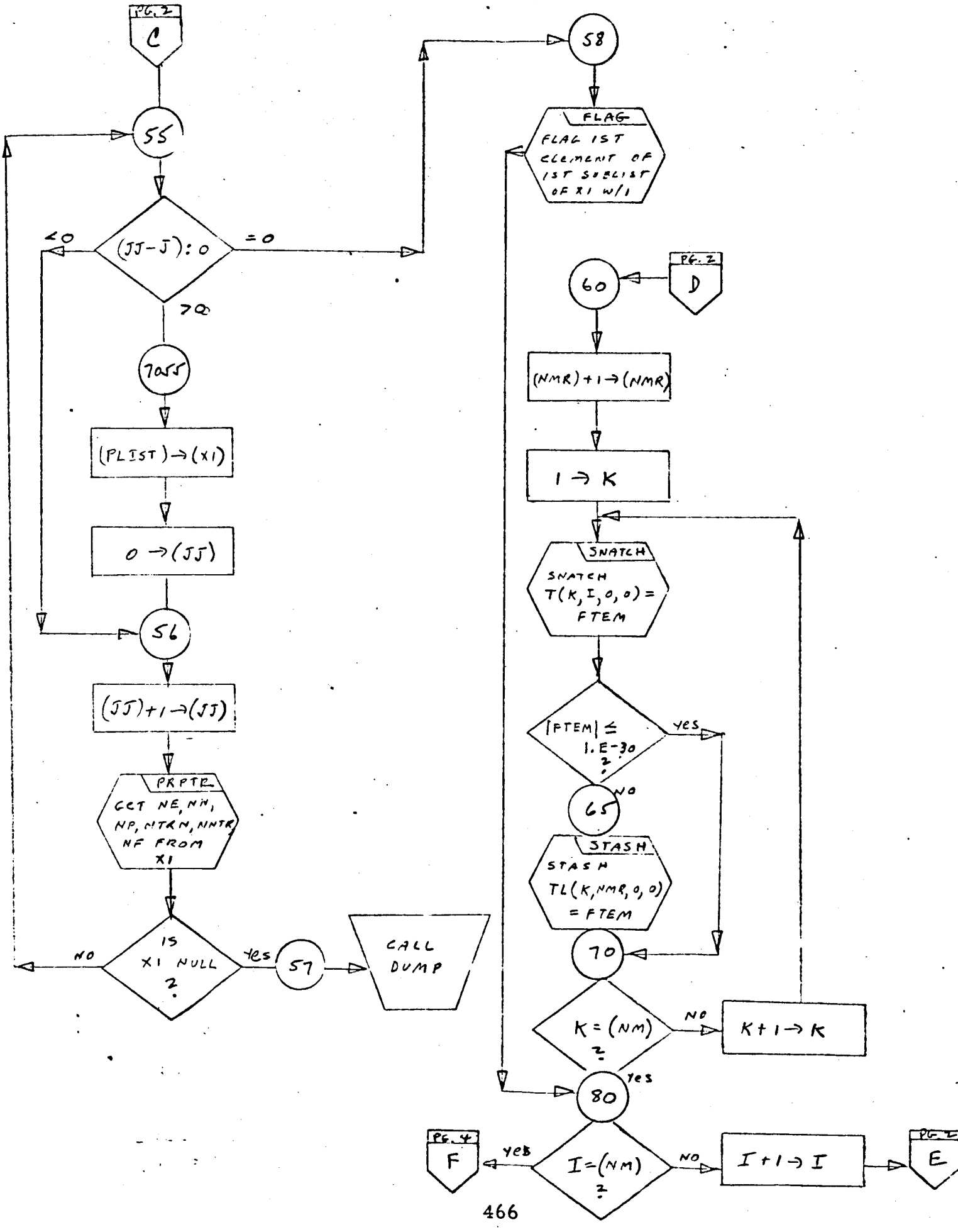
DIFA, DOWN, DUMP, ERASEA, ERASE, FLAG, FROM, LINK, PRPTG, PRPTR, SNATCH, STASH, UPDOWN.

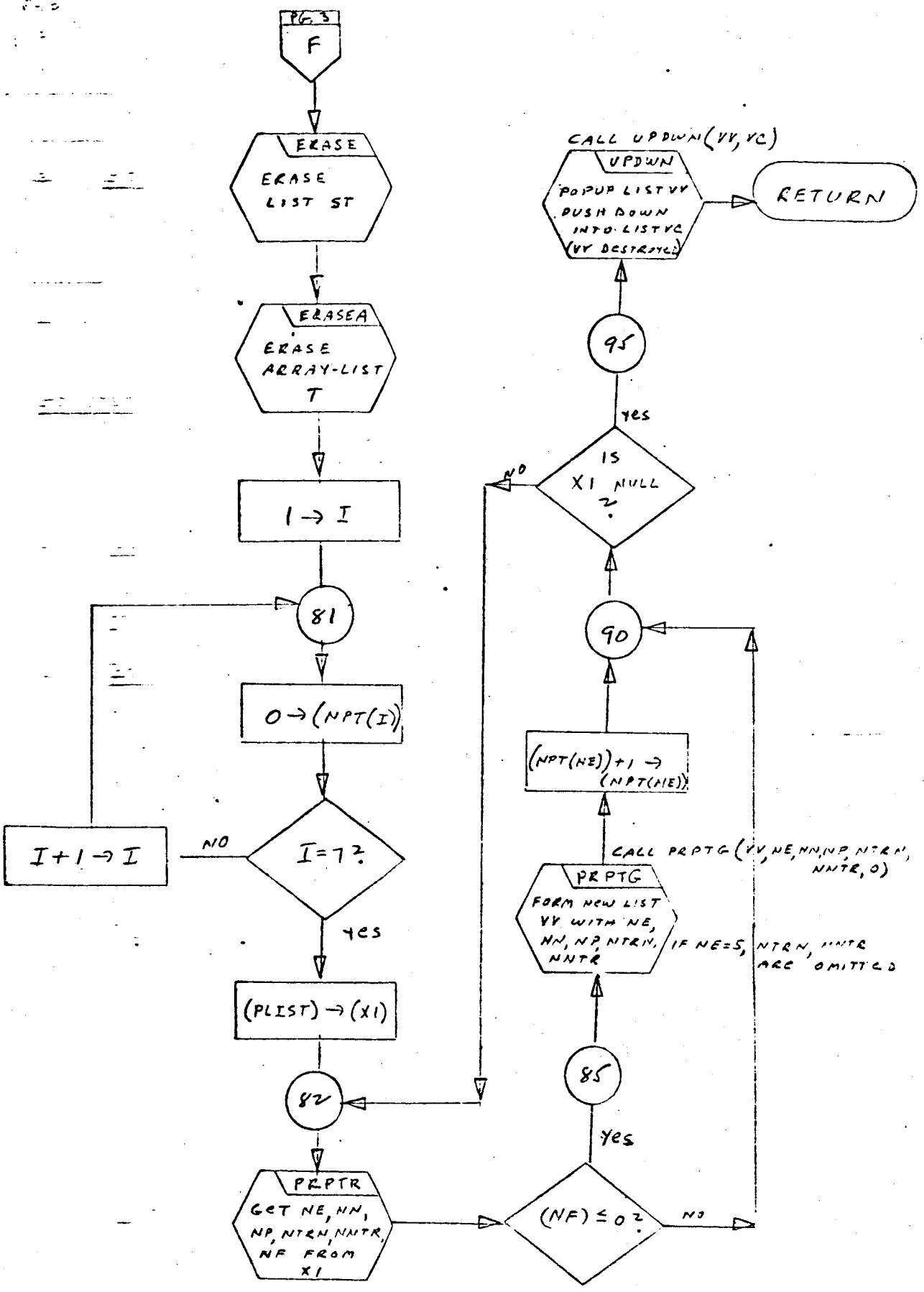
8. Using Subroutines

Main Program for Pass 1 of TAG Preprocessor.


STRIK
CALL STRIK(TQ, TL, PLIST, VC, NPT, NRR, NM, NMR)
ERASEA
ERASE ARRAY-LIST TL
ERASEA
ERASE ARRAY-LIST T
ERASE
ERASE LIST ST
ERASE
ERASE LIST VC
ERASE
ERASE LIST VV
1 → I
10
STASH
STASH T(I,I,0,0) = 1.
I+1 → I
NO
I=(NM)?
yes
1 → I
20
pg. 2
B
(NM) → (J)
22
SNATCH
SNATCH TQ(I,J,0,0) = FTEM
|FTEM-1.| ≤ 1.E-7 ?
yes
30
pg. 2
A
NO
25
|FTEM| ≤ 1.E-30 ?
NO
yes
26
(J)-1→(J)
(J)≤0?
NO
yes
28
CALL DUMP


PG. 1
A
30
1 → K
SNATCH
SNATCH TQ(I,K,0,0) = FTEM1
SNATCH
SNATCH T(J,K,0,0) = FTEM2
(FTEM2) - (FTEM1) → (FTEM3)
35
STASH
STASH T(J,K,0,0) = FTEM3
K = (NM) ?
NO
K + 1 → K
yes
CALL DOWN (ST, J)
DOWN
PUSH NEW ELEMENT DOWN INTO ST WITH d(J) AS ITEM
I = (NRR) ?
yes
0 → (J)
0 → (NMR)
(PLIST) → (X1)
1 → (JJ)
1 → I
(ST) → (X2)
PG. 3
E
51
IS X2 NULL ?
NO
52
PG. 3
D
60
yes
FRDM
ITEM FROM 1ST ELEMENT OF X2 → L(J)
CALL FRDM (X2, J)
(I-J) = 0 ?
NO
53
PG. 3
C
55
yes
LINK
LINK OF 1ST ELEMENT IN X2 → L(X2)
CALL LINK (X2, X
PG. 1
B
20
I + 1 → I
NO


PG. 2
C
55
(JJ-J): 0
<0
=0
>0
70&55
(PLIST)→(X1)
0→(JJ)
56
(JJ)+1→(JJ)
PRPTR
CCT NE, NN, NP, NTRN, NNTR, NF FROM X1
IS X1 NULL ?
NO
yes
57
CALL DUMP
58
FLAG
FLAG IST CLEMENT OF IST SUBLIST OF X1 W/I
PG. 2
D
60
(NMR)+1→(NMR)
1→K
SNATCH
SNATCH T(K, I, 0, 0) = FTEM
|FTEM| ≤ 1.E-30 ?
yes
NO
65
STASH
STASH TL(K, NMR, 0, 0) = FTEM
70
K=(NM) ?
NO
K+1→K
yes
80
I=(NM) ?
yes
PG. 4
F
NO
I+1→I
PG. 2
E


PG 3
F
ERASE
ERASE LIST ST
ERASEA
ERASE ARRAY-LIST T
1 → I
81
0 → (NPT(I))
I = 7 ?
NO
I+1 → I
yes
(PLIST) → (X1)
82
PRPTR
GET NE, NN, NP, NTRN, NNTR, NF FROM X1
(NF) ≤ 0 ?
NO
yes
85
PRPTG
FORM NEW LIST VV WITH NE, NN, NP, NTRN, NNTR
IF NE=5, NTRN, NNTR ARE OMITTED
CALL PRPTG (VV, NE, NN, NP, NTRN, NNTR, 0)
(NPT(NE)) + 1 → (NPT(NE))
90
IS X1 NULL ?
NO
yes
95
UPDWN
POPUP LIST VV PUSH DOWN INTO LIST VC (VV DESTROYED)
CALL UPDWN (VV, VC)
RETURN

Program Description

1. Identification

a. Routine Label

SUBST

2. Function

An abstract statement description is transformed into an executable statement in FORTRAN format.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

Call SUBST (R, T, N, PRO, FLS, KE, NPT, EQST, INLST, ISSW, IV34, ISIS, LSTMK)

b. Entry Conditions

R = A simple list containing, in reverse order, the characters representing the abstract statement

T = A four-dimensional matrix represented in list-structure form (matrix TC of Main 2)

N = The number of nodes

NPT = The array NPT of Main 2

FLS = The matrix XFG of Main 2

INLST = A list which holds input-variable names. This list is updated

ISSW = A switch

ISSW = 1 if processing a nonlinear statement

ISSW = 2 if processing a standard statement

IV34 = Newton-Raphson switch, which may be updated

ISIS = An indicator, which may be updated

LSTMK = A list which holds dependent stop-function identifiers. This list is updated.

c. Exit Conditions

PRO	A list holding the transformed statement in FORTRAN format.
FLS	May be updated by inserting new items.
KE	An integer indicating the type of statement in PRO
EQST	A list containing the left side of the statement, if the statement was an equation.
INLST	May be updated by inserting new input-variables
IV34	May be updated
ISIS	May be updated
LSTMK	May be updated by inserting new stop-function names.

5. Definition of Identifiers

IOP(8) contains these punctuation characters:

IOP(1)	=
IPP(2)	)
IOP(3)	(
IOP(4)	*
IOP(5)	/
IOP(6)	+
IOP(7)	-
IOP(8)	,

IDLR contains the character $.

PMAST	A list containing a containing a copy of R, in reverse order
J	Holds the index to the current punctuation chracter
KSTB	Holds the index to the previous punctuation character
ITEM	Used to hold the character being popped out of R
IFLG	Holds the flag of the element being popped out of R
ITEM3	Holds first character of symbol being examined
ITEM5	Holds first character of symbol being examined
ITEM4	Holds punctuation character corresponding to index J
ITEM6	Holds second character of symbol
ITEM7	A list of characters representing a dependent stop variable

ITEM8	A list-structure of all ITEM7's in the statement
INLST2	A list of characters representing an input-variable name
INLST1	A list-structure of all the INLST2's in the statement
SYMB	A list used to hold the symbol being examined
KTMDF	= 1 if c(SYMB) is 'FT' or '$UTF' = 0 otherwise
KSV	= 2 if c(SYMB) is 'SS....' or 'SV....' = 1 otherwise
IV34S	Used to hold the revised value of IV34
KS	Type code of c(SYMB)
NE	Element type code of c(SYMB)
NO	First node (number of c(SYMB)
NT	Second node (number of c(SYMB)
TEM, TEM1, SAV, PL2, PL3	Lists used for temporary construction of character strings
DLRVR	Holds (NE, NO, NT) triplet for each $S.... symbol occurring on right side.

6. Method

List R is popped up and saved in SYMB until a punctuation character is recognized. The symbol (in SYMB) is examined by SYMTP and indicators KS, NE, NO, NT, KTMDF, KSV are set. The symbol is examined further: KE is set; if it is an input variable, it is saved in INLST2/INLST1; if it is a dependent stop-function variable, it is saved in ITEM7/ITEM8. If the symbol is a system variable of the type SV.... or SS.... then COMBN is called to transform the abstract symbol into a linear combination of computable variables. As the symbols are processed, they are pushed into PRO, along with the existing punctuation and any new punctuation (e.g., parentheses) necessary to preserve the computational integrity of the statement. If an = is

encountered as a punctuation character, the statement is recognized as an equation and the left side is processed. If no = is found, R is examined for other statement types:

a. Equation type

The symbol to the "left" of the = sign (the remaining contents of R) is saved in EQST. INLST1 is OR-ed into INLST; ITEM8 is OR-ed into LSTMK. The symbol on the left side is examined. If it is an SV .... then COMBN is used to replace the symbol with its equivalent computational variable. The result is pushed into PRO. KE is set, depending on the contents of the left and right sides of the euqation. IV34 is updated if KE $\neq$ 8. If the left side is a network variable but not SV or SS, if KE>3 and this is a FORM-1 statement, then (NE, NO, NT) of the left side are entered in FLS through MRKLST.

b. Statement type

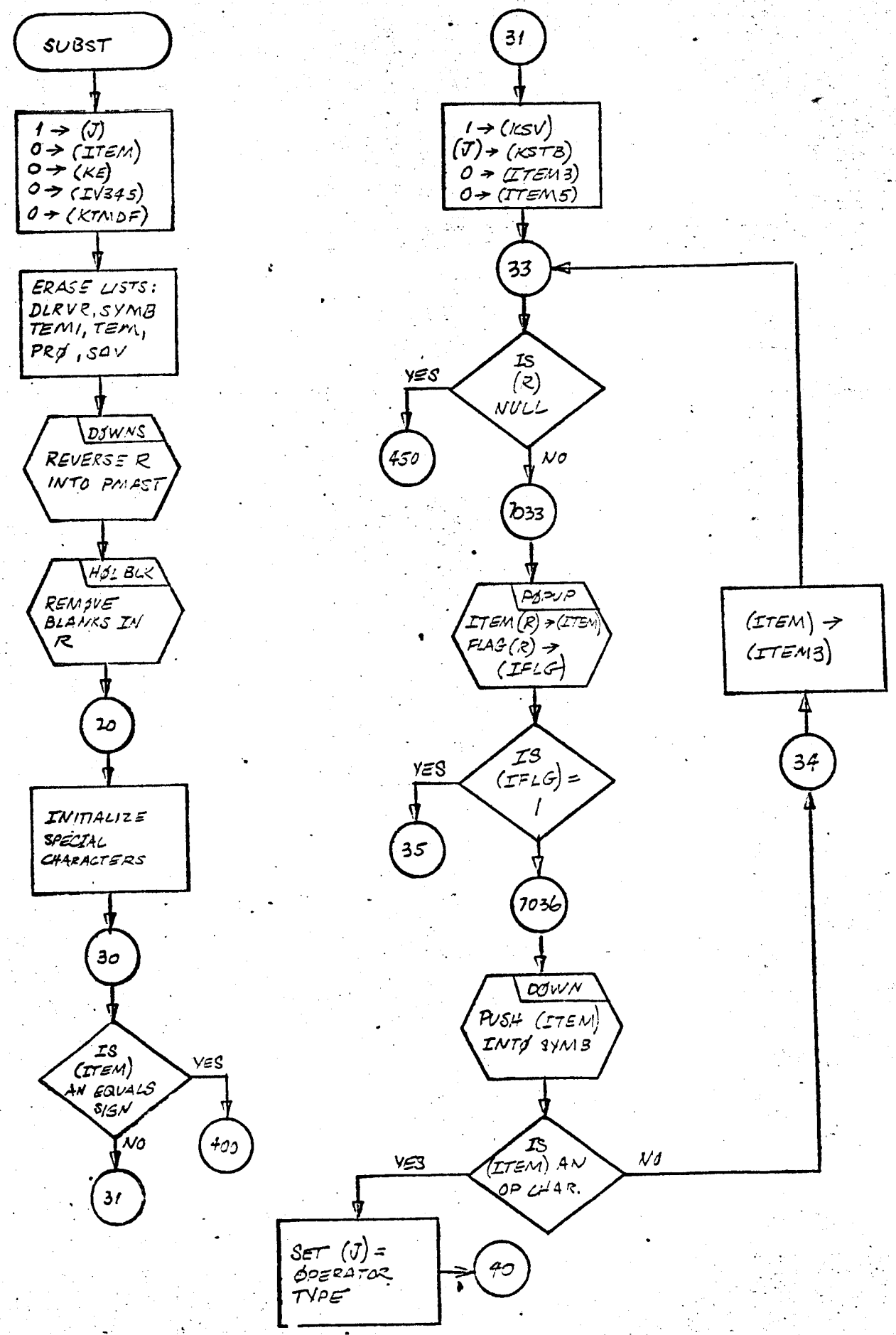
If R contains:

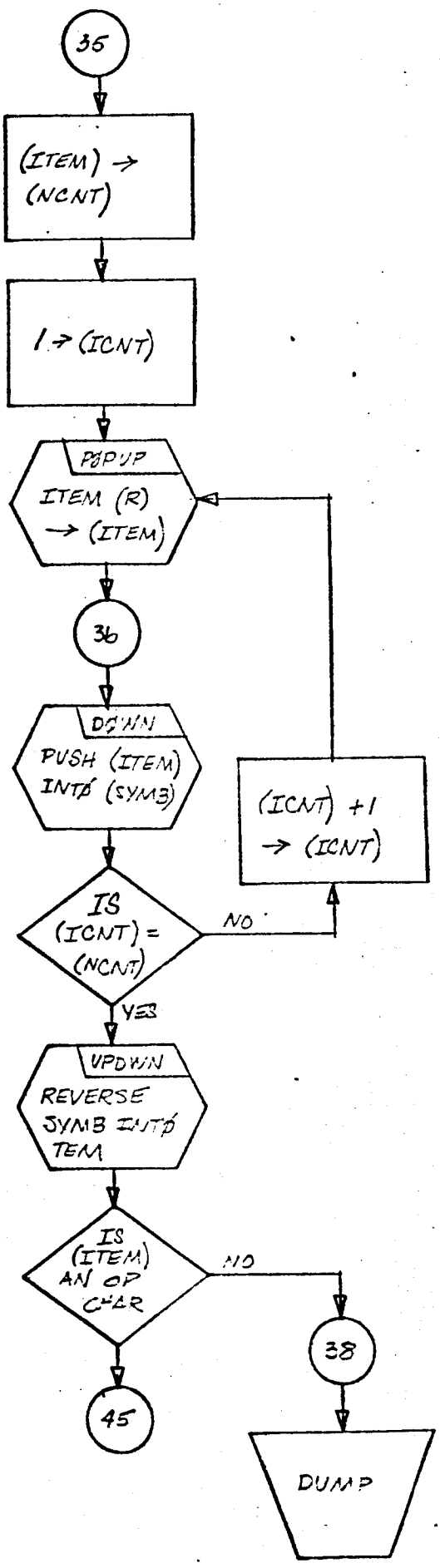
(1) DIMENSION of DEFINE, INLST1 is erased, and the array names are extracted and placed in INLST1. Then INLST1 is OR-ed into INLST. PRO is set to contain the original statement. KE is set = 1.

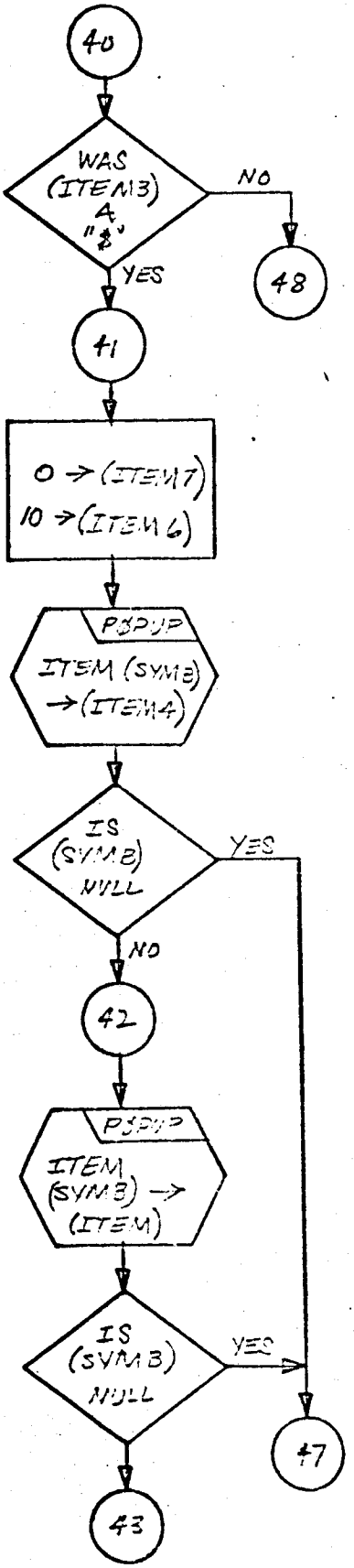
(2) IF: If any characters follow IF and precede the left parenthesis, then INLST1 is OR-ed into INLST, ITEM8 is OR-ed into LSTMK, and PRO remains in its transformed form. If no characters follow IF, then PRO is set to contain the original statement. KE is set = 1.

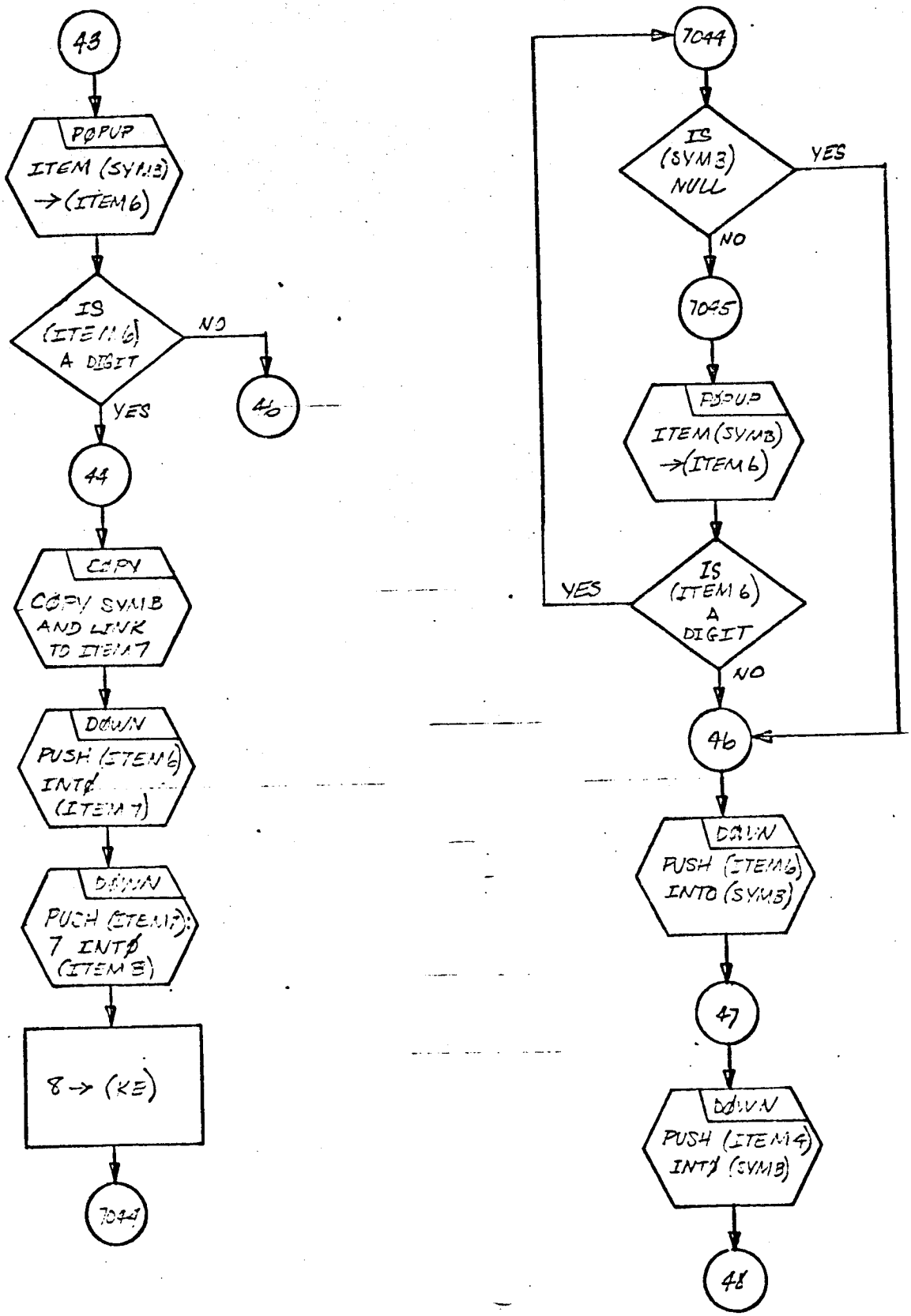
(3) GO TO: If it is a computed GO-TO, INLST1 is OR-ed into INLST; else INLST1 is ignored. PRO is set to contain the original statement. KE is set = 1.

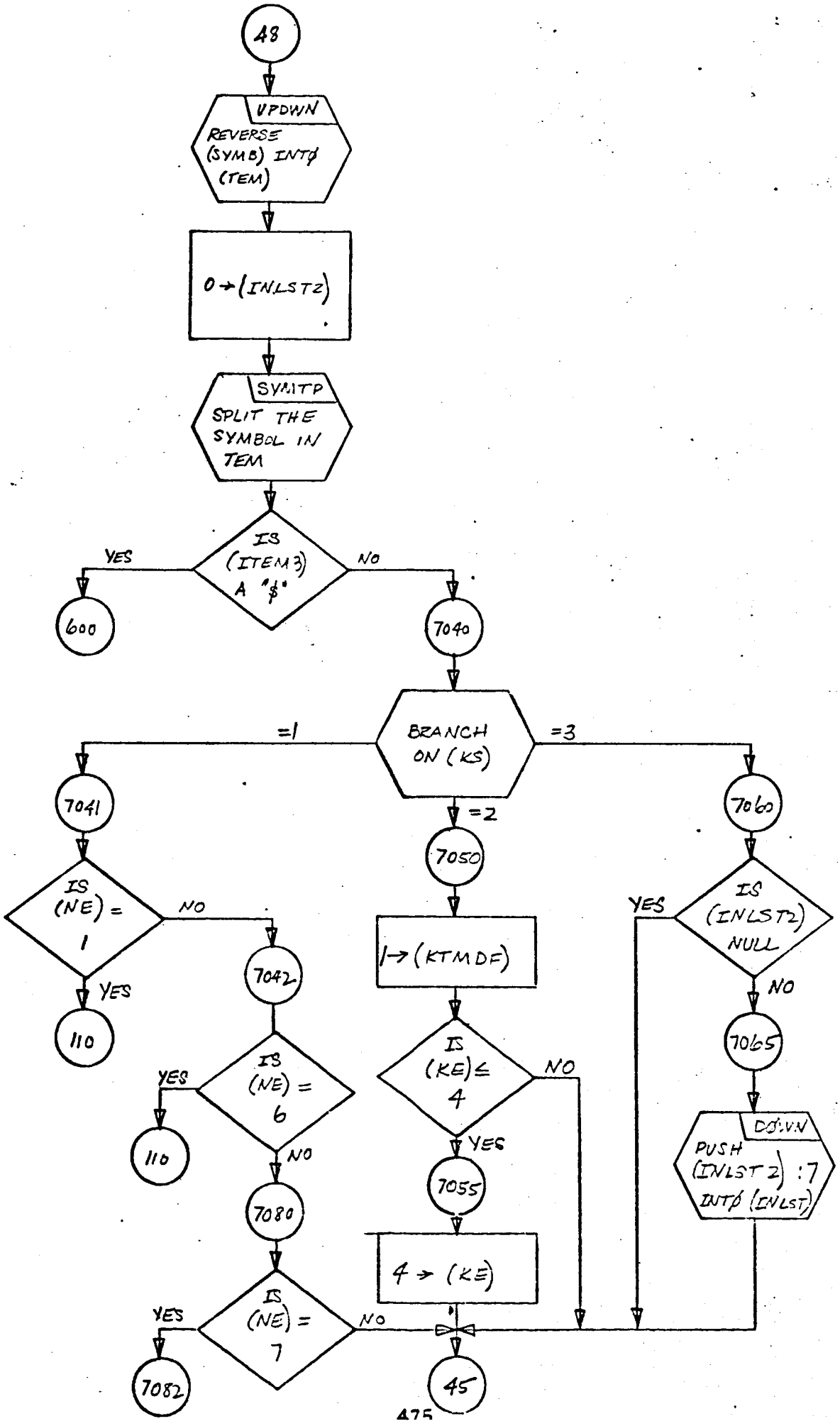
(4) CALL: If DLRVR is null, and then if ITEM8 is null, KE is set = 1. If ITEM8 is not null, KE is left undisturbed. If DLRVR is not null and KE $\leq$ 3, KE is set = 3. If DLRVR is not null and KE>3, then each triplet in DLRVR is examined. If the network variable is not FT or SS.... or SV.... and this is a FORM-1 statement, then enter (NE, NO, NT) in FLS, using MRKLST and parameter KE. If any of these network variables was neither FT nor SS.... and KE $\neq$ 8, then set IV34 = max(IV34, IV34S).

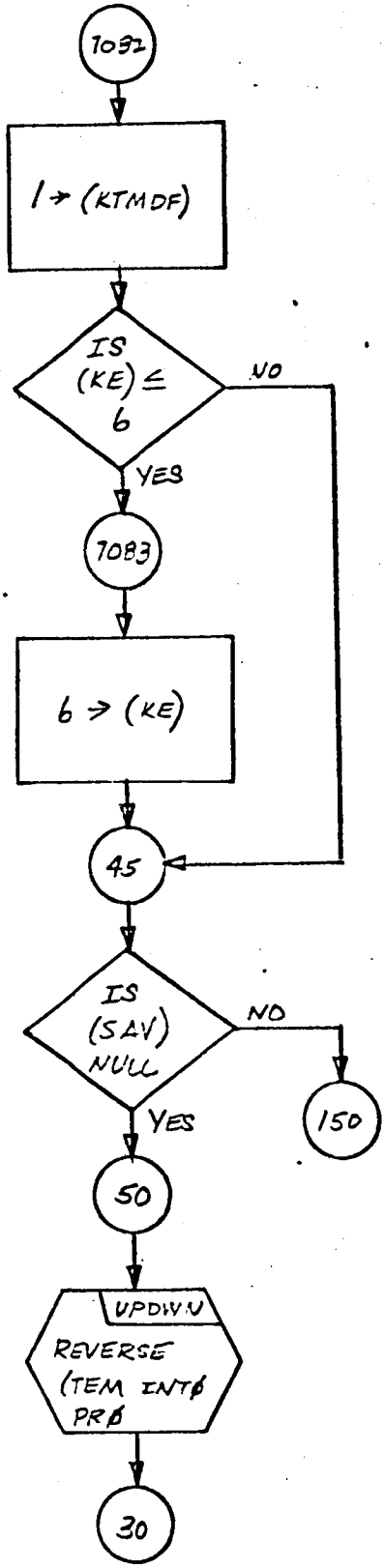

SUBST
1 → (J)
0 → (ITEM)
0 → (KE)
0 → (IV345)
0 → (KTMDF)
ERASE LISTS: DLRVR, SYMB TEMI, TEM, PRØ, SAV
DØWNS
REVERSE R INTO PMAST
HØL BLK
REMØVE BLANKS IN R
20
INITIALIZE SPECIAL CHARACTERS
30
IS (ITEM) AN EQUALS SIGN
YES
NO
400
31
31
1 → (ICSV)
(J) → (KSTB)
0 → (ITEM3)
0 → (ITEM5)
33
IS (R) NULL
YES
NO
450
7033
PØPUP
ITEM (R) → (ITEM)
FLAG (R) → (IFLG)
IS (IFLG) = 1
YES
35
7036
DØWN
PUSH (ITEM) INTØ SYMB
IS (ITEM) AN OP CHAR.
YES
NO
SET (J) = ØPERATOR TYPE
40
(ITEM) → (ITEM3)
34

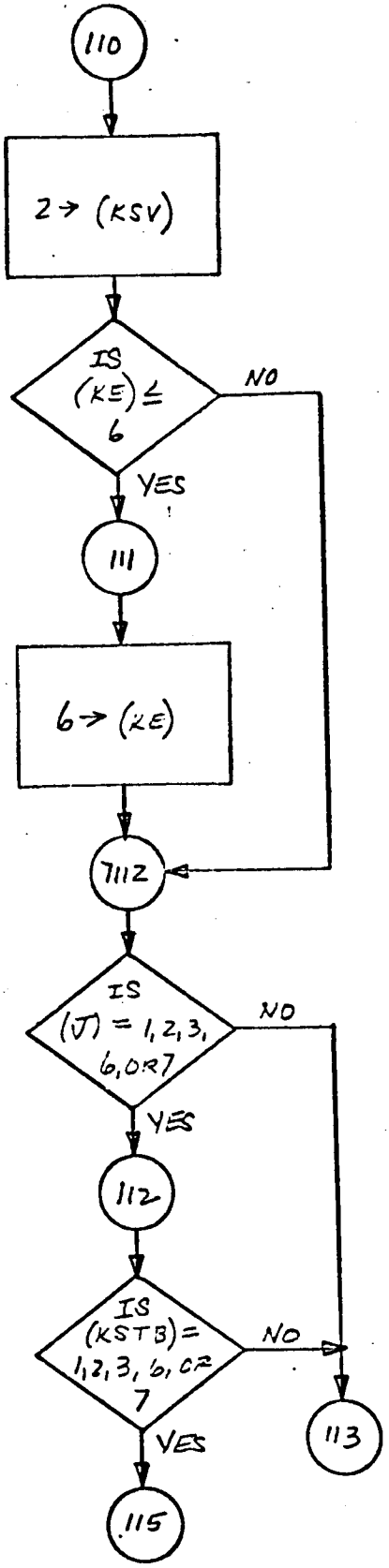

35
(ITEM) → (NCNT)
1 → (ICNT)
POPUP
ITEM (R) → (ITEM)
36
DOWN
PUSH (ITEM) INTO (SYMB)
(ICNT) +1 → (ICNT)
IS (ICNT) = (NCNT)
NO
YES
UPDWN
REVERSE SYMB INTO TEM
IS (ITEM) AN OP CHAR
NO
38
45
DUMP

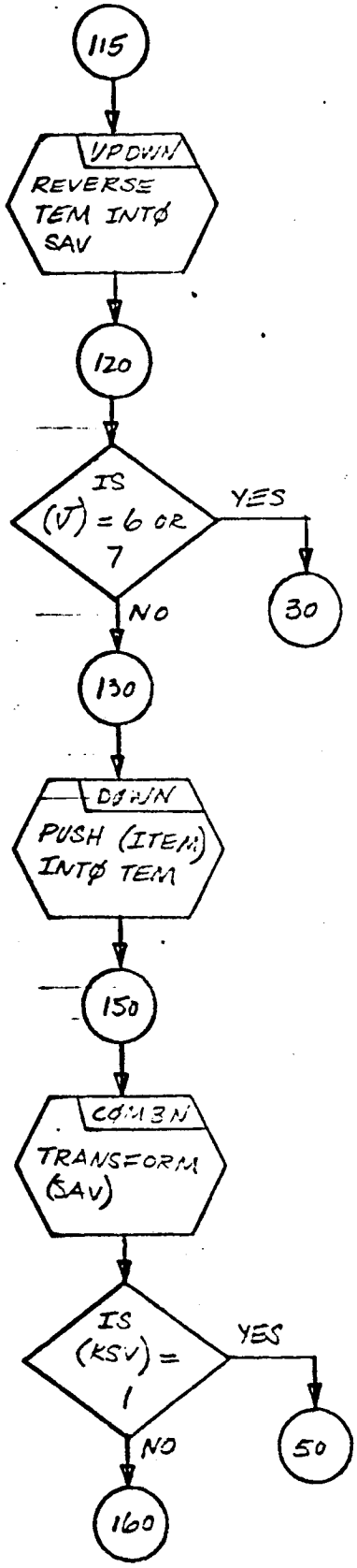

40
WAS (ITEMB) 4, "$"
NO
YES
48
41
0 → (ITEM7)
10 → (ITEM6)
POPUP
ITEM (SYMB) → (ITEM4)
IS (SYMB) NULL
YES
NO
42
POPUP
ITEM (SYMB) → (ITEM)
IS (SYMB) NULL
YES
47
43

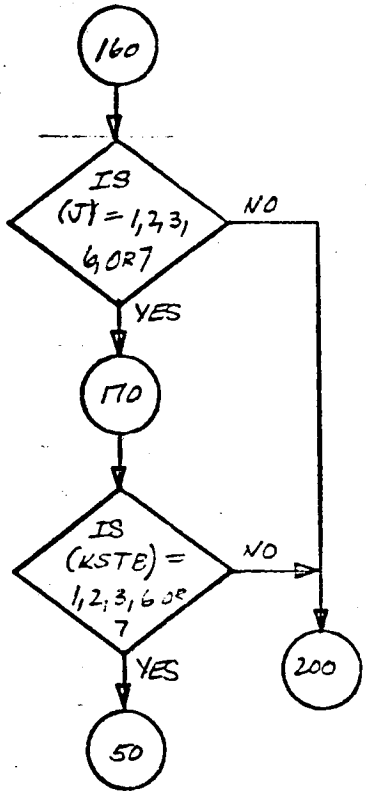

43
POPUP
ITEM (SYM3)
→(ITEM6)
IS
(ITEM6)
A DIGIT
NO
YES
46
44
COPY
COPY SYMB
AND LINK
TO ITEM7
DOWN
PUSH (ITEM6)
INTO
(ITEM7)
DOWN
PUSH (ITEM7);
7 INTO
(ITEM3)
8 → (KE)
7044
7044
IS
(SYM3)
NULL
YES
NO
7045
POPUP
ITEM (SYM3)
→(ITEM6)
IS
(ITEM6)
A
DIGIT
YES
NO
46
DOWN
PUSH (ITEM6)
INTO (SYM3)
47
DOWN
PUSH (ITEM4)
INTO (SYM3)
48

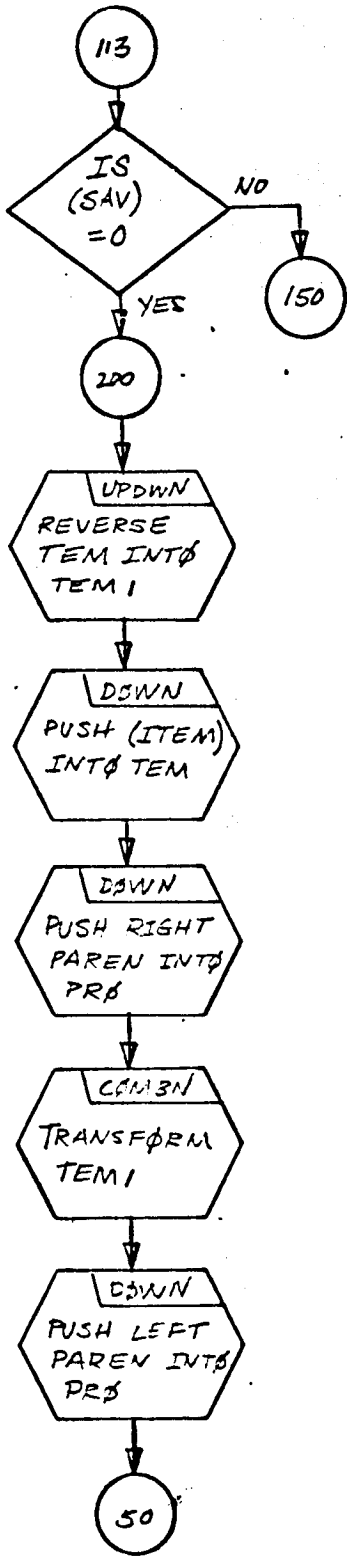

48
UPDWN
REVERSE (SYMB) INTO (TEM)
0 → (INLST2)
SYMTP
SPLIT THE SYMBOL IN TEM
IS (ITEM3) A "$"
YES
NO
600
7040
BRANCH ON (KS)
=1
=2
=3
7041
7050
7060
IS (NE) = 1
NO
YES
110
7042
1 → (KTMDF)
IS (INLST2) NULL
YES
NO
7065
IS (NE) = 6
YES
NO
110
IS (KE) ≤ 4
NO
YES
7055
DOWN
PUSH (INLST2) :7 INTO (INLST)
7080
4 → (KE)
IS (NE) = 7
YES
NO
7082
45

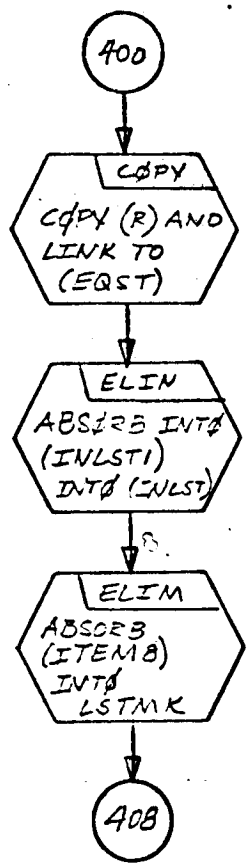

7092
1 → (KTMDF)
IS (KE) ≤ 6
NO
YES
7083
6 → (KE)
45
IS (SAV) NULL
NO
150
YES
50
UPDWN
REVERSE (TEM INTØ PRØ
30

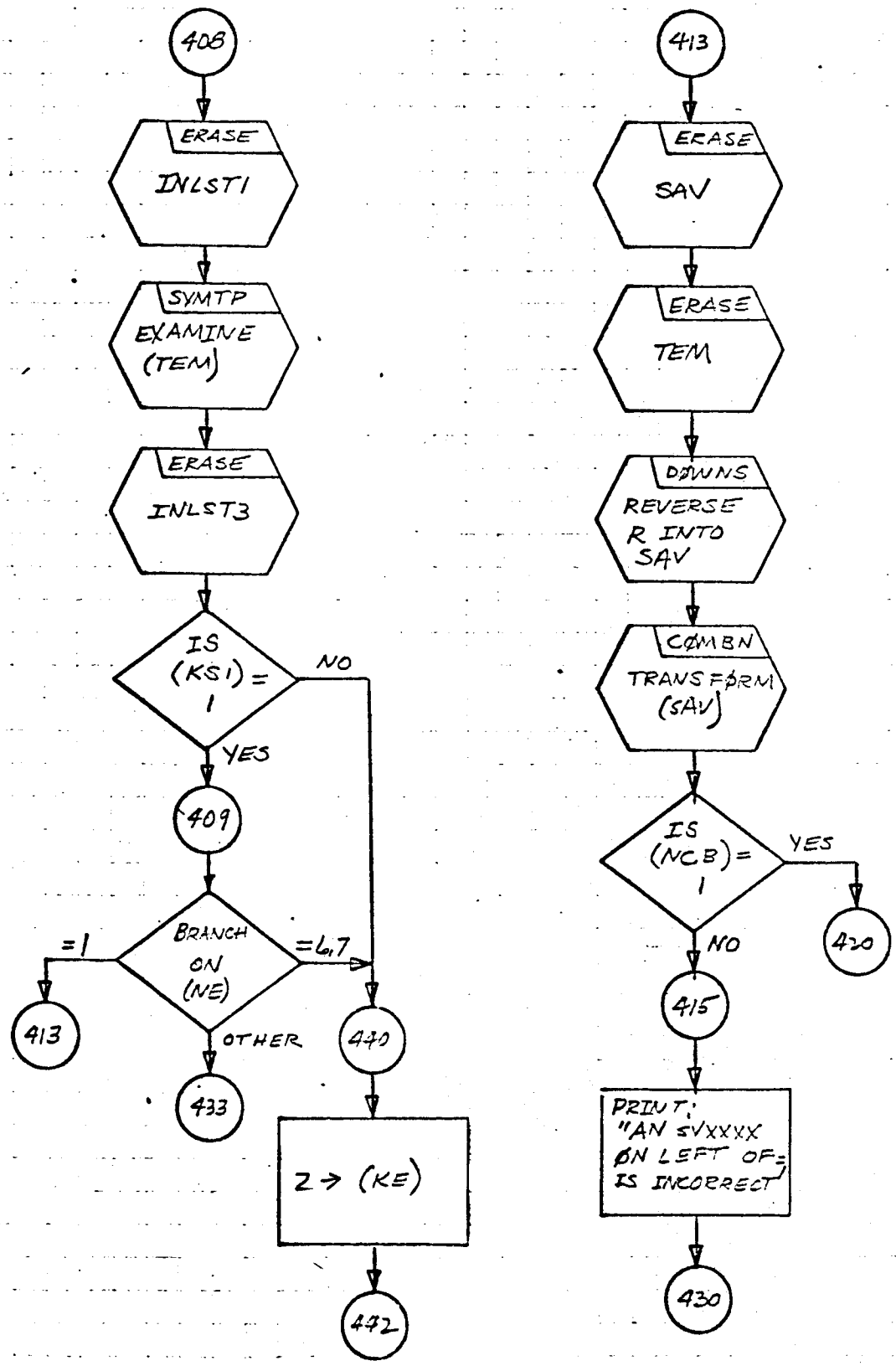

110
2 → (KSV)
IS (KE) ≤ 6
NO
YES
111
6 → (KE)
7112
IS (J) = 1, 2, 3, 6, OR 7
NO
YES
112
IS (KSTB) = 1, 2, 3, 6, OR 7
NO
113
YES
115

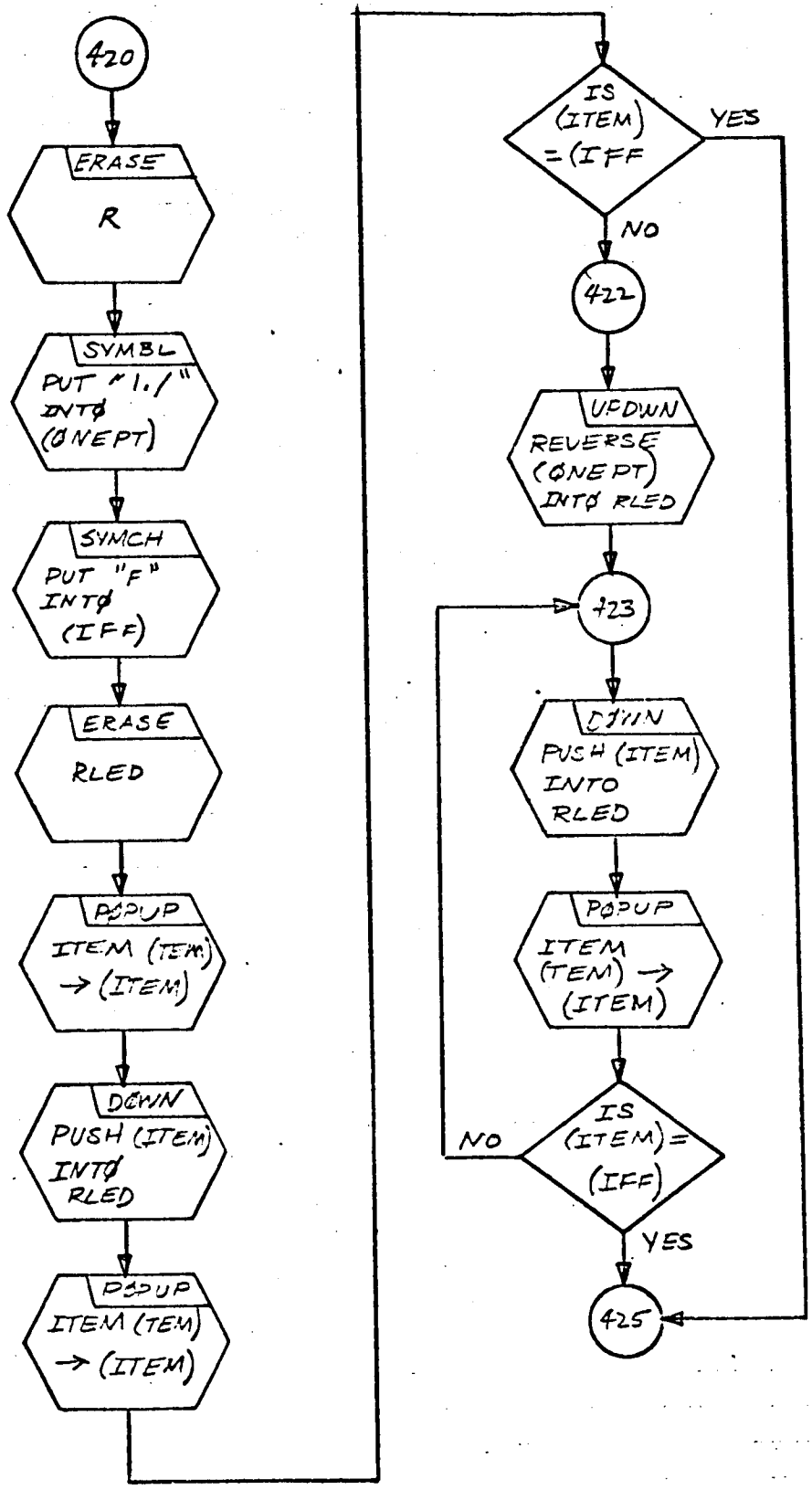

115
VPDWN
REVERSE TEM INTØ SAV
120
IS (V) = 6 OR 7
YES
30
NO
130
DØWN
PUSH (ITEM) INTØ TEM
150
COM3N
TRANSFORM (SAV)
IS (KSV) = 1
YES
50
NO
160

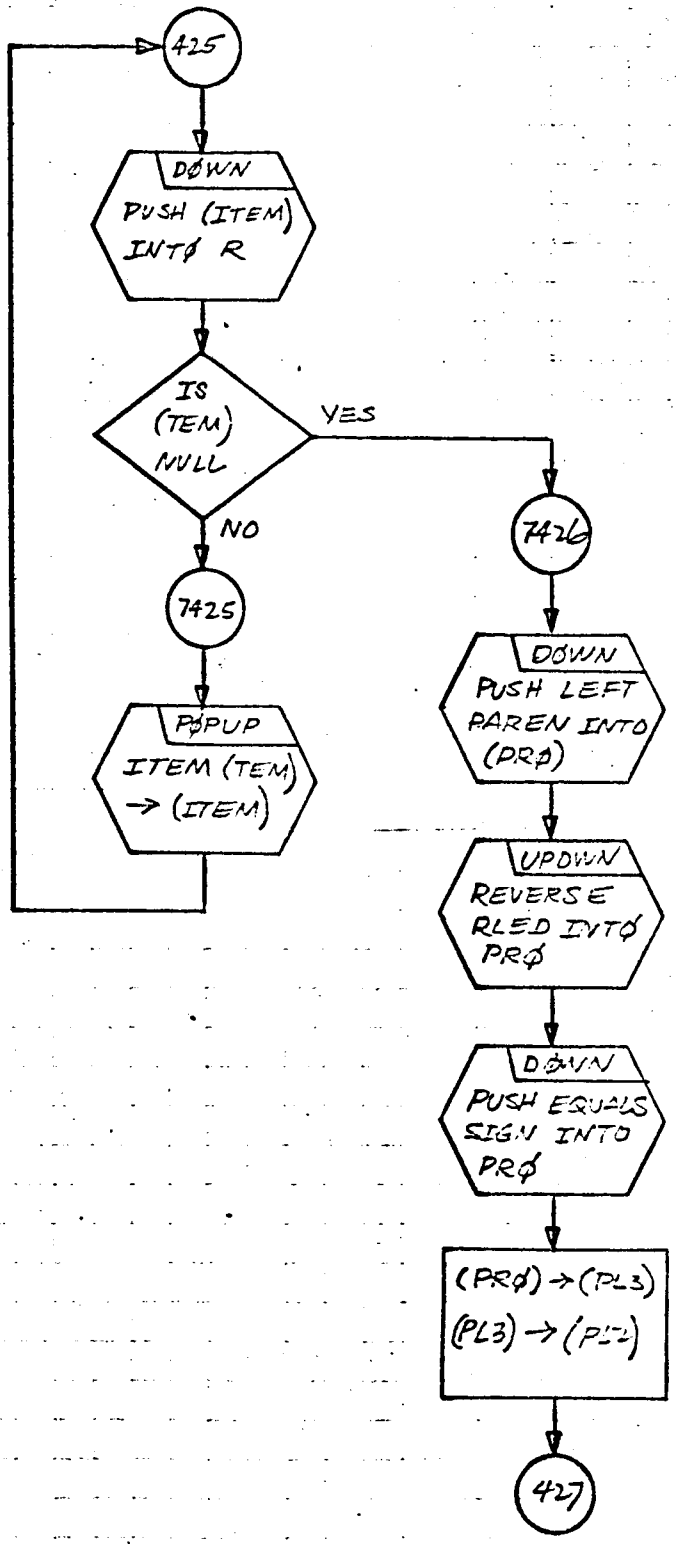

160
IS (J) = 1,2,3, 4, OR 7
NO
YES
170
IS (KSTE) = 1, 2, 3, 6 OR 7
NO
200
YES
50

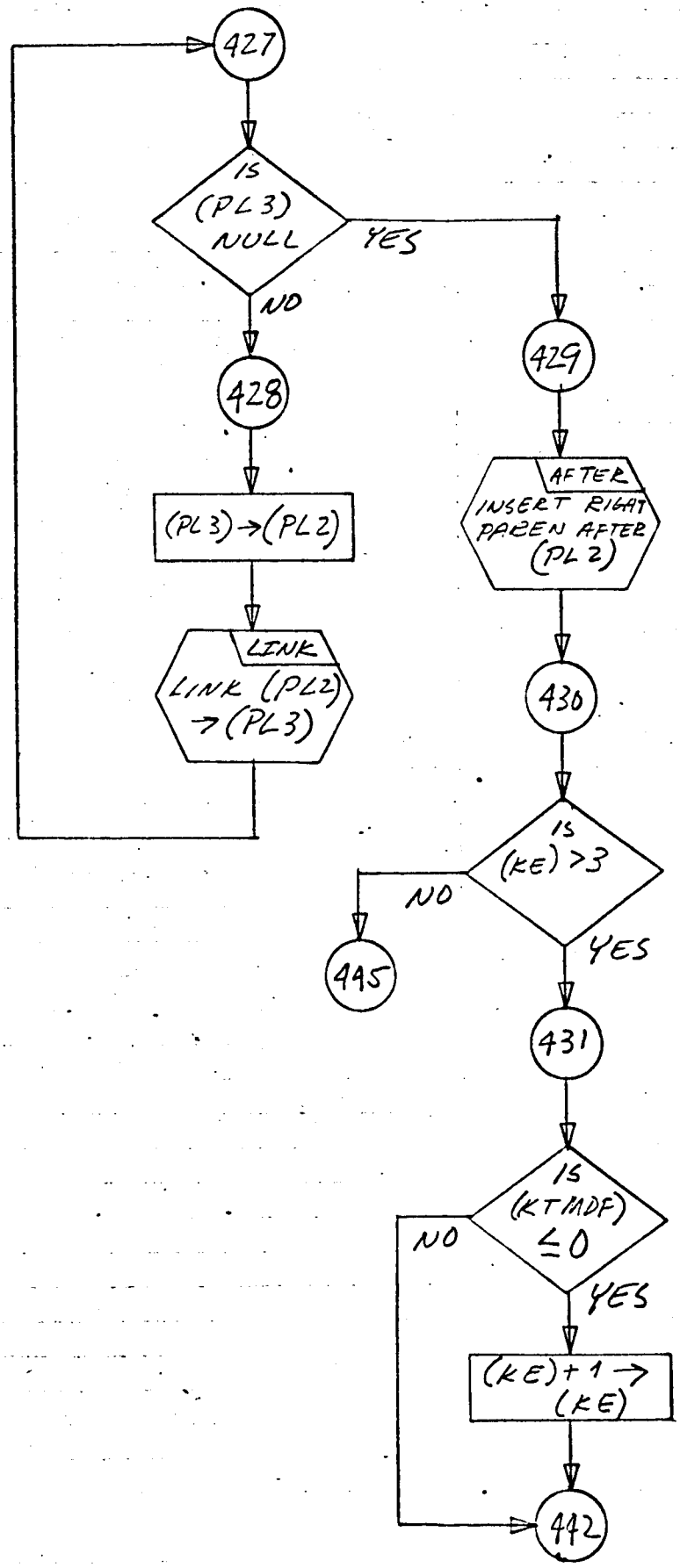

113
IS (SAV) =0
NO
YES
150
200
UPDWN
REVERSE TEM INTØ TEM1
DOWN
PUSH (ITEM) INTØ TEM
DOWN
PUSH RIGHT PAREN INTØ PRØ
COMBN
TRANSFØRM TEM1
DOWN
PUSH LEFT PAREN INTØ PRØ
50

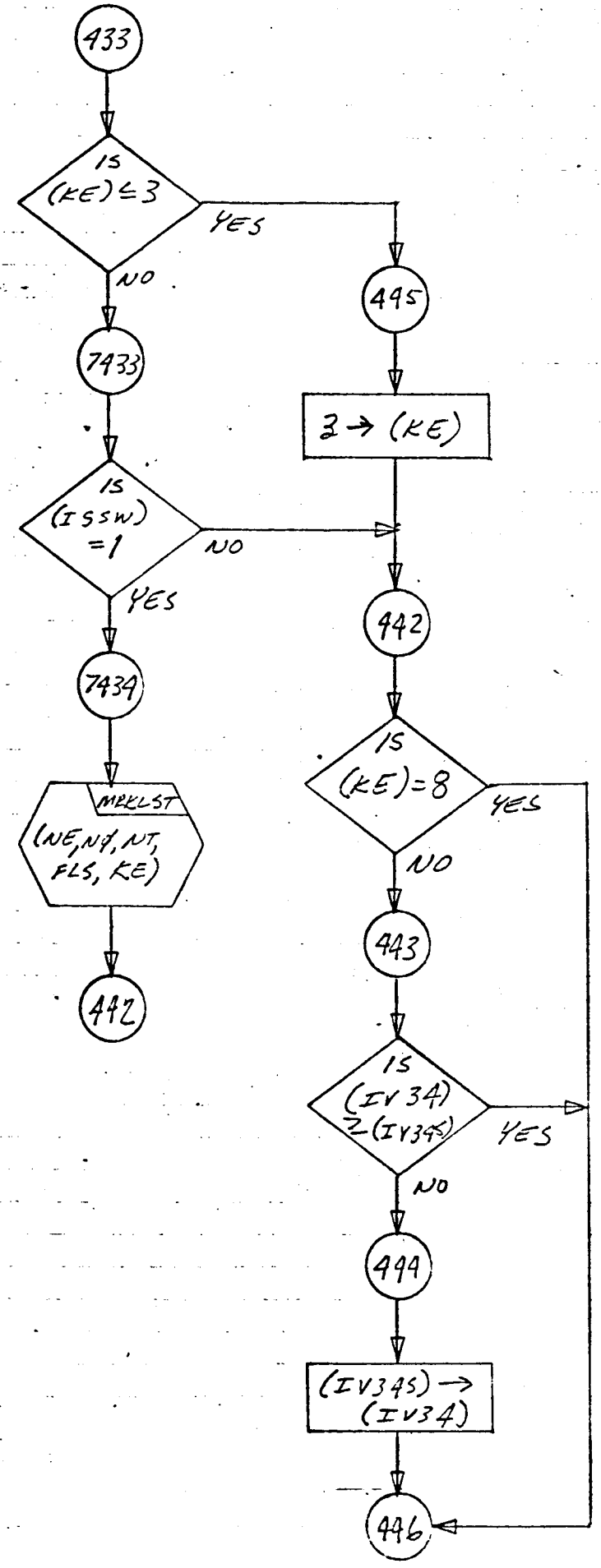

400
COPY
COPY (R) AND LINK TO (EQST)
ELIN
ABSORB INTØ (INLST1) INTØ (INLST)
ELIM
ABSORB (ITEM8) INTØ LSTMK
408


408
ERASE
INLST1
SYMTP
EXAMINE
(TEM)
ERASE
INLST3
IS
(KS1) =
1
NO
YES
409
BRANCH
ON
(NE)
=1
=6,7
OTHER
413
433
440
2 → (KE)
442
413
ERASE
SAV
ERASE
TEM
DOWNS
REVERSE
R INTO
SAV
COMBN
TRANSFORM
(SAV)
IS
(NCB) =
1
YES
NO
420
415
PRINT:
"AN SVXXXX
ON LEFT OF=
IS INCORRECT"
430

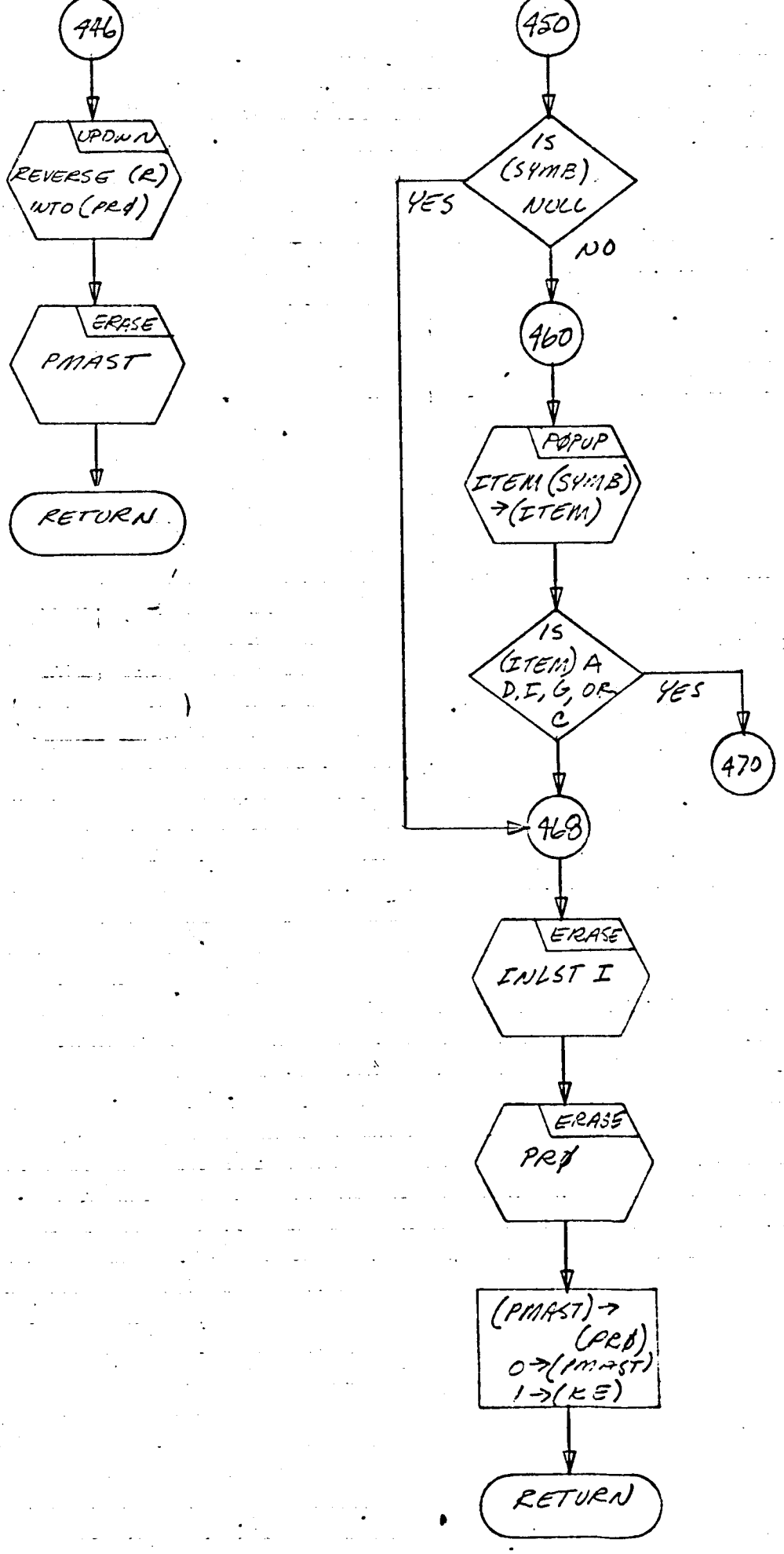

420
ERASE
R
SYMBL
PUT "1./" INTO (ONEPT)
SYMCH
PUT "F" INTO (IFF)
ERASE
RLED
POPUP
ITEM (ITEM) → (ITEM)
DOWN
PUSH (ITEM) INTO RLED
POPUP
ITEM (ITEM) → (ITEM)
IS (ITEM) = (IFF
YES
NO
422
UPDWN
REVERSE (ONEPT) INTO RLED
423
DOWN
PUSH (ITEM) INTO RLED
POPUP
ITEM (ITEM) → (ITEM)
IS (ITEM) = (IFF)
NO
YES
425

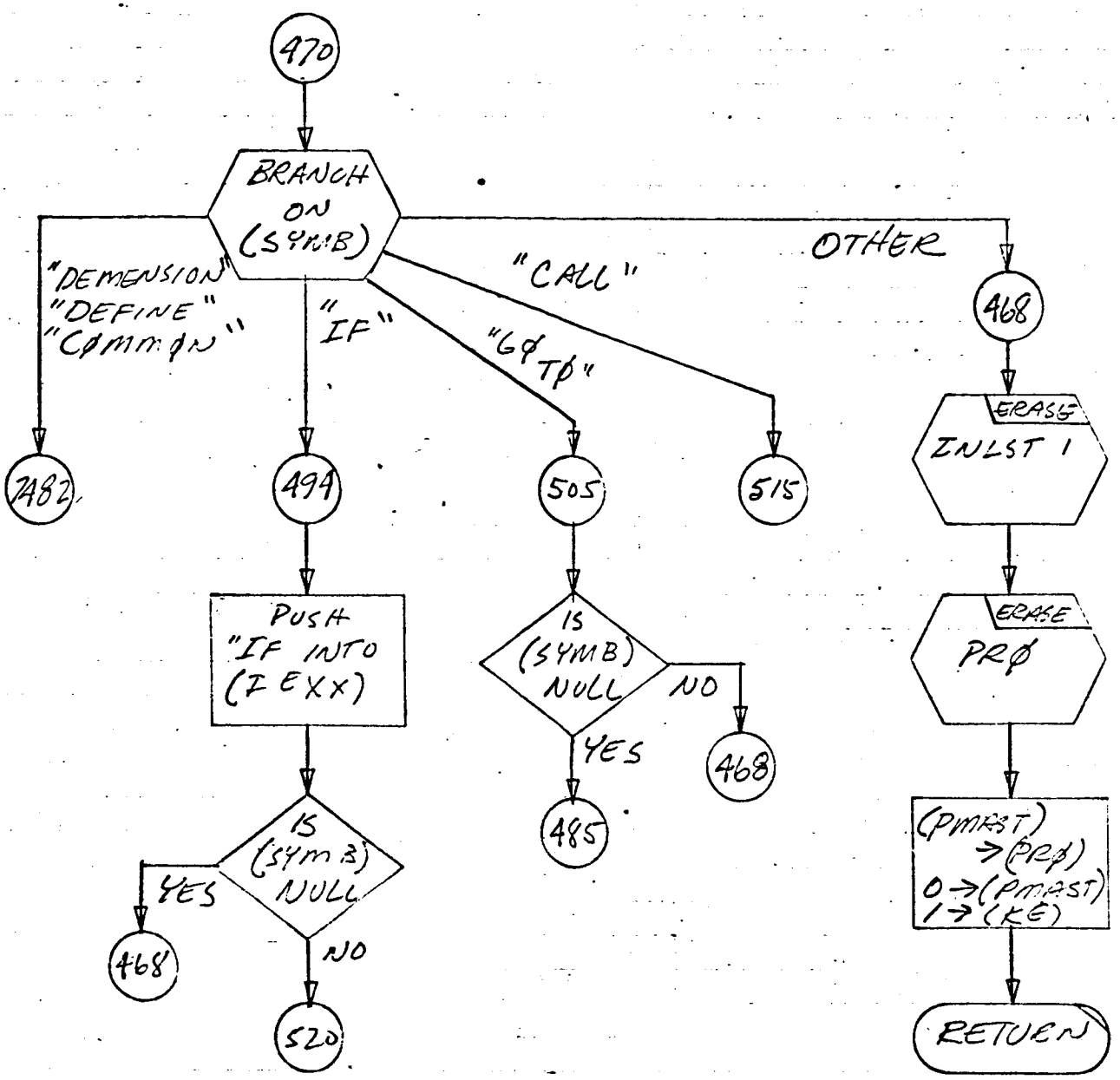

425
DOWN
PUSH (ITEM) INTO R
IS (TEM) NULL
YES
NO
7426
7425
POPUP
ITEM (TEM) → (ITEM)
DOWN
PUSH LEFT PAREN INTO (PRO)
UPDOWN
REVERSE RLED INTO PRO
DOWN
PUSH EQUALS SIGN INTO PRO
(PRO) → (PL3)
(PL3) → (PL2)
427


427
IS (PL3) NULL
YES
NO
428
429
(PL3)→(PL2)
AFTER
INSERT RIGHT PAREN AFTER (PL2)
LINK
LINK (PL2) → (PL3)
430
IS (KE) >3
NO
YES
495
431
IS (KTMDF) ≤0
NO
YES
(KE)+1 → (KE)
442

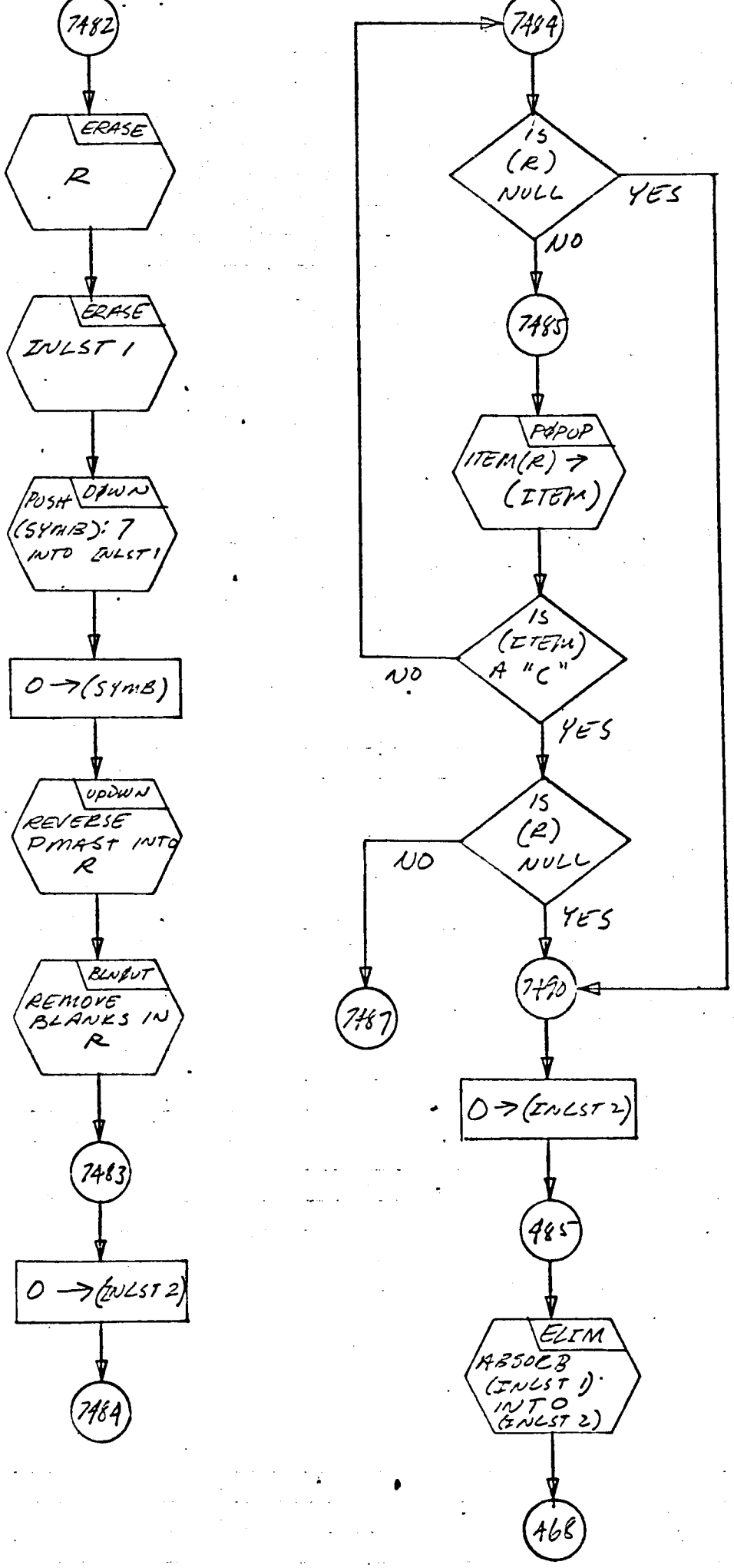

433
IS (KE) ≤ 3
YES
NO
495
7433
3 → (KE)
IS (ISSW) = 1
NO
YES
442
7434
IS (KE) = 8
YES
MRKLST
(NE, NV, NT, FLS, KE)
NO
443
447
IS (IV34) ≠ (IV34S)
YES
NO
444
(IV34S) → (IV34)
446

SUBST 12

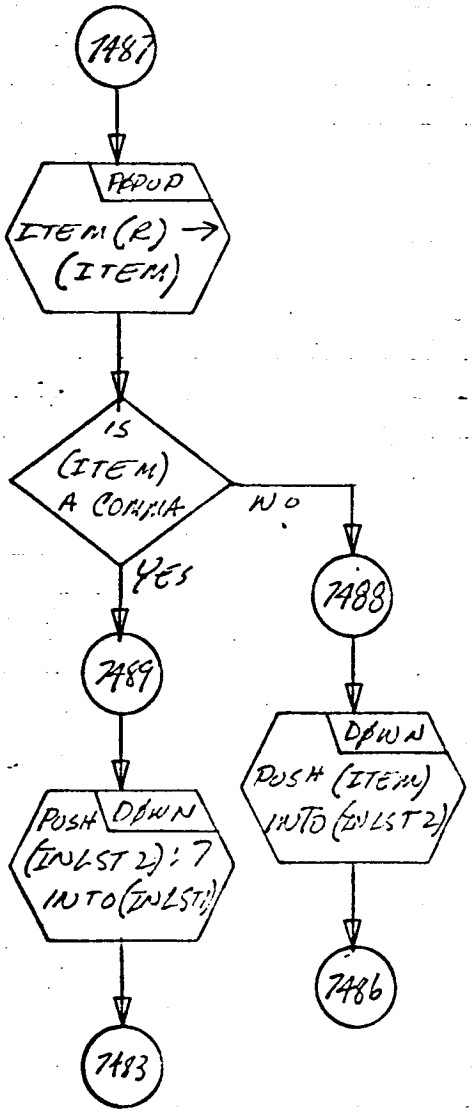

446
UPDWN
REVERSE (R)
INTO (PRØ)
ERASE
PMAST
RETURN
450
IS
(SYMB)
NULL
YES
NO
460
PØPUP
ITEM (SYMB)
→(ITEM)
IS
(ITEM) A
D, E, G, OR
C
YES
470
468
ERASE
INLST I
ERASE
PRØ
(PMAST)→
(PRØ)
0→(PMAST)
1→(KE)
RETURN

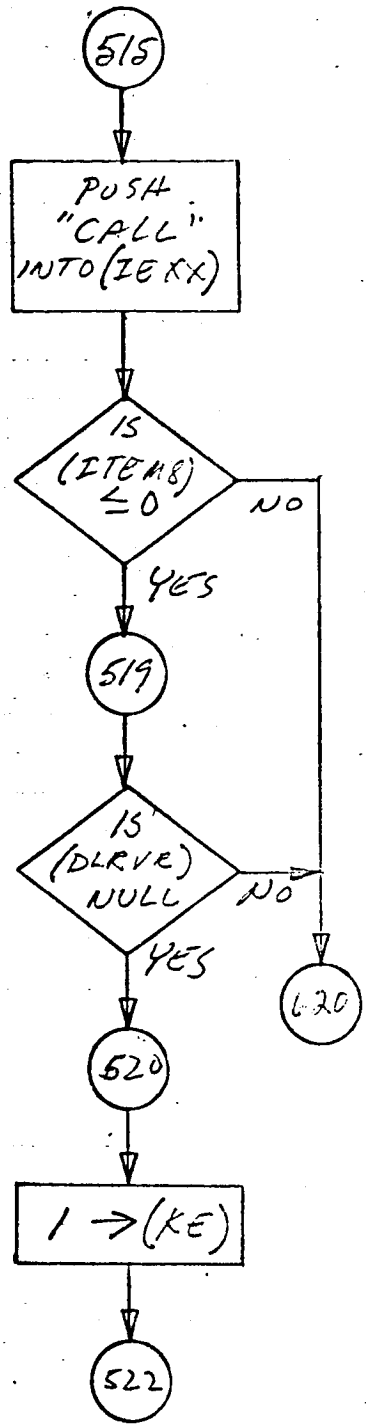

470
BRANCH ON (SYMB)
"DEMENSION"
"DEFINE"
"COMMON"
"IF"
"GOTO"
"CALL"
OTHER
482
494
505
515
468
ERASE
INLST 1
ERASE
PRØ
(PMAST) → (PRØ)
0 → (PMAST)
1 → (KE)
RETUEN
PUSH "IF INTO (IEXX)
IS (SYMB) NULL
YES
NO
468
520
IS (SYMB) NULL
YES
NO
485
468

SUBST 14

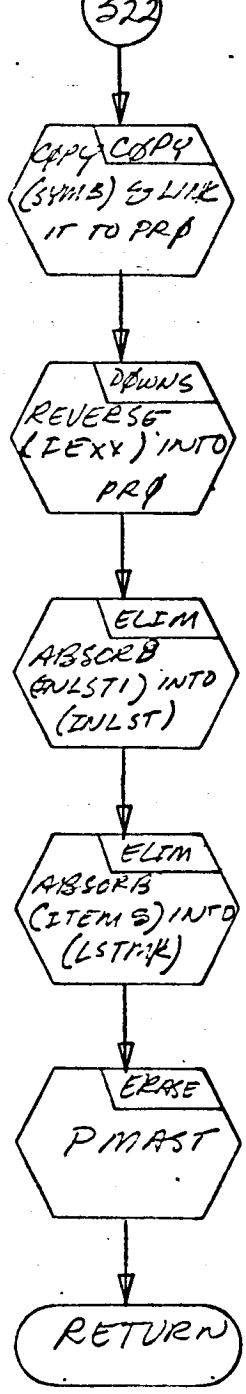

7482
ERASE
R
ERASE
INLST 1
PUSH DOWN
(SYMB): 7
INTO INLST 1
0 → (SYMB)
UPDWN
REVERSE
DMAST INTO
R
BLNBUT
REMOVE
BLANKS IN
R
7483
0 → (INLST 2)
7484
7484
IS
(R)
NULL
YES
NO
7485
POPOP
ITEM(R) →
(ITEM)
IS
(ITEM)
A "C"
NO
YES
IS
(R)
NULL
NO
YES
7487
7490
0 → (INLST 2)
485
ELIM
ABSORB
(INLST 1)
INTO
(INLST 2)
468

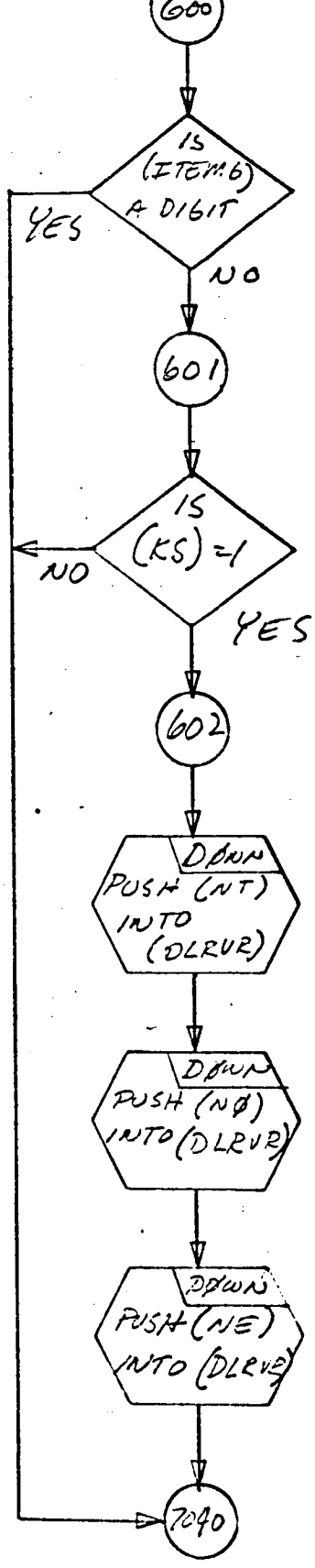
SUBST 15


7487
POPUP
ITEM(R) → (ITEM)
IS (ITEM) A COMMA
NO
YES
7488
7489
DOWN
PUSH (ITEM) INTO (INLST2)
PUSH DOWN
(INLST2) : 7 INTO (INLST1)
7486
7483

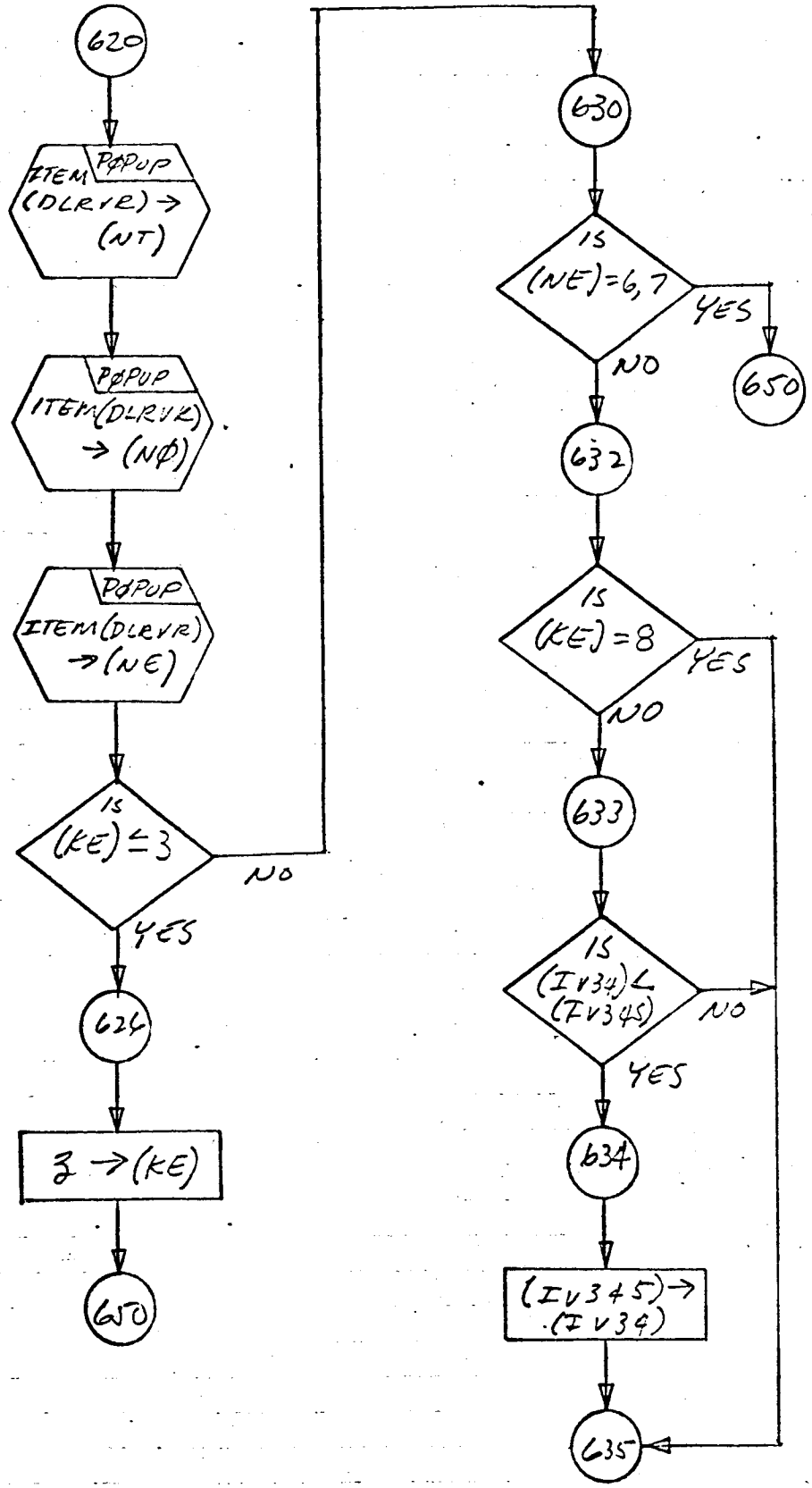

515
PUSH "CALL" INTO (IEXX)
IS (ITEM8) ≤ 0
NO
YES
519
IS (DLRVE) NULL
NO
YES
620
520
1 → (KE)
522

SUBST 16

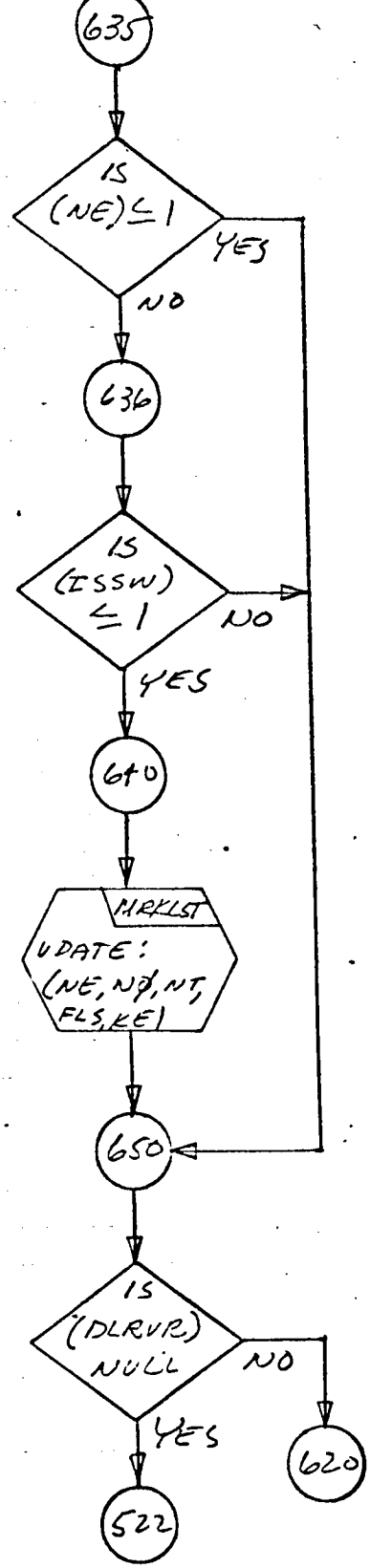

322
COPY
COPY (SYMB) & LINK IT TO PRØ
DØWNS
REVERSE (IEXX) INTO PRØ
ELIM
ABSORB (NULST1) INTO (INLST)
ELIM
ABSORB (ITEM 3) INTO (LSTINK)
ERASE
PMAST
RETURN


600
IS (ITEM 6) A DIGIT
YES
NO
601
IS (KS) = 1
NO
YES
602
DØWN
PUSH (NT) INTO (DLRVR)
DØWN
PUSH (NØ) INTO (DLRVR)
DØWN
PUSH (NE) INTO (DLRVR)
7040

SUBST 17


620
POPUP
ITEM (DLRVR) → (NT)
POPUP
ITEM(DLRVR) → (NØ)
POPUP
ITEM(DLRVR) → (NE)
IS (KE) ≤ 3
NO
YES
624
3 → (KE)
650
630
IS (NE) = 6, 7
YES
NO
650
632
IS (KE) = 8
YES
NO
633
IS (IV34) < (IV345)
NO
YES
634
(IV345) → (IV34)
635

SUBST 18


635
IS (NE) ≤ 1
YES
NO
636
IS (ISSW) ≤ 1
NO
YES
640
MRKLST
UDATE: (NE, NP, NT, FLS, KE)
650
IS (DLRVR) NULL
NO
YES
620
522

SUBST 19

Program Description

1. Identification

   a. Routine Label

      SYMBL

   b. Name

      Insert an array of Hollerith characters into a simple list.

2. Function

A Hollerith array is scanned and inserted into a list, character by character, such that the first character of the array is the first element and the last character is the last element.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SYMBL (L, N, H)

   b. Entry Conditions

      L = Head of a list
      N = Number of characters in array H to be inserted in list L
      H = An array of Hollerith characters

   c. Exit Conditions

      N characters of the Hollerith array H are placed in list L, right side up.

   d. Error Exits

      None.

5. Definition of Identifiers

   J = Temporary head cell of the simple list being created from N characters of the array H.

6. Method

a. List L is erased (restored to AVS by ERASE) and the head cell L cleared to zero.

b. A temporary head cell J is also cleared to zero.

c. I is initialized to one.

d. The following steps are executed for each character of H, until N characters have been inserted into the list headed by L:

(1) The Ith character of H is extracted from the array and shifted into the decrement of a temporary cell S as an integer.

(2) L is tested:

(a) If L = 0, (S) is inserted as the item of the first element of J, with a link of zero. J is set to point to this first element. The pointer in J is also saved in L (after N characters have been inserted, L will point to the top of the list, or the first character of the array H).

(b) If L ≠ 0, the converted character in S becomes the item of a new element inserted immediately after the first element of list J, and J is set to point to this new element. The link of the element preceding the newly inserted element is also set to point to the new element, thus maintaining the subsequent linkage down the list. Note that each time a new element is placed into the list, J is pointing to the last element inserted or what can be considered as the current bottom of the list.

(3) I is tested:

(a) If I = N, all characters have been inserted and exit is made from the routine.

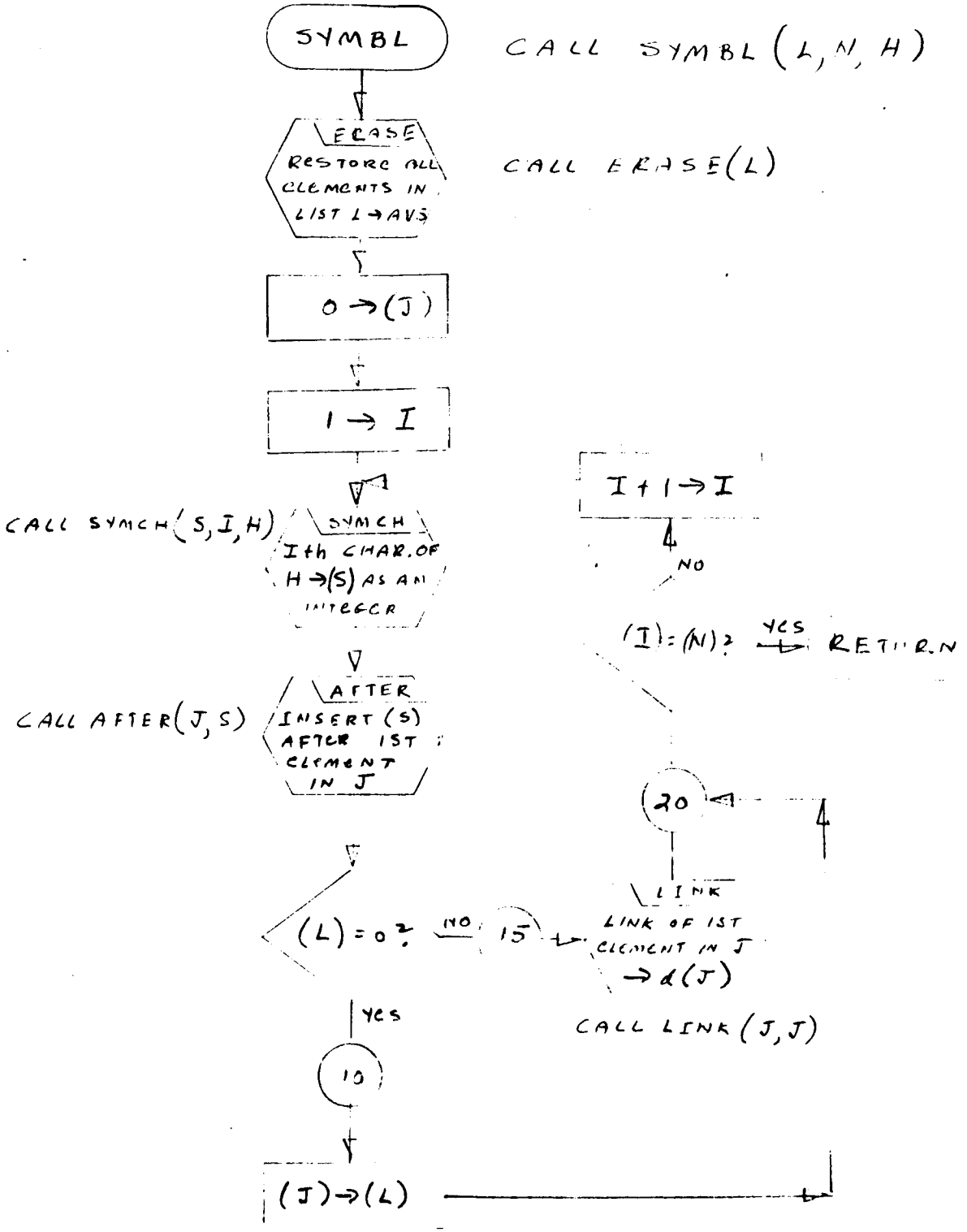
(b) If $I \neq N$, I is incremented by one and steps d(1), d(2), and d(3) are repeated until I = N.

7. Other Subroutines Used

AFTER, ERASE, LINK, SYMCH.

8. Using Subroutines

COMBN, DIMEN, DPDST, EQFS41, EQFV31, GOBLE, INPUTX, NLINDM, PARTS, STAT, SUBST, SYMCRD, WRTEQ, Main Program for Pass 2 of TAG Preprocessor.


SYMBL
CALL SYMBL (L, N, H)
ERASE
RESTORE ALL ELEMENTS IN LIST L → AVS
CALL ERASE(L)
0 → (J)
1 → I
CALL SYMCH (S, I, H)
SYMCH
Ith CHAR. OF H → (S) AS AN INTEGER
I + 1 → I
NO
(I) = (N) ? YES
RETURN
CALL AFTER (J, S)
AFTER
INSERT (S) AFTER 1ST ELEMENT IN J
20
LINK
LINK OF 1ST ELEMENT IN J → α(J)
CALL LINK (J, J)
(L) = 0 ? NO
15
YES
10
(J) → (L)

Program Description

1. Identification

a. Routine Label

SYMCH

b. Name

Place one character of a Hollerith array into the decrement of a word.

2. Function

The Nth character of a Hollerith array is placed into the decrement of a word as an integer (exactly like LNECH but with reversed calling sequence).

3. Programming System

FAP

4. Usage

a. Calling Sequence

CALL SYMCH (IX, N, BCDV)

b. Entry Conditions

N = Character position in BCDV
BCDV = An array of Hollerith characters

c. Exit Conditions

IX = Nth character of BCDV as an integer.

d. Error Exits

None.

5. Definition of Identifiers

SY90 = A table of six entries, each a shift operation

6. Method

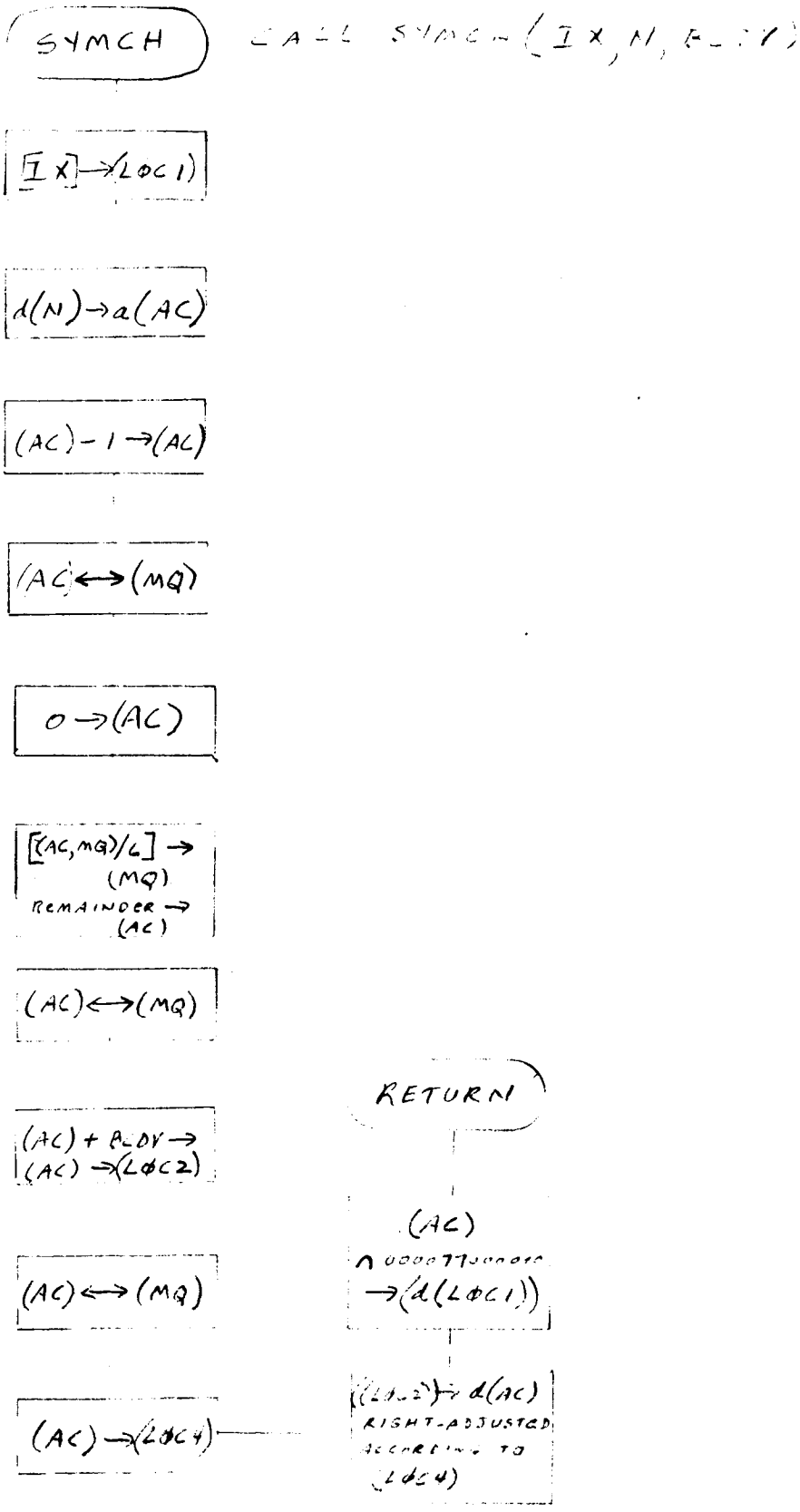
Each time the subroutine is entered, the character corresponding to the value of N is shifted into the decrement of IX as an integer, right-adjusted. A table, whose location is SY90, determines the appropriate shift.

7. Other Subroutines Used

None.

8. Using Subroutines

BLNOUT, COMBN, DBPCHC, DIMEN, DPDST, ELIM, EXCPT, EXTRX, FLTCON, GOBLE, HOLBK, IDNTC, IDNTP, INPUTX, INZERO, MATFT, MATOT, PARTS, PUSPCH, READCH, RECOVR, SUBST, SYMBL, SYMCRD, SYMTP, WRTEQ, ZEROX, Main Program for Pass 2 of TAG Preprocessor.


SYMCH
CALL SYMCH (IX, N, RLOV)
[IX] → (LOC 1)
d(N) → a(AC)
(AC) − 1 → (AC)
(AC) ↔ (MQ)
0 → (AC)
[(AC, MQ)/L] → (MQ)
REMAINDER → (AC)
(AC) ↔ (MQ)
(AC) + RLOV → (AC) → (LOC 2)
(AC) ↔ (MQ)
(AC) → (LOC 4)
RETURN
(AC)
→ (d(LOC 1))
→ d(AC)
RIGHT-ADJUSTED ACCORDING TO (LOC 4)

Program Description

1. Identification

   a. Routine Label

      SYMCRD

   b. Name

      Output a symbolic card

2. Function

   To output a series of cards of the form

   ABKL(M, N) = X*CDNONT

which establish the transformation from circuit parameters to TAG-defined arrays.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call SYMCRD (X, N, NPT, TYPEH, PARH, NLEV, NTAPE2)

   b. Entry Conditions

X	= The list structure from which the transformation is derived
N	= The maximum size of the I, J indices within X
NPT	= The NPT array of MAIN No. 2
TYPEH	= The two characters represented by "AB" above
PARH	= The two characters represented by "CD" above
NLEV	= The flag for the items to be extracted from X
NTAPE2	= The tape number of the tape from which X is acquired

c. Exit Conditions

X is updated by absorbing a file from NTAPE2.
Cards will be output.

d. Error Exits

None.

5. Definition of Identifiers

RIGHT	BCD Right parenthesis
COMMA	BCD Comma
LEFT	BCD Left parenthesis
EQUAL	BCD Equal sign
PLUS	BCD Plus sign
MINUS	BCD Minus sign
TYPE	List (type A) containing the two characters of TYPEH
PAR	List (type A) containing the two characters of PARH
I	I (index to X)
J	J (index to X)
ITERMS	Used to count items in X
P	List (type A) used to construct the output card image
LJ, L2, L4, L5, L6	Used to hold links while examining X
KSW	Indicator used in scanning X
NEF	Used to hold flag portion of a list element
IVECT	Holds second character of TYPE
NXP, IX, IP, NSP	Used to compute K, L, M, N from I, J, NPT

KYN NZ1 NZ2	Indicators used in examining X
NSIJ	Used to hold M, N parameters
ITST	Temporary storage, holds character "I" or "V" for testing
NF	Holds a flag portion of list X
NO	Node number extracted from X
NT	Node number extracted from X
XX	Decimal number associated with I, J, NO, NT

6. Method

a. A file is read from NTAPE2 and merged into X using MATFT. The characters in TYPE are pushed into P.

b. For all I, J (I, J = 1, N), X is searched, using LOCATA, to find a sublist entered under I and J which has a flag portion = NLEV. If none is found, the search through X is restarted using new values for I, J.

c. If found, then K, L, M, N are computed from I, J, and NPT values. For I, K would indicate the submatrix number and M the index within that submatrix where I is found. L and N are computed the same way for J; i.e.,

$$J = N + \sum_{r=1}^{L-1} NPT(r)$$

As they are computed, K and M are pushed into P; L and N are saved in NSIJ(1) and NSIJ(2).

d. The symbol so constructed in P is sent to EXCPT for testing; If it is illegal, a CALL DUMP is executed.

e. L and N are pushed into P, enclosed in parentheses, and followed by an =:

(L, N) =

f. The sublist of X is then examined further to see if NO, NT are present. If an element is found that contains a 5 in the

flag portion, then the item portion (pointing to a BCD symbol) is pushed into P and output as a card. This would result in a statement like this:

ABKL(M, N) = 'variable name'

If a flag of 5 is not found, the sublist is searched for NO and NT. If found, then the link following NT points to a decimal number. This number is converted to BCD and pushed into P, followed by an asterisk.
The characters in PAR are pushed into P, followed by NO and NT, and the string in P is output. This would be of the form

ABKL(M, N) = X*CDNONT

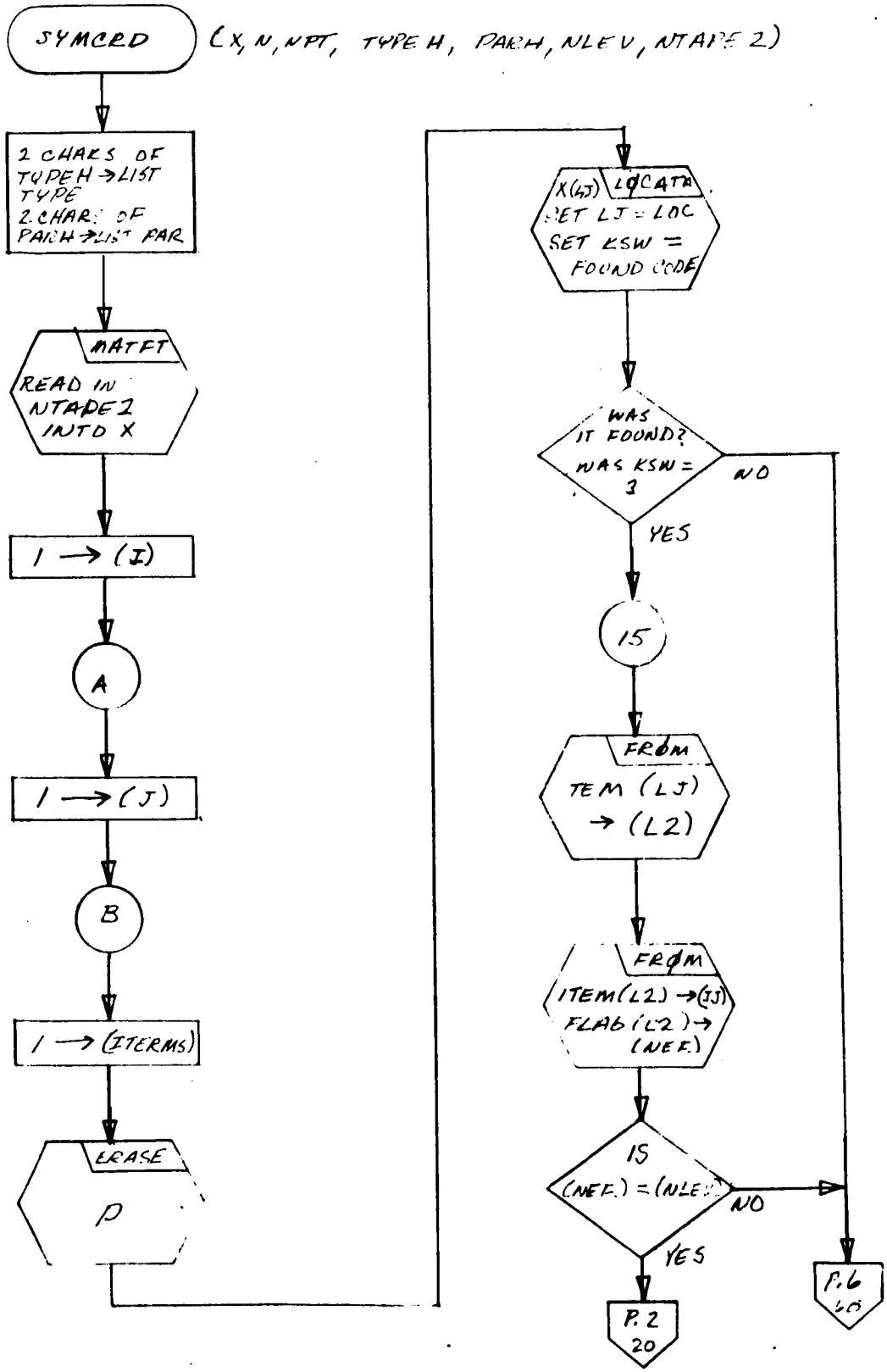
K, L are digits; M, N, NO, NT are integers; X is a decimal number; 'AB' is the contents of TYPEH, and 'CD' is the contents of PARH.

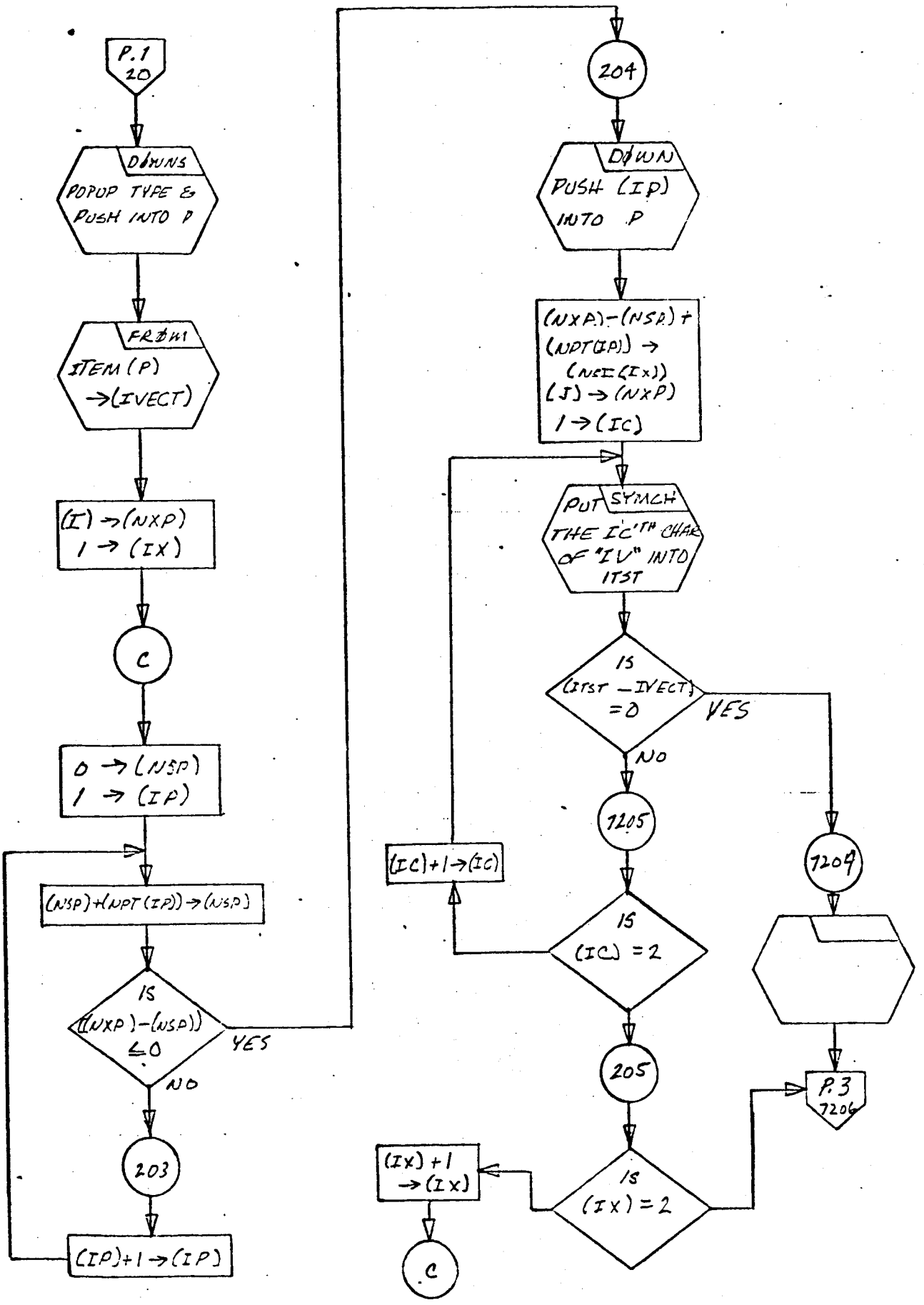
7. Other Subroutines Used

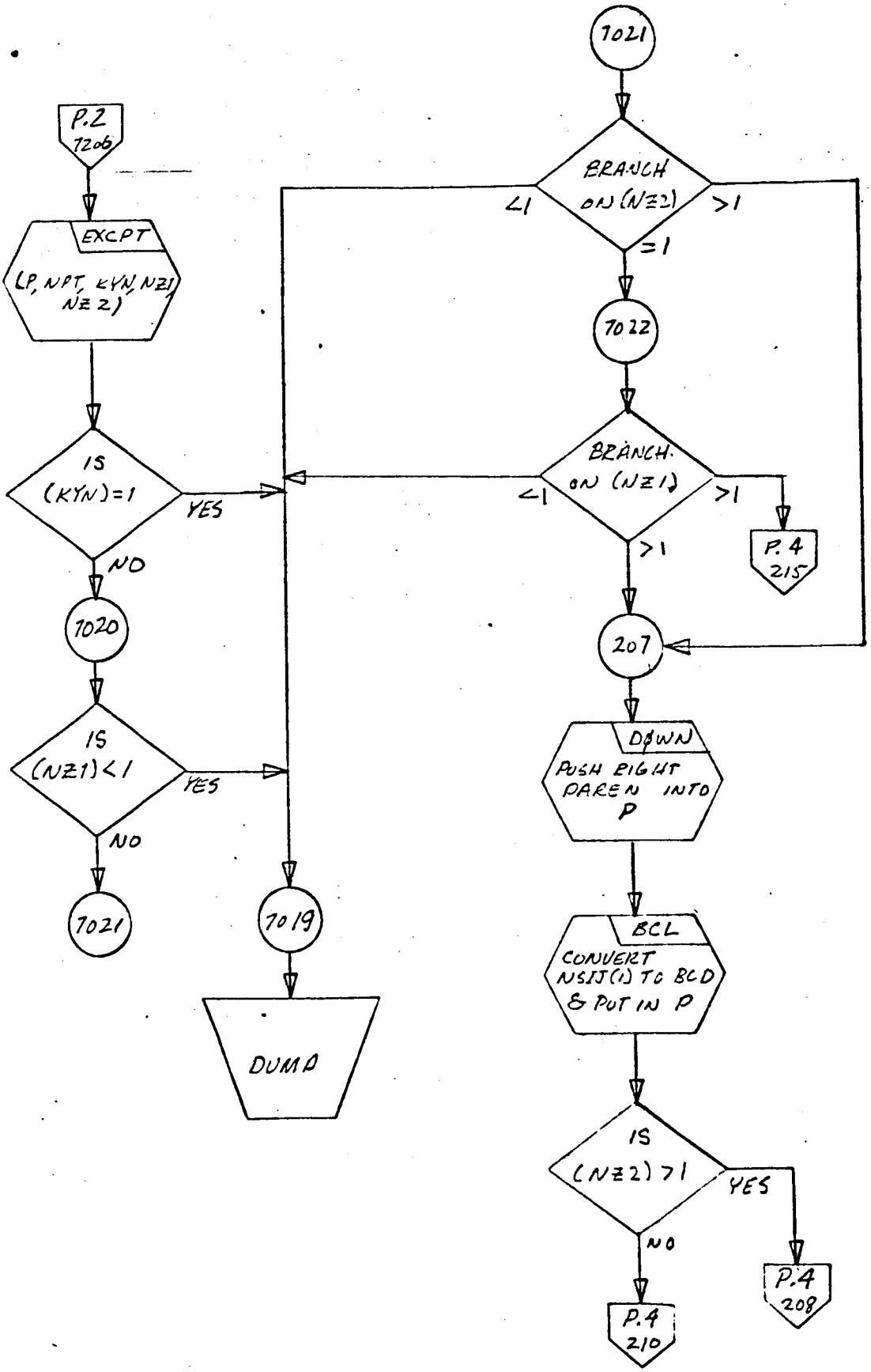
BCL, DBPCH, DONBD, DOWN, DOWNS, ERASEA, ERASE, EXCPT, FLTCON, FRFL, FROM, LINK, LOCATA, MATFT, SYMBL, SYMCH.

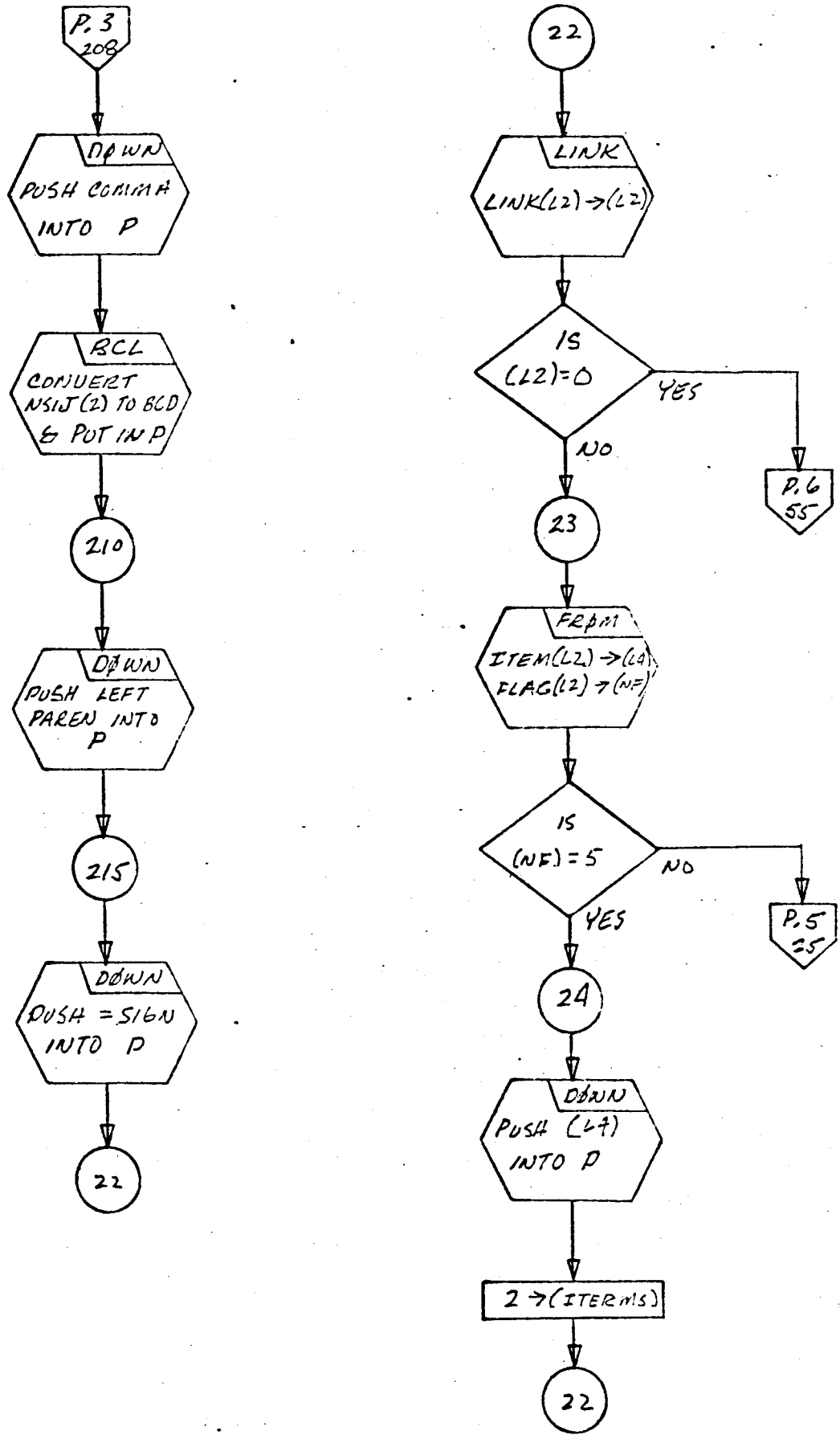
8. Using Subroutines

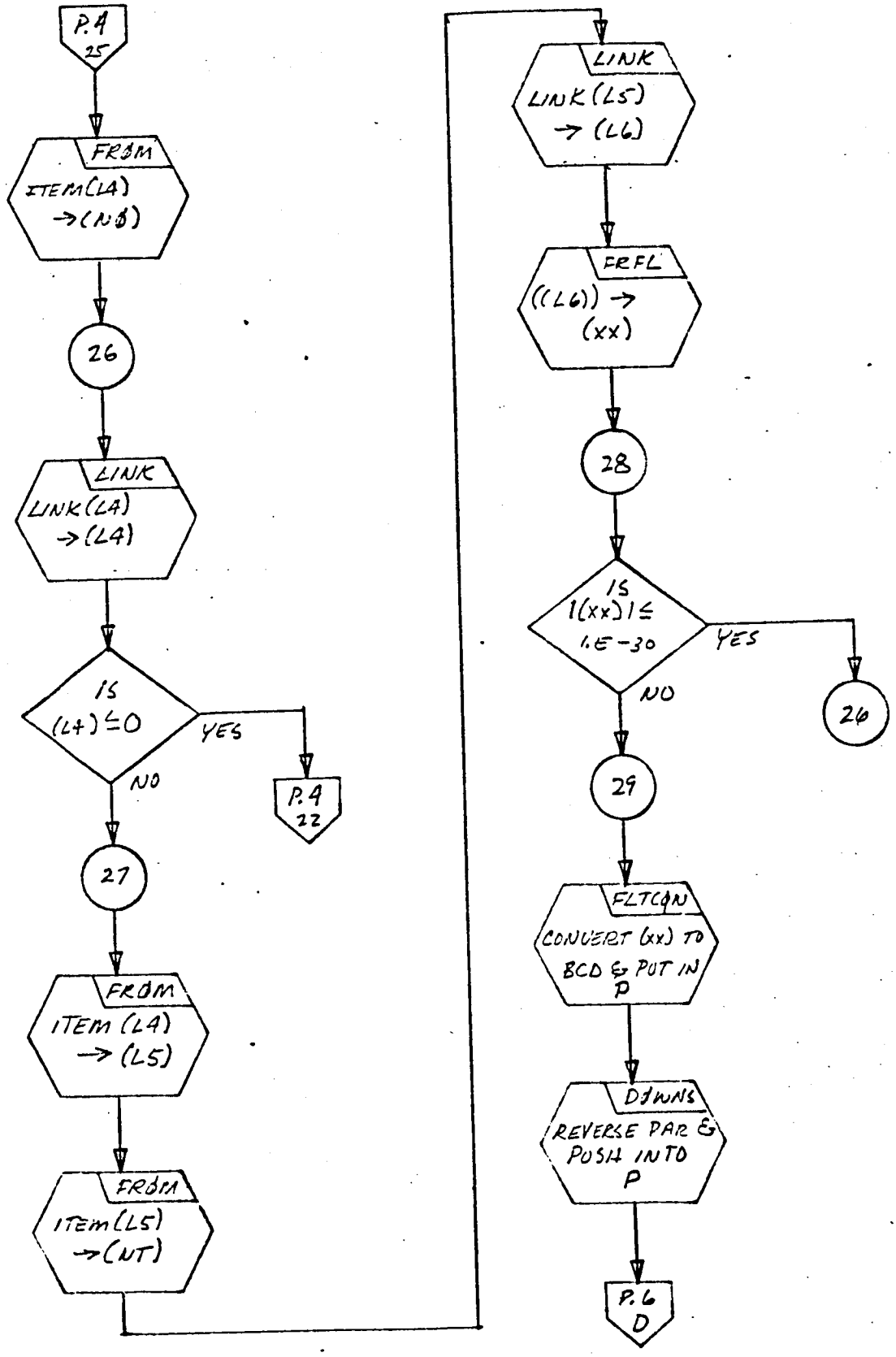
Main Program for Pass 2 of TAG Preprocessor.

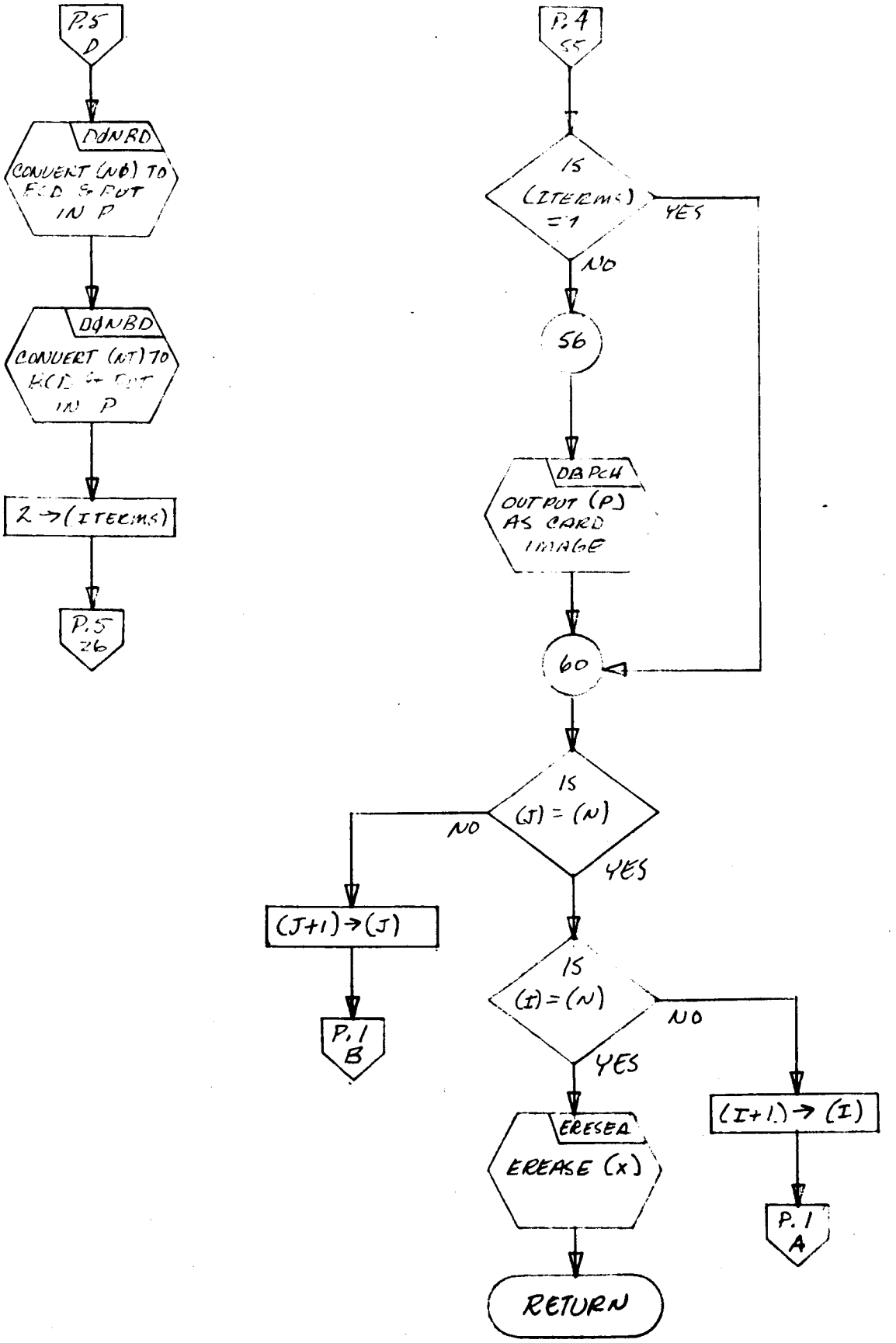

SYMCRD (X, N, NPT, TYPEH, PARH, NLEV, NTAPE 2)
2 CHARS OF TYPEH → LIST TYPE
2 CHARS OF PARH → LIST PAR
MATFT
READ IN NTAPE 2 INTO X
1 → (I)
A
1 → (J)
B
1 → (ITERMS)
ERASE
P
X(4J) LOCATN
SET LJ = LOC
SET KSW = FOUND CODE
WAS IT FOUND? WAS KSW = 3
NO
YES
15
FROM
TEM (LJ) → (L2)
FROM
ITEM(L2) → (IJ)
FLAB(L2) → (NEF)
IS (NEF) = (NLE
NO
YES
P. 2
20
P. 6


P.1
20
DOWNS
POPUP TYPE &
PUSH INTO P
FROM
ITEM (P)
→(IVECT)
(I) →(NXP)
1 → (IX)
C
0 →(NSP)
1 → (IP)
(NSP)+(NPT(IP))→(NSP)
IS
((NXP)-(NSP))
≤0
YES
NO
203
(IP)+1→(IP)
204
DOWN
PUSH (IP)
INTO P
(NXP)-(NSP)+
(NPT(IP)) →
(NEI(IX))
(J) → (NXP)
1→(IC)
PUT SYNCH
THE IC'TH CHAR
OF "IV" INTO
ITST
IS
(ITST -IVECT)
=0
YES
NO
7205
(IC)+1→(IC)
IS
(IC) = 2
7209
205
P.3
7206
(IX)+1
→(IX)
IS
(IX) = 2
C


P.2
7206
EXCPT
(LP, NPT, KYN, NZ1, NZ2)
IS
(KYN)=1
YES
NO
7020
IS
(NZ1)<1
YES
NO
7021
7019
DUMP
7021
BRANCH
ON (NZ2)
<1
>1
=1
7022
BRANCH
ON (NZ1)
<1
>1
>1
P.4
215
207
DOWN
PUSH RIGHT PAREN INTO P
BCL
CONVERT NSIJ(1) TO BCD & PUT IN P
IS
(NZ2)>1
YES
NO
P.4
208
P.4
210


P. 3
208
DOWN
PUSH COMMA
INTO P
BCL
CONVERT
NSIJ(2) TO BCD
& PUT IN P
210
DOWN
PUSH LEFT
PAREN INTO
P
215
DOWN
PUSH =SIGN
INTO P
22
22
LINK
LINK(L2)→(L2)
IS
(L2)=0
YES
NO
P. 6
55
23
FRPM
ITEM(L2)→(L4)
FLAG(L2)→(NF)
IS
(NF)=5
NO
YES
P. 5
25
24
DOWN
PUSH (L4)
INTO P
2→(ITERMS)
22


P.4
25
FROM
ITEM(L4)
→(N$)
26
LINK
LINK(L4)
→(L4)
IS
(L4)≤0
YES
NO
P.4
22
27
FROM
ITEM(L4)
→(L5)
FROM
ITEM(L5)
→(NT)
LINK
LINK(L5)
→(L6)
FRFL
((L6))→
(xx)
28
IS
|(xx)|≤
1.E-30
YES
NO
26
29
FLTCON
CONVERT (xx) TO
BCD & PUT IN
P
DOWNS
REVERSE PAR &
POSN INTO
P
P.6
D


P.5
D
DONRD
CONVERT (NB) TO
BCD & PUT
IN P
DONBD
CONVERT (NT) TO
BCD & PUT
IN P
2 → (ITEMS)
P.5
26
P.4
55
IS
(ITEMS)
=1
YES
NO
56
DBPCH
OUTPUT (P)
AS CARD
IMAGE
60
IS
(J) = (N)
NO
YES
(J+1) → (J)
P.1
B
IS
(I) = (N)
NO
YES
ERESEA
ERASE (x)
RETURN
(I+1) → (I)
P.1
A

Program Description

1. Identification

   a. Routine Label

      SYMTP

   b. Name

      Check for symbol.

2. Function

   Examines a symbolic name.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      Call SYMTP (TEM, KS, KSTB, INLST, NE, NO, NT)

   b. Entry Conditions

      TEM = List containing the characters of the symbol
      KSTB = Contains a constant

   c. Exit Conditions

      See "Method."

   d. Error Exits

      None.

5. Definition of Identifiers

TEM1, TEM2	Temporary lists
IDECP	BCD decimal point
IKND1	BCD "S"
I	Temporary index

IOP, ITEM, ISF	Temporary to hold a character
J	Temporary index
TEMIX	Used to hold a link

6. Method

a. If (KSTB) = 3, set KS = 2.
If (TEM) contains $UTF
Set NE = 0
NO = 0
NT = 0
Else set KS = 3
Return

b. If (KSTB) ≠ 3, set KS = 1.
If (TEM) contains

(1) FT
Set NE = 7
NO = 0
NT = 0

(2) SXn1n2
Set NE = 1 to 6

for "X" = V	Set NE = 1
C	2
G	3
L	4
I	5
S	6

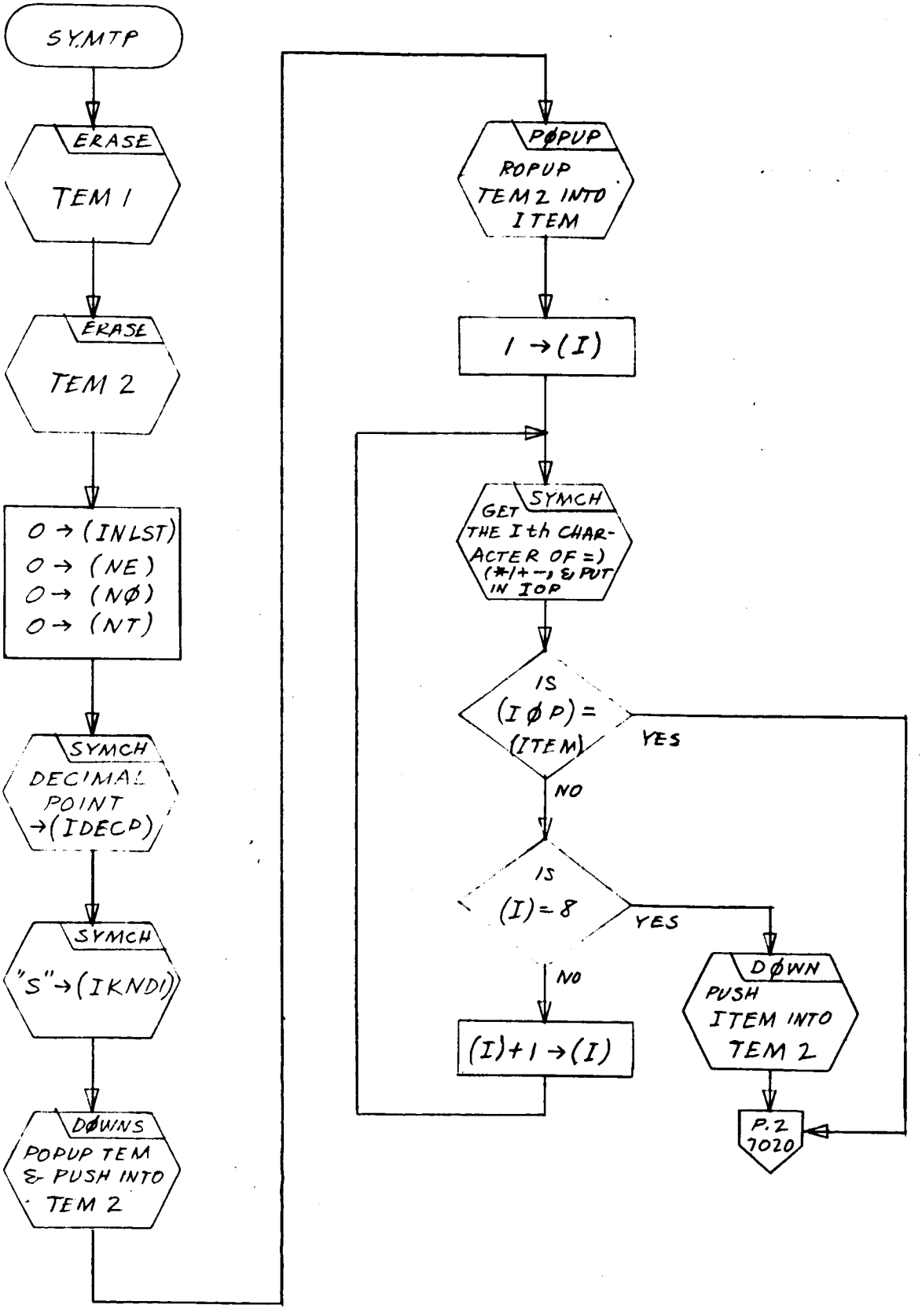
Set NO = n1
Set NT = n2

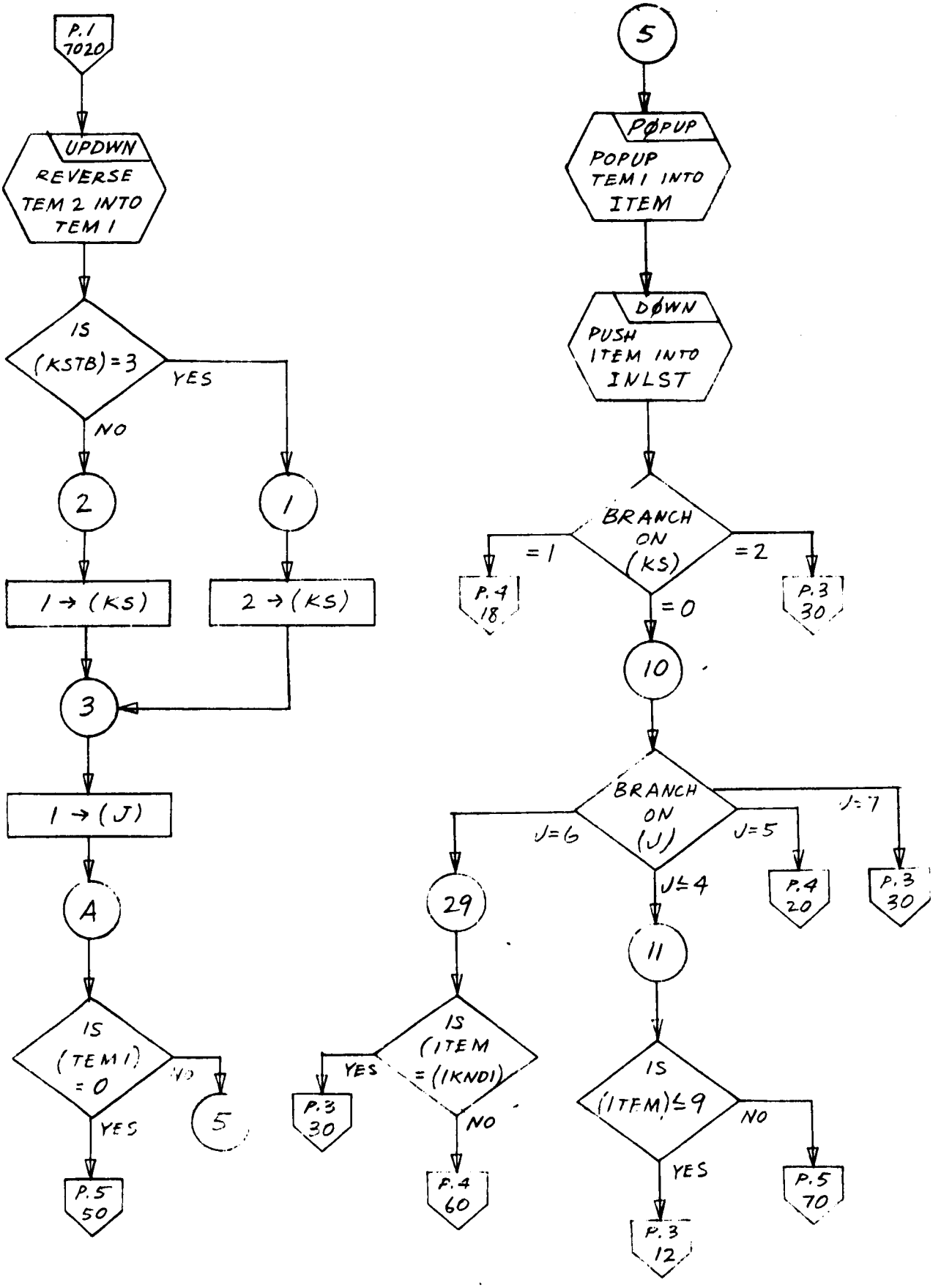
(3) Else set KS = 3
Return

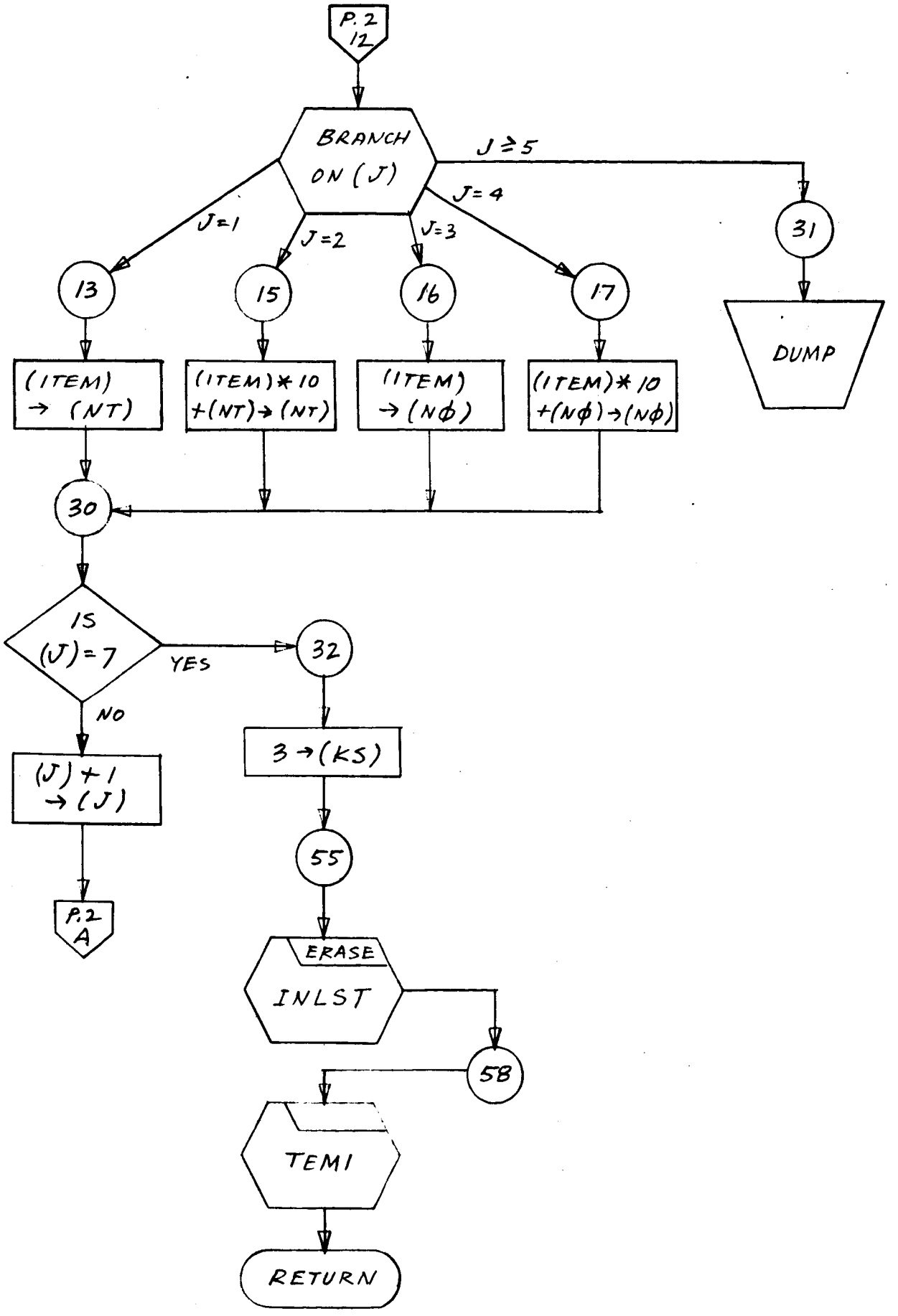
c. Except in illegal case (KS = 3), put c(TEM) into INLST.
Else erase INLST.

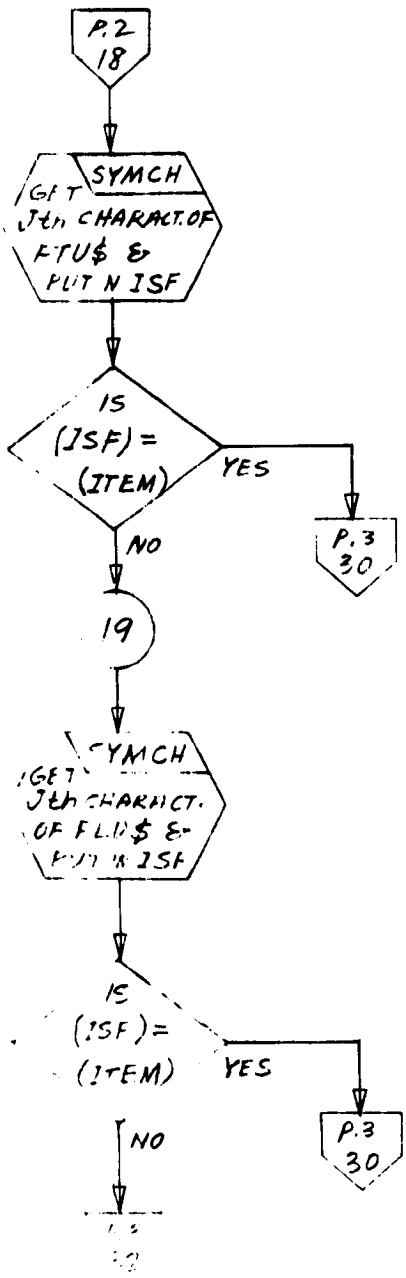
7. Other Subroutines Used

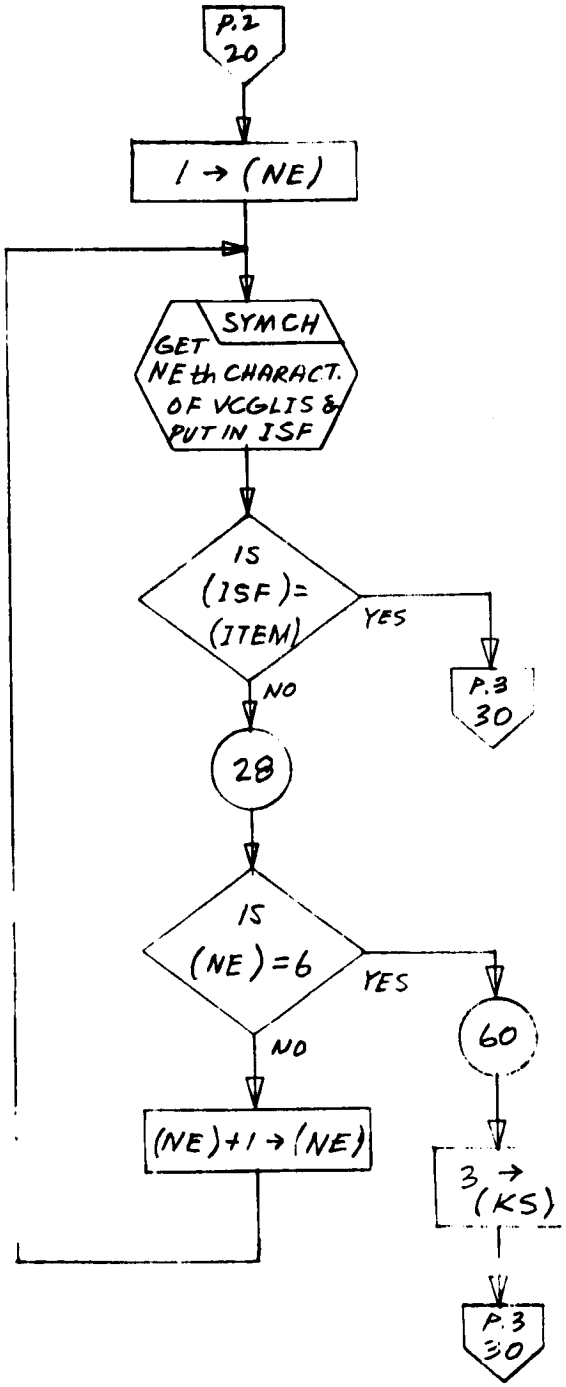
DOWN, DOWNS, ERASE, FROM, LINK, POPUP, SYMCH, UPDWN.

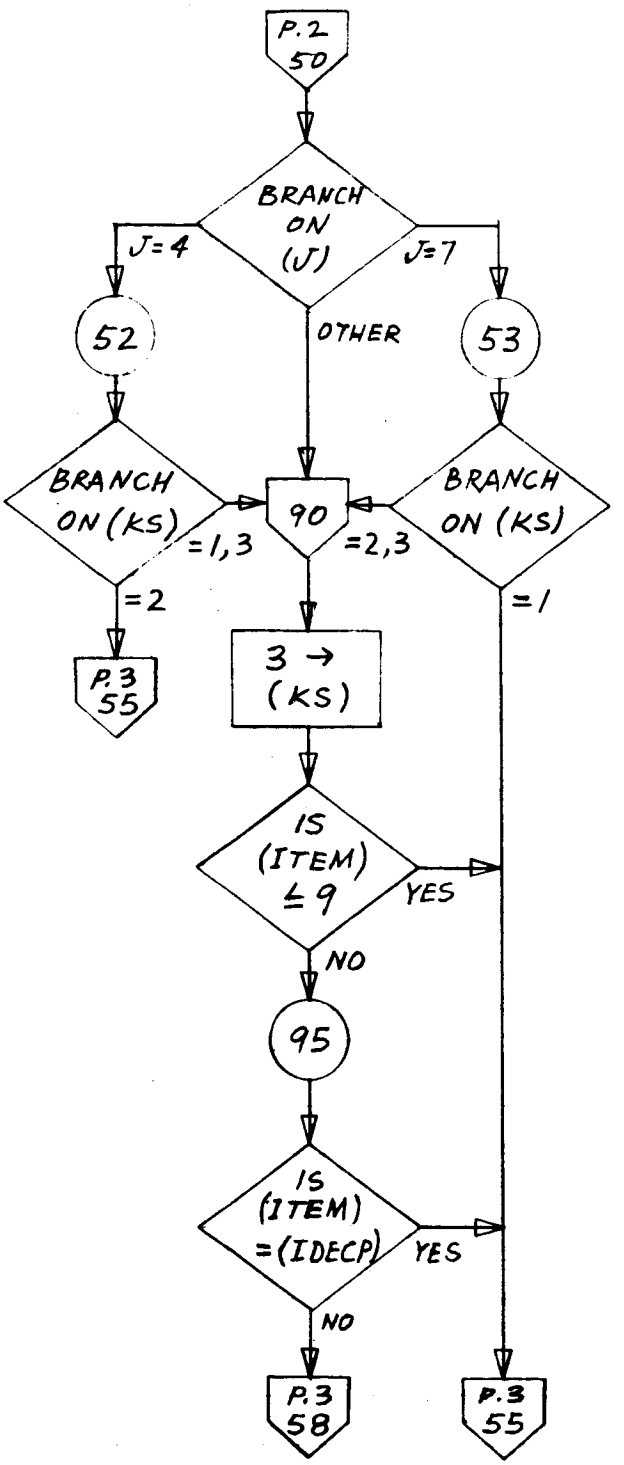

SYMTP
ERASE
TEM 1
ERASE
TEM 2
0 → (INLST)
0 → (NE)
0 → (NØ)
0 → (NT)
SYMCH
DECIMAL POINT →(IDECP)
SYMCH
"S" → (IKNDI)
DØWNS
POPUP TEM & PUSH INTO TEM 2
PØPUP
POPUP TEM 2 INTO ITEM
1 → (I)
SYMCH
GET THE I th CHARACTER OF =) (*/+ −, & PUT IN IOP
IS (IØP) = (ITEM)
YES
NO
IS (I) = 8
YES
NO
DØWN
PUSH ITEM INTO TEM 2
(I) + 1 → (I)
P.2 7020

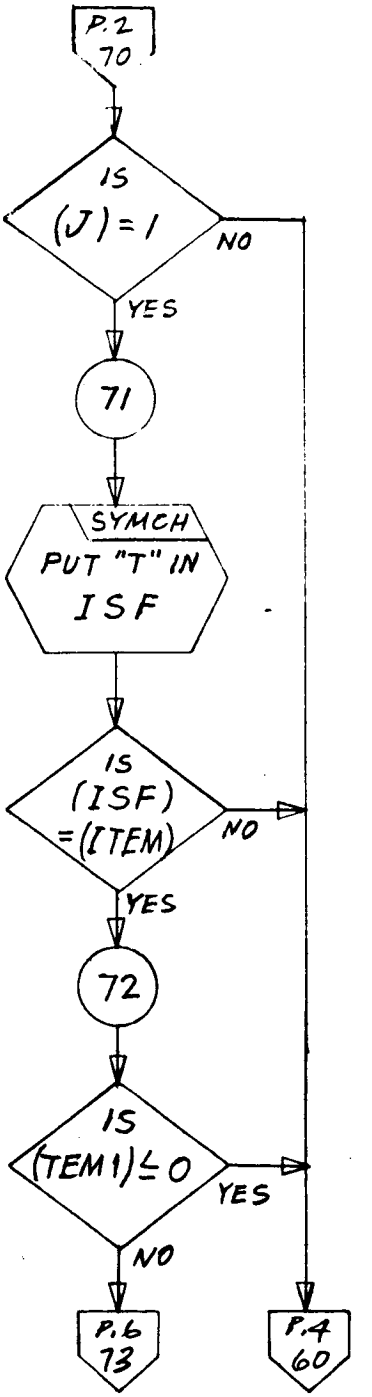

P.1
7020
UPDWN
REVERSE
TEM 2 INTO
TEM 1
IS
(KSTB) = 3
YES
NO
2
1
1 → (KS)
2 → (KS)
3
1 → (J)
A
IS
(TEM 1)
= 0
NO
YES
5
P.5
50
5
POPUP
POPUP
TEM 1 INTO
ITEM
DOWN
PUSH
ITEM INTO
INLST
BRANCH
ON
(KS)
=1
=2
=0
P.4
18
P.3
30
10
BRANCH
ON
(J)
J=6
J=5
J=7
J≤4
29
11
P.4
20
P.3
30
IS
(ITEM
= (IKND1)
YES
NO
P.3
30
P.4
60
IS
(ITEM) ≤ 9
NO
YES
P.5
70
P.3
12

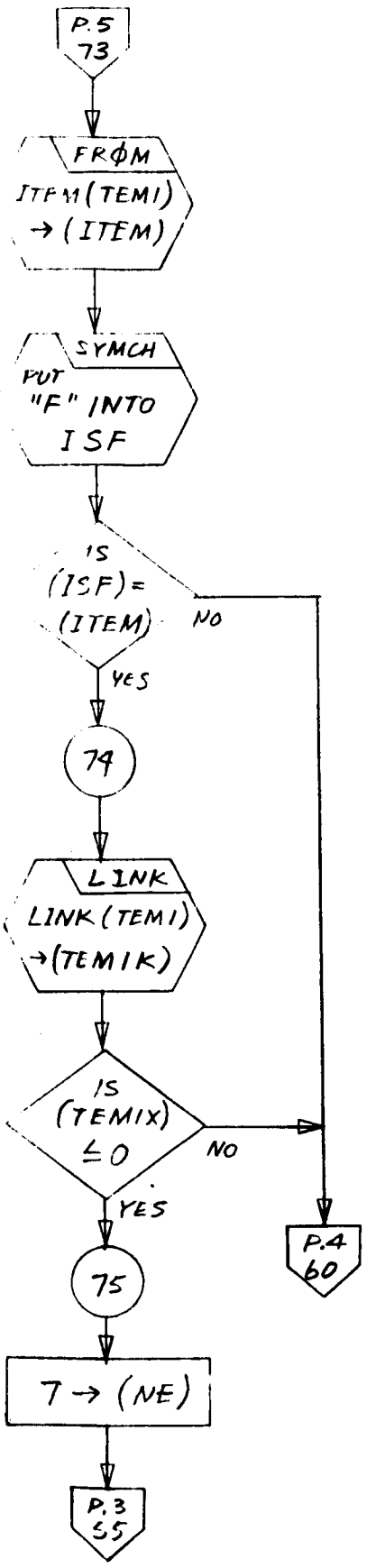

P. 2
12
BRANCH
ON (J)
J ≥ 5
J = 4
J = 3
J = 2
J = 1
31
13
15
16
17
DUMP
(ITEM)
→ (NT)
(ITEM) * 10
+(NT) → (NT)
(ITEM)
→ (Nϕ)
(ITEM) * 10
+(Nϕ) → (Nϕ)
30
IS
(J) = 7
YES
32
NO
3 → (KS)
(J) + 1
→ (J)
55
P. 2
A
ERASE
INLST
58
TEMI
RETURN


P.2
18
SYMCH
GET
Jth CHARACT. OF
FTU$ &
PUT N ISF
IS
(JSF) =
(ITEM)
YES
P.3
30
NO
19
SYMCH
GET
Jth CHARACT.
OF FLD$ &
PUT N ISF
IS
(JSF) =
(ITEM)
YES
P.3
30
NO


P.2
20
1 → (NE)
SYMCH
GET
NE th CHARACT.
OF VCGLIS &
PUT IN ISF
IS
(ISF) =
(ITEM)
YES
P.3
30
NO
28
IS
(NE) = 6
YES
60
NO
(NE) + 1 → (NE)
3 →
(KS)
P.3
30


P.2
50
BRANCH
ON
(J)
J=4
J=7
OTHER
52
53
BRANCH
ON (KS)
=1,3
=2
90
=2,3
BRANCH
ON (KS)
=1
P.3
55
3 →
(KS)
IS
(ITEM)
≤ 9
YES
NO
95
IS
(ITEM)
=(IDECP)
YES
NO
P.3
58
P.3
55


P.2
70
IS
(J) = 1
NO
YES
71
SYMCH
PUT "T" IN
ISF
IS
(ISF)
=(ITEM)
NO
YES
72
IS
(TEM1) ≤ 0
YES
NO
P.6
73
P.4
60


P.5
73
FRΦM
ITEM(TEMI)
→(ITEM)
SYMCH
PUT
"F" INTO
ISF
IS
(ISF)=
(ITEM)
NO
YES
74
LINK
LINK(TEMI)
→(TEMIK)
IS
(TEMIX)
≤0
NO
YES
P.4
60
75
7→(NE)
P.3
55

Program Description

1. Identification

   a. Routine Label

      TRANS

   b. Name

      Transpose a two-dimensional matrix.

2. Function

The transpose of an NM x NMR matrix TF (in array-list format) is created as NMR x NM matrix TFT (also in array-list format) by extracting each element from its position I, J in TF and placing it in position J, I of TFT.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL TRANS (TF, TFT, NM, NMR)

   b. Entry Conditions

      TF = The final coordinate transformation matrix with NM rows and NMR columns. TF expresses all the node voltages as linear combinations of a reduced set of true voltages.

      NM = The maximum node numbers in the connection list

      NMR = NM - NRR

      NRR = The total number of transformer windings minus the number of different transformers

   c. Exit Conditions

The NMR x NM matrix, TFT, has been created in array-list format such that [TFT](I, J) = [TF](J, I).

d. Error Exits

None.

5. Definition Identifiers

I = Row index for TF, column index for TFT

J = Column index for TF, row index for TFT

FTEM = Local variable which holds the data word identified by I, J

6. Method

The matrix transpose of TF into TFT is performed as follows:

a. Array-list TFT is erased.

b. The row index I for TF is initialized to 1.

c. The column index J for TF is initialized to 1.

d. Using SNATCH, list TF is searched for data identified by I, J. When found, the data is saved in FTEM.

e. If FTEM $\neq$ 0, the subroutine STASH is executed to insert the data word into TFT, identified by J, I.

f. J is tested:

(1) If J = NMR, I is tested.

(a) If I = NM, the transpose is complete and exit is made from the routine.

(b) If I $\neq$ NM, I is increased by 1 and execution returns to step c to continue the search with a new I.

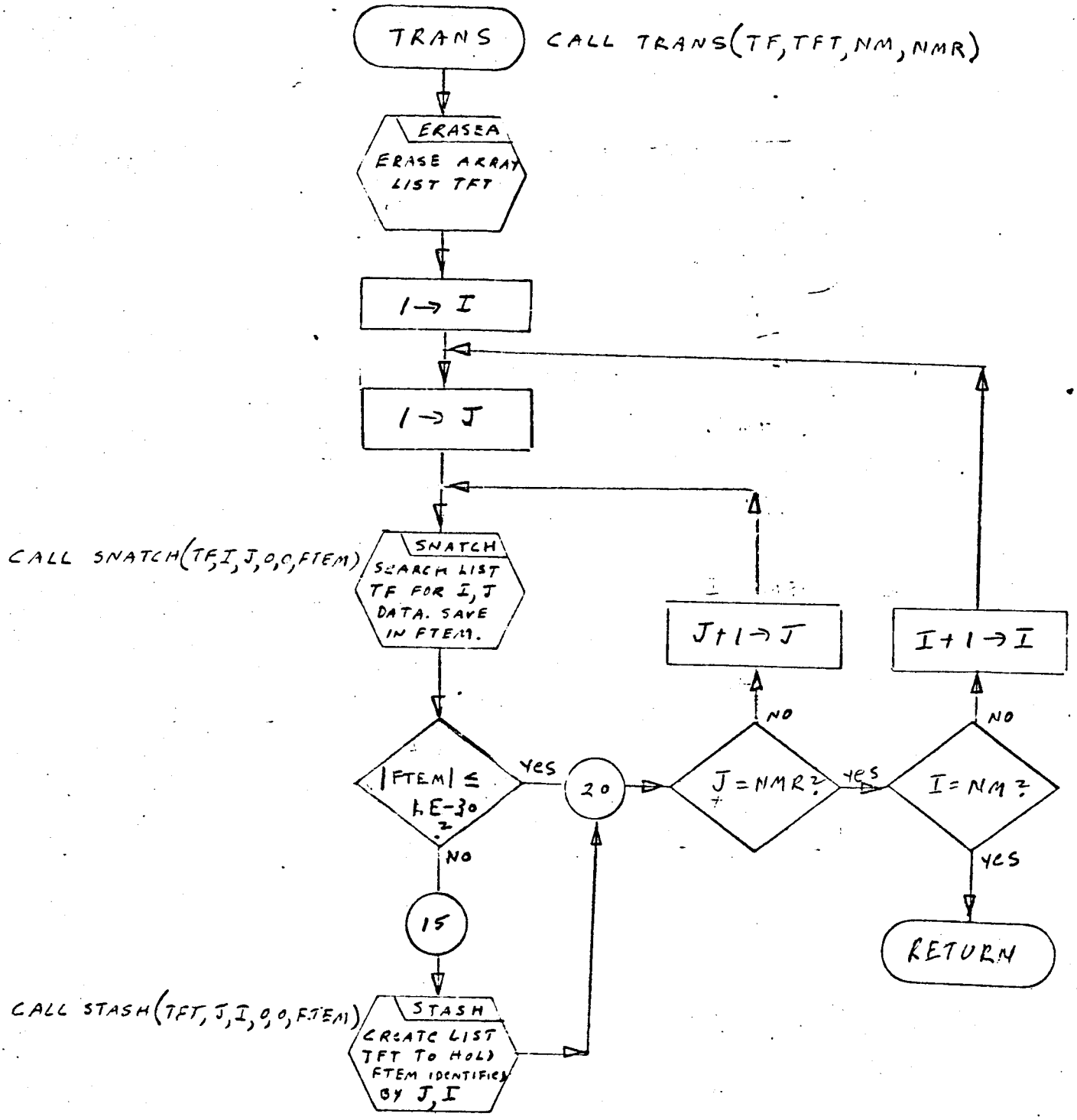
(2) If J $\neq$ NMR, J is increased by 1 and execution returns to step d to continue the search with a new J.

7. Other Subroutines Used

ERASEA, SNATCH, STASH.

8. Using Subroutines

Main Program for Pass 1 of TAG Preprocessor.


TRANS
CALL TRANS(TF,TFT,NM,NMR)
ERASEA
ERASE ARRAY LIST TFT
1 → I
1 → J
CALL SNATCH(TF,I,J,0,0,FTEM)
SNATCH
SEARCH LIST TF FOR I,J DATA. SAVE IN FTEM.
J+1 → J
I+1 → I
|FTEM| ≤ 1.E-30
yes
NO
20
J = NMR ?
yes
NO
I = NM ?
yes
15
RETURN
CALL STASH(TFT,J,I,0,0,FTEM)
STASH
CREATE LIST TFT TO HOLD FTEM IDENTIFIED BY J,I

Program Description

1. Identification

   a. Routine Label

      TREE

   b. Name

      Form a proper TAG tree from PLIST.

2. Function

The tree routine forms a special type of proper C tree from the ordered connection list, PLIST. The proper C tree covers all nodes while forming no loops and contains all voltage sources, a maximum of capacitive elements, a minimum of inductive elements, and no current sources. The TAG tree is a directed graph in which every node but one has one and only one branch positively incident with it. The single exception is the O node, which has no branches positively incident with it. Tree is formed as a list of element descriptors in the type D, two-dimensional list, FLIST.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL TREE (PLIST, FLIST, NM)

   b. Entry Conditions

      PLIST = A copy of the connection list, WLIST, whose branches have been ordered according to element type in a VCG LNI sequence. PLIST, like WLIST, is in type D format.

      NM = The maximum node number, which is equal to the total number of nodes less one and to the number of elements that should be in the tree.

c. Exit Conditions

FLIST = A two-dimensional type D list which holds the NM branch descriptors of the proper TAG tree. The order of PLIST is not maintained in FLIST.

d. Error Exits

(1) If a tree of NM elements cannot be found which connects to node O and is completely connected itself, the error comment "PLIST EXHAUSTED TREE" is printed out, and a CALL DUMP is executed.

(2) If a voltage source is excluded from the tree, the error comment "VOLTAGE CODE/DUP. NODE/TREE is printed out and a CALL DUMP is executed.

5. Definition of Identifiers

FLIST is a local variable name for XLIST.

NFL keeps a running count of the number descriptors that have been entered into FLIST.

X1, XL1, and X2 are local variable head cells used to search PLIST and FLIST.

NE, NN, NP, NTRN, NNTR, and NF are the constituents of the descriptors extracted from PLIST.

NEF, NNF, NPF, NTRNF, NNTRF, and NFF are the constituents of the descriptor extracted from FLIST.

NENT is a flag used to indicate whether NN and/or NP is already covered by FLIST.

NENT = 1 (neither NN or NP are in FLIST)

NENT = 2 (at least NN is in FLIST)

NENT = 3 (at least NP is in FLIST)

6. Method

The proper TAG tree is constructed in FLIST from the branches listed in PLIST to cover all nodes and form no loops. The process starts at node zero and proceeds one branch at a time to build sucessively larger but always connected partial trees until the entire network

is covered. The next eligible tree branch, which must always have one of its nodes covered by the partial tree, is always selected by searching down PLIST from the top. Because PLIST is ordered in VCGLNI sequence, priority for membership in the tree also has the VCGNLI order. If the network contains no voltage source loops or current source cut-sets, the process will ensure that the final tree will contain all voltage sources, a maximum of capacitors, a minimum of inductors, and no current sources. In addition, the tree is constructed such that every node except the zero node is touched by the positive node of one and only one tree branch. Tree branches incident with the 0 node touch it with their negative node only. The algorithm that mechanizes the tree selection process is described below.

a. Erase FLIST and set the tree branch counter, NFL, to one.

b. Search PLIST from the top for the descriptor of the first branch which touches the 0 node. If, for this descriptor, NN = 0, push down the descriptor, as is, into FLIST. If NP = 0, reverse the node order and push the resultant descriptor down into FLIST. Proceed to step d. If no such element is found proceed to step c.

c. PRINT "PLIST EXHAUSTED TREE" and CALL DUMP.

d. Flag the descriptor in PLIST that was just placed in FLIST. Test NFL. If NFL = NM, control is returned to the main program. If NFL < NM, increase NFL by 1 and continue to step e.

e. Search PLIST, starting at the top, for the next unflagged element which has one or both of its nodes already covered by the partial tree in FLIST.

(1) If no such element is found go to step c.

(2) If both nodes are already covered by FLIST, flag the element descriptor in PLIST. If the element is a voltage source PRINT, "VOLTAGE CODE/DUP. NODE/TREE" and CALL DUMP. If the element is not a voltage source descriptor, but is the last entry in PLIST, go to step c. Otherwise start step e over.

(3) If only the negative node of the element is already covered by the partial tree in FLIST, the element descriptor is pushed down, as is, into FLIST and the process returns to step d.

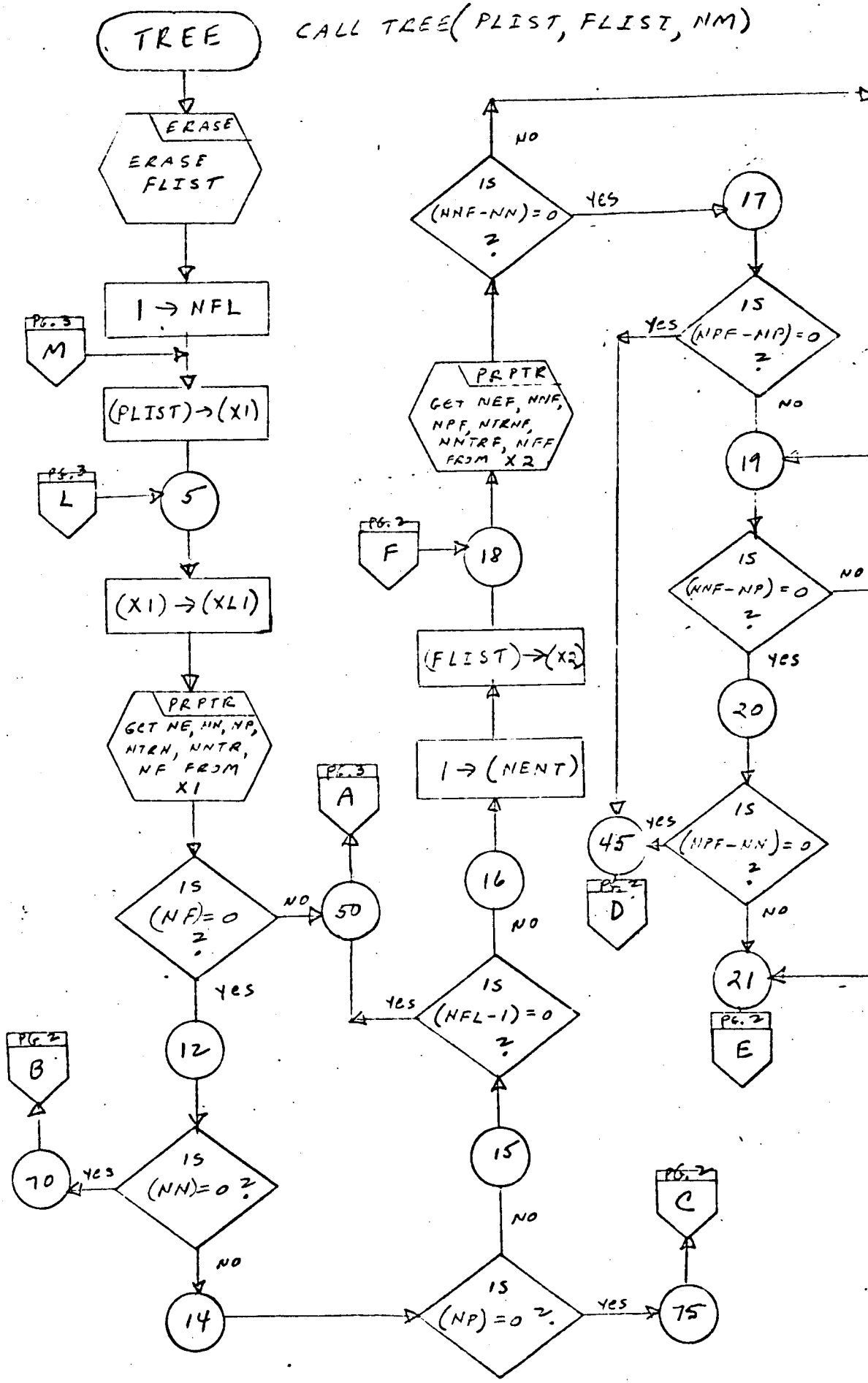
(4) If only the positive node of the element is already covered by the partial tree in FLIST, the node numbers are reversed and the resultant element descriptor is pushed down into FLIST. The process returns to step d.

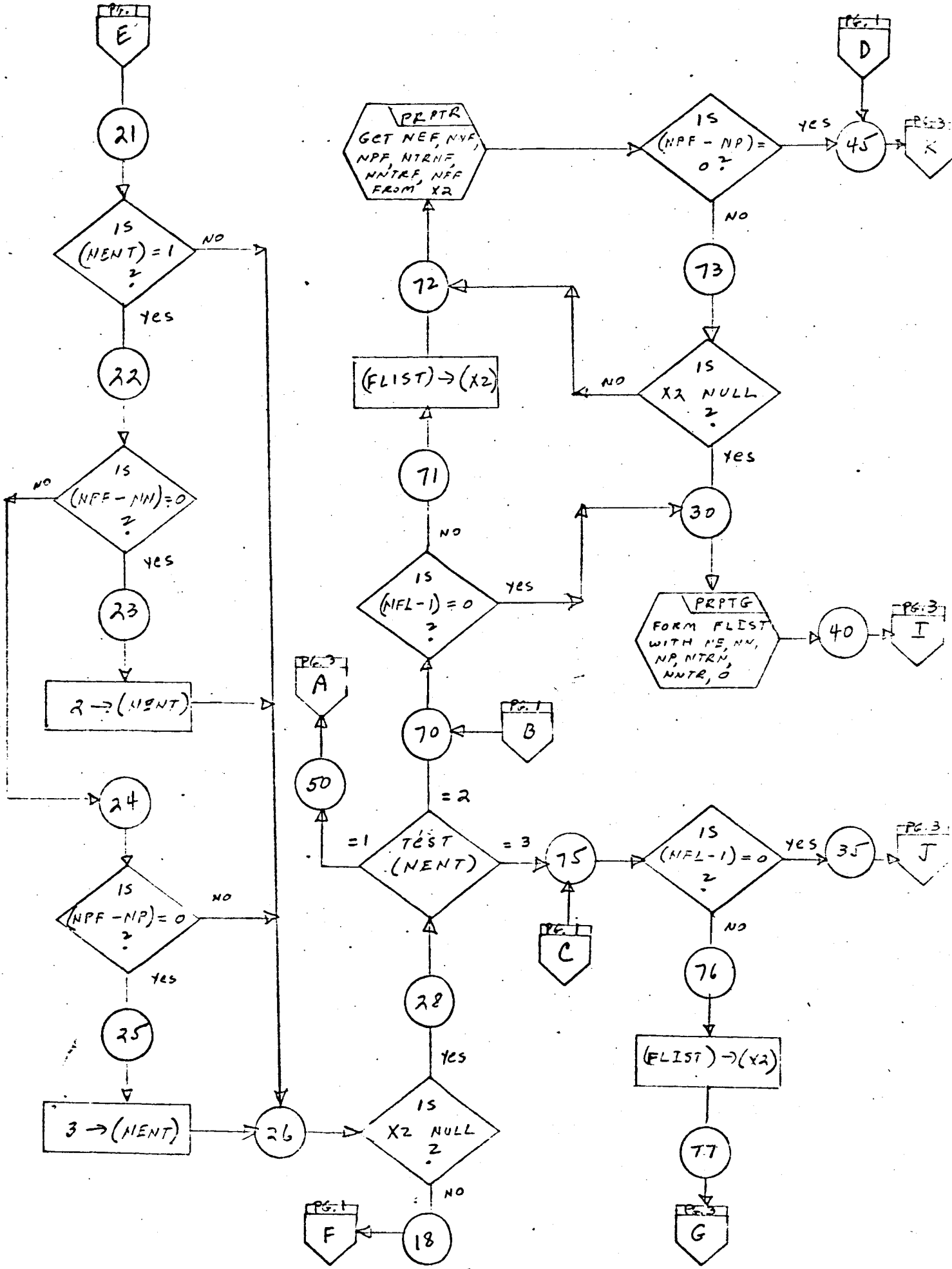
7. Other Subroutines Used

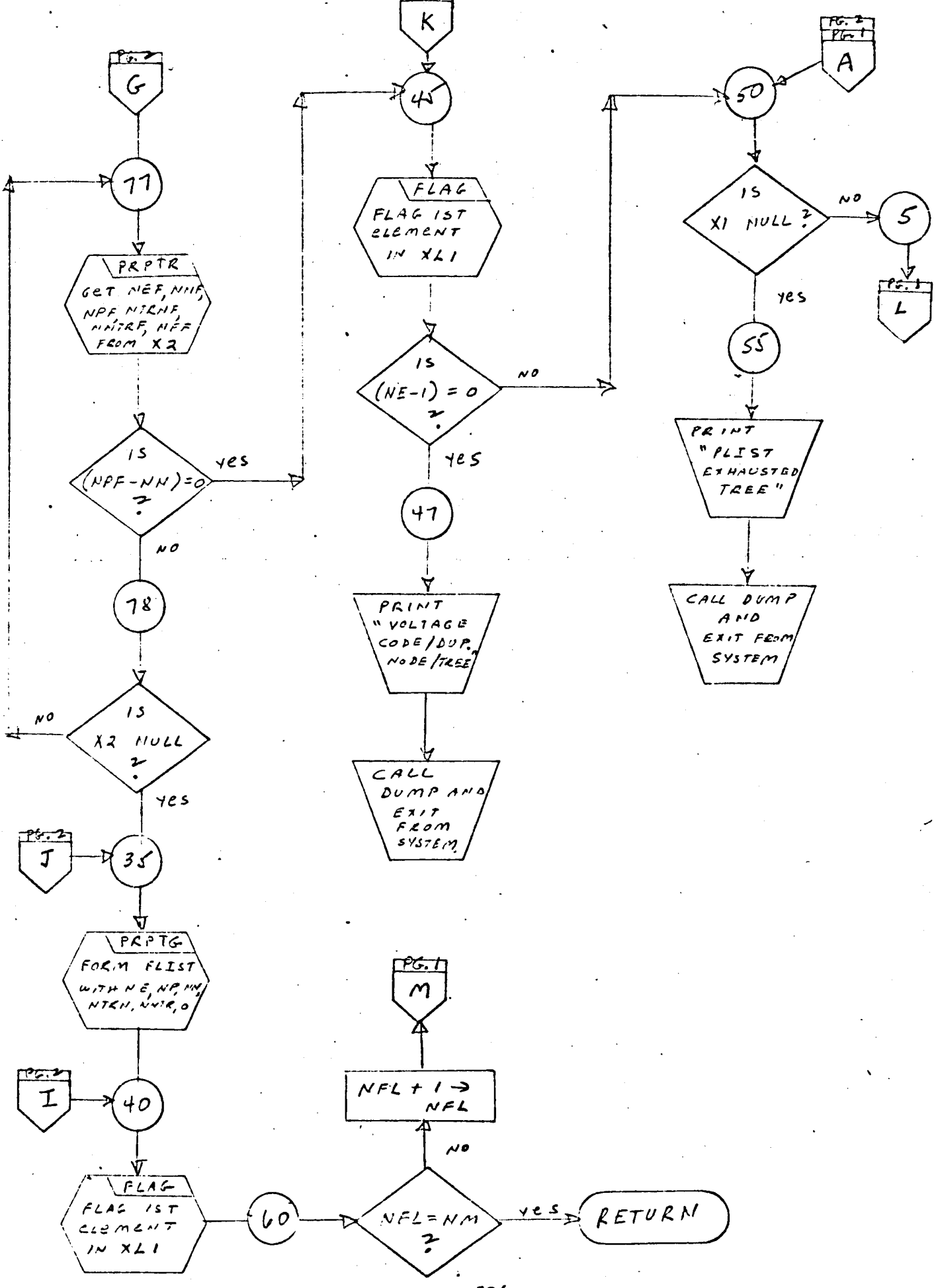
DUMP, ERASE, FLAG, PRPTG, PRPTR.

8. Using Subroutines

Preprocessor Pass 1 Main Routine only.


TREE
CALL TREE(PLIST, FLIST, NM)
ERASE
ERASE FLIST
1 → NFL
Pg. 3 M
(PLIST) → (X1)
Pg. 3 L
5
(X1) → (XL1)
PRPTR
GET NE, NN, NP, NTRN, NNTR, NF FROM X1
IS (NF) = 0 ?
NO
50
Pg. 3 A
YES
12
Pg. 2 B
IS (NN) = 0 ?
YES
70
NO
14
1 → (NENT)
16
IS (NFL-1) = 0 ?
YES
NO
15
IS (NP) = 0 ?
YES
75
Pg. 2 C
(FLIST) → (X2)
Pg. 2 F
18
PRPTR
GET NEF, NNF, NPF, NTRNF, NNTRF, NFF FROM X2
IS (NNF-NN) = 0 ?
YES
17
NO
IS (NPF-NP) = 0 ?
YES
45
Pg. 2 D
NO
19
IS (NNF-NP) = 0 ?
YES
NO
20
IS (NPF-NN) = 0 ?
YES
NO
21
Pg. 2 E


E
21
IS (NENT) = 1 ?
NO
yes
22
IS (NPF - NN) = 0 ?
NO
yes
23
2 → (NENT)
24
IS (NPF - NP) = 0 ?
NO
yes
25
3 → (NENT)
26
IS X2 NULL ?
NO
F
18
A
50
=1
TEST (NENT)
=2
=3
B
70
71
IS (NFL-1) = 0 ?
yes
NO
(FLIST) → (X2)
72
PRPTR
GET NEF, NVF, NPF, NTRNF, NNTRF, NFF FROM X2
IS (NPF - NP) = 0 ?
yes
NO
D
45
K
73
IS X2 NULL ?
NO
yes
30
PRPTG
FORM FLIST WITH NE, NV, NP, NTRN, NNTR, 0
40
I
C
75
IS (NFL-1) = 0 ?
yes
NO
35
J
76
(FLIST) → (X2)
77
G
28
yes


K
4
FLAG
FLAG IST ELEMENT IN XL1
IS (NE-1) = 0 ?
NO
yes
47
PRINT "VOLTAGE CODE/DUP. NODE/TREE"
CALL DUMP AND EXIT FROM SYSTEM
PG. 2 PG. 1
A
50
IS X1 NULL ?
NO
5
PG. 1
L
yes
55
PRINT "PLIST EXHAUSTED TREE"
CALL DUMP AND EXIT FROM SYSTEM
PG. 1
G
77
PRPTR
GET NEF, NNF, NPF, NTRNF, NNTRF, NFF FROM X2
IS (NPF-NN)=0 ?
yes
NO
78
IS X2 NULL ?
NO
yes
PG. 2
J
35
PRPTG
FORM FLIST WITH NE, NP, NN, NTRN, NNTR, 0
PG. 2
I
40
FLAG
FLAG IST ELEMENT IN XL1
60
NFL = NM ?
yes
RETURN
NO
NFL + 1 → NFL
PG. 1
M

Program Description

1. Identification

   a. Routine Label

      UPDWN

2. Function

This subroutine pops up a list, one element at a time, and then pushes each element down into another list (similar to DOWNS except that UPDWN destroys the list that was popped up).

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL UPDWN (T, S)

   b. Entry Conditions

      T = Head of a list
      S = Head of a list

   c. Exit Conditions

List T is popped up and pushed down into List S. List T is destroyed.

   d. Error Exits

      None.

5. Definition of Identifiers

ITEM = A temporary cell whose decrement contains the item of the element popped up from list T

NF = A temporary cell whose decrement contains the flag of the element popped up from list T

6. Method

a. If list T is not null upon entry, the elements in list T are popped up, one at a time. The item and flag of each element are saved in ITEM and NF respectively. The element popped up from T is restored to AVS. A new element containing ITEM AND NF is then pushed down into list S. This operation continues until the end of list T is encountered.

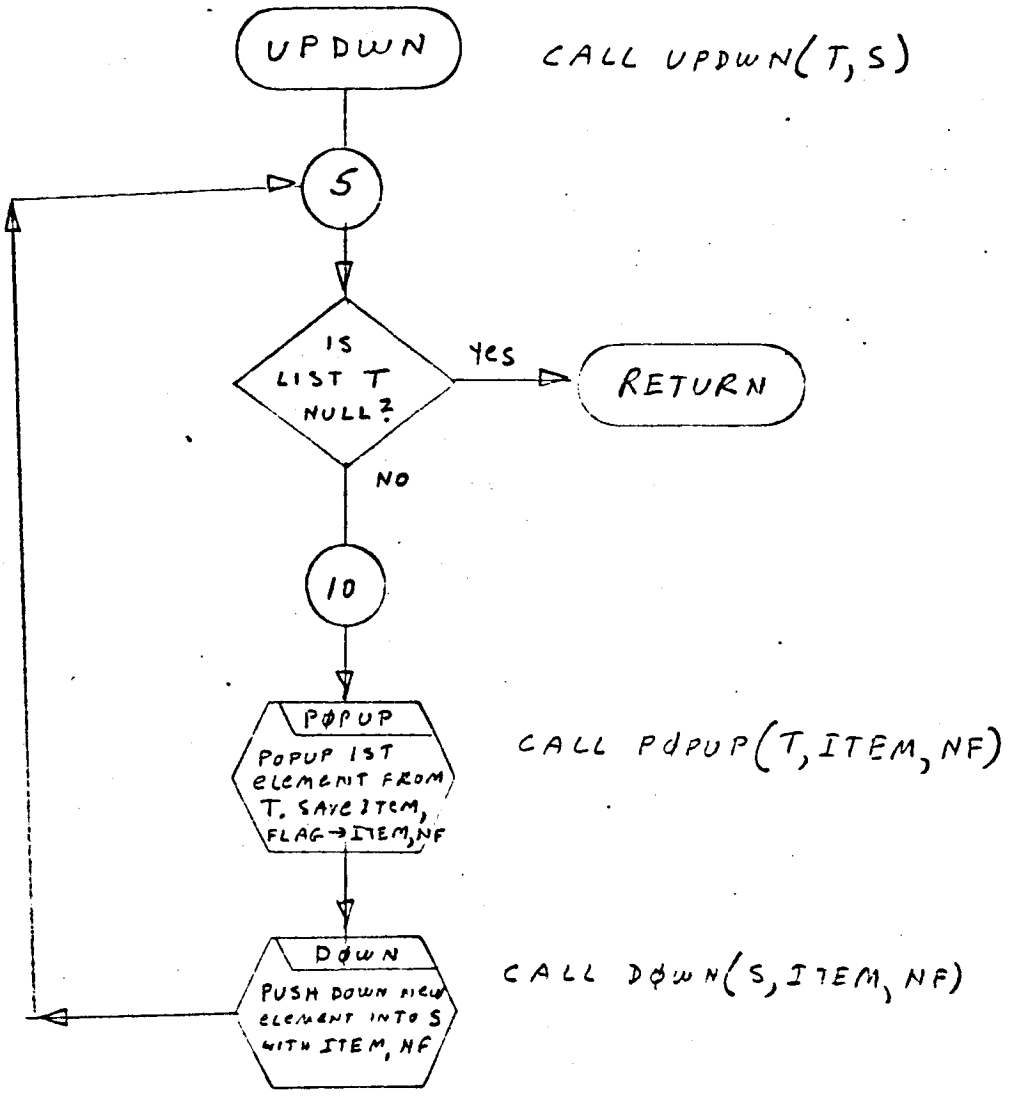
b. If list T is null upon entry, exit is made from the routine.

7. Other Subroutines Used

DOWN, POPUP.

8. Using Subroutines

BLNOUT, COMBN, DIMEN, DPDST, HOLBK, INPUTX, NLINDM, PARTS, STRIK, SUBST, SYMTP, WRTEQ, Main Programs for Pass 1 and Pass 2 of TAG Preprocessor.


UPDWN
CALL UPDWN(T,S)
5
IS LIST T NULL?
YES
RETURN
NO
10
POPUP
POPUP 1ST ELEMENT FROM T. SAVE ITEM, FLAG→ITEM,NF
CALL PDPUP(T,ITEM,NF)
DOWN
PUSH DOWN NEW ELEMENT INTO S WITH ITEM, NF
CALL DOWN(S,ITEM,NF)

Program Description

1. Identification

   a. Routine Label

      WRTEQ

   b. Name

      Write equations.

2. Function

Expresses a symbolic equation involving matrix manipulations into a series of subroutine calls, and outputs these statements as card images.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL WRTEQ(HLST, SLST, NPT, NEQ, NNEQ)

   b. Entry Conditions

HLST	Type A list containing the first part of the symbolic equation
SLST	Type A list containing the second part of the symbolic equation.
NPT	The NPT array of Main number 1
NEQ	Statement number to be output
NNEQ	Index indicating the submatrix to be involved in the matrix manipulations

   c. Exit Conditions

A series of statements will be output which effect the computation of the symbolic equation. Each statement will be one of these three:

CALL PSUM (FTEM2, FTEM1, FTEM1, n, 1)
CALL MSUM (tem, FTEM1, FTEM1, n, 1)
CALL MULT (tem, tem1, temi, n, 1, n1)

n = NPT (NNEQ) = Row dimension

tem, tem1 represent matrix names extracted from the symbolic equation

temi is either FTEM1 or FTEM2

n1 = Column dimension

The first statement to be output will carry the statement number (NEQ).

d. Error Exits

None.

5. Definition of Identifiers

ICHC	BCD "C"
ICMA	BCD COMMA
IBRKR	BCD right parenthesis
PLC	List used to form the initial equation from HLST and SLST
TEM	Temporary list used to hold one name
TEM1	Temporary list used to hold a series of names (push-down)
KYN, N1, N2	Parameters used to evaluate a single name
DUMY	Dummy-list
IPO	Holds a code for "previous op"
NEQ1	Holds running equation number
IT	Number of temporary matrix being used
PLCC	Used to hold a copy of PLC for output as a comment card
IDMY, IDUMY	Dummy variables

TEMT	Temporary list
ND1, ND2	Parameters used to evaluate a single name
IO	Holds a code for "this op"
IOL	Push-down list used to save IO
RST	Temporary list used to construct constituent parts of final output
CAL	List containing the card image of the statement to be output

6. Method

The initial format of HLST is something like this:

FV31 = FGI33 * (FI31

The format of SLST is like this:

-FG31 * FV11$ - FG32 * FV21$ --- etc.

a. HLST is pushed (reversed) into PLC.

b. SLST is split into the segments separated by $, and each segment is sent to EXCPT for testing (EXCPT will test the left symbol of each pair). If it is a legal symbol, then that segment is pushed into PLC. If not it is erased. When done, a right parenthesis is pushed into PLC to complete the equation.

c. The symbolic equation in PLC is now examined, starting from the right, and the infix notation of matrix multiplications and subtractions is transformed into a series of CALL statements. The process is a restricted infix-to-polish transformation which is generally as follows:

(1) Initialize:
O → IP O
O → IT
Output PLC as a comment card.
Remove the right parenthesis (top element) from PLC.

(2) Scan PLC for a left parenthesis, equal sign, minus sign, or asterisk, preceded by a name. The name is placed in TEMT, and the hierarchy order of the punctuation character is placed in IO. This order is

No character found	0
(	1
=	2
-	3
*	4

(3) EXCPT is called to determine n1 (the column dimension) fo the symbol in TEMT. TEMT is placed in TEM.

(4) If IO>IPO, go to (5) else go to (6).

(5) Push IO into IOL. If TEM is not null, push TEM into TEM1. Set IO = IPO. Go to (2).

(6) If IPO was an asterisk, go to (9); if it was a minus sign, go to (7). Else DUMP.

(7) If IT = 2, set up this equation:

CALL PSUM (FTEM2

If IT ≠ 2, set up this equation:

CALL MSUM (C(TEM)

and erase TEM.

(8) Append to the previous equation:

, FTEM1, FTEM1, n, 1)

Output this as a card image. Set IT = 1 and go to (12).

(9 Set up this equation:

CALL MULT (C(TEM),

and erase TEM. If TEM1 is not null, pop up the top symbol and append it to the equation. If TEM1 is null, append "FTEM1" and set n1 = n. If IO is an equal sign, append the rest of PLC to the equation and go to (11), else go to (10).

(10) Set IT = IT+1, and construct a symbol "FTEMi" where i = IT. Then append this symbol to the equation.

(11) Append to the equation:

, n, 1, nl)

and output it as a card image. If IO was an equal sign, exit from WRTEQ, else continue to (12).

(12) Pop up one element of IOL. If IOL is null, then go to (5), else set IPO = the top item of IOL and go to (4).

d. WRTEQ assumes that the equation must end with a right parenthesis, and that the right-most symbol-pair must be a multiplication; i.e., that the right end of the equation must look like this

A*B)

Except for this, any combination of single names or multiplications is permissible, e.g.,

A = B- C- D* (E- F*G*H*J)

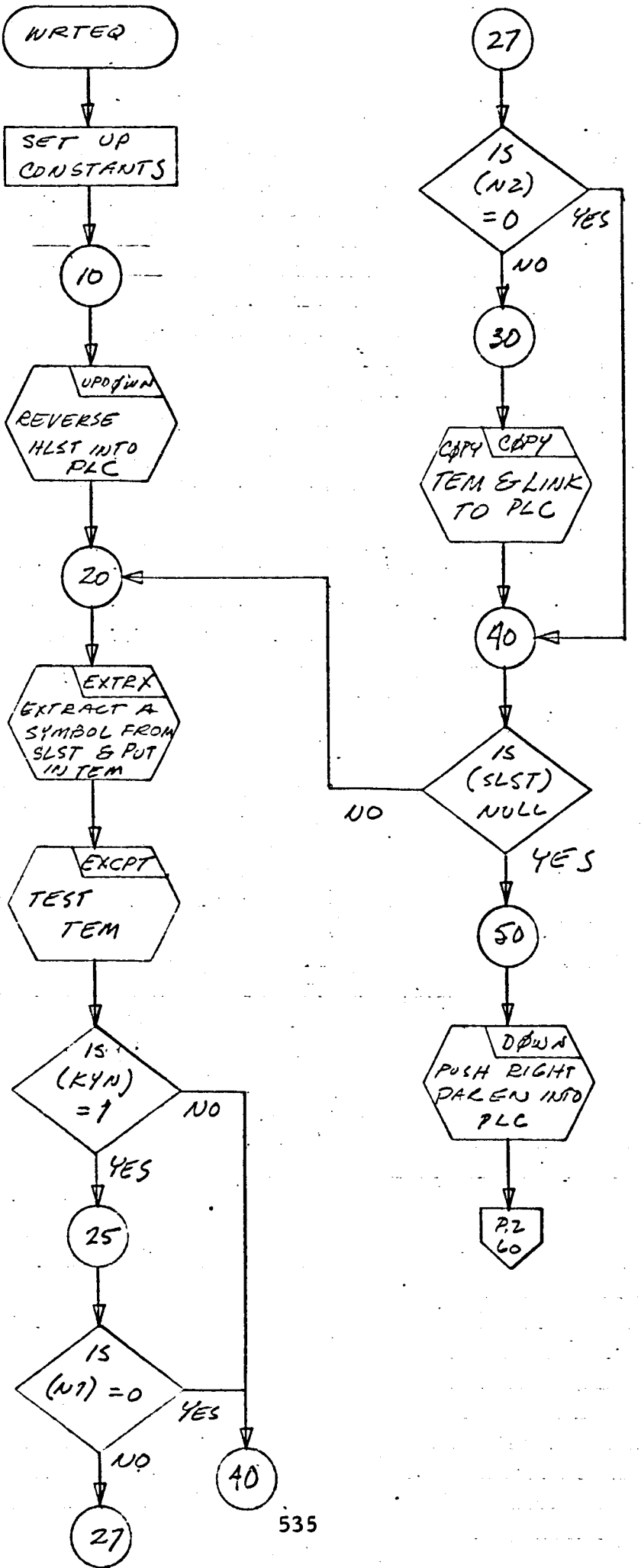
There can be only one nested (enclosed in parentheses) expression.

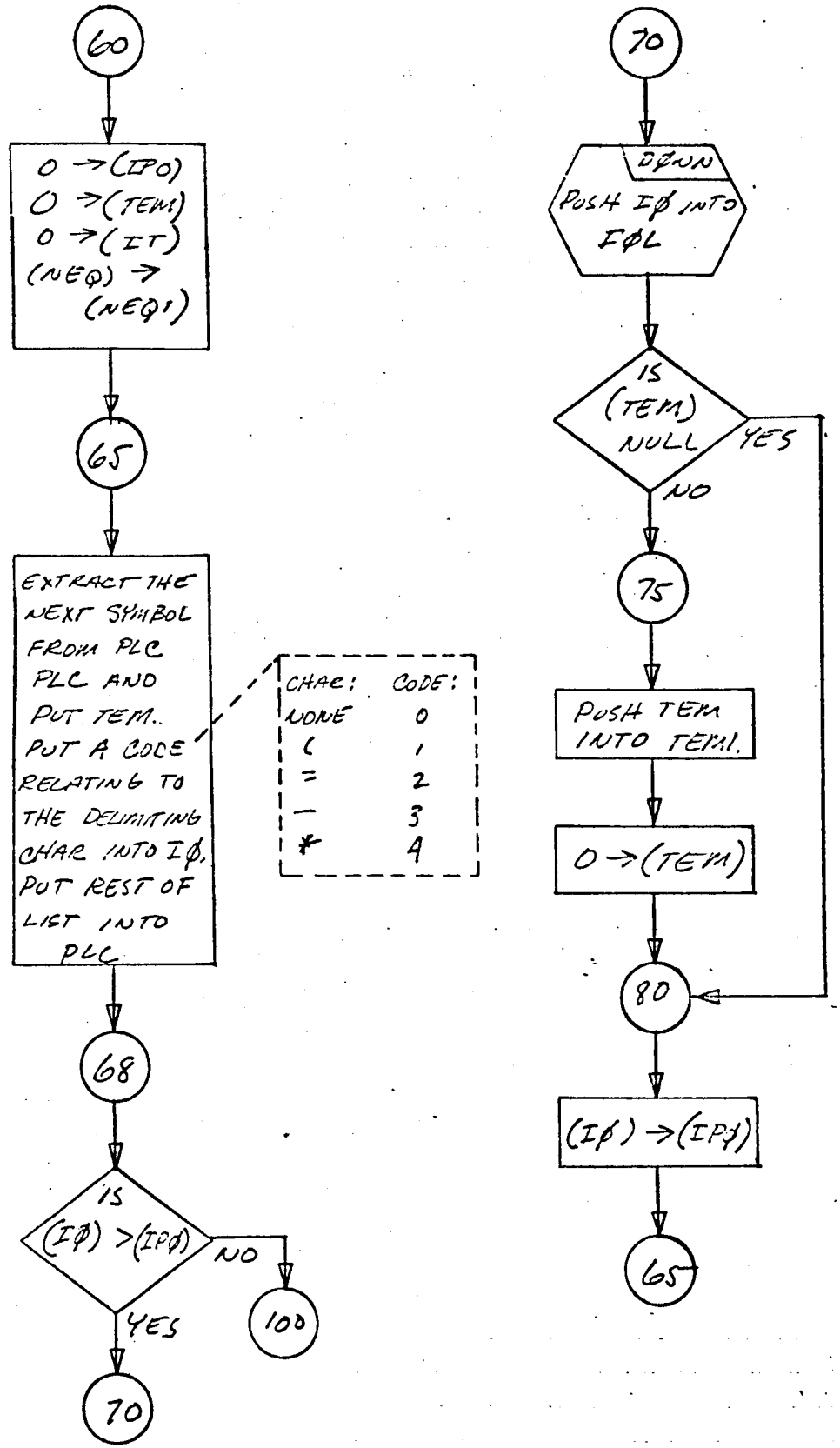
7. Other Subroutines Used

BCL, COPY, DBPCH, CBPCHC, DOWN, DUMP, ERASE, EXCPT, EXTRX, FROM, POPUP, SEGMNT, SYMBL, SYMCH, UPDWN.

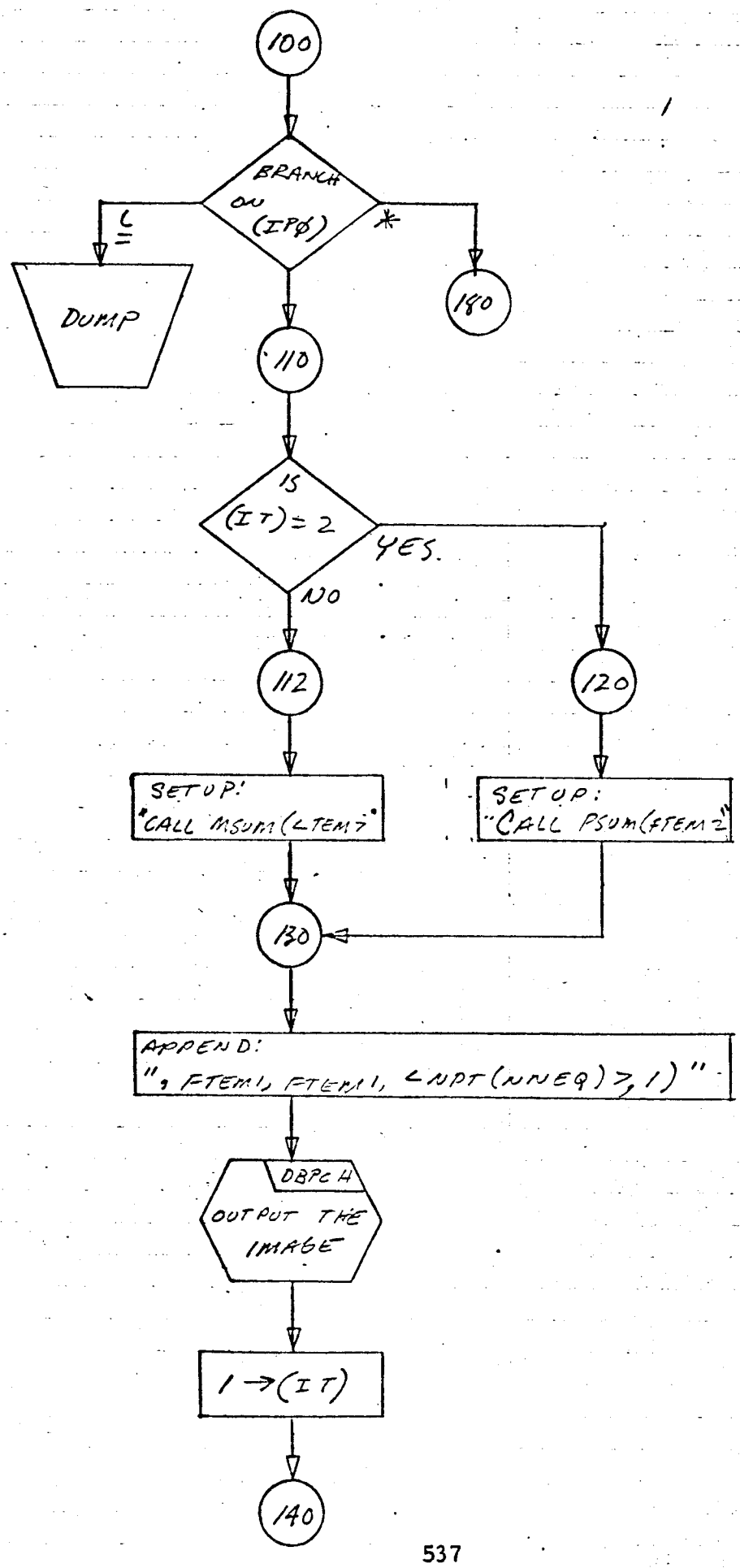
8. Using Subroutines

EQFS41, EQFV21, EQFV31.

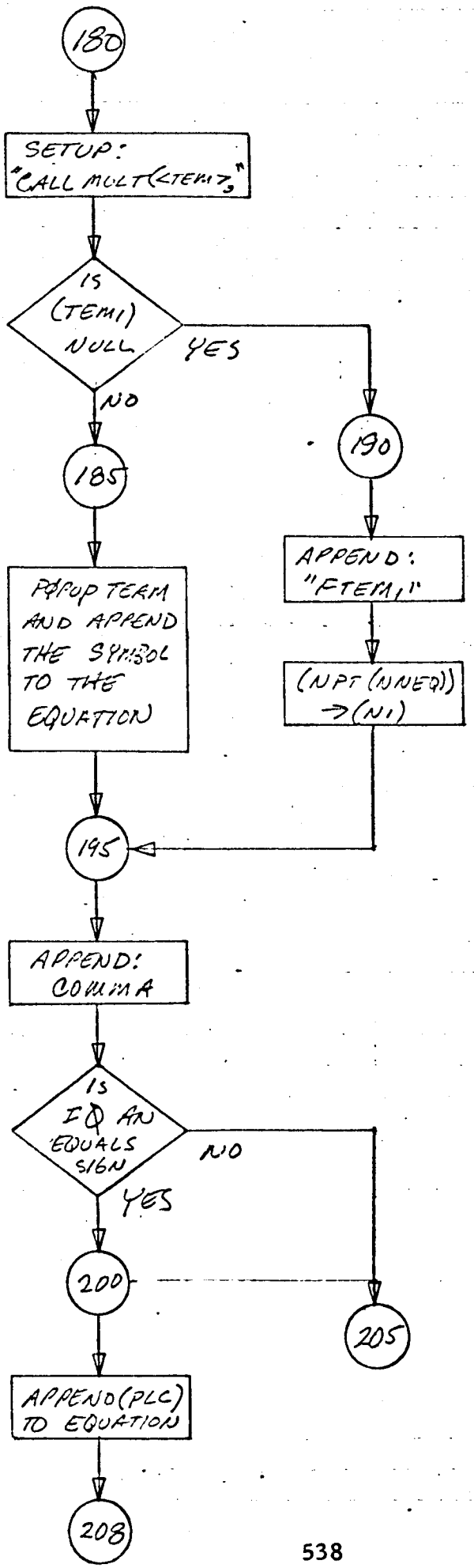

WRTEQ
SET UP CONSTANTS
10
UPOPTION
REVERSE HLST INTO PLC
20
EXTRX
EXTRACT A SYMBOL FROM SLST & PUT IN TEM
EXCPT
TEST TEM
IS (KYN) =1
NO
YES
25
IS (N1) =0
YES
NO
27
40
27
IS (N2) =0
YES
NO
30
COPY
COPY
TEM & LINK TO PLC
40
IS (SLST) NULL
NO
YES
50
DOWN
PUSH RIGHT PAREN INTO PLC
P.2 60

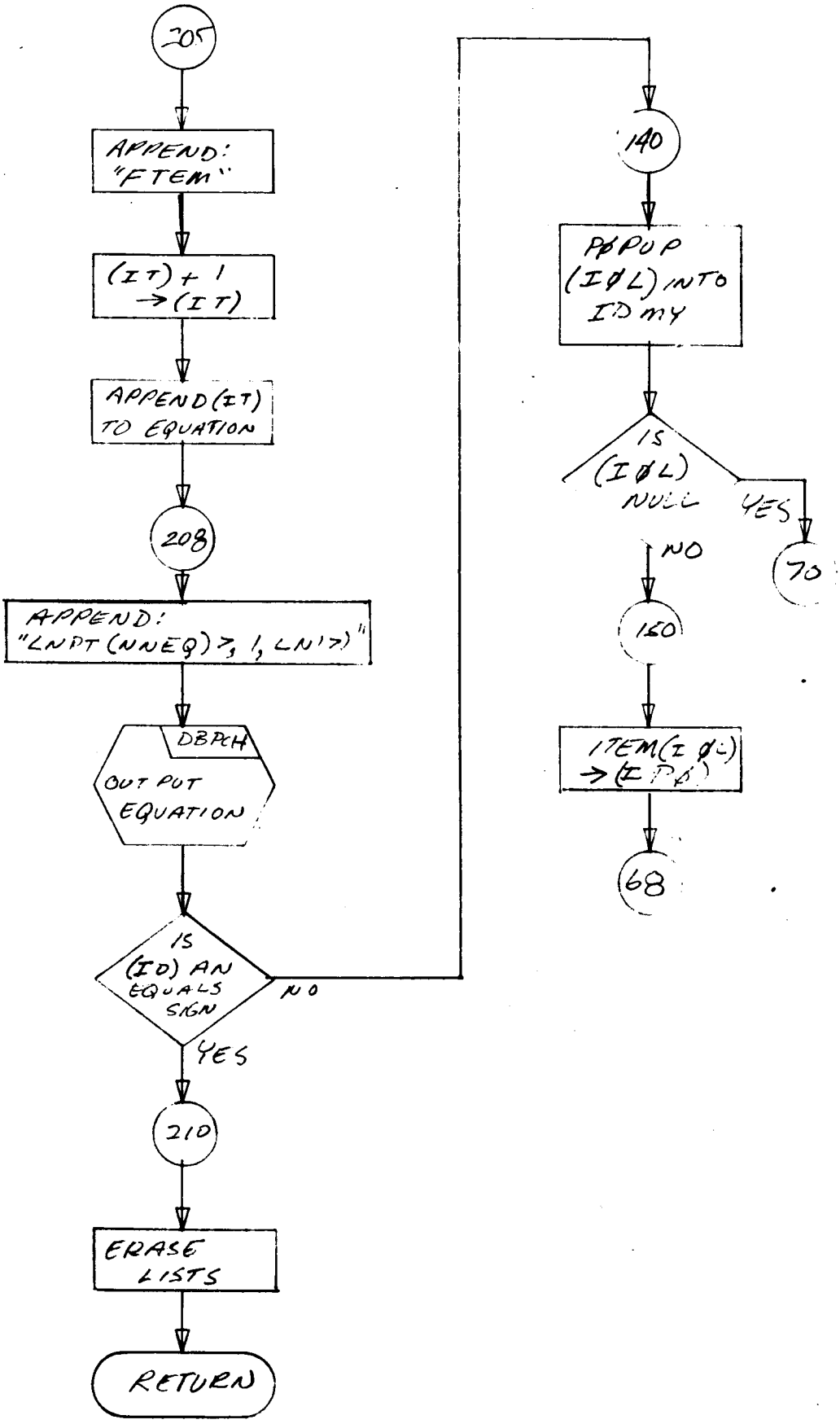

60
0 → (IPO)
0 → (TEM)
0 → (IT)
(NEQ) → (NEQ1)
65
EXTRACT THE NEXT SYMBOL FROM PLC PLC AND PUT TEM.. PUT A CODE RELATING TO THE DELIMITING CHAR INTO IØ. PUT REST OF LIST INTO PLC
CHAR: CODE:
NONE 0
(1
= 2
— 3
* 4
68
IS (IØ) > (IPØ)
NO
YES
100
70
DØWN
PUSH IØ INTO IØL
IS (TEM) NULL
YES
NO
75
PUSH TEM INTO TEM1.
0 → (TEM)
80
(IØ) → (IPØ)
65

WRTEQ 2


100
BRANCH ON (IPØ)
=
*
DUMP
180
110
IS (IT) = 2
YES.
NO
112
120
SETUP: "CALL MSUM(<ITEM>"
SETUP: "CALL PSUM(ITEM2"
130
APPEND: ", ITEM1, ITEM1, <NDT(NNEQ)>, 1)"
DBPCH
OUTPUT THE IMAGE
1 → (IT)
140

WRTEQ 3


180
SETUP:
"CALL MULT(<ITEM1>,"
IS (ITEM1) NULL
YES
NO
185
190
POP UP TEAM AND APPEND THE SYMBOL TO THE EQUATION
APPEND: "ITEM1,"
(NPT (NNEQ)) →(N1)
195
APPEND: COMMA
IS IQ AN EQUALS SIGN
NO
YES
200
205
APPEND(PLC) TO EQUATION
208


205
APPEND: "FTEM"
(IT) + 1 → (IT)
APPEND (IT) TO EQUATION
208
APPEND: "LNPT (NNEQ) ⁊, 1, LN1⁊)"
DBPCH
OUT PUT EQUATION
IS (ID) AN EQUALS SIGN
NO
YES
210
ERASE LISTS
RETURN
140
POP (IOL) INTO IDMY
IS (IOL) NULL
YES
NO
70
160
ITEM(IOL) → (IPO)
68

WRTEQ 5

Program Description

1. Identification

   a. Routine Label

      XFORM

   b. Name

      Form the ideal transformer voltage constraint matrix.

2. Function

This subroutine forms the node system ideal transformer constraint matrix TTR, as a two-dimensional array-list of NRR rows and NM columns such that [TTR] * VN] = 0.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL XFORM (WLIST, TTR, NRR)

   b. Entry Conditions

      WLIST = A type D list representative of the TAG connection list.

   c. Exit Conditions

      TTR contains a two-dimensional list representation of the transformer voltage constraint matrix.

      NRR equals the total number of transformer windings minus the number of different transformers.

   d. Error Exits

      None.

5. Definition of Identifiers

NM = The number of nodes less one

X1 = Local variable used to search down through WLIST

X1L = The first unflagged transformer descriptor

X2L = The first transformer descriptor whose transformer number NTRN is equal to that of X1L.

6. Method

The matrix representation, TTR, of the transformer node voltage constraint equations is generated from the WLIST such that TTR * VN = 0. For each transformer having N windings of T_i(i = 1, 2 - N) turns respectively, N - 1 equations are formed which relate the node pair voltage of the first winding to that of the N - 1 other windings. Thus, N - 1 equations of the form $T_i(V_{NP} - V_{NN})_{(\text{of } N1)} - T_1(V_{NP} - V_{NN})_{(\text{of } N_i)} = 0$ are generated for each transformer in WLIST and recorded as the four-dimensional array-list TTR. TTR has dimensions NRR x NM.

The following steps are implemented in XFORM to accomplish this end.

a. Search WLIST from X1 on for the first unflagged X former descriptor and, when found, label it X1L and flag it.

b. Search WLIST from X1L + 1 for the first transformer descriptor whose transformer number NTRN is equal to that of X1L. When found, label it X2L, flag it and add one count to NRR.

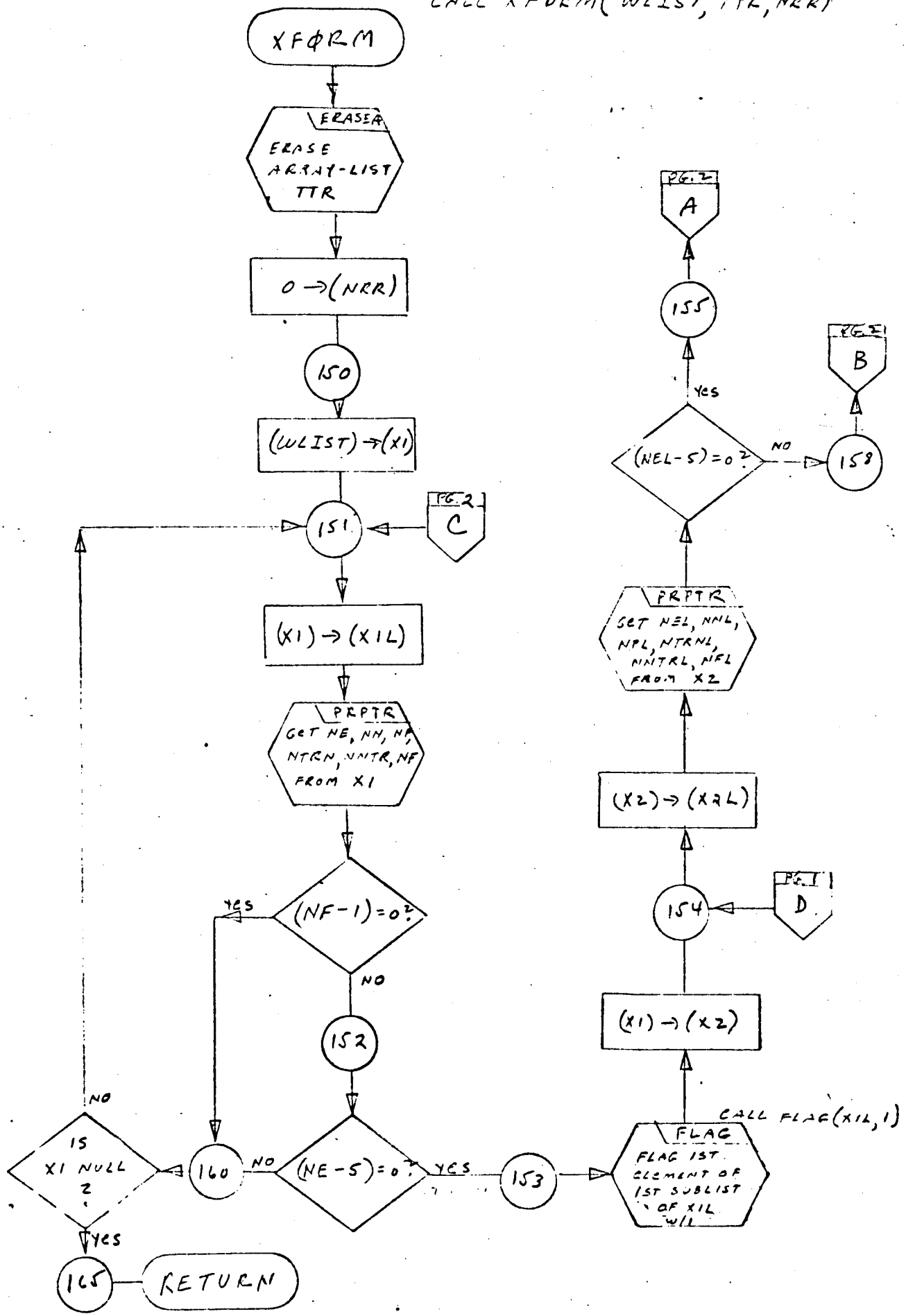
c. If the negative node number, NN(X1L), of winding X1L ≠ 0, the number of turns, NNTR(X2L), of winding X2L is subtracted from the value of the NRR, NN(X1L) position of the TTR matrix.

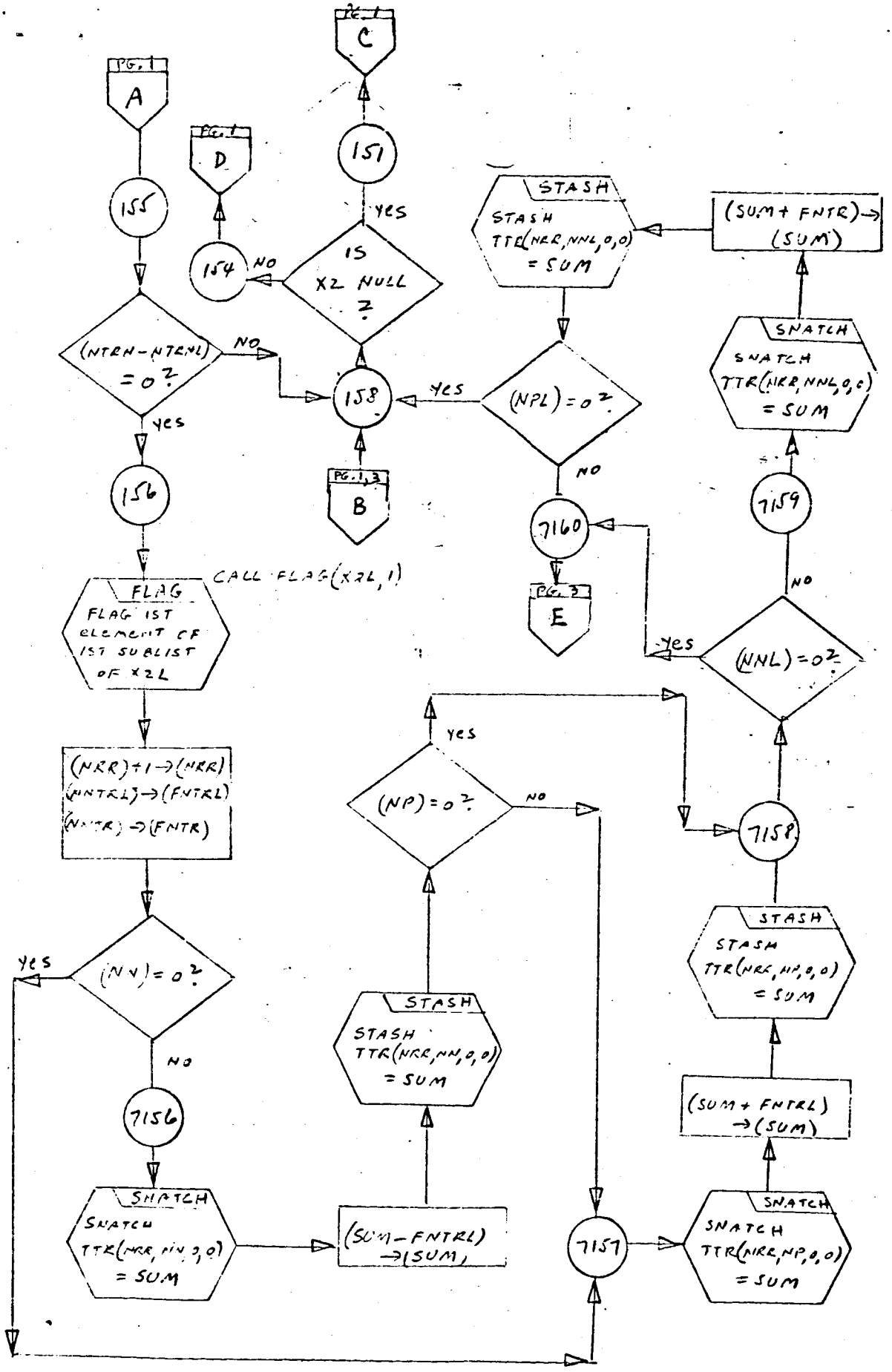
d. If NN(X1L) = 0 or NP(X1L) ≠ 0, NNTR(X2L) is added to the value of the NRR, NP(X1L) position of the TTR matrix.

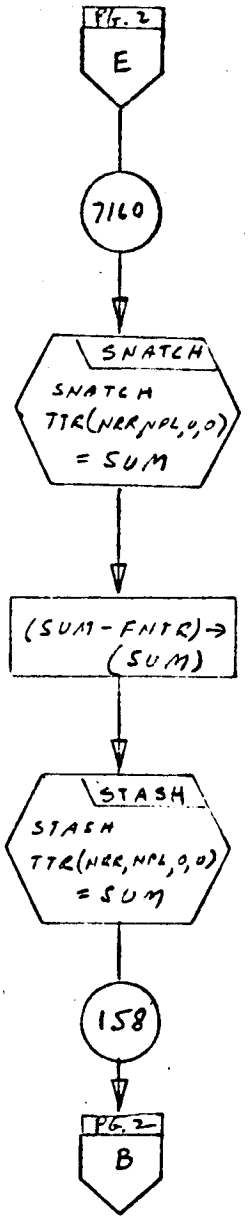
e. If NP(X1L) = 0 or NN(X2L) ≠ 0, NNTR(X1L) is added to the value of the NRR, NN(X2L) position of the TTR matrix.

f. If NN(X2L) = 0 or NP(X2L) ≠ 0, NNTR(X1L) is subtracted from the value of the NRR, NP(X2L) position of the TTR matrix.

CALL XFORM(WLIST, TTR, NRR)




PG. 1
C
151
PG. 1
A
155
PG. 1
D
154
NO
IS X2 NULL ?
yes
(NTRN-NTRNL) = 0 ?
NO
yes
156
158
PG. 1, 3
B
CALL FLAG(X2L,1)
FLAG
FLAG 1ST ELEMENT OF 1ST SUBLIST OF X2L
(NRR)+1→(NRR)
(NNTRL)→(FNTRL)
(NNTR)→(FNTR)
(NN) = 0 ?
yes
NO
7156
SNATCH
SNATCH TTR(NRR,NN,0,0) = SUM
(NP) = 0 ?
yes
NO
STASH
STASH TTR(NRR,NN,0,0) = SUM
(SUM-FNTRL) →(SUM)
7157
(NPL) = 0 ?
yes
NO
STASH
STASH TTR(NRR,NNL,0,0) = SUM
(SUM+FNTR)→(SUM)
7160
PG. 3
E
(NNL) = 0 ?
yes
NO
7158
STASH
STASH TTR(NRR,NP,0,0) = SUM
(SUM+FNTRL) →(SUM)
SNATCH
SNATCH TTR(NRR,NP,0,0) = SUM
7159
SNATCH
SNATCH TTR(NRR,NNL,0,0) = SUM


PG. 2
E
7160
SNATCH
SNATCH
TTR(NRR,NPL,0,0)
= SUM
(SUM - FNTR) →
(SUM)
STASH
STASH
TTR(NRR,NPL,0,0)
= SUM
158
PG. 2
B

Program Description

1. Identification

   a. Routine Label

      ZEROX

   b. Name

      Output ZEROX statements.

2. Function

   To output the CALL ZEROX statements.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL ZEROX (IS, NPT)

   b. Entry conditions

      IS contains statement number; NPT is the NPT matrix of Main number 2.

   c. Exit Conditions

      CALL ZEROX statements output.
      Output "FT = 0."

5. Definition of Identifiers

IZ	Temp. to hold IS
CALL	List (type A) contains "CALL ZEROX("
ICOMA	BCD comma
IBRK	BCD right parenthesis
TEM, PLC	Temp. lists
N1, N2	Used by EXCPT to hold matrix dimensions

SDEF contains (in list type A format):

FS41$FL44$FI41$FL41$FS11$FL42$FS21$FL43$
FS31$FV31$FG33$FI31$FG31$FV11$FG32$FV21$
FL31$FL32$FL33$FL34$FC11$FI11$FC12$FG11$
FG12$FG13$FL11$FL12$FL13$FL14$FVD21$FC22$
FI21$FG21$FG22$FG23$FL21$FL22$FL23$FL24$
FSD11$FSD21$FSD31$FC21$FVD11$FVO11$FV41$FSO41

6. Method

The symbols in SDEF are extracted one at a time and sent to EXCPT for examination. If legal, their dimensions are multiplied to compute a single-array dimension, and the statement

CALL ZEROX (<symbol>, <dimension>)

is output.

After all such statements have been output, the statement

FT = 0

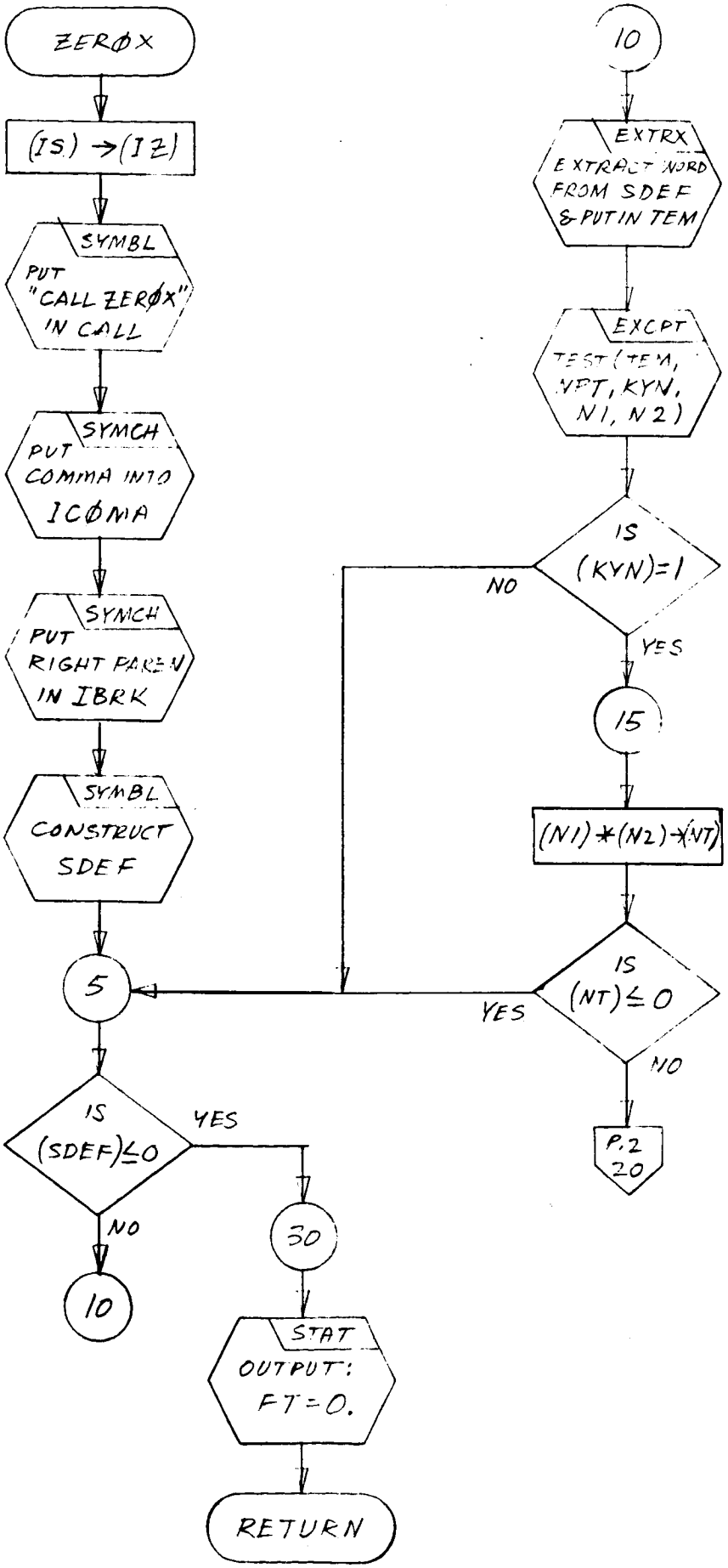
is output.

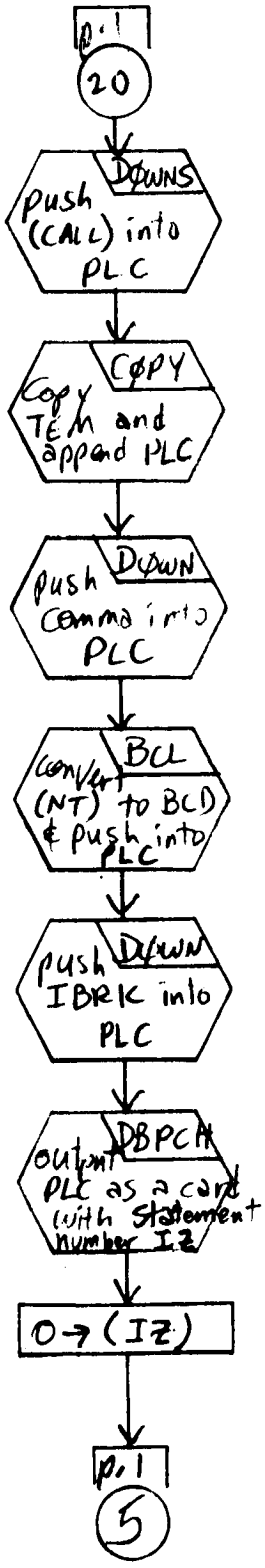
7. Other Subroutines Used

BCL, COPY, DBPCH, DOWN, DOWNS, EXCPT, EXTRX, STAT, SYMBL, SYMCH.

8. Using Subroutines

Main Program for Pass 2 of TAG Preprocessor.


ZERØX
(IS) → (IZ)
SYMBL
PUT "CALL ZERØX" IN CALL
SYMCH
PUT COMMA INTO ICØMA
SYMCH
PUT RIGHT PAREN IN IBRK
SYMBL
CONSTRUCT SDEF
5
IS (SDEF) ≤ 0
YES
NO
10
30
STAT
OUTPUT: FT = 0.
RETURN
10
EXTRX
EXTRACT WORD FROM SDEF & PUT IN TEM
EXCPT
TEST (TEM, NPT, KYN, N1, N2)
IS (KYN) = 1
NO
YES
15
(N1) * (N2) → (NT)
IS (NT) ≤ 0
YES
NO
P.2 20


p.1
20
DOWNS
Push (CALL) into PLC
COPY
Copy TEXT and append PLC
DOWN
Push comma into PLC
BCL
Convert (NT) to BCD & push into PLC
DOWN
Push IBRK into PLC
DBPCH
Output PLC as a card with statement number IZ
0→(IZ)
p.1
5

ZIRDX

C. Subroutine Writeups and Flow Charts for TAG Execution Program

Program Description

1. Identification

a. Routine Label

INV

b. Name

Matrix Inversion Subroutine

2. Function

The subroutine computes the inverse of a real N x N matrix.

3. Programming System

Coded in FAP for use in FORTRAN II

4. Usage

a. Calling Sequence

CALL INV (A, B, N)

b. Entry Conditions

A = A real N x N matrix to be inverted.

B = The N x N matrix into which the inverse of A is to be stored.

N = The dimensions of A and B.

c. Exit Conditions

If A has the properties described in "Method," the inverse of A is placed in B.

d. Error Exits

The routine has no error exits. However, whenever a division by zero has occurred during inversion, prior to exit, a SINGULAR MATRIX message is printed.

5. Definition of Identifiers

The following identifiers appear on the flow diagram:

A(i) = The ith location of matrix A. A is being treated as a column matrix

B(i) = The ith location of matrix B. B is being treated as a column matrix

N = The number of rows and columns in A and in B

[B] = The last location address of the matrix B

LF_k = The first address of the pivot row

L_k = The address of the element in the pivot row which is currently being operated upon

LF_i = The first address in the row which is currently being operated upon

L_i = The address of an element in the above row

n_c = An index used to count columns during row operations. n_c usually ranges between N-1 and 1.

CIl = The value of the first element in the current pivot row

CKl = The value of the first element in the row being operated upon

n_1 = An index used to count the N-1 other rows that a particular pivot row must operate upon

n_2 = An index used to count the required N pivot row selections

6. Method

The INV algorithm is essentially a FAP implementation of the INVRS subroutine method, but without maximum pivot element selection.

In INV, the diagonal elements of B are taken, in sequence, as pivot elements. When the space-saving column shifting algorithm is applied, the diagonal elements always appear in their correct rows, but in column one. The flow diagram shows the address manipulation required to carry out the procedure in FAP.

The algorithm may be stated in standard matrix element notation in the following way:

$$
\left.\begin{array}{l}
b_{k1} \rightarrow C_1 \\
\dfrac{b_{k(j+1)}}{C_1} \rightarrow b_{kj} \\
\text{for } 1 \le j \le N-1 \\
1/C_1 \rightarrow b_{kN} \\
b_{i1} \rightarrow C_2 \\
b_{i(j+1)} - C_2 b_{kj} \rightarrow b_{ij} \\
\text{for } 1 \le j \le N-1 \\
-C_2 b_{kN} \rightarrow b_{iN}
\end{array}\right\} \quad 1 \le i \le N,\ i \ne k \qquad \left.\begin{array}{l}\\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} \quad i \le k \le N
$$

where b_{ij} is a typical element of B, in the ith row and jth column. The kth row is the pivot row.

The matrix A must have the following properties:

a. A must be nonsingular.

b. The first element of the first row of A must be nonzero.

c. A zero may not develop in the pivot element position of the matrix during inversion.

Example:

The following matrices will be considered to be singular by INV:

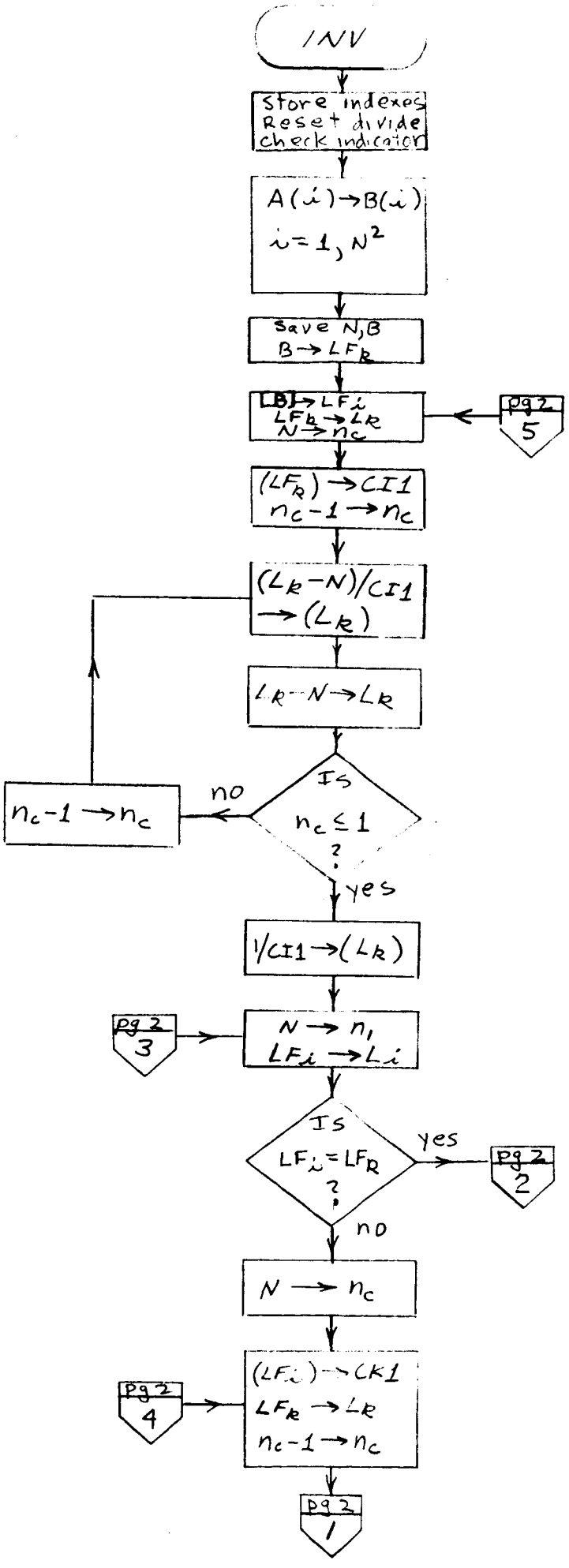
$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

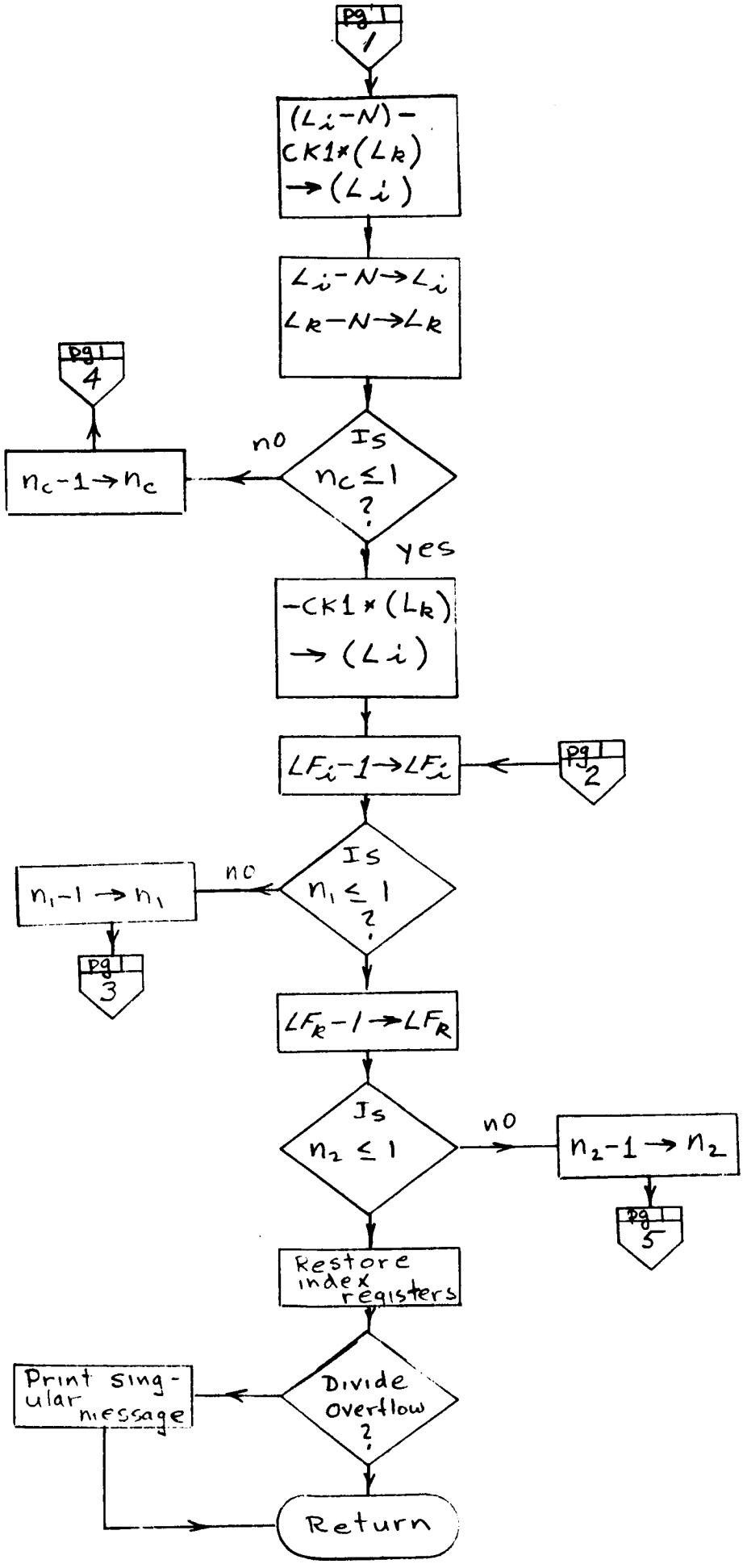
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

7. Subroutines Used

CDOUT


INV
Store indexes
Reset divide
check indicator
A(i)→B(i)
i=1, N²
Save N,B
B→LFR
[B]→LFi
LFk→LR
N→nc
pg 2
5
(LFR)→CI1
nc-1→nc
(Lk-N)/CI1
→(Lk)
Lk-N→Lk
Is
nc≤1
?
no
nc-1→nc
yes
1/CI1→(Lk)
pg 2
3
N→n1
LFi→Li
Is
LFi=LFR
?
yes
pg 2
2
no
N→nc
(LFi)→CK1
LFk→Lk
nc-1→nc
pg 2
4
pg 2
1


pg 1
(Li-N)-CK1*(Lk)→(Li)
Li-N→Li
Lk-N→Lk
pg 4
nc-1→nc
no
Is nc≤1 ?
yes
-CK1*(Lk)→(Li)
LFi-1→LFi
pg 2
Is n1≤1 ?
no
n1-1→n1
pg 3
LFk-1→LFk
Is n2≤1 ?
no
n2-1→n2
pg 5
Restore index registers
Divide Overflow ?
Print singular message
Return

Program Description

1. Identification

   a. Routine Label

      INVRS

   b. Name

      Real Matrix Inversion Subroutine

2. Function

The subroutine transforms a matrix to its inverse. Singularity is treated in several different ways, as described below.

3. Programming Language

FORTRAN II

4. Usage

   a. Calling Sequence

      CALL INVRS (P, N, KSIG)

   b. Entry Conditions

      P = An N x N real matrix to be inverted
      N = The dimension of P
      KSIG = The matrix singularity flag (see "Method")

   c. Exit Conditions

      P is transformed to the inverse of P or some other matrix as described under "Method."
      N is unchanged.
      KSIG is set to either zero or one as described under "Method."

4. Error Exits

No error exit is ever made. However, under certain conditions, KSIG is set, and a message "SINGULAR MATRIX" is printed.

5. Definition of Identifiers

P = The N x N real matrix to be inverted

N = The dimension of P

KSIG = The singularity flag

DET = The determinate of the matrix

J = An index used for counting at several places in the routine. When used in P, it is usually the column number.

I = An index used for counting. In P, it is usually the row number.

J1 = A counter which is set to one plus the number of transformations that have been completed. It ranges from 1 to N.

IMAX = The row number of the maximum element in the first column of P, prior to an inversion step

PIVOT = The value of the maximum element in column one prior to an inversion step

PMAX = The absolute value of PIVOT

L = A vector representing the row permutation matrix. L(J1) contains the number of pivot row chosen for the J1th inversion step. If no pivot was chosen for the J1th step, L(J1) will be zero. The dimension of L is assumed to be N but can be no greater than 100.

L1 = A vector representing the column permutation matrix. The location L(i) contains the step number (J1) at which the ith row was pivot row. If L(i) is zero, the ith row has not been used. The dimension of L1 is assumed to be N but can be no greater than 100.

L2 = A vector used to record row and column exchanges that are required after the transformation has been completed. As with L and L1, its dimension is assumed to be N, but is actually 100.

L3 = A vector whose L(J) entry is zero if every element in the jth column of the original P matrix was less than 10^{-30}.

If some element was greater than 10^{-30}, then L(J) contains the row number of the last element, counting from row one, that was greater than 10^{-30}.

PM = The first element of row currently being operated upon during an inversion step

PPM = The product of PM/PIVOT times the element in the pivot row that is in the column currently being operated upon

DIFA = A subroutine that inspects additions and subtractions during inversion to determine whether or not the result is largely roundoff.

6. Method

The basic inversion algorithm is standard and is discussed in a number of sources.[1] The basic algorithm will be briefly described. A more thorough presentation of the subroutine's deviations from standard procedures will be made.

a. Basic Algorithm

Matrix inversion in its basic form can be explained in the following way: Consider the real N x N matrix P, augmented by the identity matrix.

$$\left[W\right] = \left[P \vdots I\right] = \left[w_{ij}\right]$$

A sequence of N matrices B_i is found which transforms P into the identity matrix and I into some other matrix.

$$B_1 \times B_2 \times \cdots B_N \left[P \vdots I\right] = \left[I \vdots B_1 \times B_2 \cdots B_N\right]$$

[1] Ralston and Wilf, Mathematical Methods for Digital Computers. New York: John Wiley and Sons, 1960.

However, the definition of P^{-1} is

$$P^{-1} P = I$$

and the process, if it exists, replaces the original identify matrix with P^{-1}. The process of transforming one column of P into a unit vector is the well known[1] pivot transformation formula.

$$w_{ij} \longleftarrow w_{ij} - w_{kj} \frac{w_{ik}}{w_{kk}} \qquad \text{for all i except i = k and for all j}$$

$$1 \leq i \leq N$$

$$1 \leq j \leq 2N$$

$$w_{kj} \leftarrow \frac{w_{kj}}{w_{kk}} \qquad \text{for all j}$$

where the kth column is being transformed into the kth unit vector. This is equivalent to the kth basis change matrix.

$$B_k = \begin{bmatrix} & & -(w_{ik}/w_{kk}) & \\ 1 & & & \\ & & -w_{2k}/w_{kk} & \\ & 1 & & 0 \\ & \cdot & \cdot & \\ 0 & & & \\ & & (1/w_{kk}) & \\ & & & 1 \\ & & -w_{nk}/w_{kk} & \end{bmatrix}$$

[1] Ibid.

In more complicated algorithms, an effort is made to minimize the roundoff error by choosing w_{kk} not in sequence, but in a way that maximizes its magnitude. The result of this "positioning for size" is that P is not transformed into the identity matrix, but into an identity matrix with its rows permuted. Let this matrix be Q. N transformations D_i have then been defined such that

$$D_1 \times D_2 \cdots D_N \left[P \vdots I \right] = \left[Q, \; D_1 \times D_2 \cdots D_N \right]$$

A property of any permutation matrix is

$$Q^{-1} = Q^T$$

where Q^{-1} is also a permutation matrix. We now have

$$(D_1 \times D_2 \cdots D_N) \, P = Q$$

and

$$Q^T (D_1 \times D_2 \cdots D_N) \, P = Q^T Q = Q^{-1} Q = I$$

By definition

$$P^{-1} = Q^T (D_1 \times D_2 \cdots D_N)$$

Thus, when the pivot rows are not chosen in sequence, the inverse may be retrieved by rearranging the rows of the resulting matrix.

One additional complication is introduced to save computer storage space. It is that P is not augmented by I, but instead by Q. The process then becomes:

$$D_1 D_2 \cdots D_N \left[P \; Q \right] = \left[Q \; D_1 D_2 \cdots D_N Q \right]$$

$$\left[D_1 \times D_2 \cdots D_N \times Q^T \right] Q \;\; P = Q$$

$$Q^T \left[D_1 \times D_2 \cdots D_N \times Q \right] Q \;\; P = I$$

$$P^{-1} = Q^T \left[D_1 \times D_2 \cdots D_N \times Q \right] Q^T$$

The rows and the columns of the result must now be rearranged to get the inverse.

Example 1:

$$P = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$Q^T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = Q = Q^{-1}$$

$$P \;\; Q^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$D_1 D_2 = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$D_1 D_2 \left[P \;\; Q^T \right] = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$P^{-1} = Q^T \left| D_1 D_2 Q^T \right| Q = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

$$P\,P^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

b. Modified Algorithm

The INVRS routine requires the permutation of both rows and columns of the resultant matrix and is thus similar to the example above. To save space, however, Q is not initially stored in the computer. In a like manner, Q is not retained in core as it is formed. The computation proceeds in roughly the following manner:

(1) L, L1, and L3 are set to zero. L2 is set to 1, 2, 3, $\cdots$ N, J1 is set to 1.

(2) A search of column one is performed to find the largest magnitude element which

- is not in a previous pivot row and
- is not in a row that has the same number as a column, in the original P matrix, having no element greater in magnitude than 10^{-30}.

(3) IMAX is set to the pivot row number and PIVOT is set to the pivot element value. L1(MAX) is set to J1 to indicate that row IMAX has been used on the J1st step. L(J1) is set to IMAX to indicate that the J1st column of Q is the IMAXth unit vector.

(4) Row IMAX is divided by PIVOT and shifted one column to the left. The Nth element of row IMAX is set to 1/PIVOT.

(5) Remembering that row IMAX is shifted, the subtraction phase of the elimination algorithm is performed on each of the N - 1 rows. Starting from the left in each row, the new element value is computed and shifted left one column. The Nth element of each non-pivot row is replaced by -1/PIVOT times the value of the first element of the row, prior to the row-shifting operation.

(6) When the elimination process has been performed for each row, J1 is increased. If J1 is greater than N, we go to (8), otherwise back to step (2).

(7) If, after the search for a pivot element, it is found that no eligible element is greater than 10^{-30}, one of two courses is taken.

  (a) If any element of the J1st column was greater than 10^{-30} in the original P matrix, KSIG is set to zero, "SINGULAR MATRIX" is printed, and the routine is exited.

  (b) If all elements of the J1st column of the original P were less than 10^{-30}, then all columns of the current P matrix are shifted one column to the left, and step (6) is carried out.

(8) When N steps have been performed, L contains a representation of the Q (permutation) matrix. It is used to perform an inverse row permutation upon the matrix that has replaced P. During permutation, if a zero

entry is found in L, the corresponding row is cleared to zero. Such rows have the same numbers as columns in the original P that were less than 10^{-30}.

(9) Using the L2 vector representation of the Q (permutation) matrix, the columns are permuted. Any column corresponding to a column of the original P which had no element greater in magnitude than 10^{-30} is set at zero. A -1 is then inserted in the diagonal element of such columns.

(10) The subroutine is exited with KSIG = 1.

Example 2:

$$P = \begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

From left to right and down, the sequence of operations is approximately:

$$\begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 1/2 \\ 0 & 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 1/2 \\ 0 & 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 1/2 \\ 2 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 1/2 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix}$$

At this point, the elimination is completed. Pivot rows were 2, 3, and 1, in that order. Therefore, L and L1 are

$$L = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \qquad L1 = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$$

L2 is always

$$L2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The permutation proceeds as follows:

$$\begin{bmatrix} 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 0 & 1/2 \\ 0 & 1/2 & 0 \end{bmatrix} L2 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \qquad L2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The row permutation is complete. Reset L2 to:

$$L2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 1/2 & 0 \\ 1/2 & 0 & 0 \end{bmatrix} L2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix} \qquad L2 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

The column permutation is complete.

$$\begin{bmatrix} 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3:

If P has zero columns:

$$P = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Briefly, the results are

$$\begin{bmatrix} 1/2 & 1/2 & 1/2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad L1 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} \qquad L2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The results of permutation and column and row zeroing are

$$\begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \qquad L2 = \begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix}$$

$$L2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ 1/2 & 0 & 0 \end{bmatrix} \qquad L2 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

c. Comments on Modified Algorithm

The column shifting and the use of L, L1, and L2 for permutation are effective space-saving devices.

When P initially has zero columns, clearly no inverse exists. In this one instance of singularity, the routine essentially strikes out

all rows that correspond to zero columns. The resulting reduced matrix is then inverted. Except for the inverted submatrix, all other elements are set to zero. Then, minus ones are inserted on the diagonal locations of the zero rows and columns. Although the minus ones may be scattered throughout the matrix, a simpler situation can be arranged by defining a P of the following form:

$$P = \begin{bmatrix} 0 & P_{12} \\ 0 & P_{22} \end{bmatrix}$$

In this case, the routine will produce a matrix of the form:

$$H = \begin{bmatrix} -I_{11} & 0 \\ 0 & P_{22}^{-1} \end{bmatrix}$$

This is clearly not the inverse, since

$$H \quad P = \begin{bmatrix} 0 & -P_{12} \\ 0 & I_{21} \end{bmatrix}$$

However, if we have a set of simultaneous equations to be solved

$$P \quad X = B$$

The zero column condition yields

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} -I_{11} & 0 \\ 0 & P_{22}^{-1} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

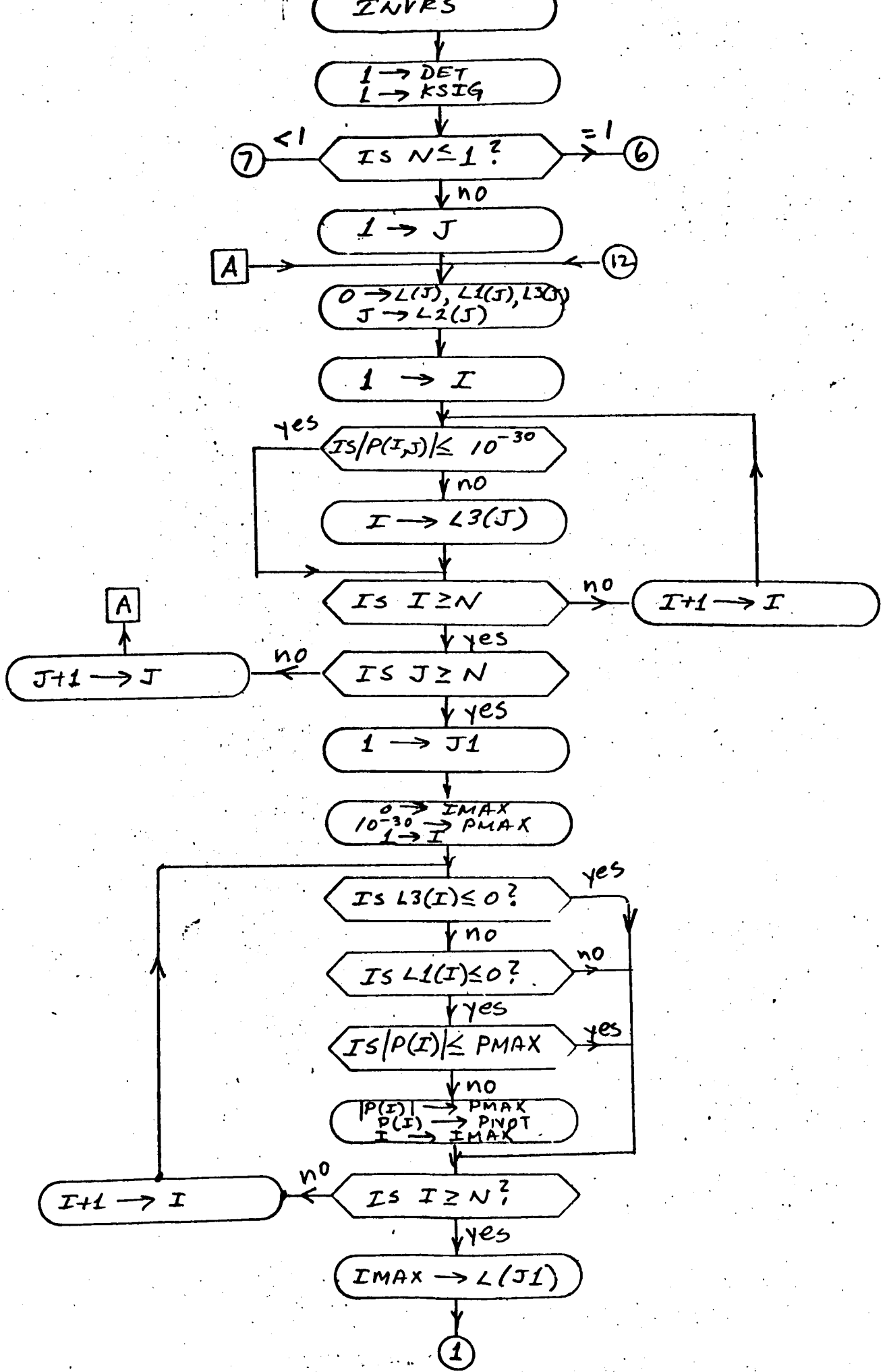
$$X_1 = -I_{11} \quad B_1$$

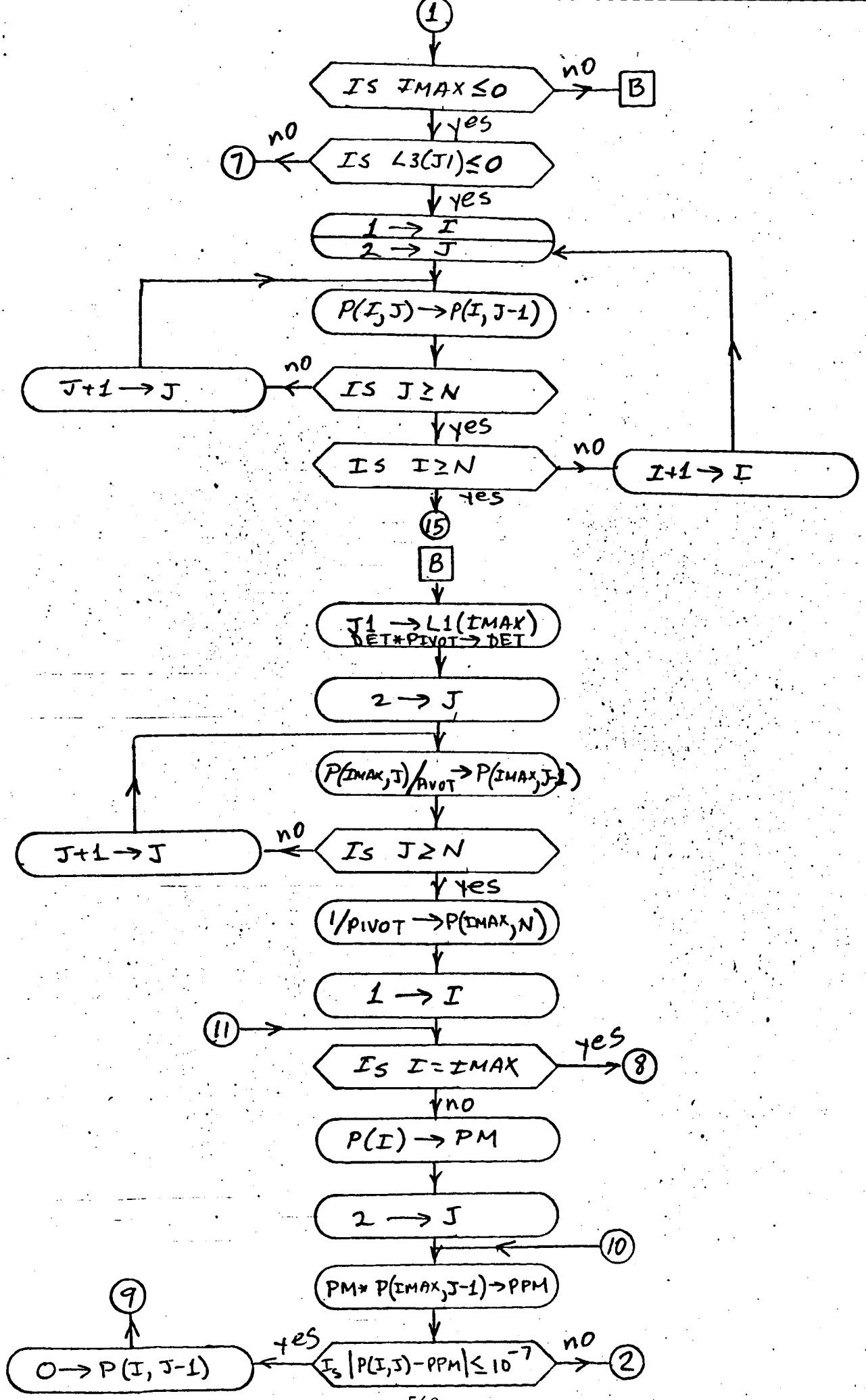
$$X_2 = P_{22}^{-1} \quad B_2$$

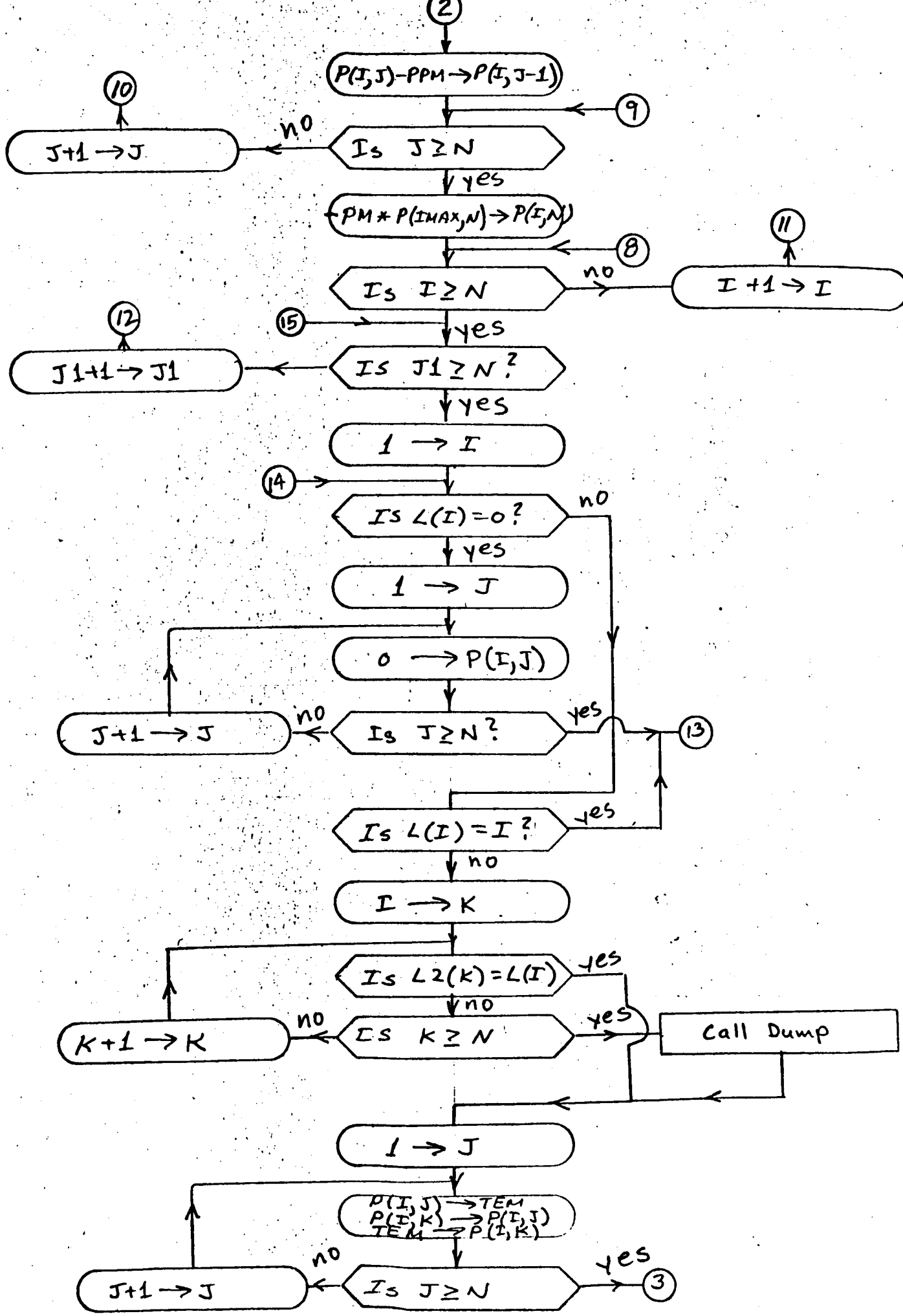
It must therefore be concluded that INVRS is not a generalized inversion routine and should not be used as such.

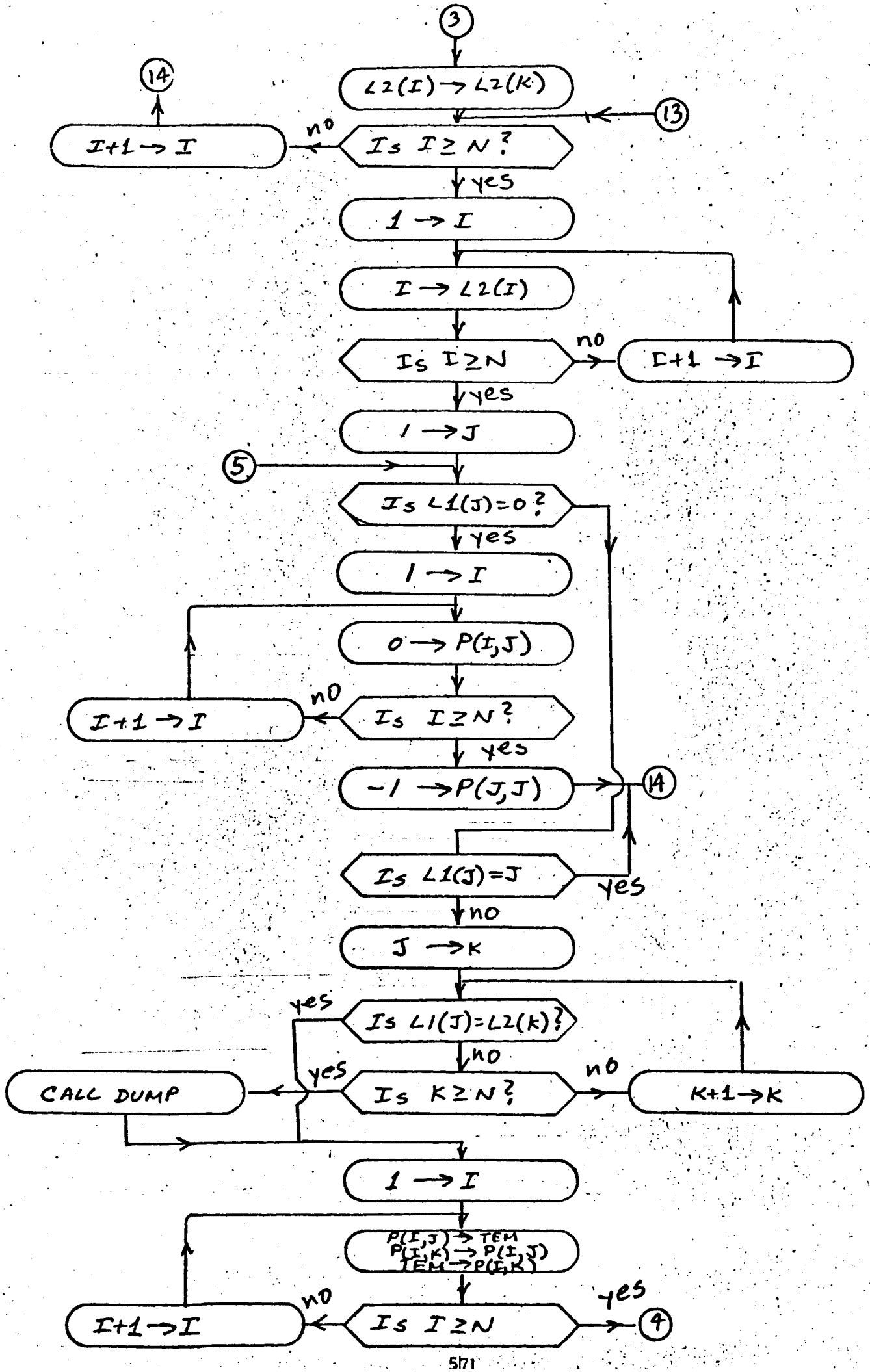
8. Other Subroutines Used

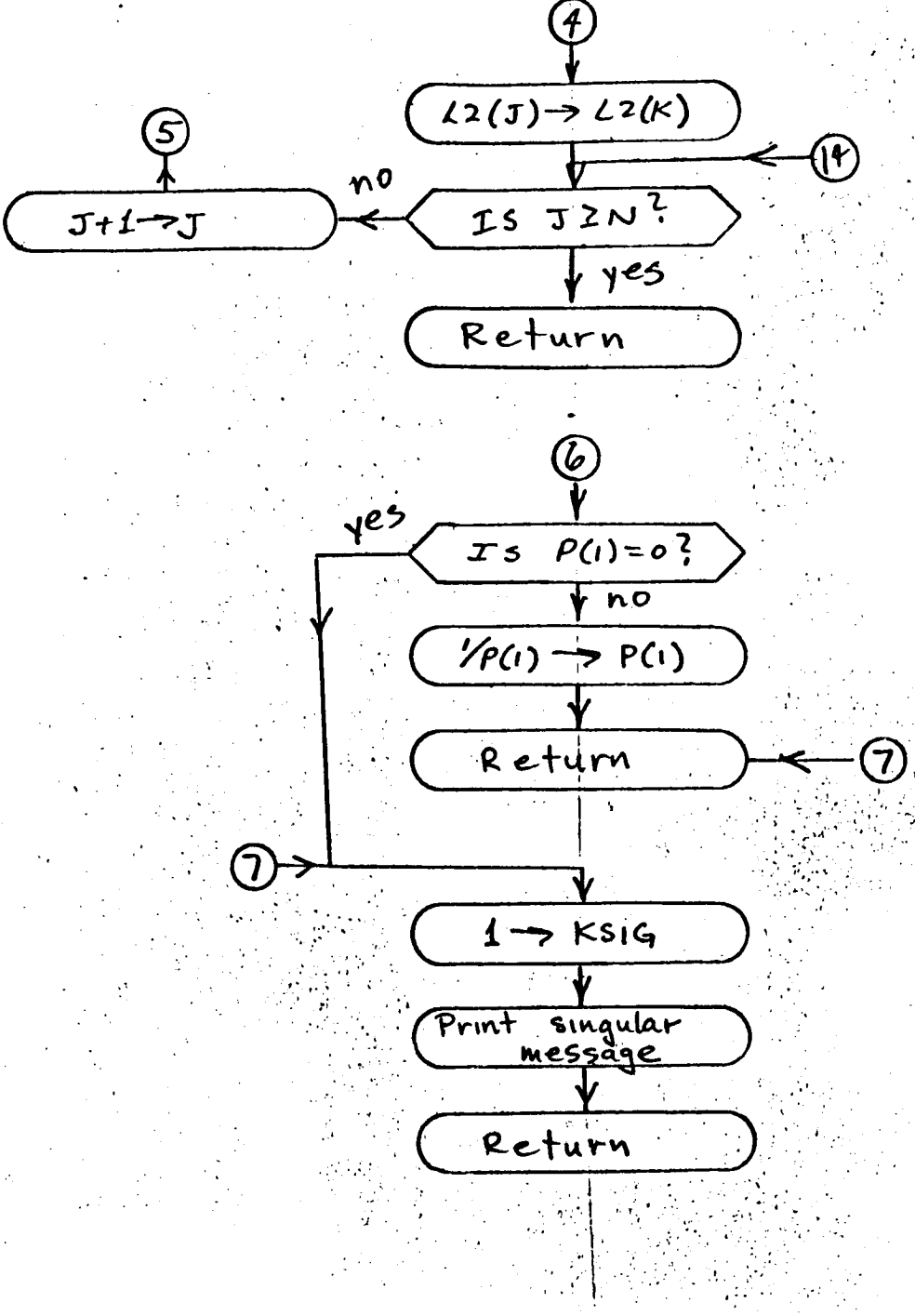
ABSF, DIFA, DUMP.


INVRS
1 → DET
1 → KSIG
Is N ≤ 1 ?
<1
=1
7
6
no
1 → J
A
12
0 → L(J), L1(J), L3(J)
J → L2(J)
1 → I
yes
Is |P(I,J)| ≤ 10^-30
no
I → L3(J)
Is I ≥ N
no
I+1 → I
yes
A
J+1 → J
no
Is J ≥ N
yes
1 → J1
0 → IMAX
10^-30 → PMAX
1 → I
Is L3(I) ≤ 0 ?
yes
no
Is L1(I) ≤ 0 ?
no
yes
Is |P(I)| ≤ PMAX
yes
no
|P(I)| → PMAX
P(I) → PIVOT
I → IMAX
I+1 → I
no
Is I ≥ N ?
yes
IMAX → L(J1)
1


1
IS IMAX≤0
no
B
yes
7
no
IS L3(J1)≤0
yes
1 → I
2 → J
P(I,J) → P(I,J-1)
J+1 → J
no
IS J≥N
yes
IS I≥N
no
I+1 → I
yes
15
B
J1 → L1(IMAX)
DET*PIVOT → DET
2 → J
P(IMAX,J)/PIVOT → P(IMAX,J-1)
J+1 → J
no
Is J≥N
yes
1/PIVOT → P(IMAX,N)
1 → I
11
IS I=IMAX
yes
8
no
P(I) → PM
2 → J
10
PM* P(IMAX,J-1) → PPM
9
0 → P(I,J-1)
yes
Is |P(I,J)-PPM|≤10^-7
no
2


2
P(I,J)-PPM→P(I,J-1)
10
9
J+1→J
no
Is J≥N
yes
PM*P(IMAX,N)→P(I,N)
11
8
Is I≥N
no
I+1→I
12
15
yes
J1+1→J1
Is J1≥N?
yes
1→I
14
Is L(I)=0?
no
yes
1→J
0→P(I,J)
J+1→J
no
Is J≥N?
yes
13
Is L(I)=I?
yes
no
I→K
Is L2(K)=L(I)
yes
no
K+1→K
no
Is K≥N
yes
Call Dump
1→J
P(I,J)→TEM
P(I,K)→P(I,J)
TEM→P(I,K)
J+1→J
no
Is J≥N
yes
3


3
L2(I) → L2(K)
13
14
I+1 → I
no
Is I ≥ N ?
yes
1 → I
I → L2(I)
Is I ≥ N
no
I+1 → I
yes
1 → J
5
Is L1(J) = 0 ?
yes
1 → I
0 → P(I,J)
I+1 → I
no
Is I ≥ N ?
yes
-1 → P(J,J)
14
Is L1(J) = J
yes
no
J → K
yes
Is L1(J) = L2(k) ?
no
CALL DUMP
yes
Is K ≥ N ?
no
K+1 → K
1 → I
P(I,J) → TEM
P(I,K) → P(I,J)
TEM → P(I,K)
I+1 → I
no
Is I ≥ N
yes
4


4
L2(J)→L2(K)
14
5
no
J+1→J
IS J≥N?
yes
Return
6
yes
Is P(1)=0?
no
1/P(1)→P(1)
Return
7
7
1→KSIG
Print singular message
Return

Program Description

1. Identification

   a. Routine Label

      MULT

   b. Name

      Matrix Multiply Subroutine

2. Function

   Computes the product of two real, single precision matrices.

3. Programming Language

   FAP coded for use in FORTRAN II

4. Usage

   a. Calling Sequence

      CALL MULT (A, B, C, M, N, L)

   b. Entry Conditions

      A = A matrix with M rows and L columns

      B = A matrix with L rows and N columns

   A and B may be the same matrix, but they must be different from C.

   c. Exit Conditions

      C is a matrix, with M rows and N columns, computed in the following way:

$$C = A * B$$

where * represents matrix multiplication and A and B are unchanged.

   d. Error Exits

      None.

Definition of Identifiers

The following identifiers are used in the flow diagram, but not necessarily in the subroutine:

[A] = The last address in matrix A

[B] = The last address in matrix B

[C] = The last address in matrix C

M = The number of columns in A

N = The number of columns in B

L = The number of columns in A and rows in B

ar = The first address of the row currently being operated upon in matrix A

bc = The first address of the column currently being operated upon in matrix B

L_a = The address of the element in A currently being operated upon

(L_a) = The element of A currently being operated upon

L_b = The address of the element in B currently being operated upon

(L_b) = The element in B currently being operated upon

L_c = The address of the element in C currently being operated upon

(L_c) = The element of C currently being operated upon

k = The dummy index used in forming the inner product of a row in A with a column in B. k ranges from L to 1.

m = The index used for counting the rows of A that have been processed. m ranges from M to 1.

n = The index used for counting the columns of B that have been processed. n ranges from N to 1.

5. Method

C is defined by the following formula:

$$C(i, j) = \sum_{k=1}^{L} A(i, k) * B(k, j)$$

$$1 \leq i \leq M \qquad 1 \leq j \leq N$$

In the particular FAP implementation, the elements of C are generated column by column. The general logic is similar to the following FORTRAN program:

```
DO  1  J = 1, N
DO  1  I = 1, M
SUM = 0.0
DO  2  K = 1, L
2 SUM = SUM + A(I, K) * B(K, J)
1 C(I, J) = SUM
END
```

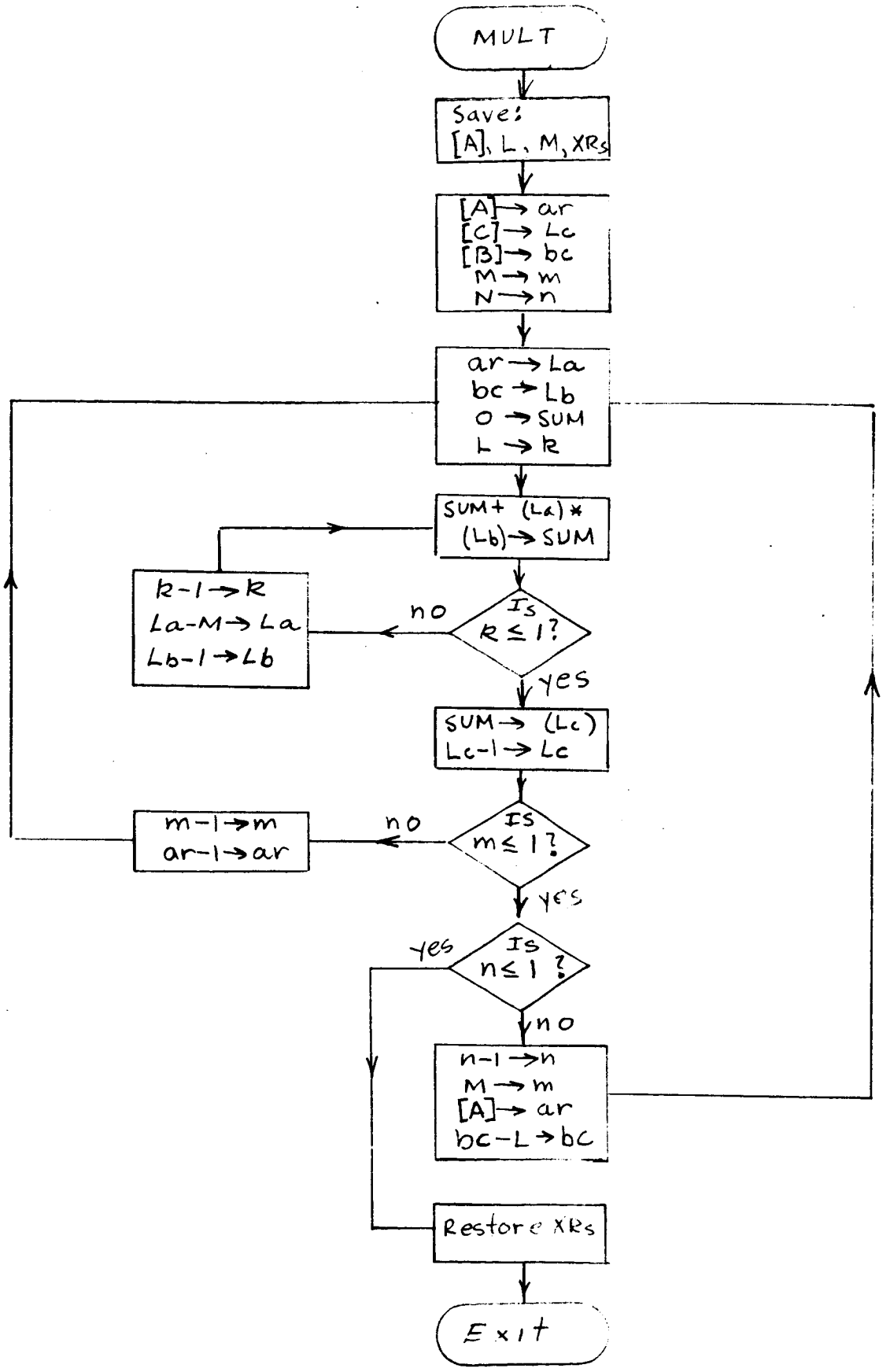
The flow diagram represents more closely the method by which the subroutine accomplishes the multiplication. For the purposes of increased speed, the logic is arranged so that the appropriate matrix element addresses may be rapidly formed from the previous one by incrementing an index register. To accomplish this, the first address of a row in A and a column in B are used as variables. To increment across a row of A, one starts with the first address of the row and repeatedly subtract M. In a like manner, to increment down a column of B, one starts with the first address of the column and repeatedly subtracts one. The first row of A starts at [A] , the second row of A starts at [A]-1, and the ith row of A starts at [A]-(i-1). The first column of B starts at [B], the second column starts at [B]-L, and the jth column starts at [B]-(j-1)L.

To summarize the incrementing, let L_a be the location of any element in A and let L_b be the location of any element in B. We then have

1. Increment the column number by 1:
   $L_a - M \rightarrow L_a$
   $L_b - L \rightarrow L_b$
2. Increment the row number by 1:
   $L_a - 1 \rightarrow L_a$
   $L_b - 1 \rightarrow L_b$

Remembering that the last location of A is [A] and the last location of B is [B], the address of any element may be generated by addition and subtraction only.

The subroutine forms the addresses for the required M*N inner products by the above described method.


MULT
Save:
[A], L, M, XRs
[A] → ar
[C] → Lc
[B] → bc
M → m
N → n
ar → La
bc → Lb
0 → SUM
L → R
SUM + (La) *
(Lb) → SUM
R-1 → R
La-M → La
Lb-1 → Lb
no
Is
R ≤ 1?
yes
SUM → (Lc)
Lc-1 → Lc
m-1 → m
ar-1 → ar
no
Is
m ≤ 1?
yes
yes
Is
n ≤ 1 ?
no
n-1 → n
M → m
[A] → ar
bc-L → bc
Restore XRs
Exit

Program Description

1. Identification

   a. Routine Label

      ROOT

   b. Name

      Solution of N Simultaneous Nonlinear Equations

2. Function

The subroutine enters SOLVE to perform Newton Raphson iteration steps. If a step does not meet certain criteria, the independent variable increment is modified to improve convergence.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL ROOT (F, X, P, FEPSL, FEPSL1, ITER, N, K, IRRNT)

   b. Entry Conditions

      F = The N dimensional vector of functions, f_i, which are to be reduced to zero

      X = The N dimensional vector of independent variables, x_i

      P = The N x N matrix defined by

$$p_{ij} = \frac{\partial f_i}{\partial x_j}$$

at various stages of completion, or its inverse.

      FEPSL = A constant used in testing the independent variable step size for convergence

      FEPSL1 = A constant used in testing the vector F for convergence

ITER = The maximum number of iterations allowed

N = The dimension of F and X. The number of equations

IPRNT = The number of iterations that will elapse before diagnostic printing will commence

K = The convergence flag, set by SOLVE

X1 = The previous value of X

X2 = The current value of the X correction

c. Exit Conditions

X = the value of X, computed by SOLVE, may be modified by ROOT

d. Error Exits

None.

5. Definition of Identifiers

ICNT = An index that counts the number of exits from SOLVE. ROOT performs a function only when ICNT = 0 and ICNT = N + 1 .

KSW = A flag that indicates the mode of operation of the ROOT routine:

KSW = 0 - the first entry of a particular solution

KSW = 1 - the ΔX magnitude reduction mode (ΔX = X2)

KSW = 2 - the ΔX one coordinate at a time reduction mode (ΔX = X2)

I = An index used at various places in the subroutine

BETA = A constant, ranging between 1 and 2^{-10}, by which X is multiplied in the magnitude reduction mode (KSW = 1, ΔX = X2)

SMP = For testing purposes, the sum of the absolute value of the elements of F

SM = The value of SMP for the last Newton Raphson iteration or for the last ΔX reduction step

XMAX = The element in X that has the largest magnitude

IMAX = The index of XMAX in X

6. Method

ROOT performs an operation every N + 1 entries into the routine. For intermediate entries, it passes control to SOLVE. After SOLVE has changed the X vector to a new value, ROOT inspects the change that the correction has made in the function:

$$SMP = \sum_{I=1}^{N} |F_{n+1}(I)| \qquad \text{evaluated at } X_{n+1}$$

Let SM be the value of the same function prior to the change in X:

$$SM = \sum_{I=1}^{N} |F_{n}(I)| \qquad \text{evaluated at } X_{n}$$

The magnitudes of SMP and SM are compared.

$$\text{IS} \qquad SMP \leq 100\ SM \ ?$$

If the above is not true, KSW is set to 1, and X is changed to the following value:

$$X_{n+2} = X1 + \left(\frac{1}{2}\right) X2$$

where X1 is the previous value of X and X2 is the Newton Raphson correction. SMP is now evaluated at X_n.

$$SMP = \sum_{I=1}^{N} |F_{n+2}(I)| \qquad \text{evaluated at } X_{n+2}$$

The test is repeated with the new SMP, and the old SM evaluated at X_n. For k failures of the test,

$$X_{n+1+k} = X1 + \frac{1}{2}^{k} X2$$

$$SMP = \sum_{I=1}^{N} |F_{n+1+k}(I)| \qquad \text{evaluated at } X_{n+1+k}$$

$$SM = \sum_{I=1}^{N} |F_{n}(I)| \qquad \text{evaluated at } X_{n}$$

During this process, SOLVE is bypassed. If SMP is not reduced sufficiently in 10 steps (k = 10), an alternative approach is tried.

When 10 of the steps have been completed, and the criterion is not satisfied, KSW is set to 2 and X is recomputed as follows:

$$X_{n+12} = X_n + \begin{bmatrix} 1 & & & & 0 \\ & 1 & & & \\ & & 1/2 & & \\ & & & 1 & \\ 0 & & & & 1 \end{bmatrix} X2$$

where the 1/2 is in the same row as the largest magnitude element of X2. Let M_h be defined by

$$M_h = \begin{bmatrix} 1 & & & & 0 \\ & 1 & & & \\ & & 1/2 & & \\ & & & 1 & \\ 0 & & & & 1 \end{bmatrix}$$

where the 1/2 is in the same row as the largest magnitude element of the vector

$$M_{h-1} \quad M_{h-2} \cdots\cdots M_1 \; X_2$$

After h steps, X is then

$$X_{n+1|1+h} = M_h \; M_{h-1} \; M_{h-2} \cdots\cdots M_1 \; X2$$

For each of these reductions the following test is made:

$$I_S \quad SMP \leq 5 \; SM$$

where SM is evaluated at X_n and SMP is evaluated at $X_{n+1|1+h}$.

Whenever one of the tests is satisfied, KSW is set to 1 and a new Newton Raphson step is initiated. From this point on, the values of X and F generated by the above process are used in place of the X_n and F_n generated by SOLVE.

Example

Assume that we have entered SOLVE and have the following vectors:

$$X1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad X2 = \begin{bmatrix} 1024 \\ 1024 \end{bmatrix} \quad X = \begin{bmatrix} 1024 \\ 1024 \end{bmatrix}$$

We also have SM evaluated at X1 and SMP evaluated at X. Assume

$$SMP < 100 \; SM$$

for the following 10 steps:

$$X = \begin{bmatrix} 1024 \\ 1024 \end{bmatrix} \quad \begin{bmatrix} 512 \\ 512 \end{bmatrix} \quad \begin{bmatrix} 256 \\ 256 \end{bmatrix} \quad \begin{bmatrix} 128 \\ 128 \end{bmatrix} \quad \begin{bmatrix} 64 \\ 64 \end{bmatrix}$$

$$\begin{bmatrix} 32 \\ 32 \end{bmatrix} \quad \begin{bmatrix} 16 \\ 16 \end{bmatrix} \quad \begin{bmatrix} 8 \\ 8 \end{bmatrix} \quad \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

When both elements have been reduced to 1 with no success, a new strategy is tried:

$$X = \begin{bmatrix}1024\\1024\end{bmatrix} \begin{bmatrix}512\\1024\end{bmatrix} \begin{bmatrix}512\\512\end{bmatrix} \begin{bmatrix}256\\512\end{bmatrix}$$

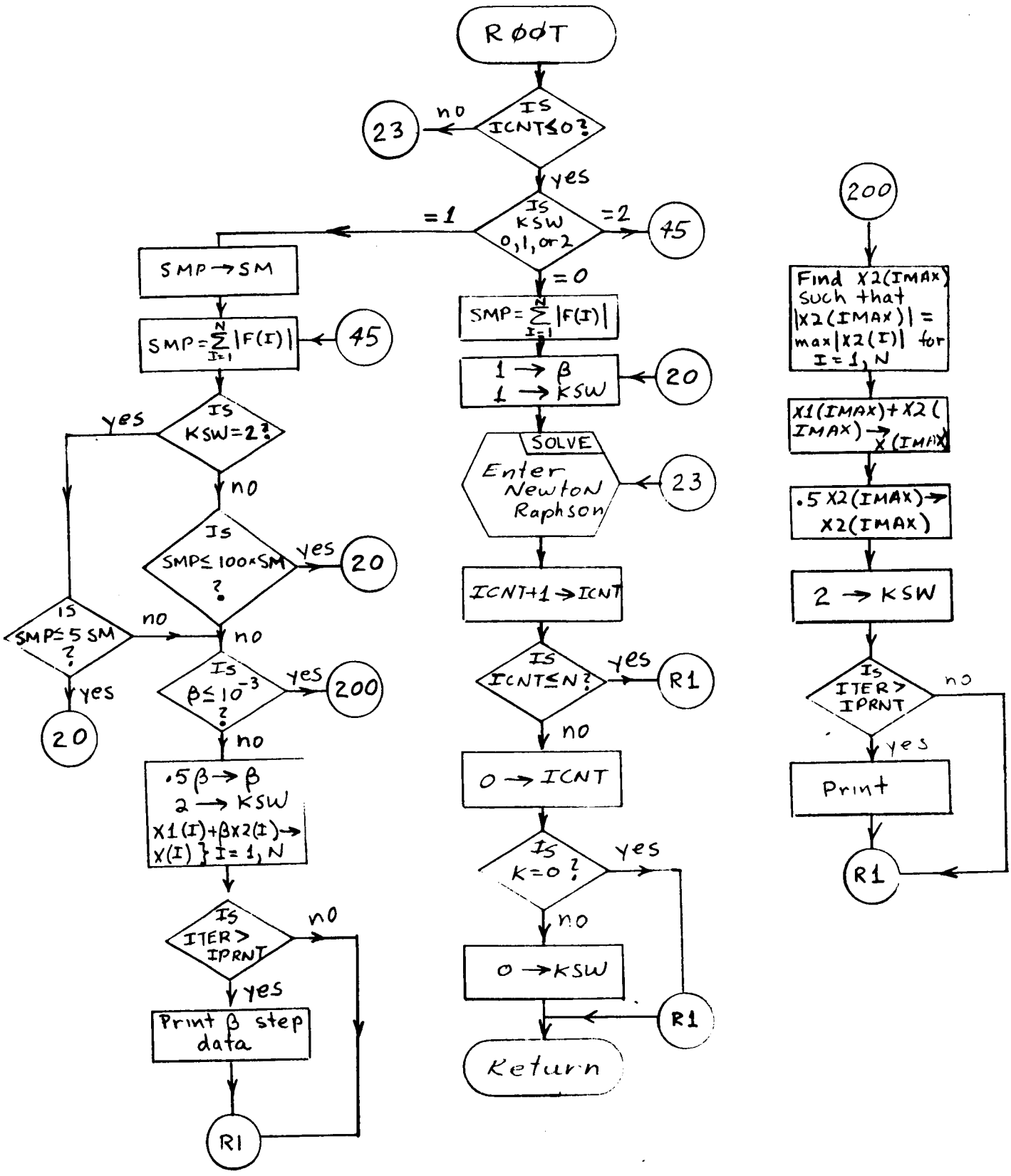
$$\begin{bmatrix}256\\256\end{bmatrix} \begin{bmatrix}128\\256\end{bmatrix} \begin{bmatrix}128\\128\end{bmatrix} \begin{bmatrix}64\\128\end{bmatrix}$$

The above is continued until:

$$SMP \leq 5\,SM$$

7. Subroutines Used

SOLVE


ROOT
IS ICNT≤0?
no
23
yes
IS KSW 0,1, or 2
=1
=2
45
=0
SMP→SM
SMP=Σ|F(I)|
45
IS KSW=2?
yes
no
IS SMP≤100×SM?
yes
20
IS SMP≤5 SM?
no
yes
20
no
IS β≤10^-3?
yes
200
no
.5β→β
2→KSW
X1(I)+βX2(I)→X(I) } I=1,N
IS ITER>IPRNT
no
yes
Print β step data
R1
SMP=Σ|F(I)|
1→β
1→KSW
20
SOLVE
Enter Newton Raphson
23
ICNT+1→ICNT
IS ICNT≤N?
yes
R1
no
0→ICNT
IS K=0?
yes
no
0→KSW
R1
Return
200
Find X2(IMAX) such that |X2(IMAX)| = max|X2(I)| for I=1,N
X1(IMAX)+X2(IMAX)→X(IMAX)
.5 X2(IMAX)→X2(IMAX)
2→KSW
IS ITER>IPRNT
no
yes
Print
R1

Program Description

1. Identification

   a. Routine Label

      SHFDI

   b. Name

      Move Residual to FVR

2. Function

   The difference between FVP and another vector is inserted into FVR.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SHFDI (FVR, FV31, FV41, FVP, LNG, LNL)

   b. Entry Conditions

      FV31 = A column vector with LNG elements
      FV41 = A column vector with LNL elements
      FVP = A column vector with LNL + LNG elements
      LNG = The dimension of FV31
      LNL = The dimension of FV41

   c. Exit Conditions

      FVR = A vector with LNL + LNG elements which is set to a value defined under "Method."

   d. Error Exits

      None.

5. Method

   The first LNG elements of FVR are computed as follows:

$$FV31(i) - FVP(i) \longrightarrow FVR(i)$$

$$1 \leq i \leq LNG$$

The next LNL elements of FVR are

$$FV41(i) - FVP(i) \longrightarrow FVR(i)$$

$$LNG + 1 \leq i \leq LNG + LNL$$

Either LNG, LNL, or both, may be zero.

In matrix notation, the function of the subroutine is

$$\begin{bmatrix} FV31 \\ FV41 \end{bmatrix} \qquad -FVP \longrightarrow FVR$$

Program Description

1. Identification

a. Routine Label

SHFIN

b. Name

Move FVP into FV31 and FV41

2. Function

The first subvector of FVP is moved to FV31, and the second subvector of FVP is moved to FV41.

3. Programming System

FORTRAN II

4. Usage

a. Calling Sequence

Call SHFIN (FVP, FV31, FV41, LNG, LNL)

b. Entry Conditions

FVP = A column vector with LNG + LNL elements
LNG = The number of elements in FV31
LNL = The number of elements in FV41

c. Exit Conditions

FV31 = A column vector with LNG elements
FV41 = A column vector with LNL elements
The first LNG elements of FVP are moved to FV31. The next LNL elements of FVP are moved to FV41. FVP remains unchanged.

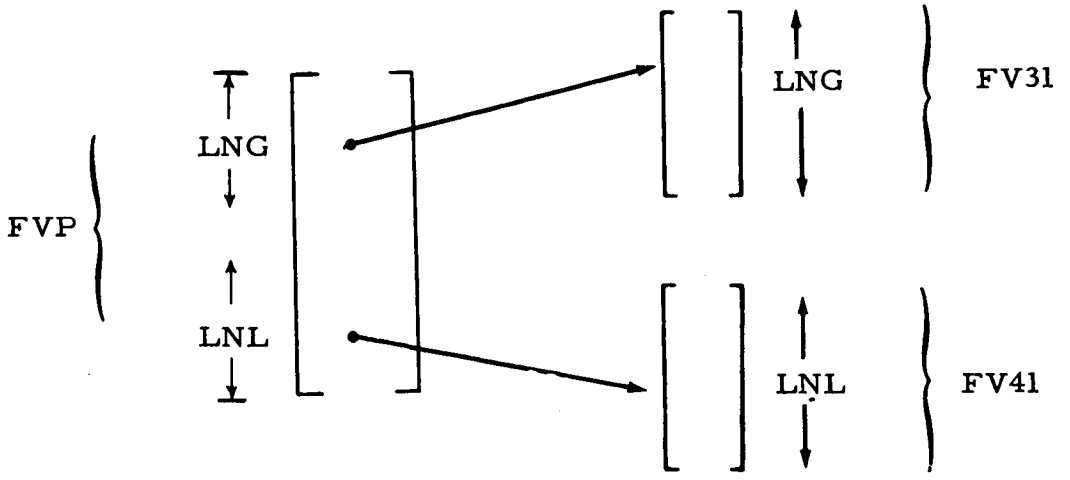
d. Error Exits

None.

5. Method

The dimensions LNG and LNL are tested so that either LNG or LNL, or both, may be zero.

The move may be represented by the following diagram:



Program Description

1. Identification

   a. Routine Label

      SHFTO

   b. Name

2. Function

   The vectors FV31 and FV41 are inserted into FVP.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SHFTO (FVP, FV31, FV41, LNG, LNL)

   b. Entry Conditions

      FV31 = A vector with LNG elements
      FV41 = A vector with LNL elements
      LNG = The dimension of FV31
      LNL = The dimension of FV41

   c. Exit Conditions

      FVP = A vector with LNG + LNL elements

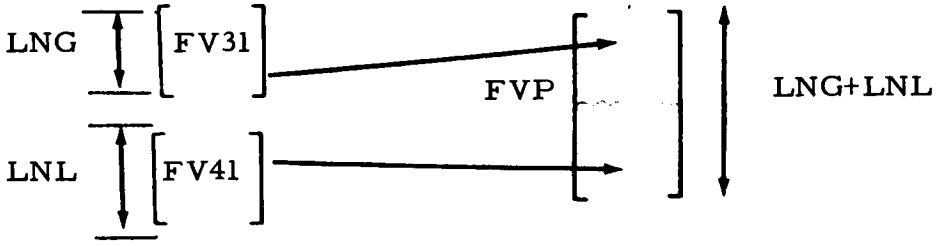
   The vectors FV31 and FV41 are moved to FVP as described under "Method."

   d. Error Exits

      None

5. Method

FV31 and FV41 are treated as subvectors of the same vector and moved to FVP. FV31 is inserted into locations 1 to LNG of FVP. Locations LNG + 1 to LNG + LNL are filled by FV41. The vectors FV31 and FV41 are left unchanged.



Either LNG, LNL, or both, may be zero.

Program Description

1. Identification

   a. Routine Label

      SOLVE

   b. Name

      Simultaneous Nonlinear Equation Solver

2. Function

In conjunction with a user-supplied FORTRAN II program for evaluating the functions representing the equations, the subroutine employs the Newton Raphson method to compute the solution of N simultaneous nonlinear equations to a specified accuracy.

3. Programming System

   FORTRAN II

4. Usage

   a. Calling Sequence

      CALL SOLVE (F, X, P, FEPSL, INTER, N, K, IPRNT, X2, CR)

   b. Entry Conditions

F	= An N dimensional column vector whose entries are the values of the N functions that are being reduced to zero. F is evaluated at X.
X	= The N dimension independent variable vector
FEPSL	= A parameter used to test ΔX for convergence
FEPSL1	= A parameter used to test F for convergence
ITER	= The maximum number of iterations that may be performed
N	= The number of equations
IPRNT	= The number of iterations that will elapse before iteration data begins to print

c. Exit Conditions

X = After the convergence criteria have been satisfied, X contains the solutions to the simultaneous equations

P = An N x N matrix containing the inverse of the partial matrix of the system of equations

K = the convergence flag having the following meanings:

K = 0 (no convergence)

K = 1 (convergence criteria satisfied)

K = 2 (maximum iterations exceeded)

X2 = the last correction to X (ΔX)

CR = The previous value of X

The above conditions, with the exception of those for K, hold true only after convergence is attained (K=1). For intermediate exit conditions, see "Method."

d. Error Exits

Although no error exits are made by the subroutine, when K is set to 1, the maximum number of iterations has been exceeded.

5. Definition of Identifiers

K = The convergence flag

SSQ = The square of the magnitude of the vector F

SSQS = The maximum SSQ computed during the solution

ITR = The internal subroutine index that is used to count the number of iterations. An iteration requires N + 1 subroutine entries.

IPRNT = The control constant for printing iteration data. Printing commences after IPRNT iterations have been completed.

F = The N dimensional vector of functions to be reduced to zero

FP = An N dimensional vector containing the negative of the value of F computed prior to the first entry of an iteration

NI = A row index used at various places in the subroutine

CRR = The X vector that produced the larges SSQ

CR = The value of the X vector upon the first subroutine entry of an iteration

NJ = The index used for counting the N subroutine entries during which P is computed

PMAX = Not used

FSTEP = The number by which one of the elements of X is incremented for computing one of the columns of P

FPL = A location containing -FP(NI) during the P matrix evaluation

XRND = A constant used to determine whether or not an excessive roundoff error has occurred in the computation of of a particular element of F and FP

P = Prior to matrix inversion, P contains the approximate partial matrix defined by:

$$D_X F = \frac{\partial F(I)}{\partial X(J)} \cong \frac{\Delta F(NI)}{\Delta X(NJ)} = P$$

INVRS = The subroutine used for inverting P

KSIG = The singular matrix flag, set by INVRS, which has the following meaning:
KSIG = 0 (nonsingular matrix)
KSIG = 1 (singular matrix)

SUM = The location used for accumulating the row-column inner product during matrix multiplication

6. Method

a. The Algorithm

It is assumed that the SOLVE subroutine is imbedded in a FORTRAN II program of the form shown in Exhibit 18.

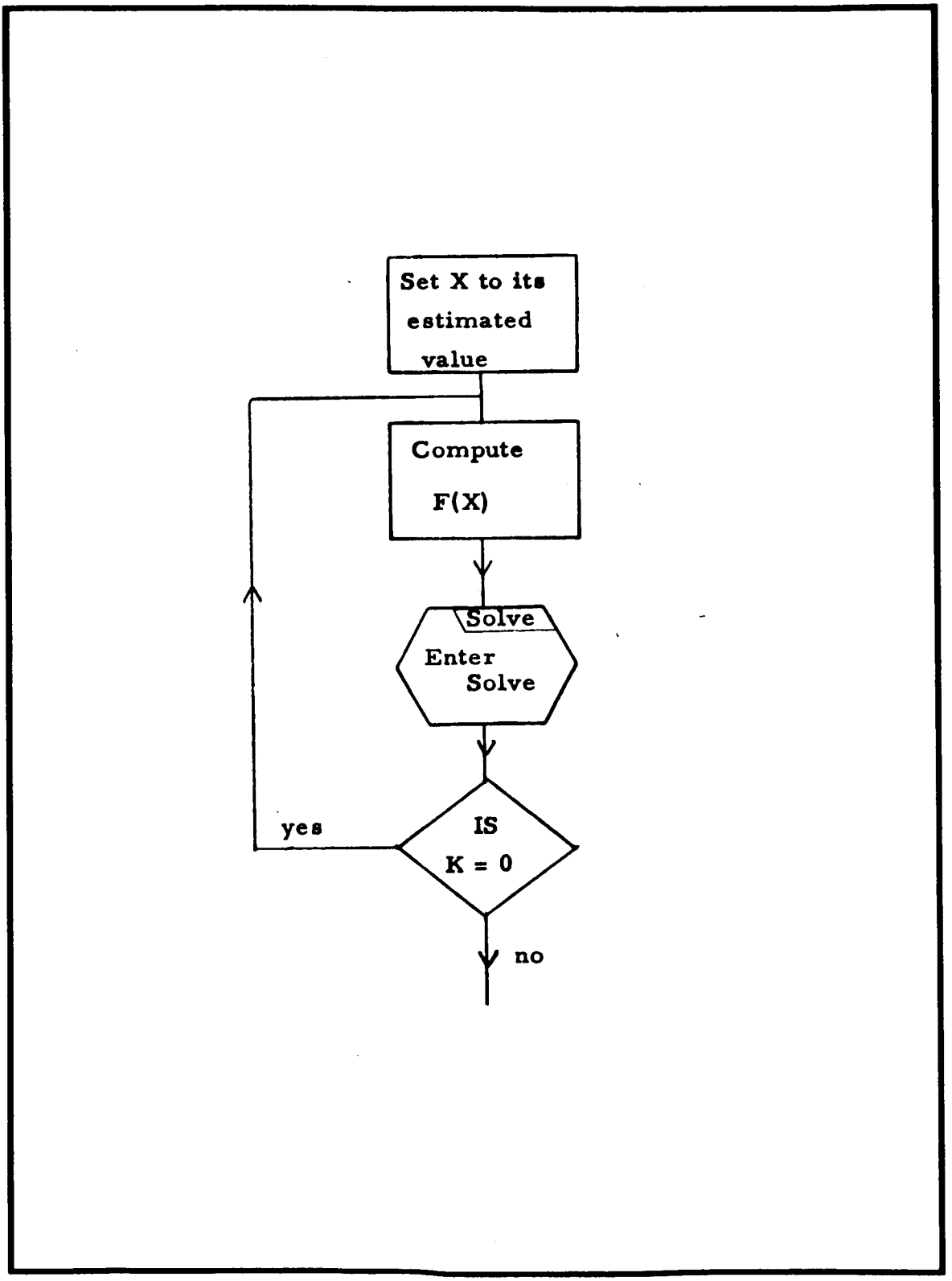


EXHIBIT 18 - FORTRAN II PROGRAM USING SOLVE FOR THE SOLUTION OF EQUATIONS

The routine is not completely general in that the function vector F is assumed to have a specified form.

$$F = -X + G^{-1}I$$

where G^{-1} is a real, nonsingular, N x N matrix and I is an N dimensional vector of functions of X. The partial derivative of F is defined with respect to X as

$$D_X F = \left[\frac{\partial F(i)}{\partial X(j)}\right]$$

where F(i) is the ith function of F and X(j) is the jth variable of X.

Assuming the nth estimate for X is known, the Newton Raphson formula for finding X_{n+1} is

$$X_{n+1} = X_n - (D\ F_n)^{-1}\ F_n$$

In the subroutine, F_n is computed upon the first entry. For each of N subsequent entries, one of the elements of X is perturbed in order to approximate a column of $D_x F$. Let the kth perturbation of X_n be defined as:

$$X_n^k = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ d^* X_n(k) \\ \cdot \\ 0 \\ 0 \end{bmatrix}$$

where $X_n(k)$ is the kth element of vector X_n. The jth column of $D_x F$ is then approximately:

$$\text{jth column of } D_x F \cong \frac{-F[X_n] + F[X_n + X_n^k]}{X_n(k)\ (d)}$$

where the quantity within the brackets following F defines the value of X at which F is evaluated.

The constant d is chosen as follows:

$$\text{if } X_n(k) > 10^{-20} \qquad \text{then } d = 10^{-3}$$

$$\text{if } X_n(k) \leq 10^{-2} \qquad \text{then } d = 10^{-20}$$

Convergence is considered to have been attained if:

$$\left|\frac{X_{n+1}(i) - X_n(i)}{X_n(i)}\right| \leq 5 \times 10^{-8}$$

$$1 \leq i \leq N$$

or if the following two criteria are met:

$$\left|\frac{X_{n+1}(i) - X_n(i)}{X_n(i)}\right| \leq \text{FEPSL}$$

$$\left|\frac{F_n(i)}{X_{n+1}(i)}\right| \leq \text{FEPSL1}$$

$$1 \leq i \leq N$$

b. Implications of the Assumed Equation Form

In a number of places in the program, it is assumed that the functions, F , are of the form

$$F = G^{-1}\ I - X - X$$

Whenever the INVRS routine indicates singularity of the P matrix by setting KSIG to one, the normal Newton Raphson formula is abandoned, and X_{n+1} is computed by the following formula:

$$X_{n+1} = -X_n + F_n = -X_n + X_n + G^{-1}I = G^{-1}I$$

A feature of the INVRS routine will result in the use of the above formula for selected elements of X_{n+1} and the Newton Raphson formula for the remaining elements. If it is assumed that the first k columns of the partial derivative matrix are zero, we have

$$P = \begin{bmatrix} 0 & P_{12} \\ 0 & P_{22} \end{bmatrix} \qquad \cong D_X F$$

The inverse routine will compute a matrix of the form

$$\begin{bmatrix} -I & 0 \\ 0 & P_{22}^{-1} \end{bmatrix}$$

Applying the Newton Raphson formula yields:

$$X_{n+1} = \begin{bmatrix} X_{n+1}^1 \\ X_{n+1}^2 \end{bmatrix} = \begin{bmatrix} X_n^1 \\ X_n^2 \end{bmatrix} + \begin{bmatrix} -I & 0 \\ 0 & P_{22}^{-1} \end{bmatrix} \begin{bmatrix} -F_n^1 \\ -F_n^2 \end{bmatrix} = X_n + P_n^{-1} * (-F_n)$$

*the transformation of P produced by INVRS

$$X_{n+1}^1 = X_n^1 + F_n^1$$

$$X_{n+1}^2 = X_n^2 - P_{22}^{-1} F_n^2$$

Note that the first equation above is similar to the formula used in the singular matrix case. The second equation is similar to the Newton Raphson iteration formula.

The assumed form of the function F

$$F = G^{-1}I - X$$

has certain implications. Some of these are

(1) The matrix inverse G^{-1} must exist in some neighborhood of the solution.

(2) The inverse of the matrix:

$$D_x F = D_x \left| G^{-1}I - U \right|$$

must also exist in some neighborhood of the solution. U is the identity matrix.

If the matrix G is constant and nonsingular, the Newton Raphson formula may be expressed as

$$F = G^{-1} I - X$$

$$D_x F = G^{-1} D_x I - U$$

$$\left[D_x F^{-1}\right] = \left[G^{-1} ({}_x I - G)\right]^{-1} = (D_x I - G)^{-1} G$$

$$X_{n+1} = X_n - (D_x I_n - G)^{-1} G \left| G^{-1} I_n - X_n \right| = X_n - (D_x I_n - G X_n$$

It should be noted that this is the same formula that would be derived from the function F_1 defined as

$$F_1 = I - GX$$

$$D_x F_1 = D_x I - G$$

$$X_{n+1} = X_n - \left\{ D_X I_n - G \right\}^{-1} (I_n - G X_n)$$

If the G matrix is a function of X , G^{-1} can no longer be factored out of the expression, and the formula is

$$F = G^{-1} I - X$$

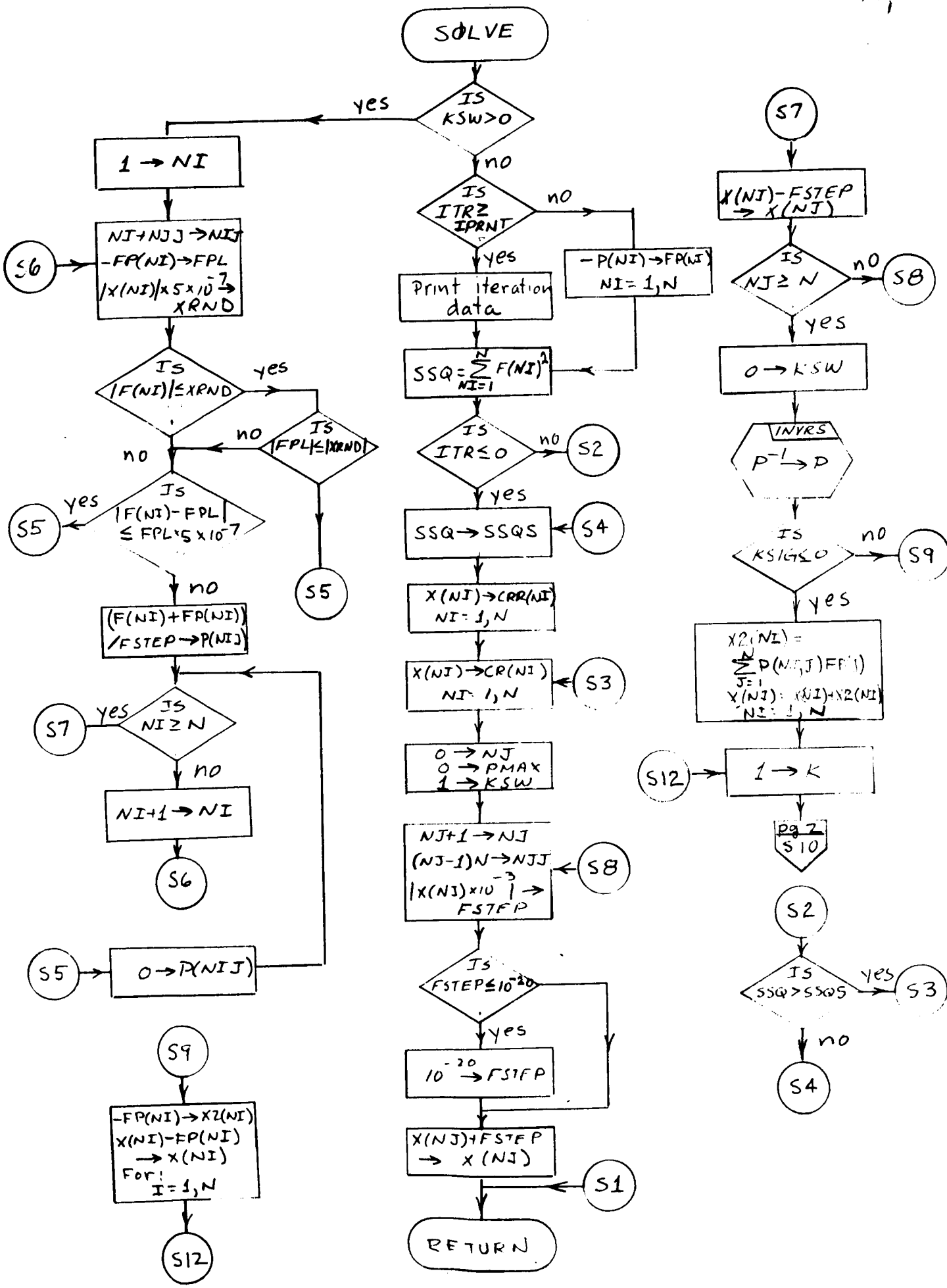
$$X_{n+1} = X_n - \left\{ D_X (G^{-1} I - X) \right\}^{-1} (G^{-1} I - X)$$

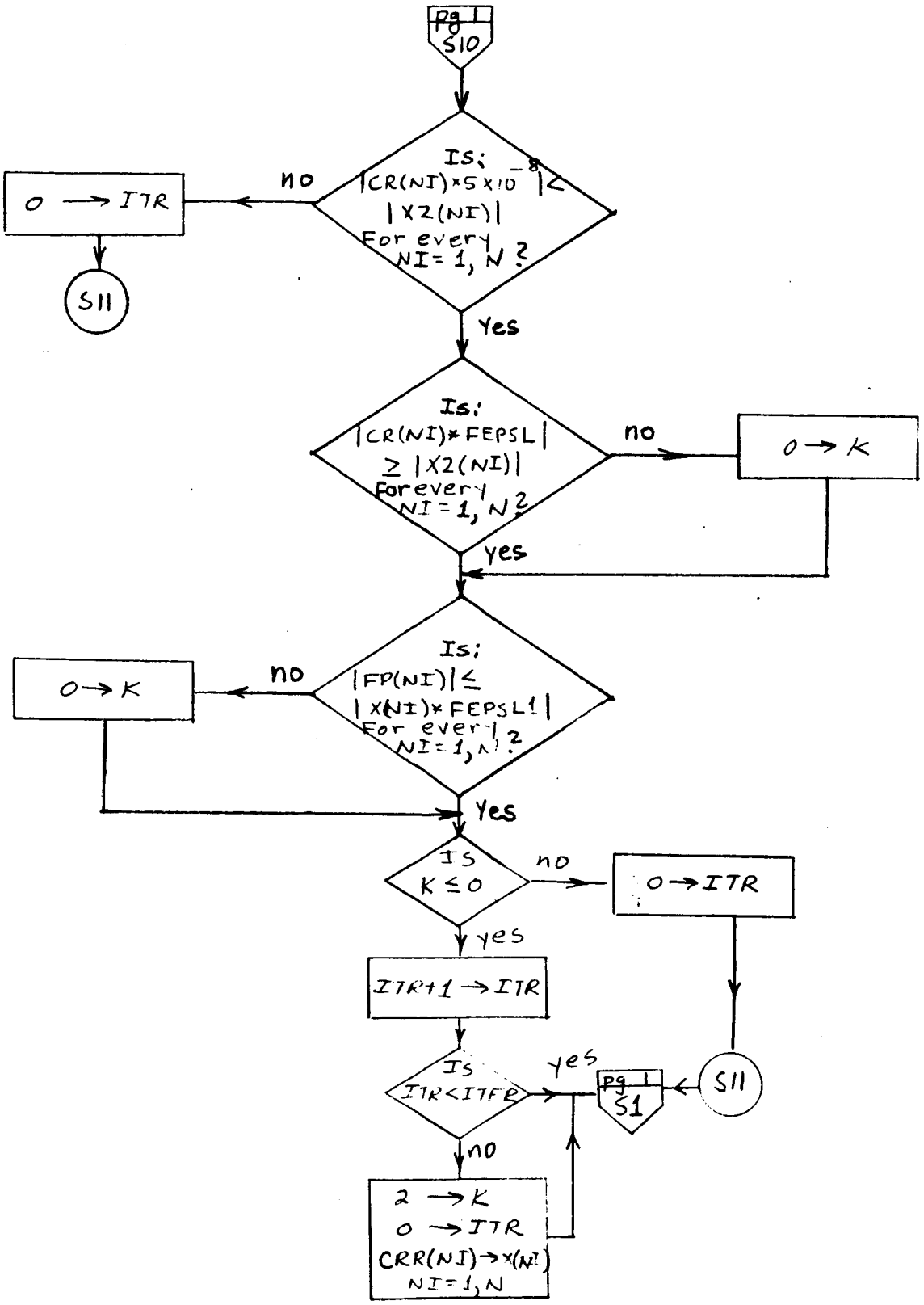
c. Diagnostic Printout

When the number of iterations exceeds IPRNT, the values of the index u with X(u) and F(u) are printed prior to each iteration.

SUBROUTINES USED

INVRS


SOLVE
IS KSW>0
yes
no
1 → NI
NI+NJJ → NIJ
-FP(NI) → FPL
|X(NI)|x5x10^-7 → XRND
S6
IS |F(NI)|≤XRND
yes
no
IS |FPL|≤|XRND|
yes
no
S5
IS |F(NI)-FPL| ≤ FPL*5x10^-7
yes
no
(F(NI)+FP(NI))/FSTEP → P(NI)
S7
IS NI≥N
yes
no
NI+1 → NI
S6
S5
0 → P(NIJ)
S9
-FP(NI) → X2(NI)
X(NI)-FP(NI) → X(NI)
For: I=1,N
S12
IS ITR≥IPRNT
yes
no
Print iteration data
-P(NI) → FP(NI)
NI=1,N
SSQ = Σ F(NI)^2 (NI=1 to N)
IS ITR≤0
yes
no
S2
SSQ → SSQS
S4
X(NI) → CRR(NI)
NI=1,N
X(NI) → CR(NI)
NI=1,N
S3
0 → NJ
0 → PMAX
1 → KSW
NJ+1 → NJ
(NJ-1)N → NJJ
|X(NI)x10^-3| → FSTEP
S8
IS FSTEP≤10^-20
yes
10^-20 → FSTEP
X(NJ)+FSTEP → X(NJ)
S1
RETURN
S7
X(NI)-FSTEP → X(NJ)
IS NJ≥N
yes
no
S8
0 → KSW
INVRS
P^-1 → P
IS KSIG≤0
yes
no
S9
X2(NI) = Σ P(NI,J)FP(J) (J=1 to N)
X(NI) = X(NI)+X2(NI)
NI=1,N
S12
1 → K
pg 2
S10
S2
IS SSQ>SSQS
yes
no
S3
S4


Pg 1
S10
Is;
|CR(NI)*5X10^-8| <
|X2(NI)|
For every
NI=1, N ?
no
0 → ITR
S11
Yes
Is:
|CR(NI)*FEPSL|
≥ |X2(NI)|
For every
NI=1, N ?
no
0 → K
yes
Is:
|FP(NI)| ≤
|X(NI)*FEPSL1|
For every
NI=1, N ?
0 → K
no
Yes
IS
K ≤ 0
no
0 → ITR
yes
ITR+1 → ITR
Is
ITR<ITFE
yes
Pg 1
S1
S11
no
2 → K
0 → ITR
CRR(NI) → X(NI)
NI=1, N

Program Description

1. Identification

   a. Routine Label

      SUM

   b. Name

      Matrix Sum or Difference Subroutine

2. Function

   Computes the sum or difference of two matrices.

3. Programming System

   Coded in FAP for use in FORTRAN II.

4. Usage

   a. Calling Sequence

      CALL PSUM (A, B, C, M, N)

      CALL MSUM (A, B, C, M, N)

   b. Entry Conditions

      A = A real matrix with M rows and N columns
      B = A real matrix with M rows and N columns
      M = The number of rows in A, B, and C
      N = The number of columns in A, B, and C

   c. Exit Conditions

      For the PSUM entry C is set to:

$$C = A + B$$

where the + represents matrix addition.

5. Definition of Identifiers

The following identifiers are used in the flow diagram:

[A] = The last address of matrix A

[B] = The last address of matrix B

[C] = The last address of matrix C

a = The current address in A

b = The current address in B

c = The current address in C

(a) = The contents of a

(b) = The contents of b

(c) = The contents of c

M = The number of rows in A, B, and C

N = The number of columns in A, B, and C

k = An index used to count the M x N steps of the sum or difference

For the MSUM entry C is set to:

$$C = A - B$$

where the - represents matrix subtraction.

In this subroutine, A, B, and C may be the same or different matrices.

6. Error Exits

None.

7. Method

If a_{ij}, b_{ij}, and C_{ij} are typical elements in A, B, and C respectively, then the sum and difference are defined as follows:

P SUM:

$$c_{ij} = a_{ij} + b_{ij}$$

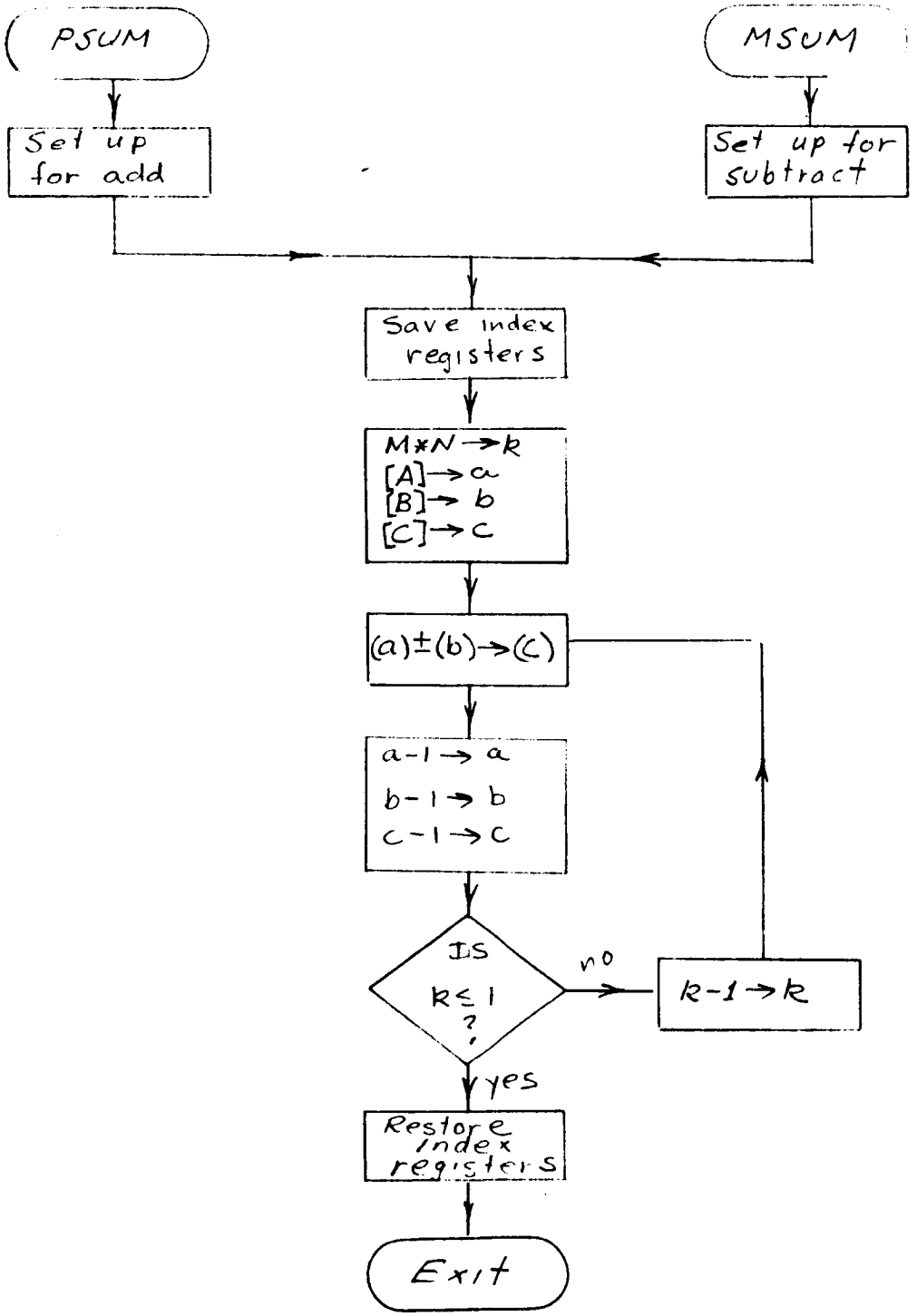
M SUM:

$$c_{ij} = a_{ij} - b_{ij}$$

In the FAP implementation, the matrices are treated as column matrices having M x N elements. A single loop of M x N steps accomplishes the addition or subtraction. The PSUM entry inserts an add command into the loop, and the MSUM entry inserts a subtract command.

8. Other Subroutines Used

None.


PSUM
MSUM
Set up for add
Set up for subtract
Save index registers
M*N → k
[A] → a
[B] → b
[C] → c
(a) ± (b) → (c)
a-1 → a
b-1 → b
c-1 → c
Is k ≤ 1 ?
no
k-1 → k
yes
Restore index registers
Exit

Program Description

1. Identification

   a. Routine Label

      ZEROX

   b. Name

      Clear Matrix to Zero

2. Function

   Zeros a specified number of locations in a matrix.

3. Programming

   FAP coded for use in FORTRAN II.

4. Usage

   a. Calling Sequence

      CALL ZEROX (A, N)

   b. Entry Conditions

      A = The matrix to be cleared to zero
      N = The number of locations, starting with the first, that are to be cleared

   c. Exit Conditions

      Zeros are inserted, column by column, into A until N locations have been cleared.

   d. Error Exits

      None.

5. Method

   A FAP program uses an indexed store zero command to clear N sequential locations in descending order, starting with the last location of A.

Program Description

1. Identification

   a. Routine Label

      UTF, RSTOP, STOP, ULF

   b. Name

      Independent variable stop maintenance routine.

2. Function

The subroutine updates the output time stop during integration. In addition, two rectangular pulse functions are provided for forming discontinuous functions of time.

3. Programming System

   FAP

4. Usage

   a. Calling Sequence

      RSTOP (FSTOP, FT, FHC) UTF(X)
      CALL STOP (FOUT, LINT) ULF(X)

   b. Entry Conditions

      FSTOP = Value of FT at which output has or will occur
      FT = The independent variable time
      FHC = The integration time step
      FOUT = The time intervals at which output will occur
      X = The time at which UTF or ULF is to change state

   c. Exit Conditions

      FOUT will be updated to next output time and/or next UTF or ULF change-of-state time.
      LINT is set to flag no discontinuity (0) or a discontinuity (1) in some function.
      UFT = one for $0 \leq FT \leq X$ and zero otherwise
      ULF = zero for $0 \leq FT \leq X$ and one otherwise.

d. Error Exits

Name.

5. Definition of Identifiers

TEM1 = The time of the next regular output time stop
STOP2 = The time of the next UTF or ULF time stop

6. Method

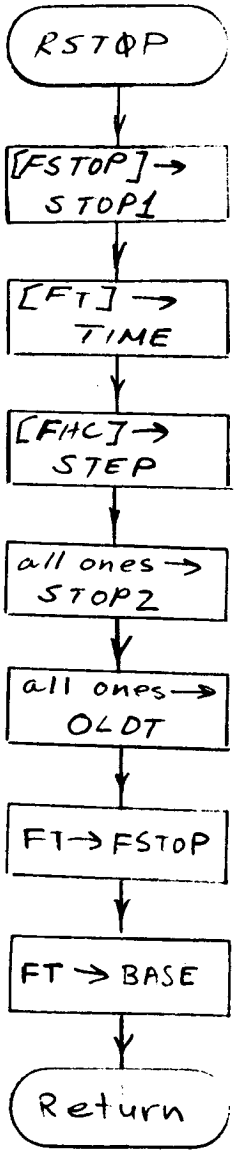
The independent variable stop time, FSTOP, is set to the next regular printout time or the next ULF or UTF change-of-state time, whichever is closest to the present time FT. Time differences are computed in un-normalized modulo two arithmetic. Thus two times are considered equal when the number of significant digits in their difference is less than a specified threshold.

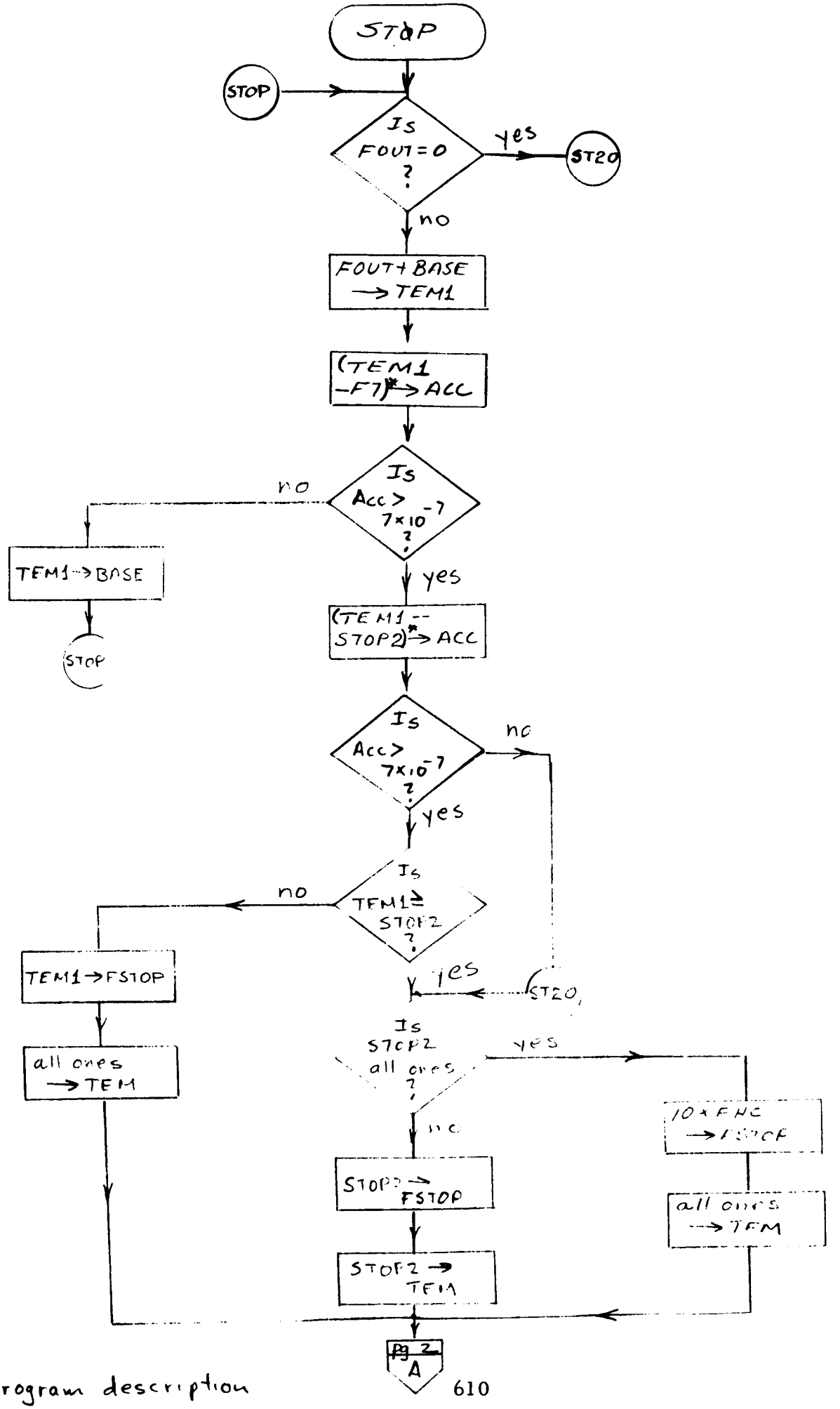
7. Other Subroutines Used

None.

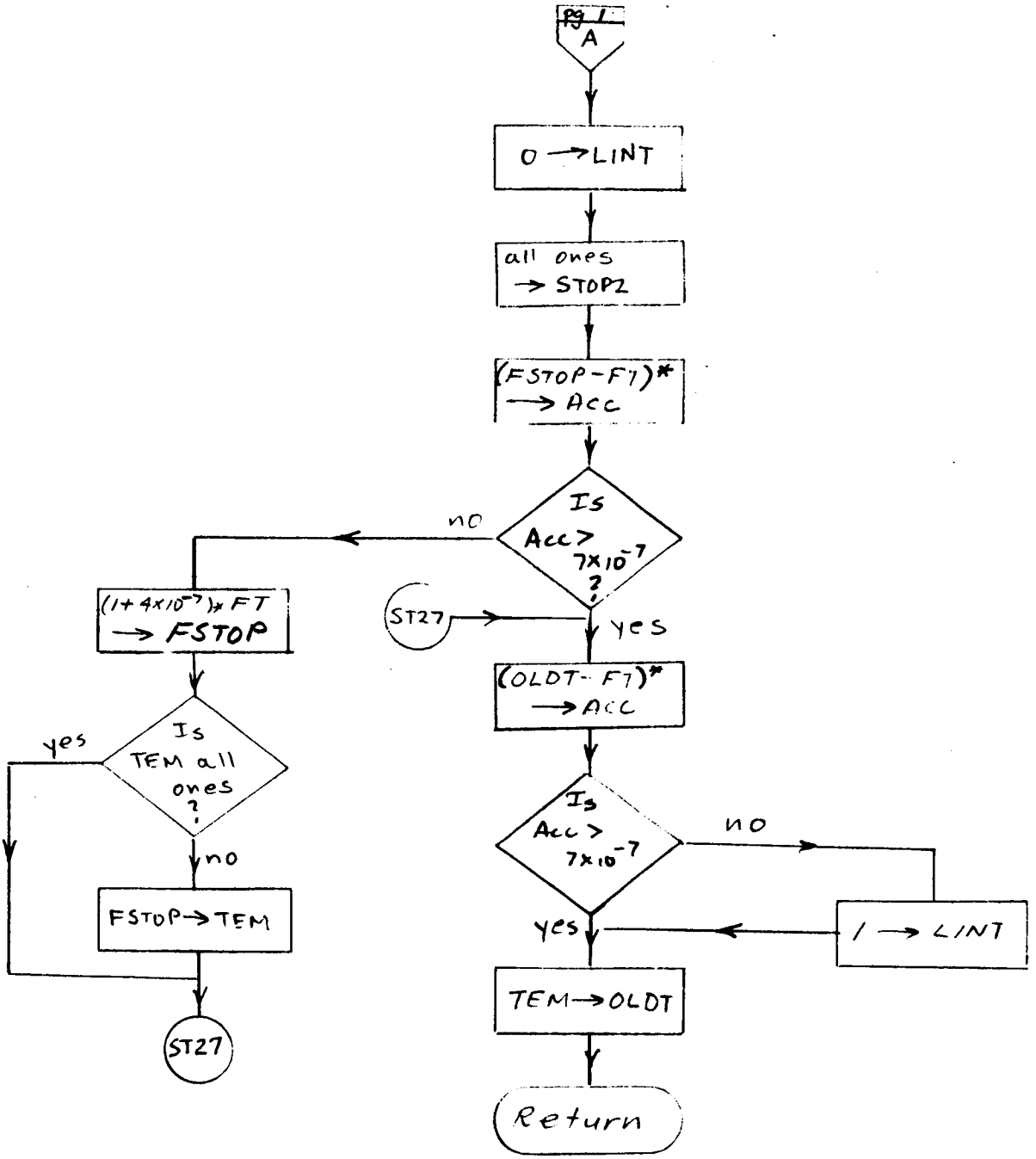
8. Using Subroutines

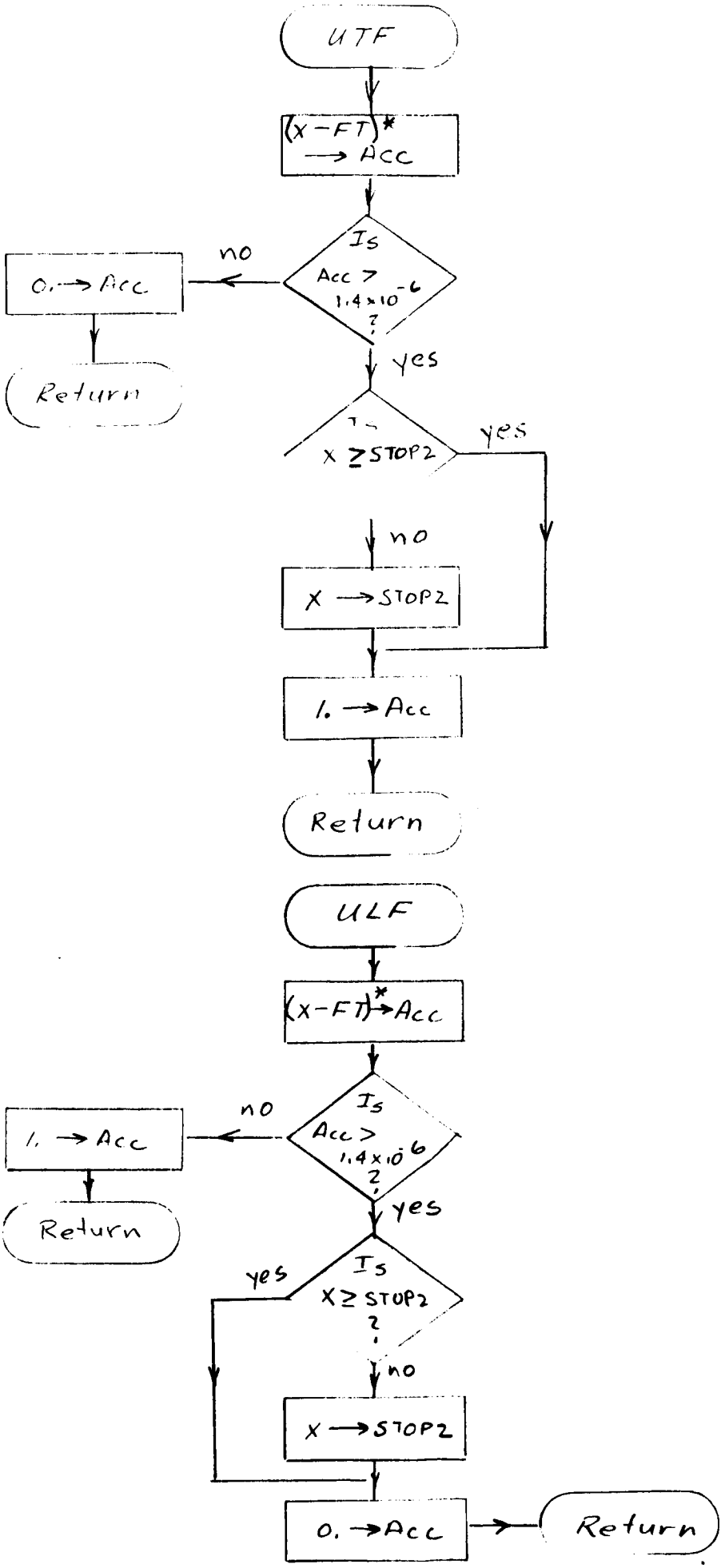
Execution program.


RSTOP
[FSTOP] → STOP1
[FT] → TIME
[FINC] → STEP
all ones → STOP2
all ones → OLDT
FT → FSTOP
FT → BASE
Return



* See program description


Pg 1
A
0 → LINT
all ones → STOP2
(FSTOP - F7)* → Acc
Is Acc > 7x10^-7 ?
no
yes
(1+4x10^-7)* FT → FSTOP
ST27
(OLDT - F7)* → Acc
Is TEM all ones ?
yes
no
FSTOP → TEM
Is Acc > 7x10^-7
no
yes
1 → LINT
ST27
TEM → OLDT
Return



* See program description

VII. DOCUMENTATION SYMBOLS AND CONVENTIONS

$<$	Less than
$\leq$	Less than or equal to
$=$	Equal to
$\geq$	Greater than or equal to
$>$	Greater than
$\neq$	Not equal to
$\cup$	Logical inclusive OR
$\oplus$ or $\cap \sim$	Logical exclusive OR
$\cap$	Logical AND
$\sim$	Logical Negation
:	A:B, compare A with B
X	Location X
(X)	Contents of location X
$(X)_{m,n}$	Contents of bit positions $(msb)_m$ through $(lsb)_n$ at address X
d(X)	Decrement portion of X
a(X)	Address portion of X
t(X)	Tag portion of X
((X))	Contents of location whose address is contained in X (indirect address)
[X]	Address of location whose contents are X
→	Replaces
ϕ	Letter "O"
0	Digit zero
I	Letter I
1	Digit one
Z	Letter "Z"
LϕC(i)	Parameter that indicates index register manipulation where i = 1, 2, 3 or 4

Other Conventions

For one-dimensional lists (strings): If the list name is L, then

First element = L_1

Second element = L_2

:

etc.

:

L_n

In the TAG data structure, if A is the head of the list:

$L_1 \equiv (d(A))$

$L_2 \equiv (a(d(A))) = (a(L_1))$ etc.