

2

# A Reproduced Copy OF

---

(NASA-CR-125887) SNAP: A COMPUTER PROGRAM  
FOR GENERATING SYMBOLIC NETWORK FUNCTIONS  
P.M. Lin, et al (Purdue Univ.) Aug. 1970  
125 p CACL 09B

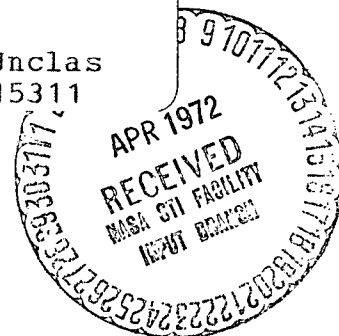
N72-20226

Unclas  
15311

FAC (NASA CR OR TMX OR AD NUMBER)

(CATEGORY)

G3/10



Reproduced for NASA  
by the  
**NASA** Scientific and Technical Information Facility

Reproduced by  
NATIONAL TECHNICAL  
INFORMATION SERVICE  
U S Department of Commerce  
Springfield VA 22151

124

TR-EE-70-16

SNAP  
A COMPUTER PROGRAM FOR GENERATING  
SYMBOLIC NETWORK FUNCTIONS

P. M. Lin  
G. E. Alderson

School of Electrical Engineering  
Purdue University  
Lafayette, Indiana

Supported by  
National Aeronautics and Space Administration  
Grant NGL 15-005-021

## TABLE OF CONTENTS

I. INTRODUCTION . . . . .	1
II. A GENERAL DISCUSSION OF THE BASIC ALGORITHMS . . . . .	6
II-1. Formulating the Signal-Flow Graph (SFG) . . . . .	6
a. Data Required . . . . .	6
b. Finding a Tree . . . . .	7
c. Rules for Formulating the Compact SFG . . . . .	8
d. The Gain Formula for "Closed" SFG . . . . .	11
II-2. Manipulating SFG Branch Weights . . . . .	12
II-3. Generating First Order Loops . . . . .	16
a. General Description . . . . .	16
b. A detailed Description of the Path-Finding Algorithm . . . . .	18
II-4. Generating Nontouching Loops of Order Two or More . . . . .	21
III. USER'S GUIDE . . . . .	28
III-1. Information Needed by User . . . . .	28
Appendix A (A Sorting Technique for Handling Multi- Input, Multi-Output Networks) . . . . .	36
Appendix B (A Brief List of Limitations on the Size and Type of Network Allowed) . . . . .	39
Appendix C (Selecting a "Good" Tree) . . . . .	40
III-2. Modifying the Dimension of Arrays . . . . .	42
IV. PROGRAMMER'S GUIDE . . . . .	47
IV-1. Definitions . . . . .	47
IV-2. Flow Charts . . . . .	50
IV-3. Program Listing . . . . .	100
REFERENCES . . . . .	124

## I. INTRODUCTION

The majority of computer aided circuit analysis programs belong to the class of "numerical programs", that is, the output is some numerical value. At the time of our research a few programs, most notably ANPI<sup>(8)\*</sup>, NASAP<sup>(9)</sup>, and CORNAP<sup>(10)</sup> could generate network functions as rational functions of  $s$  but did not allow the value of any element to be left in symbol form. The research project reported here represents, we believe, the first effort to generate symbolic network functions. By a symbolic network function we mean  $\frac{V_{out}}{V_{in}}$ ,  $\frac{V_{out}}{I_{in}}$ ,  $\frac{I_{out}}{V_{in}}$ , or  $\frac{I_{out}}{I_{in}}$  as a ratio of two polynomials of one of the following types:

- (1) all network element values are represented by symbols (the symbols need not all be different)

$$\text{Examples: } \frac{V_{out}}{V_{in}} = \frac{s^2 LRC}{s^2 2LRC + s(L+R^2C) + R}$$

$$\frac{V_{out}}{V_{in}} = \frac{ZYR}{2ZYR + Z + R^2Y + R}$$

- (2) some element values are specified numerically, some symbolically,

$$\text{Example: } \frac{V_{out}}{V_{in}} = \frac{s^2 R}{s^2 2R + s(.5 \times 10^6 + 150R^2) + .75 \times 10^8 R}$$

or

- (3) all element values are given numerically,

$$\text{Example: } \frac{V_{out}}{V_{in}} = \frac{s^2}{2s^2 + 2 \times 10^4 s + .75 \times 10^8}$$

There are many reasons why one may be interested in totally or partially symbolic network functions. The following presents a few of the more important ones.

- (1) Insight. To illustrate the added "insight" symbolic programs can provide

in comparison to numerical type programs, suppose we have been asked to verify that the network in Fig. 1 is a negative impedance converter for large  $\beta$ , i.e.,

$$Z_{in}(s) \rightarrow -Z_L(s) \text{ as } \beta \rightarrow \infty \quad (1)$$

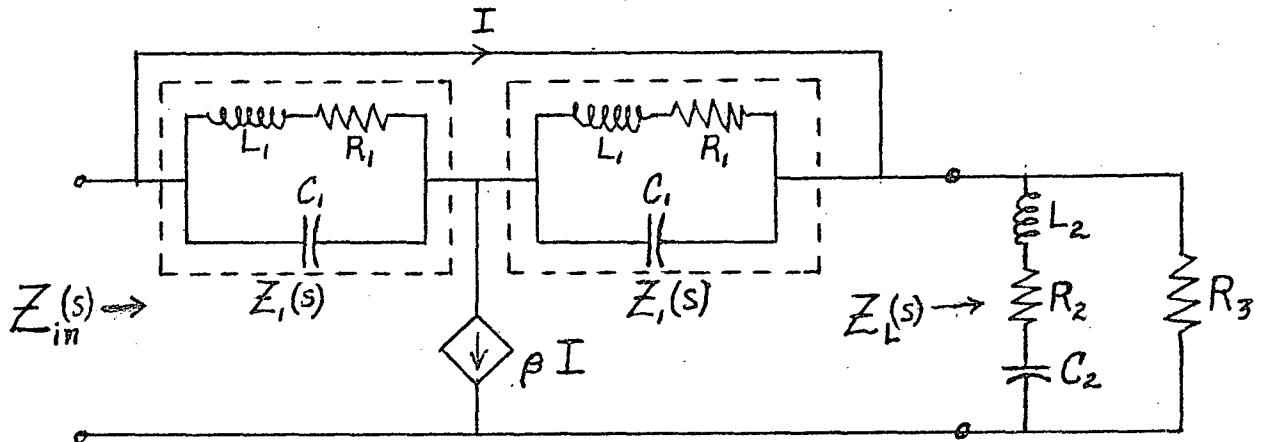


Figure 1

To verify (1) with some degree of certainty using a numerical program would require evaluating  $Z_{in}(s)$  (and  $Z_L(s)$ ) for many different values of  $\beta$  and frequency  $\omega$ , a time consuming process at best since most programs must completely re-evaluate the network response for every relatively large change in parameter values. Furthermore, the resulting verification would only be valid for the particular structure and component values chosen for  $Z_L(s)$ . With a symbolic program, one computer run gives the symbolic transfer function

$$Z_{in}(s) = Z_L(s) \left( \frac{2 + \beta}{2 - \beta} \right)$$

from which (1) follows immediately.

- (2) Error Control. To demonstrate how a symbolic program can be used to effectively control round-off error, consider the differential amplifier<sup>(1)</sup>

shown in Figures 1 and 2 of Section II-1. If network branches 1, 2, 5, and 11 are chosen as the tree for deriving the signal-flow graph (SFG), then the set of nontouching loops (all orders with sign attached) which belong to the numerator of the low frequency transfer function

$$\left. \frac{I_{out}}{V_{in}} \right|_{s=0} = \frac{N}{\Delta}$$

is given by

$$\left\{ \frac{R1R3A2}{(RE)^2 R2}, \frac{-R1R3A1}{(RE)^2 R2}, \frac{R1R3A1}{(RE)^2 R2}, \frac{-R1A1}{RER2}, \frac{-R3R1A2}{(RE)^2 R2}, \frac{R1A2}{RER2} \right\}$$

and the corresponding set for the denominator is given by

$$\left\{ \frac{R1}{R2}, \frac{R1}{R2}, \frac{R3}{RE}, \frac{R3}{RE}, \frac{R1R3}{R2RE}, \frac{R1R3}{R2RE}, \frac{R1R3}{R2RE}, \frac{R1R3}{R2RE} \right\}$$

Letting  $A2=A1$ ,  $R1=5K$ ,  $R2=15K$ ,  $R3=10K$ , and  $RE=25$ , evaluate  $N$  and  $\Delta$  by summing the terms in the order given in the above sets keeping each number generated to 8 significant digits. Then

$$\begin{aligned} N &= A1 [5.33333333 - 5.33333333 + 5.33333333 - .0133333333 - 5.33333333 + 0.133333333] \\ &= 3.3 \times 10^{-8} A1 \end{aligned}$$

and  $\Delta = 1335$

$$\text{Thus } \left. \frac{I_{out}}{V_{in}} \right|_{s=0} = \frac{3.3 \times 10^{-8}}{1335} A1$$

which is incorrect since  $N \neq 0$ . Although the above transfer function was derived using SFG theory, round-off errors which cause erroneous results can occur in any computer program restricted to numerical evaluation, and are generally very difficult to predict or control. Because round-off error enhancement in the evaluation of network functions often occurs as a result of widely separated values of some of the network elements, one method of error control would be to leave such element values in

symbolic form. This technique can be applied to the above example by noting that RE should be kept as a symbol since its value is considerably less than the other resistance values. Thus, keeping RE as a symbol and re-evaluating N gives

$$N=A1 \left[ \frac{3333.3333}{(RE)^2} - \frac{3333.3333}{(RE)^2} + \frac{3333.3333}{(RE)^2} - \frac{.33333333}{RE} \right. \\ \left. - \frac{3333.3333}{(RE)^2} + \frac{.33333333}{RE} \right] = 0$$

That is,

$$\left. \frac{I_{out}}{V_{in}} \right|_{s=0} = 0$$

- (3) Sensitivity Analysis. Sensitivity of the network function to changes in a particular network parameter can be found using a symbolic program by keeping this parameter as a symbol and then performing the required differentiation. Although there exist powerful numerical techniques for sensitivity analysis, the above procedure using a symbolic program has the particularly desirable feature of being less susceptible to round-off errors.
- (4) Parameter Variation. Suppose we wanted to evaluate the network function for many different values of one or more network parameters. Using a symbolic program, we could leave these parameters in symbol form and then efficiently and accurately perform the large number of required evaluations on the resulting symbolic network function. On the other hand numerical programs now available must re-derive the transfer function for every relatively large parameter change.
- (5) Iterative Piecewise Linear Analysis of Resistive Nonlinear Networks. <sup>(11)</sup>  
Part of this powerful analysis technique requires the solution of a

resistive linear network where some resistances and some d-c sources are kept in symbol form.

The primary objective\* of this project has been the development of some new or improved concepts needed to make a symbolic network analysis program efficient with respect to program storage and execution time. The project culminated in the program SNAP (Symbolic Network Analysis Program) which finds symbolic network functions for networks containing R, L, and C type elements and all four types of controlled sources. SNAP contains the following unique features:

- (1) The extensive use of a path-finding algorithm in place of matrix operations,
- (2) Efficient techniques for finding all loops of the SFG and for enumerating all higher order loops,
- (3) The use of the "compact signal-flow graph" instead of the "primitive signal-flow graph", and
- (4) A simple coding technique which is used
  - (a) manipulate symbols thereby allowing the complete program to be written in Fortran (another important aspect of the coding scheme is that it permits repeated symbols to be treated as one symbol), and
  - (b) determine whether or not loop sets touch in the algorithm for enumerating higher order loops.

New techniques for handling multi-inputs and multi-outputs are also presented in this report although they have not yet been incorporated into the program SNAP.

---

\* At about the same time the results of this project were disseminated,<sup>(3)</sup> another symbolic program<sup>(1)</sup> (by coincidence also called SNAP) whose primary concern is the on-line use for design purposes made its appearance.



## II. A GENERAL DISCUSSION OF THE BASIC ALGORITHMS

### II-1. Formulating the Signal-Flow Graph (SFG)

#### a. Data Required

A SFG is generated by SNAP (Symbolic Network Analysis Program) from data specifying the topological structure of the network, the input-output variables, and the characteristics of each network branch. The network topology is described by

- (a) A unique number for each branch, and
- (b) the initial and terminal nodes of each branch as determined by the assigned current direction.

The input to the network must be a single independent voltage or current source and the output requested must be the voltage or current associated with a network branch or the voltage between any two nodes of the network.\* Finally, each branch is characterized by

- (a) a symbol which specifies its type, i.e.,
  - passive branches: R, G, L, C, Y, Z
  - control sources: VV, VC, CV, CC
  - independent sources: E, I
- (b) a symbol representing the branch name together with the branch value if specified, and
- (c) the branch number of the control (for dependent sources only)

As an example, consider the network given below (from a paper by A. DeMari<sup>(1)</sup>).

---

\* Refer to Appendix A of section III-1 for a discussion on how to handle multi-inputs and multi-outputs.

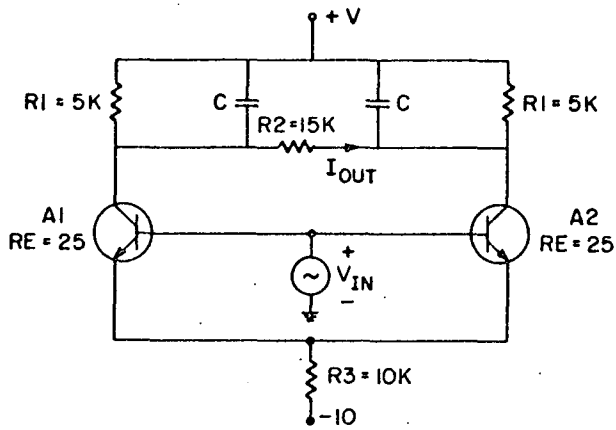


Figure 1

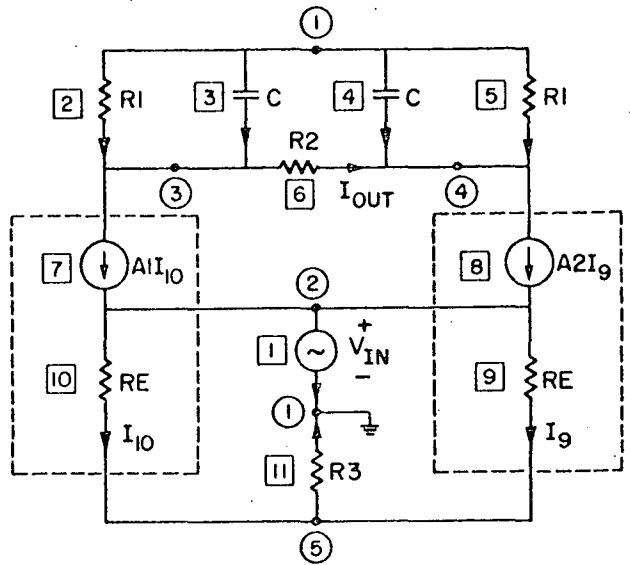


Figure 2

Table 1 (Network Data)

Branch Type	Branch Number	Initial Node	Terminal Node	Symbol	Value	Control
E	1	2	1			
R	2	1	3	R1	$= 5 \times 10^3$	
C	3	1	3	C		
R	5	1	4	R1	$= 5 \times 10^3$	
C	4	1	4	C		
R	6	3	4	R2	$= 15 \times 10^3$	
CC	7	3	2	A1		10
CC	8	4	2	A2		9
R	9	2	5	RE	$= 25$	
R	10	2	5	RE	$= 25$	
R	11	5	1	R3	$= 10 \times 10^3$	

b. Finding a Tree

The formulation of a SFG starts with the choice of a network tree. The selection of network branches to be used in the tree is made as follows: Independent voltage sources and controlled voltage sources are the first ones to be used. Then come the passive RLC elements in any order. In choosing the (J+1)th branch, the undirected graph formed by the J branches already selected is tested to determine whether a path exists between the two terminal nodes of the (J+1)th branch. If so, the branch under consideration is disqualified. If not, the (J+1)th branch is added to the tree. Let n be the number of nodes

of the network graph. When  $n-1$  branches are successfully chosen by the above process, we have obtained a tree.

As an example, consider the network of Fig. 2. If, following the selection of the voltage source, the passive branches are examined in the order by which they are listed, the tree shown in Fig. 3 results

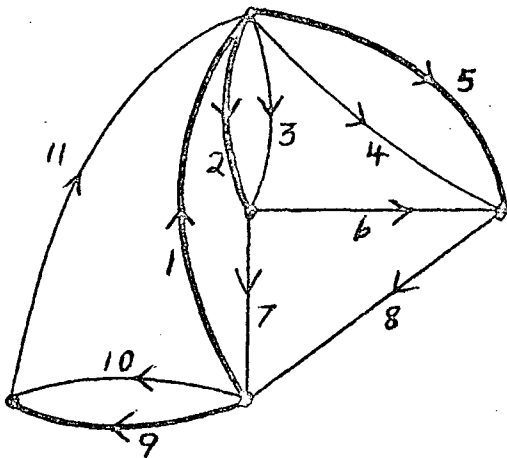


Figure 3

Tree branches:

$$b_1 = 1, b_2 = 2, b_3 = 5, b_4 = 9$$

Links:

$$l_1 = 3, l_2 = 4, l_3 = 6, l_4 = 7$$

$$l_5 = 8, l_6 = 10, l_7 = 11$$

It is important to note that the complexity of the SFG and consequently the time required to evaluate the transfer function depends on the tree selected. (2) A brief summary of the rules for choosing a "good" tree is given in Appendix C at the end of Section III-1.

### c. Rules for Formulating the Compact SFG

A "compact SFG" is a signal-flow graph whose node variables consist only of tree branch voltages and link currents except when additional nodes are needed for control sources or for the output variable. This type of SFG can be more efficiently evaluated than the so-called primitive SFG which contains one node for the branch voltage and another for the branch current.

The compact SFG is constructed according to the following rules: (An example as derived from Fig. 2, Fig. 3, and Table 1 is given in Fig. 4).

Rule (1): For each link  $l_k$ , the unique fundamental circuit  $C_k$  containing

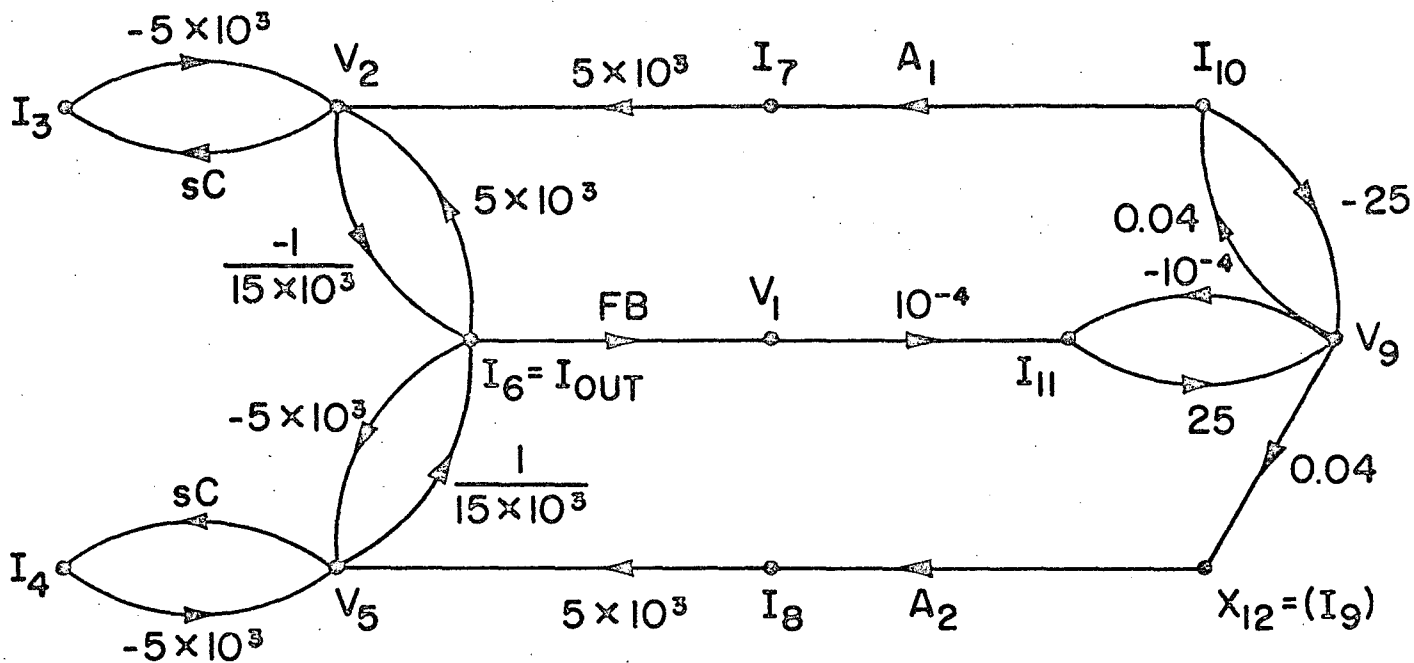


Figure 4: SFG

Table 2 (SFG DATA)

Initial Node	Terminal Node	Exponent of s	Branch Value	Branch Symbol
6	1	0		FB
2	3	1	1	C
3	2	0	$-5 \times 10^3$	-
5	4	1	1	C
4	5	0	$-5 \times 10^3$	-
2	6	0	$\frac{-1}{15 \times 10^3}$	-
6	2	0	$5 \times 10^3$	-
5	6	0	$\frac{1}{15 \times 10^3}$	-
6	5	0	$-5 \times 10^3$	-
10	7	0	1	A1
7	2	0	$5 \times 10^3$	-
9	12	0	.04	-
12	8	0	1	A2
8	5	0	$5 \times 10^3$	-
9	10	0	.04	-
10	9	0	-25	-
9	11	0	$-10^{-4}$	-
11	9	0	25	-
1	11	0	$10^{-4}$	-

branches  $b_i$ ,  $i=1,2,\dots,m$  is found. Two sets of SFG branches can then be created.

Set (a): For each passive branch in the tree branch set  $b_i$ ,  $i=1,2,\dots,m$ , a directed branch in the SFG is formed from node  $I_{l_k}$  to node  $V_{b_i}$  with weight equal to the impedance of branch  $b_i$ , prefixed with the proper sign (positive, if the directions of  $l_k$  and  $b_i$  concur in  $C_k$ , and negative otherwise).

Set (b): If the link  $l_k$  is a passive branch, a directed branch in the SFG is formed from each node  $V_{b_i}$ ,  $i=1,2,\dots,m$ , to node  $I_{l_k}$ , having weight equal to the admittance of link  $l_k$ , prefixed with the proper sign (negative, if the directions of  $l_k$  and  $b_i$  concur in  $C_k$ , positive otherwise).

Rule (2): If any of the four types of controlled sources are present, a directed branch is created in the SFG from the controlling variable to the controlled source, having weight equal to the constant of proportionality ( $g_m$ , beta, etc.). If the controlling variable is a link voltage or a tree branch current, one more node is added to the SFG to represent this controlling variable  $X$  (node  $X_{12}$  in Fig. 4 is a node of this type).  $X$  is then expressed in terms of the tree branch voltage or link current through a simple immittance relationship.

Rule (3): If the desired output  $Y$  is neither a tree branch voltage nor a link current, then one node is added to the SFG to represent  $Y$ .  $Y$  is then expressed in terms of tree branch voltages or in terms of a link current through a simple immittance relationship.

Rule (4): Finally, the SFG is "closed" by adding a branch with a symbolic weight (FB), directed from the output to the input node.

d. The Gain Formula for "Closed" SFG <sup>(4)</sup>

The purpose for introducing the closed SFG is because only all orders of non-touching loops need be found as opposed to the evaluation of Mason's formula which required enumerating certain paths as well as loops.

To derive the gain expression for the closed SFG consider first Mason's equation for the transfer function.

$$T = \frac{X_o}{X_i} = \frac{\sum_{i=1}^m P_i \Delta_i}{\Delta}$$

where

$\Delta = 1 + \sum_j (-1)^j \sum_k L_{k,j}$  is the determinant of the SFG

$L_{k,j}$  is the product of the transmittances of the  $k^{\text{th}}$  set of non-intersecting loops of order  $j$ .

$P_i$  is the transmittance product of the  $i^{\text{th}}$  path between  $X_i$  and  $X_o$

$\Delta_i$  is the partial determinant obtained from  $\Delta$  after removal of all loops intersecting the  $i^{\text{th}}$  path between  $X_i$  and  $X_o$ .

Let  $\Delta_c$  be the determinant of the closed SFG. It is then noted that since

$\{P_i\}_{i=1}^m$  is the set of all paths from  $X_i$  to  $X_o$ , the loops present in the closed SFG not present in the original SFG will be precisely  $\{(FB)P_i\}_{i=1}^m$

where FB is the symbol assigned to the added branch. Further, since the path FB contains only nodes  $X_i$  and  $X_o$  which, in turn, are present in every

path  $P_i$ ,  $i=1,2,\dots,m$ , it follows that the non-intersecting loop combinations that do not touch the loops  $(FB)P_i$ ,  $i=1,2,\dots,m$  will be precisely those combinations which do not touch the path  $P_i$ ,  $i=1,2,\dots,m$ . It follows that

$$\Delta_c = (FB) \sum_{i=1}^m P_i \Delta_i + \Delta$$

Thus, the transfer function can be found by simply sorting the terms of the determinant of the closed SFG.

## II-2. Manipulating SFG Branch Weights

Each branch weight in the SFG is of the form

$$\text{Constant} \cdot \text{Symbol} \cdot s^n$$

If an arbitrary branch has an initial node  $X_i$  and a final node  $X_f$ , then the three parameters

$$C(X_i, X_f) = \text{constant}$$

$$S(X_i, X_f) = \text{symbol}$$

and  $E(X_i, X_f) = \text{exponent of } s$

completely define the weight of the branch. After a loop or a set of nontouching loops has been found in the SFG, say by some path-finding technique, it is desirable to combine the weight parameters of each branch in the loop set to form a composite loop set weight. The loop set constant may be easily formed by taking the product of the constants associated with each branch. Similarly, the loop set exponent parameter is readily found by summing the exponents assigned to each branch. However, because computers are not particularly adept at symbol manipulation, it is inefficient with respect to both time and storage to form directly a composite loop set symbol. A much better technique is to convert each branch symbol into a numeric code. These codes are assigned to the SFG branches as follows: Each distinct symbol in the SFG is stored in the array  $S(j)$  and assigned a code  $B^j$  where  $B$  is some base  $B \in \{2, 4, \dots, 2^m\}$ . Now for an arbitrary SFG branch having initial node  $X_i$  and final node  $X_f$  which contains the symbol  $S(n)$ , the code

$$K(X_i, X_f) = B^n$$

is assigned.

The real value of this coding technique stems from the fact that the composite loop set code formed by summing the codes representing the individual branch symbols can be uniquely decoded provided the number of identical symbols combined into any code is less than B.

As an example of the above concepts for manipulating the SFG branch weights, refer to the SFG shown in Fig. 4. Consider, in particular, the loop defined by the node sequences

$$V_2 - I_3 - V_2 \quad \text{and} \quad V_4 - I_5 - V_4$$

Then

$$\begin{aligned} \text{composite loop set constant} &= (-5 \times 10^3)(1)(-5 \times 10^3)(1) \\ &= 25 \times 10^6 \end{aligned}$$

and

$$\text{composite loop set s power} = 0 + 1 + 0 + 1 = 2$$

To find the loop set code, an array of distinct symbols of the SFG and their corresponding codes must be set up.

$$\text{no symbol} \longleftrightarrow 0$$

$$S(1) = \text{FB} \longleftrightarrow 4^0$$

$$S(2) = \text{C} \longleftrightarrow 4^1$$

$$S(3) = \text{A1} \longleftrightarrow 4^2$$

$$S(4) = \text{A2} \longleftrightarrow 4^3$$

Note that because there will be at most two identical symbols in any code, the base 4 was chosen. Using the above codes gives

$$\begin{aligned} \text{composite set code} &= K(V_2, I_3) + K(I_3, V_2) \\ &\quad + K(V_4, I_5) + K(I_5, V_4) \\ &= 4^1 + 0 + 0 + 4^1 \\ &= 8 \end{aligned}$$

Now to decode this number, say in the output, we would write

$$\begin{aligned} 8 &= 2(4^1) \\ &= (S(2))^2 \\ &= \text{C}^2 \end{aligned}$$



which is indeed the symbol associated with the loop admittance product. The above coding scheme for manipulating symbols is easily adapted to the computer by incorporating the masking operation .AND. . To determine the number of S(1) type symbols contained in a given code, the .AND. operation is applied to the code and B-1. In general, the number of S(J) symbols is found by dividing (using integer division so as to truncate the remainder) the code used to determine the number of S(J-1) symbols by B and then applying the .AND. operation. For example, consider the loop set previously discussed.

$$\begin{aligned} \text{loop set code} &= 8 = (000000000100)_2 \\ B - 1 &= 3 = (000000000011)_2 \\ (\text{loop set code}).\text{AND.}(B-1) &= 8.\text{AND.}3 \\ &= 0 \end{aligned}$$

Thus, the symbol S(1) = FB is not present. Now divide the loop code by B and repeat the above procedure

$$\begin{aligned} \text{new code} &= \frac{8}{4} = 2 \\ (\text{new code}).\text{AND.}(B-1) &= 2.\text{AND.}3 \\ &= (000000000010)_2 \\ &= 2 \end{aligned}$$

This implies C is contained in the code 8 and that its exponent is 2, i.e. C<sup>2</sup>. The process stops when the code is reduced to zero.

Each loop set (of any order) contributes to a term in the network function. As each loop set (of any order) is generated and coded, it is compared with existing terms. If a term with the same symbol code and power of s exists, then the constant of the term is updated by adding to it the constant of the new loop set. Otherwise, a new term is created. Note that the above process is an important step towards reducing the storage requirements.

After all loop sets have been found, the transfer function is complete, and it remains only to transform the symbol code of each term into its corresponding symbol set by the .AND. operation previously described.

### II-3. Generating First Order Loops

#### a. General Description

Let the nodes of the SFG be labelled 1,2,...,N. All first order loops which contain node J ( $J=1$  initially) can be found by conceptually splitting node J into two nodes, one node containing all incoming branch and the other containing all outgoing branches, and then enumerating all paths between these two nodes. All branches going into node J are then removed and the process repeated for node J+1. Clearly, this procedure will produce all circuits with no duplications.

The problem of efficiently finding all circuits now becomes one of finding paths. The path-finding algorithm utilized by SNAP is based on a routing technique which conceptually resembles that proposed by Kroft<sup>(5)</sup>. However, because our ultimate objective is a flexible user-oriented program, we have chosen to use FORTRAN instead of SNOBOL as Kroft did. A general description of the concepts contained in the algorithm will be given here in addition to a rigorous step-by-step description presented at the end of this section.

Consider the SFG given in Fig. 4. The topological structure of the SFG can be completely described by the following routing table where the entries in the  $J^{\text{th}}$  row are the set of all nodes of distance one from node J. Note that the entries of each row are made to decrease as the column subscript M increases. This facilitates modifying the table after all paths through a particular node, say node J, have been found because only the right most non-zero entry of each row must be tested, i.e. if that entry equals J, it is a set to zero. As an example in using the routing table, the following two circuits can easily be shown to compose the complete set of circuits containing node 1.

1 - 11 - 9 - 12 - 8 - 5 - 6 - 1

1 - 11 - 9 - 10 - 7 - 2 - 6 - 1

	1	11		
	2	6	3	
	3	2		
	4	5		
	5	6	4	
R(J,M) =	6	5	2	1
	7	2		
	8	5		
	9	12	11	10
	10	9	7	
	11	9		
	12	8		

Routing Table

A particularly important feature of the path-finding algorithm is the method by which each new node generated from the routing table must be tested to prevent loops from being formed. Rather than comparing the prospective node to each node already in the path, it is much more efficient to define the function

$$F(I) = \begin{cases} 1 & \text{if } I \text{ is contained in the path node sequence} \\ 0 & \text{if } I \text{ is not contained in the path node sequence} \end{cases}$$

on which only one logic test need be made.

Additional insight may be obtained by viewing the path-finding technique graphically. That is, the process by which paths are generated can be observed by applying the following two rules directly to the SFG.

- (1) Let node J be the last node added to the path node sequence (initially J = input node). To select the next node, traverse that branch connected to node J that goes to the highest numbered node satisfying both the following requirements:

- a. we did not just back up from this node while applying rule 2, and
- b. this node is not included in the path node sequence.

Repeat this process until the output node is reached (then store the node sequence and go to rule 2) or until no new node can be found that satisfies (a) and (b) (then go to rule 2)

- (2) Back up along the path just found (this is always possible unless we are at the input node in which case all paths have been found) until a new route can be taken according to rule 1.

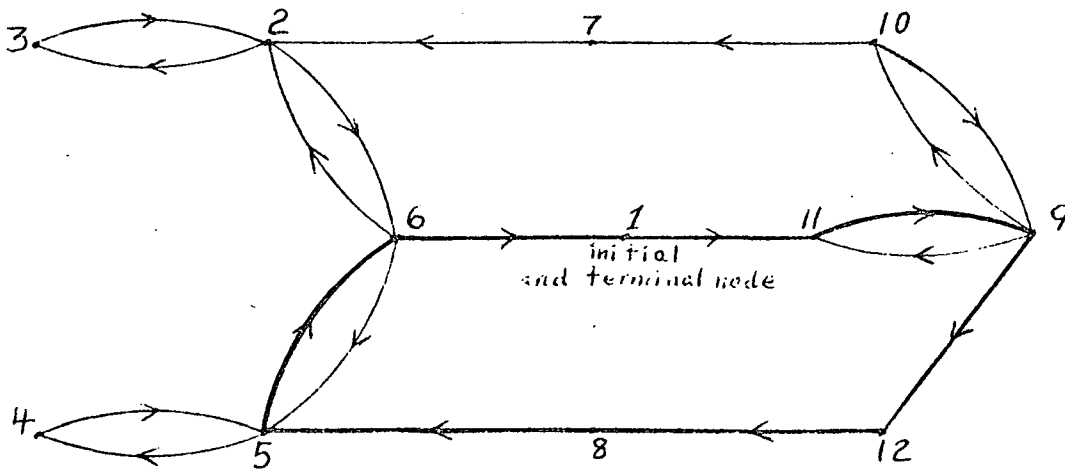


Figure 5

The heavy lines of Fig. 5 show the path which results from applying rule 1 when node 1 is considered both the initial and terminal node. Generating a second path requires backtracking to node 9, then continuing the sequence 10-7-2-6-1. Note that the above graphical technique for listing all paths can be helpful when solving problems by hand.

b. A Detailed Description of the Path-Finding Algorithm <sup>(3)</sup>

Algorithm PF\* (Path-finding): This algorithm finds all paths between two nodes of a directed graph (without parallel edges) whose nodes are labelled 1,2,...,N. The only modification necessary to adapt the algorithm to finding

\*The format used to describe the path-finding algorithm follows the style of Knuth(6).

all circuits thru node L is to set  $I \leftarrow L$  where L and I are defined below.

Notations:

- I: Initial path node
- L: Last path node
- N: Number of nodes in graph
- $E_J$ : Number of branches leaving node J
- $R(J,M)$ : Routing table
- $C_J$ : Column counter for the  $J^{\text{th}}$  row of the table R
- $P(V,W)$ : The  $V^{\text{th}}$  node in the node sequence of path W
- $U_W$ : Number of nodes in path W
- $F(K)$ : A function used to test whether node K is repeated, and whether the last node is reached.

PF1. (Preliminary)

Set  $R(J,1), R(J,2), \dots, R(J,E_J)$  to the group of  $E_J$  nodes of distance one from node J. When using the algorithm to find circuit, made the entries of each row decrease as M increases.

$$\text{Set } R(J,M) \leftarrow \begin{cases} -1 & \text{for } M = E_J + 1 \text{ and } J=I \\ 0 & \text{for } M = E_J + 1 \text{ and } J \neq I \end{cases}$$

$$\text{Set } F(K) \leftarrow \begin{cases} 1 & \text{for } K=I \\ 0 & \text{for } K=J \text{ and } J \neq I, L \\ -1 & \text{for } K=L \end{cases}$$

$$\text{Set } C_J \leftarrow 1 \text{ for } J=1, 2, \dots, N$$

$$\text{Set } W \leftarrow 1, V \leftarrow 2, J \leftarrow I, P(1,1) \leftarrow I$$

PF2. (Find the next node)

$$\text{Set } P(V,W) \leftarrow R(J, C_J)$$

PF3. (Test R)

$$\text{IF } R(J, C_J) \begin{cases} < 0 & \text{stop; all paths have been found} \\ = 0 & \text{set } F(J) \leftarrow 0, \text{ go to step PF6} \\ > 0 & \text{go to step PF4} \end{cases}$$

PF4. (Test F)

IF  $F[R(J, C_J)]$   $\left\{ \begin{array}{l} < 0 \text{ path completed; go to step PF7} \\ = 0 \text{ go to step PF5} \\ > 0 \text{ set } C_J \leftarrow C_J + 1; \text{ go to step PF2} \end{array} \right.$

PF5. (Prepare for next node)

Set  $J \leftarrow P(V, W)$ ,  $F(J) \leftarrow 1$ ,  $V \leftarrow V + 1$ , go to step PF2

PF6. (Back step)

Set  $C_J \leftarrow 1$ ,  $J \leftarrow P(V - 2, W)$ ,  $C_J \leftarrow C_J + 1$ ,  $V \leftarrow V - 1$ , go to step PF2

PF7. (Finish path)

Set  $C_J \leftarrow C_J + 1$ ,  $P(K, W + 1) \leftarrow P(K, W)$ ,  $K = 1, 2, \dots, U_W - 1$ ,  $W \leftarrow W + 1$ , go to step PF2.

II-4. Generating Nontouching Loops of Order Two or More\*

Preliminary results from SNAP indicate that of the following subprograms, (1) finding a SFG, (2) coding and de-coding, (3) enumerating first order loops, and (4) finding all higher order nontouching loops, the last will generally require the most time unless the network contains many distinct symbols in which case subprogram (2) may dominate. It is therefore necessary to exercise considerable care in developing an algorithm for finding all orders of nontouching loops.

In general, to find loop sets of all orders, some comparison between the node sequences of the different loops must be made. A brute force technique is simply to store all the node sequences of the first order loops and to find nontouching loops by direct comparison of the nodes contained in the loops. Of course, storage is also needed to indicate the loops contained in some of the higher order combinations, but this storage is necessary even in the more efficient techniques which follow.

The above method is improved considerably if instead of directly comparing the nodes of loop A and loop B to determine if they touch, a function F(I) is defined as

$$F(I) = \begin{cases} 1 & I \in \{\text{nodes in loop A}\} \\ 0 & \text{otherwise} \end{cases}$$

and then tested as follows:

$$\text{If } F(J) \begin{cases} =0 & \text{all } J \in \{\text{nodes in loop B}\} \Rightarrow \text{loops do not touch} \\ =1 & \text{any } J \in \{\text{nodes in loop B}\} \Rightarrow \text{loops touch} \end{cases}$$

For those computers which can accommodate the .AND. operation (or equivalent), the following coding technique reduces the number of logic

---

\* Although the program was correct, the algorithm was incorrectly described in reference (3).



comparisons needed to determine if two loops touch to one and, perhaps what is even more important, requires only a single code be stored for each first order loop instead of the complete node sequence. As each first order loop is generated, it is assigned an integer code whose binary representation shows the set of nodes in the loop. For example, if loop A contains the nodes {11, 9, 12, 8, 5, 6, 1} and loop B contains the nodes {2, 6}, then the codes are

$$A = (110110110001)_2 = 3505$$

$$B = (000000100010)_2 = 34$$

To determine whether the two loops touch or not, the masking operation .AND. is used. Thus,

$$(A) \text{ .AND. } (B) = (000000100000)_2 \neq 0$$

The result is not zero, indicating that loops A and B touch.

Using the coding scheme the complete set of nontouching pairs of loops is found and stored in the one dimension array N. Let n = number of first order of loops. Then

$$\{N(1), N(2), \dots, N[P(1)], N[P(1)+1], \dots, N[P(2)], N[P(2)+1], \dots, N[P(3)], \\ N[P(3)+1], \dots, N[P(n-1)], N[P(n-1)+1], \dots, N[P(n)]\}$$

is the complete set of nontouching pairs of loops, where the

$$\begin{aligned} \{N(1), N(2), \dots, N[P(1)]\} &= \text{set of loops numbered higher than 1 which do not} \\ &\quad \text{touch loop 1.} \\ \{N[P(1)+1], N[P(1)+2], \dots, N[P(2)]\} &= \text{set of loops numbered higher than 2} \\ &\quad \text{which do not touch loop 2.} \\ \vdots & \\ \{N[P(i-1)+1], N[P(i-1)+2], \dots, N[P(i)]\} &= \text{set of loops numbered higher than } i \\ &\quad \text{which do not touch loop } i \\ \vdots & \\ \{N[P(n-1)+1], N[P(n-1)+2], \dots, N[P(n)]\} &= \text{empty set because there are no} \\ &\quad \text{loops numbered higher than } n. \end{aligned}$$

Note that the array P is simply used to partition the array N such that the set

$$\{N[P(i-1)+1], N[P(i-1)+2], \dots, N[P(i)]\} \text{ does not touch loop } i.$$

Example: (Consider the SFG of Fig. 4)

The first order loops are

loop	node sequence
1	1-11-9-12-8-5-6-1
2	1-11-9-10-7-2-6-1
3	2-6-2
4	2-3-2
5	4-5-4
6	5-6-5
7	9-11-9
8	9-10-9

To find the array of nontouching pairs P, SNAP codes the above loops and proceeds to use the .AND. operator. The results are

$$N = \{4, 5, 5, 7, 8, 5, 6, 7, 8, 7, 8, 7, 8\}$$

and  $P(1)=1, P(2)=2, P(3)=5, P(4)=9, P(5)=11, P(6)=13, P(7)=13, P(8)=13.$

To systematically continue the process, an array S is created from which all higher order loop sets (2 or more) not touching loop  $l$  can be found. By incrementing  $l$  from 1 to n, all higher order loops will then be enumerated.

Let

$$S = \begin{matrix} S(1,1)S(1,2)\dots S[1,U(1)] \dots & S[1,J(1)],0,\dots \\ S(2,1)S(2,2)\dots & S[2,U(2)+1]\dots S[2,J(2)],0,\dots \\ \vdots & \vdots \\ S(K,1)S(K,2)\dots S[K,U(K)] \dots & S[K,J(K)],0,\dots \\ \vdots & \vdots \end{matrix}$$

where

$$\begin{aligned} S(1,1) &= N[P(L-1)+1] \\ S(1,2) &= N[P(L-1)+2] \\ &\vdots \\ S[1,J(1)] &= N[P(L)] \end{aligned}$$

and where the entires (loop numbers) of row  $M (M \geq 2)$  are those loops in the set

$$\{S[M-1, U(M-1)+1], S[M-1, U(M-1)+2], \dots, S[M-1, J(M-1)]\}$$

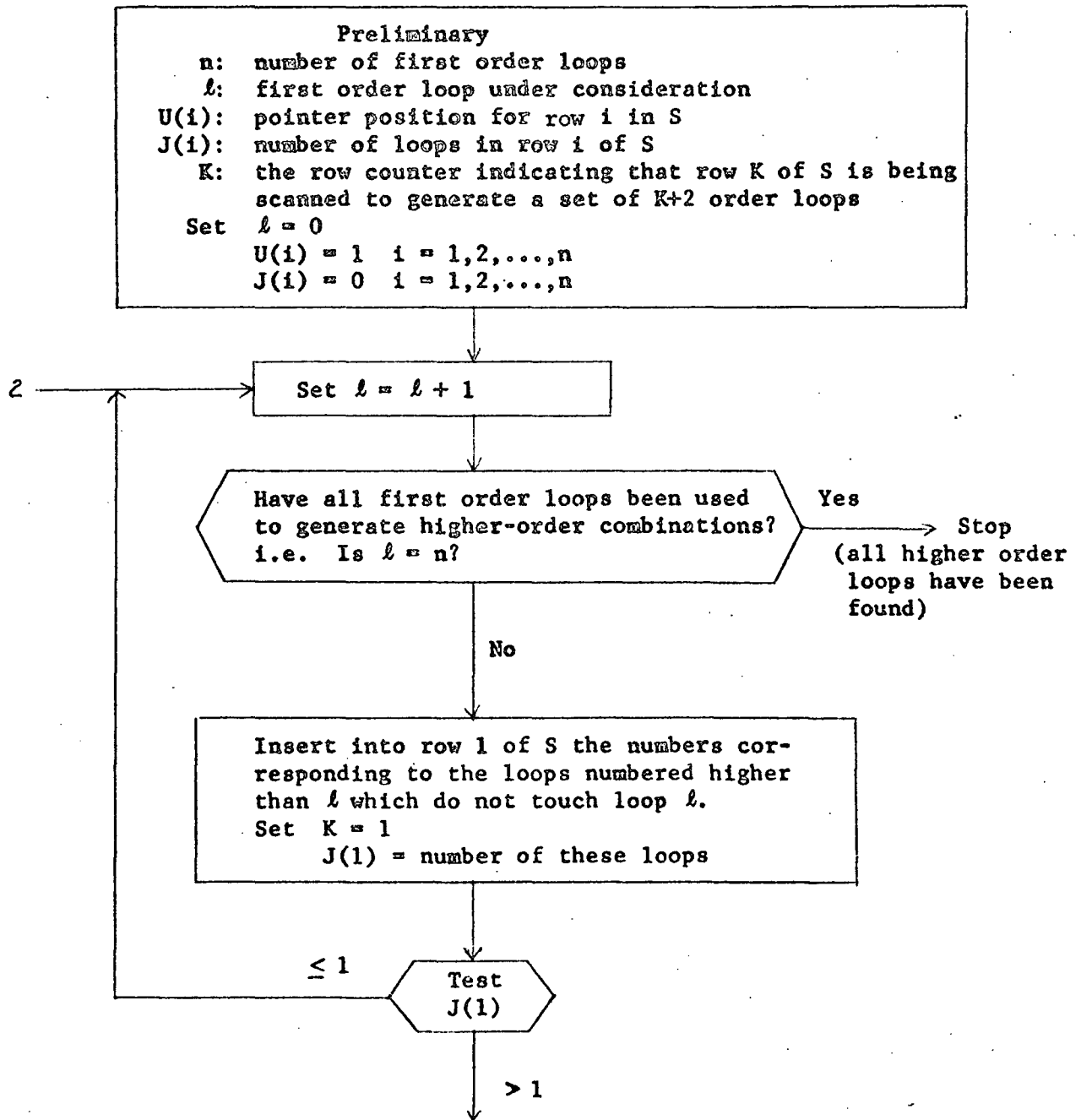
which do not touch the loop  $S[M-1, U(M-1)]$ .

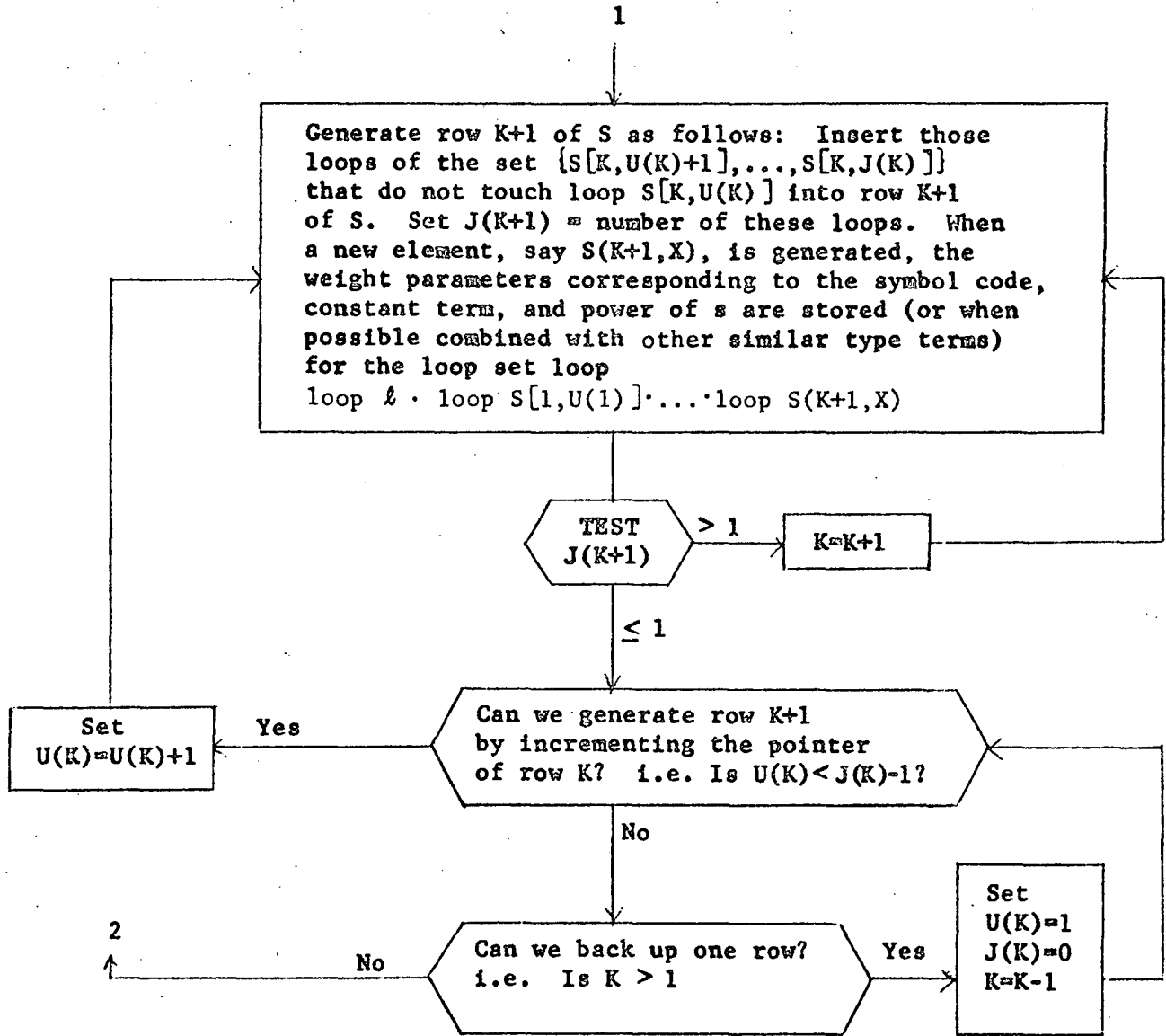
The arrows shown in the array  $S$  given above are referred to as "pointers".

Note that  $U(J)$  indicates the position of "pointer" of the  $J^{\text{th}}$  row. Example:

$U(3) = 5$  means the pointer of row 3 is currently located at the  $5^{\text{th}}$  column.

The procedure for finding all higher order loop combinations is given in the following flow chart:





Example:

From the preceding example,

$$N = \{4, 5, 5, 7, 8, 5, 6, 7, 8, 7, 8, 7, 8\}$$

and  $P(1) = 1, P(2) = 2, P(3) = 5, P(4) = 9, P(5) = 11$

$$P(6) = 13, P(7) = 13, P(8) = 13$$

Arrays N & P are more easily interpreted by setting up the following table:

Table 3.

loop J	loops numbered higher than J that do not touch loop J
1	4
2	5
3	5, 7, 8
4	5, 6, 7, 8
5	7, 8
6	7, 8
7	-
8	-

The sequence for producing the higher order loops is as follows:

loop 1

loops not touching loop 1 are inserted into first row of S (see Table 3)

S array

$$\begin{matrix} \downarrow \\ \left[ \begin{array}{cccc} 4 & 0 & \dots & \end{array} \right] \\ 0 \\ \vdots \\ \vdots \end{matrix}$$

Output

no 3<sup>rd</sup> order loops

loop 2

S array

$$\begin{matrix} \downarrow \\ \left[ \begin{array}{cccc} 5 & 0 & \dots & \end{array} \right] \\ 0 \\ \vdots \\ \vdots \end{matrix}$$

Output

no 3<sup>rd</sup> order loops

loop 3

S array

↓	5	7	8	0
	0			
	⋮			
	⋮			

loop 5 does not touch loop 7 or loop 8 (this is determined by comparing loop codes--see section II-4)

↓	5	7	8	0
↓	7	8	0	0
	0	0	0	0
	0	0	0	0

Output

loop 3·loop 5·loop 7  
loop 3·loop 5·loop 8

loop 7 touches loop 8; thus, there is no 3<sup>rd</sup> row. Further if the pointer of row 1 is incremented by 1, no new 2<sup>nd</sup> row can be created. Thus, we are done with loop 3.

loop 4

S array

↓	5	6	7	8
	0	0	0	0
	⋮			
	⋮			

loop 5 does not touch loop 7 or loop 8

↓	5	6	7	8
↓	7	8	0	0
	0	0	0	0
	0	0	0	0

Output

loop 4·loop 5·loop 7  
loop 4·loop 5·loop 8

loop 7 touches loop 8; thus, there is no 3<sup>rd</sup> row. Increment pointer of row 1.

$$\begin{bmatrix} 5 & 6 & 7 & 8 \\ 0 & 0 & 0 & 0 \\ \vdots & & & \end{bmatrix}$$

↓ loop 6 does not touch loop 7 or loop 8

$$\begin{bmatrix} 5 & 6 & 7 & 8 \\ \downarrow & & & \\ 7 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

loop 4·loop 6·loop 7  
loop 4·loop 6·loop 8

↓ incrementing pointers give no additional third order loops

loop 5

S array

$$\begin{bmatrix} \downarrow \\ 7 & 8 & 0 & \dots \\ 0 \\ \vdots \end{bmatrix}$$

Output

no 3<sup>rd</sup> order loops

↓ loop 7 touches loop 8

loop 6

S array

$$\begin{bmatrix} \downarrow \\ 7 & 8 & 0 & 0 \\ 0 \\ \vdots \end{bmatrix}$$

Output

no 3<sup>rd</sup> order loops

↓ loop 7 touches loop 8

loop 7

S array

[0]

Output

no 3<sup>rd</sup> order loops

loop 8

S array

[0]

Output

no 3<sup>rd</sup> order loops

### III. USER'S GUIDE

#### III-1. Information Needed by User

Program: SNAP (Symbolic Network Analysis Program)

Purpose: To obtain the network functions\*  $\frac{V_{out}}{V_{in}}$ ,  $\frac{V_{out}}{I_{in}}$ ,  $\frac{I_{out}}{V_{in}}$ , or  $\frac{I_{out}}{I_{in}}$  as a ratio of two polynomials of the following type:

- (1) all network element values are represented by symbols (the symbols need not all be different),

Examples: 
$$\frac{V_{out}}{V_{in}} = \frac{s^2 LRC}{s^2 2LRC + s(L+R^2C) + R}$$

$$\frac{V_{out}}{V_{in}} = \frac{ZYR}{2ZYR + Z + R^2Y + R}$$

- (2) some element values are specified numerically, some symbolically,

Example: 
$$\frac{V_{out}}{V_{in}} = \frac{s^2 R}{s^2 2R + s(.5 \times 10^6 + 150R^2) + .75 \times 10^8 R}$$

- (3) all element values are given numerically,

Example: 
$$\frac{V_{out}}{V_{in}} = \frac{s^2}{2s^2 + 2 \times 10^4 s + .75 \times 10^8}$$

Description: Program SNAP is designed to handle lumped, linear, time invariant networks\*\* containing the following type components:

- (1) two-terminal circuit elements -- resistance, inductance, and capacitance.
- (2) two-terminal networks described by an admittance or impedance parameter.

---

\* Refer to Appendix A at end of this section for a technique of handling multi-output functions.

\*\* See Appendix B for a brief list of additional limitations on the size and type of network allowed.



- (3) all four types of controlled sources (Note: Mutual inductance, ideal transformers, gyrators, etc., can be modeled with elements in (1) and (3))
- (4) one independent source; see Appendix A for a technique of handling multi-input networks.

Network Data Required: After the network components have been modeled by the type elements allowed, the branches and nodes are to be numbered consecutively starting with 1 and reference directions for each branch current are to be chosen. The following gives the sequence of data cards needed to describe the network.

CARD 1

Columns	Contents
1-72	Title card (all 72 columns are reproduced in output)

CARD 2

Columns	Contents
1-5 (right adjusted)	Number of nodes in the network
6-10 (right adjusted)	Number of branches in the network

The following three entries are optional.

11-15 (right adjusted)	Number base of symbol codes (automatically set to 8 if left blank)
21	1 if a description of the SFG is to be listed, blank otherwise
22	1 if all loops (circuits) in the SFG are to be listed (node sequence), blank otherwise

CARD 3

Columns	Contents
1-5 ( right adjusted )	Network branch number of source
6-10 ( right adjusted )	Network branch number associated with output (leave blank if output is a voltage across more than one branch)
11-15 ( right adjusted )	Node number corresponding to the positive output voltage terminal (these columns can be left blank if columns 6-10 are <u>not</u> blank)
16-20 ( right adjusted )	Node number corresponding to the negative output voltage terminal (these columns can be left blank if columns 6-10 are <u>not</u> blank)

CARDS 4 thru (b+3)

(b = number of network branches)

Note 1: Each card describes one network branch (element).

Note 2: If output is a voltage (current) associated with a particular branch, then the data card describing this branch should be entered first (last) among the branch data cards (cards 4 thru (b+3)) to insure that this branch will be chosen as part of the tree (cotree).

Note 3: When a large number of branches share one common terminal, it is better to place these branches first starting with card 4 (card 5 if note 2 applies). The reason is given in Appendix C at the end of this section.

Columns	Contents
1-2 ( left adjusted )	Element type; E: voltage source I: current source G: conductance R: resistance L: inductance C: capacitance Z: impedance Y: admittance CC: current controlled current source CV: current controlled voltage source VC: voltage controlled current source VV: voltage controlled voltage source

Continued

Columns

Contents

3-5 ( right adjusted )	Element number--all elements of the network must be assigned a distinct number (positive integer). For greatest efficiency, the numbering should be consecutive.
6-10 ( right adjusted )	Initial node--this is relative to the arbitrarily chosen current direction.
11-15 ( right adjusted )	Terminal node--this is relative to the arbitrarily chosen current direction.
17-19 ( right adjusted )	Element symbol--the element's value, if not specified, is represented by this symbol.
20	Equal sign (=) if element is to be assigned a value. Leave blank if element value is to be represented in symbolic form.
21-32 ( right adjusted )	Element value (if known)--Format is E12.5. Units should be compatible with element type as specified in columns 1-2; for example, R is expressed in ohms, G in mhos.
33-35 ( right adjusted )	If element is a dependent source, enter the element number of its control.

An Example: We wish to find  $I_{out}/V_{in}$ , Keeping A1, A2, and C as symbols.

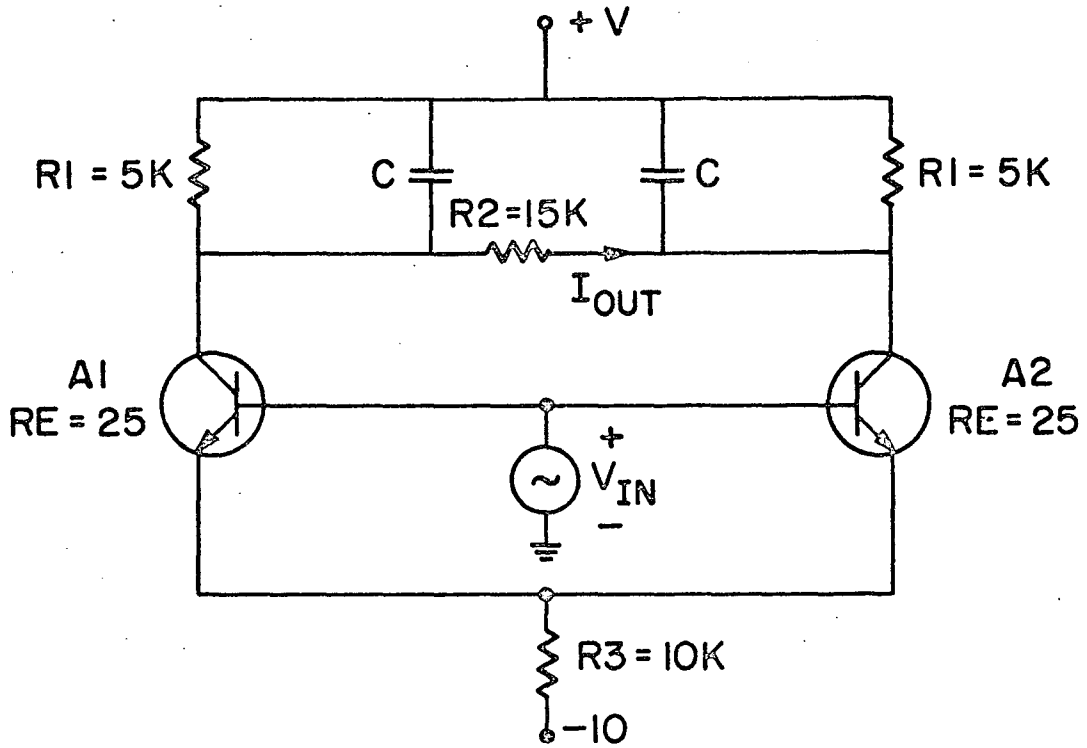


FIGURE 1. ORIGINAL NETWORK.

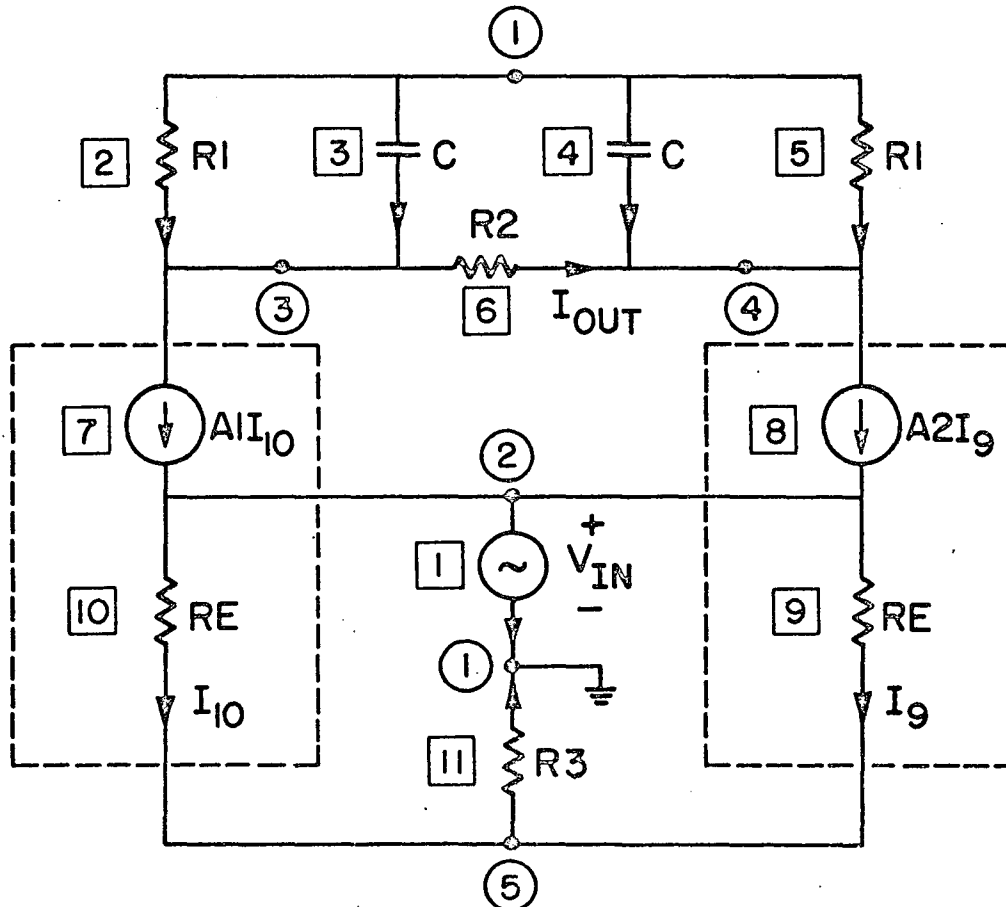


FIGURE 2. MODELED NETWORK.



TABLE 2. Program Output Information Showing Signal-Flow Graph, Circuits, and Execution Times.

SFG						
INITIAL NODE	TERMINAL NODE	EXPONENT OF S	BRANCH VALUE	BRANCH SYMBOL	1 IF SYMBOL IS INVERTED	1 IF SYMBOL IS USED
6	1	0	-1.00000E+00	FB	0	1
2	3	1	1.00000E+00	C	0	1
3	2	0	-5.00000E+03	R1	0	0
5	4	1	1.00000E+00	C	0	1
4	5	0	-5.00000E+03	R1	0	0
2	6	0	-6.66667E-05	R2	1	0
6	2	0	5.00000E+03	R1	0	0
5	6	0	6.66667E-05	R2	1	0
6	5	0	-5.00000E+03	R1	0	0
10	7	0	1.00000E+00	A1	0	1
7	2	0	5.00000E+03	R1	0	0
9	12	0	4.00000E-02	RE	1	0
12	8	0	1.00000E+00	A2	0	1
8	5	0	5.00000E+03	R1	0	0
9	10	0	4.00000E-02	RE	1	0
10	9	0	-2.50000E+01	RE	0	0
9	11	0	-1.00000E-04	R3	1	0
11	9	0	2.50000E+01	RE	0	0
1	11	0	1.00000E-04	R3	1	0

TIME FOR FORMULATING SIGNAL FLOW GRAPH IN SECONDS .252

CIRCUITS

NO.	NODE LIST
1	1 11 9 12 8 5 6 1
2	1 11 9 10 7 2 6 1
3	2 6 2
4	2 3 2
5	4 5 4
6	5 6 5
7	9 11 9
8	9 10 9

TIME FOR FINDING 8 FIRST ORDER LOOPS IN SECONDS .046

TIME FOR FINDING 19 SETS OF NONTOUCHING LOOPS, IN SECONDS .027

TIME FOR DECODING SYMBOLS IN SECONDS .124

TABLE 3. Network Transfer Function and Total Execution Time.

\*\*\*\*\*

NUMERATOR POLYNOMIAL

= (3.33333 x 10^-5 + .166667 s C)(A2 - A1)

COLUMN	SYMBOL FOR GIVEN COLUMN			
1	A2	/	1	
2	A1	/	1	
3	C	A2	/	1
4	C	A1	/	1

POWER OF S

CONSTANT COEFS. IN THE POLYNOMIAL

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4
0	3.33333E-05	-3.33333E-05	0.	0.
1	0.	0.	1.66667E-01	-1.66667E-01
2	0.	0.	0.	0.

\*\*\*\*\*

DENOMINATOR POLYNOMIAL

= 3.3375 + 2.67 x 10^4 s C + 5.00625 x 10^7 s^2 C^2

COLUMN	SYMBOL FOR GIVEN COLUMN			
1	1	/	1	
2	C	/	1	
3	C**2	/	1	

POWER OF S

CONSTANT COEFS. IN THE POLYNOMIAL

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4
0	3.33750E+00	0.	0.	
1	0.	2.67000E+04	0.	
2	0.	0.	5.00625E+07	

EXECUTION TIME IN SECONDS, .509

AUGUST 1970 VERSION OF SNAP

APPENDIX A

A Sorting Technique for Handling Multi-Input, Multi-Output Networks.

Multi-Inputs

Program SNAP (the August 1970 revision) permits only one independent source branch. However, networks containing more than one source can easily be handled with the following technique. Let  $W_i$   $i = 1, 2, \dots, n$  represent a set of  $n$  independent sources, either voltage or current. Assign  $W_1$  as the permitted independent source and make  $W_2, W_3, \dots, W_n$  dependent sources which are dependent on  $W_1$  with proportionality factors

$$k_2 = \frac{W_2}{W_1}, k_3 = \frac{W_3}{W_1}, \dots, k_n = \frac{W_n}{W_1} \text{ respectively}$$

Only the numerator polynomial in the output will contain these parameters thus permitting the user to easily put the output function into the form

$$\frac{W_{out}}{W_1} = \frac{P_1 + P_2 k_2 + P_3 k_3 + \dots + P_n k_n}{\Delta}$$

where  $\Delta$  and  $P_i$   $i = 1, 2, \dots, n$  are polynomials. The output function can then be written

$$W_{out} = \frac{P_1 W_1 + P_2 W_2 + \dots + P_n W_n}{\Delta} \tag{1}$$

Although at present SNAP does not give the output function in the form of Eq. (1) directly, only a few program modifications are necessary to effect such a result. For example, the program could internally create a new input node,  $I_{new}$ , of the SFG and then make each independent source,  $W_i$ , dependent on  $I_{new}$  with weight  $P_i$  as shown in Fig. 1 below.



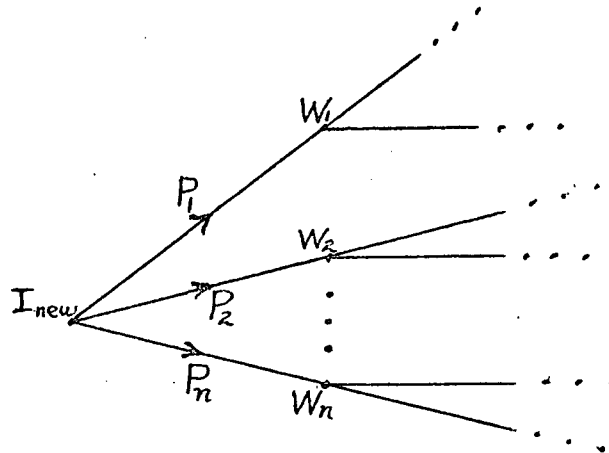


Figure 1

Multi-Outputs

The following technique can be used to obtain more than one output function in a single computer run: Augment the original network by appending one end of a series connection of dependent voltage sources to the given network such that

- (a) to each branch current,  $I_j$ , desired as an output, there corresponds a dependent voltage source which depends on  $I_j$  and has symbolic weight  $I_{oj}$ ,
- and (b) to each voltage  $V_{OAB}$  desired as an output, there corresponds a set of the dependent voltage sources each dependent upon a voltage across one of the branches in the path between nodes A and B and all having symbolic weight  $V_{OAB}$ .

By specifying the output to be the voltage across the entire series connection of dependent voltage sources, outputs  $I_j$  and  $V_{AB}$  will be those output terms which contain  $I_{oj}$  and  $V_{OAB}$  respectively. Only a few modifications of the present version of SNAP would be necessary to have the program internally perform the network augmentation described above (at present, the user must do the augmenting).

As an example, Fig. 2 illustrates the network augmentation needed to find the voltage  $V_{14}$  and current  $I_5$  for the given bridge network in one computer run.

$$V_{47} = V_{014}(V_{12} + V_{24}) + I_{05} I_5$$

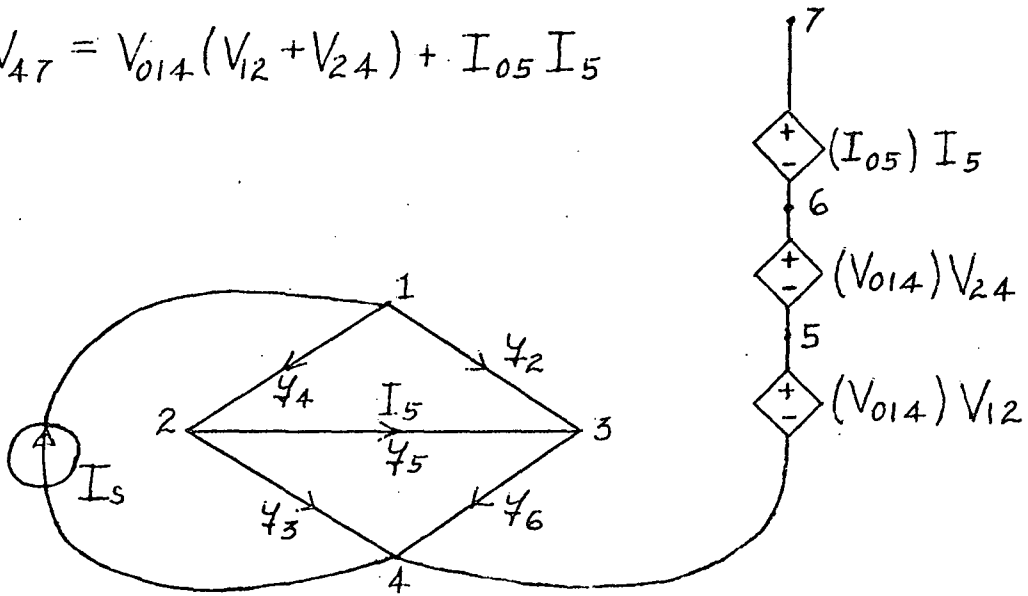


Figure 2.

Appendix B

A Brief List of Limitations on the Size and Type of Network Allowed

	Version: August, 1970	Version:	Remarks (on August 1970 version)
Number of network branches	35		SNAP cannot handle all networks having 35 branches or less. Other factors such as time considerations, SFG characteristics (number of higher order loops, for example), and number of network symbols to name a few can further limit the size of the network.
Maximum number of elements that can be represented by the same symbol	7		This number can be increased to $2^n - 1$ by increasing the symbol code base used to $2^n$ , $n > 3$ , on the input data card 2.
Number of different powers of s	15		Sufficient for networks containing no more than 15 reactive elements.
Estimate of the maximum number of distinct network symbols permitted	12		This restriction results from the fact that SNAP can contain no more than 150 different symbol combinations in the output.

APPENDIX C

Selecting a "Good" Tree

The network tree used to generate the SFG has a very significant effect on the number of loops and higher order loops present in the SFG. The loop enumeration and evaluation, in turn, often determines the time and storage needed by a computer to solve a given network. The ladder network of Figure 1 together with Table 1 illustrated the interrelationship between the tree selected, the number of loops (all orders), computer execution time, and computer storage.

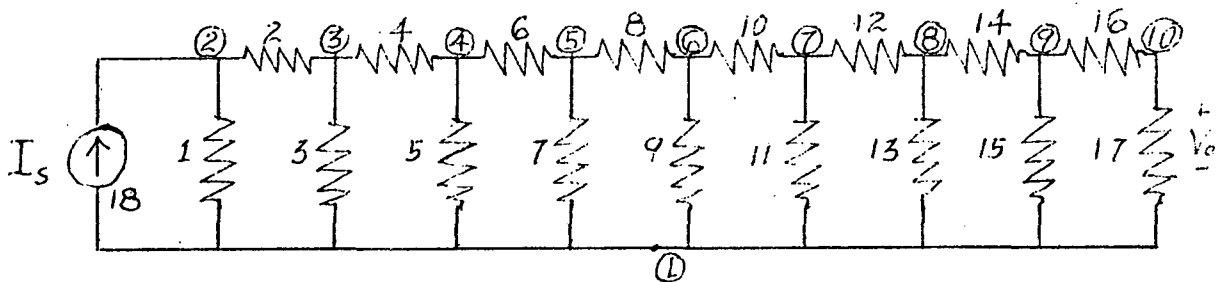


Figure 1

Table 1

Tree Branches	Number of loops	Number of higher order loops	Time required to find $V_o/I_s$
Star tree: 1,3,5,7,9,11,13,15,17	17	2567	1.55 seconds
1,2,4,7,9,11,13,15,17	38	8096	3.93 seconds
1,2,4,6,9,11,13,15,17	117	19719	9.42 seconds
1,2,4,6,8,11,13,15,17	476	- (storage for first order exceeded)	--

Unfortunately, choosing the "best" tree, that is, a tree which will minimize the number of loop combinations of all orders is a very involved process. See reference 2 and 7 for a detailed discussion of this problem.

For most networks, however, a tree that will result in a reasonable amount of execution time and computer storage can be selected by applying one of the following rules (rule 2 results in a better tree than rule 1)

Rule 1: Select a tree in which as many branches as possible form a star, that is, the branches share a common node. Modify this tree, if necessary, to include any branch which has two or more branches in parallel with it.

Rule 2: Let  $T_k$  be some tree (not necessarily the best) of the network graph. For each link  $\ell_i$  of the graph, define  $B_{\ell_i}$  as the number of tree branches which form a circuit with  $\ell_i$ . Then form the sum

$$S_{T_k} = \sum_{i=1}^L B_{\ell_i} \quad \text{where } L = \text{number of links in the graph having } T_k \text{ as a tree.}$$

Select that tree, say  $T_j$ , which satisfies the inequality

$$S_{T_j} < S_{T_k} \quad k = 1, 2, \dots, N \text{ where } N = \text{number of trees}$$

The example given in section III-1 uses a tree,  $T_j$ , having  $S_{T_j} = 11$ .

The tree generated internally by program SNAP includes all voltage sources together with those passive branches read in first (starting with input data card 4) which complete the tree. Thus, to have SNAP select the tree that has been chosen by the user, it is necessary that the user's tree include all voltage sources and that all its passive branches be listed first starting with input data card 4.

### III-2. Modifying the Dimension of Arrays

In order to make SNAP applicable to many different type networks, a flexible yet simple procedure is needed for modifying the dimension of the arrays. For example, storage requirements for networks containing many symbols will be determined by the number of symbols, symbol codes, etc., whereas the storage needed for networks having no symbols will be determined by the number of loops, nontouching loops of all orders, and related network characteristics. Because it is not possible to determine apriori reasonable bounds for all the network characteristics, error diagnostics have been built into the program to inform the user as to which arrays have been inadequately dimensioned. As a result, the technique for adjusting the array dimension, in SNAP can be outlined as follows:

- (1) Check that those network characteristics which can be determined before running the program are within the specified limits. These limits are listed following the dimension statements of the main program for convenient reference.
- (2) Run the program. If an array dimension is exceeded an error message will result which specifies the network characteristic involved. For example, if the SFG of a given network has an excessive number of circuits, the message "No. of circuits exceeds limit--increase dimensions containing NPAC" will result. The definition of NPAC (number of paths and circuits) are found immediately following the array dimensions in the main program. It is important to point out that a computer run may continue to completion even if the dimension of some arrays have been exceeded (an error message is still given, however). In this situation, the results cannot be considered reliable.

(3) Once it has been ascertained by (1) and (2) that dimension modifications are in order, refer to the next few pages to determine the arrays associated with the network characteristics of interest. Increase the dimension of all the arrays indicated by say 20% (several runs may be necessary to achieve adequate program dimensions). Then update the value of the parameter (NPAC, for example) corresponding to the network characteristic involved. This parameter is used throughout the program (as limits on DO loops etc.) thereby making it unnecessary to do any additional program modifications.

NBN = Number of Network Branches (Presently 35)

PROGRAM MAIN

IG(NBN),           KODES(NBN),   KODE(NBN,NBN)  
SMBOL(NBN),       KONC(NBN),   IXPO(NBN,NBN)  
IFLOW(NBN),       N(NBN,NBN),   CONS(NBN,NBN)  
LT(NBN),           NP(NBN),

SUBROUTINE SFG

JROW(NBN),        TYPB(NBN),    IQUALX(NBN),    JBX(NBN)  
NP(NBN),           JB(NBN),        VALX(NBN),       LBX(NBN)  
IVV(NBN),          LB(NBN),        NUMLX(NBN),     IB(NBN,NBN)  
NUML(NBN),         MSYM(NBN),    INTRE(NBN),     NS(NBN,NBN)  
ICV(NBN),          IQUAL(NBN),   NOTREE(NBN),    NF(NBN,NBN)  
INTREE(NBN),       VAL(NBN),     TYPX(NBN),  
LINC(NBN),         SYM(NBN),     NUMX(NBN),

SUBROUTINE FTREE

TYPX(NBN),        INTRE(NBN),   NF(NBN,NBN)  
JBX(NBN),          NOTREE(NBN),  
LBX(NBN),          NP(NBN)

SUBROUTINE TREP

JX(NBN),           JMEM(NBN),    NF(NBN,NBN)  
NP(NBN),           KMEM(NBN)

NBG = Number of Branches in SFG (Presently 100)

PROGRAM MAIN

NFIRST(NBG), SYMBUL(NBG), NEST(NBG)  
NLAST(NBG), MIX(NBG), TYPE(NBG)  
IXPON(NBG), CVAL(NBG),  
WEIGHT(NBG), KONSO(NBG),

SUBROUTINE SFG

NFIRST(NBG), MAPY(NBG), SYMBUL(NBG)  
NLAST(NBG), KONSO(NBG), MIX(NBG)  
IXPON(NBG), NEST(NBG), CVAL(NBG)  
WEIGHT(NBG), TYPE(NBG),

NPAC = Number of Paths Plus Circuits (Presently 300)

PROGRAM MAIN

CONST(NPAC), MAPO(NPAC), JAC(NPAC)  
KODET(NPAC), NOCTOT(NPAC), NPCODE(NPAC)  
IXPOT(NPAC), NUP(NPAC),



NTO = Number of Terms in Output (Presently 150)

PROGRAM MAIN

NA(NTO),        POLYU(NEXPS,NTO),        SEMPON(NTO,NSPT/2)  
NB(NTO),        POLY(NEXPS,NTO),        SEMPOD(NTO,NSPT/2)  
KSORT(NTO),     SIMBON(NTO,NSPT/2),  
ITOP(NTO),       SIMBOD(NTO,NSPT/2)

SUBROUTINE ARRAY

KSORT(NTO),     POLY(NEXPS,NTO)

SUBROUTINE DECODE

ITOP(NTO)

NSPT = Number of Symbols per term in Output (Presently 20)

PROGRAM MAIN

KONS(NSPT),     KODF(NSPT),        SEMPON(NTO,NSPT/2)  
KODI(NSPT),     SIMBON(NTO,NSPT/2),   SEMPOD(NTO,NSPT/2)  
SEMBOL(NSPT),   SIMBOLD(NTO,NSPT/2),

SUBROUTINE FTREE

KCOL(NSPT)

SUBROUTINE DECODE

SEMBOL(NSPT),   KODF(NSPT),        KODI(NSPT)

NEXPS = Number of Different Powers of s (Presently 15)

PROGRAM MAIN

MSORT(NEXPS),POLYU(NEXPS,NTO),POLY(NEXPS,NTO)

SUBROUTINE ARRAY

MSORT(NEXPS),POLY(NEXPS,NTO)

NRI = Maximum Number of Nontouching Loops (Presently 15)

PROGRAM MAIN

ISET(NRI,NCI)

NCI = Maximum Number of Loops Not Touching any Given Loop (Presently 100)

PROGRAM MAIN

ISET(NRI,NCI)

NEON = Number of Nontouching Pairs of Loops (Presently 1200)

PROGRAM MAIN

NOTCH(NEON)

NRS = Number of Repeated Symbols (Presently 9)

PROGRAM MAIN

STAR(NRS)

#### IV. PROGRAMMER'S GUIDE

##### IV-1. Definitions

CONS(J,L)=WEIGT(I) FOR BRANCH I OF THE SFG WHERE  
J= FIRST(I), L=NLAST(I)  
CONST(I)=COMPOSITE CONSTANT ASSOCIATED WITH CIRCUIT I. IT IS FOUND BY  
TAKING THE PRODUCT OF THE CONSTANT VALUES OF EVERY SFG BRANCH  
IN CIRCUIT I  
CVAL(NUMC)=VALX(LINK) WHERE NUMC=NUMX(LINK)  
(USED ONLY FOR NETWORK BRANCHES NOT IN THE TREE)  
IR(LF,JF)=IB(JF,LF)=NUMC WHERE JF=JB(NUMC) AND LF=LB(NUMC)  
AND NUMC IS A NETWORK TREE BRANCH NUMBER (ASSIGNED BY USER)  
IFLOW(K)=A FLAG, FOR THE PURPOSE OF CHECKING WHETHER NODE K IS REPEATED  
AND WHETHER THE LAST NODE IS REACHED  
IG(L)=SYMBOL CODE ASSIGNED TO THE SFG BRANCHES HAVING  
TERMINAL NODE L  
INTRE(K)=I. THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST IS  
CHOSEN AS THE K-TH BRANCH OF THE NETWORK TREE  
INTRFE(NUMC)=1 IF THE NETWORK BRANCH NUMBERED NUMC BY THE USER IS  
SELECTED FOR THE TREE, 0 OTHERWISE  
IQUAL(NUMC)=IQUALX(I) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)  
IQUALX(I)=EQUAL SIGN(=) IF I-TH NETWORK BRANCH IN THE DATA BRANCH  
LIST HAS A NUMERICAL VALUE. LEFT BLANK IF I-TH BRANCH IS  
TO BE REPRESENTED BY A SYMBOL  
ISET(J,I)=THE INTEGER ARRAY WHICH TOGETHER WITH THE ARRAY NOTCH CAN  
BE USED TO FIND ALL SETS OF NONTOUCHING LOOPS OF ORDER GREATER  
THAN 2  
ITOP(JC)=1 IF THE TERMS IN COLUMN JC OF THE ARRAY POLY BELONG TO  
THE NUMERATOR OF THE OUTPUT TRANSFER FUNCTION, 0 IF THEY  
BELONG TO THE DENOMINATOR  
IVV(M)=NETWORK BRANCH NUMBER OF THE M-TH VOLTAGE CONTROLLED  
VOLTAGE SOURCE IN THE DATA BRANCH LIST  
IXPO(J,L)=IXPON(I) FOR BRANCH I OF THE SFG WHERE  
J=NFIRST(I), L=NLAST(I)  
IXPON(I)=EXPONENT OF S ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I  
IXPOT(I)=COMPOSITE EXPONENT OF S FOR CIRCUIT I. IT IS FOUND BY ADDING  
THE S POWERS ASSOCIATED WITH EACH BRANCH IN CIRCUIT I  
JAC(J)=NUMBER OF NONZERO ENTRIES IN ROW J OF ISET  
JB(NUMC)=JBX(I) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)

JRX(I)=INITIAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST

JMFM(I)=THE ROW OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN

JROW(LF)=THE NUMBER OF NON-ZERO ENTRIES IN ROW LF OF THE ARRAY NF

JX(I+1)=NP(I)

KBASIS=NUMBER BASE OF THE SYMBOL CODES. THAT IS, THE SFG CONTAINS K00 DISTINCT SYMBOLS, SEMBOL(K), K=1,2,...,K00, NOT INCLUDING THE LAP VARIABLE S, WHERE SEMBOL(K) IS ASSIGNED THE CODE KBASIS\*\*K

KHOL=COUNTER USED TO FIND THE NUMBER OF LOOPS OF ORDER 2 OR GREATER

KIK=A ROW COUNTER OF THE MATRIX POLY

KMEM(I)=THE COLUMN OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN

KODE(J,L)=CODE REPRESENTING THE SYMBOL OF THE SFG BRANCH HAVING J AS AN INITIAL NODE AND L AS THE TERMINAL NODE

KODES(J)=2\*\*(J-1) WHERE J IS A NODE OF THE SFG

KODET(I)=COMPOSITE CODE ASSOCIATED WITH CIRCUIT I. THIS CODE REPRESENTS THE SET OF SYMBOLS CORRESPONDING TO THE SET OF SFG BRANCHES CONTAINED IN CIRCUIT I

KODF(NZ) IS THE MULTIPLICITY OF THE SYMBOL CORRESPONDING TO THE CODE KODI(NZ)

KODI(NZ), NZ=1,2,...,IZ IS THE SET OF INDIVIDUAL SYMBOL CODES THAT MAKE UP THE COMPOSITE CODE KSORT(JZ)

KONC(J)=COLUMN COUNTER FOR ROW J OF THE ROUTING MATRIX N(J,K)

KONS(KOZY)=1 IF THE SYMBOL HAVING CODE KOZY IS NOT AN INVERSE SYMBOL A 0 IF THE SET OF SYMBOLS CORRESPONDING TO THE COMPOSITE CODE KSORT(J) BELONGS TO THE DENOMINATOR POLYNOMIAL

KONSO(I)=1 IF SYMBOL OF THE SFG BRANCH I=SYMBOL(I), 0 IF SYMBOL OF THE SFG BRANCH I=1/SYMBOL(I)

KSORT(K)=THE CODE ASSIGNED TO COLUMN K OF THE MATRIX POLY

LB(NUMC)=LBX(I) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)

LBX(I)=TERMINAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST

LIL=A COLUMN COUNTER OF THE MATRIX POLY

LINC(NUMC)=1 IF THE NETWORK BRANCH NUMBERED NUMC BY THE USER IS NOT IN THE TREE, 0 OTHERWISE

LIST=NUMBER OF DIRECTED BRANCHES IN THE SFG

LISTC=1 IF ALL CIRCUITS OF THE SFG ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE

LISTG=1 IF SFG INFORMATION (BRANCH SYMBOLS, WEIGHTS ETC.) ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE

LISTP=1 IF ALL PATHS FROM NODE NIN TO NODE NOUT ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE

LT(J)=NUMBER OF POSITIVE ENTRIES IN ROW J OF N(J,K)

MAPO(NIP)=NOCTOT(NIP)-NOCTOT(NIP-1) WHICH EQUALS THE NUMBER OF LOOPS NOT TOUCHING LOOP NIP

MIX(I)=MAPPING OF THE SFG BRANCH LIST INTO A LIST SATISFYING ONE OF THE FOLLOWING CONDITIONS  
NFIRST(J).GT.NFIRST(K) FOR J.GT.K  
OR NFIRST(J)=NFIRST(K), NLAST(J).LT.NLAST(K) FOR J.GT.K

MSORT(K)=THE EXPONENT OF S ASSIGNED TO ROW K OF THE MATRIX POLY N(J,K), WHERE K=1,2,...,LT(J), IS THE TERMINAL NODE OF SFG BRANCH HAVING J AS ITS INITIAL NODE. THE VALUE OF EACH NONZERO ENTRY IN A GIVEN ROW IS MADE TO DECREASE AS K INCREASES. THE ADDITION ENTRY N(NIN,LT(NIN)+1)=-1 IS ALSO MADE

NA(J)=NUMBER OF SYMBOLS (NOT COUNTING INVERSE SYMBOLS) IN THE CODE KSORT(J)

NB(J)=NUMBER OF INVERSE SYMBOLS IN THE CODE KSORT(J)

NCIR=1 IF CIRCUITS ARE TO BE FOUND, AND 0 IF CIRCUITS ARE NOT TO BE FOUND

NEST(I)=1 IF THE SFG BRANCH I CONTAINS A SYMBOL IN ADDITION TO THE LAPLACE VARIABLE S, 0 IF THE SFG BRANCH I CONTAINS NO SYMBOL EXCEPT POSSIBLY FOR THE LAPLACE VARIABLE S

NF(LF,JROJ)=ROUTING TABLE FOR THE NETWORK COMPOSED ONLY OF BRANCHES BELONGING TO THE TREE

NFIR=1 IF PATHS ARE TO BE FOUND(NFIR SET TO 1 IF LISTP=1), AND 0 IF PATHS ARE NOT TO BE FOUND

NFIRST(I)=INITIAL NODE OF THE DIRECTED SFG BRANCH I

NIN=NETWORK BRANCH NUMBER OF THE SOURCE. THIS BECOMES THE SOURCE NODE OF THE SFG

NLAST(I)=TERMINAL NODE OF THE DIRECTED SFG BRANCH I

NOB=NUMBER OF BRANCHES IN NETWORK

NOD=NUMBER OF NODES IN NETWORK

NODA=0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK ELEMENT. IN THIS CASE IT DESIGNATES THE POSITIVE TERMINAL OF THE OUTPUT VOLTAGE

NODR=0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK ELEMENT. IN THIS CASE IT DESIGNATES THE NEGATIVE TERMINAL OF THE OUTPUT VOLTAGE

NOL=NUMBER OF CIRCUITS(LOOPS)

NOP=NUMBER OF PATHS FROM NODE NIN TO NODE NOUT IN THE SFG

NOUT=NETWORK BRANCH NUMBER ASSOCIATED WITH THE OUTPUT (VOLTAGE ACROSS OF CURRENT THRE). THIS BECOMES THE SFG NODE CORRESPONDING TO I OUTPUT VARIABLE

NOTCH(NOC) AND NOCTOT(K). CONSIDER THE INTEGER SET (I)=(1,2,...,N2) WHERE N2=NUMBER OF NONTOUCHING PAIRS OF LOOPS. NOW CONSIDER THE FOLLOWING SUBSET OF (I). S(I)=(NOCTOT(K-1)+1,NOCTOT(K-1)+2,...,NOCTOT(K)) WHERE NOCTOT(0)=0. THEN THE SET (NOTCH(J), J IN S(I)) IS THE SET OF LOOPS THAT DO NOT TOUCH LOOP K

NOTREE(I)=1 IF THE I-TH NETWORK BRANCH IN THE DATA LIST IS CHOSEN FOR THE TREE, 0 OTHERWISE

NP(I)=THE NODE SEQUENCE OF A PATH BETWEEN NODE NIN AND NODE NOUT OF THE SFG. IF NIN=NOUT THIS IS THE NODE SEQUENCE FOR A CIRCUIT

NPCODE(K)=COMPOSITE CODE USED TO IDENTIFY CIRCUIT I. FOUND BY SUMMING THE CODES, KODES(J), ALLOTTED TO EACH NODE, J, IN THE CIRCUIT

NUML(NUMC)=NUMLA(I) WHERE NUMC=NUMX(I) (USED ONLY FOR NETWORK TREE BRANCHES)

NS(LF,JF)=1 IF THE NETWORK TREE BRANCH TR(LF,JF) HAS INITIAL NODE LF AND TERMINAL NODE JF AND EQUALS -1 IF THE NETWORK TREE BRANCH HAS INITIAL NODE JF AND TERMINAL NODE LF

NUMX(I). IF I-TH NETWORK BRANCH IN THE DATA BRANCH LIST IS A DEPENDENT SOURCE, THIS ARRAY EQUALS THE NETWORK BRANCH NUMBER ASSIGNED TO ITS CONTROL

NUMX(I)=THE NETWORK BRANCH NUMBER ASSIGNED BY THE USED TO THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST

NUP(J) DESIGNATES THE LOOP, ISET(J,NUP(J)), OF ROW J WHICH IS NOT TOUCHED BY THE LOOPS ENTERED IN ROW J+1 OF ISET.

POLY(K,L)=MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A TERM IN THE NUMERATOR OR DENOMINATOR OUTPUT POLYNOMIAL HAVING THE S POWER OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L

POLYU(K,L)=MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A TERM IN THE NUMFRATOR OF THE OUTPUT POLYNOMIAL HAVING THE S POWR OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L

SEMBOL(KO)=SYMBOL CORRESPONDING TO THE CODE KBASIS\*(KO-1)

SEMPOB(J1,J2), J2=1,2,...,NA(J1), AND SEMPOD(J1,J3), J3=1,2,...,NB(J1) ARE RESPECTIVELY THE MULTIPLICITY OF THE SYMBOLS SIMBON(J1,J2), J2=1,...,NA(J1), AND SIMBOD(J1,J3), J3=1,...,NB(J1)

SIMBON(J1,J2), J2=1,2,...,NA(J1), AND SIMBOD(J1,J3), J3=1,2,...,NB(J1) ARE RESPECTIVELY THE SYMBOLS AND INVERSE SYMBOLS CORRESPONDING TO THE SYMBOL CODE KSORT(J1)

SMBOL(K)=SYMBOL(I) FOR THE SFG BRANCH I WHERE I=MIX(K)  
STAR(I)=\*\*I THIS ARRAY IS GENERATED FROM DATA STATEMENTS AND IS  
USED IN FORMING THE ARRAYS SEMPON AND SEMPOD  
SYM(NUMC)=SYM(X) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)  
SYMBOL(I)=SYMBOL ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I  
SYM(X(I)=SYMBOL(3 CHARACTERS AT MOST) ASSIGNED BY USER TO THE I-TH  
NETWORK BRANCH IN THE DATA BRANCH LIST. THE ELEMENTS VALUE,  
IF NOT SPECIFIED IS REPRESENTED BY THIS SYMBOL  
TYPB(NUMC)=TYP(X(I) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)  
TYPE(NUMC)=TYP(X(LINK) WHERE NUMC=NUMX(LINK)  
(USED ONLY FOR NETWORK BRANCHES NOT IN THE TREE)  
TYP(X(I)=SPECIFIES THE ELEMENT TYPE OF THE I-TH NETWORK BRANCH  
IN THE DATA BRANCH LIST, (MUST BE E, I, G, R, L, C, CC, CV, VC, OR VV  
AND MUST BE COMPATIBLE WITH THE UNITS OF THE ELEMENTS VALUE)  
VAL(NUMC)=VAL(X(I) WHERE NUMC=NUMX(I)  
(USED ONLY FOR NETWORK TREE BRANCHES)  
VAL(X(I)=ELEMENT VALUE(E12.5) OF I-TH NETWORK BRANCH IN THE DATA  
BRANCH LIST  
WEIGT(I)=CONSTANT TERM ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I

IV-2. Flow Charts

Program SNAP is divided into the following sections:

Program MAIN (Subprograms 1 thru 12)

Subroutine SFG (Subprograms A thru J)

Subroutine FTREE

Subroutine TREP

Subroutine ARRAY

Subroutine DECODE

As indicated above, program MAIN is further broken down into 12 subprograms and subroutine SFG is divided into 10 subprograms.

Subprogram MAIN-1

This program reads in some preliminary network data.

Read in

- (a) problem name
- (b) NOD, NOB, KBASIS, LISTG,  
LISTC, LISTP, NIN, NOUT,  
NODA, NODB

Set KBASIS to 8 if a zero valve  
has been read in.

Write out the above information  
for reference purposes.



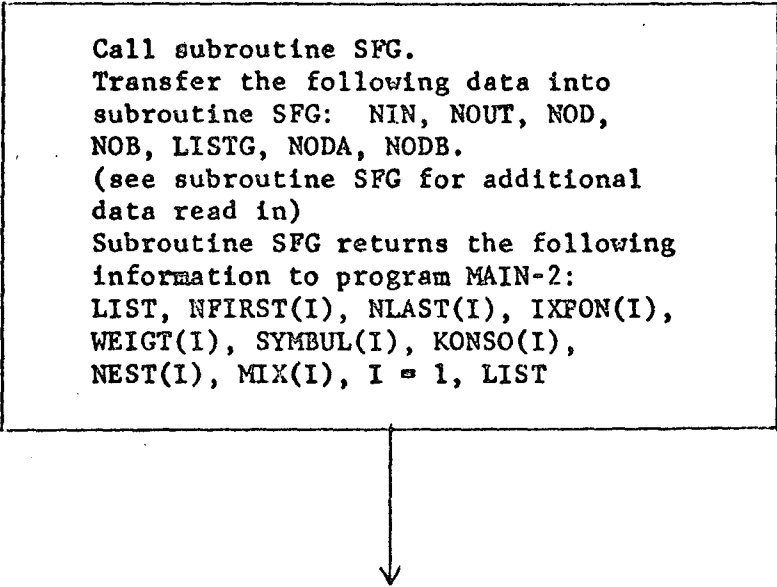
To Subprogram MAIN-2

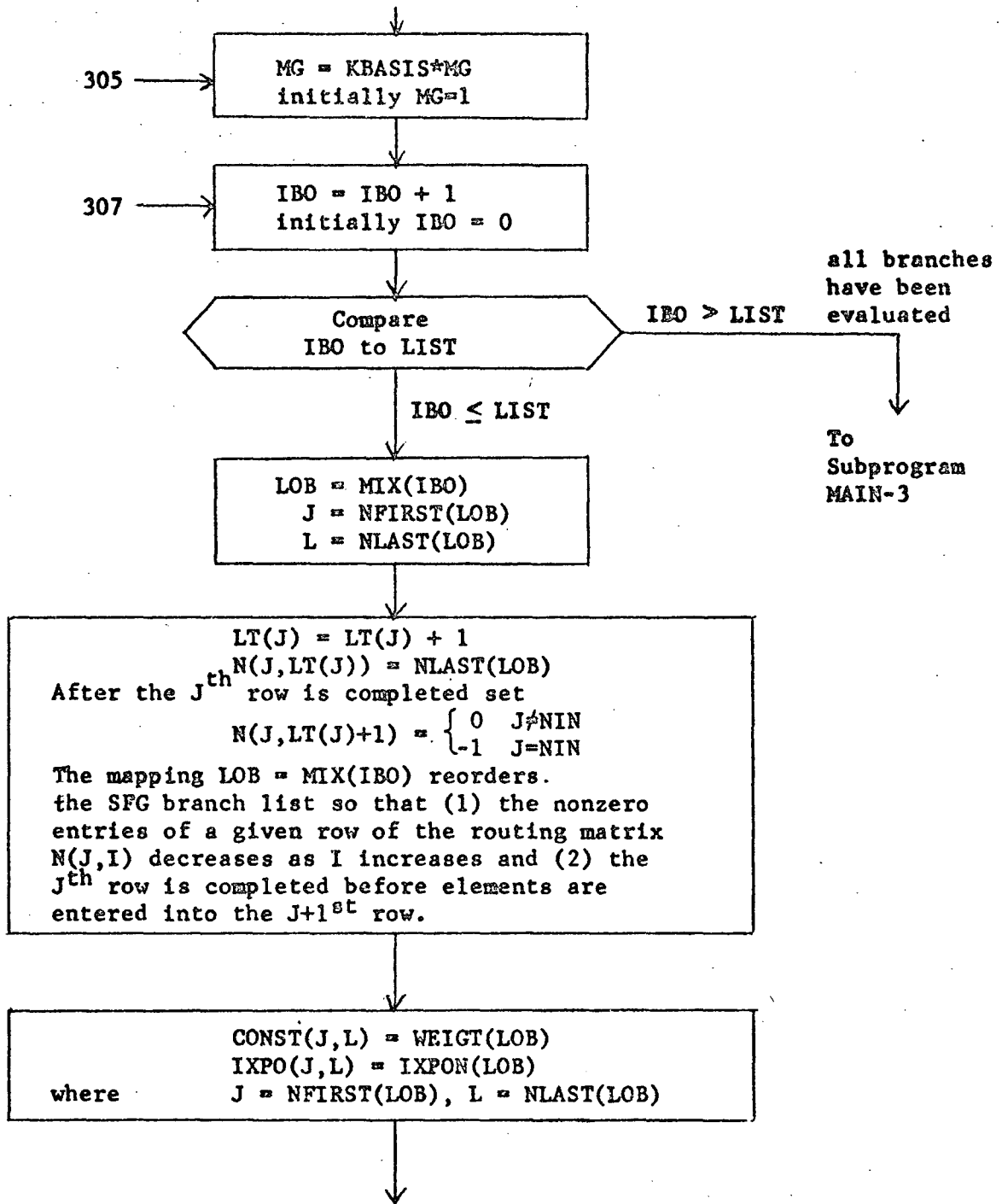


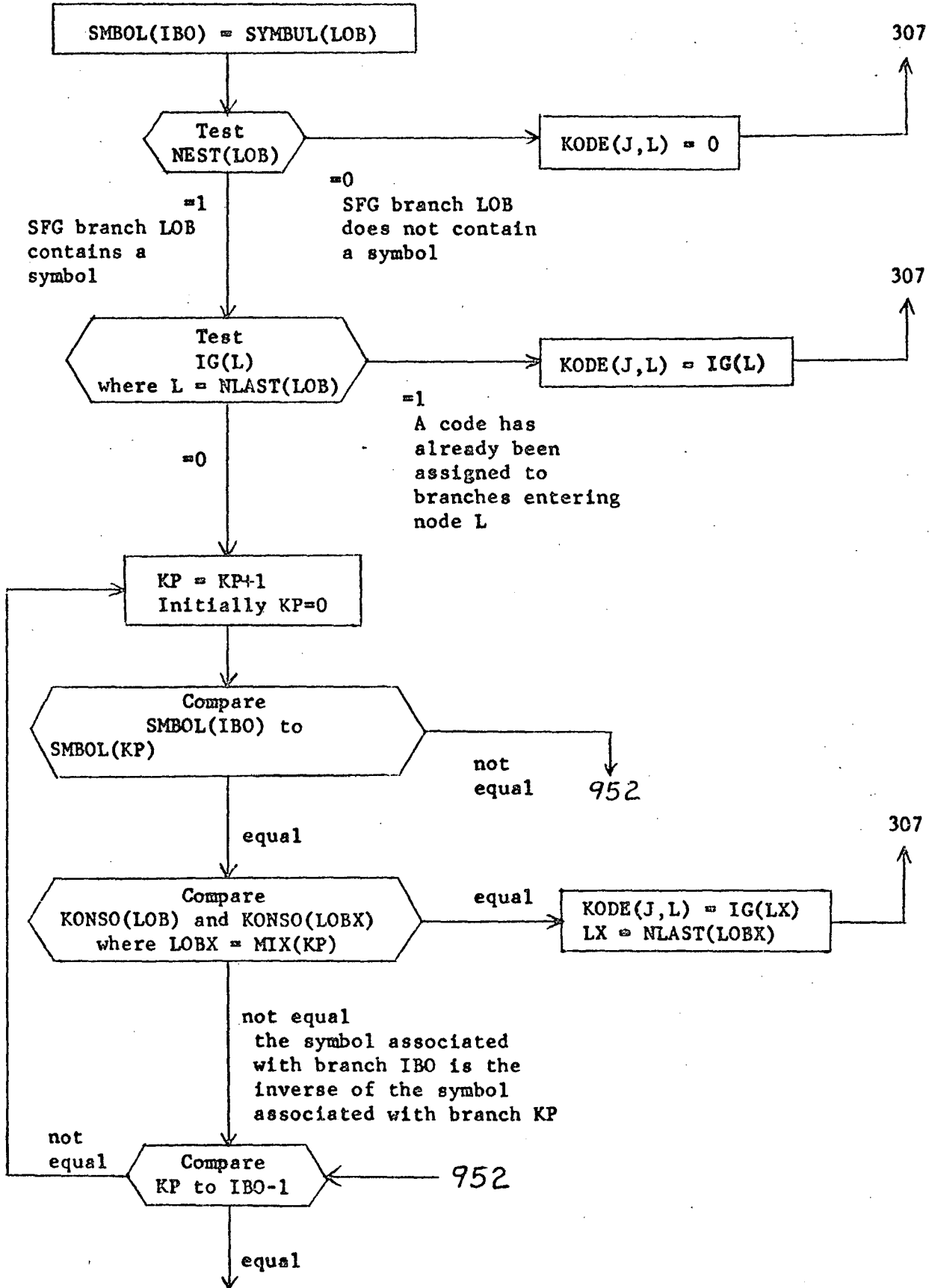
Subprogram MAIN-2

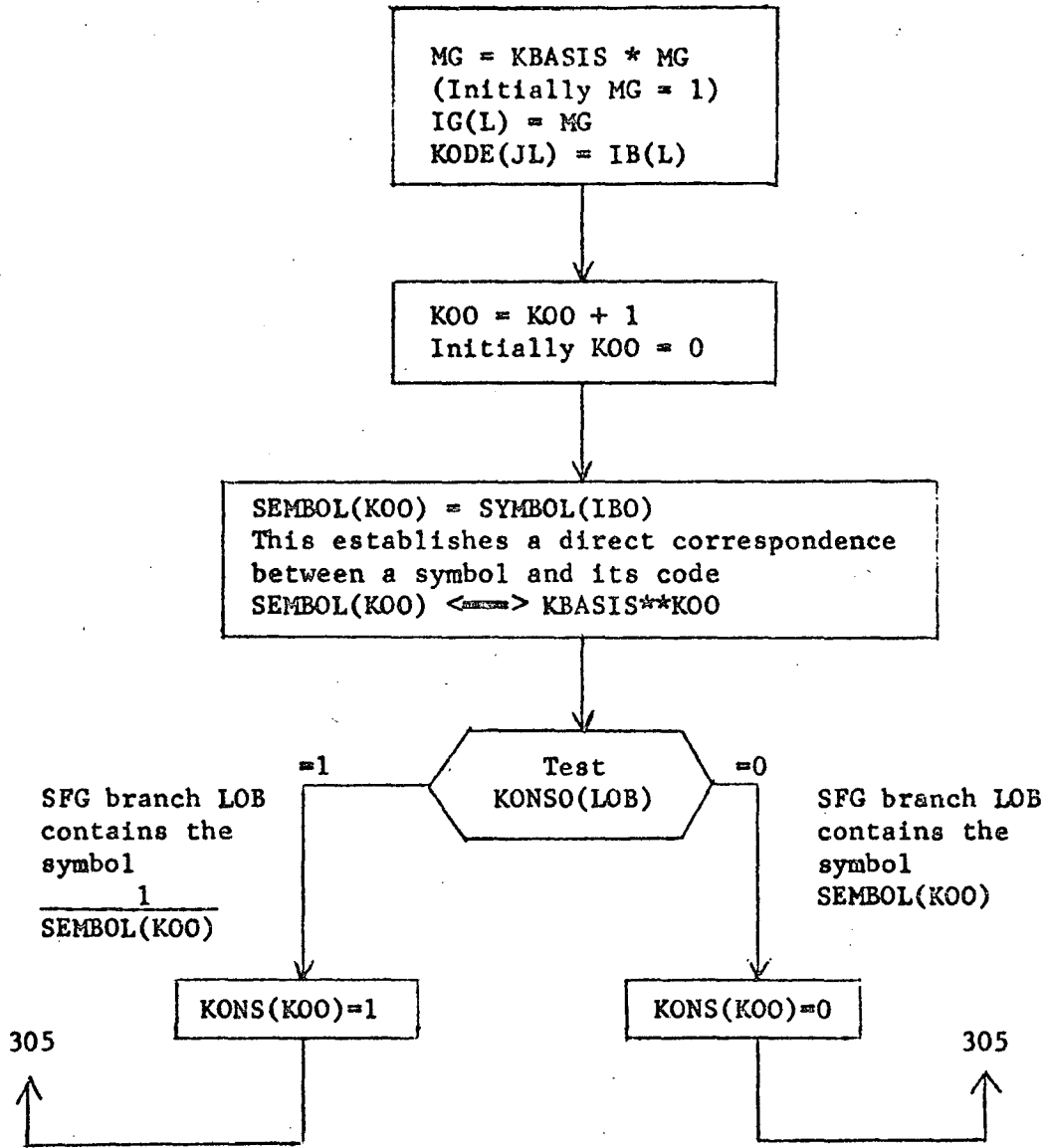
This program generates the SFG routing matrix, creates a code for each symbol (excluding s), and sets up arrays for the constants and powers of s associated with the branch weights.

Call subroutine SFG.  
Transfer the following data into  
subroutine SFG: NIN, NOUT, NOD,  
NOB, LISTG, NODA, NODB.  
(see subroutine SFG for additional  
data read in)  
Subroutine SFG returns the following  
information to program MAIN-2:  
LIST, NFIRST(I), NLAST(I), IXPON(I),  
WEIGT(I), SYMBUL(I), KONSO(I),  
NEST(I), MIX(I), I = 1, LIST



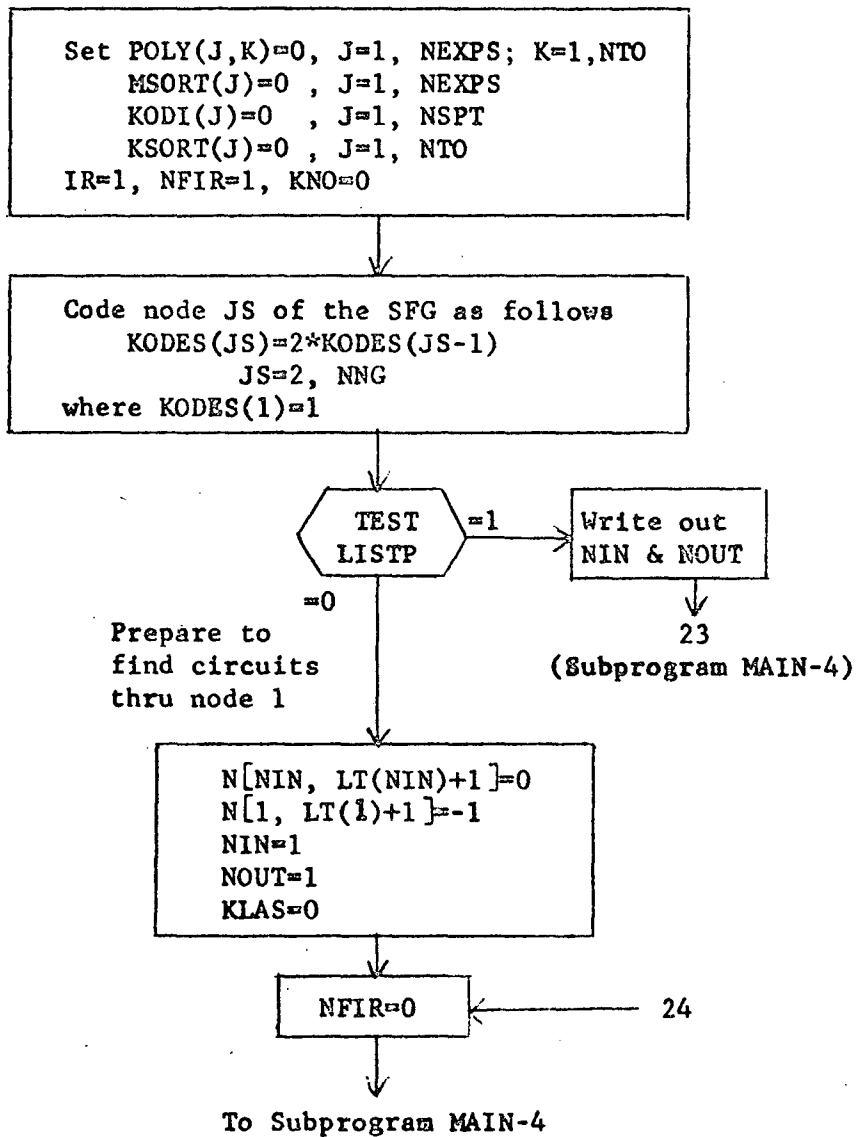






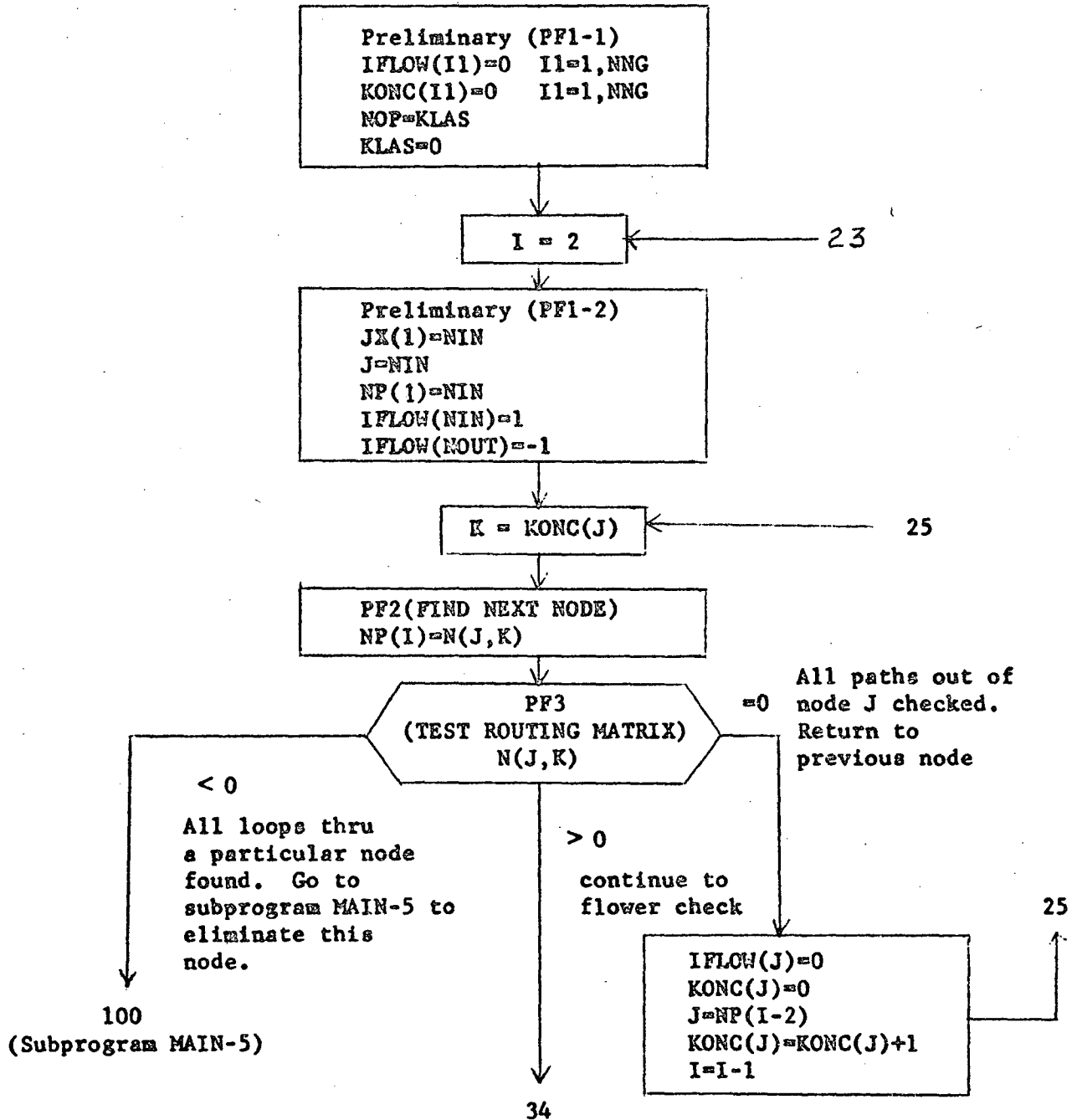
Subprogram MAIN-3

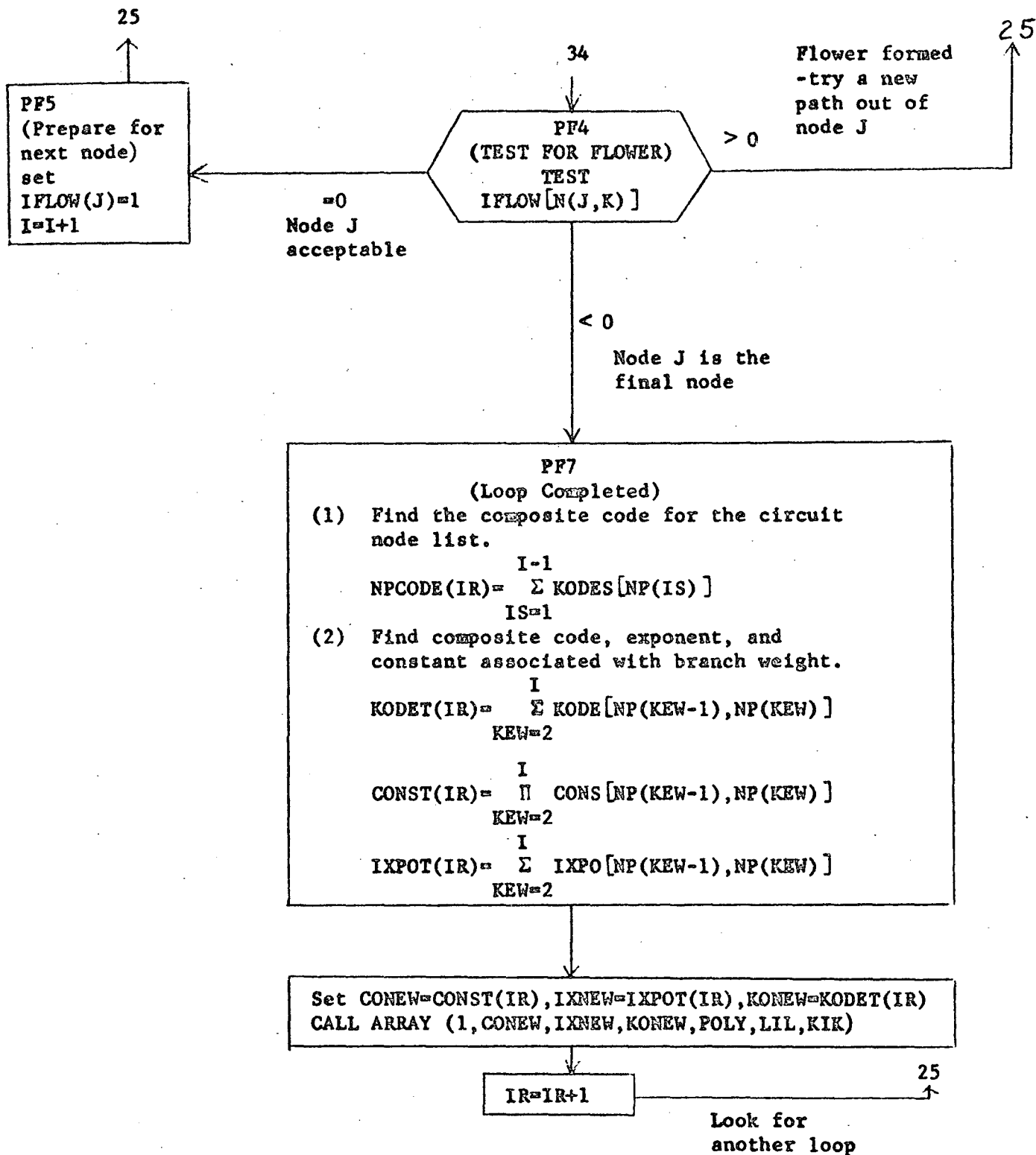
This program codes the nodes of the SFG, and prepares the counters for finding all paths and/or circuits.



Subprogram MAIN-4

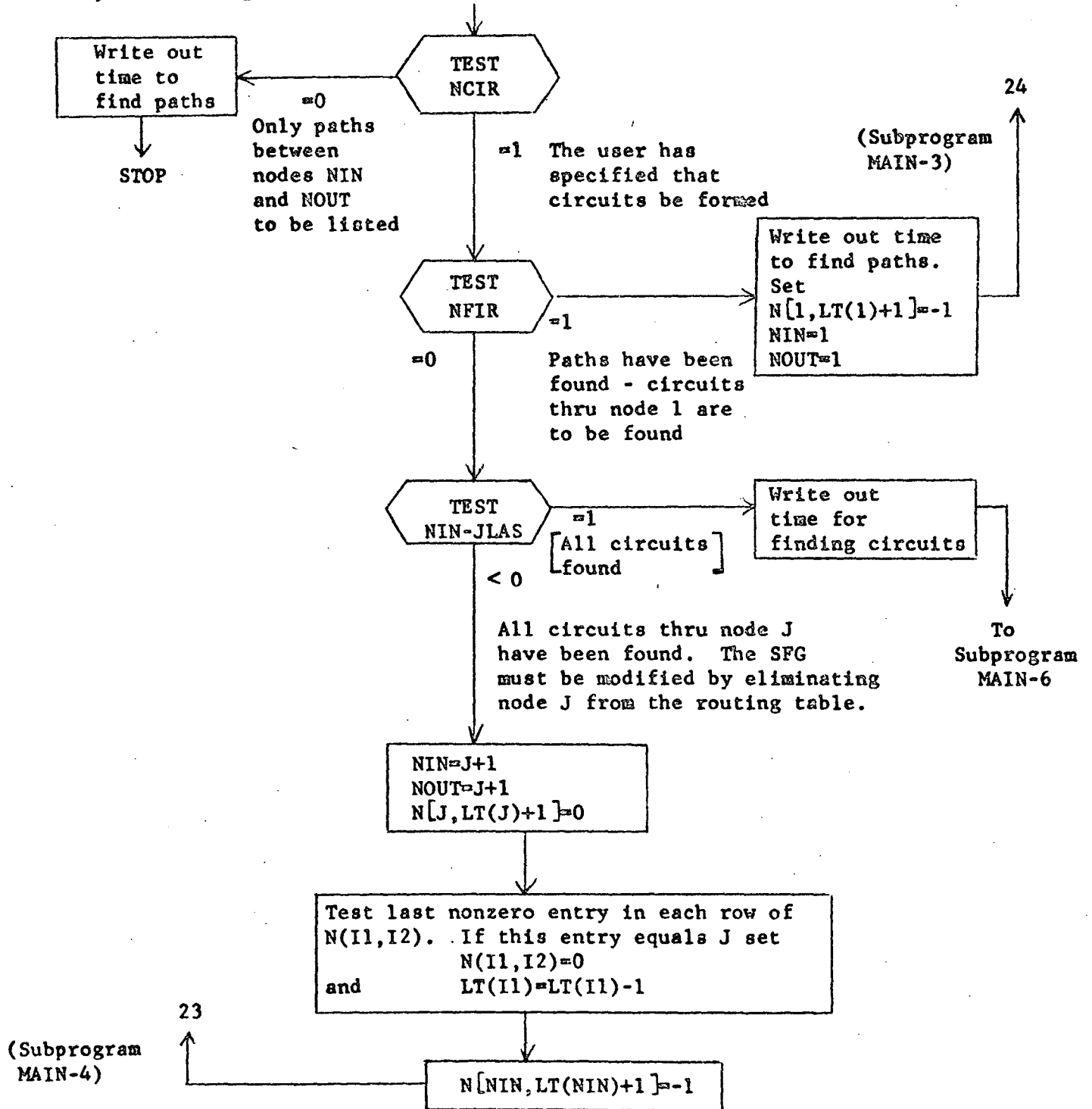
This program finds all paths from node NIN to node NOUT and/or all circuits of the SFG.





Subprogram MAIN-5

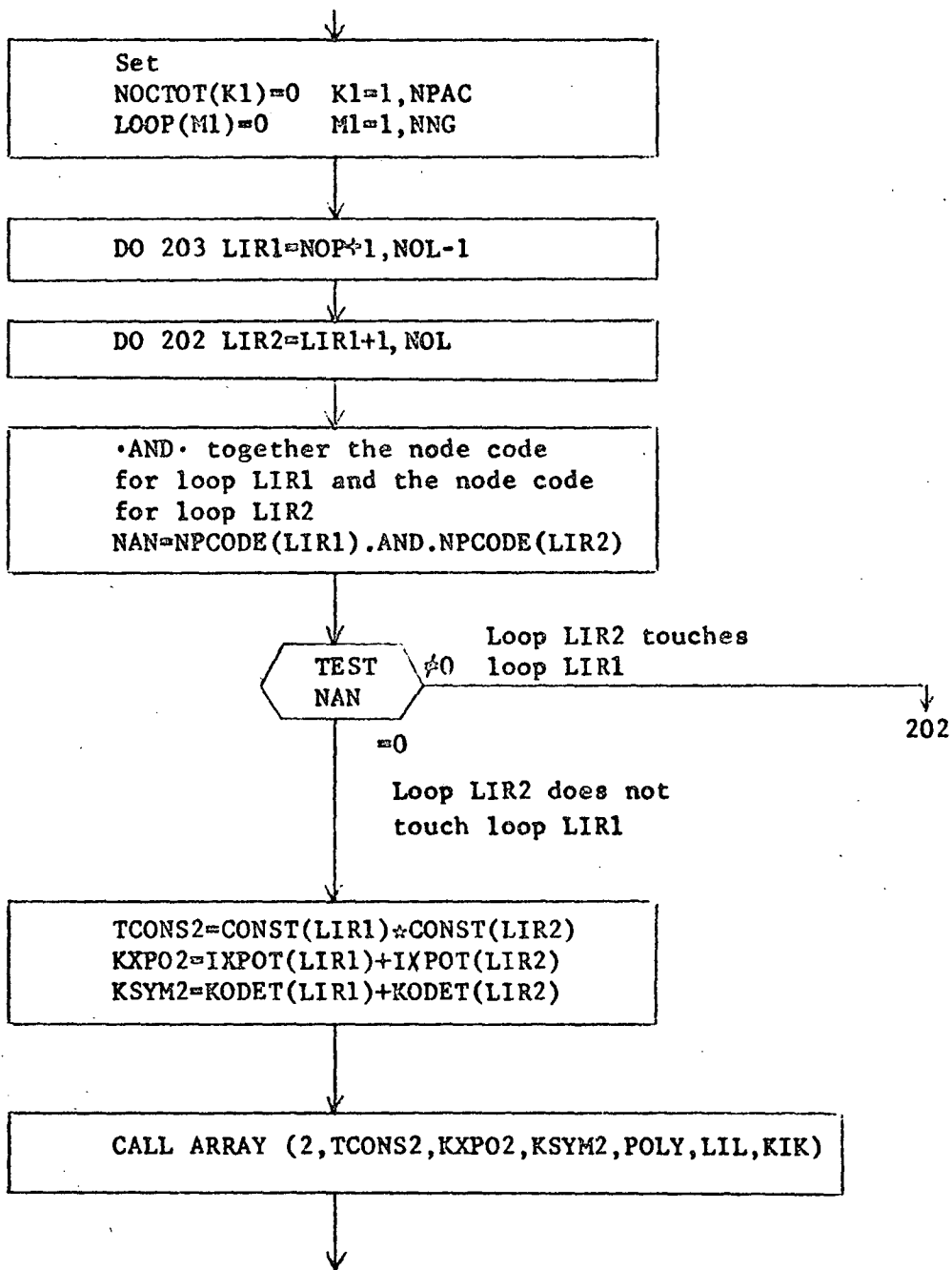
This program determines if circuits are to be found and if so modifies the SFG by eliminating node J.

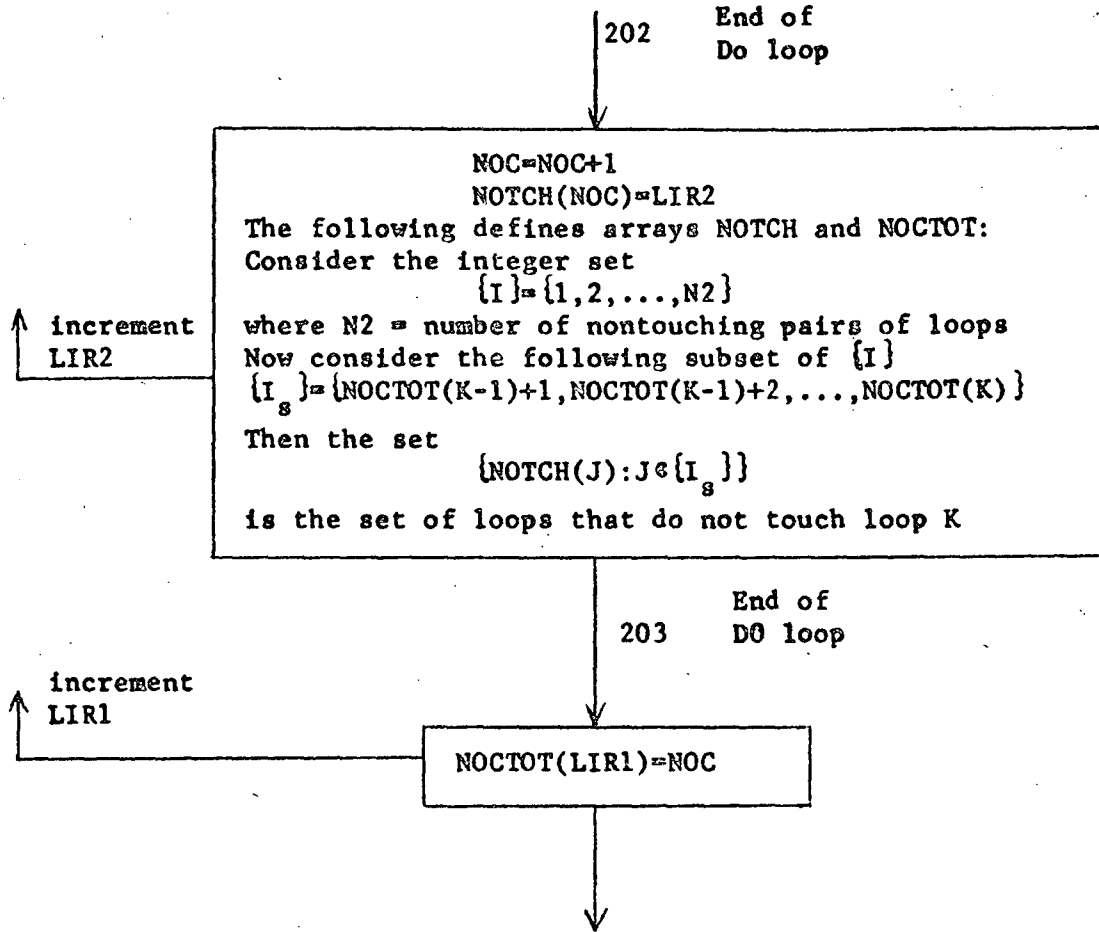




Subprogram MAIN-6

This program finds and stores all 2<sup>nd</sup> order nontouching loops





Subprogram MAIN-7

This program finds all nontouching loops of order greater than 2, and stores the associated code, power of s, and constant term.

The matrix ISET is given below in its general form to aid in understanding the flow chart of subprogram MAIN-7.

$$\begin{bmatrix} \text{ISET}(1,1)\text{ISET}(1,2)\cdots\text{ISET}[1,\text{NUP}(1)]\cdots\text{ISET}[1,\text{JAC}(1)] \\ \text{ISET}(2,1)\text{ISET}(2,2)\cdots\text{ISET}[2,\text{NUP}(2)]\cdots\text{ISET}[2,\text{JAC}(2)] \\ \vdots \\ \text{ISET}(\text{KAP},1)\text{ISET}(\text{KAP},2)\cdots\text{ISET}[\text{KAP},\text{NUP}(\text{KAP})]\cdots\text{ISET}[\text{KAP},\text{JAC}(\text{KAP})] \\ \text{ISET}(\text{KAP}+1,\text{NUP}(\text{KAP}+1)]\cdots \end{bmatrix}$$

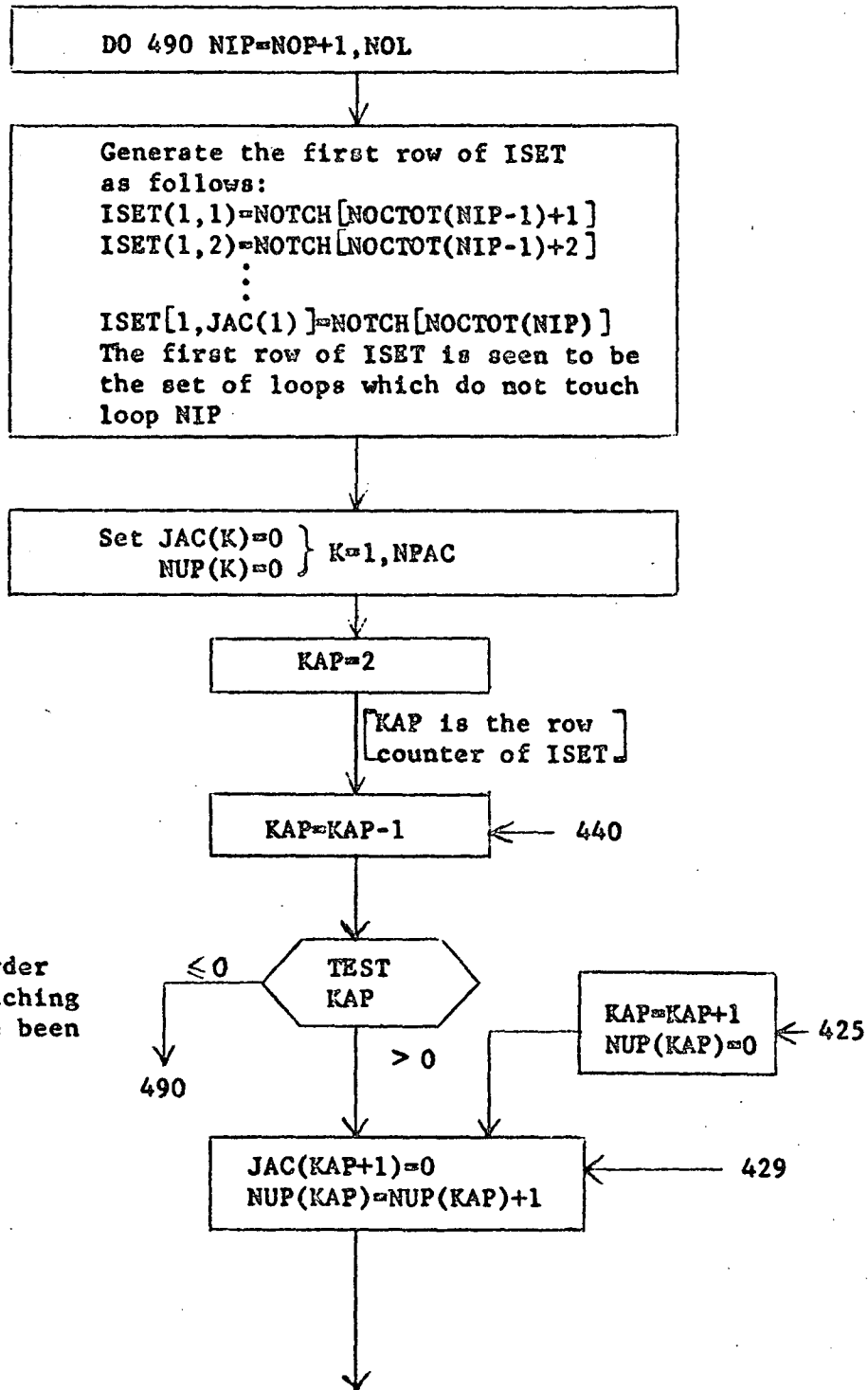
where  $\text{ISET}(J,I)$ ,  $I=1,2,\dots,\text{JAC}(J)$

is the subset of

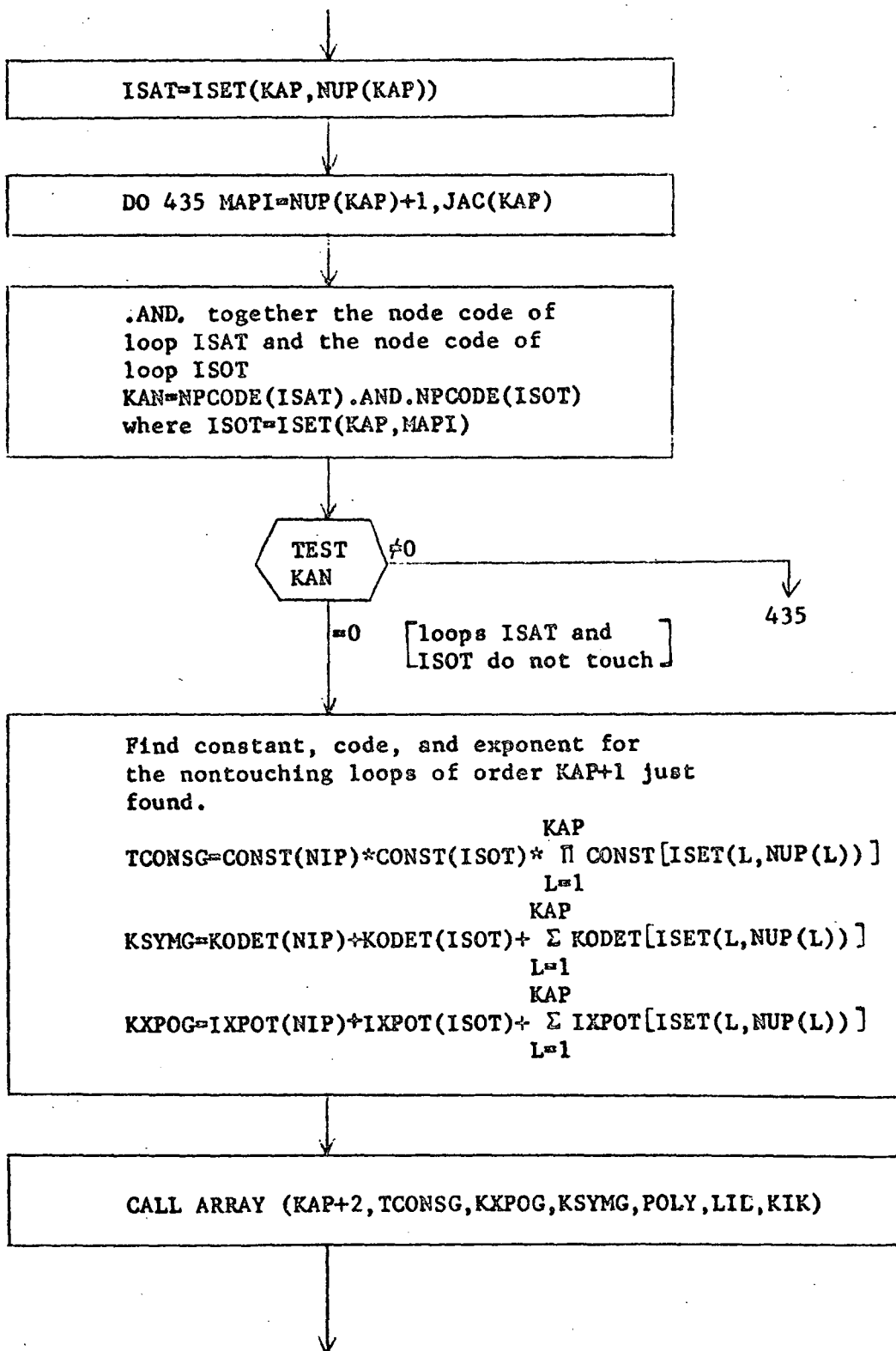
$$\{\text{ISET}[J-1,\text{NUP}(J-1)+1], \text{ISET}[J-1,\text{NUP}(J-1)+2], \dots, \text{ISET}[J-1,\text{JAC}(J-1)]\}$$

which does not touch the loop

$$\text{ISET}[J-1,\text{NUP}(J-1)]$$



All higher order loops not touching loop NIP have been found



increment  
MAPI

435 [End of  
DO loop]

```

Update column counter of row KAP+1
of ISET and insert the last loop
found into ISET

JAC(KAP+1)=JAC(KAP+1)+1
ISET [KAP+1, JAC(KAP+1) ]=ISET(KAP, MAPI)

```

TEST  
 $JAC(KAP+1) - 2 \geq 0$

425

Row KAP+1 of ISET  
has now been found  
and it has more than  
1 non-zero entries.  
Thus, procede to  
find the entieres  
of row KAP+2.

< 0

TEST  
 $JAC(KAP) - NUP(KAP) - 1 > 0$

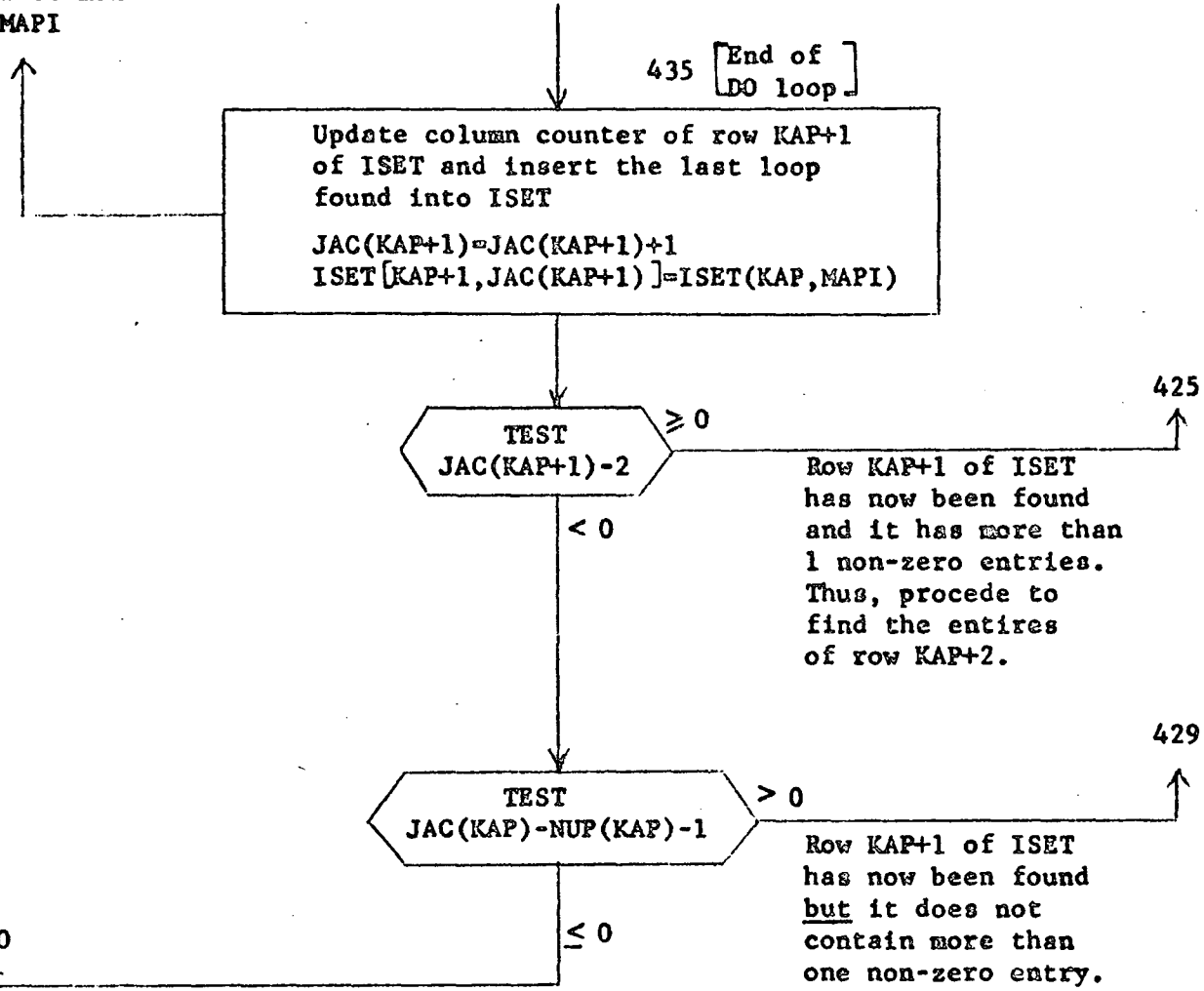
429

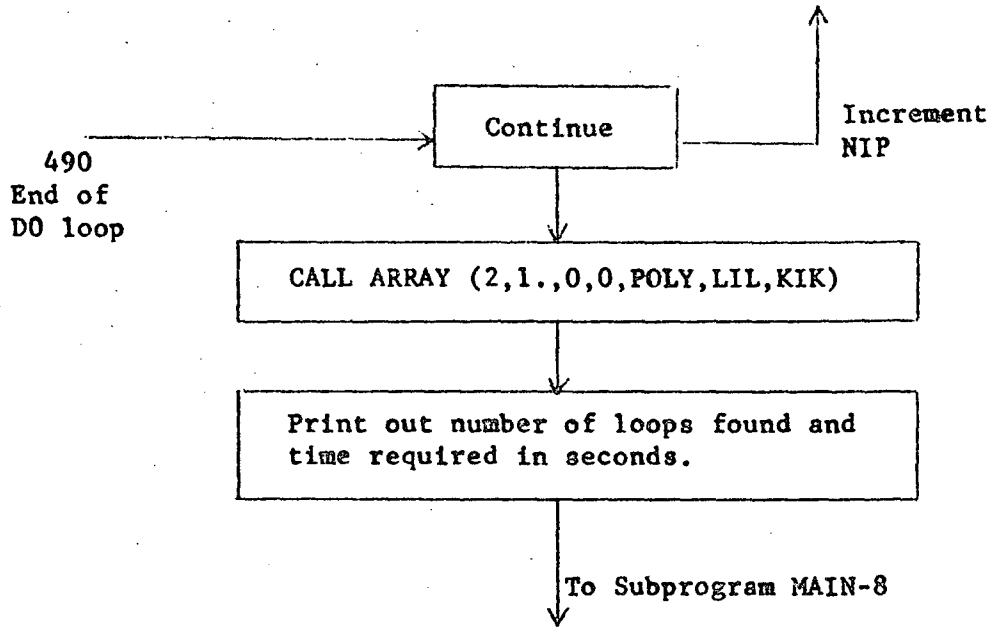
Row KAP+1 of ISET  
has now been found  
but it does not  
contain more than  
one non-zero entry.  
Since not all loops  
of row KAP have been  
exhausted, increment  
NUP(KAP) by one and  
re-evaluate row KAP+1

$\leq 0$

440

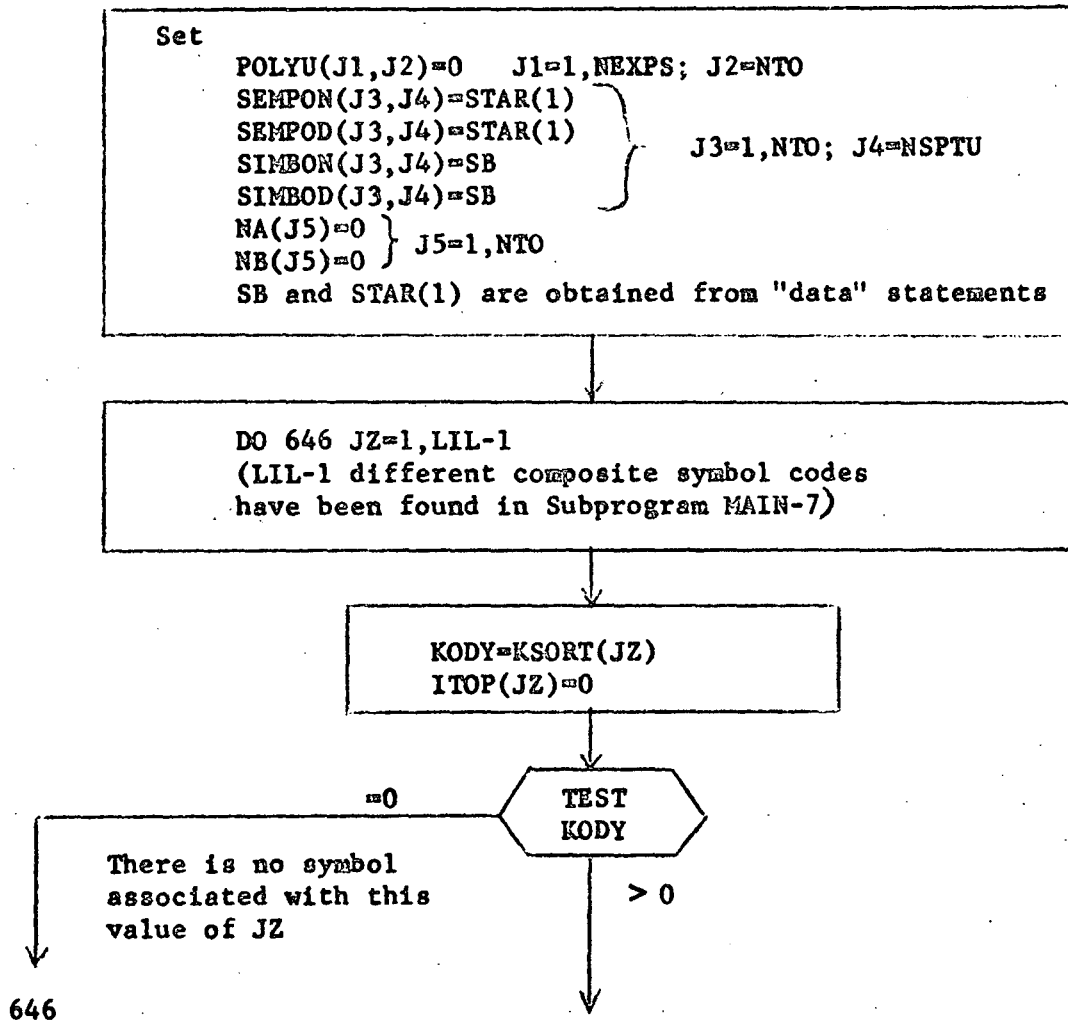
Row KAP+1 of ISET  
has now been found  
but it does not contain  
more than one non-zero  
element. Further, all  
loops of row KAP have  
been exhausted. Thus,  
it is necessary to  
back up one row and  
re-evaluated row KAP





Subprogram MAIN-8

This program decodes composite codes representing nontouching loops and sets up tags for use in printing out the symbolic transfer function.





CALL DECODE (KOO,KBASIS,KODY,IZ,FB,JZ,SEMBOL,KODF,  
KODI,ITOP)

This subroutine (a) sets

ITOP(JZ)=  $\begin{cases} 1 & \text{if terms having the code KSORT(JZ)} \\ & \text{belong in numerator of output polynomial} \\ 0 & \text{if terms belong in denominator} \end{cases}$

(b) finds the set

KODI(I), I=1,IZ

where SEMBOL[KODI(I)], I=1,IZ are the corresponding set of symbols, and

(c) finds the multiplicity of each individual symbol, SEMBOL[KODI(I)], and records these values in the array KODF(I), I=1,IZ

DO 645 NZ=1, IZ

TEST  
KONS[KODI(NZ)]

=0

=1

The symbol corresponding to KODI(NZ) is not to be inverted in the output

The symbol corresponding to KODI(NZ) is to appear inverted in the output.  
i.e.

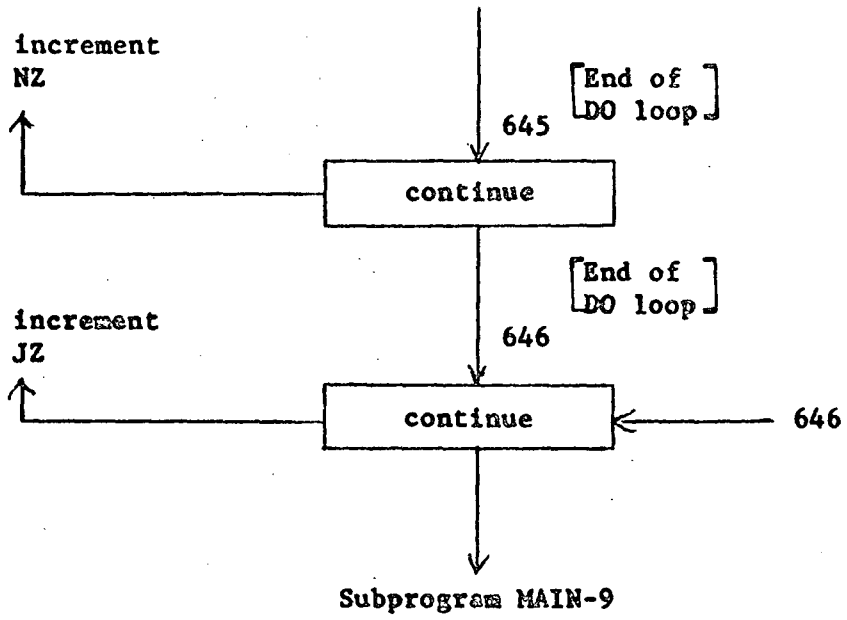
$\frac{1}{\text{SEMBOL}[\text{KODI}(\text{NZ})]}$

NAK=NAK+1  
SIMBON(JZ,NAK)=SEMBOL[KODI(NZ)]  
SEMPOB(JZ,NAK)=STAR[KODF(NZ)]

NAT=NAT+1  
SIMBOD(JZ,NAT)=SEMBOL[KODI(NZ)]  
SEMPOD(JZ,NAT)=STAR[KODF(NZ)]

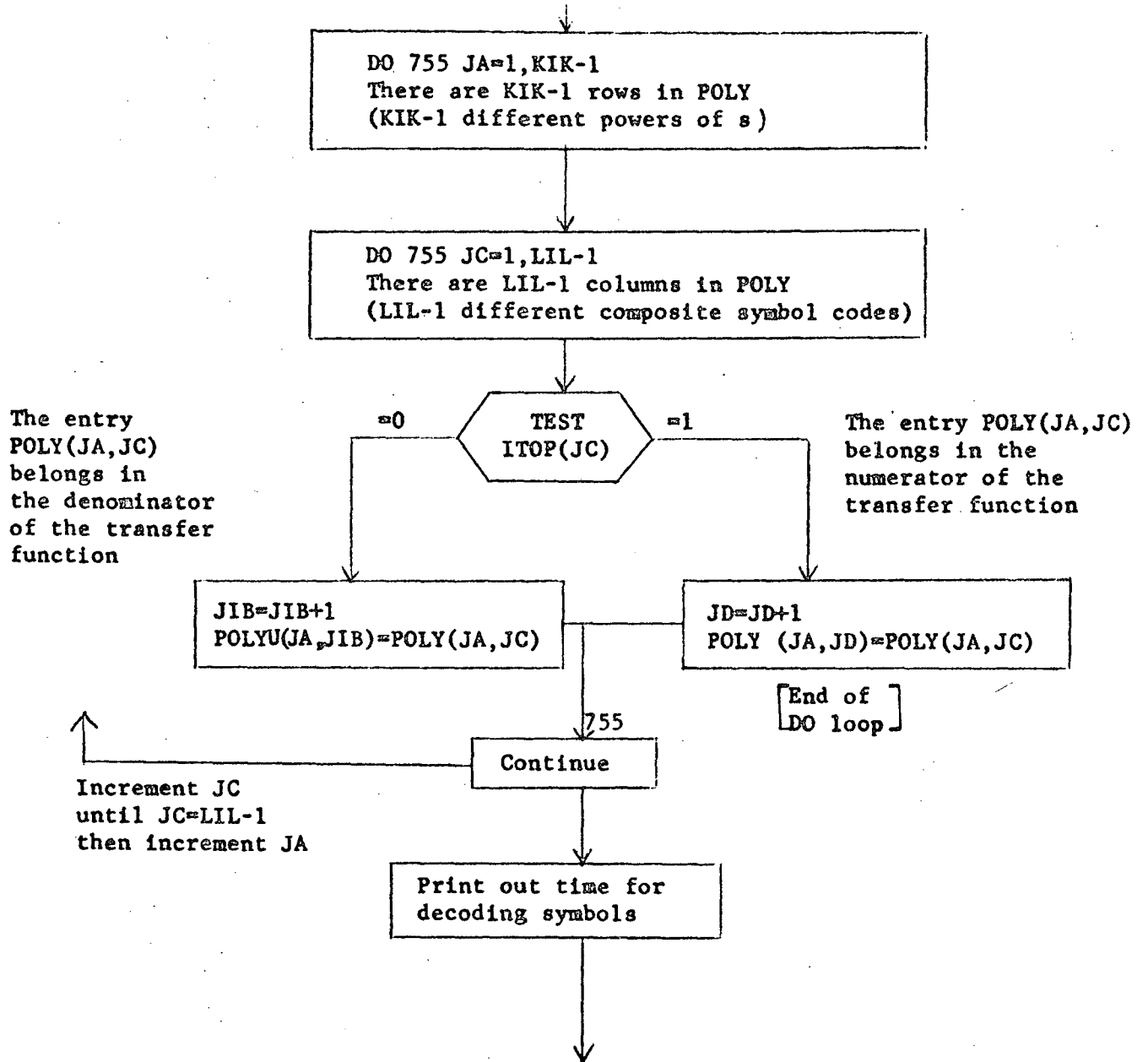
NA(JZ)=NA(JZ)+1

NB(JZ)=NB(JZ)+1



Subprogram MAIN-9

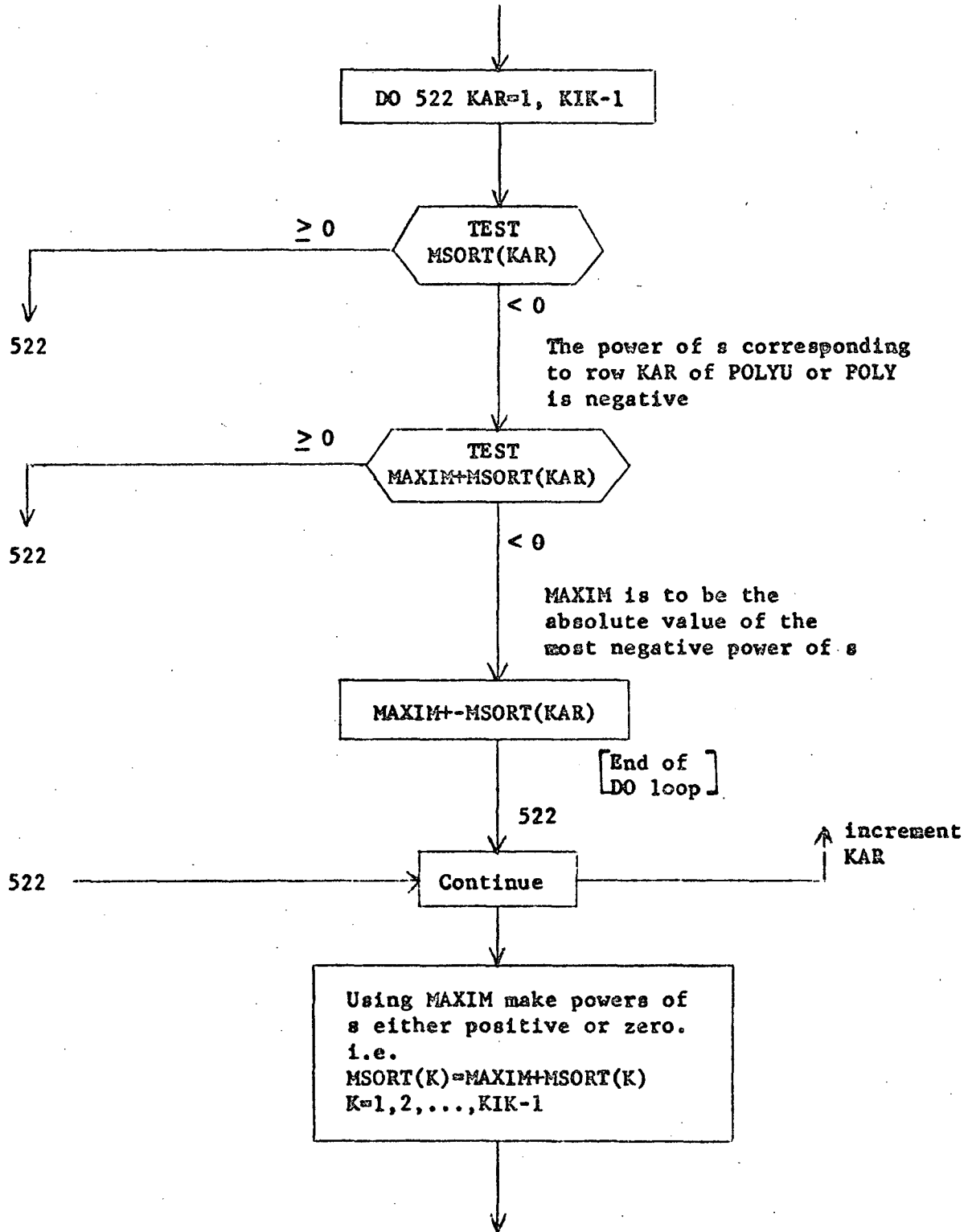
This program separates POLY into the arrays POLYU and POLY for use in printing out the constant terms of the transfer function.



Subprogram MAIN-10

Subprogram MAIN-10

This program normalizes the transfer function so as to have all positive powers of s



**Subprogram MAIN-11**

Write out the matrix of constant coefficients for the numerator polynomial of the transfer function. Also write out the symbols and s powers that correspond respectively to the columns and rows of the array of constants.

**Subprogram MAIN-12**

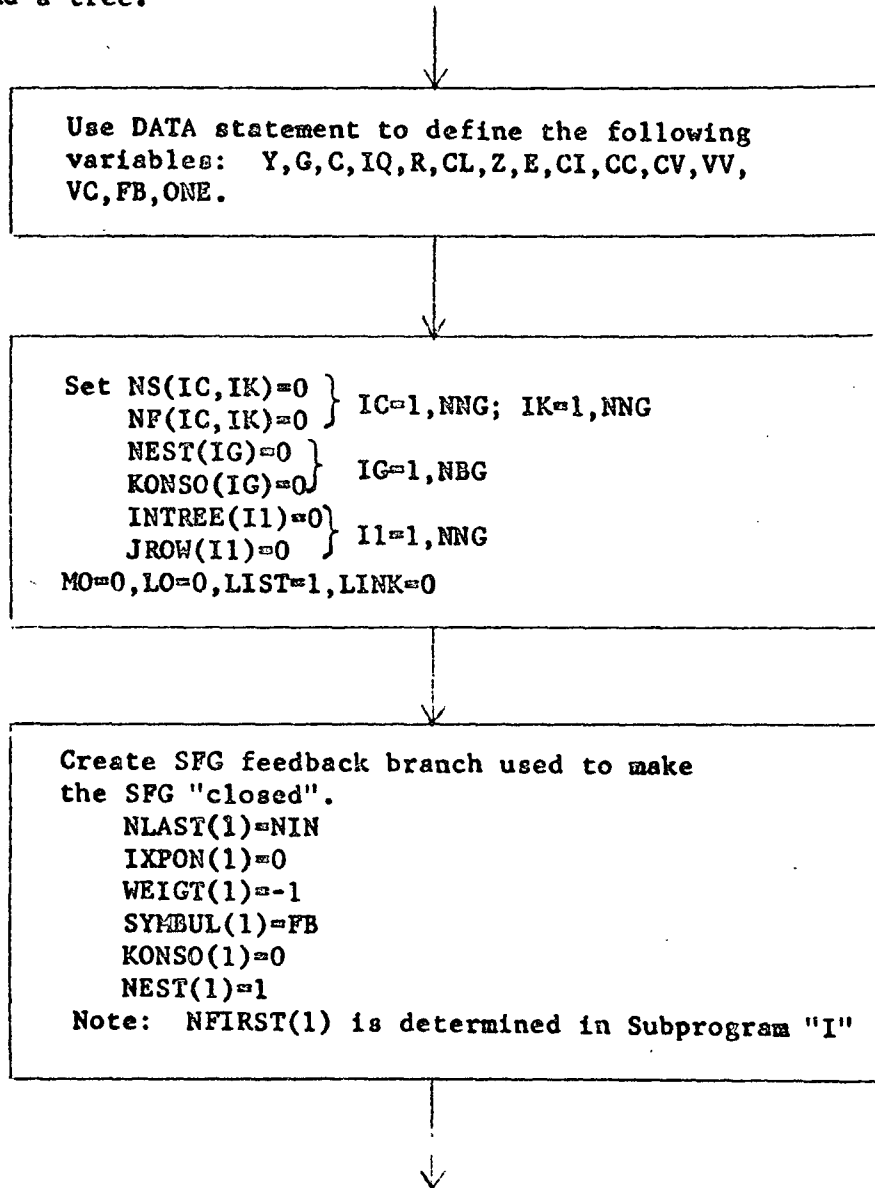
Write out the matrix of constant coefficients for the denominator polynomial of the transfer function. Also write out the symbols and s powers that correspond respectively to the columns and rows of the array of constants.

Subroutine SFG(NFIRST,NLAST,IXPON,WEIGHT,SYMBUL,KONSO,MIX,NEST,LIST,NIN,  
NOUT,NOD,NOB,LISTG,NODA,NODE)

This subroutine generates a signal flow-graph (SFG) for the given network.  
The program is subdivided into subprograms A thru J.

### Subprogram "A"

This program uses DATA statements to define certain variables, nulls arrays, creates the SFG feedback branch, reads in network branch information and calls FTREE to find a tree.

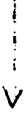




Read in the following network branch  
information:  
TYPX(I), NUMX(I), JBX(I), LBX(I), SYMX(I),  
IQUALX(I), VALX(I), NUMLX(I)  
I=1, NOB



Choose a tree of the network for  
use in finding the SFG  
CALL FTREE(TYPX, JBX, LBX, INTRE, NOTREE,  
NOD, NOB)

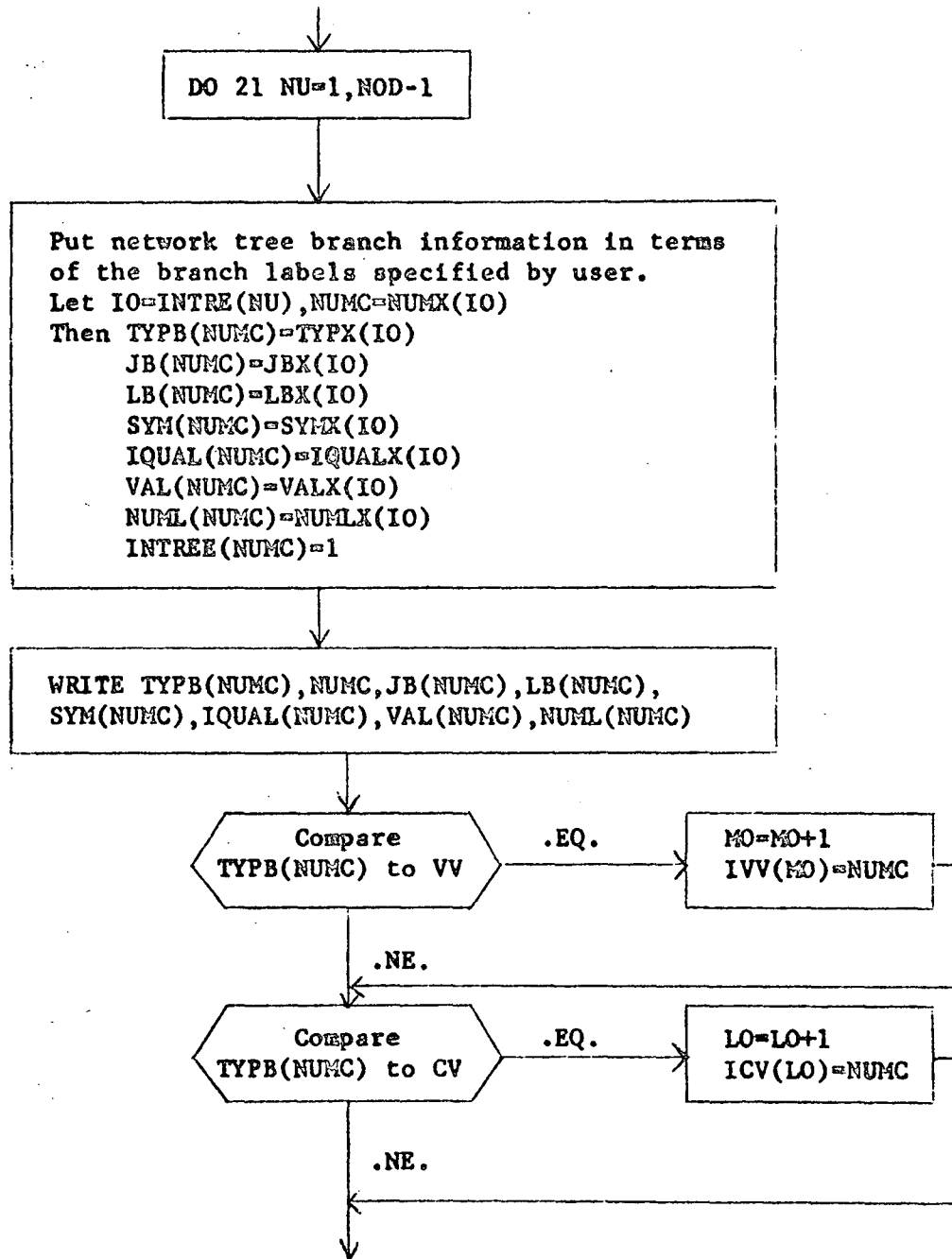


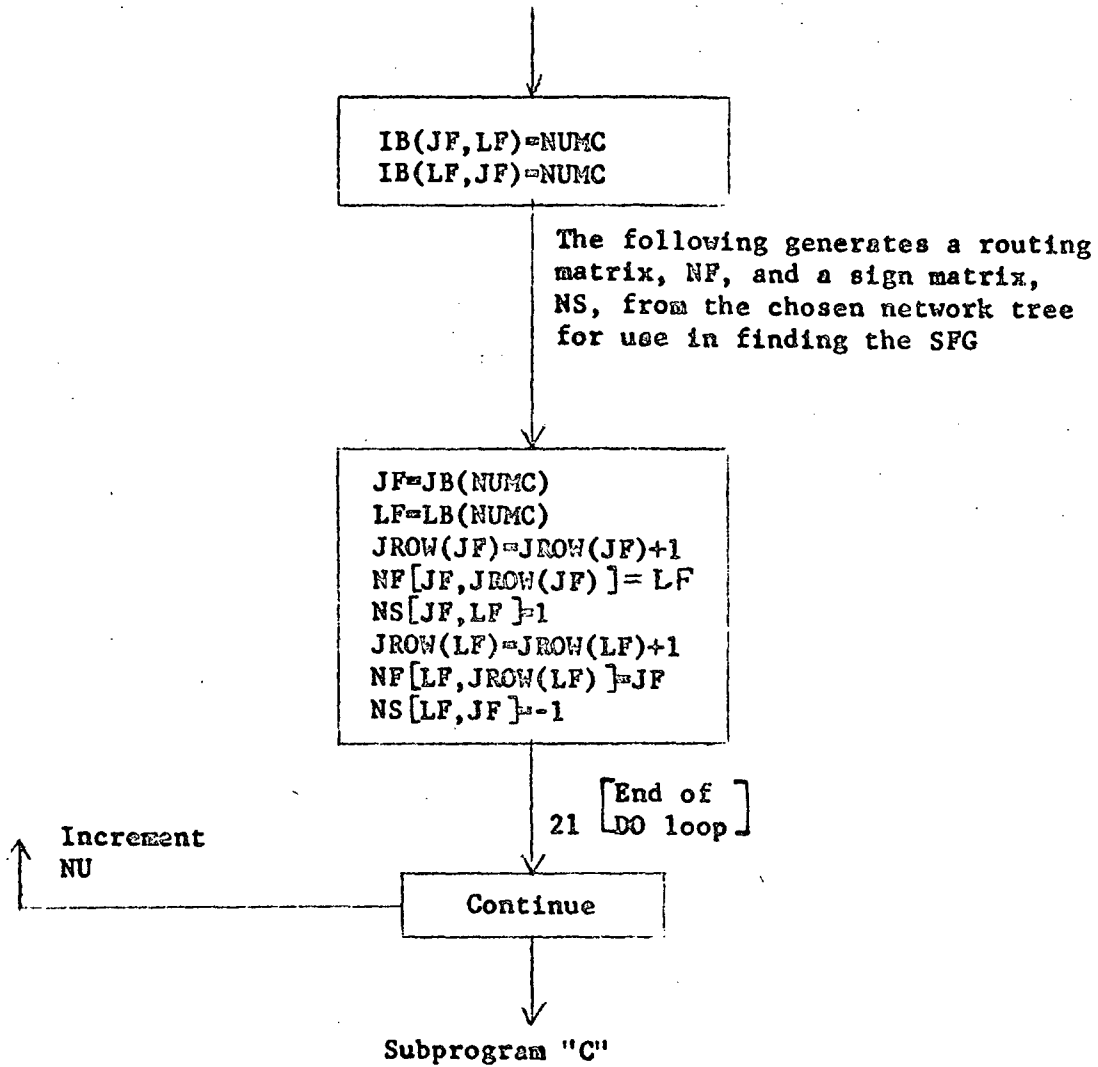
Subprogram "B"



### Subprogram "B"

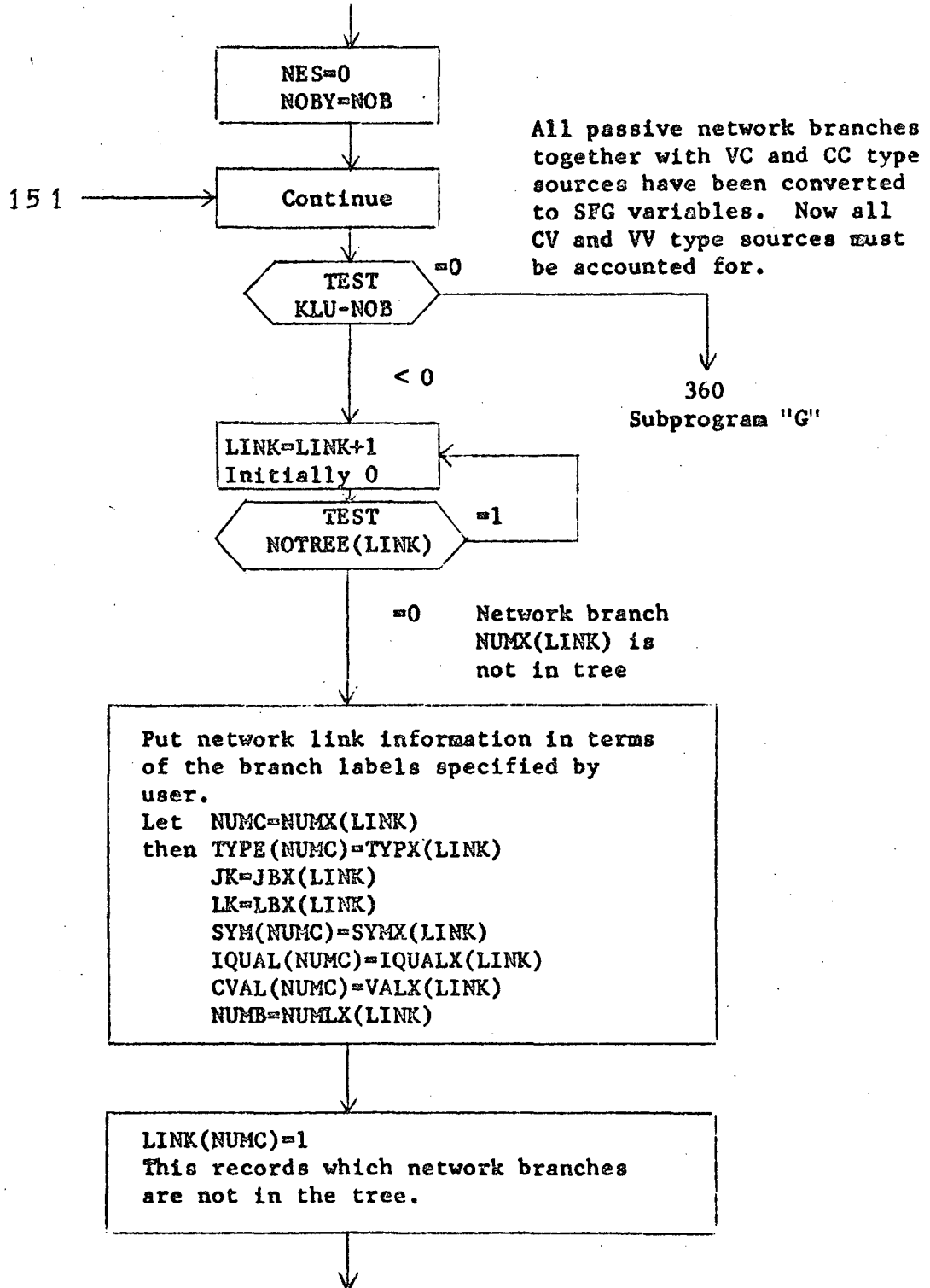
This program sets up tree branch information and creates a routing matrix and sign matrix for the tree.

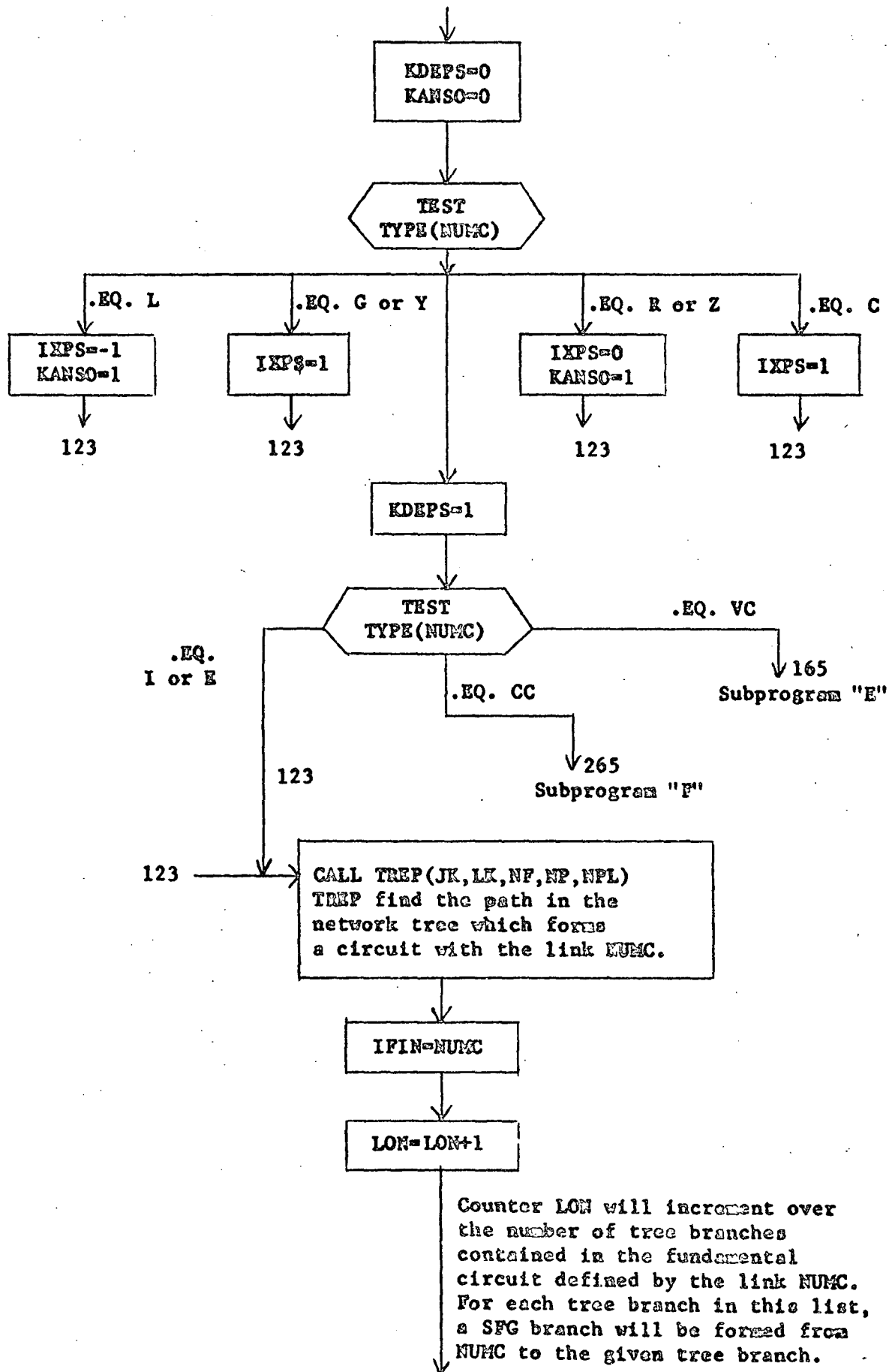


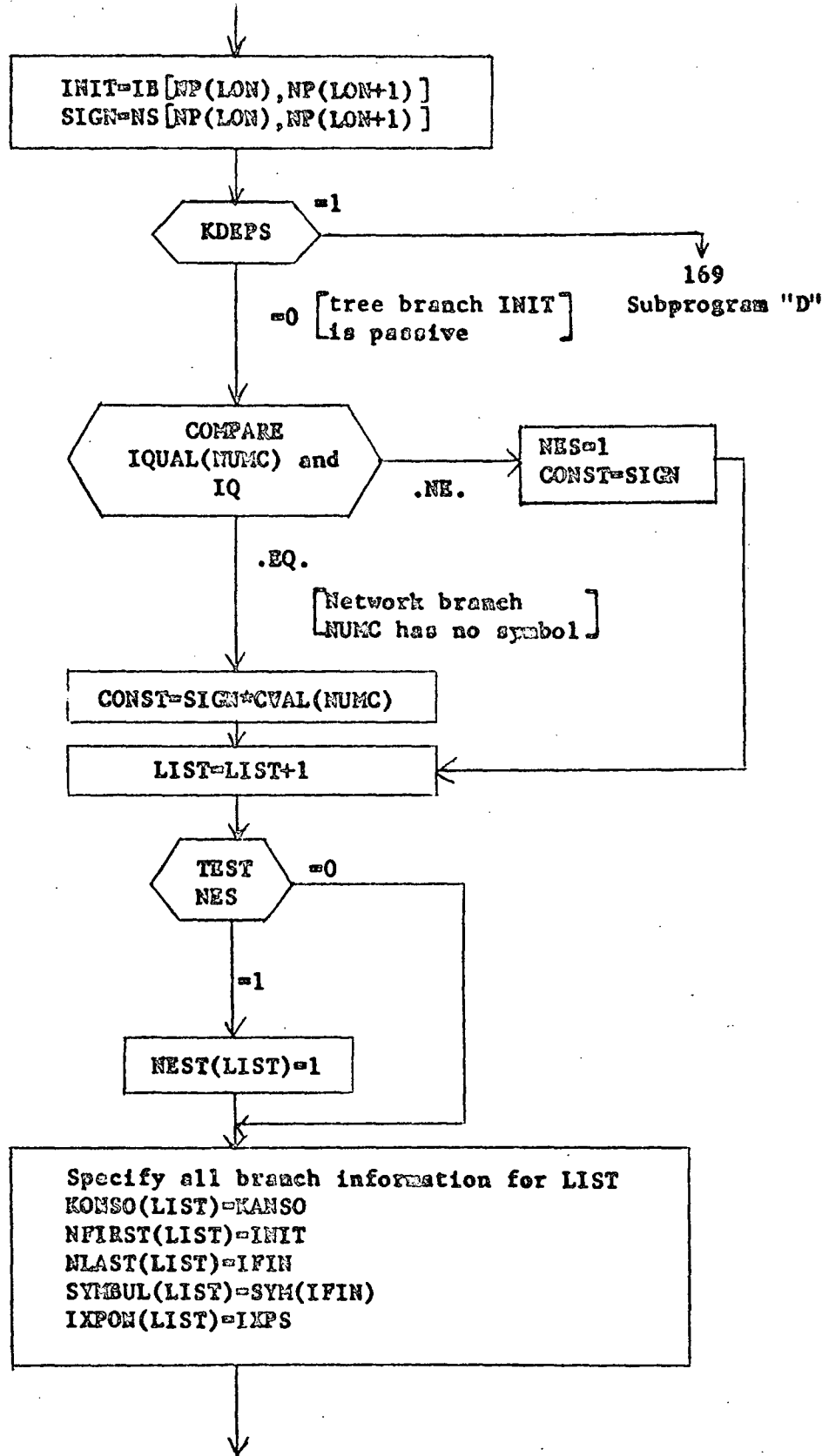


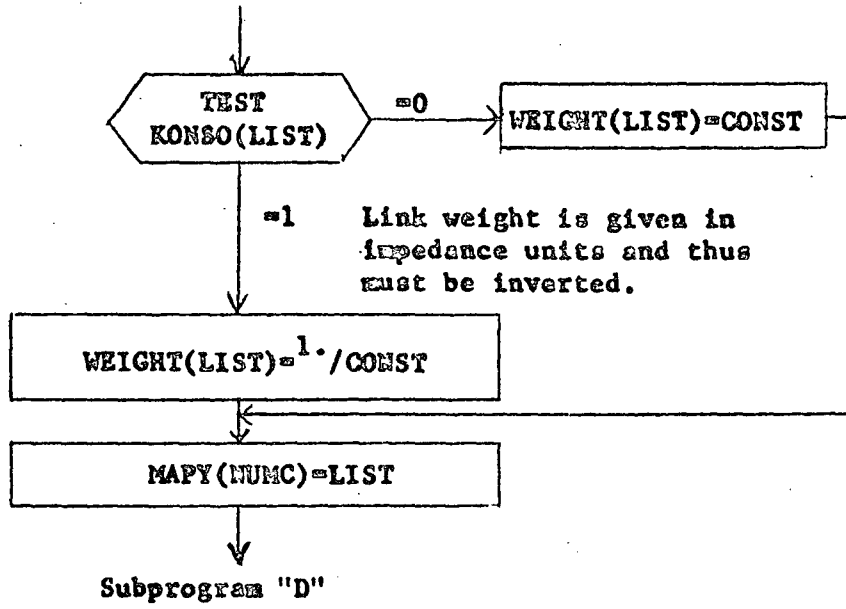
Subprogram "C"

This program generates SFG information from branch node to link node.



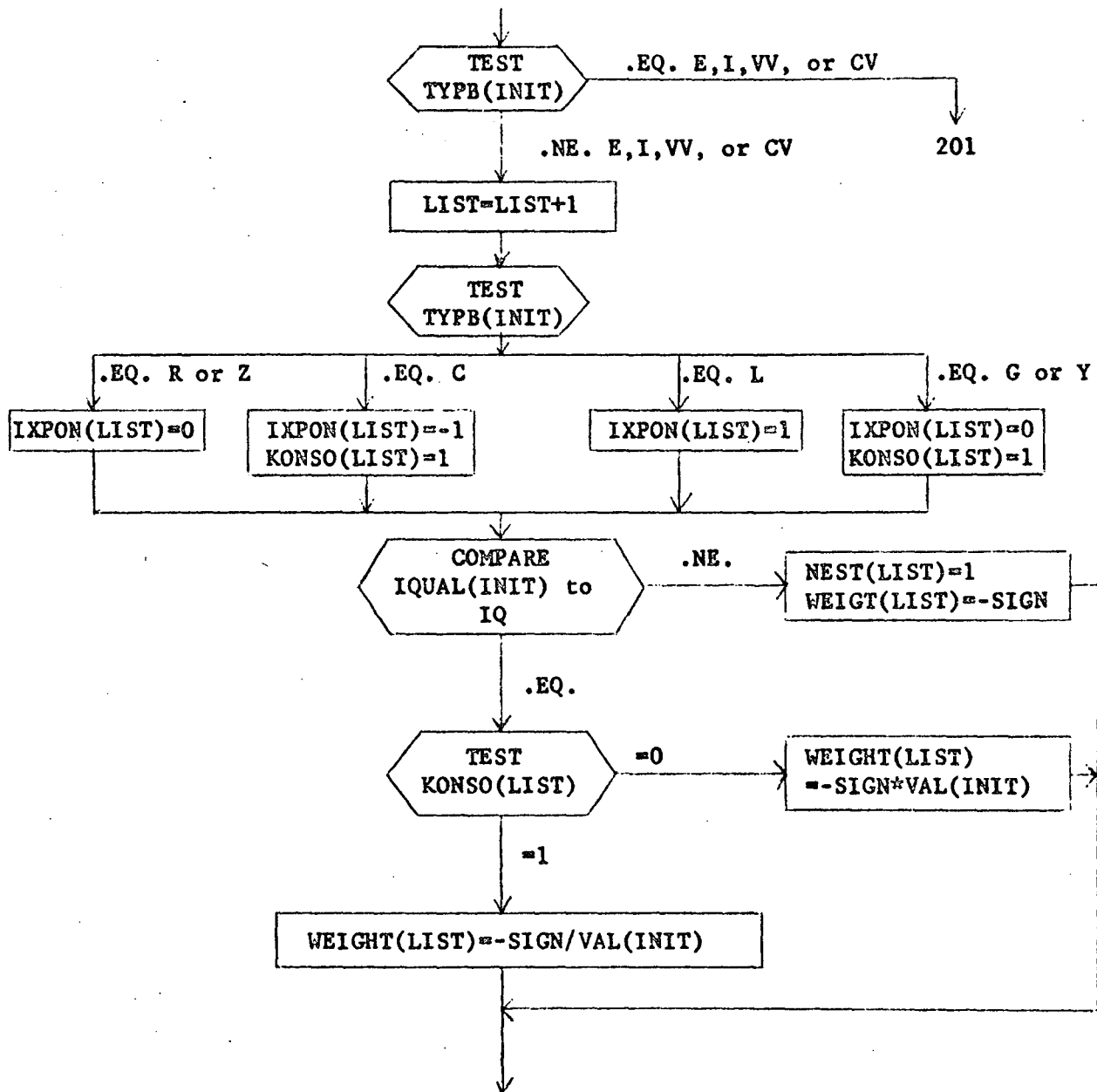


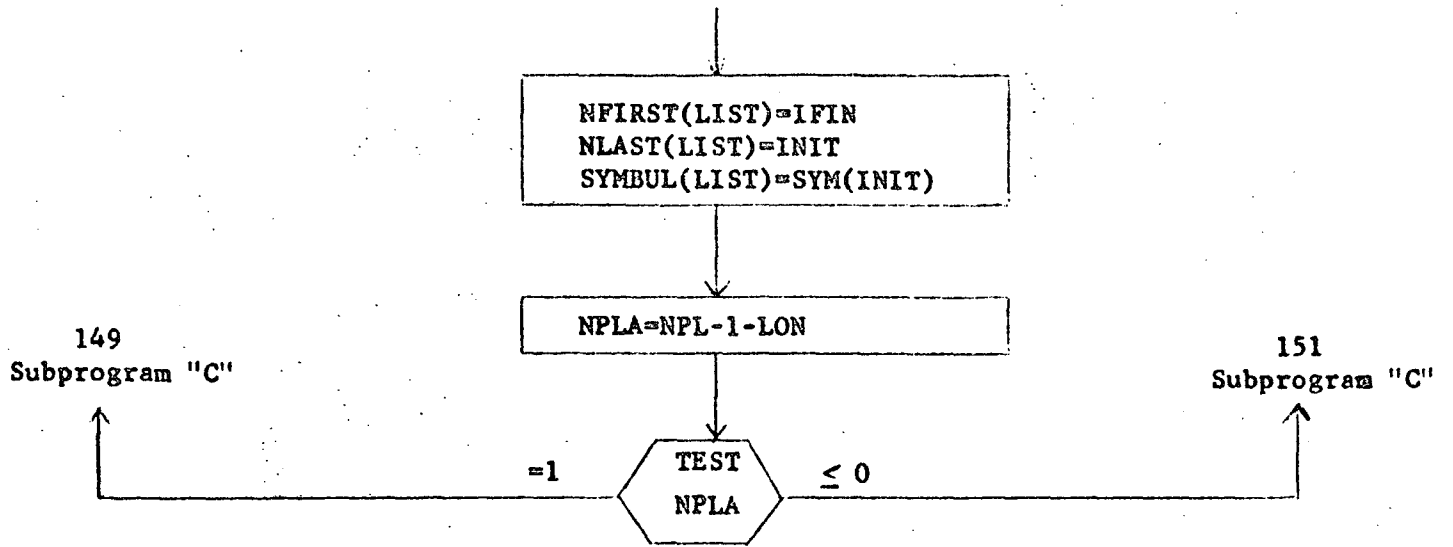




Subprogram "D"

This program generates SFG information from link node to branch node.

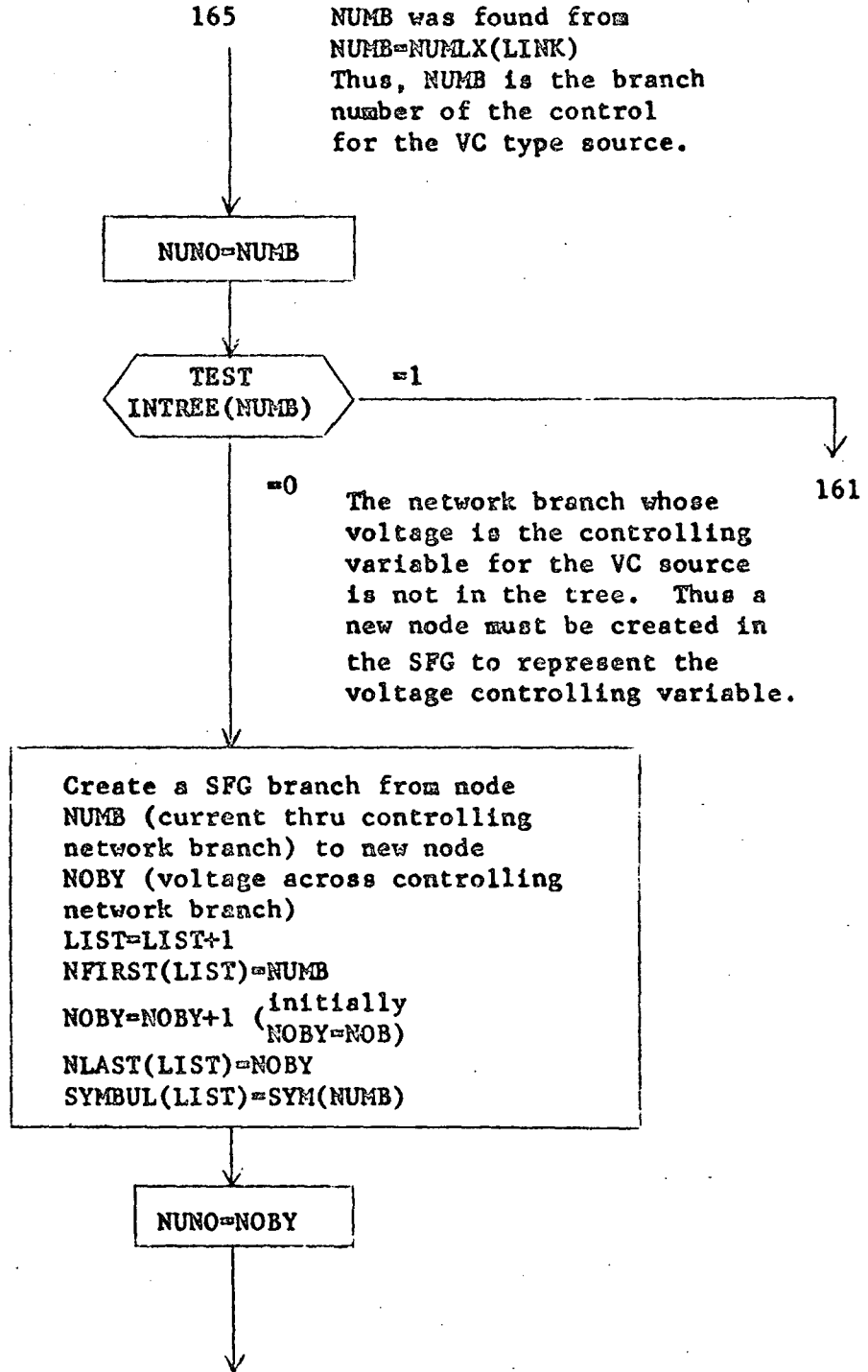


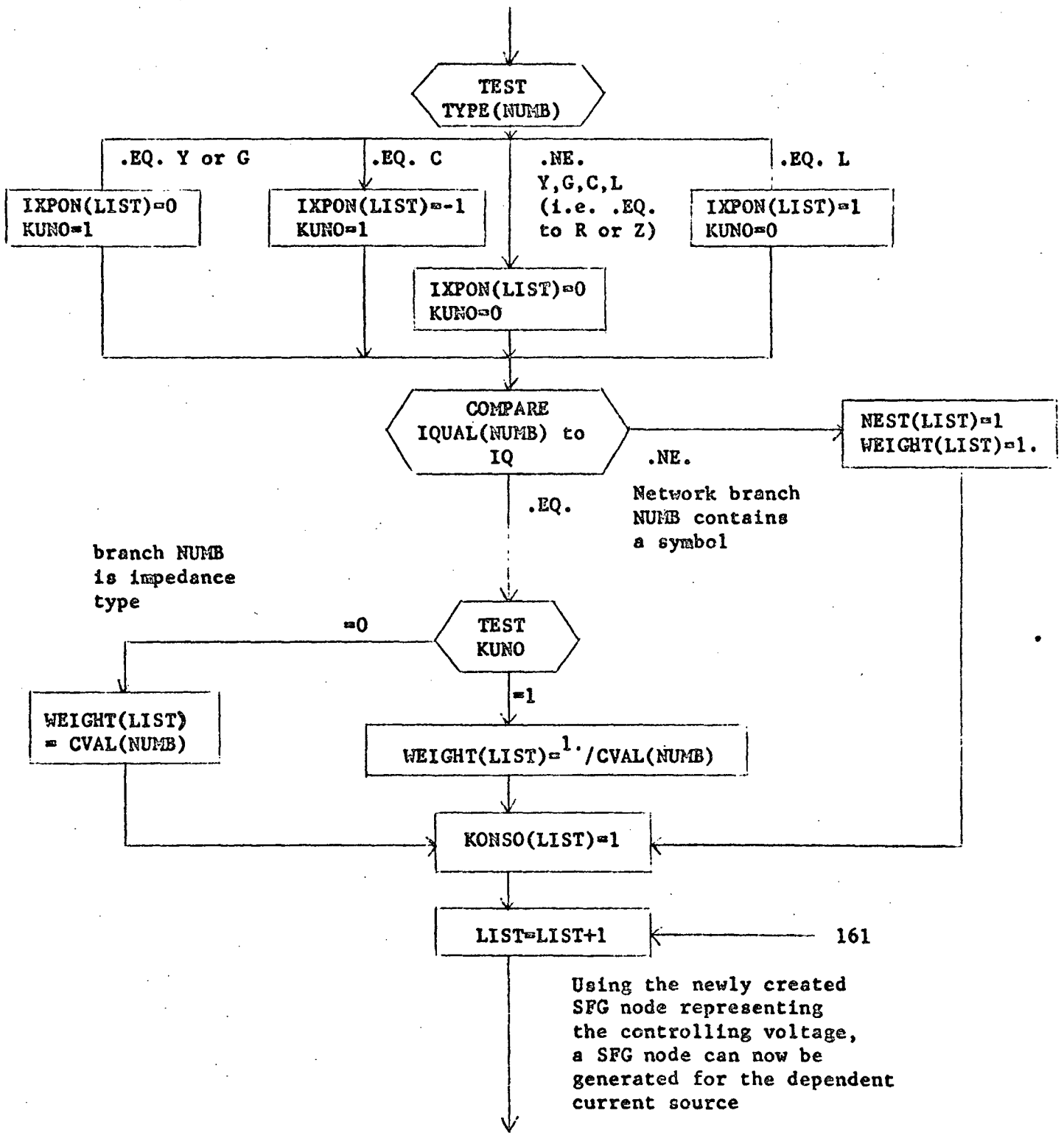




Subprogram "E"

This program sets up SFG information for VC type control sources.





NFIRST(LIST)=NUMO  
NLAST(LIST)=NUMC  
SYMBUL(LIST)=SYM(NUMC)  
IXPON(LIST)=0

COMPARE  
EQUAL(NUMC) to IQ

.EQ.

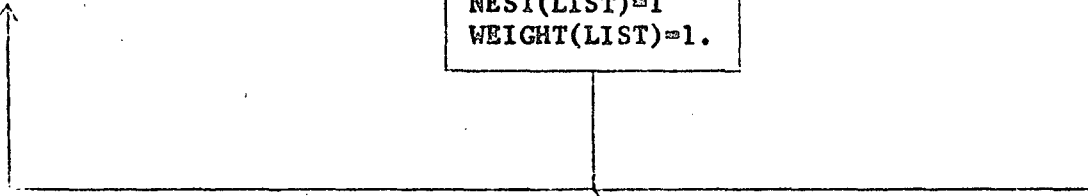
WEIGHT(LIST)=CVALU

.NE.

Network branch  
NUMC contains  
a symbol

NEST(LIST)=1  
WEIGHT(LIST)=1.

123  
Subprogram "C"



Subprogram "F"

This program sets up SFG information for CC type control sources.

265

The structure of subprogram "F" is completely analogous to subprogram "E".

123 Subprogram "C"

Subprogram "G"

This program sets up SFG information for VV type control sources.

360

This program is cycled over all voltage controlled voltage sources in the network. A list of these sources is maintained in the array

IVV(MI),MI=1,MO

Other than the above, the structure of subprogram "G" is completely analogous to subprogram "E".

460 Subprogram "H"

### Subprogram "H"

This program sets up SFG information for CV type control sources.

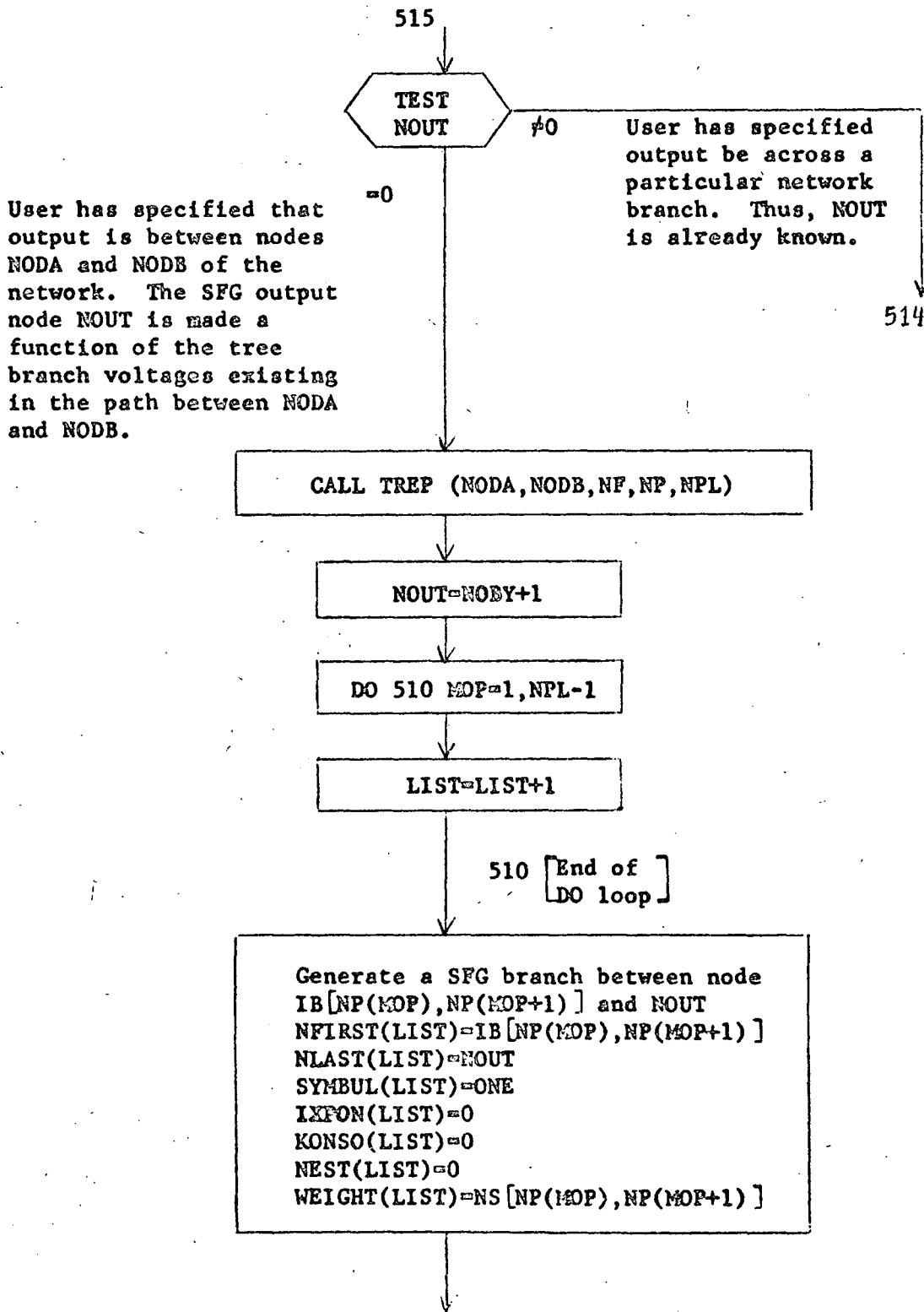
This program is cycled over all current controlled voltage sources in the network. A list of these sources is maintained in the array  
ICV(MI),MI=1,LO  
Other than the above, the structure of subprogram "H" is completely analogous to subprogram "E"

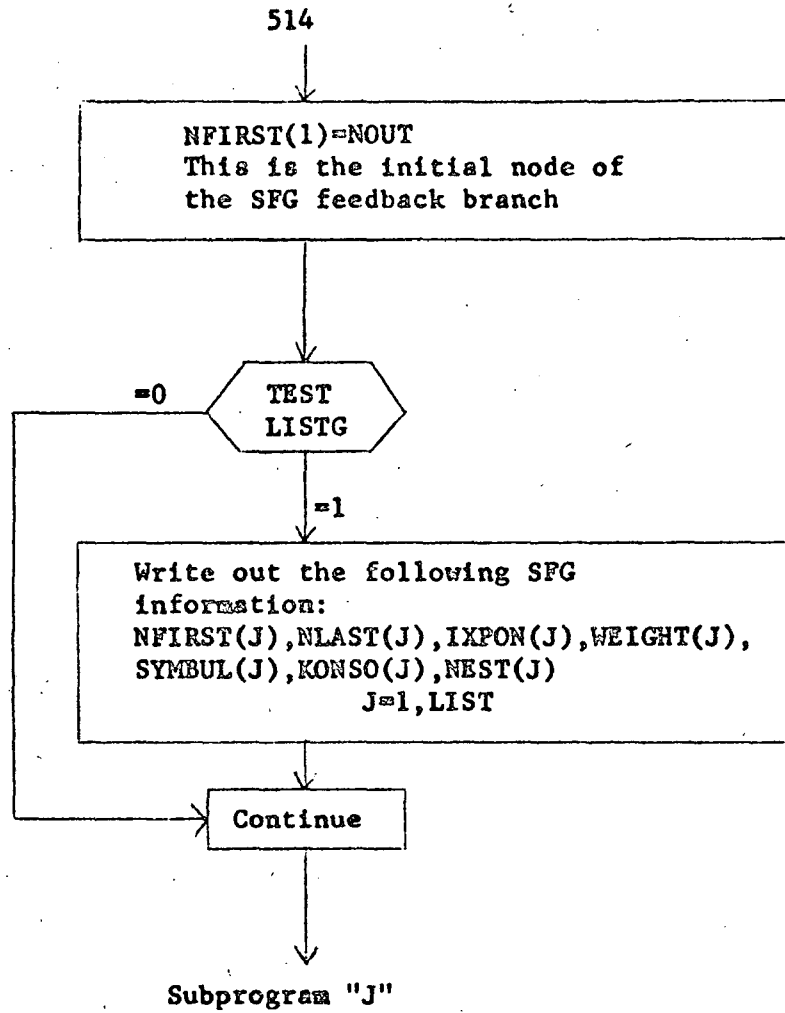
### 515 Subprogram "I"

The SFG is now complete except for the output node.

Subprogram "I"

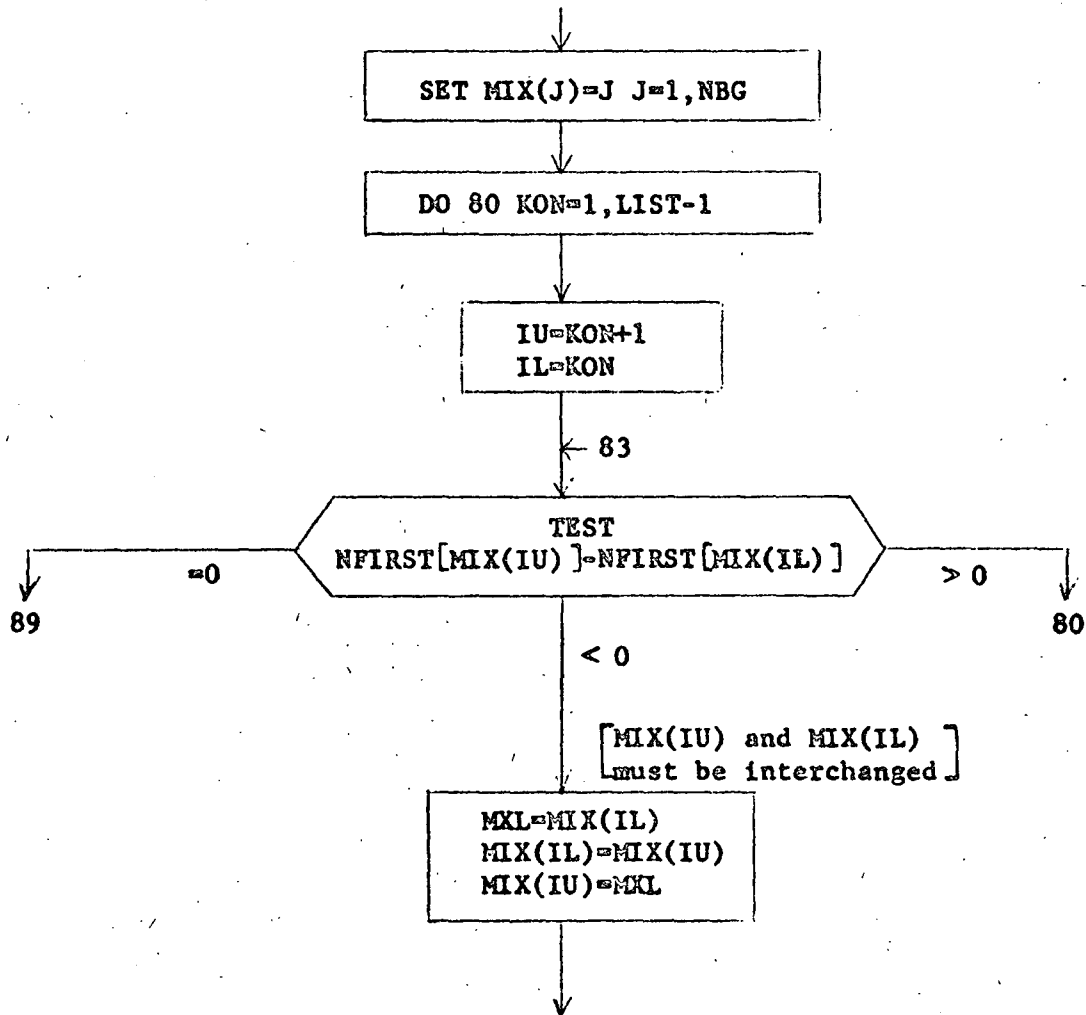
This program generates the output node of the SFG.



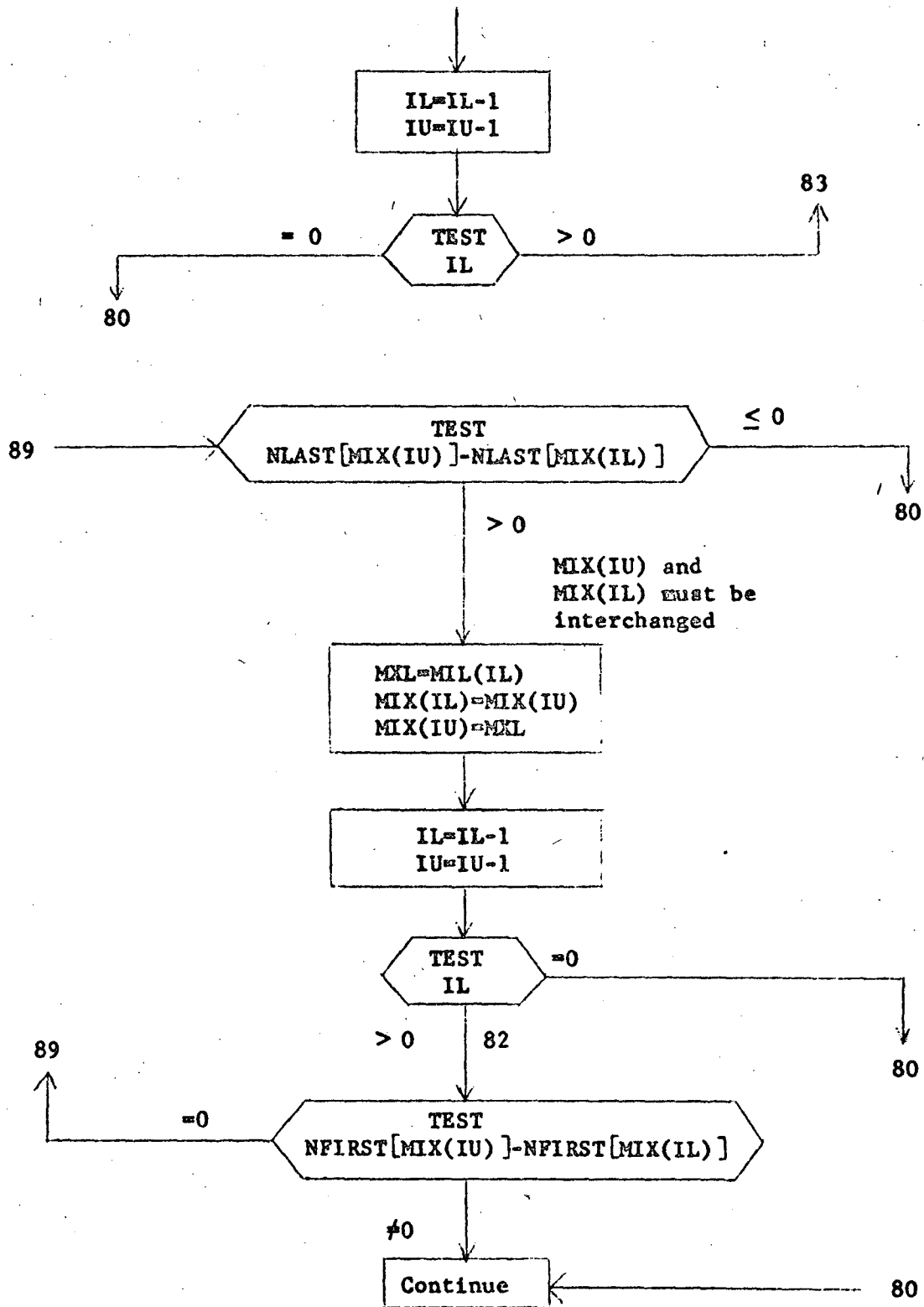


Subprogram "J"

This program orders SFG information for input to main program. That is, a mapping function MIX is found which reorders the SFG information so that the routing matrix  $N(J,K)$  as calculated in MAIN will automatically have its entries decrease as K increases.

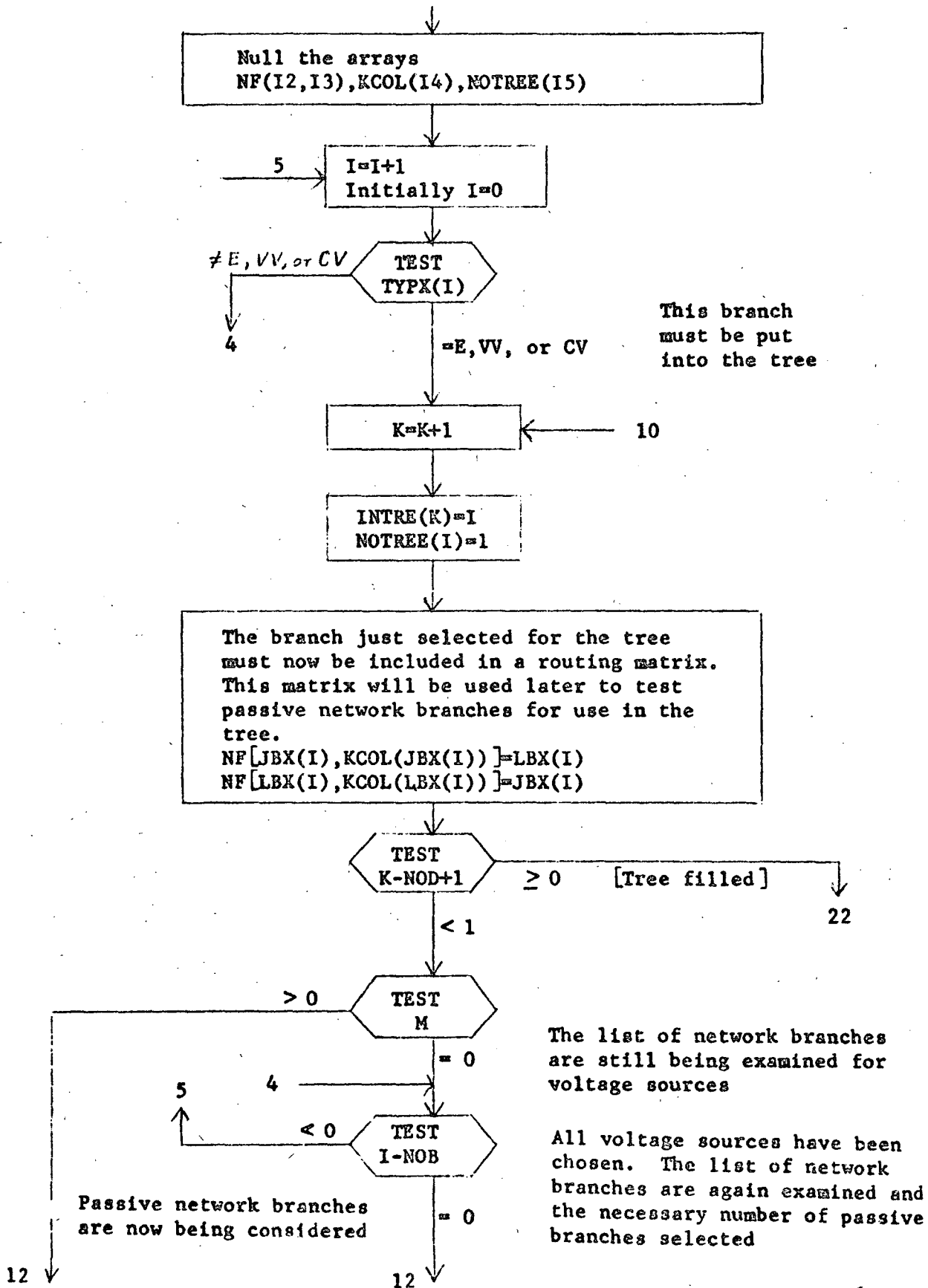


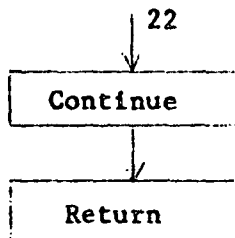
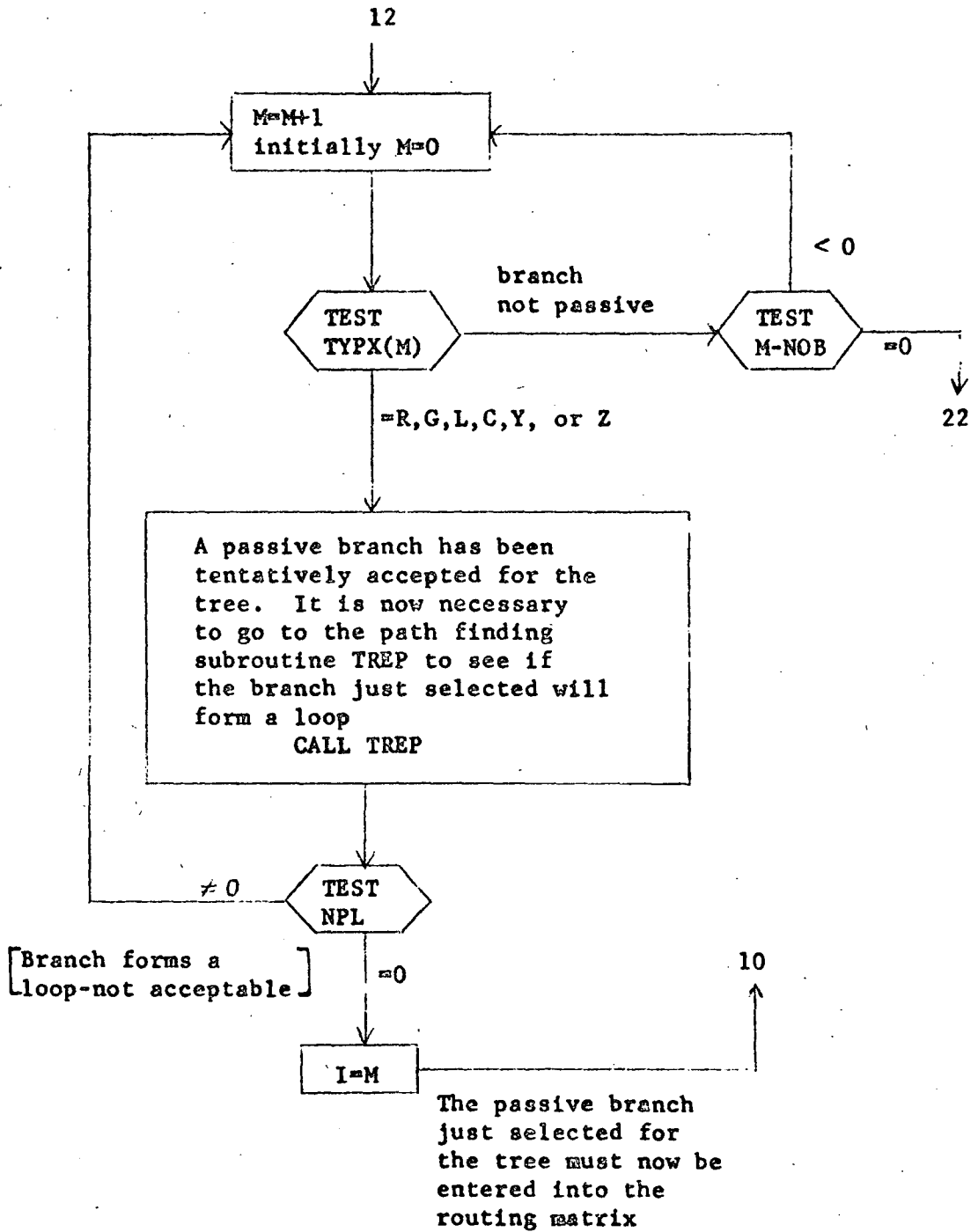




Subroutine FTREE(TYPX,JBX,  
LBX,INTRE,NOTREE,NOD,NOB)

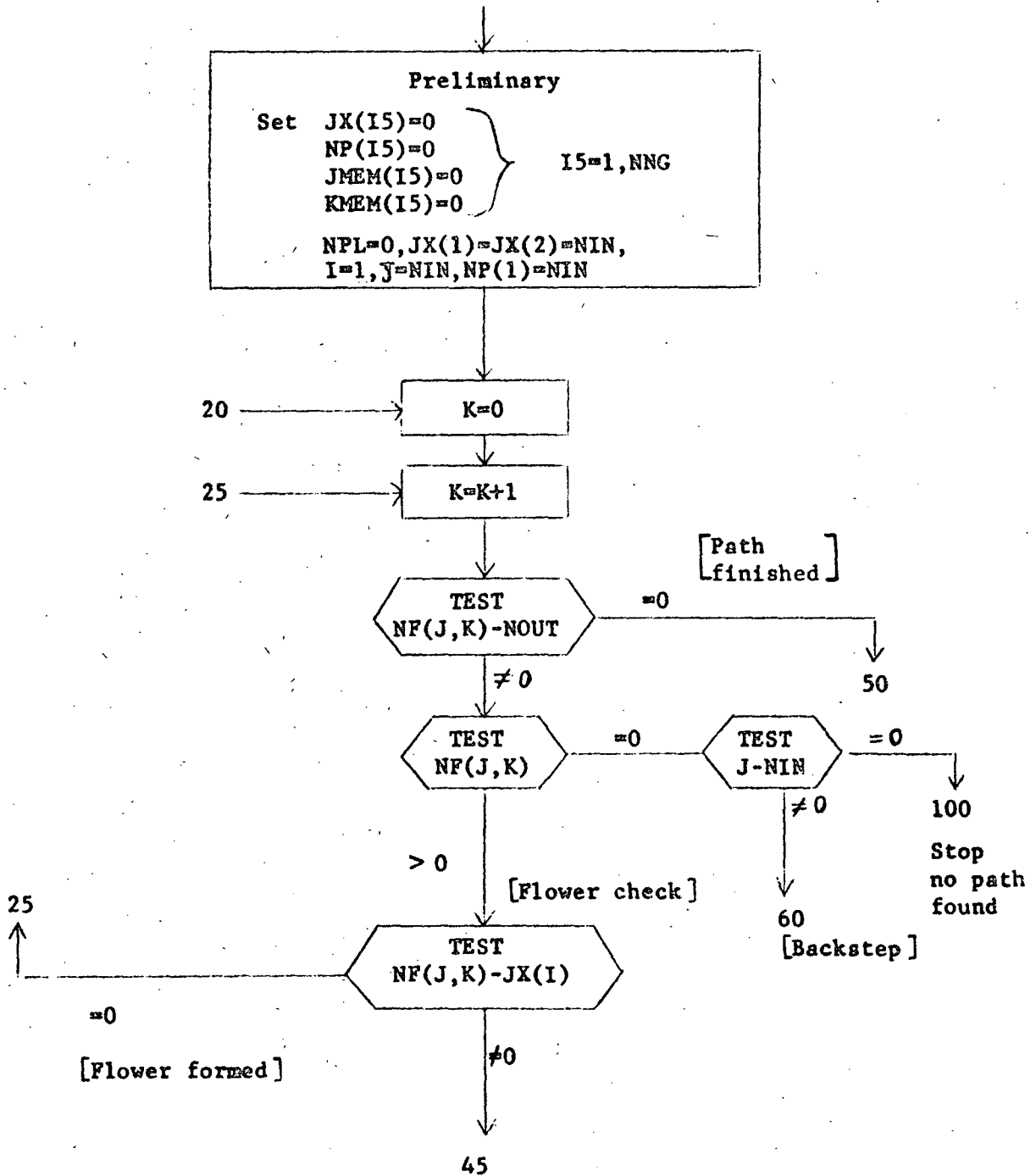
This program finds a tree of the network to use in generating a SFG.

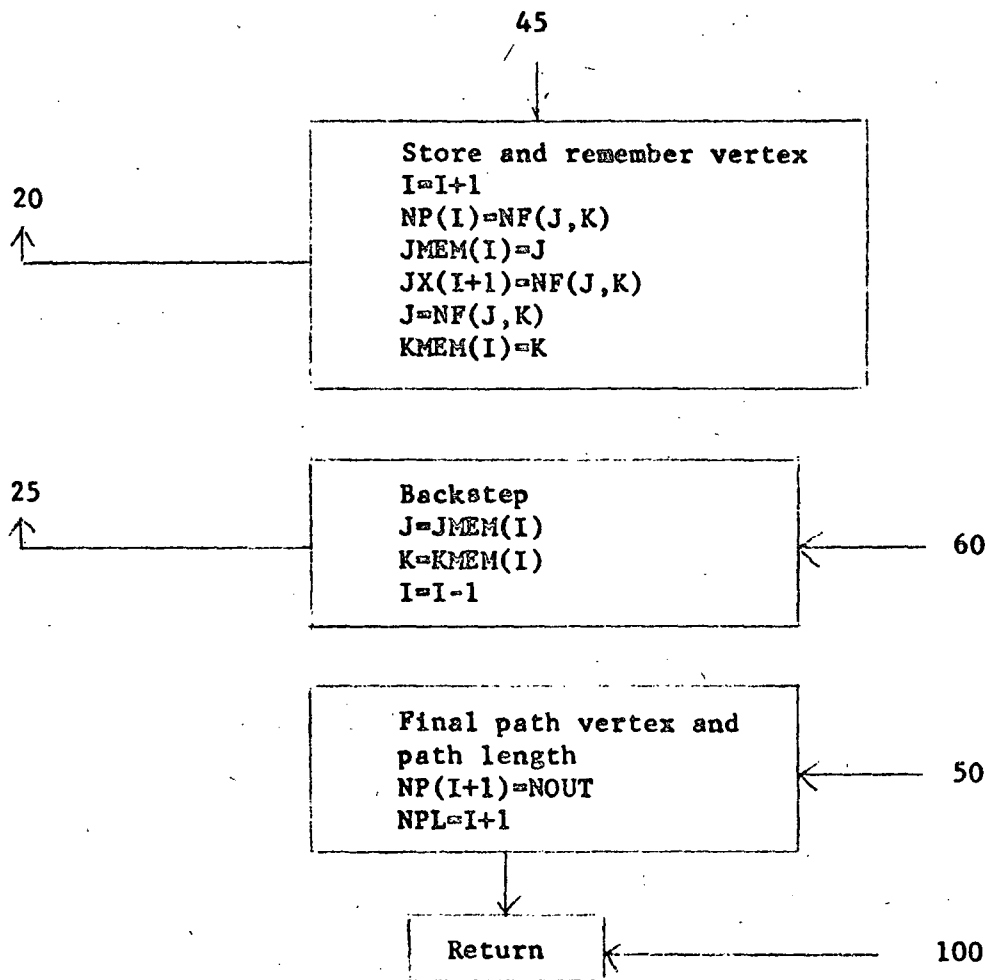




Subroutine TREP(NIN, NOUT, NF, NP, NPL)

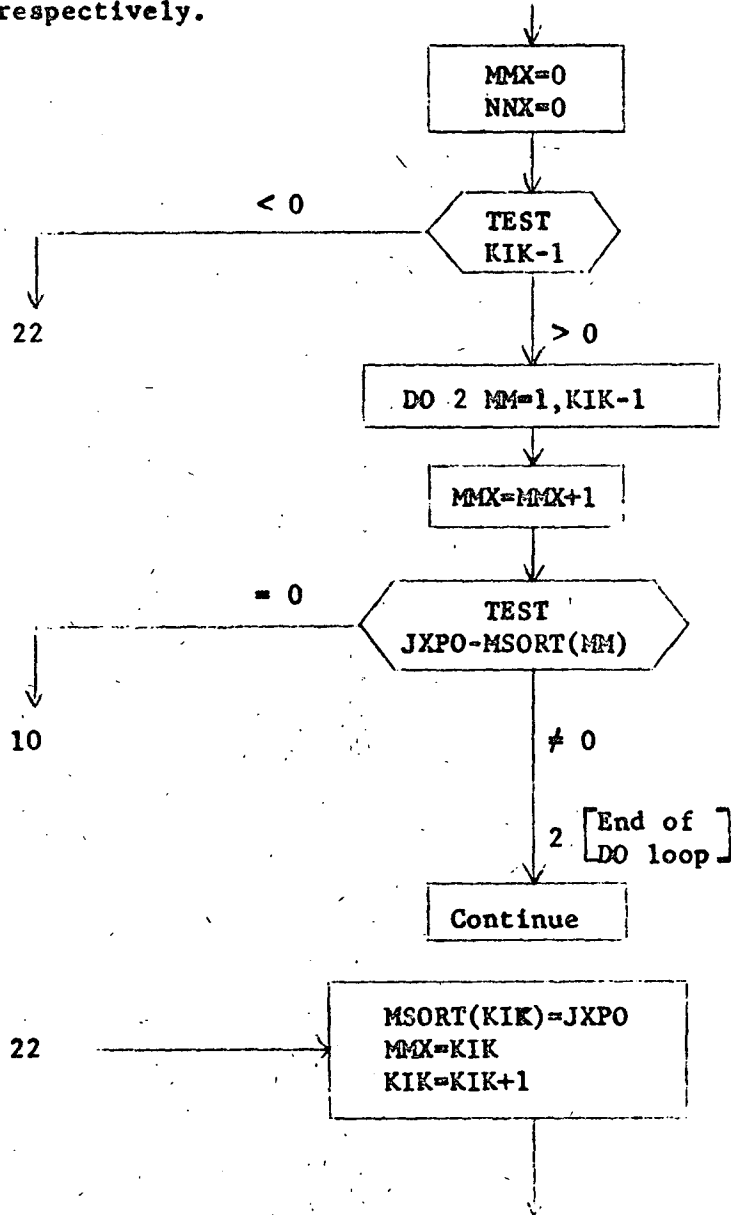
Given the routing matrix for the network tree with input and output nodes specified, this subroutine finds a node list representing the path between these two nodes.

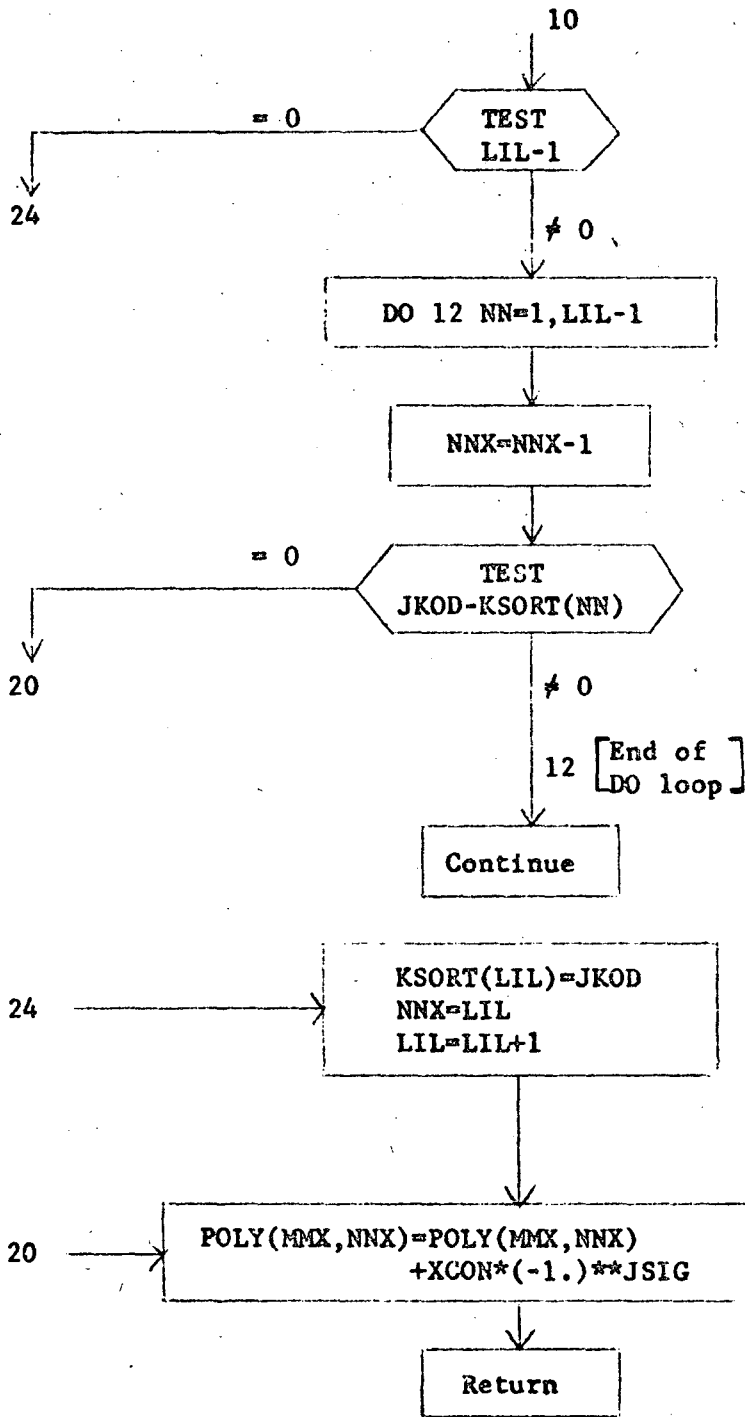




Subroutine ARRAY(JSIG,XCON,JXPO,JKOD,POLY,LIL,KIK)

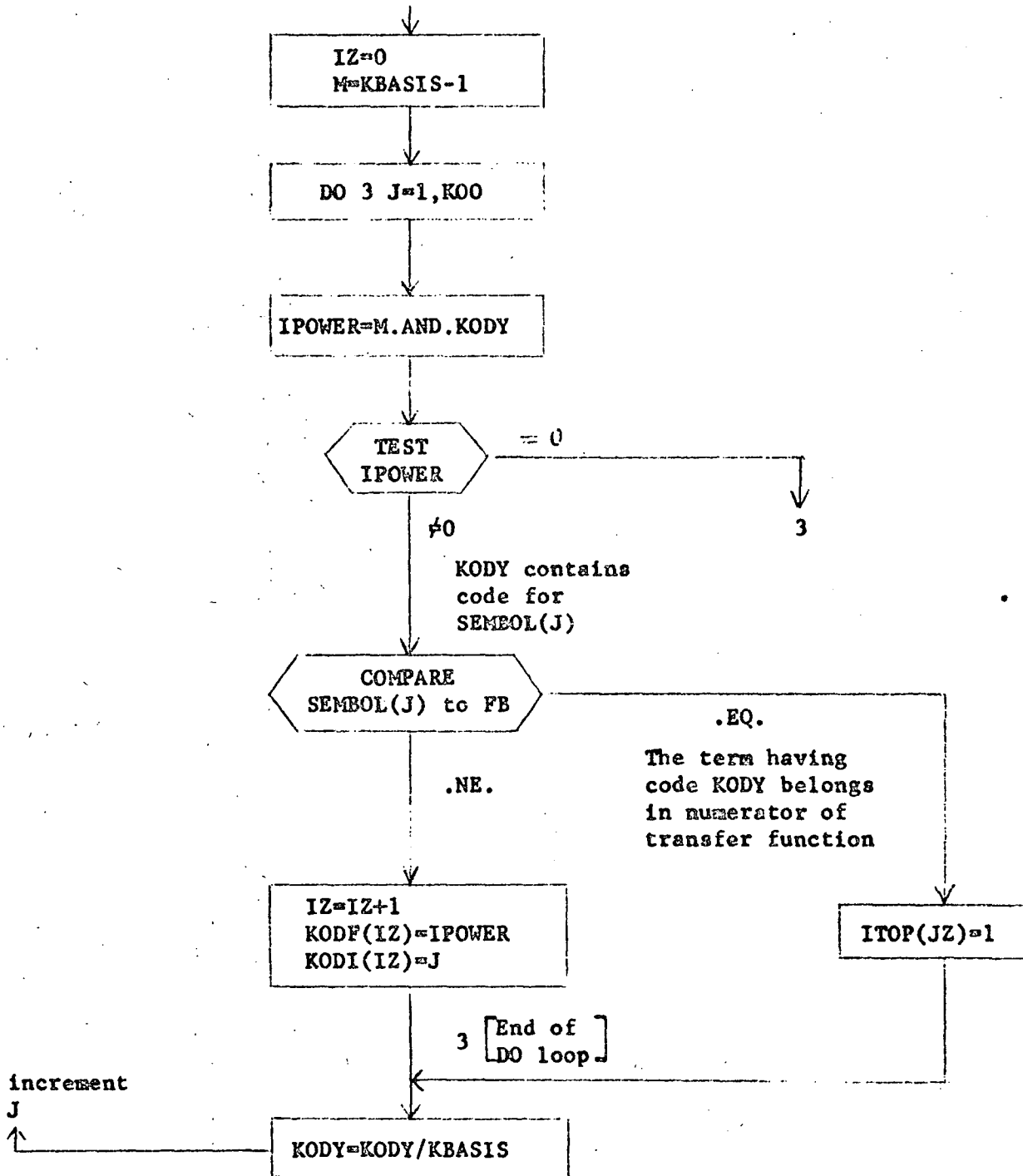
This subroutine takes the constant associated with each loop or nontouching combination of loops in the SFG and stores it in the matrix POLY. It does this by comparing the code and exponent of the given loop combination with the codes and exponents assigned to the columns and rows of POLY respectively.





Subroutine DECODE(KOO,KODY,IZ,FB,JZ,SEMBOL,KODF,KODI,ITOP,KBASIS)

This program decodes the composite symbol codes.





IV-3. Program Listing

```

C                                     ****SNAP****
C
C                                     THIS PROGRAM FINDS THE SYMBOLIC TRANSFER
C                                     FUNCTION OR IMPEDANCE FUNCTION OF A
C                                     LUMPED LINEAR TIME INVARIANT NETWORK
C
C
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NBN(DEFINED IN PROGRAM MAIN-1)
  DIMENSION LT(35),IG(35),SYMBOL(35)
  DIMENSION ILOW(35),NP(35),KODES(35),KONC(35)
  DIMENSION N(35,35),CONS(35,35),KODE(35,35),IXPO(35,35)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC N3G
  DIMENSION NFIRST(100),NLAST(100),IXPON(100),WEIGT(100)
  DIMENSION SYMBOL(100),MIX(100),CVAL(100)
  DIMENSION KONS0(100),NEST(100),TYPE(100)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NPAC
  DIMENSION CONST(300),KODET(300),IXPOT(300),MAPO(300)
  DIMENSION NQCTOT(300),NUP(300),JAC(300)
  DIMENSION N6CODE(300)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NTO, NSPT, AND NEXPS
  DIMENSION NA(150),NB(150)
  DIMENSION KONS(20),KODI(20),SEMBOL(20),KODE(20)
  DIMENSION MSORT(15),KSORT(150),POLYU(15,150)
  DIMENSION POLY(15,150),ITOP(150)
  DIMENSION SIMBON(150,10),SIMBOD(150,10)
  DIMENSION SEMPON(150,10),SEMPOD(150,10)

```

10  
20  
30  
40  
50  
60  
70  
80  
90  
100  
110  
120  
130  
140  
150  
160  
170  
180  
190  
200  
210  
220  
230  
240  
250  
260  
270  
280  
290  
300  
310  
320  
330  
340  
350  
360  
370  
380

```

C***** 390
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK 400
C CHARACTERISTICS NRI, NCI, NEON, AND NRS 410
  DIMENSION ISET(15,100),NOTCH(1200),STAR(9) 420
C***** 430
  COMMON SEMPOD,SEMPD,POLY 440
  COMMON/C1/MSORT,KSORT 450
  COMMON/C2/NNG,NBG 460
  COMMON/C3/NEXPS,NT0 470
  COMMON/C4/NSPT 480
  EQUIVALENCE (IXPO(1,1),NOTCH(1),SIMBON(1,1)) 490
  EQUIVALENCE (CONS(1,1),ISET(1,1),SIMBOD(1,1)) 500
  EQUIVALENCE (KODE(1,1),POLYU(1,1)) 510
  DATA DASH/2H // 520
  DATA FB,SH/3H FH,3H 1 / 530
  DATA STAR(1),STAR(2),STAR(3)/3H ,3H**2,3H**3/ 540
  DATA STAR(4),STAR(5),STAR(6)/3H**4,3H**5,3H**6/ 550
  DATA STAR(7),STAR(8),STAR(9)/3H**7,3H**8,3H**9/ 560
  DATA ONE/3H 1/ 570
C 580
C PROGRAM MAIN--1 590
C PRELIMINARY INPUT INFORMATION 600
C 610
C 620
C 630
C 640
C NRN=NUMBER OF BRANCHES IN NETWORK 650
C NBG=NUMBER OF BRANCHES OF SFG 660
C NTO=NUMBER OF TERMS IN OUTPUT 670
C NSPT=NUMBER OF SYMBOLS PER TERM 680
C NEXPS=NUMBER OF DIFFERENT POWERS OF S 690
C NPAC=NUMBER OF PATHS PLUS CIRCUITS 700
C NRI=MAXIMUM NUMBER OF NONTOUCHING LOOPS 710
C NCI=MAXIMUM NUMBER OF LOOPS NOT TOUCHING ANY GIVEN LOOP 720
C NEON=NUMBER OF NONTOUCHING PAIRS OF LOOPS 730
C NRS=NUMBER OF REPEATED SYMBOLS(NUMBER OF NETWORK 740
C ELEMENTS ASSIGNED SAME SYMBOL) 750
  NBN=35 760
  NBG=100 770
  NTO=150 780
  NSPT=20 790
  NEXPS=15 800
  NPAC=300 810
  NRI=15 820
  NCI=100 830
  NEON=1200 840
  NRS=9 850
C NSPTU=NUMBER OF SYMBOLS IN NUMERATOR OF EACH TERM 860
C NRTG=NUMBER OF BRANCHES IN TREE OF SFG 870
C NNG=NUMBER OF NODES IN SFG 880
  NNG=NRN 890
  NSPTU=NSPT/2 900
  NRTG=NBN 910
  CALL SECOND (TCOMP) 920
  WRITE(6,1600) TCOMP 930
1600 FORMAT (1X,40H COMPILATION TIME IN SECONDS,F15.8/) 940
1111 CONTINUE 950
  WRITE (6,519) 960
 519 FORMAT(1H1) 970
  CALL SECOND (TSTART) 980
C THE NEXT 6 CARDS ARE FOR PROBLEM IDENTIFICATION ON THE 1ST DATA CARD 990

```

```

1150 READ(5,1150) (WEIGT(J),J=1,72)
1150 FORMAT (72A1)
IF(EOF,5)11111,11112
11111 STOP
11112 CONTINUE
WRITE (6,1160) (WEIGT(J),J=1,71)
1160 FORMAT(1X,71A1//)
DO 1151 J=1,72
1151 WEIGT(J)=0.
READ(5,1240) NOD,NOB,KBASIS,LISIG,LISIC,LISIP
1240 FORMAT (3I5,5X,3I1)
IF(KBASIS)1357,1357,1358
1357 KBASIS=8
1358 CONTINUE
READ(5,1) NIN,NOUT,NODA,NODB
1 FORMAT (4I5)
WRITE(6,720) NOD
720 FORMAT(2X,10HNUMBER OF NODES=,I3)
WRITE(6,721)NOB
721 FORMAT(2X,10HNUMBER OF BRANCHES=,I3)
IF(LISIG)722,723,722
722 CONTINUE
C LIST SFG
723 IF(LISIC)725,725,724
724 CONTINUE
C LIST ALL CIRCUITS
725 IF(LISIP)726,726,727
727 WRITE(6,728)
728 FORMAT(2X,24HLIST ALL PATHS FROM NODE, I3, 2X, 7HTO NODE, I3)
726 WRITE(6,729) NIN
729 FORMAT(2X,25HELEMENT NUMBER OF SOURCE=,I3)
IF(NOUT)1802,1802,1804
1804 CONTINUE
WRITE(6,730) NOUT
730 FORMAT (2X,38HELEMENT NUMBER ASSOCIATED WITH OUTPUT=,I3)
GO TO 1806
1802 CONTINUE
WRITE(6,731) NODA
731 FORMAT(2X,33HPOSITIVE OUTPUT VOLTAGE TERMINAL=,I3)
WRITE(6,732) NODB
732 FORMAT(2X,33HNEGATIVE OUTPUT VOLTAGE TERMINAL=,I3)
1806 CONTINUE
WRITE(6,850)KBASIS
850 FORMAT(2X,22HBASE FOR SYMBOL CODES=,I4)

C
C PROGRAM MAIN--2
C TAKE SFG BRANCH INFORMATION AS FOUND
C BY SUBROUTINE SFG AND GENERATE
C (1) ROUTING MATRIX INFORMATION
C N(J,K), AND LI(J)
C (2) SFG BRANCH VALUES IXPO(J,L), CONS(J,L),
C KOE(J,L) WHERE J=NFIRST(I), L=NLAST(I), AND
C I=BRANCH NUMBER
C TOGETHER WITH THE SYMBOL SEMBOL(K), K=1,2,...,MI
C
C CALL SUBROUTINE TO FORMULATE THE SIGNAL FLOW GRAPH, SFG
CALL SFG (NFIRST,NLAST,IXPON,WEIGT,SYMBOL,KONSO,MIX,NEST,LIST,
ININ,NOUT,NOD,NOB,LISIG,NODA,NODB)

```

1000  
1010  
1020  
1030  
1040  
1050  
1060  
1070  
1080  
1090  
1100  
1110  
1120  
1130  
1140  
1150  
1160  
1170  
1180  
1190  
1200  
1210  
1220  
1230  
1240  
1250  
1260  
1270  
1280  
1290  
1300  
1310  
1320  
1330  
1340  
1350  
1360  
1370  
1380  
1390  
1400  
1410  
1420  
1430  
1440  
1450  
1460  
1470  
1480  
1490  
1500  
1510  
1520  
1530  
1540  
1550  
1560  
1570  
1580  
1590  
1600

IF (NOR) 1111, 1111, 1920	1610
1920 CONTINUE	1620
CALL SECOND (T2)	1630
WRITE (6, 932)	1640
TSFG=T2-TSTART	1650
WRITE (6, 1602) TSFG	1660
1602 FORMAT(1X, 2(MTIME FOR FORMULATING SIGNAL/ 122H FLOW GRAPH IN SECONDS, F15.3/)	1670
IRO=0	1680
K0=0	1690
MICH=1	1700
K=0	1710
MG=1	1720
JLAS=1	1730
NCIR=1	1740
ININ=NIN	1750
INOUT=NOUT	1760
K00=0	1770
DO 301 INK=1, NSPT	1780
301 KONS(INK)=0	1790
DO 300 INK=1, NNG	1800
300 IG(INK)=0	1810
	1820
	1830
C FIND IXPO(J, L), CONS(J, L)	1840
GO TO 307	1850
305 MG=KBASIS*MG	1860
MICH=MICH+1	1870
307 IRO=IRO+1	1880
IF (LIST=IRO) 19, 4, 4	1890
4 CONTINUE	1900
LOB=MIX(IRO)	1910
J=NFIRST(LOB)	1920
L=NLAST(LOB)	1930
IXPO(J, L)=IXPON(LOB)	1940
CONS(J, L)=WEIGT(LOB)	1950
	1960
	1970
C FIND ROUTING MATRIX	1980
8 IF (J.EQ.JLAC) GO TO 10	1990
LT(JLAS)=K	2000
K1=K+1	2010
IF (JLAS=NIN) 28, 27, 28	2020
27 N(JLAS, K1)=-1	2030
GO TO 29	2040
28 N(JLAS, K1)=0	2050
29 JLAS=JLAS+1	2060
K=0	2070
GO TO 8	2080
10 K=K+1	2090
N(J, K)=L	2100
	2110
C FIND KODE(J, L) AND SEMBOL(K00)	2120
SEMBOL(IRO)=SEMBOL(LOB)	2130
MODE=NEST(LOB)	2140
IF (MODE) 335, 316, 335	2150
335 IF (IG(L)) 5, 960, 5	2160
5 KODE(J, L)=IG(L)	2170
GO TO 307	2180
960 CONTINUE	2190
KPU=IRO-1	2200
IF (KPU) 953, 953, 315	2210
315 DO 952 KP=1, KPU	

```

IF (SMBOL (IBO) .NE. SMBOL (KP)) GO TO 952
LORX=MIX (KP)
IF (KONSO (LOR) -KONSO (LOBX)) 952,956,952
956 LX=NLAST (LORX)
KODE (J,L)=IC (LX)
GO TO 307
952 CONTINUE
IF (SMBOL (IBO) .EQ. ONE) GO TO 316
953 IG (L)=MG
KOO=KOO+1
SEMBOL (KOO)=SMBOL (IBO)
KODE (J,L)=IC (L)
IF (KONSO (LOR)) 3,3,2
2 KONS (KOO)=1
3 CONTINUE
GO TO 305
316 KODE (J,L)=0
GO TO 307
19 LT (JLAS)=K
K11=K+1
N (JLAS,K11)=0

```

```

C          PROGRAM MAIN--3
C          NULL CERTAIN ARRAYS, SET COUNTERS, AND DEFINE
C          A CODE FOR EACH NODE OF THE SFG

```

```

MPL=0
KIK=1
LIL=1
DO 601 KAM=1,NEXPS
DO 601 KIM=1,NT0
601 POLY (KAM,KIM)=0
DO 602 KP1=1,NEXPS
602 MSORT (KP1)=0
DO 950 K02=1,NSPT
950 KODI (K02)=0
DO 603 KP2=1,NT0
603 KSORT (KP2)=0
IR=1
NFIR=1
KNO=0
KODES (1)=1
DO 2000 JS=2,NNG
2000 KODES (JS)=2*KODES (JS-1)
IF (LISTP) 175,175,1116
1116 WRITE (6,170) NIN,NOUT
170 FORMAT (17H PATHS FROM NODE 12.9H TO NODE 12//)
WRITE (6,1905)
1905 FORMAT (5X,17HNO. NODE LIST)
175 CONTINUE
IF (LISTP) 1113,1113,23
1113 K3=LT (NIN)+1
N (NIN,K3)=0
K2=LT (1)+1
N (1,K2)=-1
NIN=1
NOUT=1
KLAS=0
24 NFIR=0

```

222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282

IF(LISTC)1209,1209,1219	2830
1219 CONTINUE	2840
WRITE(6,177)	2850
177 FORMAT(1X,8MCIRCUITS//)	2860
WRITE(6,1905)	2870
KNO=0	2880
1209 CONTINUE	2890
	2900
	2910
C PROGRAM MAIN--4	2920
C PATH-FINDING ALGORITHM	2930
C IN ADDITION, STEP PF7 CALCULATES THE COMPOSITE	2940
C CODE, CONSTANT, AND EXPONENT OF THE PATH	2950
	2960
	2970
C PF1 (PRELIMINARY)	2980
DO 1112 IZO=1,NNG	2990
1112 IFLOW(IZO)=0	3000
DO 31 I1=1,NNG	3010
31 KONC(I1)=1	3020
NOP=KLAS	3030
KLAS=0	3040
23 I=2	3050
J=NIN	3060
NP(1)=NIN	3070
IFLOW(NIN)=1	3080
IFLOW(NOUT)=-1	3090
C	3100
25 K=KONC(J)	3110
C	3120
C PF2 (FIND NEXT NODE)	3130
NP(I)=N(J,K)	3140
C	3150
C PF3 (TEST ROUTING MATRIX)	3160
IF(N(J,K))100,60,34	3170
C	3180
C PF4 (TEST FOR FLOWER)	3190
34 NJK=N(J,K)	3200
IF(IFLOW(NJK))50,38,26	3210
26 KONC(J)=KONC(J)+1	3220
GO TO 25	3230
	3240
C PF5 (PREPARE FOR NEXT NODE)	3250
38 J=NP(I)	3260
IFLOW(J)=1	3270
I=I+1	3280
GO TO 25	3290
C	3300
C PF6 (BACKSTEP)	3310
60 IFLOW(J)=0	3320
KONC(J)=1	3330
J=NP(I-2)	3340
KONC(J)=KONC(J)+1	3350
I=I-1	3360
GO TO 25	3370
C	3380
C PF7 (FINISH PATH)	3390
50 KONC(J)=KONC(J)+1	3400
KLAS=KLAS+1	3410
C	3420
C FIND CODE FOR NODE PATH	3430

```

NPCODE(IR)=A
ISU=I-1
DO 2002 IS=1,ISU
NODS=NP(IS)
2002 NPCODE(IR)=NPCODE(IR)+KODES(NODS)
C
C CALL ARRAY AND WRITE
IF(NFIR.EQ.1)GO TO 179
IF(LISTC)1208,1208,1206
1206 CONTINUE
KRU=I
179 KNO=KNO+1
WRITE(6,110) KNO,(NP(KR),KR=1,KRU)
110 FORMAT (4X,13,6X,35I3)
1208 CONTINUE
C
IF(NFIR.EQ.1)GO TO 320
KODET(IR)=0
CONST(IR)=1.
IXPOT(IR)=0
IEND=I
DO 319 KEW=2,IEND
JNODE=NP(KEW-1)
LNODE=NP(KEW)
KODET(IR)=KODET(IR)+KODE(JNODE,LNODE)
CONST(IR)=CONST(IR)*CONS(JNODE,LNODE)
IXPOT(IR)=IXPOT(IR)+IXPO(JNODE,LNODE)
319 CONTINUE
CONEW=CONST(IR)
IXNEW=IXPOT(IR)
KONEW=KODET(IR)
CALL ARRAY(1,CONEW,IXNEW,KONEW,POLY,LIL,KIK)
320 CONTINUE
C
C
IR=IR+1
IF(IR=NPAC)1361,1361,1360
1360 WRITE(6,1362)
1362 FORMAT(1X,39HNO. OF CIRCUITS EXCEEDS LIMIT-INCREASE ,
126HDIMENSIONS CONTAINING NPAC)
1361 CONTINUE
GO TO 25
C
PROGRAM MAIN--5
C
MODIFY THE SFA BY REMOVING EVERY BRANCH CONNECTED
C
TO THE NODE THROUGH WHICH ALL CIRCUITS HAVE JUST
C
BEEN FOUND
C
100 I3=0.
IF(NCIR-1)2010,102,2010
102 CONTINUE
IF(NFIR-1)104,2010,104
103 K4=LT(NIN)+1
N(NIN,K4)=0
K5=LT(1)+1
N(1,K5)=-1
NIN=1
NOUT=1
GO TO 24

```

344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404

```

104 IF(NIN-JLAS,105,200,200                                4050
105 NIN=J+1                                                4060
    NOUT=J+1                                                4070
    KONC(J)=1                                                4080
    NY=LT(J)+1                                              4090
    N(J,NY)=0                                                4100
    DO 109 JC=NIN,JLAS                                     4110
    LCOL=LT(JC)                                             4120
    IF(LCOL.EQ.0) GO TO 109                                  4130
    IF(N(JC,LCOL)-J)109,107,109                             4140
107 N(JC,LCOL)=0                                           4150
    LT(JC)=LT(JC)-1                                         4160
109 CONTINUE                                               4170
    NZ=LT(NIN)+1                                            4180
    N(NIN,NZ)=-1                                            4190
    NOUT=NIN                                                4200
    GO TO 23                                                4210
2010 CALL SECOND(T3)                                        4220
    TPATH=T3-T2                                             4230
    WRITE(6,2015)NOP,TPATH                                   4240
2012 FORMAT(1X,10HTIME FOR FINDING,110,17H PATHS,IN SECONDS,F15.3/) 4250
    IF(NCIR-1)250,103,250                                   4260
200 CONTINUE                                               4270
    NOL=KLAS                                                4280
    CALL SECOND(T4)                                          4290
    IF(T3)2014,2020,2014                                    4300
2014 TCIR=T4-T3                                            4310
    GO TO 2016                                              4320
2020 TCIR=T4-T2                                            4330
2016 WRITE(6,1603)NOL,TCIR                                 4340
1603 FORMAT(1X,16HTIME FOR FINDING,110,18H FIRST ORDER LOOPS,/ 4350
    111H IN SECONDS,F15.3/)                                4360
                                                         4370
                                                         4380
C                                                         4390
C PROGRAM MAIN--6                                         4400
C FIND SECOND ORDER LOOPS                                 4410
                                                         4420
    NOL=KLAS                                                4430
    KHOL=0                                                  4440
    DO 257 KOW=1,NPAC                                       4450
257 NOCTOT(KOW)=0                                          4460
    LOW1=NOP+1                                             4470
    NOC=0                                                  4480
    NOL1=NOL-1                                             4490
    DO 203 LIR1=LOW1,NOL1                                    4500
    LOW2=LIR1+1                                            4510
    DO 202 LIR2=LOW2,NOL                                    4520
    NAN=NPCODE(LIR1).AND.NPCODE(LIR2)                     4530
    IF(NAN)202,201,202                                     4540
201 CONTINUE                                               4550
    TCONS2=CONST(LIR1)*CONST(LIR2)                        4560
    KXP02=IXPOT(LIR1)+IXPOT(LIR2)                         4570
    KSYM2=KODET(LIR1)+KODET(LIR2)                         4580
    CALL ARRAY(2,TCONS2,KXP02,KSYM2,POLY,LIL,KIK)          4590
    KHOL=KHOL+1                                            4600
    NOC=NOC+1                                              4610
    IF(NOC-NEON)1396,1396,1395                             4620
1395 WRITE(6,1397)                                         4630
1397 FORMAT(1X,46HINCREASE NEON-THE DIMENSION OF THE ARRAY NOTCH) 4640
1396 CONTINUE                                             4650

```



```

NOTCH(NOC)=LIR2
202 CONTINUE
203 NOCTOT(LIR1)=NOC
NOCTOT(NOL)=NOC

```

```

C          PROGRAM MAIN--7
C          FIND ALL LOOPS OF ORDER
C          GREATER THAN 2

```

```

C          GENERATE THE FIRST ROW OF ISET

```

```

NIPL=NOP+1
KAPMAX=1
INK0=1
DO 1170 ISC=NIPL,NOL
  INK1=NOCTOT(ISC)
  IF(ISC-1)1171,1171,1172
1172 INK2=NOCTOT(ISC-1)+1
  GO TO 1173
1171 INK2=1
1173 IF(INK1-INK2-INK0)1170,1170,1175
1175 INK0=INK1-INK2
1170 CONTINUE
  IF(INK0-NCI)1391,1391,1390
1390 WRITE(6,1392) INK0
1392 FORMAT(1X,52HINCREASE NCI-THE NO. OF COLUMNS IN DIMENSION OF ISET)
1391 CONTINUE
  DO 490 NIP=NIPL,NOL
    INKU=NOCTOT(NIP)
    IF(NIP-1)210,210,211
  211 INKL=NOCTOT(NIP-1)+1
  GO TO 212
  210 INKL=1
  212 CONTINUE
    IF(INKU-INKI)490,490,410
  410 JIP=0
    DO 480 INK=INKL,INKU
      JIP=JIP+1
  480 ISET(1,JIP)=NOTCH(INK)
      MAPO(NIP)=INKU-INKL+1

```

```

C          INIATE PROCESS FOR FINDING
C          HIGHER ORDER LOOPS

```

```

DO 430 KAT=1,NPAC
  JAC(KAT)=0
  430 NUP(KAT)=0
  JAC(1)=MAPO(NIP)
  KAP=2
  440 KAP=KAP-1
  IF(KAP)490,490,429
  425 KAP=KAP+1
  IF(KAP-NRI)1350,1350,1351
1351 WRITE(6,1352)
1352 FORMAT(1X,49HINCREASE NRI-THE NO. OF ROWS IN DIMENSION OF ISET)
1350 CONTINUE
  NUP(KAP)=0
  429 KAP1=KAP+1
  JAC(KAP1)=0
  NUP(KAP)=NUP(KAP)+1

```

C

466  
467  
4680  
469  
470  
4710  
4720  
473  
474  
4750  
476  
477  
4780  
4790  
480  
481  
4820  
483  
484  
4850  
4860  
487  
488  
4890  
490  
491  
4920  
4930  
494  
495  
4960  
497  
498  
499  
5000  
501  
502  
5030  
504  
505  
506  
5070  
508  
509  
5100  
511  
512  
513  
5140  
515  
516  
5170  
518  
519  
520  
5210  
522  
523  
5240  
5250  
526

```

C LABEL LOOP OF FIRST CKT 5270
  NAP=NUP(KAP) 5280
  IF(KAPMAX=KAP)1347,1348,1348 5290
1347 KAPMAX=KAP 5300
1348 CONTINUE 5310
  ISAT=ISET(KAP,NAP) 5320
C 5330
C TEST LOOP OF REMAINING CKTS 5340
  MAPU=JAC(KAP) 5350
  MAPL=NUP(KAP)+1 5360
  DO 435 MAPI=MAPL,MAPU 5370
  ISOT=ISET(KAP,MAPI) 5380
  KAN=NPCODE(ISAT).AND.NPCODE(ISOT) 5390
  IF(KAN)435,455,435 5400
455 CONTINUE 5410
C 5420
C WRITE 5430
  TCONSG=CONST(NIP) 5440
  KXPOG=IXPOT(NIP) 5450
  KSYMG=KODET(NIP) 5460
  DO 477 LPO=1,KAP 5470
  ITIC=NUP(LPO) 5480
  ITUCH=ISET(LPO,ITIC) 5490
  TCONSG=TCONSG*CONST(ITUCH) 5500
  KXPOG=KXPOG+IXPOT(ITUCH) 5510
477 KSYMG=KSYMG+KODET(ITUCH) 5520
  TCONSG=TCONSG*CONST(ISOT) 5530
  KXPOG=KXPOG+IXPOT(ISOT) 5540
  KSYMG=KSYMG+KODET(ISOT) 5550
  KAPP=KAP*2 5560
  CALL ARRAY(KAPP,TCONSG,KXPOG,KSYMG,POLY,LIL,KIK) 5570
  KHOL=KHOL+1 5580
C SET COUNTERS 5590
423 KAP1=KAP+1 5600
  JAC(KAP1)=JAC(KAP1)+1 5610
  JACK=JAC(KAP1) 5620
  ISET(KAP1,JACK)=ISET(KAP,MAPI) 5630
435 CONTINUE 5640
  JACK=JAC(KAP1) 5650
  IF(JACK=2)421,425,425 5660
431 IF(JAC(KAP)=NUP(KAP)-1)440,440,429 5670
490 CONTINUE 5680
  CALL ARRAY(2,1,,0,0,POLY,LIL,KIK) 5690
  CALL SECOND(15) 5700
  TNL=T5-T4 5710
  WRITE (6,1604) KHOL,TNL 5720
1604 FORMAT(1X,1(HTIME FOR FINDING ,I10,8H SETS OF/ 5730
  130H NONTOUCHING LOOPS, IN SECONDS, F15.3/) 5740
  5750
  5760
C PROGRAM MAIN--R 5770
C DECODE COMPOSITE SYMBOL CODE 5780
C AND ISOLATE SYMBOLS FROM 5790
C INVERSE SYMBOLS 5800
  5810
  5820
  NANU=LIL-1 5830
  DO 641 J1=1,NEXPS 5840
  DO 691 J2=1,NT0 5850
691 POLYU(J1,J2)=0 5860
  DO 693 J1=1,NT0 5870

```

```

DO 693 J2=1,NSPTU
SEMPOD(J1,J2)=STAR(1)
SEMPOD(J1,J2)=STAR(1)
SIMBON(J1,J2)=SB
693 SIMBOD(J1,J2)=SB
DO 951 J4=1,NTU
NA(J4)=0
951 NR(J4)=0
C
C DECODE KSORT(J7) AND RECORD TERMS
C CONTAINING FEEDBACK SYMBOL *FB*
JZ=LIL-1
DO 646 JZ=1,JZU
KODY=KSORT(JZ)
ITOP(JZ)=0
IF(KODY)715,646,715
715 CALL DECODE(KOO,KODY,IZ,FB,JZ,SEMBOL,KODF,KODI,ITOP,KBASIS)
C
C ISOLATE NUM. SYMBOLS AND INVERSE SYMBOLS
C OF KSORT(JZ)
637 NAK=0
NAT=0
IF(IZ)646,646,647
647 CONTINUE
DO 645 NZ=1,IZ
KOZY=KODI(NZ)
IARG=KODF(NZ)
IF(IARG=NRS)1340,1340,1341
1341 WRITE(6,1342)
1342 FORMAT(1X,30HINCREASE THE DIMENSION OF STAR)
1340 CONTINUE
IF(KONS(KOZY))657,657,659
657 NAK=NAK+1
IF(NAK=NSPT(I)-1)1376,1375,1375
1375 WRITE(6,1376)
1377 FORMAT(1X,40HNSPT EXCEEDS LIMIT-INCREASE DIMENSIONS ,
115HCONTAINING NSPT)
1376 CONTINUE
SIMBON(JZ,NAK)=SEMBOL(KOZY)
SEMPOD(JZ,NAK)=STAR(IARG)
NA(JZ)=NA(JZ)+1
GO TO 645
659 NAT=NAT+1
IF(NAT=NSPT(I)-1)1381,1380,1380
1380 WRITE(6,1381)
1382 FORMAT(1X,40HNSPT EXCEEDS LIMIT-INCREASE DIMENSIONS ,
115HCONTAINING NSPT)
1381 CONTINUE
SIMBOD(JZ,NAT)=SEMBOL(KOZY)
SEMPOD(JZ,NAT)=STAR(IARG)
NR(JZ)=NR(JZ)+1
645 CONTINUE
646 CONTINUE
C
C PROGRAM MAIN--9
C SEPARATE POLY INTO ARRAYS FOR THE
C NUMERATOR AND DENOMINATOR OF THE
C TRANSFER FUNCTION

```

5880  
5890  
5900  
5910  
5920  
5930  
5940  
5950  
5960  
5970  
5980  
5990  
6000  
6010  
6020  
6030  
6040  
6050  
6060  
6070  
6080  
6090  
6100  
6110  
6120  
6130  
6140  
6150  
6160  
6170  
6180  
6190  
6200  
6210  
6220  
6230  
6240  
6250  
6260  
6270  
6280  
6290  
6300  
6310  
6320  
6330  
6340  
6350  
6360  
6370  
6380  
6390  
6400  
6410  
6420  
6430  
6440  
6450  
6460  
6470  
6480

```

C THE CONSTANT COEFFICIENTS IN THE TRANSFER FUNCTION 6490
C ARE SEPARATED INTO ARRAYS FOR THE NUMERATOR 6500
C AND DENOMINATOR 6510
  KIKU=KIK-1 6520
  DO 755 JA=1,KIKU 6530
  JIB=0 6540
  JD=0 6550
  DO 755 JC=1,NANU 6560
  IF(ITOP(JC))753,753,751 6570
751 JIB=JIB+1 6580
  POLYU(JA,JIB)=POLY(JA,JC) 6590
  GO TO 755 6600
753 JD=JD+1 6610
  POLY(JA,JD)=POLY(JA,JC) 6620
755 CONTINUE 6630
  CALL SECOND(T6) 6640
  TDECOD=T6-T5 6650
  WRITE (6,1605) TDECOD 6660
1605 FORMAT (1X,36HTIME FOR DECODING SYMBOLS IN SECONDS,F15.3/) 6670
6680
6690
PROGRAM MAIN--10 6700
C MAKE POWERS OF S IN OUTPUT 6710
C TRANSFER FUNCTION POSITIVE 6720
6730
6740
MAXIM=0 6750
KARU=KIK-1 6760
DO 522 KAR=1,KARU 6770
IF(MSORT(KAR))521,522,522 6780
521 IF(MAXIM+MSORT(KAR))523,522,522 6790
523 MAXIM=-MSORT(KAR) 6800
522 CONTINUE 6810
DO 524 KIT=1,KARU 6820
524 MSORT(KIT)=MAXIM+MSORT(KIT) 6830
6840
6850
6860
PROGRAM MAIN--11 6870
C PRINT OUT NUMERATOR OF 6880
C THE TRANSFER FUNCTION 6890
6900
LUK=0 6910
IKU=LIL-1 6920
WRITE (6,931) 6930
WRITE (6,930) 6940
WRITE (6,925) 6950
920 FORMAT (25X*20HNUMERATOR POLYNOMIAL///) 6960
WRITE (6,921) 6970
921 FORMAT (1X,9HCOLUMN,12X,23HSYMBOL FOR GIVEN COLUMN) 6980
DO 905 IK=1,IKU 6990
IF(ITOP(IK))905,905,901 7000
901 ILU=NA(IK) 7010
IF(1LU)710,710,711 7020
710 ILU=1 7030
711 JLU=NB(IK) 7040
IF(JLU)712,712,713 7050
712 JLU=1 7060
713 CONTINUE 7070
LUK=LUK+1 7080
WRITE (6,902) LUK,(SIMBON(IK,1L),SEMPOK(IK,1L), 7090

```

```

      IIL=1,ILU),DASH,(SIMBOD(IK,JL),SEMPOD(IK,JL),JL=1,JLU)
903 FORMAT(1X,I5,20X,30A3)
905 CONTINUE
      WRITE (6,930)
930 FORMAT (//)
      WRITE(6,1821)
1821 FORMAT(1X,7H POWER)
      WRITE (6,925)
922 FORMAT (1X,6H OF S ,17X,33HCONSTANT COEFS. IN THE POLYNOMIAL)
      LML=1
      LMU=7
      IF(JIB-LMU)820,818,818
820 LMU=JIB
818 WRITE (6,806) (LO,LO=LML,LMU)
806 FORMAT(2X,7(8X,6HCOLUMN,12))
      KROWU=KIK-1
      DO 808 KROW=1,KROWU
      WRITE (6,812) MSORT(KROW),(POLYU(KROW,LM),LM=LML,LMU)
810 FORMAT (15,5H ,7(E12.5,4H ))
808 CONTINUE
      IF(JIB-LMU)814,814,812
812 LML=LML+7
      LMU=LMU+7
      IF(JIB-LMU)816,818,818
816 LMU=JIB
      GO TO 818
814 CONTINUE

```

```

C          PROGRAM MAIN--12
C          PRINT OUT DENOMINATOR OF
C          THE TRANSFER FUNCTION

```

```

      LUK=0
      IKU=LIL-1
      WRITE (6,931)
931 FORMAT (50H*****
      WRITE (6,930)
      WRITE (6,923)
923 FORMAT (25X,22HDENOMINATOR POLYNOMIAL///)
      WRITE (6,924)
924 FORMAT (1X,6HCOLUMN,12X,23HSYMBOL FOR GIVEN COLUMN)
      DO 705 IK=1,IKU
      IF(ITOP(IK),701,701,705)
701 ILU=NA(IK)
      LUK=LUK+1
      IF(ILU)915,915,916
915 ILU=1
916 JLU=NB(IK)
      IF(JLU)917,917,918
917 JLU=1
918 CONTINUE
      WRITE(6,703) LUK,(SIMBON(IK,IL),SEMPON(IK,IL),
      IIL=1,ILU),DASH,(SIMBOD(IK,JL),SEMPOD(IK,JL),JL=1,JLU)
703 FORMAT(1X,I5,20X,30A3)
705 CONTINUE
      WRITE (6,930)
      WRITE(6,1822)
1822 FORMAT(1X,7H POWER)
      WRITE (6,925)

```

7100  
7110  
7120  
7130  
7140  
7150  
7160  
7170  
7180  
7190  
7200  
7210  
7220  
7230  
7240  
7250  
7260  
7270  
7280  
7290  
7300  
7310  
7320  
7330  
7340  
7350  
7360  
7370  
7380  
7390  
7400  
7410  
7420  
7430  
7440  
7450  
7460  
7470  
7480  
7490  
7500  
7510  
7520  
7530  
7540  
7550  
7560  
7570  
7580  
7590  
7600  
7610  
7620  
7630  
7640  
7650  
7660  
7670  
7680  
7690  
1700

```

925 FORMAT(1X,8H OF S ,17X,33HCONSTANT COEFS. IN THE POLYNOMIAL) 7710
    LML=1 7720
    LMU=7 7730
    IF(JD-LMU)520,518,518 7740
520 LMU=JD 7750
518 WRITE (6,506) (LO,LO=LML,LMU) 7760
506 FORMAT(2X,7(8X,6HCOLUMN,12)) 7770
    KROWU=KIK-1 7780
    DO 508 KROW=1,KROWU 7790
    WRITE (6,517) MSURT(KROW), (POLY(KROW,LM),LM=LML,LMU) 7800
510 FORMAT(15,3H ,7(E12.5,4H )) 7810
508 CONTINUE 7820
    IF(JD-LMU)514,514,512 7830
512 LML=LML+7 7840
    LMU=LMU+7 7850
    IF(JD-LMU)516,518,518 7860
516 LMU=JD 7870
    GO TO 518 7880
514 CONTINUE 7890
    WRITE(6,930) 7900
    CALL SECOND (TEND) 7910
    TEXEC=TEND-TSTART 7920
    WRITE (6,1161) TEXEC 7930
1161 FORMAT (1X,27H EXECUTION TIME IN SECONDS,F15.3// 7940
    128H AUGUST 1970 VERSION OF SNAP) 7950
250 GO TO 1111 7960
    END 7970
    SUBROUTINE SFG(NFIRST,NLAST,IXPON,WEIGT,SYMBUL,KONSO,MIX,NEST, 7980
    ILIST,NIN,NOUT,NOD,NOB,LISIG,NODA,NODB) 7990
C***** 8000
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK 8010
C CHARACTERISTIC NBN(DEFINED IN PROGRAM MAIN-1) 8020
    DIMENSION JROW(35),NP(35),IVV(35),NUML(35),ICV(35),INTREE(35) 8030
    DIMENSION LANC(35) 8040
    DIMENSION NF(35,35),IB(35,35),NS(35,35) 8050
    DIMENSION TYPB(35),JB(35),LB(35),MSYM(35) 8060
    DIMENSION IQUAL(35),VAL(35),SYM(35) 8070
    DIMENSION IQUALX(35),VALX(35),NUMLX(35),INTPE(35),NOTREE(35) 8080
    DIMENSION TYPX(35),NIJMX(35),JBX(35),LBX(35),SYMX(35) 8090
C***** 8100
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK 8110
C CHARACTERISTIC NNG 8120
    DIMENSION CVAL(100) 8130
    DIMENSION NFIRST(100),NLAST(100),IXPON(100),WEIGT(100) 8140
    DIMENSION MAPY(100),KONSO(100),NEST(100),TYPE(100) 8150
    DIMENSION SYMBUL(100),MIX(100) 8160
C***** 8170
COMMON NF,NS,IB 8180
COMMON/C2/NNG,NBG 8190
C 8200
C SUBPROGRAM #A# 8210
DATA Y,G,C,I,U,R,CL,Z/2HY ,2HG ,2HC ,1H=,2HR ,2HL ,2HZ / 8220
DATA E,CI,CC,CV,VV,VC/2HE ,2HI ,2HCC,2HCV,2HVV,2HVC/ 8230
DATA FB/3H FB/ 8240
DATA ONE/3H 1/ 8250
DO 710 IC=1,NNG 8260
DO 710 IK=1,NNG 8270
NS(IC,IK)=0 8280
710 NF(IC,IK)=0 8290
8300
8310

```

```

LINK=0
DO 152 IG=1,NBG
NEST(IG)=0
152 KONSO(IG)=0
WRITE (6,260)
260 FORMAT (//)
WRITE(6,717)
717 FORMAT(30X,7HNNETWORK/)
NLAST(1)=NIN
IXPON(1)=0
WEIGT(1)=-1.
SYMBOL(1)=FR
KONSO(1)=0
NEST(1)=1
MO=0
LO=0
LIST=1
KLU=0
DO 5 I1=1,NMG
INTREE(I1)=0
5 JROW(I1)=0
DO 528 I=1,NOR
READ(5,9) TYPX(I),NUMX(I),JBX(I),LBX(I),SYMX(I),
1 IQUALX(I),VALX(I),NUMLX(I)
IF(TYPX(I).EQ.CC)GO TO 1300
IF(TYPX(I).EQ.CV)GO TO 1300
IF(TYPX(I).EQ.VV)GO TO 1300
IF(TYPX(I).EQ.VC)GO TO 1300
GO TO 1301
1300 IF(NUMLX(I),1301,1302,1301
1302 WRITE(6,1303)
1303 FORMAT(1X,49H***ERROR***CONTROL SPECIFICATION FOR DEP. SOURCE ,
17HMISSING)
NOR=0
GO TO 1305
1301 CONTINUE
528 CONTINUE
9 FORMAT (A2,I3,2I5,1X,A3,A1,E12.5,I3)
WRITE(6,261)
261 FORMAT(2X,32HELEMENT ELEMENT INITIAL TERMINAL,
128H ELEMENT ELEMENT ELEMENT NO.)
WRITE(6,262)
262 FORMAT(2X,30H TYPE NUMBER NODE NODE SYMBOL,
120H VALUE OF CONTROL)
DO 601 M=1,NOR
601 WRITE(6,600) TYPX(M),NUMX(M),JBX(M),LBX(M),SYMX(M),
1 IQUALX(M),VALX(M),NUMLX(M)
600 FORMAT (4X,A2,6X,I2,6X,I2,6X,I2,6X,A3,A1,E12.5,2X,I2)
CALL FTREE(TYPX,JBX,LBX,INTRE,NOTREE,NOD,NOR)
C
C SUBPROGRAM #B#
WRITE (6,518)
518 FORMAT(30X,13HTREE SELECIED)
NUMU=NOD-1
DO 21 NU=1,NUMU
IO=INTRE(NU)
NUMC=NUMX(IO)
TYPB(NUMC)=TYPX(IO)
JB(NUMC)=JBX(IO)
LB(NUMC)=LBX(IO)
SYM(NUMC)=SYMX(IO)

```

8320  
8330  
8340  
8350  
8360  
8370  
8380  
8390  
8400  
8410  
8420  
8430  
8440  
8450  
8460  
8470  
8480  
8490  
8500  
8510  
8520  
8530  
8540  
8550  
8560  
8570  
8580  
8590  
8600  
8610  
8620  
8630  
8640  
8650  
8660  
8670  
8680  
8690  
8700  
8710  
8720  
8730  
8740  
8750  
8760  
8770  
8780  
8790  
8800  
8810  
8820  
8830  
8840  
8850  
8860  
8870  
8880  
8890  
8900  
8910  
8920

30

```

      IQUAL(NUMC)=IQUALX(IO)
      VAL(NUMC)=VALX(IO)
      NUML(NUMC)=NUMLX(IO)
      INTREE(NUMC)=1
      WRITE(6,517),TYPB(NUMC),NUMC,JH(NUMC),LB(NUMC),SYM(NUMC),
1 IQUAL(NUMC),VAL(NUMC),NUML(NUMC)
517 FORMAT(4X,A2,6X,I2,6X,I2,6X,I2,6X,A3,A1,E12.5,2X,I2)
      KLU=KLU+1
      LINC(NUMC)=0
      IF(TYPB(NUMC).NE.VV) GO TO 3
      MO=MO+1
      IVV(MO)=NUMC
3 IF(TYPB(NUMC).NE.CV) GO TO 4
      LO=LO+1
      ICV(LO)=NUMC
4 JF=JH(NUMC)
      LF=LB(NUMC)
      IR(JF,LF)=NIJMC
      IR(LF,JF)=NIJMC
      JROW(JF)=JROW(JF)+1
      JROJ=JROW(JF)
      NF(JF,JROJ)=LF
      NS(JF,LF)=1
      JROW(LF)=JROW(LF)+1
      JROL=JROW(LF)
      NF(LF,JROL)=JF
      NS(LF,JF)=-1
21 CONTINUE
      WRITE(6,260)
      WRITE(6,715)
715 FORMAT(30X,JHSFG/)
      DO 13 ILL=1,NOD
      JROI=JROW(ILL)+1
13 NF(ILL,JROI)=0
C
C      SUBPROGRAM #C#
C      THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
C      FROM BRANCH NODE TO LINK NODE
      NOBY=NOB
151 CONTINUE
      NES=0
      LON=0
      IF(KLU=NOB) 532,360,532
532 LINK=LINK+1
      IF(NOTREE(LINK)) 534,534,532
534 NUMC=NUMX(LINK)
      TYPE(NUMC)=TYPX(LINK)
      JK=JHX(LINK)
      LK=LRX(LINK)
      SYM(NUMC)=SYMX(LINK)
      IQUAL(NUMC)=IQUALX(LINK)
      CVAL(NUMC)=VALX(LINK)
      NUMB=NUMLX(LINK)
      TYP2=TYPE(NUMC)
      CVALU=CVAL(NUMC)
      KLU=KLU+1
      LINC(NUMC)=1
      KDEPS=0
      KANSO=0
      IF(TYPE(NUMC).EQ.CL)GO TO 117
      IF(TYPE(NUMC).EQ.G)GO TO 119
```



```

IF (TYPE(NUMC).EQ.Y) GO TO 119
IF (TYPE(NUMC).EQ.R) GO TO 700
IF (TYPE(NUMC).EQ.Z) GO TO 700
IF (TYPE(NUMC).EQ.C) GO TO 121
KDEPS=1
IF (TYPE(NUMC).EQ.E) GO TO 123
IF (TYPE(NUMC).EQ.CI) GO TO 123
IF (TYPE(NUMC).EQ.VC) GO TO 165
IF (TYPE(NUMC).EQ.CC) GO TO 265
117 IXPS=-1
KANSO=1
GO TO 123
119 IXPS=0
GO TO 123
700 IXPS=0
KANSO=1
GO TO 123
121 IXPS=1
123 CALL TREP(JK,LK,NF,NP,NPL)
IFIN=NUMC
149 LON=LON+1
NP1=NP(LON)
NP2=NP(LON+i)
107 INIT=IB(NP1,NP2)
109 SIGN=NS(NP1,NP2)
IF (K EPS) 167,167,169
167 IF (EQUAL(NUMC).EQ.IQ) GO TO 111
NES=1
CONST=SIGN
GO TO 125
111 CONST=SIGN*CVALU
125 LIST=LIST+1
IF (NES) 502,503,502
502 NEST(LIST)=1
503 KONSO(LIST)=KANSO
NFIRST(LIST)=INIT
NLAST(LIST)=IFIN
SYMBUL(LIST)=SYM(IFIN)
IXPON(LIST)=IXPS
IF (KONSO(LIST)) 505,505,504
504 WEIGT(LIST)=1./CONST
GO TO 506
505 WEIGT(LIST)=CONST
506 MAPY(NUMC)=LIST
127 FORMAT (3I5,E12.5)
129 FORMAT (A4)

```

9540  
9550  
9560  
9570  
9580  
9590  
9600  
9610  
9620  
9630  
9640  
9650  
9660  
9670  
9680  
9690  
9700  
9710  
9720  
9730  
9740  
9750  
9760  
9770  
9780  
9790  
9800  
9810  
9820  
9830  
9840  
9850  
9860  
9870  
9880  
9890  
9900  
9910  
9920  
9930  
9940  
9950  
9960  
9970  
9980  
9990  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014

```

C
C SUBPROGRAM #D#
C THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
C FROM LINK NODE TO BRANCH NODE
169 CONTINUE
IF (TYPE(INIT).EQ.E) GO TO 201
IF (TYPE(INIT).EQ.CI) GO TO 201
IF (TYPE(INIT).EQ.VV) GO TO 201
IF (TYPE(INIT).EQ.CV) GO TO 201
LIST=LIST+1
IF (TYPE(INIT).EQ.R) GO TO 133
IF (TYPE(INIT).EQ.Z) GO TO 133
IF (TYPE(INIT).EQ.G) GO TO 702
IF (TYPE(INIT).EQ.Y) GO TO 702
IF (TYPE(INIT).EQ.CL) GO TO 135

```

IF (TYPE(INIT).EQ.C)GO TO 137	10150
133 IXPON(LIST)=0	10160
GO TO 141	10170
702 IXPON(LIST)=0	10180
KONSO(LIST)=1	10190
GO TO 141	10200
135 IXPON(LIST)=1	10210
GO TO 141	10220
137 IXPON(LIST)=-1	10230
KONSO(LIST)=1	10240
141 IF (IQUAL(INIT).EQ.10)GO TO 139	10250
NEST(LIST)=1	10260
WEIGT(LIST)=-1.*SIGN	10270
GO TO 147	10280
139 IF (KONSO(LIST))608,608,607	10290
607 WEIGT(LIST)=-SIGN/VAL(INIT)	10300
GO TO 147	10310
608 WEIGT(LIST)=-SIGN*VAL(INIT)	10320
147 NFIRST(LIST)=IFIN	10330
NLAST(LIST)=INIT	10340
SYMBUL(LIST)=SYM(INIT)	10350
201 NPLA=NPL-1-LUN	10360
IF (NPLA)151,151,149	10370
C	10380
C SUBPROGRAM E	10390
C THIS PROGRAM SETS UP SFG INFO. FOR VC	10400
C TYPE CONTROL SOURCES	10410
165 NUNO=NUMB	10420
IF (INTREE(NUMB))163,163,161	10430
163 LIST=LIST+1	10440
NFIRST(LIST)=NUMB	10450
NOBY=NOBY+1	10460
NLAST(LIST)=NOBY	10470
SYMBUL(LIST)=SYM(NUMB)	10480
NUNO=NOBY	10490
IF (TYPE(NUMB).EQ.Y)GO TO 912	10500
IF (TYPE(NUMB).EQ.G)GO TO 912	10510
IF (TYPE(NUMB).EQ.C)GO TO 914	10520
IF (TYPE(NUMB).EQ.CL)GO TO 916	10530
NUNO=0	10540
IXPON(LIST)=0	10550
GO TO 918	10560
912 IXPON(LIST)=0	10570
KUNO=1	10580
GO TO 918	10590
914 IXPON(LIST)=-1	10600
KUNO=1	10610
GO TO 918	10620
916 IXPON(LIST)=1	10630
KUNO=0	10640
918 IF (IQUAL(NUMB).EQ.IQ)GO TO 920	10650
NEST(LIST)=1	10660
WEIGT(LIST)=1.	10670
GO TO 209	10680
920 IF (KUNO)922,922,924	10690
922 WEIGT(LIST)=CVAL(NUMB)	10700
GO TO 209	10710
924 WEIGT(LIST)=1./CVAL(NUMB)	10720
209 KONSO(LIST)=1	10730
161 LIST=LIST+1	10740
NFIRST(LIST)=NUNO	10750

```

NLAST(LIST)=NUMC
SYMBUL(LIST)=SYM(NUMC)
IXPON(LIST)=0
IF(IQUAL(NUMC).EQ.IQ)GO TO 171
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 203
171 WEIGT(LIST)=CVALU
203 CONTINUE
GO TO 123

```

```

C
C SUBPROGRAM #F#
C THIS PROGRAM SETS UP SFG INFO. FOR CC
C TYPE CONTROL SOURCES

```

```

265 MUNO=NUMB
IF(IPTREE(NUMB))621,621,620
620 LIST=LIST+1
NFIRST(LIST)=NUMB
NOBY=NOBY+1
NLAST(LIST)=NOBY
SYMBUL(LIST)=SYM(NUMB)
MUNO=NOBY
IF(TYPB(NUMB).EQ.Z)GO TO 233
IF(TYPB(NUMB).EQ.R)GO TO 233
IF(TYPB(NUMB).EQ.CL)GO TO 235
IF(TYPB(NUMB).EQ.C)GO TO 237
KUNO=0
IXPON(LIST)=0
GO TO 241
233 IXPON(LIST)=0
KUNO=1
GO TO 241
235 IXPON(LIST)=-1
KUNO=1
GO TO 241
237 IXPON(LIST)=1
KUNO=0
241 IF(IQUAL(NUMB).EQ.IQ)GO TO 239
NEST(LIST)=1
WEIGT(LIST)=1
GO TO 247
239 IF(KUNO)900,900,902
900 WEIGT(LIST)=VAL(NUMB)
GO TO 247
902 WEIGT(LIST)=1./VAL(NUMB)
247 KONS0(LIST)=1.
621 LIST=LIST+1
NFIRST(LIST)=MUNO
NLAST(LIST)=NUMC
SYMBUL(LIST)=SYM(NUMC)
IXPON(LIST)=0
IF(IQUAL(NUMC).EQ.IQ)GO TO 271
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 281
271 WEIGT(LIST)=CVALU
GO TO 281
281 CONTINUE
GO TO 123

```

```

C
C SUBPROGRAM #G#

```

1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136

NOBY=NOBY+1	11980
NLAST(LIST)=NOBY	11990
SYMBUL(LIST)=SYM(NUNO)	12000
IF(TYPB(NUNO).EQ.Z)GO TO 433	12010
IF(TYPB(NUNO).EQ.R)GO TO 433	12020
IF(TYPB(NUNO).EQ.CL)GO TO 435	12030
IF(TYPB(NUNO).EQ.C)GO TO 437	12040
KUNO=0	12050
IXPON(LIST)=0	12060
GO TO 441	12070
433 IXPON(LIST)=0	12080
KUNO=1	12090
GO TO 441	12100
435 IXPON(LIST)=1	12110
KUNO=1	12120
GO TO 441	12130
437 IXPON(LIST)=-1	12140
KUNO=0	12150
441 IF(IQUAL(NUNO).EQ.IQ)GO TO 439	12160
NEST(LIST)=1	12170
WEIGT(LIST)=1.	12180
GO TO 448	12190
439 IF(KUNO)908,908,910	12200
908 WEIGT(LIST)=VAL(NUNO)	12210
GO TO 448	12220
910 WEIGT(LIST)=1./VAL(NUNO)	12230
448 KONSO(LIST)=1	12240
447 CONTINUE	12250
NUNO=NOBY	12260
461 LIST=LIST+1	12270
NFIRST(LIST)=NUNO	12280
NLAST(LIST)=LI	12290
SYMBUL(LIST)=SYM(LI)	12300
IXPON(LIST)=0	12310
IF(IQUAL(LI).EQ.IQ)GO TO 471	12320
NEST(LIST)=1	12330
WEIGT(LIST)=1.	12340
GO TO 403	12350
471 WEIGT(LIST)=VAL(LI)	12360
403 CONTINUE	12370
405 CONTINUE	12380
C	12390
C SUBPROGRAM #I#	12400
C GENERATING OUTPUT NODE OF SFG	12410
C	12420
515 CONTINUE	12430
IF(NOUT)514,512,514	12440
512 CALL TREP(NODA,NODB,NF,NP,NPL)	12450
NOUT=NOBY+1	12460
MOPU=NPL-1	12470
DO 510 MOP=1,MOPU	12480
N1=NP(MOP)	12490
N2=NP(MOP+1)	12500
LIST=LIST+1	12510
NFIRST(LIST)=I6(N1,N2)	12520
NLAST(LIST)=NOUT	12530
SYMBUL(LIST)=ONE	12540
IXPON(LIST)=0	12550
KONSO(LIST)=0	12560
NFST(LIST)=0	12570
510 WEIGT(LIST)=NS(N1,N2)	12580

```

C     THIS PROGRAM SETS UP SFG INFO. FOR VV
C     TYPE CONTROL SOURCES
360 IF (MO)460,460,364
364 DO 305 MI=1,MO
      KI=IVV(MI)
      NUNO=NUML(KI)
      IF (LINC(NUNO))361,361,363
363 LIST=LIST+1
      NFIRST(LIST)=NUML(KI)
      NOBY=NOBY+1
      NLAST(LIST)=NOBY
      SYMBUL(LIST)=SYM(NUNO)
      IF (TYPE(NUNO).EQ.Y)GO TO 333
      IF (TYPE(NUNO).EQ.G)GO TO 333
      IF (TYPE(NUNO).EQ.C)GO TO 335
      IF (TYPE(NUNO).EQ.CL)GO TO 337
      KUNO=0
      IXPON(LIST)=0
      GO TO 341
333 IXPON(LIST)=0
      KUNO=1
      GO TO 341
335 IXPON(LIST)=-1
      KUNO=1
      GO TO 341
337 IXPON(LIST)=1
      KUNO=0
341 IF (EQUAL(NUNO).EQ.IQ)GO TO 339
      NEST(LIST)=1
      WEIGT(LIST)=1.
      GO TO 348
339 IF (KUNO)904,904,906
904 WEIGT(LIST)=CVAL(NUNO)
      GO TO 348
906 WEIGT(LIST)=1./CVAL(NUNO)
348 KONSQ(LIST)=1
347 CONTINUE
      NUNO=NOBY
361 LIST=LIST+1
      NFIRST(LIST)=NUNO
      NLAST(LIST)=KI
      SYMBUL(LIST)=SYM(KI)
      IXPON(LIST)=0
      IF (EQUAL(KI).EQ.IQ)GO TO 371
      NEST(LIST)=1
      WEIGT(LIST)=1.
      GO TO 303
371 WEIGT(LIST)=VAL(KI)
303 CONTINUE
305 CONTINUE

```

```

C
C     SUBPROGRAM #H#
C     THIS PROGRAM SETS UP SFG INFO. FOR CV
C     TYPE CONTROL SOURCES
460 IF (LO)515,515,464
464 DO 405 MI=1,LO
      LI=ICV(MI)
      NUNO=NUML(LI)
      IF (LINC(NUNO))463,463,461
463 LIST=LIST+1
      NFIRST(LIST)=NUML(LI)

```

1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197

```

511 CONTINUE
514 NFIRST(1)=N(1)UT
482 IF(LISTG)484,486,1200
1200 WRITE(6,263)
263 FORMAT(1X,37HINITIAL TERMINAL EXPONENT BRANCH ,
135HBRANCH 1 IF SYMBOL 1 IF SYMBOL)
WRITE(6,264)
264 FORMAT(1X,37H NODE NODE OF S VALUE,
137H SYMBOL IS INVERTED IS USED)
DO 1202 J=1,LIST
WRITE (6,485) NFIRST(J),NLAST(J),JXPN(J),WF1GT(J),
1SYMBOL(J),KONSU(J),NEST(J)
485 FORMAT(3X,12,7X,12,6X,12,4X,E12.5,1X,A3,8X,12,14X,12)
1202 CONTINUE
486 CONTINUE
C
C SUBPROGRAM #J#
C THIS PROGRAM ORDERS SFG INFORMATION
C FOR INPUT TO MAIN PROGRAM
DO 87 J=1,NBG
87 MIX(J)=J
KONU=LIST-1
DO 80 KON=1,KONU
IU=KON+1
IL=KON
GO TO 83
81 MXL=MIX(IL)
MIX(IL)=MIX(IU)
MIX(IU)=MXL
IL=IL-1
IU=IU-1
IF(IL)80,80,83
83 MIU=MIX(IU)
MIL=MIX(IL)
IF(NFIRST(MIU)-NFIRST(MIL))81,89,80
84 MXL=MIX(IL)
MIX(IL)=MIX(IU)
MIX(IU)=MXL
IL=IL-1
IU=IU-1
IF(IL)80,80,82
82 MIU=MIX(IU)
MIL=MIX(IL)
IF(NFIRST(MIU)-NFIRST(MIL))80,89,80
89 IF(NLAST(MIU)-NLAST(MIL))80,80,84
80 CONTINUE
1305 CONTINUE
RETURN
END
SUBROUTINE FTREE(TYPX,JBX,LBX,INTRE,NOTREE,NOD,NOH)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NHN, AND NSPT (DEFINED IN PROGRAM MAIN-1)
DIMENSION TYPX(35),JBX(35),LBX(35),INTRE(35),NOTREE(35)
DIMENSION NF(35),NF(35),KCOL(20)
C*****
COMMON/C2/NNG,NBG
DATA E,VV,CV/2HE ,2HV,2HCV/
DATA R,CL,C,Y,Z/2HR ,2HL ,2HC ,2HY ,2HZ /
DATA G/2HG /

```

```

12590
12600
12610
12620
12630
12640
12650
12660
12670
12680
12690
12700
12710
12720
12730
12740
12750
12760
12770
12780
12790
12800
12810
12820
12830
12840
12850
12860
12870
12880
12890
12900
12910
12920
12930
12940
12950
12960
12970
12980
12990
13000
13010
13020
13030
13040
13050
13060
13070
13080
13090
13100
13110
13120
13130
13140
13150
13160
13170
13180
13190

```

```

DO 40 I2=1,NNG
DO 40 I3=1,NNG
40 NF(I2,I3)=0
M=0
K=0
KC=0
DO 1 I=1,N0n
1 KCOL(I)=0
DO 3 I7=1,N0B
3 NOTREE(I7)=0
I=0
5 I=I+1
IF(TYPX(I).EQ.E)GO TO 10
6 IF(TYPX(I).EQ.VV)GO TO 10
8 IF(TYPX(I).EQ.CV)GO TO 10
GO TO 4
10 K=K+1
14 INTRE(K)=I
JBX1=JBX(I)
KCOL(JBX1)=KCOL(JBX1)+1
KCOL1=KCOL(JBX1)
NF(JBX1,KCOL1)=LHX(I)
IBX1=LBX(I)
KCOL(IBX1)=KCOL(IBX1)+1
KCOL2=KCOL(IBX1)
NF(IBX1,KCOL2)=JBX1
NOTREE(I)=1
IF(K=N0n+1)2,22,22
2 IF(M)4,4,12
4 IF(I=N0B)5,12,12
12 M=M+1
IF(TYPX(M).EQ.R)GO TO 16
IF(TYPX(M).EQ.G)GO TO 16
17 IF(TYPX(M).EQ.CL)GO TO 16
18 IF(TYPX(M).EQ.C)GO TO 16
19 IF(TYPX(M).EQ.Y)GO TO 16
20 IF(TYPX(M).EQ.Z)GO TO 16
IF(M=N0B)12,22,22
16 NINX=JBX(M)
NOUTX=LBX(M)
CALL TREP(NINX,NOUTX,NF,NP,NPL)
IF(NPL)21,21,12
21 I=M
GO TO 10
22 CONTINUE
RETURN
END
SUBROUTINE TREP(NIN,NOUT,NF,NP,NPL)

```

C\*\*\*\*\*

```

C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NBN(DEFINED IN PROGRAM MAIN-1)
DIMENSION JX(35),NP(35),JMEM(35),KMEM(35)
DIMENSION NF(35,35)

```

C\*\*\*\*\*

```

COMMON/C2/NNG,NBG
DO 80 I5=1,NNG
JX(I5)=0
NP(I5)=0
JMEM(I5)=0
80 KMEM(I5)=0

```

13200  
13210  
13220  
13230  
13240  
13250  
13260  
13270  
13280  
13290  
13300  
13310  
13320  
13330  
13340  
13350  
13360  
13370  
13380  
13390  
13400  
13410  
13420  
13430  
13440  
13450  
13460  
13470  
13480  
13490  
13500  
13510  
13520  
13530  
13540  
13550  
13560  
13570  
13580  
13590  
13600  
13610  
13620  
13630  
13640  
13650  
13660  
13670  
13680  
13690  
13700  
13710  
13720  
13730  
13740  
13750  
13760  
13770  
13780  
13790  
13800

```

NPL=0
JX(1)=NIN
JX(2)=NIN
I=1
J=NIN
NP(I)=NIN
20 K=0
25 K=K+1
IF(NF(J,K)=NOUT)30,50,30
30 IF(NF(J,K))34,32,34
32 IF(J=NIN)60,100,60
C
C FLOWER CHECK
34 NJK=NF(J,K)
IF(NJK=JX(I))45,25,45
C
C STORE AND REMEMBER VERTEX
45 I=I+1
NP(I)=NF(J,K)
JMEM(I)=J
IA=I+1
JX(IA)=NF(J,K)
42 J=NF(J,K)
KMEM(I)=K
GO TO 20
C
C BACKSTEP
60 J=JMEM(I)
K=KMEM(I)
I=I-1
GO TO 25
C
C FINAL PATH VERTEX AND PATH LENGTH
50 II=I+1
NP(II)=NOUT
62 NPL=II
100 CONTINUE
RETURN
END
SUBROUTINE ARRAY(JSIG,XCON,JXPO,JKOD,POLY,LIL,KIK)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NTO, AND NEXPS (DEFINED IN PROGRAM MAIN-1)
DIMENSION MSORT(15),KSORT(150),POLY(15,150)
C*****
COMMON/C1/MSORT,KSORT
COMMON/C3/NEXPS,NTO
MMX=0
NNX=0
IF(KIK-1)3,22,3
3 MMU=KIK-1
DO 2 MM=1,MMU
MMX=MMX+1
IF(JXPO=MSORT(MM))2,10,2
2 CONTINUE
22 MSORT(KIK)=JXPO
MMX=KIK
KIK=KIK+1
IF(KIK=NEXPS-1)1386,1385,1385
1385 WRITE(6,1387)

```

```

13810
13820
13830
13840
13850
13860
13870
13880
13890
13900
13910
13920
13930
13940
13950
13960
13970
13980
13990
14000
14010
14020
14030
14040
14050
14060
14070
14080
14090
14100
14110
14120
14130
14140
14150
14160
14170
14180
14190
14200
14210
14220
14230
14240
14250
14260
14270
14280
14290
14300
14310
14320
14330
14340
14350
14360
14370
14380
14390
14400
14410

```



```

1387 FORMAT(1X,42MS-POWER EXCEEDS LIMIT-INCREASE DIMENSIONS ,
116HCONTAINING NEXPS)
1386 CONTINUE
10 IF(LIL-1)11,24,11
11 NNU=LIL-1
DO 12 NN=1, NNU
NNX=NNX+1
IF(JKOU=KSORT(NN))12,20,12
12 CONTINUE
24 KSORT(LIL)=JKOU
NNX=LIL
LIL=LIL+1
IF(LIL-NT0-1)1367,1365,1365
1365 WRITE(6,1366)
1366 FORMAT(1X,46HNO. OF TERMS IN OUTPUT EXCEEDS LIMIT-INCREASE ,
125HDIMENSIONS CONTAINING NT0)
1367 CONTINUE
20 POLY(MMX,NNX)=POLY(MMX,NNX)+XCUN*(-1.)**JSIG
RETURN
END
SUBROUTINE DECODE(KOU,KOBY,IZ,FB,JZ,SEMBOL,KODF,KODI,ITOP,KBASIS)
C*****
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NSPT, AND NT0 (DEFINED IN PROGRAM MAIN-1)
DIMENSION ITOP(150),SEMBOL(20),KODF(20),KODI(20)
C*****
COMMON/C4/NSPT
IZ=0
M=KBASIS-1
DO 3 J=1,KOU
IPOWER=M,ANY,KOBY
IF(IPOWER)3,3,2
2 IF(SEMBOL(J).EQ.FB)GO TO 4
IZ=IZ+1
IF(IZ-NSPT-1)1371,1370,1370
1370 WRITE(6,1372)
1372 FORMAT(1X,48HNO. OF SYMBOLS PER TERM EXCEEDS OUTPUT-INCREASE ,
126HDIMENSIONS CONTAINING NSPT)
1371 CONTINUE
KODF(IZ)=IPOWER
KODI(IZ)=J
GO TO 3
4 ITOP(JZ)=1
3 KOBY=KOBY/KBASIS
RETURN
END

```

14420  
14430  
14440  
14450  
14460  
14470  
14480  
14490  
14500  
14510  
14520  
14530  
14540  
14550  
14560  
14570  
14580  
14590  
14600  
14610  
14620  
14630  
14640  
14650  
14660  
14670  
14680  
14690  
14700  
14710  
14720  
14730  
14740  
14750  
14760  
14770  
14780  
14790  
14800  
14810  
14820  
14830  
14840  
14850  
14860  
14870  
14880

REFERENCES

1. A. De Mari, "On-Line Computer Active Network Analysis and Design in Symbolic Form", Proc. 2nd Cornell Elec. Engr. Conference, pp. 94-106, August 1969.
2. J.E. Barbay and G.W. Zobrist, "Distinguishing Characteristics of the Optimum Tree", 5th Allerton Conf. on Circuit and System Theory, pp. 730-737, 1967.
3. P.M. Lin and G.E. Alderson, "Symbolic Network Functions by a Single Path-Finding Algorithm", 7th Allerton Conf. on Circuit and System Theory, pp. 196-205, 1969.
4. H.W. Happ, "Flowgraph Techniques for Closed Systems", IEEE Trans. AES-2, May 1966.
5. D. Kroft, "All Paths Through a Maze", Proc. IEEE, Vol. 55, pp. 88-90, January 1967.
6. D.E. Knuth, The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, 1968.
7. J.E. Barbay and G.W. Zobrist, "Optimum Tree Generation by Tree Transformation", 11th Midwest Symposium on Circuit Theory, pp. 463-470, 1968.
8. E.V. Sorensen, "A Preliminary Note on the Analytical Network Program ANPl," Technical Report LKT 23, University of Denmark, Oct. 30, 1967.
9. L.P. McNamee, H. Potash, "A User's Guide and Programmer's Manual for NASAP", Dept. of Engineering, University of California, Los Angeles, August 1968.
10. CORNAP, User's Manual, School of Electrical Engineering, Cornell University, 1968.
11. L.O. Chua, "Analysis and Synthesis of Multivalued Memoryless Nonlinear Networks", IEEE Trans. on Circuit Theory, Vol. CT 14, pp. 192-209, June 1967.