PROGRAM DOCUMENTATION MANUAL

for

THE TEXAS SMALL NETWORK PACKAGE

by

J. D. Benson
Assistant Research Planner

Charles E. Bell
Data Processing Programmer

and

Vergil G. Stover
Study Supervisor

Research Report 167-3

Urban Travel Forecasting
Research Study Number 2-10-71-167

Sponsored by the
Texas Highway Department
in cooperation with
U. S. Department of Transportation
Federal Highway Administration

Texas Transportation Institute
Texas A&M University
College Station, Texas
April 1972

The opinions, findings, and conclusions expressed in this publication are those of the authors and are not necessarily those of the Federal Highway Administration.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

ABSTRACT

The Texas Small Network Package is a collection of computer programs designed to assign traffic to small transportation networks. The purpose of this manual is to provide data processing personnel with a link between the Operating Manual for the Texas Small Network Package (Research Report 119-1) and the programs contained in the package. The manual describes the operation of the package and provides flowcharts of the programs in the package. Cross references for significant variables and arrays used in the package and formats for all data sets and data cards associated with the package are provided.

Keywords: traffic assignment computer programs, transportation planning computer programs, Texas Small Network Package, computer program descriptions, computer program flowcharts.

SUMMARY

Traffic assignment is a technique which has been developed to aid transportation planning in the evaluation of future transportation system alternatives. Due to the vast quantity of data and the tedious computations involved, reliance upon computers and automated data processing is almost imperative.

The Texas Small Network Package is a collection of computer programs designed to assign traffic to small transportation networks. The package has been prepared for use with both IBM 360 and IBM 370 computer systems.

Several special features are available in the Texas Small Network Package in addition to the usual programs regarding the assignment of traffic to minimum time paths, and the assignment of traffic to "spider" networks connecting zone centroids. A self-balancing assignment program is included which can improve the agreement of assigned volumes with counted volumes. The self-balancing assignment program can also be used to induce a compliance of the assigned volumes with capacity limitations. Corridor intercepts may be coded to obtain corridor analysis summaries; travel routes may be coded to obtain volume profile comparisons and/or plots; and, selected links may be indicated for a special analysis of all traversing movements. Under normal operation, each assignment is preserved and compared with previous assignments.

The Texas Small Network Package is comprised of eighty-one control sections. The control sections perform the nineteen user program options available under the package.

The package basically operates in sequential mode. As each control card specifying a user program option is encountered in the data card input stream, the card is interpreted to determine the desired program option and the appropriate program option is executed.

IMPLEMENTATION STATEMENT

The Texas Small Network Package has been operational on the IBM 360 computer installation of the Texas Highway Department since January, 1968. It has been used extensively by the Texas Highway Department since that time.

Numerous additions, revisions and improvements have been implemented since the original transmittal. The cooperative research program between the Texas Highway Department and the Texas Transportation Institute has produced many research results which have been converted to a useable form through the preparation or modification of computer programs, and the programs have then been inserted into the Texas Small Network Package. Since research and development is dynamic in nature, this documentation will become obsolete as continuing research efforts produce new results to be implemented in the package.

INTRODUCTION

The purpose of this manual is to provide data processing personnel with a link between the operating manual for the Texas Small Network Package and the programs contained in the package. This manual, therefore, assumes the working knowledge and understanding of the operating manual, and general familiarity with the terminology associated with both traffic assignment and computer science. Both the operating manual and the programs (with their own internal documentation) are each a form of documentation. The objective of this manual, therefore, is to provide intermediate levels of documentation between the operating manual and the actual program listings, thereby providing a logical sequence of levels of documentation through which one may proceed from the operating manual to the particular program listing(s) of interest.

This documentation, contained in Sections I - VII of this manual, is organized as follows:

- Section I, ORGANIZATION OF PACKAGE - This section explains the organization of the programs. It includes a complete list of the programs in the Small Package including the date of their latest revision; a chart of the overlay structure for the package; and a chart of the logical divisions into which the programs may be subdivided.
- Section II, LOGICAL DIVISIONS - This portion of the manual describes the functions and operations performed in each of the logical divisions. It explains the general organization of the programs

within that division and gives a brief description of the functions performed in each of the programs within that logical division. It is felt that the program descriptions provided for each of the logical divisions will be sufficient for the programmer to identify the particular program or programs in which he is interested while at the same time providing him with an understanding of how it relates to other programs within the package.

- Section III, PROGRAM CROSS-REFERENCE AND FLOWCHARTS - This section contains a cross-reference of calling programs versus programs called and the flowcharts (or program descriptions) associated with each individual program in the Small Network Package. The objective of the flowcharts is to provide the programmer with an overview of the operation of each individual program within the package. The level of detail contained in each individual flowchart is felt to be minimal for an understanding of the individual programs. It should also be noted that these flowcharts are intended to be used in conjunction with information contained in sections IV, V, and VI when reviewing or studying a particular program listing.
- Section IV, SIGNIFICANT VARIABLES AND ARRAYS - This section contains the significant variable, arrays, data structures and control variables used by the various subroutines.
- Section V, DATA SET FORMATS - This section contains formats for various intermediate data sets formed and/or used during the operation of the Small Network Package.

- Section VI, OTHER INFORMATION - This section contains additional information which is felt to be pertinent to the understanding of the programs contained in the Small Network Package. For example, this section presently contains an explanation of the procedure used in saving turning movements during the assignment process.
- Section VII, RECENT CHANGES AND MODIFICATIONS - This section is provided for information relative to changes which have been implemented since the original documentation, and therefore, serves an "update" function for this manual.

O R G A N I Z A T I O N O F P A C K A G E

OVERLAY STRUCTURE

LOGICAL DIVISION STRUCTURE

OVERLAY STRUCTURE

The Texas Small Network Package is comprised of eighty-one control sections. These control sections are listed in Table 1 along with the date of their latest revision. The diagram shown in Figure 1 illustrates the overlay structure in which all but two of the control sections operate. The two control sections (i.e., MAIN (Output Selected Links) and E35) are used to perform the user program option $OUTPUT SELECTED LINKS which, because of core storage requirements, is run as a separate JOB.

LOGICAL DIVISION STRUCTURE

In order to explain the relationship between the control sections, they have been grouped into twenty-one logical divisions as shown in Figure 2 (note that Logical Division 21 contains the control sections for $OUTPUT SELECTED LINKS). The function (or functions) performed by each of the logical divisions is described in Section III of this manual. In addition, the sequence in which the programs are executed along with a brief description of each of the programs is included for each logical division. As can be seen from Figure 2, nine of the logical divisions contain only one control section and five of the divisions contain only two or three control sections. These small logical divisions were necessitated either by the highly specialized functions performed within them which could not readily be related to any of the other logical divisions or, in some instances because the logical division simply contains all the control sections needed to perform one of the user program

TABLE 1: CONTROL SECTIONS COMPRISING THE TEXAS SMALL NETWORK PACKAGE

Program Control Sections	Revision Date	Program Control Sections	Revision Date
ABEND	*	OUTSLN	2/26/71
ALCP	11/10/71	OUTSNT	2/26/71
BLDNET	2/26/71	OUTTRE	*
BLOCK DATA	11/10/71	OUTWLT	6/ 8/71
CLOSE	*	PARAM	*
CLOSFT	*	PATHCL	8/30/71
CMPVH	11/10/71	PATHSP	2/26/71
COPYFT	8/30/71	PRPBLD	6/ 8/71
CRD	2/26/71	PRPCTV	2/26/71
CRDINT	1/14/71	PRPNET	2/26/71
E35	*	PTLNK	*
FASPTH	*	READVL	*
FMTLNE	*	REGRES	*
FRATAR	2/26/71	RTPFL	1/14/71
GETDAT	*	RTPLT	7/30/71
GETRN	6/ 8/71	SC	1/14/71
GETRNS	*	SELECT	10/27/67
GETVOL	6/ 8/71	SLOAD	*
GTLD	11/10/71	SUBFND	*
INITL1	*	SUMEND	2/26/71
LOAD	*	SUMRY	6/ 8/71
LOAD2	8/30/71	TIME	*
LOPS	7/30/71	TREBLD	9/20/71
LNKLST	6/ 8/71	TRN	6/ 8/71
MAIN	6/ 8/71	TRNMV	*
MAIN (for Output Selected Links)	*	TURNM	*
MERG	6/17/71	UPDTNT	2/26/71
MOORE	*	VREC	8/30/71
MRGREC	8/30/71	VSORT	*
NEWNET	8/30/71	WGT	6/ 8/71
OPENFT	12/ 2/70	WGTA	6/ 8/71
OUTLLT	8/30/71	WGTLD	6/ 8/71
OUTLNT	8/30/71	WRT	8/30/71
OUTNET	8/30/71	WTLNT	6/ 8/71
OUTRIP	1/ 2/69	Overlay Structure	7/30/71

Labeled Common Control Sections: ALLIGN, CAPREP, CAPRES, CD, DELETE, FILES, GROUP1, HEADR, OUTDCB, SDATE, STOP, VOLTP

Library Subroutines: AXIS, DSQRT, EXP, LINE, LOG, NUMBER, PLOTS, SIN, SQRT, SYMBOL

*These programs have not been modified since the institution of the revision date policy on individual subroutines.

Figure 1: OVERLAY STRUCTURE FOR TEXAS SMALL NETWORK PACKAGE

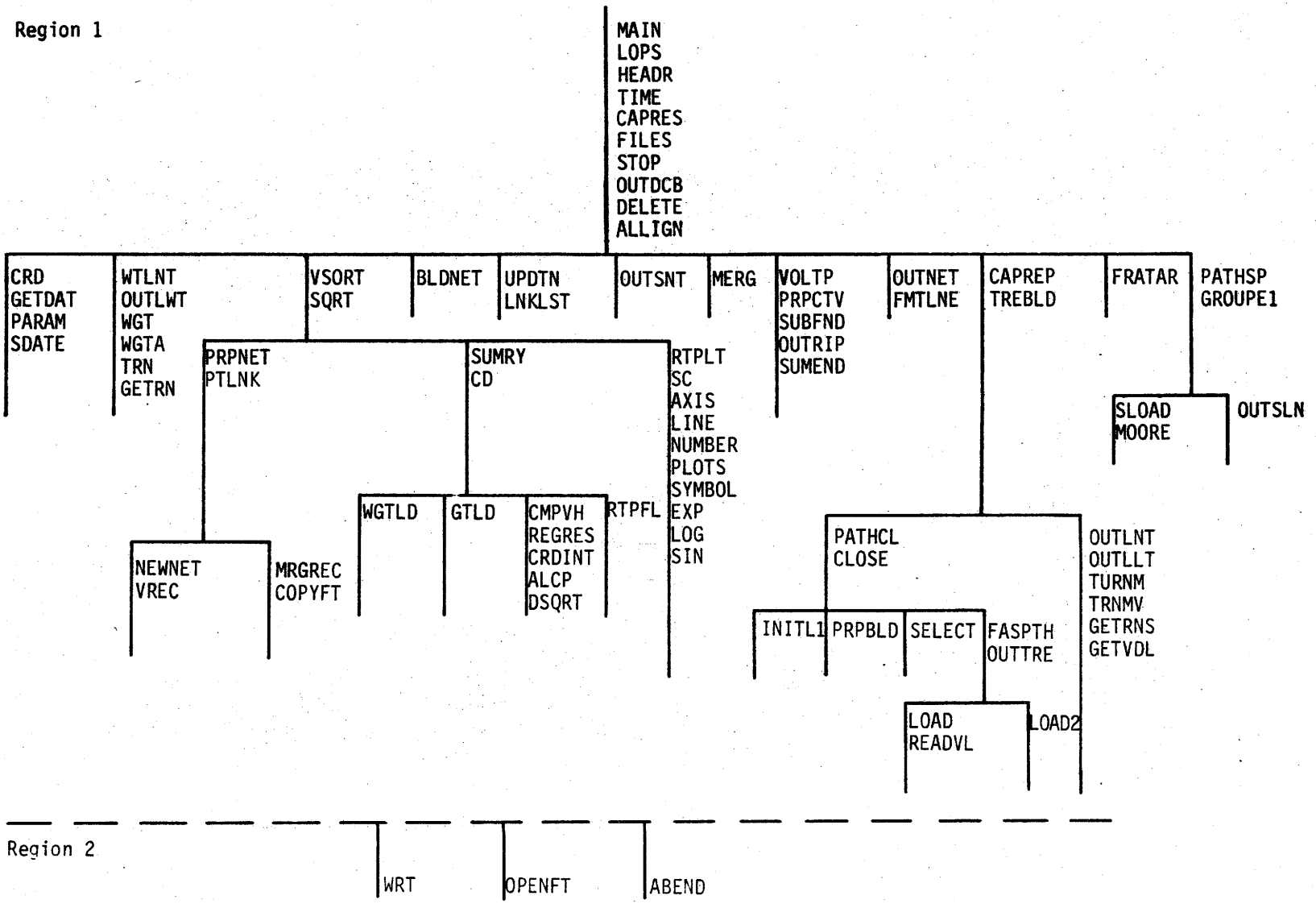
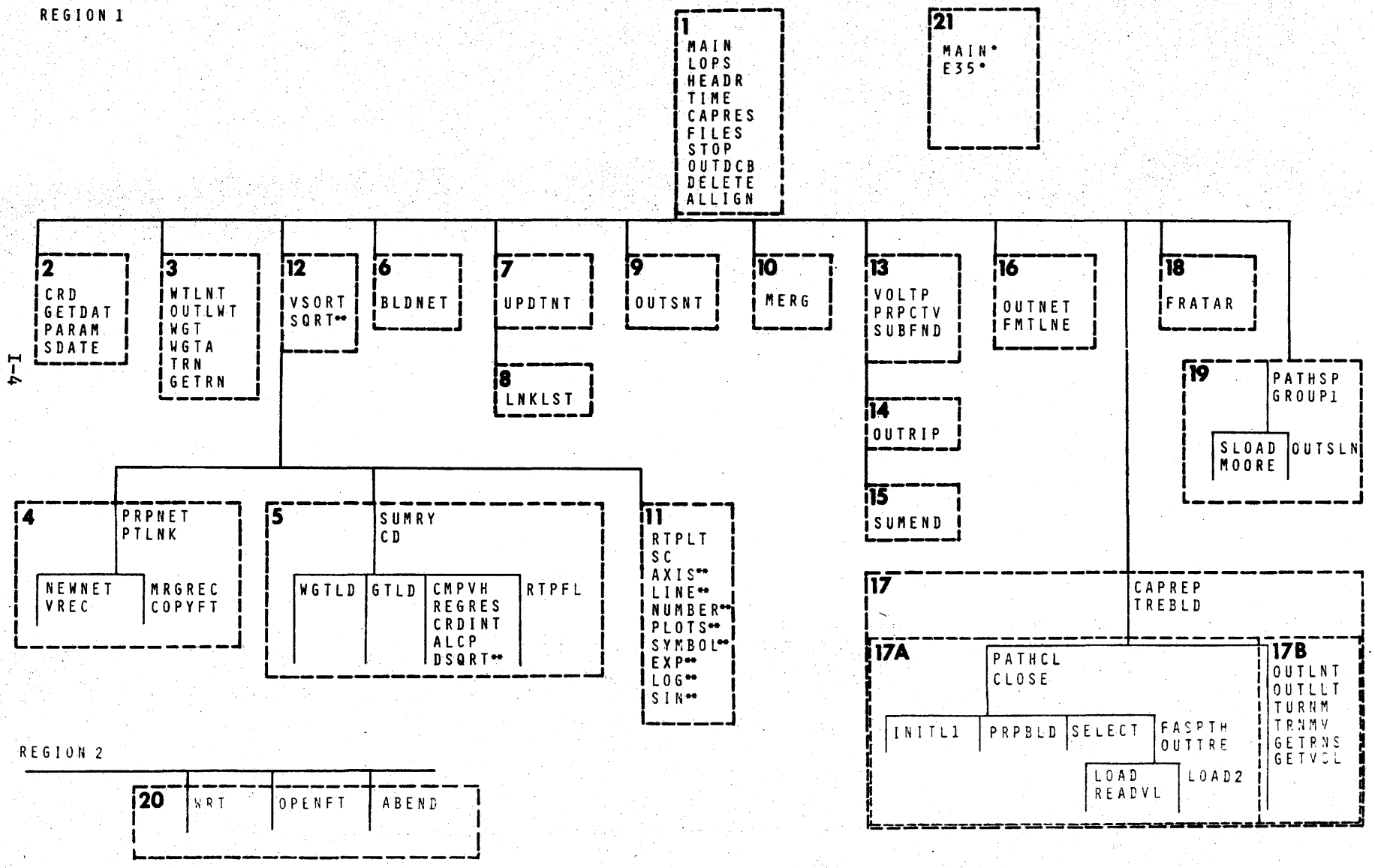


FIGURE 2: LOGICAL DIVISIONS FOR TEXAS SMALL NETWORK PACKAGE



*THESE PROGRAMS ARE NOT CONTAINED IN THE OVERLAY STRUCTURE

**LIBRARY SUBROUTINES

options described in the operating manual. On the other hand, it may be noted that logical division 17 contains eighteen control sections. It is within this division that trees are built, the network is loaded, and the loaded network is printed. For convenience, therefore, sixteen of the eighteen control sections in logical division 17 have been grouped into two logical subdivisions as seen in Figure 2.

It should also be noted that a number of the programs have multiple entry points. To avoid possible confusion, these programs along with the names of their other entry points are listed in Table 2.

TABLE 2: PROGRAMS WITH MULTIPLE ENTRY POINTS

PROGRAM	OTHER ENTRY POINTS
ABEND	PLOAD
FASPTH*	ENTER
GETVOL*	WGT, WGTA
LOAD2*	CLOSE, OPEN, WRITE, TRDCB, NBIN
LOPS*	LGRS, LGLS, LANA, LORA, LEX, LANAD, LANAL, LANAH
PRPNET	ASMNET, REVNET
PTLNK*	GTLNK
TREBLD	TREE, SELLD
WRT*	OPENFT, CLOSFT, OUTDCB

*Assembly language routines

L O G I C A L D I V I S I O N S

INTRODUCTION

The eighty-one control sections comprising the Texas Small Network Package have been grouped for the convenience of discussion, into twenty-one logical divisions. These logical divisions are not independent entities but are functional units or simply convenient groupings. There are three or more logical divisions associated with each of the program options available to the user except the $OUTPUT SELECTED LINKS option.

The documentation functions served by this section are:

- To identify the logical divisions associated with each of the user program options.
- To describe the relationship (i.e., calling sequence) between the logical divisions with regard to each of the user program options.
- To describe the functions performed by each of the logical divisions.
- To provide the calling sequence of the subprograms within each logical division.
- To provide sufficient information regarding the operation of each of the subprograms within a logical division so that the particular program(s) of interest may be identified.

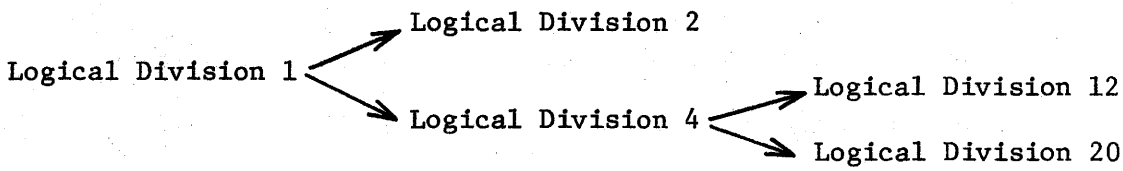
After having identified the particular program(s) of interest, the flowcharts (contained in Section IV) used in conjunction with the information concerning significant variables and arrays (Section VI) should provide the next level of documentation.

LOGICAL DIVISIONS AND USER PROGRAM OPTIONS

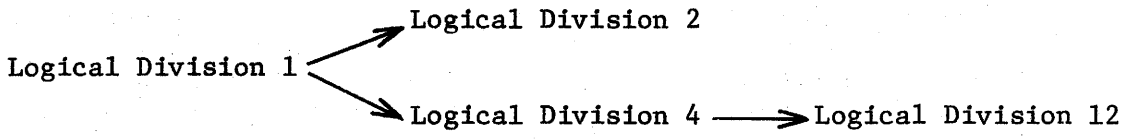
A cross-reference of the logical divisions and the user program options is provided by Table 3. As can be seen from this table, three or more logical divisions are associated with each of the user program options (except $OUTPUT SELECTED LINKS). It should likewise be noted that many of the logical divisions are associated with more than one of the user program options.

The relationships between each of the logical divisions under each of the user program options are illustrated in the following diagrams:

$PREPARE NETWORK
$ASSEMBLE NETWORK



$REVISE NETWORK



$OUTPUT NETWORK

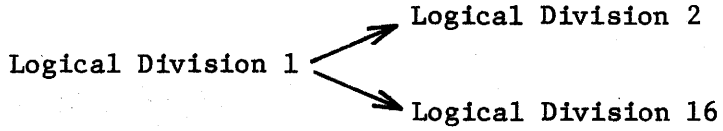
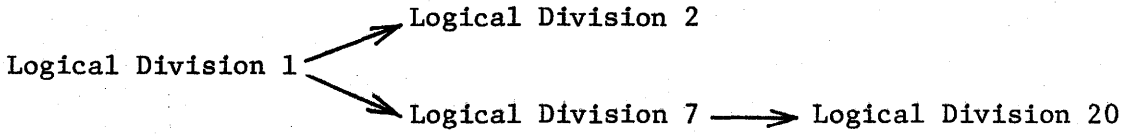


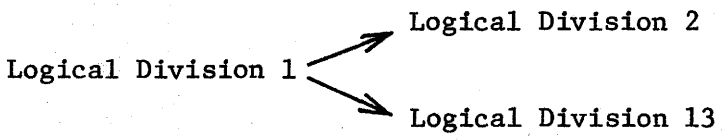
TABLE 3: CROSS-REFERENCE OF USER PROGRAM OPTIONS AND LOGICAL DIVISIONS

USER PROGRAM OPTIONS	LOGICAL DIVISIONS																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$PREPARE NETWORK	X	X		X								X								X	
$ASSEMBLE NETWORK	X	X		X								X								X	
$REVISE NETWORK	X	X		X								X									
$OUTPUT NETWORK	X	X														X					
$DELETE ASSIGNMENTS	X	X					X													X	
$PREPARE TRIP VOLUMES	X	X											X								
$OUTPUT TRIP VOLUMES	X	X												X							
$BUILD TREES	X	X															X				
$ASSIGN	X	X			X							X					X			X	
$ASSIGN SELECTED LINKS	X	X	X		X			X				X					X			X	
$ASSIGN SELF-BALANCING	X	X			X							X					X			X	
$OUTPUT SELECTED LINKS																					X
$PLOT ROUTE PROFILES	X	X									X	X									
$FRATAR FORECAST	X	X																X			
$SUM TRIP ENDS	X	X													X						
$MERGE	X	X								X											
$PREPARE SPIDER NETWORK	X	X				X															
$OUTPUT SPIDER NETWORK	X	X							X										X		
$ASSIGN SPIDER NETWORK	X	X																			

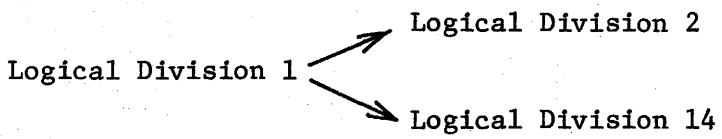
$DELETE ASSIGNMENTS



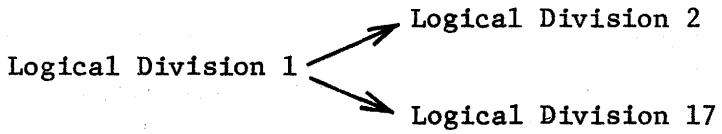
$PREPARE TRIP VOLUMES



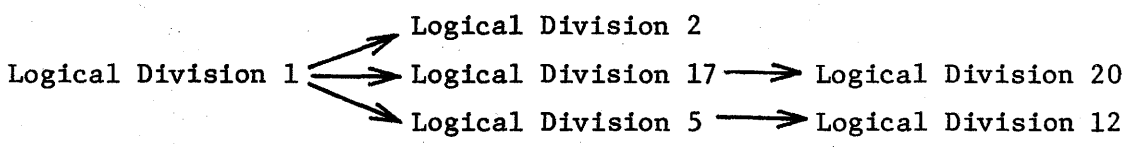
$OUTPUT TRIP VOLUMES



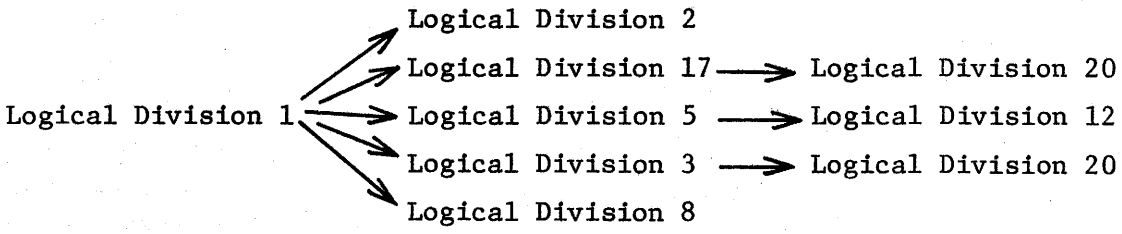
$BUILD TREES



$ASSIGN
$ASSIGN SELECTED LINKS



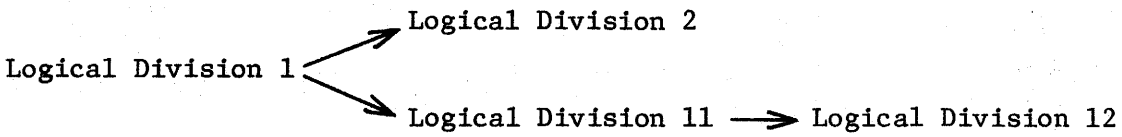
$ASSIGN SELF-BALANCING



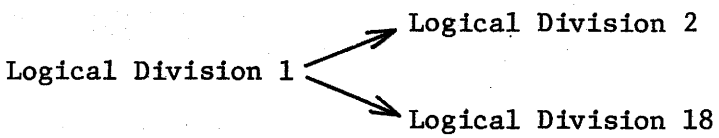
$OUTPUT SELECTED LINKS

Logical Division 21

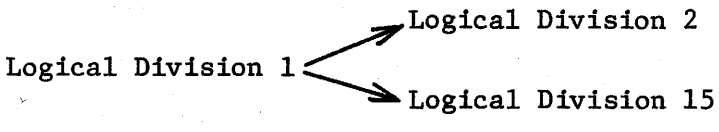
$PLOT ROUTE PROFILES



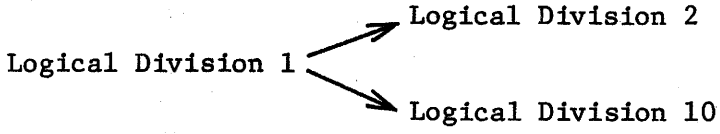
$FRATAR FORECAST



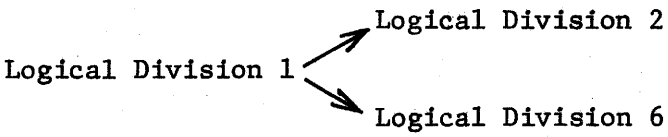
$SUM TRIP ENDS



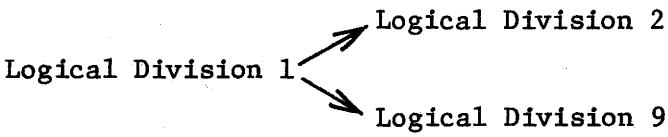
$MERGE



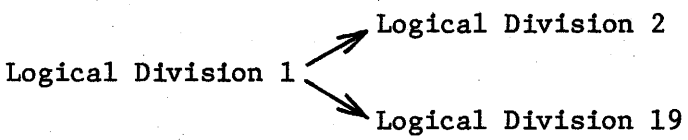
$PREPARE SPIDER NETWORK



$OUTPUT SPIDER NETWORK



$ASSIGN SPIDER NETWORK



DESCRIPTIONS OF LOGICAL DIVISIONS

The description of each of the logical divisions in the Texas Small Network Package has been divided into three sections. These sections describe the logical division's general function, the input/output requirements, the control sections used, the sequence of subroutines called,and provide a brief description of each of the subroutines (or control sections).

The first section, entitled "General", briefly describes the functions or operations performed by the logical division. It also lists the input required, output produced, and the control sections used by the logical division.

The second section, entitled "Sequence of Subroutines Called", provides a diagram illustrating the sequence of subroutines called during the execution of the logical division. This section not only provides a convenient "trace back" capability but identifies those control sections which are subroutines executed within the logical division. In addition, when the given logical division calls another logical division, the diagram identifies both the logical division and the subroutine called within that logical division.

The third section is entitled "Descriptions of Individual Control Sections". This section contains a brief description of the function of each of the control sections contained in the logical division.

LOGICAL DIVISION 1

General

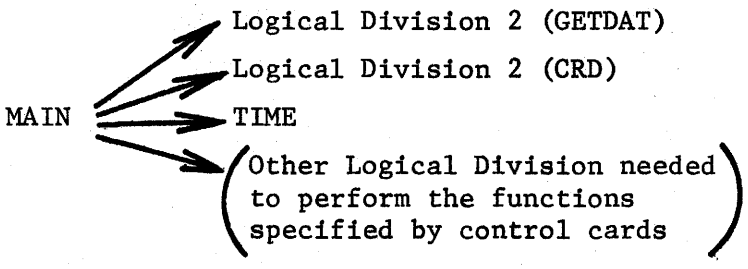
This division serves as the control program for the entire package. It first issues a call to Logical Division 2 (Subroutine GETDAT) to initialize the date. It also issues calls to Logical Division 2 (Subroutine CRD) to read and interpret control cards and unit control cards. The appropriate Logical Divisions are then called to perform the actions specified by the control cards. Because of the multiple usage of various logical divisions in the ASSIGN SELF-BALANCING process, the program MAIN also serves as the control program for this process. For convenience and efficiency, this division also contains small subroutines and labeled commons which are used by many of the other logical divisions.

Input: None

Output: Prints the difference in time of day of when each program specified by a Control card started and when it ended.

Control Sections: MAIN, TIME, CAPRES, FILES, HEADR, LOPS, ALLIGN, STOP, OUTDCB, DELETE

Sequence of Subroutines Called:



Descriptions of Individual Control Sections

ALLIGN: This labeled common forces a half word array used by subroutine MRGREC to a full word boundary.

DELETE: This labeled common contains one word used to sum the number of errors in the programs PREPARE NETWORK, ASSEMBLE NETWORK, and REVISE NETWORK.

OUTDCB: This labeled common has two arrays where data control blocks are built by subroutine OPENFT when this subroutine opens data sets.

STOP: This labeled common is not needed.

TIME: This subroutine returns the time of day in units of $\frac{1}{100}$ of a second.

CAPRES: This is a labeled common which is used by ASSIGN SELF-BALANCING.

FILES: This is a labeled common in which the variable unit numbers are stored.

HEADR: This is a labeled common used to store the date and the header from the last $HEADR card read.

LOPS: This is a control section which contains 9 function subroutines which are used for bit manipulation for packed data by other logical divisions.

MAIN: This is the main program for the entire package. Initially it issues calls to GETDAT (in Logical Division 2) and TIME to get the date and time the program began execution. It then performs the following steps iteratively (Until a $STOP control card is encountered or an end of data set is encountered on unit 5):

- A call is issued to subroutine CRD (in Logical Division 2) to read and interpret a control card.
- The appropriate subroutine(s) are called to execute the program specified by the control card.
- A call is issued to subroutine TIME to get the time of day.
- The time used by the execution of the program is calculated and printed.

LOGICAL DIVISION 2

General

This division is called by Logical Division 1. Although it contains the routine used to initialize the date, its primary purpose is to read and interpret control cards and unit control cards. When a unit control card is read, the appropriate variable unit number in labeled common FILES is changed. When a $HEADR card is encountered, the contents of columns 7 - 80 are placed in the array in the labeled common HEADR. If an invalid control card or unit control card is read, an error message is printed and the job is terminated. When a valid control card (other than a $HEADER card) is read, this division returns an integer which identifies the control card read.

INPUT: Control cards and unit control cards on Unit 5.

OUTPUT: Prints all valid and invalid control cards and unit control cards. Variable unit numbers are printed if any were changed by a unit control card.

Control Sections Used: CRD, PARAM, GETDAT, SDATE

Sequence of Subroutines Called

Descriptions of Individual Control Sections

CRD: This subroutine reads control cards and unit control cards and sets an integer which is returned to the main program indicating the

control card encountered. When a unit control card is encountered, the subroutine PARAM subroutine is called. After returning from PARAM, another control card is read. When a $HEADR card is encountered, the information in columns 7 - 80 is placed in the HEADR labeled common and another control card is read. If an invalid control card or unit control card is encountered, an error message is printed and the job is terminated.

PARAM: This subroutine interprets unit control cards read by CRD and changes the variable unit numbers specified in the FILES labeled common.

GETDAT: This subroutine gets the date from the operating system with a TIME macro and converts it to a twelve byte literal in the form:

XXX YY, ZZZZ

where:

XXX = abbreviation of the month (3 bytes)

YY = day of the month (2 bytes)

ZZZZ = year (4 bytes)

This subroutine is called by the program MAIN.

SDATE: This labeled common contains the date of the last modification to the package and it is printed in a message after every control card recognized by subroutine CRD.

LOGICAL DIVISION 3

General

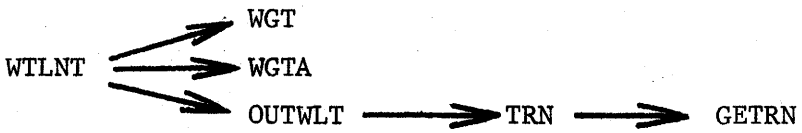
This section calculates the weighted assignment and the loaded network produced by combining the assignments from each iteration in ASSIGN SELF-BALANCING. The weighted assignment is calculated by applying the iteration weights (percentages) to their respective assigned volumes from each iteration and summing. The resulting link volumes are rounded by adding 50 and dividing by 100. The turn volumes are calculated before they are rounded.

Input: Unit 3, unit NEWNET

Output: The loaded network produced by the weighted assignment is written on unit 6 (the print data set)

Control Sections used: WTLNT, OUTWLT, WGT, WGTA, TRN, GETRN

Sequence of Subroutines Called



Descriptions of Individual Control Sections

WTLNT: This subroutine reads the link volumes and turn volumes from the individual iterations and calls subroutine WGT and WGTA to apply the iteration weighting and sum the resulting volumes. The subroutine OUTWLT is called to print the weighted loaded network.

WGT: This subroutine multiplies from 1 to 4000 volumes by an integer percent and puts the result in another array.

WGTA: This subroutine multiplies from 1 to 4000 volumes by an integer percent and sums the results into another array to form weighted volume sums.

OUTWLT: This subroutine uses the weighted link volume sums and the weighted turn volume sums to print a weighted loaded network. This subroutine calls subroutine TRN for each node which is connected in the network to get the weighted link volumes and calculate the weighted turn volumes which were not saved.

TRN: This subroutine gets the weighted directional and nondirection volumes and calculates the weighted turn volumes which were not saved. It also flags the turn volumes to be printed.

GETRN: This subroutine places those weighted turn volumes which were be saved in the turning movements matrix.

LOGICAL DIVISION 4

General

This section basically performs the following functions:

- $PREPARE NETWORK
- $ASSEMBLE NETWORK
- $REVISE NETWORK

Input: Link data cards or link data revision cards from the INLNK data set

Output: New or revised Flexible Record Data Set on the NETWORK data set

Control Sections Used: PRPNET, PTLNK, NEWNET, VREC, MRGREC, and COPYFT

Sequence of Subroutines Called

$PREPARE NETWORK

PRPNET (entry point PRPNET) → NEWNET → VREC

$ASSEMBLE NETWORK

PRPNET (entry point ASMNET) → NEWNET → VREC

$REVISE NETWORK

PRPNET (entry point REVNET) → NEWNET → MRGREC → COPYFT

Descriptions of Individual Control Sections

PRPNET: This is the control program for this section and defines storage for the arrays and variables to be shared by the other programs in this section.

PTLNK (and GTLNK): Commonly called "Put Link" or Get Link," this program has two entry points (i.e., PTLNK and GTLNK). It is a utility program which packs and unpacks the 22-byte records used to save the information from link data cards. This is the format in which the one-way links are sorted and are written on units 3 and 11.

NEWNET: Basically, this program inputs, sorts, and edits the link data cards. Due to array limitations, this program will input and sort up to approximately 2727 link data cards (recall that each link data card produces 2 link records). This program will handle up to 3 groups of approximately 2727 link data cards each with the first two sort groups saved on disks and the last saved in core. These groups are later merged by VREC. This program also outputs any node names on logical unit 4. This program also performs some preliminary edit checks to determine the validity of data. The preliminary edit checks include:

- Node number in range (i.e., 1 $\leq$ node number $\leq$ last Freeway Node Number)
- Valid time or speed code (i.e., T or S)
- Valid directional code (i.e., 0, 1, +, -)
- Calculates either time or speed and determines if impedance is less than or equal to 163.83 "minutes".

VREC: This program performs the following functions:

- If there are more than one set of sorted link data records produced in NEWNET (i.e., more than approximately 2727 link data cards), the links are then merged.
- Performs various edit checks which includes:
  a. Check for duplicate links
  b. Check to determine if each node appears to be properly connected to network (Note: basically this only checks to see that each link is connected to another node. It does not check for network fragmentation since this can presumably be found by building test trees).

- Prepares and outputs "Flexible Record Data Set".
- Also inputs and merges 22 byte link records with link records in core if there were more than 2727 link data cards.

MRGREC: Essentially this is just a modified version of VREC for the $REVISE NETWORK. It performs the same functions as VREC except it can merge up to 4 data sets instead of 3 (the additional data set is the old Flexible Record Data Set which is being revised).

COPYFT: Again, this program is only used in conjunction with $REVISE. NETWORK. This program performs the following functions:

- Updates the field in the Flexible Record Data Set which contains the number of one-way links.
- Copies the Flexible Record Data Set in VB instead of VBS record format (note: FORTRAN unformatted WRITE requires either VS or VBS).

LOGICAL DIVISION 5

General

This section reads the Flexible Record Data Set from the unit NEWNET and produces the following tables:

- Cross Classification of V/C Frequencies from Last Two Assignments
- Cross Classification of Link Counts by V/C Ratio from Last Two Assignments
- Jurisdiction Summary
- Jurisdictional/Functional Cross Classification of Assigned Volumes
- Jurisdictional/Functional Cross Classification of Counted Volumes
- Jurisdictional/Functional Cross Classification of Link Capacities
- Comparison of Assigned Volumes with Counted Volumes
- Comparison of Assigned Volumes with Link Capacities
- Comparison of Assigned Volumes (from last assignment) with Assigned Volumes (from assignment before last)
- Iteration Weighting-Multiple Regression Analysis
- Link Volumes
- Iteration Weights Applied
- Corridor Intercept Tables
- Route Profiles
- List of Volumes and Impedances for Updated Links

Some of these tables are printed only when certain conditions are met (see section on OTHER INFORMATION).

Input: Unit NEWNET.

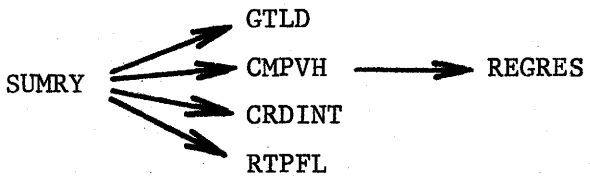
Output: The tables listed inthe general section above and Unit ROUTE.

Control Sections: SUMRY, CD, WGTLD, GTLD, CMPVH, REGRES, CRDINT, ALCP, RTPFL

Sequence of Subroutines Called

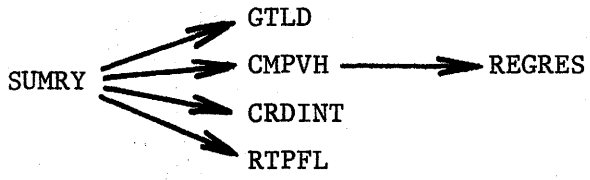
$ASSIGN and

$ASSIGN SELECTED LINKS

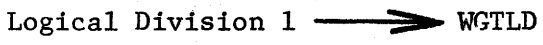


$ASSIGN SELF-BALANCING (iterations 1 - 5, and the calculated weighted assignment if "WGT" is specified on the *TURN card)

$ASSIGN SELF-BALANCING (Weighted assignment made from weighted impedances if "WGT" is specified on the *TURN card, otherwise calculated weighted assignment)



$ASSIGN SELF-BALANCING (after last iteration)



Descriptions of Individual Control Sections

SUMRY: This is the control program for the summaries produced after an assignment. The subroutines called by SUMRY are determined by three logical variables. One of the logical variables, SUM, if true causes GTLD to produce a weighted assignment on unit NETWORK and produce all tables and comparisons from this weighted assignment. Subroutine ALCP is only called if logical variable RES is true. If logical variable RTP is false then the corridor intercept and route profile tables are skipped.

GTLD: This subroutine prints the V/C cross classification table if there are two or more assignments on unit NEWNET. It computes the summations necessary for the tables printed by subroutine CMPVH and for the curve fit printed by subroutine ALCP. It saves corridor intercept information in core in labeled common CD. It writes route profile records on Unit ROUTE. If logical variable SUM is true, GTLD calculates weighted directional volumes and updates the flexible data record writing it on unit NETWORK. All comparisons and tables are made from the weighted directional volumes if SUM is true.

CMPVH: This subroutine prints the Jurisdiction Summary or the Jurisdictional/FUNCTIONAL Cross Classification Tables and the three Comparison of Assigned Volumes with link volumes, Counted volumes, and Capacities.

REGRES: This subroutine performs a linear regression analysis and prints the results of this analysis.

CRDINT: This subroutine calls VSORT (which sorts the corridor intercept records) and prints the corridor intercept tables.

ALCP: This subroutine performs a multiple regression analysis to determine the iteration weighting for the ASSIGN SELF-BALANCING process and prints the results of this analysis. Only the links with a non-zero count (or capacity depending on which is specified) are considered and centroid connectors are ignored. The count (or capacity) is the dependent variable and the assigned directional volumes from each of the iterations are the independent variables in the analysis.

RTPFL: This subroutine reads the route profiles from unit ROUTE and prints the route profile tables.

CD: This is a labeled common area used to save the corridor intercept records when GTLD is run until subroutine CRDINT runs.

LOGICAL DIVISION 6

General

This division is called by Logical Division 1 and performs the $PREPARE SPIDER NETWORK program.

Input: Link data cards for a spider network from unit INLNK.

Output: Printed errors, index and link records on unit 1, node names on unit 4, and a link speed frequency table

Control Sections used: BLDNET

Sequence of Subroutines Called

Logical Division 1 ⟶ BLDNET

Descriptions of Individual Control Sections

BLDNET: Subroutine BLDNET reads the link data cards and writes index and link records on unit 1 and node names on unit 4. The link data are edited and errors printed. The network speed is calculated and the link speed frequency table is prepared and printed.

LOGICAL DIVISION 7

General

This division is called by Logical Division 1 and uses the WRT subroutine in Logical Division 20. It basically performs the $DELETE ASSIGNMENTS program. As may be recalled, the $DELETE ASSIGNMENTS program can delete up to 20 assignments from the NETWORK data set and can also replace the impedances to be used on the next assignment with the impedances used on any previous assignment (even if the assignment is being deleted), or it can modify the impedances according to the impedance adjustment function. The WRT subroutine is used to output the flexible record data set in the desired record format type (i.e., V or VB).

Input: Old flexible data record (unit 12), and DELETE ASSIGNMENTS parameter cards from unit 5 (i.e., *IMPEDANCE, *ADJUST, *DELETE, and *END cards).

Output: Updated flexible data record (unit NETWORK).

Control Sections: UPDTNT

Sequence of Subroutines Called

UPDTNT ⟶ WRT (Logical Division 20)

Descriptions of Individual Subroutines

UPDTNT: This subroutine basically performs the functions of the $DELETE ASSIGNMENTS program. The specific functions performed are, of course, determined by the parameter cards supplied by the user (i.e., the *IMPEDANCE, *ADJUST, *DELETE, and *END cards). It should be noted

that the last parameter card must be the *END card. It should further be noted that if the *END card is the only parameter card provided then the flexible record data set will simply be copied on unit NETWORK.

The WRT subroutine (in Logical Division 20) is used to write the records (of the flexible record data set) on the unit NETWORK using the record format type V or VB. The WRT subroutine changes the record format type specified in the DCB parameter of the DD card for the unit NETWORK as either VS or VBS to V or VB respectively. Effectively, OPENFT removes the span parameter, S, from the DCB. This was implemented to avoid problems caused by the FORTRAN Input/Output requirements of certain versions of the Operating System.

LOGICAL DIVISION 8

General

This division prints the links which have non-zero count or capacity fields (whichever has been specified) during the $ASSIGN SELF-BALANCING program. The directional link volumes and the link impedance are listed for each iteration and for the calculated weighted assignment and the optional assignment made with the weighted impedances. The count or capacity field is also listed.

Input: Flexible record data set on unit NEWNET.

Output: Printed list of links with link volumes and impedances for which the link count or link capacity field, whichever was used, is non-zero.

Control Sections Used: LNKLST.

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ LNKLST

Descriptions of Individual Control Sections

LNKLST: The function of this subroutine is listed in the general section above.

LOGICAL DIVISION 9

General

This division is called by the program MAIN (in Logical Division 1) and performs the $OUTPUT SPIDER NETWORK program.

Input: Unit 1.

Output: Printed spider network.

Control Sections Used: OUTSNT

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ OUTSNT

Descriptions of Individual Control Sections

OUTSNT: This program reads a spider network from unit 1 and formats it with from 1 to 8 links per line. The program also prints the network speed. This program can not read a flexible data record.

LOGICAL DIVISION 10

General

This division is called by the program MAIN (in Logical Division 1) and performs the $MERGE program. It can be used to merge from two to six trip matrices.

Input: Units MERGIN(1) to MERGIN(N)
(where N is between 2 and 6)

Output: Unit MRGOUT

Control Sections Used: MERG

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ MERG

Descriptions of Individual Control Sections

MERG: This subroutine reads a merge parameter card which specifies the number of data sets to merge. The MERGIN and MRGOUT units must have previously been specified on a unit control card. The parameter records from these data sets are examined and the first zone of each subnet must be the same. If any are different an error message is printed and the program stops. The largest last zone of each subnet is used for the merged trip matrix which is written on MRGOUT. Then the trip matrices are summed and written on unit MRGOUT.

LOGICAL DIVISION 11

General

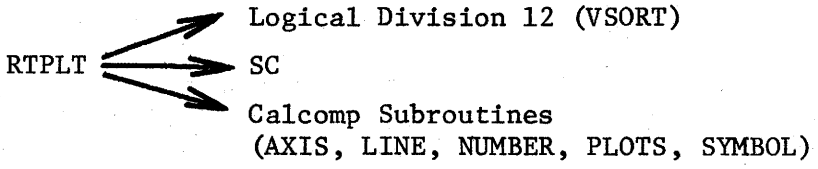
This division is called by the program MAIN (in Logical Division 1) for the $PLOT ROUTE PROFILES program. It prints the route profiles from a previous run of ASSIGN, ASSIGN SELECTED LINKS, or ASSIGN SELF-BALANCING. It also prepares calcomp plots of the routes with assignments, counts, or link capacities specified.

Input: Unit ROUTE, parameter cards to specify routes and assignments

Output: Printed route profiles of all routes and a calcomp plot tape.

Control Sections Used: RTPLT, SC, and calcomp subroutines.

Sequence of Subroutines Called



Descriptions of Individual Control Sections

RTPLT: This subroutine reads the route parameter card specifying which routes are to be plotted. It then reads the parameter card specifying which assignments, counts or capacities are to be plotted. It then reads the ROUTE data set and prints the route profiles and plots those which have been specified.

SC: This subroutine is used to round the scaling factor.

LOGICAL DIVISION 12

General

This division contains the subroutine VSORT which performs an in-core sort. It is used by Logical Divisions 4, 5, and 11.

Input: Unsorted data in core in records of from 1 to 256 bytes/record.

Output: Sorted records in core.

Control Sections Used: VSORT

Sequence of Subroutines Called

Logical Division (4, 5, or 11) ⟶ VSORT

Descriptions of Individual Control Sections

VSORT: This subroutine sorts records in core. The first argument in the calling sequence is the address of the array of records to be sorted. The second argument is the number of records. The third argument is the length of each record in bytes (must be between 1 and 256 bytes). The fourth argument is the length of the sort key in bytes (must be between 1 and 256 bytes) which can not be longer than the record length. The sort key starts at the first byte of the record. The sort key is treated as an unsigned binary number and the records are sorted into ascending order on the sort keys.

LOGICAL DIVISION 13

General

This division is called by the program MAIN (in Logical Division 1). It inputs the card trip volume records; checks to see that they are in ascending order on origin and destination zones; and builds a trip matrix which is outputted on unit CTVOUT.

Input: Parameter card on unit 5, card trip volume records on unit CTVIN.

Output: Trip matrix on unit CTVOUT.

Control Sections used: PRPCTV, SUBFND, VOLTP

Sequence of Subroutines Called

PRPCTV ⟶ SUBFND

Descriptions of Individual Control Sections

PRPCTV: This is the main part of the code for this logical division. It reads the parameter card which specifies the volume field (of the three available) to be used. This parameter card also specifies the number of subnets and the first and last zone of each subnet.

After the parameter card is read, the trip volume records are read. The program checks for records which are out of sort with regard to the origin and destination zone numbers. It also checks to see that both zones are in the zone ranges specified for the subnets by calling subroutine SUBFND, and checks for duplicate origin and destination zone numbers. It writes a trip matrix on unit CTVOUT of those trips for which there were no errors.

SUBFND: This subroutine determines the subnet containing the origin zone and the subnet containing the destination zone. It then verifies that both the origin and destination zone numbers are within the zone ranges specified on the parameter card.

VOLTP: This is a labeled common area used by subroutine PRPCTV.

LOGICAL DIVISION 14

General

This logical division is called by the program MAIN (in Logical Division 1) and performs the $OUTPUT TRIP VOLUMES program. It essentially prints the trip matrix contained on Unit CTVOUT.

Input: Unit CTVOUT.

Output: Printed trip matrix.

Control Sections used: OUTRIP

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ OUTRIP

Descriptions of Individual Control Sections

OUTRIP: This subroutine reads a trip matrix from unit CTVOUT and prints it with each origin zone starting on a new page. It prints 10 destination volumes per line. The zone numbers printed run from the first zone number for a subnet to the last zone number for that subnet in groups of 10. If a group of ten destination volumes are all zero they are not printed. The origin zones are considered in sequential order.

LOGICAL DIVISION 15

General

This division is called by the program MAIN (in Logical Division 1) and performs the $SUM TRIP ENDS program.

Input: Trip matrix on unit CTVOUT.

Output: A printed table.

Control Sections Used: SUMEND

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ SUMEND

Descriptions of Individual Control Sections

SUMEND: This subroutine performs a summation of a trip matrix by rows and columns exclusive of the diagonal elements (i.e., the intrazonal volumes). The number of non-zero trip volumes are also counted. A table is then printed containing a summary of the trip volume characteristics for each zone.

LOGICAL DIVISION 16

General

This division is called by the program MAIN and performs the $OUTPUT NETWORK program.

Input: Unit NETWORK.

Output: Printed network description.

Control Sections Used: OUTNET and FMTLNE.

Sequence of Subroutines Called

OUTNET ⟶ FMTLNE

Descriptions of Individual Control Sections

OUTNET: This subroutine writes the page headings and calls subroutine FMTLNE to format each line of the network. It reads the link records from unit NETWORK and calls subroutine FMTLNE to format this data for from 1 to 4 links per line. The subroutine prints 50 nodes per page. If a whole page of node numbers to be printed are not included in the network (i.e., they have no connecting nodes), the printing of the page is suppressed. The data for a link that is printed is ANODE, BNODE, jurisdiction, shaft, arrow, link speed, link distance and link impedance. The link impedance printed is the link impedance which will be used if this flexible data record is used as unit NETWORK when the next assignment or BUILD TREES is run.

FMTLNE: This subroutine formats the link data of from one to four links with the same ANODE to be printed on one line. If a link is a dummy one-way link the literal ONE-WAY is printed for it along with its BNODE and the other data for this link is not printed.

LOGICAL DIVISION 17

General

This division is comprised of a control program and two logical subdivisions (17A and 17B). It basically performs the following functions.

- $BUILD TREES
- $ASSIGN
- $ASSIGN SELF-BALANCING
- $ASSIGN SELECTED LINKS

Logical subdivision 17A performs the build trees and load trips functions. Logical subdivision 17B is used to print the loaded network and to update the flexible record data set.

Input: The input required by programs in this division (depending upon the function being performed) is as follows:

- Flexible Record Data Set
- Trip Table
- *TURN cards
- *TREE cards
- Parameter cards for ASSIGN SELECTED LINKS (if needed)

Output: The output from this division consists of one or more of the following (depending upon the functions being performed):

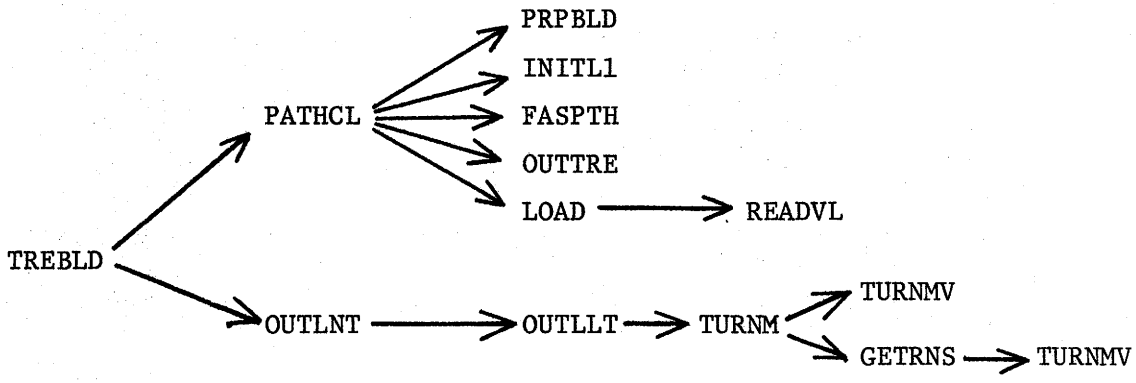
- Selected trees are printed (as specified by the *TREE cards)
- The loaded network is printed (except for certain iterations in the assigned self-balancing process)
- A new flexible data set is prepared (except when $BUILD TREES is run)

- Selected link interchanges are output on a sequential data set (under assign selected links option)
- A separation matrix is prepared.

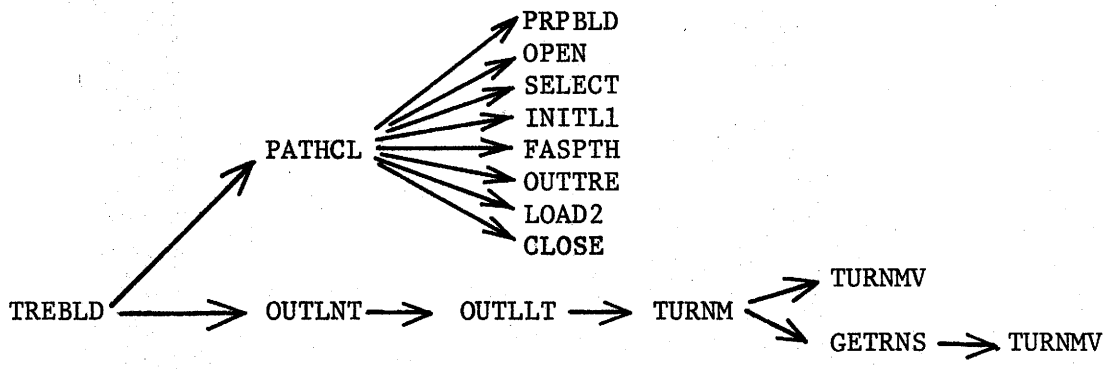
Control Sections Used: CAPREP, TREBLD, PATHCL, CLOSE, INITL1, PRPBLD, SELECT, FASPTH, OUTTRE, LOAD, READVL, LOAD2, OUTLNT, OUTLLT, TURNM, TRNMV, GETRNS, GETVOL

Sequence of Subroutines Called

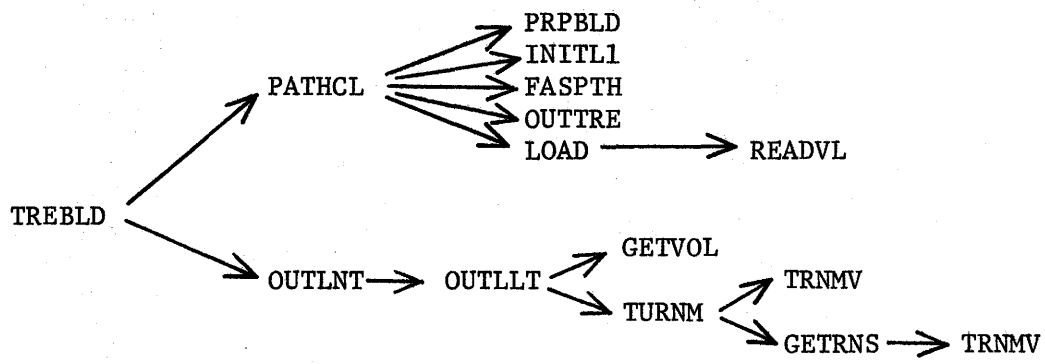
$ASSIGN:



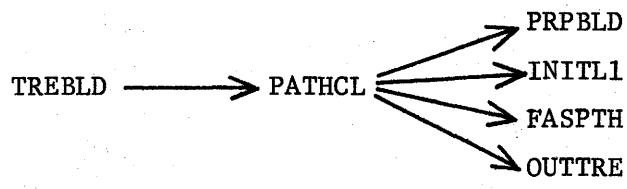
$ASSIGN SELECTED LINKS:



$ASSIGN SELF-BALANCING:



$BUILD TREES:



Descriptions of Individual Control Sections

TREBLD: This is the basic control program for Logical Division 17. It contains three entry points; TREBLD, TREE, and SELLD. The TREBLD entry point is used for $ASSIGN and for the various iterations in $ASSIGN SELF-BALANCING. The TREE entry point is used for $BUILD TREES. The SELLD entry point is used for $ASSIGN SELECTED LINKS.

Basically this control section sets 2 or 3 logical variables and calls PATHCL (the control program for logical division 17A). After the programs in logical division 17A have been executed, TREBLD then calls OUTLNT (the control program for logical division 17B).

CAPRES: This is a labeled common used in the ASSIGN SELF-BALANCING process, which is available to all programs in this division. It is used to save the information from the *TURN and *TREE cards.

PATHCL: This subroutine controls the assignment process. It reads the NETWORK data set and extracts the part of the network used for building trees. It calls subroutines that build trees, load trips, and output trees as needed to perform an assignment. It also reads the *TURN and *TREE cards by calling PRPBLD. If an $ASSIGN SELECTED LINKS is specified, it calls SELECT to read the selected link parameter cards and mark the selected links. It also writes the separation matrix. If $BUILD TREES is specified, it simply builds the trees and skips the loading of trips.

CLOSE: This subroutine closes data set SELTRP and releases its buffers.

OPEN: This subroutine is in control section CLOSE and it opens data set SELTRP.

INITL1: This subroutine initializes the volumes assigned to the network to zero and it builds the turn index array used by subroutine LOAD or LOAD2 in the assignment and by subroutine OUTLLT in presenting the loaded network. It also checks to see that there are less than 4000 nodes, 1200 centroids, 16000 one-way links, and 20,000 turning movements.

PRPBLD: This subroutine reads and examines the *TURN and *TREE cards used in BUILD TREES, ASSIGN, ASSIGN SELECTED LINKS, and ASSIGN SELF-BALANCING. The centroid numbers to build trees for are put in two arrays and a logical array is set to specify if the trees are to be printed.

SELECT: This subroutine reads the parameter cards (other than the *TURN and *TREE cards) which are used by ASSIGN SELECTED LINKS. It examines the cards for errors; prints the actions specified and sets flags on the selected links; it writes parameter records for the selected links; and, it also sets a logical variable to specify if the loaded network should be printed.

FASPTH: This subroutine builds one minimum path tree from the origin in its calling sequence for each call to it. It returns the path of the tree and the cumulative link impedances to reach each node or centroid in the path.

OUTTRE: This subroutine prints the path and cumulative impedance to reach each node and centroid in one minimum path tree for each call to it.

LOAD: This subroutine is called once for each tree to load all trips with an origin at the home zone of the path. This subroutine calls subroutine READVL to read trip volume records. This subroutine must be called once for each tree that is to be loaded.

READVL: This subroutine is called by subroutine LOAD to read trip volume records and returns the trip volume record and a flag indicating whether an end of data set has been reached on unit CTVOUT.

OUTLNT: This is the first level program in division 17B, it defines arrays and calls subroutine OUTLLT.

OUTLLT: This is the basic control program in this division. If an ASSIGN SELF-BALANCING program is being run in iterations 1 - 5 this subroutine calls subroutine GETVOL to get directional link volumes as full word integers and turning movements as full word integers and writes these on Unit 3.

For all calls to this subroutine it also reads the flexible data record from unit NETWORK and writes an updated flexible data record on unit NEWNET in which the nondirectional assigned volumes and link impedances for the present assignment are added to the flexible data record. If the present assignment is an ASSIGN SELF-BALANCING iteration 1 through 5 then the link impedance to be used on the next assignment is updated if counts were specified and the link has a non-zero count or if capacities are used and the directional link volume is greater than the capacity and the capacity field is non-zero.

The program also prints the assignment (except for ASSIGN SELF-BALANCING iterations 2 - 5 and for ASSIGN SELECTED LINKS when no output is specified).

TURNM: This subroutine gets the directional and nondirectional link volumes for a node, gets the turning movements which were saved, and calculates the turning movements which were not saved. It also marks which turn volumes are to be printed.

TRNMV: This function adds the two indexes supplied to it to form a single index to get the turn volume or directional link volume or index and flag. If it gets an index this index is used to get the actual volume from the overflow array.

GETRNS: This subroutine places the saved turn volumes in the turning movements matrix.

GETVOL: This subroutine gets from 1 to 4000 link volumes or turning movements and places them in a full word integer array.

LOGICAL DIVISION 18

General

This division is called by the program MAIN (in Logical Division 1) and performs the $FRATAR FORECAST program.

Input: Parameter card and growth factor cards on unit 5 and trip matrix on unit CTVOUT.

Output: Unit FRATAR. (Variable unit number CTVOUT is set equal to unit FRATAR after the program is run.) A table of iteration growth factor frequencies is also printed for each iteration.

Control Sections Used: FRATAR

Sequence of Subroutines Called

Logical Division 1 (MAIN) ⟶ FRATAR

Descriptions of Individual Control Sections

FRATAR: This subroutine reads a deck of zonal growth factors and uses Fratar's method of successive approximations to generate a forecasted trip matrix. Each approximation constitutes one iteration; the number of repetitions is governed by either an iteration limit or a deviation limit.

LOGICAL DIVISION 19

General

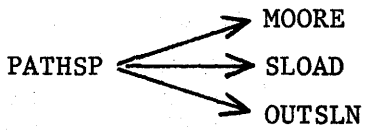
This division is called by the program MAIN (in Logical Division 1). It performs a spider network assignment and prints the loaded network. Each node in the spider network may have up to 8 links and no turn movements are printed in the loaded network output. Trees are built through both nodes and centroids.

Input: Unit 1 (index records and link records) node names on unit 4, trip matrix on unit CTVOUT.

Output: Printed loaded spider network.

Control Sections Used: PATHSP, GROUP1, SLOAD, MOORE, OUTSLN.

Sequence of Subroutines Called



Descriptions of Individual Control Sections

PATHSP: This subroutine reads the network from unit 1. It then builds trees by calling subroutine MOORE for every zone which has trips with that origin zone and it loads the trips read from unit CTVOUT on each tree immediately after the tree is built. Subroutine OUTSLN is then called to print the loaded network.

MOORE: This subroutine builds a minimum path tree from one origin centroid each time it is called.

SLOAD: This subroutine loads the trips for one origin zone in the network. It also reads the next trip record from unit CTVOUT and, if it has the same origin, it loads it also. It continues to read trip records and load the trips until it reaches an EOD on unit CTVOUT or a trip record with a different origin zone is encountered. It sets a logical variable EOF (EOF is set to true when an EOD has been reached on unit CTVOUT) and returns control to the calling program.

OUTSLN: This subroutine prints the loaded network.

GROUP1: This labeled common is used to group two scratch half word arrays of 3500 half words each used by subroutine MOORE to make a full word array of 3500 words which is used by subroutine SLOAD. It also forces the array used by SLOAD to a full word boundary.

LOGICAL DIVISION 20

General

This logical section contains the assembly language subroutines used to write records in record format VB on FORTRAN units which can be read with FORTRAN unformatted read statements. These subroutines were necessary because of an error in version 18 of IBM 360/OS which occasionally caused extra data to be read when data sets written with record format VBS were read. These subroutines are probably faster than FORTRAN write statements because they use the Queued Sequential Access Method whereas Fortran uses Basic Sequential Access Method; and, in addition, the assembly language subroutine WRT requires only one call for each list whereas FORTRAN generates one call for each variable and one call for each variable for each iteration in an implied DO loop. This logical division also contains subroutine ABEND which is called several places to write the message ERROR and a number where no error in the program operation is normally expected.

Input: none

Output: One record for each call to subroutine WRT.

Control Sections Used: OPENFT, CLOSFT, ABEND, and WRT

Sequence of Subroutines Called

The subroutines in this division may be called from many points in the package. The following, therefore, summarizes the subroutines which may execute a call to the subroutines in this division:

- ABEND may be called by TRN and TURNM.
- CLOSFT may be called by GTLD, OUTLLT, UPDTNT, and VREC.

- OPENFT may be called by GTLD, OUTLLT, UPDTNT, and VREC.
- WRT may be called by GTLD, OUTLLT, UPDTNT, and VREC.

Descriptions of Individual Control Sections

OPENFT: This is an assembly language subroutine to open a FORTRAN type DDname. The DCB is built in one of two areas (specified by either a 1 or a 2 as the first argument) in the control section OUTDCB. The FORTRAN unit number is specified by the second argument and the DDname used is FTXXF001 where the XX is the integer from the second argument. The data set is opened twice. The first time it is opened the DCB information from the DD card is obtained and the data set is closed.
The spanned code is then removed from the DCB in core and the data set is reopened. For this reason the RLSE subparameter should not be used in the SPACE allocation parameter on data sets which are used as unit NETWORK, unit NEWNET, or unit ROUTE because the primary extent is all released except for 1 track when the first CLOSE macro is executed by subroutine OPENFT.

CLOSFT: This subroutine closes the data set whose DCB is in the OUTDCB CSECT. The DCB is indexed by either a 1 or a 2 which is the argument in the call to CLOSFT.

WRT: This subroutine writes one logical record on the unit which is pointed to by the "opened" DCB in CSECT OUTDCB. The DCB is indexed by either a 1 or a 2 as the first argument in the call to subroutine WRT. The logical record written may be made up of one or more record segments. This subroutine uses the PUT macro with the locate mode to get the address of each new record segment. The rest of the calling sequence of subroutine

WRT is variable and is made up of a variable number of arguments which are in groups of arguments that correspond to an implied DO loop in a FORTRAN write. The first item of a group indicates by its sign whether the variables are half words or full words. If the sign is minus the arguments are half words. If the sign is positive they are full words. The absolute value of the first item of each group is the number of variables or array names in the group. The second item in the group is the number of implied DO loop iterations M that should be used to transmit the array(s). The next |N| arguments are the arrays or variables. Only the array or variable items are transmitted. If M is greater than 1, a loop is set up in which the addresses (from which data is being moved) are incremented by a constant at the bottom of the loop. If N is negative, the constant is set to 2; and if N is positive, the constant is set to 4. The loop is executed M times. There may be as many groups in the call as are necessary provided that the total number of arguments in a call to subroutine WRT does not exceed the limits for the Fortran compiler being used for the Fortran calling subroutine.

ABEND: The subroutine prints the message ERROR followed by the integer identification code which is passed to it through the arguments.

LOGICAL DIVISION 21

General

This division prints the selected links output (i.e., the output from $ASSIGN SELECTED LINKS). This division is unique in that it must be a separate job (or at least 3 job steps) because it uses the IBM sort program twice.

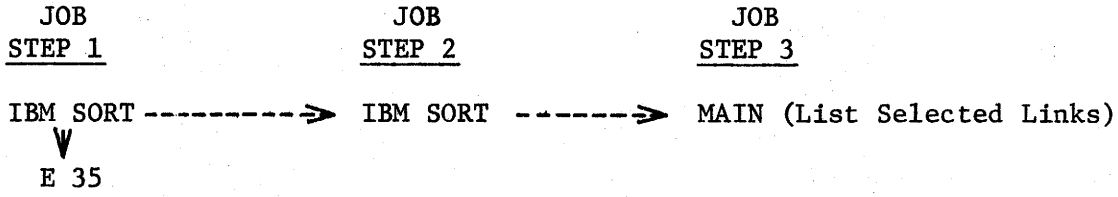
Input: Selected links data set SELTRP.

Scratch: First and second sorted data sets SORTOUT.

Output: Printed listing for each selected link of the zone pair trip interchanges assigned to the selected link.

Programs used: The IBM Sort/MERGE program, the exit program E 35, and a Fortran program to list the selected links and the trip interchanges loaded through them (i.e., MAIN).

Sequence of Program Execution:



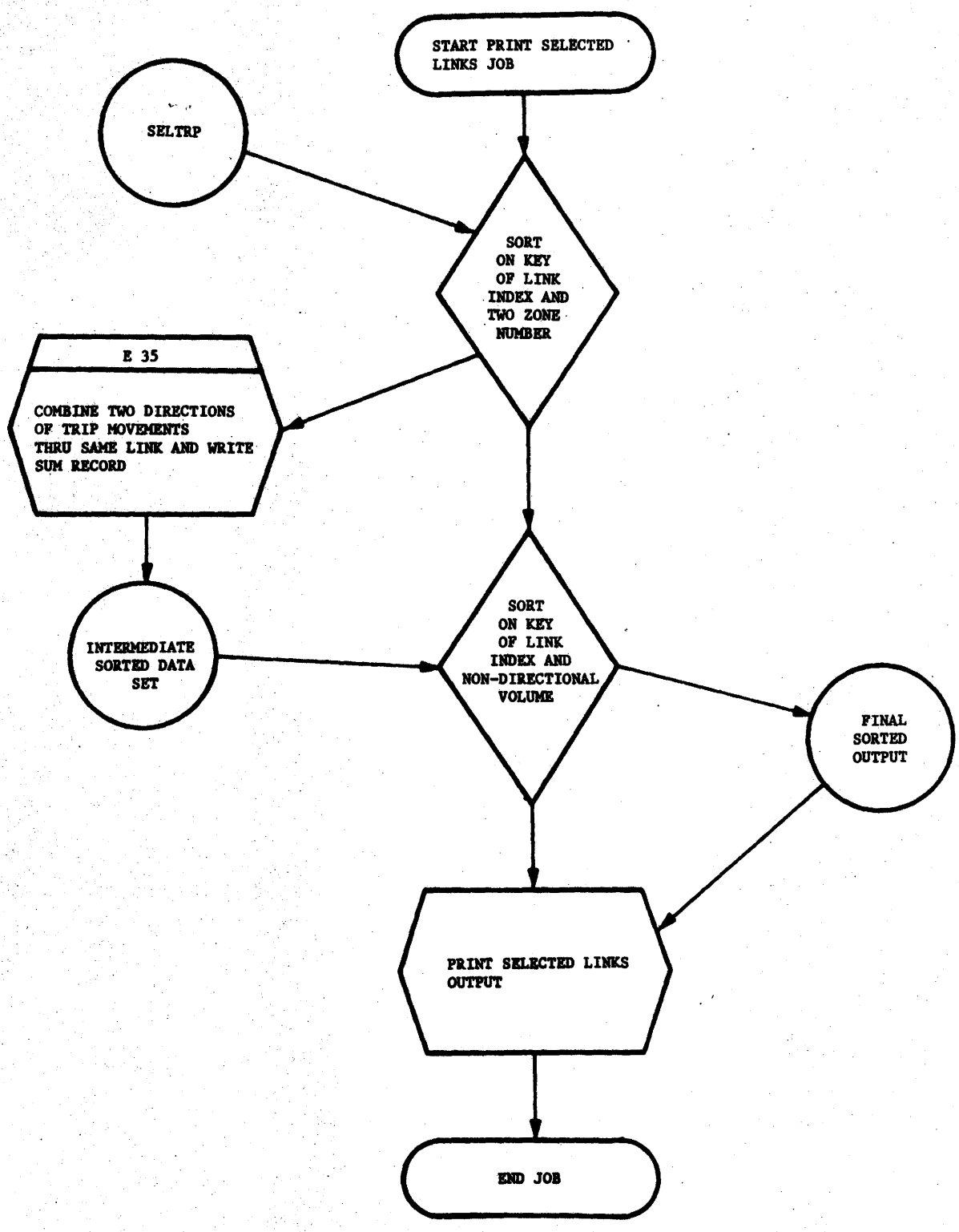
Summary of Individual Programs

IBM Sort/Merge Package: refer to the OS Sort/Merge Programmer's Guide, SC 33-4007-1.

E 35: This subroutine is called during JOB STEP 1 by the IBM Sort program. It combines the trip interchange records for each zone pair associated

with a given selected link thereby reducing the number of records to be sorted during JOB STEP 2. The combined trip interchange record, which is outputted for each zone pair interchanging trips through a selected link, contains both the directional and non-directional zone pair trips through the selected link. The total non-directional trip volume assigned to a selected link is also computed and outputted as a separate record. (During JOB STEP 2, the combined interchange records are sorted using the two sort keys: selected link number and non-directional trip volume.)

MAIN (List Selected Links): This is a Fortran program which reads the combined trip interchange records for the selected links (which were sorted during JOB STEP 2 using the keys: Selected link index number and non-directional zone pair volume) and prints the interchanges assigned to each selected link (in descending order of magnitude of the non-directional volumes) until either a limit parameter has been satisfied or until all interchanges have been printed.


START PRINT SELECTED LINKS JOB
SELTRP
SORT ON KEY OF LINK INDEX AND TWO ZONE NUMBER
E 35
COMBINE TWO DIRECTIONS OF TRIP MOVEMENTS THRU SAME LINK AND WRITE SUM RECORD
INTERMEDIATE SORTED DATA SET
SORT ON KEY OF LINK INDEX AND NON-DIRECTIONAL VOLUME
FINAL SORTED OUTPUT
PRINT SELECTED LINKS OUTPUT
END JOB

P R O G R A M C R O S S - R E F E R E N C E

A N D F L O W C H A R T S

CROSS-REFERENCE OF PROGRAMS

FLOWCHARTS

CROSS-REFERENCE
OF PROGRAMS

A complete cross-reference of calling programs versus programs called is provided in Table 4. This cross-reference serves both to identify all programs used by a given calling program and to, conversely, identify all calling programs which utilize a given program.

This cross-reference should prove especially useful when considering the modification of a program. For example, if modification is desired in OPENFT when used in conjunction with GTLD, a quick reference to Table 4 indicates that OPENFT is also called by OUTLLT, UPDTNT, and VREC. Therefore, any modifications in OPENFT should be compatible with all four calling programs.

TABLE 4: CROSS-REFERENCE OF CALLING PROGRAMS VERSUS PROGRAMS CALLED

CALLING PROGRAM

PROGRAMS CALLED	ALCP	ASMNET	BLDNET	CMPVH	CRD	CRDINT	FRATAR	GETRNS	GTLD	INITL1	LNKLST	LOAD	MAIN	MERG	MRGREC	NBRNET	OUTLLT	OUTLNT	OUTNET	OUTRIP	OUTSLN	OUTSNT	OUTTRE	OUTWLT	PATHCL	PATHSP	PRPFLD	PRPCTV	PRPNET	READVL	REGRES	REVNET	RTPFL	RTPLT	SC	SELECT	SKILD	SLOAD	SUMEND	SUMRY	TRKFLD	TREE	TRN	TRNMV	TURNN	UPDTNT	VREC	WGTLD	WTLNT
ABEND																																	X									X		X					
ALCP																																								X										
ALOG10																																	X	X															
AMAX1																																	X																
ASMNET													X																																					
BLDNET													X																																					
CLOSE																									X																									
CLOSFT									X								X																														X	X		
CMPVH																																								X										
COPYFT															X																																			
CRD													X																																					
CRDINT																																								X										
DSQRT	X																														X																		
FASPTH																									X																									
FMTLNE																			X																															
FRATAR													X																																					
FRXPI																	X																														X				
FRXPR																																		X																
GETDAT													X																																					
GETRN																																												X						
GETRNS																																													X					
GETVOL																	X																																		
GTLD																																								X										
GTLNK															X																																	X			
IBCOM	X		X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X		X		X	X			X		X	X	X	X	X
INITL1																									X																									
LANA			X						X	X					X	X	X						X		X											X							X	X	X	X			X
LANAD			X														X				X	X														X			X					X					
LANAH																								X																			X						
LANAL			X						X		X			X	X		X				X	X																X	X										X
LGLS			X						X					X	X		X											X																	X	X		
LGRS			X						X	X	X			X	X	X	X				X		X			X													X	X					X			X
LINE																																	X																
LNKLST													X																																					
LOAD																									X																									
LOAD2																									X																									
LORA			X						X	X				X	X		X								X											X			X						X	X		X
MAXO																																	X																
MERG													X																																					
MINO																																	X																
MOORE																										X																								
MRGREC																															X																		
NBIN																																				X													
NEWNET		X																										X		X																		
NUMBER																																	X																
OPEN																																			X													
OPENFT									X								X																														X	X		

TABLE 4: (continued)

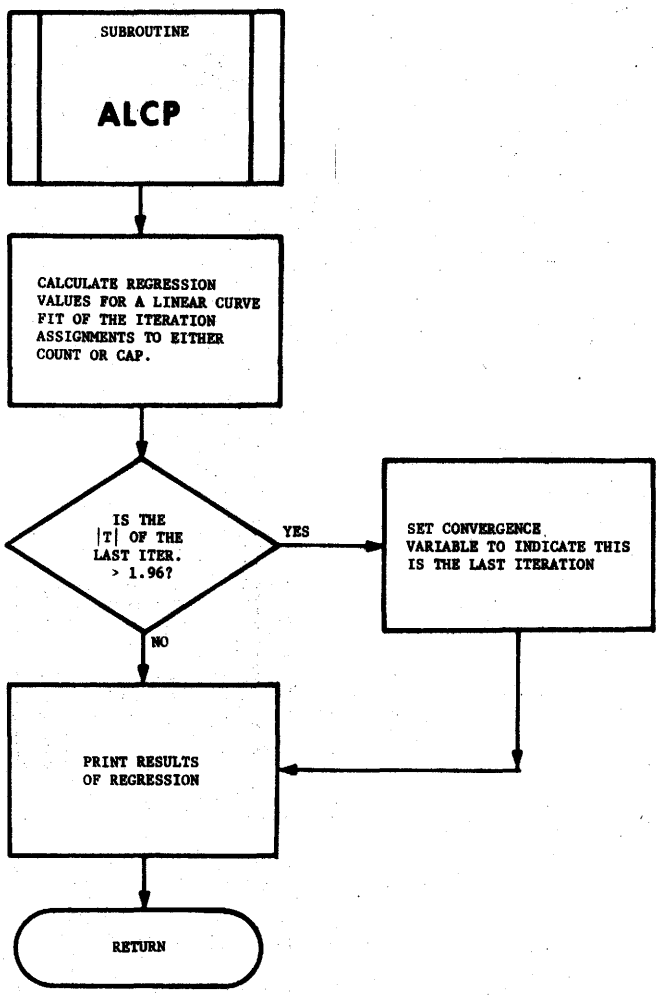
CALLING PROGRAM

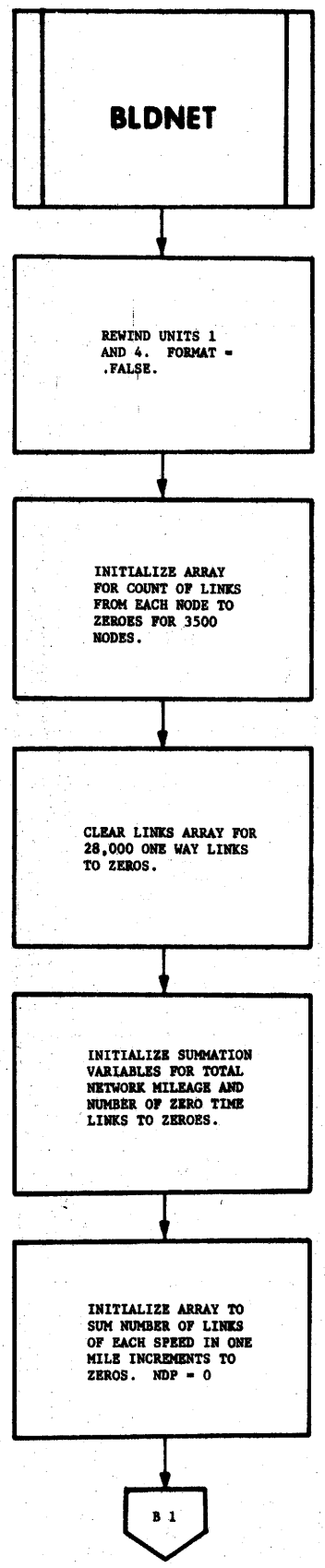
PROGRAMS CALLED	ALCP	ASMNET	BLDNET	CMPVH	CRD	CRDINT	FRATAR	GETRNS	GTLD	INITLL	LNKLST	LOAD	MAIN	MERG	MRGREC	NEWNET	OUTLLT	OUTLNT	OUTNET	OUTRIP	OUTSLN	OUTSNT	OUTTRE	OUTWLT	PATHCL	PATHSP	PRPBLD	PRPCTV	PRPNET	READVL	REGRES	REVNET	RTPFL	RTPLT	SC	SELECT	SELLD	SLOAD	SUMEND	SUMRY	TREBLD	TREE	TRN	TRNMV	TURNM	UPDTNT	VREC	WGTLD	WTLNT
OUTLLT																		X																														
OUTLNT																																			X					X								
OUTNET													X																																				
OUTRIP													X																																				
OUTSLN																										X																							
OUTSNT													X																																				
OUTTRE																									X																								
OUTWLT																																																	X
PARAM					X																																											
PATHCL																																					X				X	X						
PATHSP													X																																				
PLOT																																	X															
PLOTS																																	X															
PRPBLD																									X																								
PRPCTV													X																																				
PRPNET													X																																				
PTLNK																X																																		
READVL												X																																					
REGRES				X																																													
REVNET													X																																					
RTPFL																																								X										
RTPLT													X																																				
SC																																		X														
SELECT																									X																								
SELLD													X																																				
SLOAD																										X																							
SORT	X																																															
SUBFND																												X																					
SUMEND													X																																				
SUMRY													X																													X							
SYMBOL																																		X														
TIME													X			X									X	X																							
TREBLD													X																																				
TREE													X																																				
TRN																								X																								
TRNMV							X																																						X			
TURNM																	X																																
UPDTNT													X																																				
VREC		X																											X																			
VSORT						X										X																	X	X														
WGT																																																X
WGTA																																															X
WGTLD													X																																			
WRITE																																			X													
WRT									X								X																												X	X		
WTLNT													X																																			

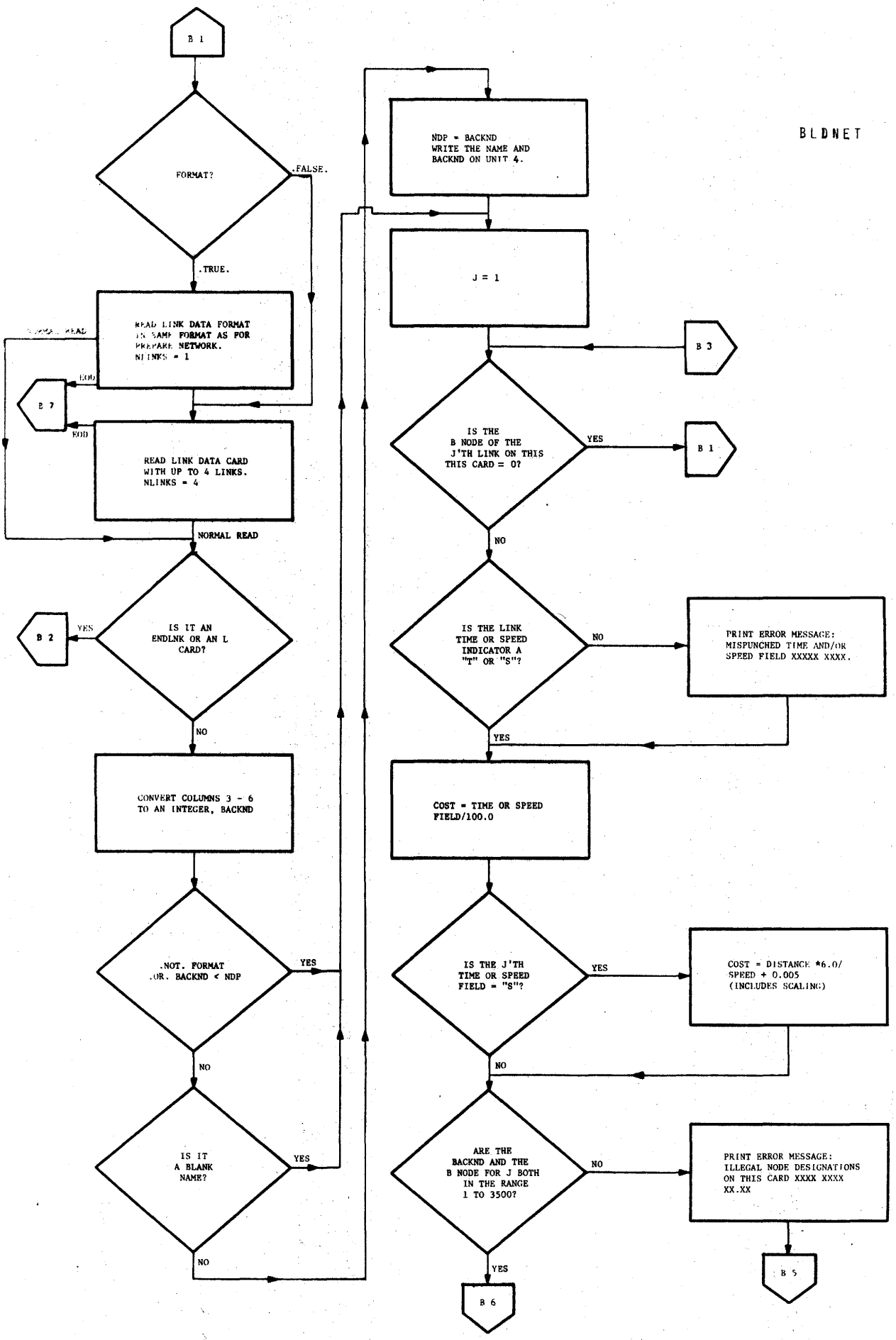
FLOWCHARTS

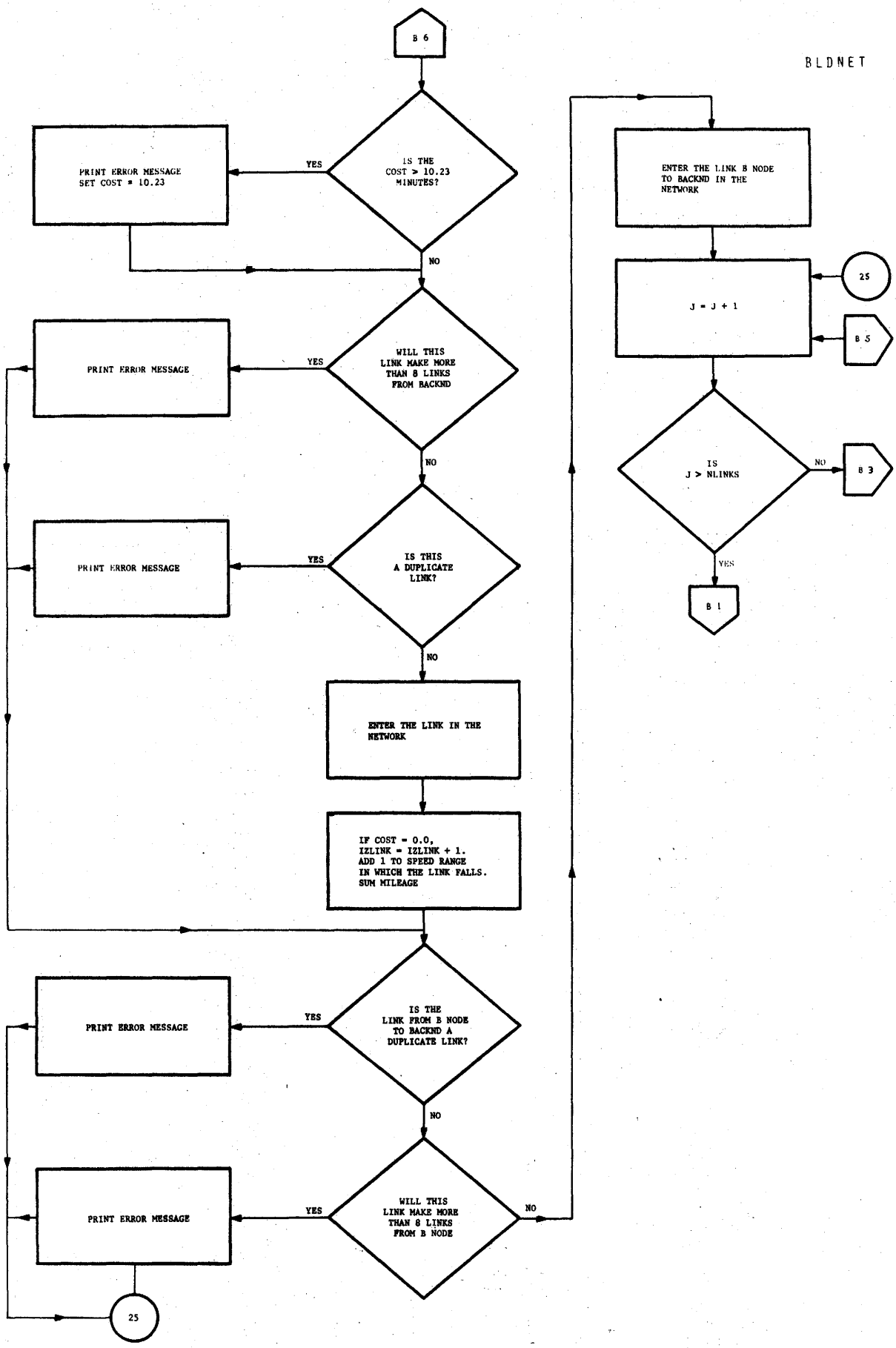
The following are the flowcharts associated with the significant subroutines in the Small Network Package. For convenience, these flowcharts are in alphabetical order.

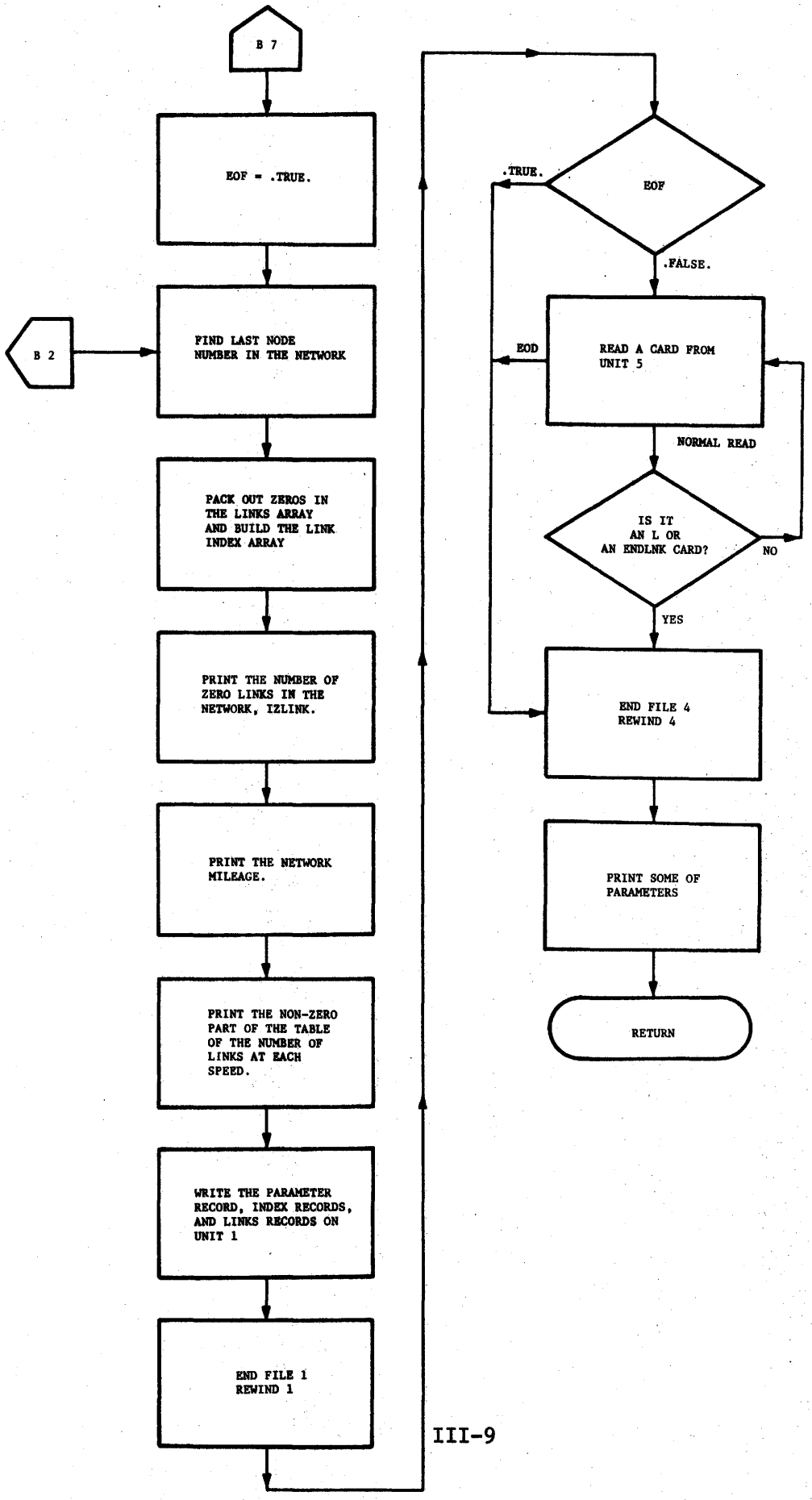
The objective of the flowcharts is to provide the programmer with an overview of the operation of each individual program. The level of detail contained in each flowchart is felt to be minimal for such an understanding. It should also be noted that these flowcharts are intended to be used in conjunction with information contained in sections V and VI (and, in some instances, section VII) when reviewing or studying a particular program listing.

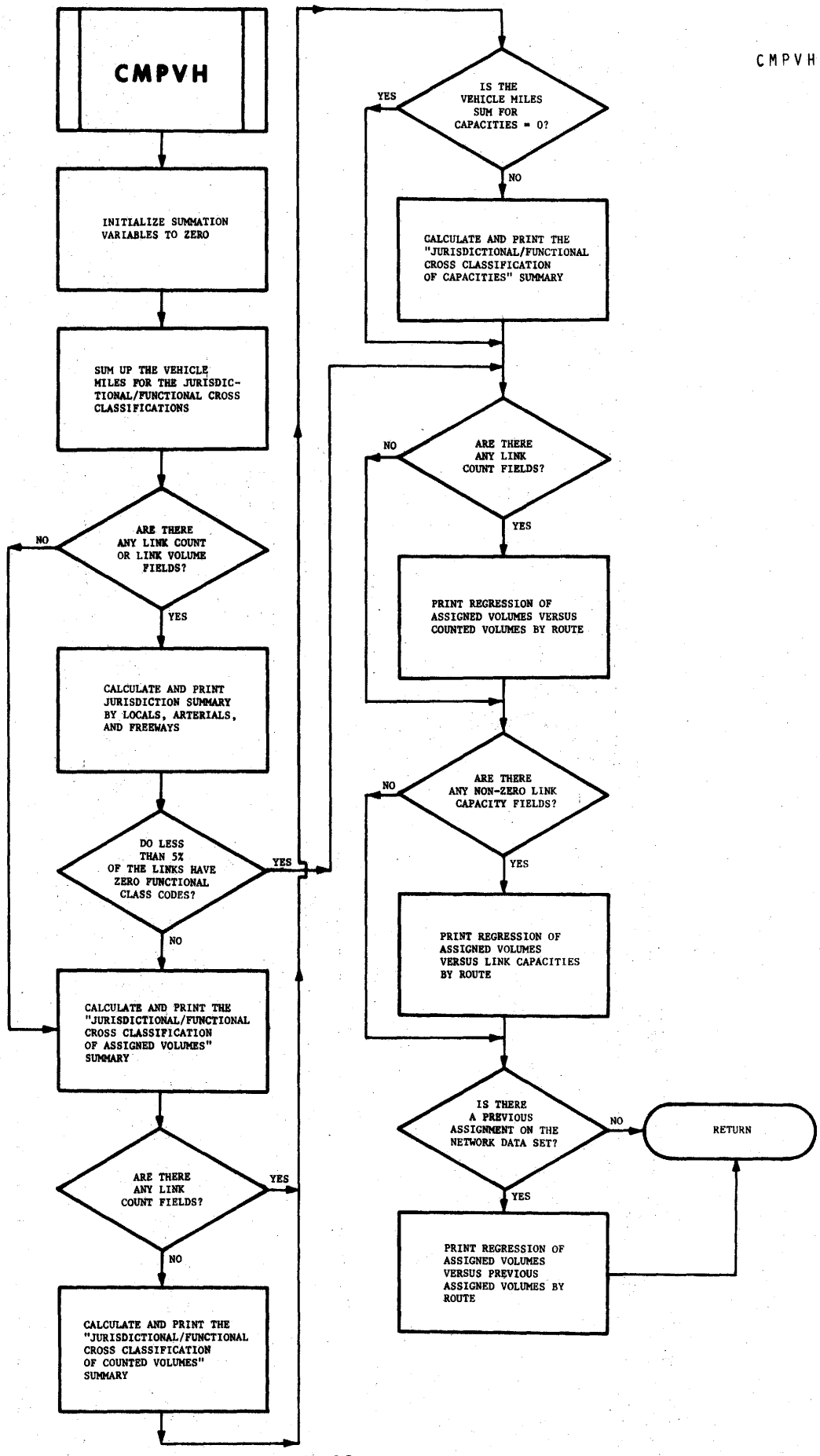

SUBROUTINE
ALCP
CALCULATE REGRESSION VALUES FOR A LINEAR CURVE FIT OF THE ITERATION ASSIGNMENTS TO EITHER COUNT OR CAP.
IS THE |T| OF THE LAST ITER. > 1.96?
YES
NO
SET CONVERGENCE VARIABLE TO INDICATE THIS IS THE LAST ITERATION
PRINT RESULTS OF REGRESSION
RETURN

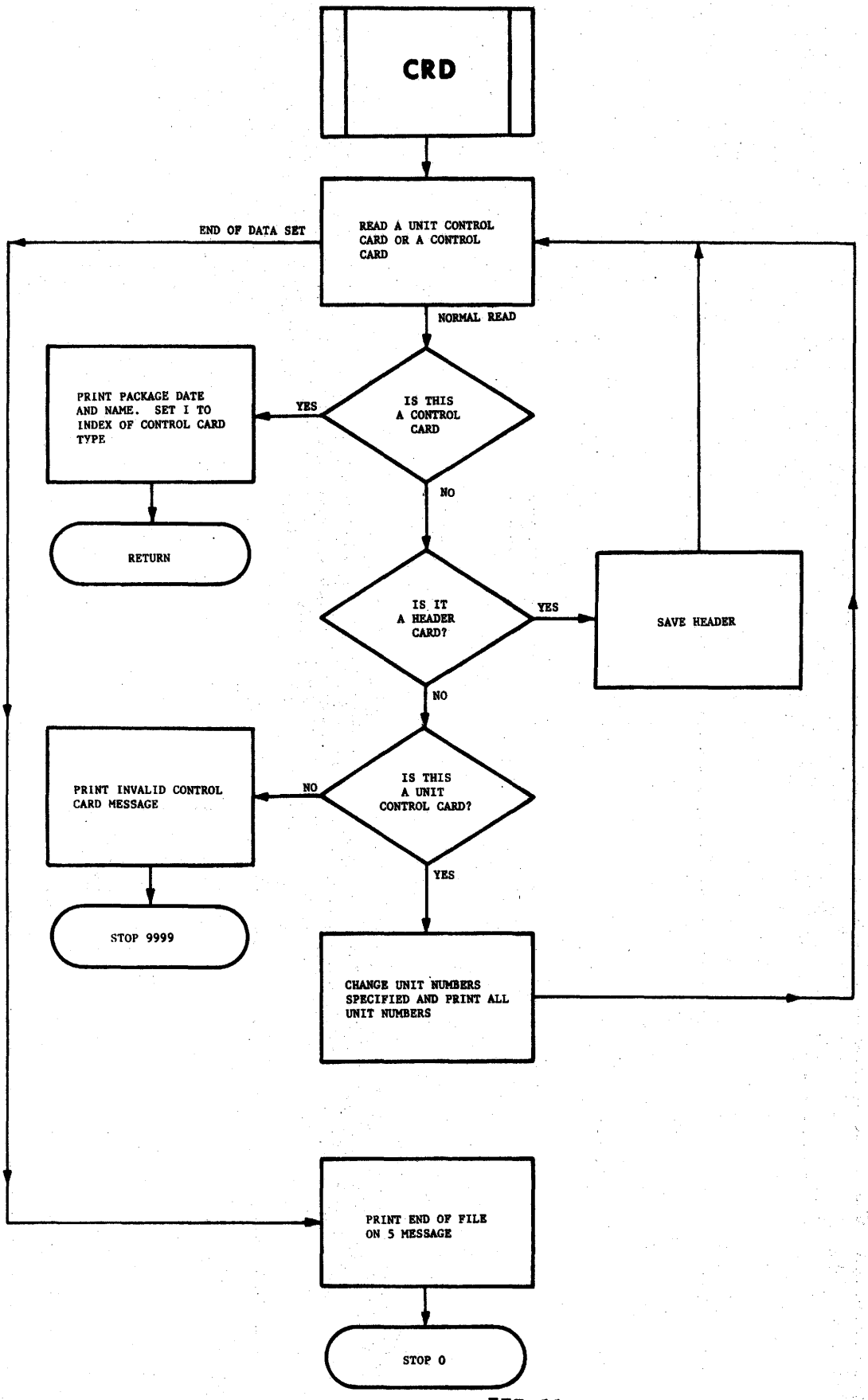

BLDNET
REWIND UNITS 1 AND 4. FORMAT = .FALSE.
INITIALIZE ARRAY FOR COUNT OF LINKS FROM EACH NODE TO ZEROES FOR 3500 NODES.
CLEAR LINKS ARRAY FOR 28,000 ONE WAY LINKS TO ZEROS.
INITIALIZE SUMMATION VARIABLES FOR TOTAL NETWORK MILEAGE AND NUMBER OF ZERO TIME LINKS TO ZEROES.
INITIALIZE ARRAY TO SUM NUMBER OF LINKS OF EACH SPEED IN ONE MILE INCREMENTS TO ZEROS. NDP = 0
B 1

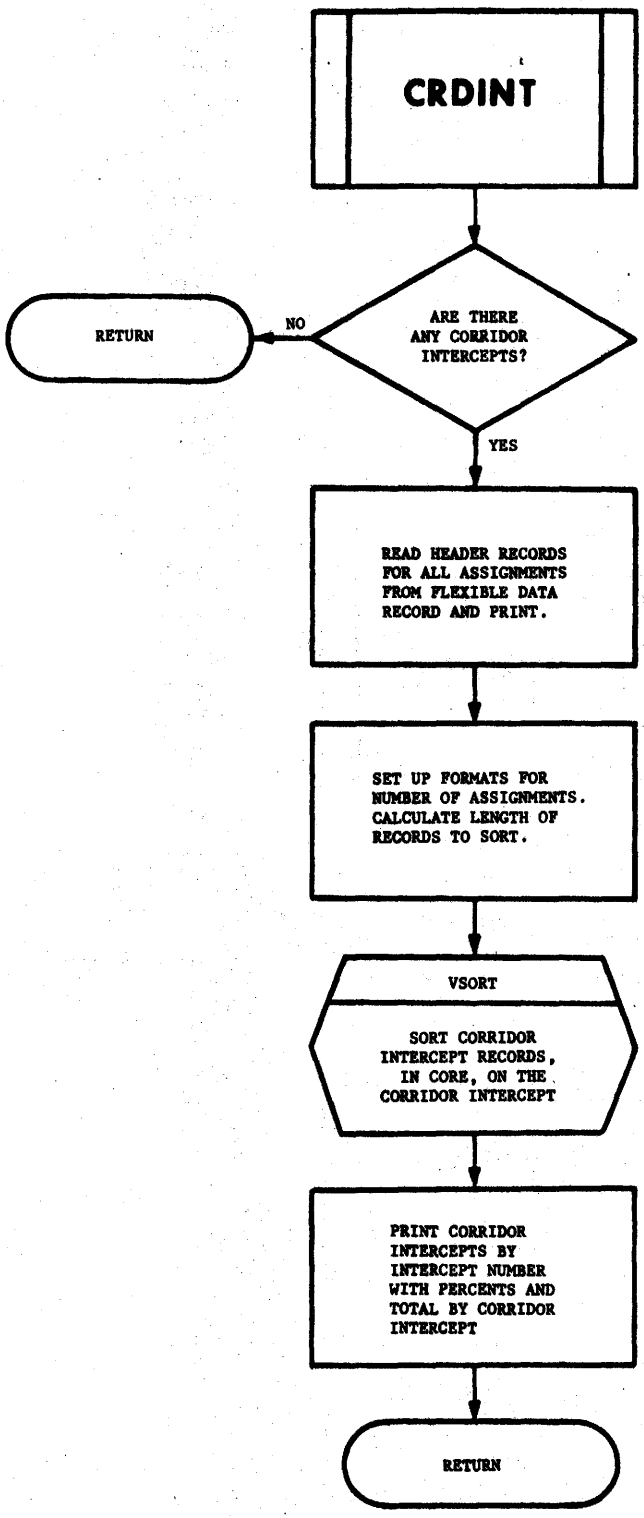

B 1
FORMAT?
.FALSE.
.TRUE.
READ LINK DATA FORMAT
IS SAME FORMAT AS FOR
PREPARE NETWORK.
NLINKS = 1
EOD
B 7
EOD
READ LINK DATA CARD
WITH UP TO 4 LINKS.
NLINKS = 4
NORMAL READ
B 2
YES
IS IT AN
ENDLNK OR AN L
CARD?
NO
CONVERT COLUMNS 3 - 6
TO AN INTEGER, BACKND
.NOT. FORMAT
.OR. BACKND < NDP
YES
NO
IS IT
A BLANK
NAME?
YES
NO
NDP = BACKND
WRITE THE NAME AND
BACKND ON UNIT 4.
J = 1
B 3
IS THE
B NODE OF THE
J'TH LINK ON THIS
THIS CARD = 0?
YES
B 1
NO
IS THE LINK
TIME OR SPEED
INDICATOR A
"T" OR "S"?
NO
PRINT ERROR MESSAGE:
MISPUNCHED TIME AND/OR
SPEED FIELD XXXXX XXXX.
YES
COST = TIME OR SPEED
FIELD/100.0
IS THE J'TH
TIME OR SPEED
FIELD = "S"?
YES
COST = DISTANCE *6.0/
SPEED + 0.005
(INCLUDES SCALING)
NO
ARE THE
BACKND AND THE
B NODE FOR J BOTH
IN THE RANGE
1 TO 3500?
NO
PRINT ERROR MESSAGE:
ILLEGAL NODE DESIGNATIONS
ON THIS CARD XXXX XXXX
XX.XX
B 5
YES
B 6

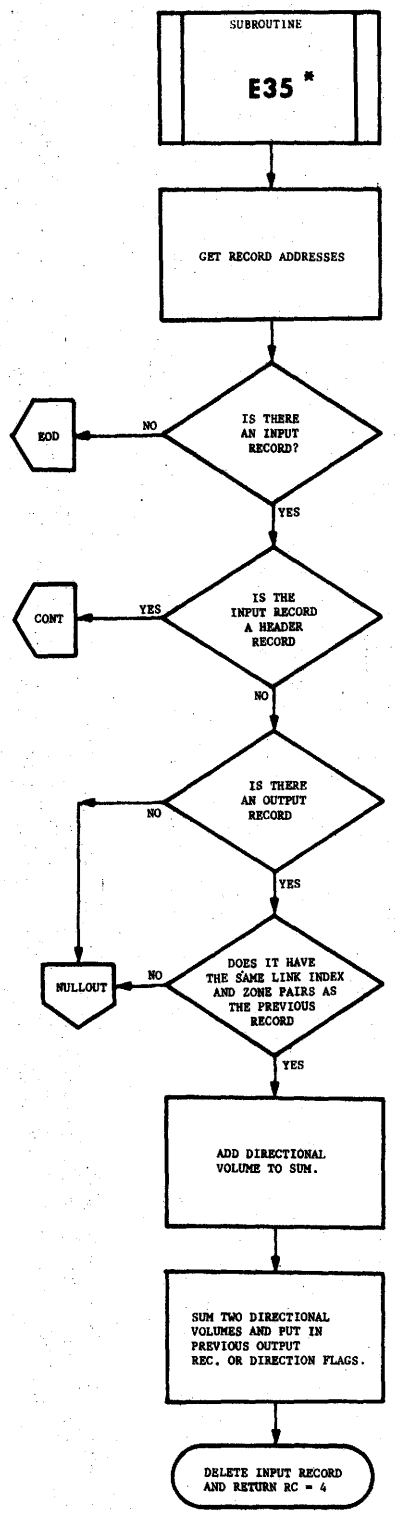

B 6
IS THE COST > 10.23 MINUTES?
YES
PRINT ERROR MESSAGE SET COST = 10.23
NO
WILL THIS LINK MAKE MORE THAN 8 LINKS FROM BACKND
YES
PRINT ERROR MESSAGE
NO
IS THIS A DUPLICATE LINK?
YES
PRINT ERROR MESSAGE
NO
ENTER THE LINK IN THE NETWORK
IF COST = 0.0, IZLINK = IZLINK + 1. ADD 1 TO SPEED RANGE IN WHICH THE LINK FALLS. SUM MILEAGE
IS THE LINK FROM B NODE TO BACKND A DUPLICATE LINK?
YES
PRINT ERROR MESSAGE
NO
WILL THIS LINK MAKE MORE THAN 8 LINKS FROM B NODE
YES
PRINT ERROR MESSAGE
NO
25
ENTER THE LINK B NODE TO BACKND IN THE NETWORK
25
J = J + 1
B 5
IS J > NLINKS
NO
B 3
YES
B 1


B 7
EOF = .TRUE.
B 2
FIND LAST NODE NUMBER IN THE NETWORK
PACK OUT ZEROS IN THE LINKS ARRAY AND BUILD THE LINK INDEX ARRAY
PRINT THE NUMBER OF ZERO LINKS IN THE NETWORK, IZLINK.
PRINT THE NETWORK MILEAGE.
PRINT THE NON-ZERO PART OF THE TABLE OF THE NUMBER OF LINKS AT EACH SPEED.
WRITE THE PARAMETER RECORD, INDEX RECORDS, AND LINKS RECORDS ON UNIT 1
END FILE 1
REWIND 1
EOF
.TRUE.
.FALSE.
READ A CARD FROM UNIT 5
EOD
NORMAL READ
IS IT AN L OR AN ENDLNK CARD?
NO
YES
END FILE 4
REWIND 4
PRINT SOME OF PARAMETERS
RETURN

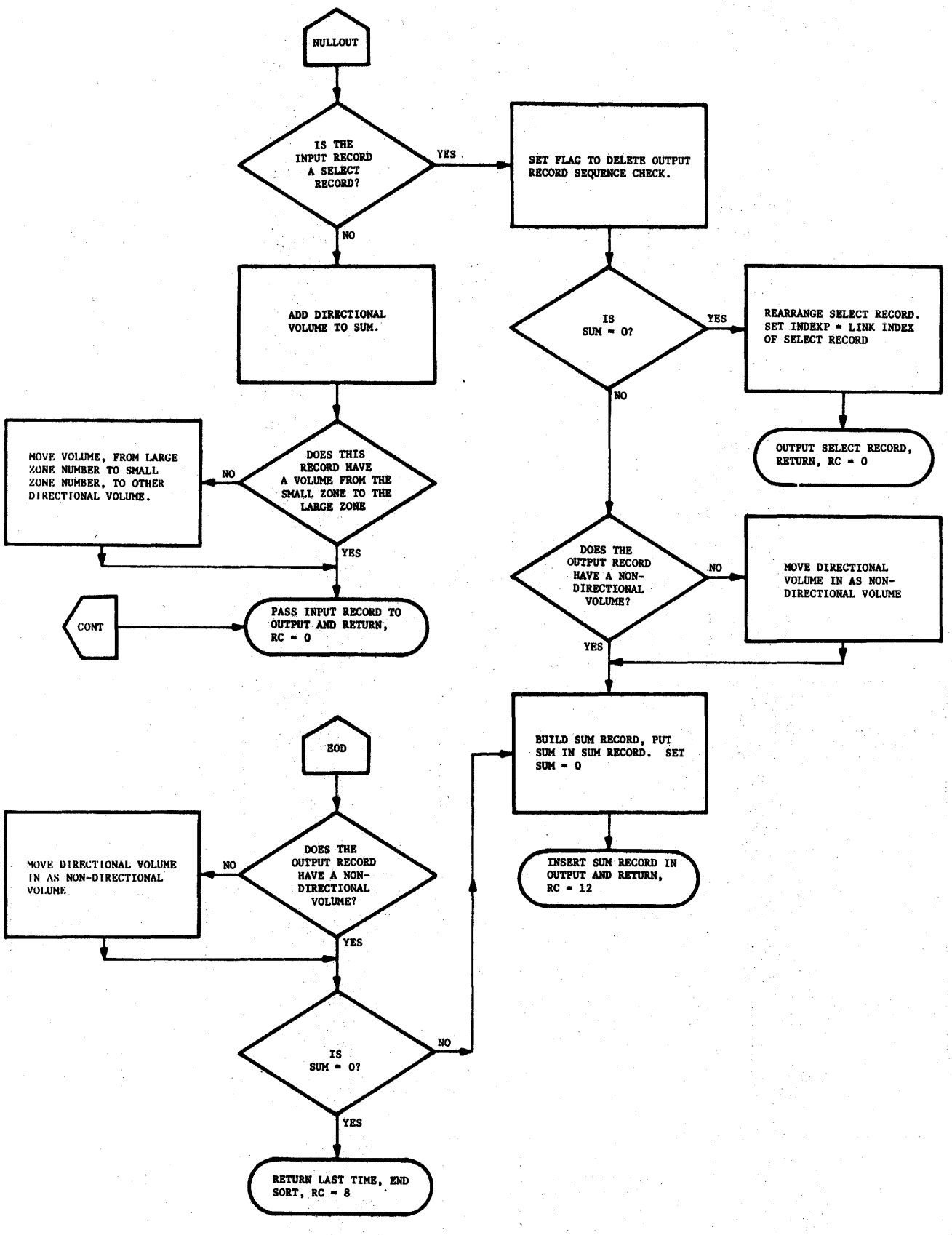

CMPVH
INITIALIZE SUMMATION VARIABLES TO ZERO
SUM UP THE VEHICLE MILES FOR THE JURISDICTIONAL/FUNCTIONAL CROSS CLASSIFICATIONS
ARE THERE ANY LINK COUNT OR LINK VOLUME FIELDS?
NO
YES
CALCULATE AND PRINT JURISDICTION SUMMARY BY LOCALS, ARTERIALS, AND FREEWAYS
DO LESS THAN 5% OF THE LINKS HAVE ZERO FUNCTIONAL CLASS CODES?
YES
NO
CALCULATE AND PRINT THE "JURISDICTIONAL/FUNCTIONAL CROSS CLASSIFICATION OF ASSIGNED VOLUMES" SUMMARY
ARE THERE ANY LINK COUNT FIELDS?
YES
NO
CALCULATE AND PRINT THE "JURISDICTIONAL/FUNCTIONAL CROSS CLASSIFICATION OF COUNTED VOLUMES" SUMMARY
IS THE VEHICLE MILES SUM FOR CAPACITIES = 0?
YES
NO
CALCULATE AND PRINT THE "JURISDICTIONAL/FUNCTIONAL CROSS CLASSIFICATION OF CAPACITIES" SUMMARY
ARE THERE ANY LINK COUNT FIELDS?
NO
YES
PRINT REGRESSION OF ASSIGNED VOLUMES VERSUS COUNTED VOLUMES BY ROUTE
ARE THERE ANY NON-ZERO LINK CAPACITY FIELDS?
NO
YES
PRINT REGRESSION OF ASSIGNED VOLUMES VERSUS LINK CAPACITIES BY ROUTE
IS THERE A PREVIOUS ASSIGNMENT ON THE NETWORK DATA SET?
NO
YES
RETURN
PRINT REGRESSION OF ASSIGNED VOLUMES VERSUS PREVIOUS ASSIGNED VOLUMES BY ROUTE

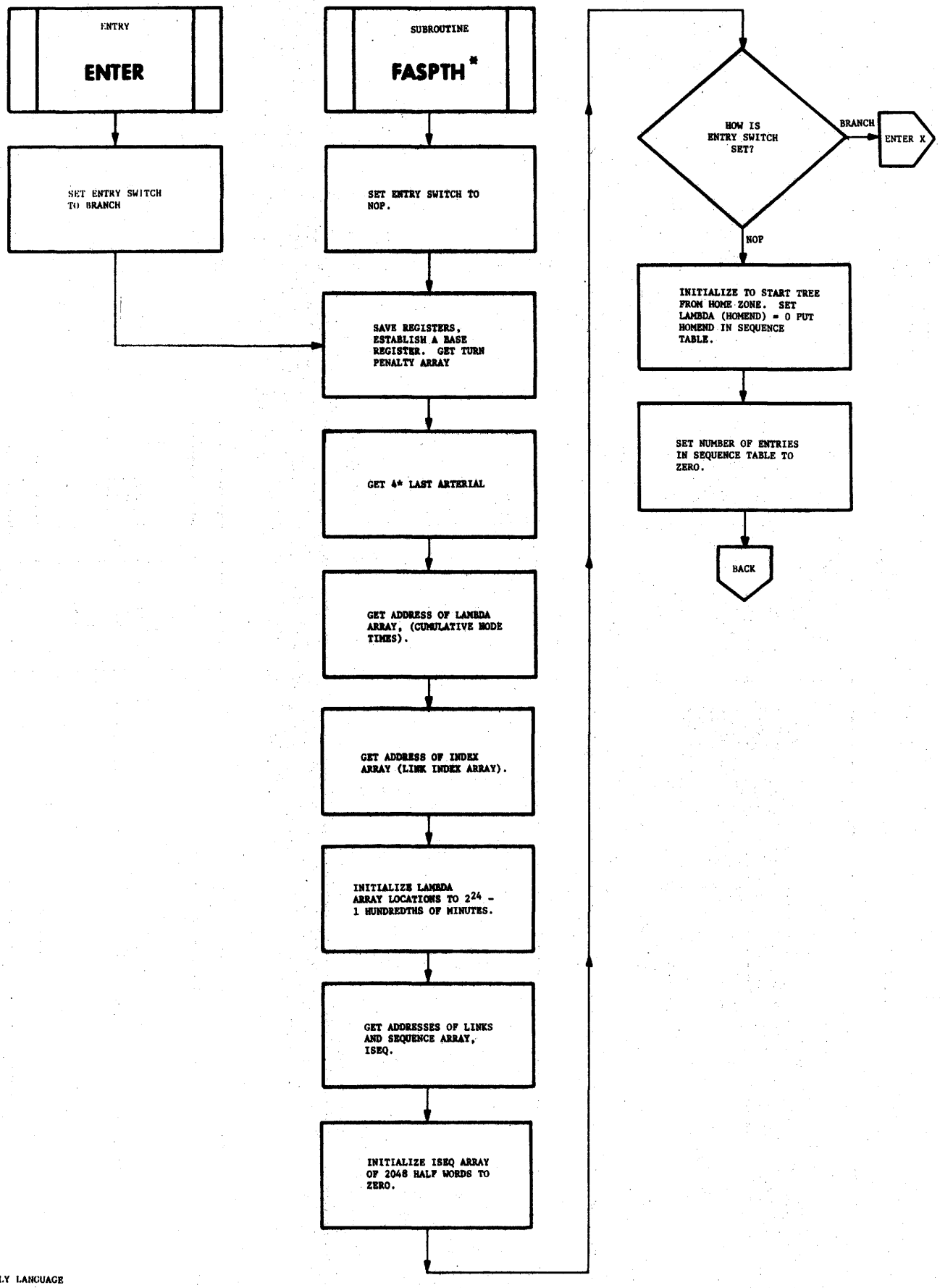

CRD
READ A UNIT CONTROL CARD OR A CONTROL CARD
END OF DATA SET
NORMAL READ
IS THIS A CONTROL CARD
YES
PRINT PACKAGE DATE AND NAME. SET I TO INDEX OF CONTROL CARD TYPE
RETURN
NO
IS IT A HEADER CARD?
YES
SAVE HEADER
NO
IS THIS A UNIT CONTROL CARD?
NO
PRINT INVALID CONTROL CARD MESSAGE
STOP 9999
YES
CHANGE UNIT NUMBERS SPECIFIED AND PRINT ALL UNIT NUMBERS
PRINT END OF FILE ON 5 MESSAGE
STOP 0


CRDINT
ARE THERE ANY CORRIDOR INTERCEPTS?
NO
RETURN
YES
READ HEADER RECORDS FOR ALL ASSIGNMENTS FROM FLEXIBLE DATA RECORD AND PRINT.
SET UP FORMATS FOR NUMBER OF ASSIGNMENTS. CALCULATE LENGTH OF RECORDS TO SORT.
VSORT
SORT CORRIDOR INTERCEPT RECORDS, IN CORE, ON THE CORRIDOR INTERCEPT
PRINT CORRIDOR INTERCEPTS BY INTERCEPT NUMBER WITH PERCENTS AND TOTAL BY CORRIDOR INTERCEPT
RETURN

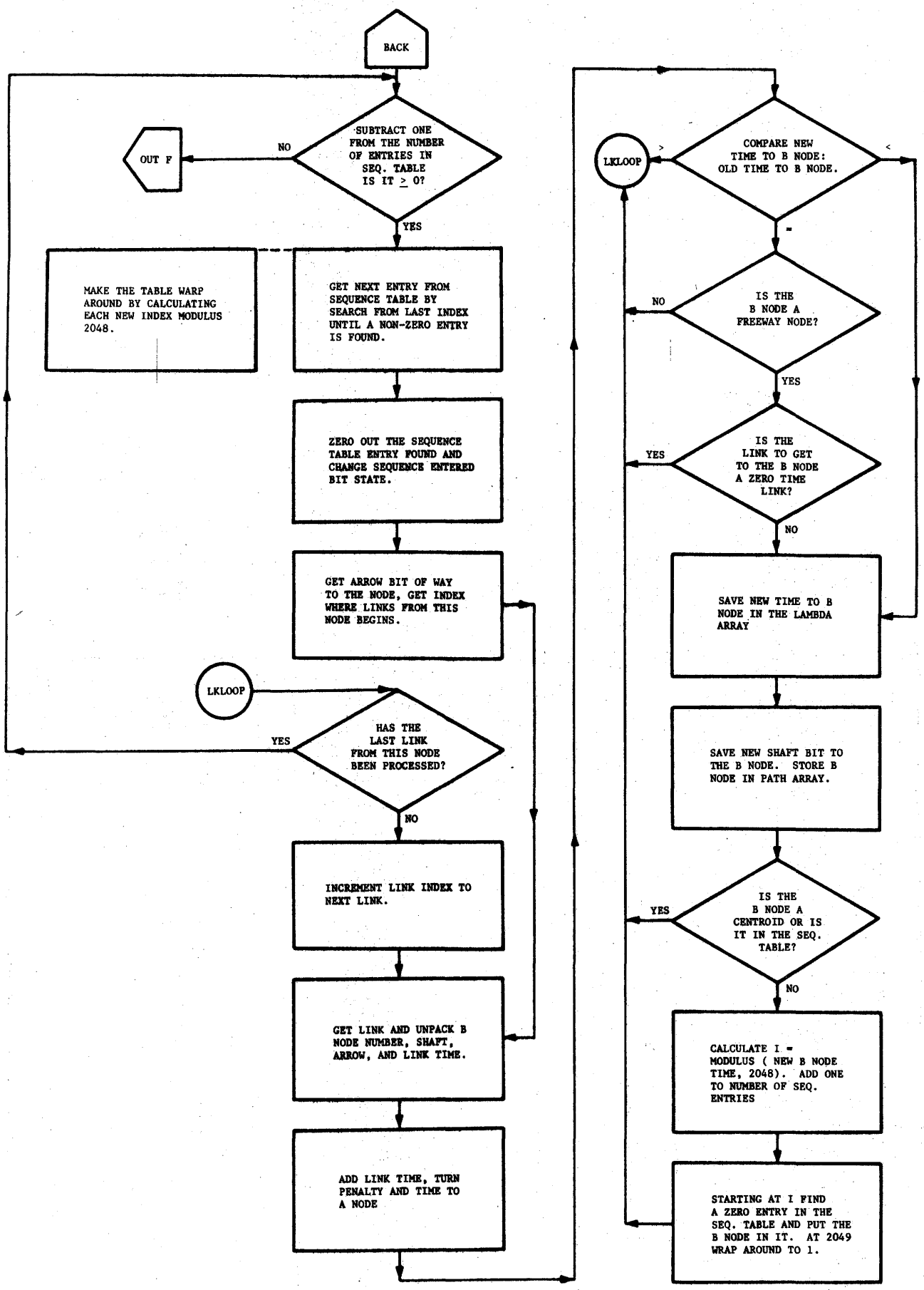


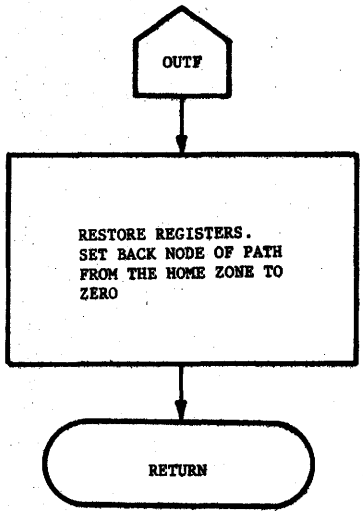
*ASSEMBLY LANGUAGE

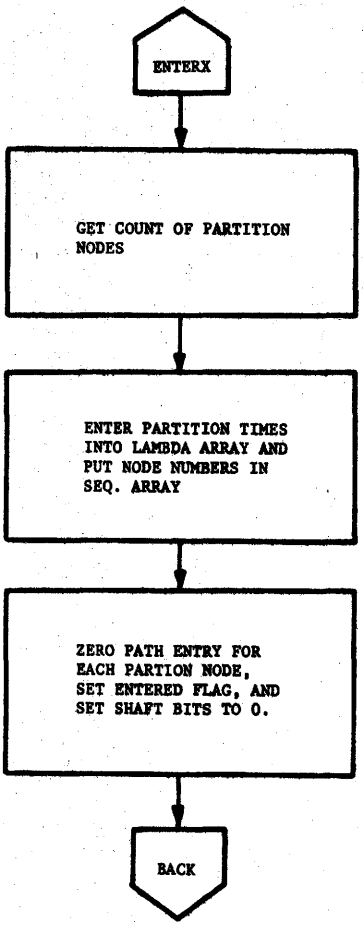

NULLOUT
IS THE INPUT RECORD A SELECT RECORD?
YES
NO
SET FLAG TO DELETE OUTPUT RECORD SEQUENCE CHECK.
ADD DIRECTIONAL VOLUME TO SUM.
IS SUM = 0?
YES
NO
REARRANGE SELECT RECORD. SET INDEXP = LINK INDEX OF SELECT RECORD
OUTPUT SELECT RECORD, RETURN, RC = 0
MOVE VOLUME, FROM LARGE ZONE NUMBER TO SMALL ZONE NUMBER, TO OTHER DIRECTIONAL VOLUME.
NO
DOES THIS RECORD HAVE A VOLUME FROM THE SMALL ZONE TO THE LARGE ZONE
YES
DOES THE OUTPUT RECORD HAVE A NON-DIRECTIONAL VOLUME?
NO
MOVE DIRECTIONAL VOLUME IN AS NON-DIRECTIONAL VOLUME
YES
CONT
PASS INPUT RECORD TO OUTPUT AND RETURN, RC = 0
BUILD SUM RECORD, PUT SUM IN SUM RECORD. SET SUM = 0
EOD
DOES THE OUTPUT RECORD HAVE A NON-DIRECTIONAL VOLUME?
NO
MOVE DIRECTIONAL VOLUME IN AS NON-DIRECTIONAL VOLUME
YES
INSERT SUM RECORD IN OUTPUT AND RETURN, RC = 12
IS SUM = 0?
NO
YES
RETURN LAST TIME, END SORT, RC = 8

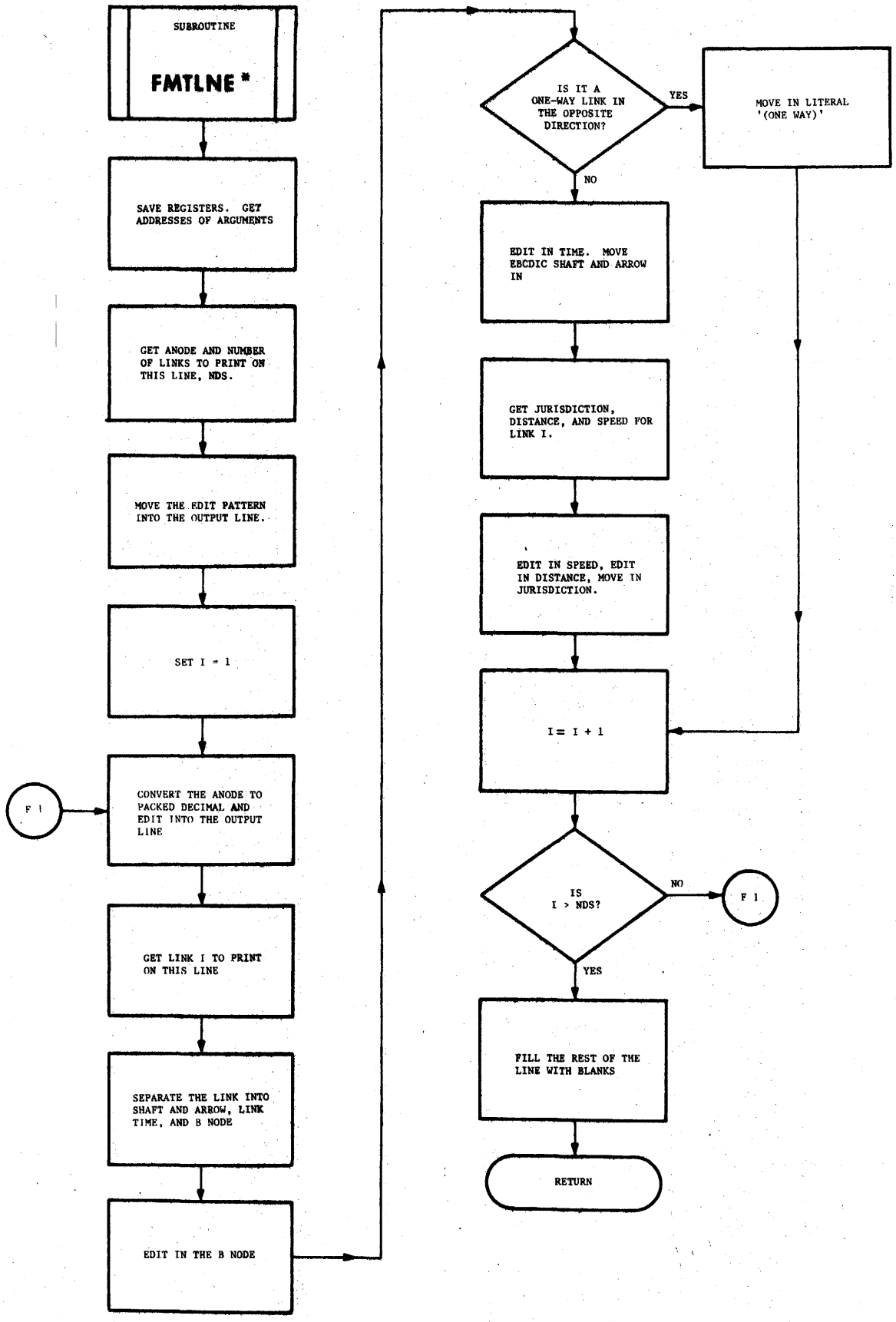


*ASSEMBLY LANGUAGE

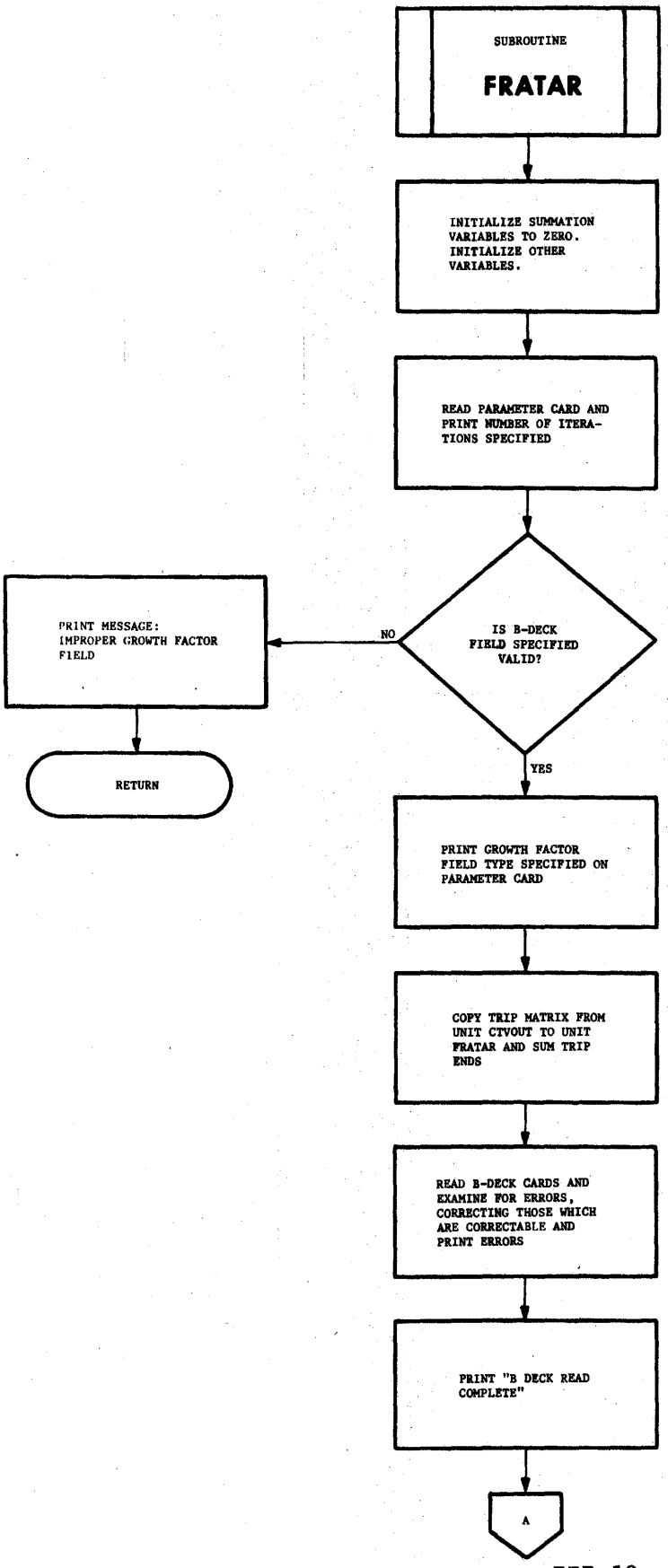

BACK
SUBTRACT ONE FROM THE NUMBER OF ENTRIES IN SEQ. TABLE IS IT ≥ 0?
NO
OUT F
YES
MAKE THE TABLE WARP AROUND BY CALCULATING EACH NEW INDEX MODULUS 2048.
GET NEXT ENTRY FROM SEQUENCE TABLE BY SEARCH FROM LAST INDEX UNTIL A NON-ZERO ENTRY IS FOUND.
ZERO OUT THE SEQUENCE TABLE ENTRY FOUND AND CHANGE SEQUENCE ENTERED BIT STATE.
GET ARROW BIT OF WAY TO THE NODE, GET INDEX WHERE LINKS FROM THIS NODE BEGINS.
LKLOOP
HAS THE LAST LINK FROM THIS NODE BEEN PROCESSED?
YES
NO
INCREMENT LINK INDEX TO NEXT LINK.
GET LINK AND UNPACK B NODE NUMBER, SHAFT, ARROW, AND LINK TIME.
ADD LINK TIME, TURN PENALTY AND TIME TO A NODE
LKLOOP
COMPARE NEW TIME TO B NODE: OLD TIME TO B NODE.
>
<
=
IS THE B NODE A FREEWAY NODE?
NO
YES
IS THE LINK TO GET TO THE B NODE A ZERO TIME LINK?
YES
NO
SAVE NEW TIME TO B NODE IN THE LAMBDA ARRAY
SAVE NEW SHAFT BIT TO THE B NODE. STORE B NODE IN PATH ARRAY.
IS THE B NODE A CENTROID OR IS IT IN THE SEQ. TABLE?
YES
NO
CALCULATE I = MODULUS (NEW B NODE TIME, 2048). ADD ONE TO NUMBER OF SEQ. ENTRIES
STARTING AT I FIND A ZERO ENTRY IN THE SEQ. TABLE AND PUT THE B NODE IN IT. AT 2049 WRAP AROUND TO 1.

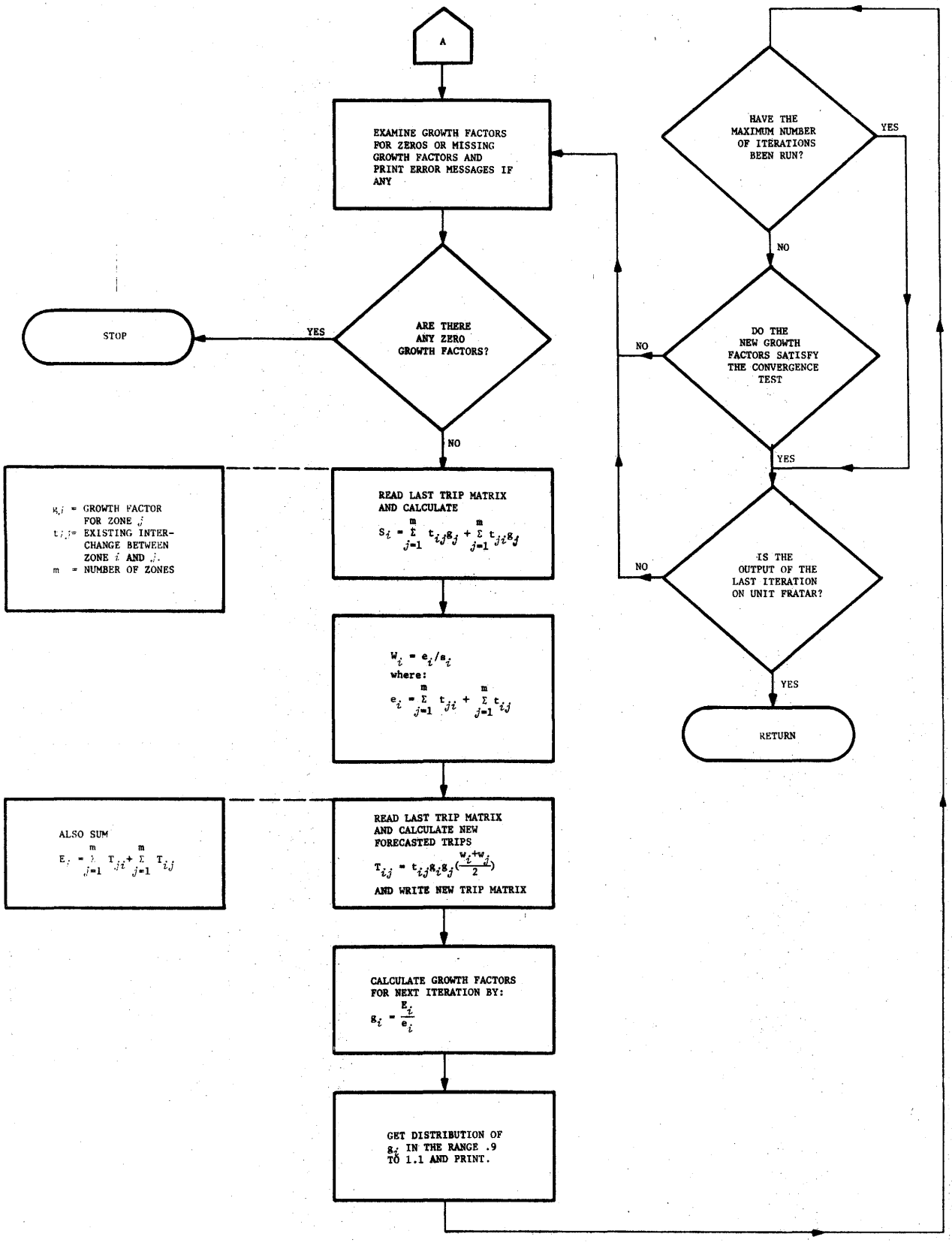

OUTF
RESTORE REGISTERS.
SET BACK NODE OF PATH
FROM THE HOME ZONE TO
ZERO
RETURN

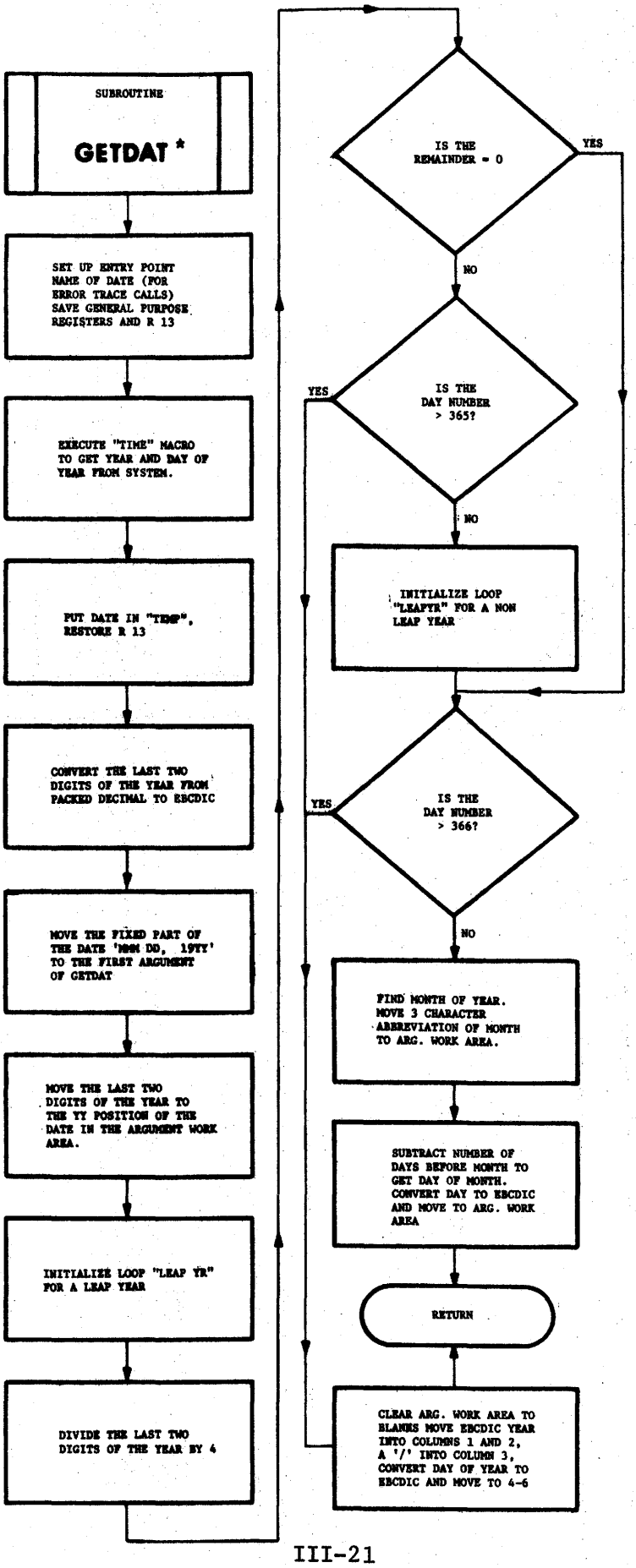

ENTERX
GET COUNT OF PARTITION
NODES
ENTER PARTITION TIMES
INTO LAMBDA ARRAY AND
PUT NODE NUMBERS IN
SEQ. ARRAY
ZERO PATH ENTRY FOR
EACH PARTION NODE,
SET ENTERED FLAG, AND
SET SHAFT BITS TO 0.
BACK



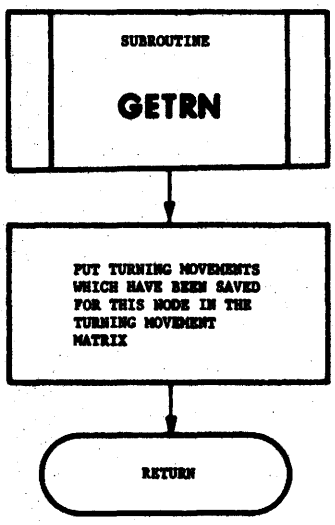
* ASSEMBLY LANGUAGE

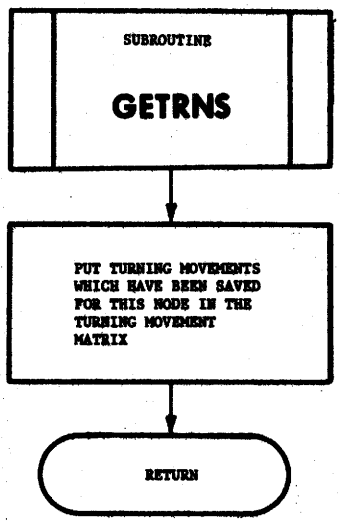

SUBROUTINE
FRATAR
INITIALIZE SUMMATION VARIABLES TO ZERO. INITIALIZE OTHER VARIABLES.
READ PARAMETER CARD AND PRINT NUMBER OF ITERATIONS SPECIFIED
IS B-DECK FIELD SPECIFIED VALID?
NO
PRINT MESSAGE: IMPROPER GROWTH FACTOR FIELD
RETURN
YES
PRINT GROWTH FACTOR FIELD TYPE SPECIFIED ON PARAMETER CARD
COPY TRIP MATRIX FROM UNIT CTVOUT TO UNIT FRATAR AND SUM TRIP ENDS
READ B-DECK CARDS AND EXAMINE FOR ERRORS, CORRECTING THOSE WHICH ARE CORRECTABLE AND PRINT ERRORS
PRINT "B DECK READ COMPLETE"
A

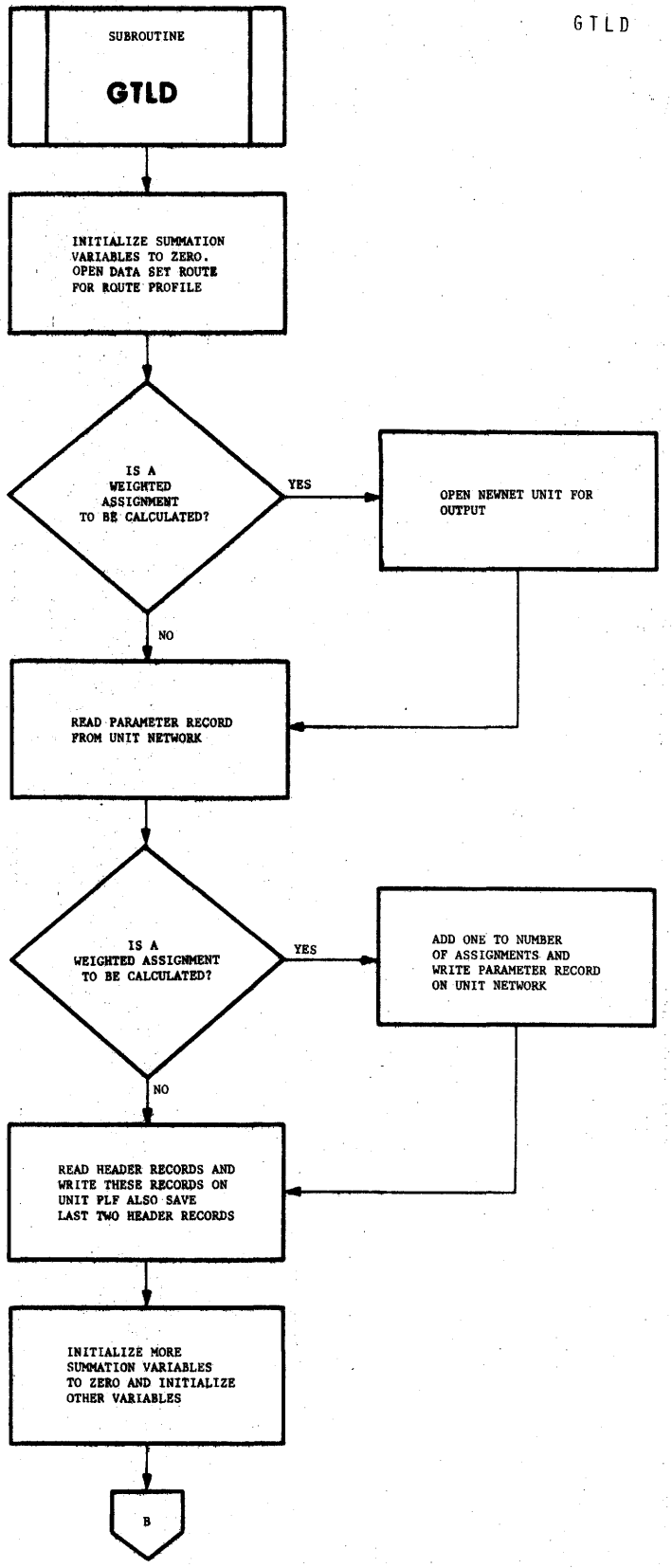

A
EXAMINE GROWTH FACTORS FOR ZEROS OR MISSING GROWTH FACTORS AND PRINT ERROR MESSAGES IF ANY
ARE THERE ANY ZERO GROWTH FACTORS?
YES
STOP
NO
g_i = GROWTH FACTOR FOR ZONE i
t_{ij} = EXISTING INTERCHANGE BETWEEN ZONE i AND j.
m = NUMBER OF ZONES
READ LAST TRIP MATRIX AND CALCULATE
$S_i = \sum_{j=1}^{m} t_{ij}g_j + \sum_{j=1}^{m} t_{ji}g_j$
$W_i = e_i/s_i$
where:
$e_i = \sum_{j=1}^{m} t_{ji} + \sum_{j=1}^{m} t_{ij}$
ALSO SUM
$E_i = \sum_{j=1}^{m} T_{ji} + \sum_{j=1}^{m} T_{ij}$
READ LAST TRIP MATRIX AND CALCULATE NEW FORECASTED TRIPS
$T_{ij} = t_{ij}g_ig_j(\frac{w_i+w_j}{2})$
AND WRITE NEW TRIP MATRIX
CALCULATE GROWTH FACTORS FOR NEXT ITERATION BY:
$g_i = \frac{E_i}{e_i}$
GET DISTRIBUTION OF g_i IN THE RANGE .9 TO 1.1 AND PRINT.
HAVE THE MAXIMUM NUMBER OF ITERATIONS BEEN RUN?
YES
NO
DO THE NEW GROWTH FACTORS SATISFY THE CONVERGENCE TEST
NO
YES
IS THE OUTPUT OF THE LAST ITERATION ON UNIT FRATAR?
NO
YES
RETURN

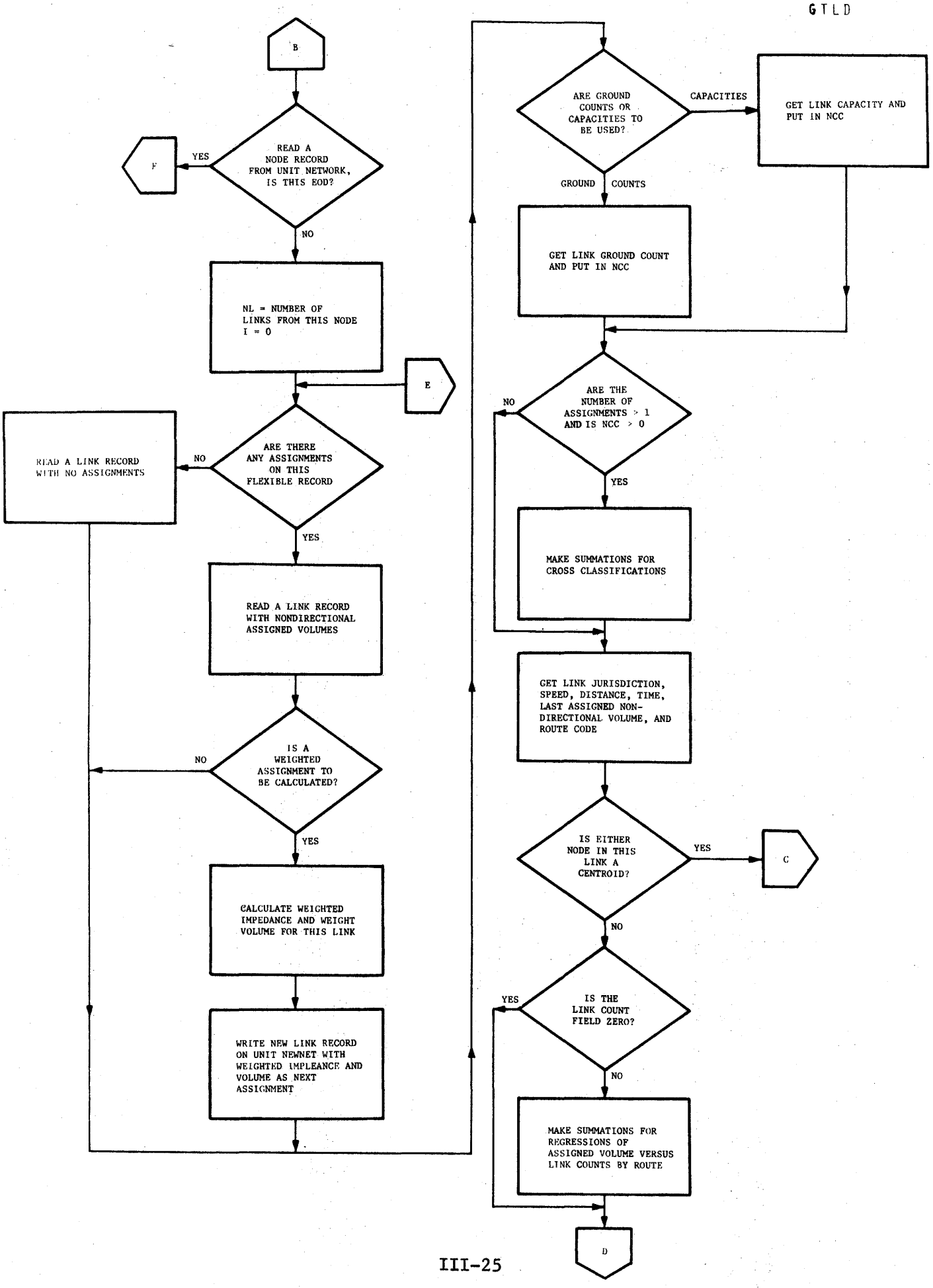


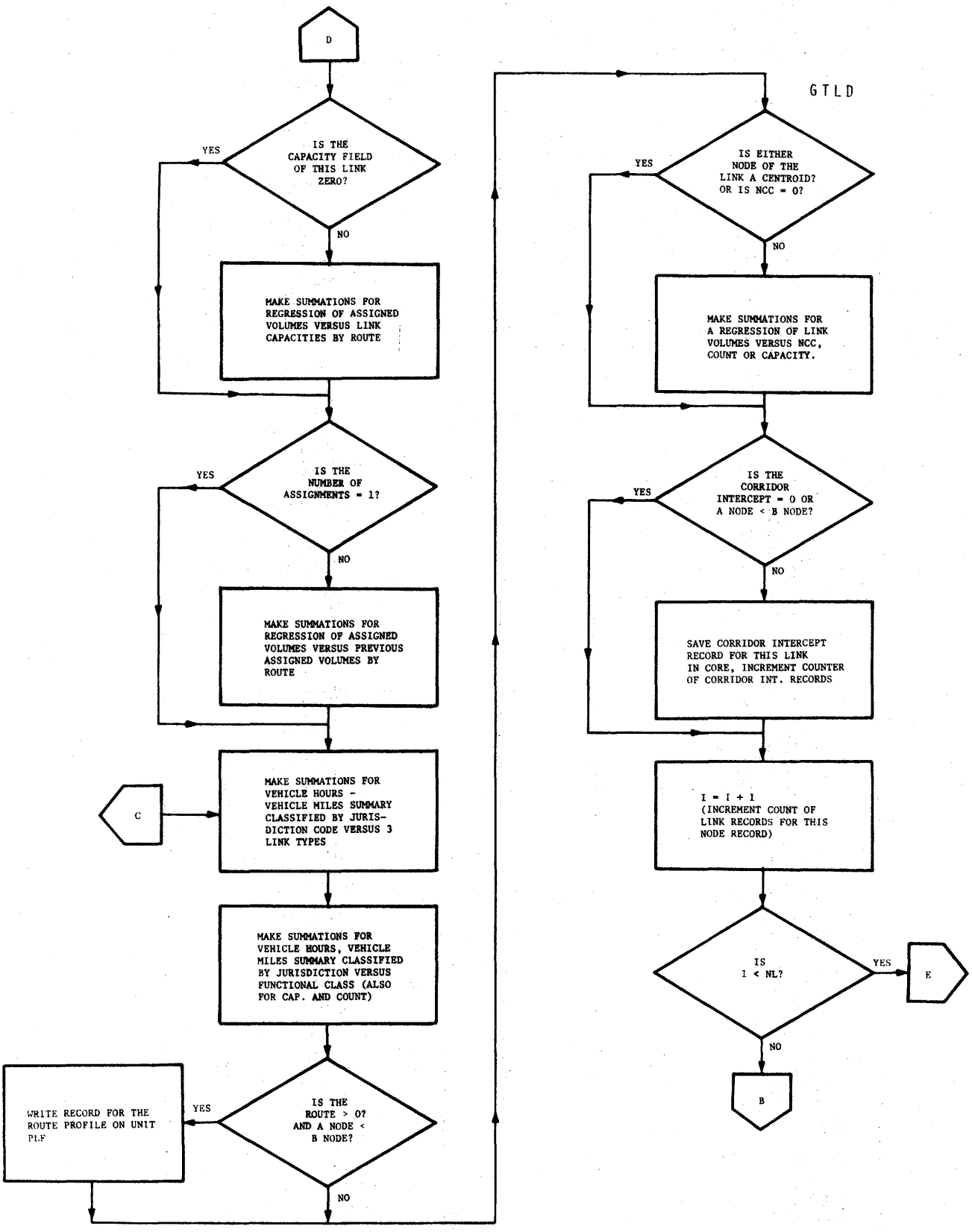
*ASSEMBLY LANGUAGE

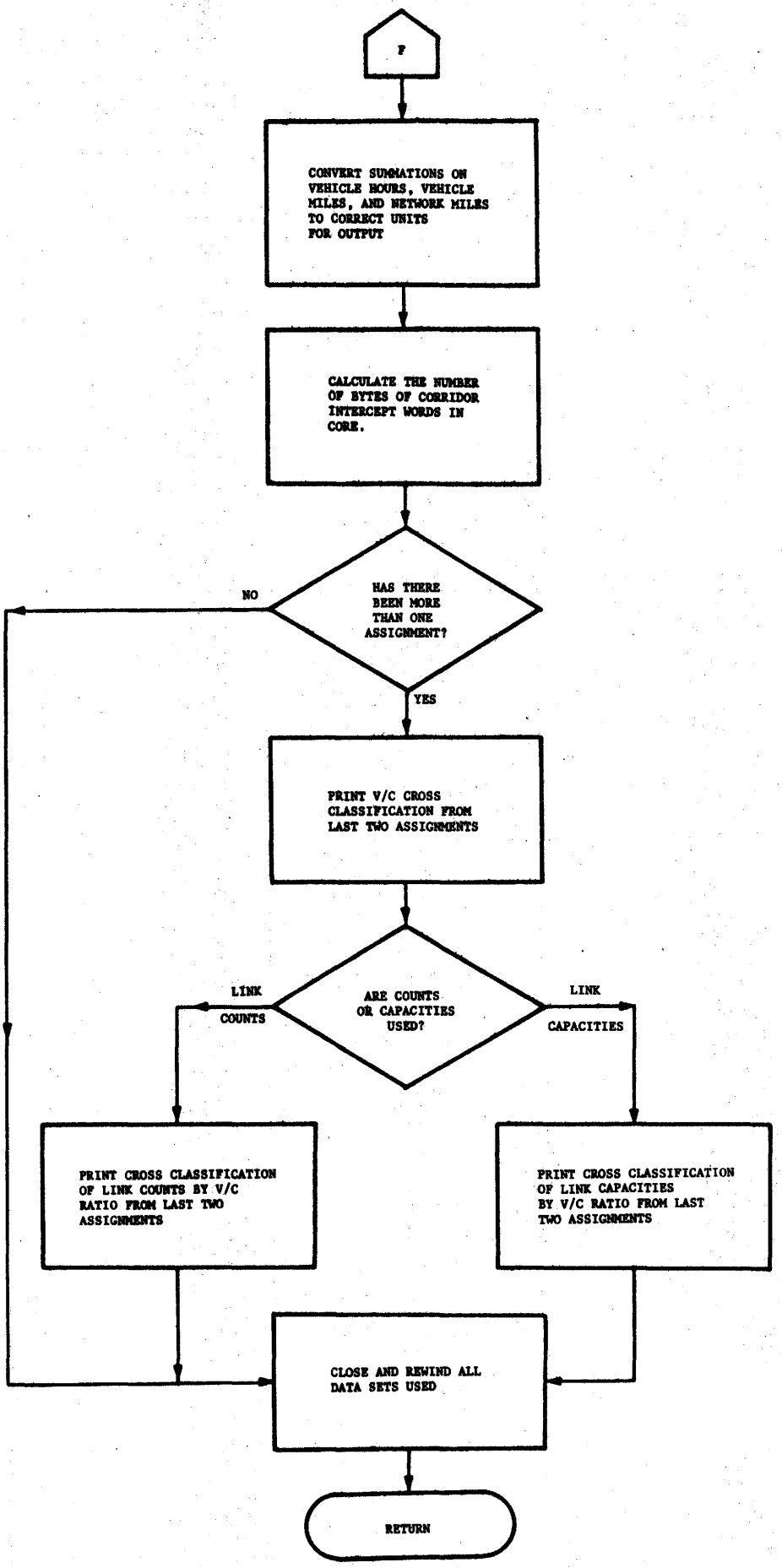

SUBROUTINE
GETRN
PUT TURNING MOVEMENTS WHICH HAVE BEEN SAVED FOR THIS NODE IN THE TURNING MOVEMENT MATRIX
RETURN

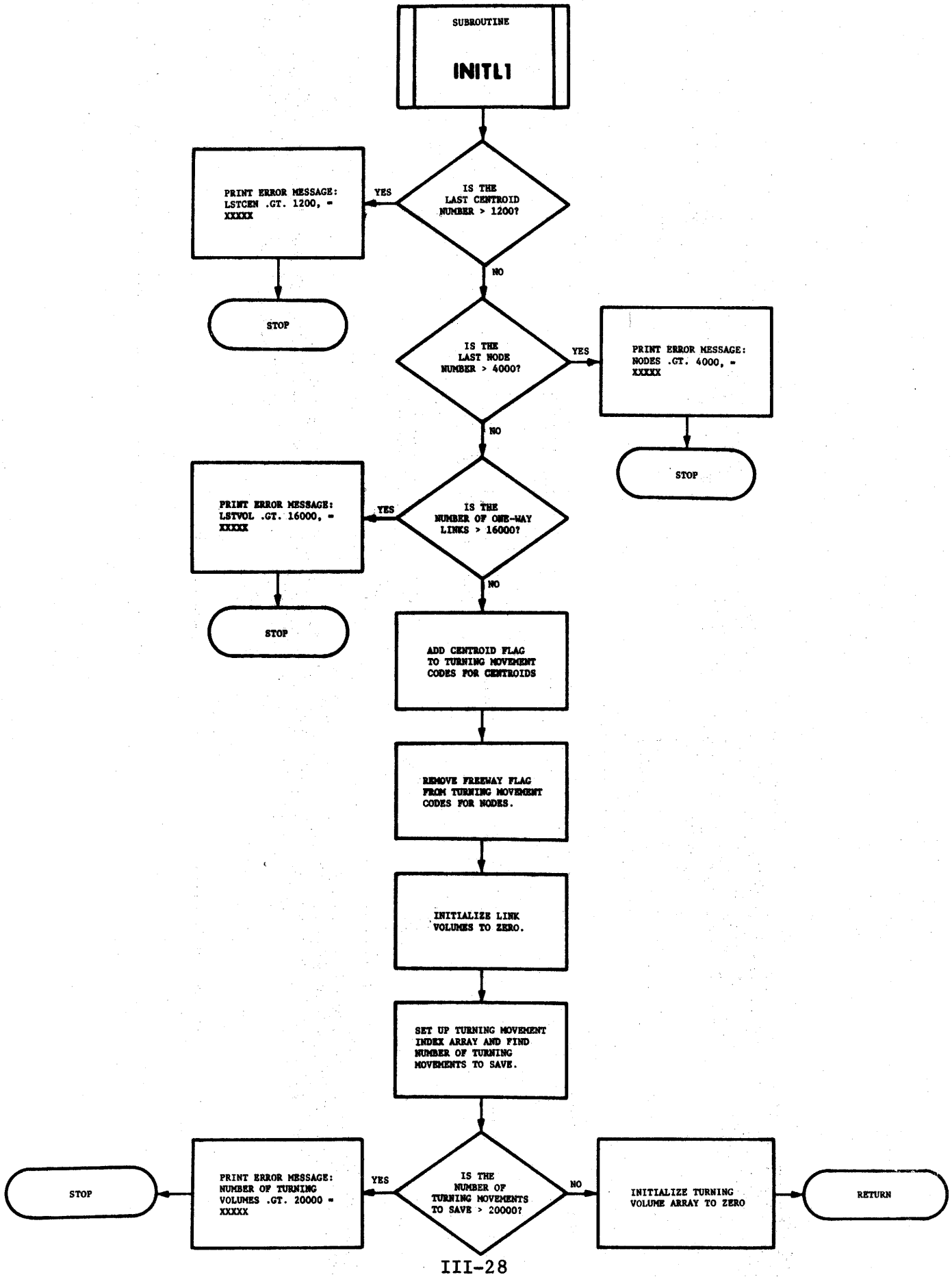

SUBROUTINE
GETRNS
PUT TURNING MOVEMENTS WHICH HAVE BEEN SAVED FOR THIS NODE IN THE TURNING MOVEMENT MATRIX
RETURN

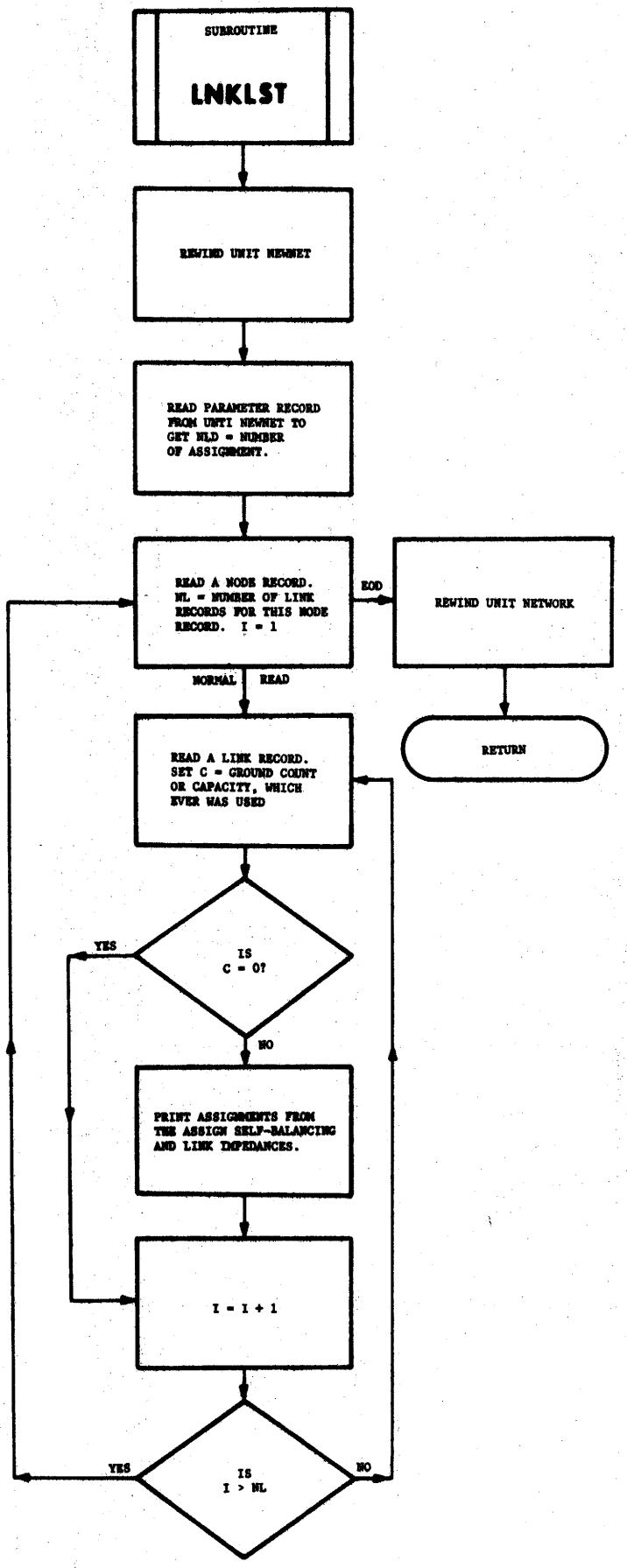

SUBROUTINE
GTLD
INITIALIZE SUMMATION VARIABLES TO ZERO. OPEN DATA SET ROUTE FOR ROUTE PROFILE
IS A WEIGHTED ASSIGNMENT TO BE CALCULATED?
YES
OPEN NEWNET UNIT FOR OUTPUT
NO
READ PARAMETER RECORD FROM UNIT NETWORK
IS A WEIGHTED ASSIGNMENT TO BE CALCULATED?
YES
ADD ONE TO NUMBER OF ASSIGNMENTS AND WRITE PARAMETER RECORD ON UNIT NETWORK
NO
READ HEADER RECORDS AND WRITE THESE RECORDS ON UNIT PLF ALSO SAVE LAST TWO HEADER RECORDS
INITIALIZE MORE SUMMATION VARIABLES TO ZERO AND INITIALIZE OTHER VARIABLES
B

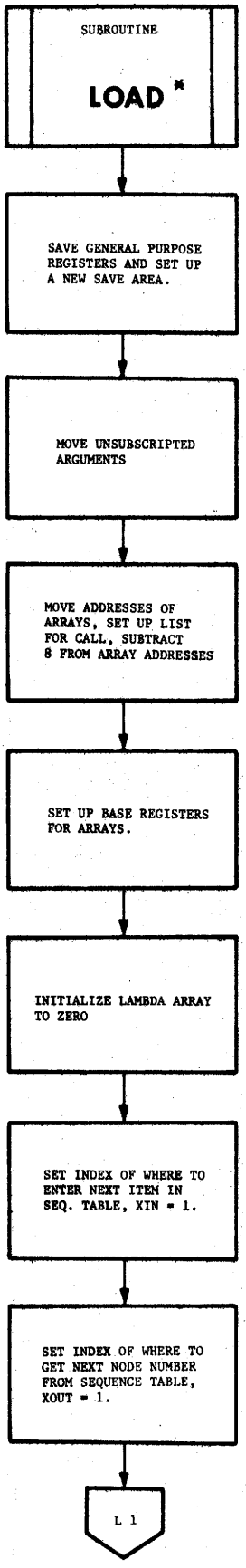

B
READ A NODE RECORD FROM UNIT NETWORK, IS THIS EOD?
YES
F
NO
NL = NUMBER OF LINKS FROM THIS NODE
I = 0
E
ARE THERE ANY ASSIGNMENTS ON THIS FLEXIBLE RECORD
NO
READ A LINK RECORD WITH NO ASSIGNMENTS
YES
READ A LINK RECORD WITH NONDIRECTIONAL ASSIGNED VOLUMES
IS A WEIGHTED ASSIGNMENT TO BE CALCULATED?
NO
YES
CALCULATE WEIGHTED IMPEDANCE AND WEIGHT VOLUME FOR THIS LINK
WRITE NEW LINK RECORD ON UNIT NEWNET WITH WEIGHTED IMPLEANCE AND VOLUME AS NEXT ASSIGNMENT
ARE GROUND COUNTS OR CAPACITIES TO BE USED?
CAPACITIES
GET LINK CAPACITY AND PUT IN NCC
GROUND COUNTS
GET LINK GROUND COUNT AND PUT IN NCC
ARE THE NUMBER OF ASSIGNMENTS > 1 AND IS NCC > 0
NO
YES
MAKE SUMMATIONS FOR CROSS CLASSIFICATIONS
GET LINK JURISDICTION, SPEED, DISTANCE, TIME, LAST ASSIGNED NON-DIRECTIONAL VOLUME, AND ROUTE CODE
IS EITHER NODE IN THIS LINK A CENTROID?
YES
C
NO
IS THE LINK COUNT FIELD ZERO?
YES
NO
MAKE SUMMATIONS FOR REGRESSIONS OF ASSIGNED VOLUME VERSUS LINK COUNTS BY ROUTE
D


D
IS THE CAPACITY FIELD OF THIS LINK ZERO?
YES
NO
MAKE SUMMATIONS FOR REGRESSION OF ASSIGNED VOLUMES VERSUS LINK CAPACITIES BY ROUTE
IS THE NUMBER OF ASSIGNMENTS = 1?
YES
NO
MAKE SUMMATIONS FOR REGRESSION OF ASSIGNED VOLUMES VERSUS PREVIOUS ASSIGNED VOLUMES BY ROUTE
C
MAKE SUMMATIONS FOR VEHICLE HOURS - VEHICLE MILES SUMMARY CLASSIFIED BY JURIS-DICTION CODE VERSUS 3 LINK TYPES
MAKE SUMMATIONS FOR VEHICLE HOURS, VEHICLE MILES SUMMARY CLASSIFIED BY JURISDICTION VERSUS FUNCTIONAL CLASS (ALSO FOR CAP. AND COUNT)
IS THE ROUTE > 0? AND A NODE < B NODE?
YES
NO
WRITE RECORD FOR THE ROUTE PROFILE ON UNIT PLF
GTLD
IS EITHER NODE OF THE LINK A CENTROID? OR IS NCC = 0?
YES
NO
MAKE SUMMATIONS FOR A REGRESSION OF LINK VOLUMES VERSUS NCC, COUNT OR CAPACITY.
IS THE CORRIDOR INTERCEPT = 0 OR A NODE < B NODE?
YES
NO
SAVE CORRIDOR INTERCEPT RECORD FOR THIS LINK IN CORE, INCREMENT COUNTER OF CORRIDOR INT. RECORDS
I = I + 1 (INCREMENT COUNT OF LINK RECORDS FOR THIS NODE RECORD)
IS I < NL?
YES
E
NO
B

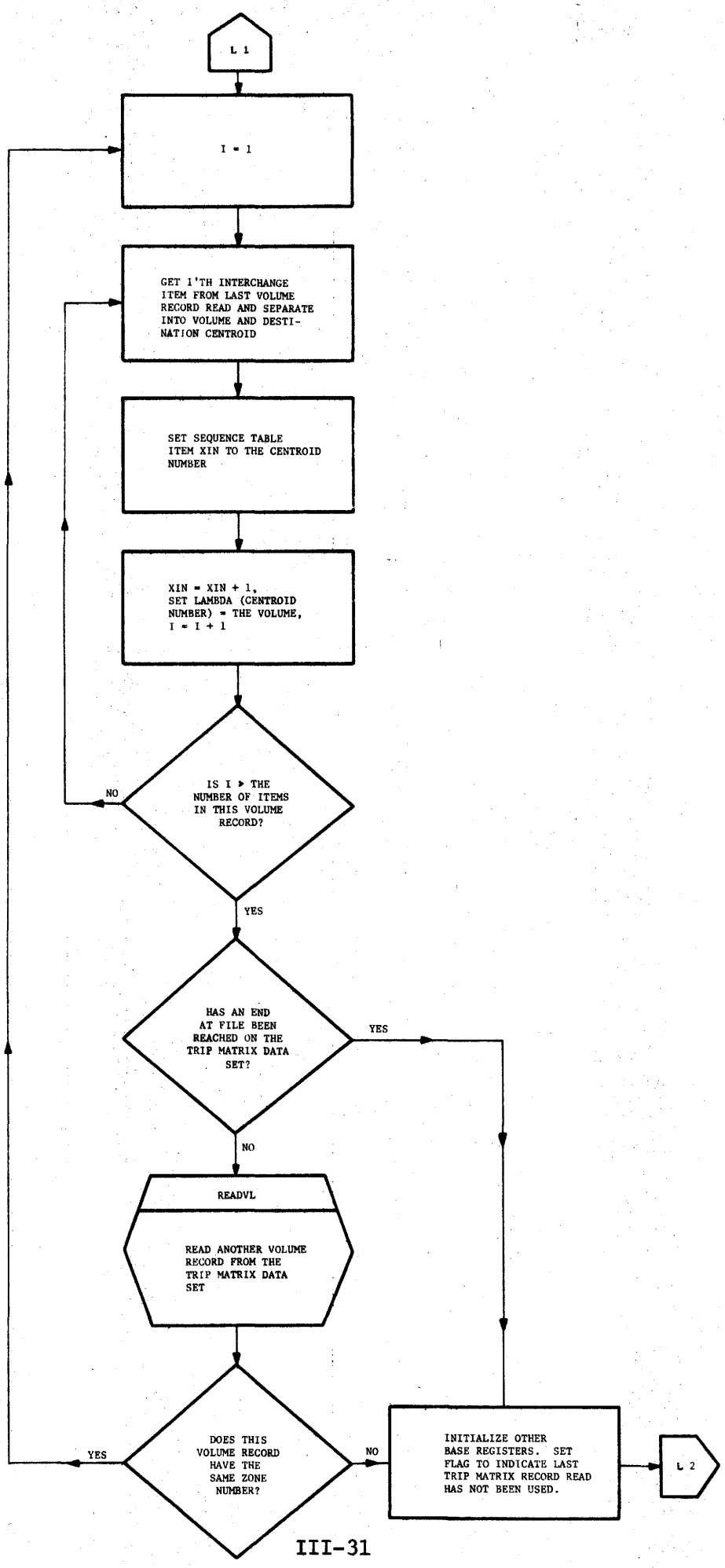

F
CONVERT SUMMATIONS ON VEHICLE HOURS, VEHICLE MILES, AND NETWORK MILES TO CORRECT UNITS FOR OUTPUT
CALCULATE THE NUMBER OF BYTES OF CORRIDOR INTERCEPT WORDS IN CORE.
HAS THERE BEEN MORE THAN ONE ASSIGNMENT?
NO
YES
PRINT V/C CROSS CLASSIFICATION FROM LAST TWO ASSIGNMENTS
ARE COUNTS OR CAPACITIES USED?
LINK COUNTS
LINK CAPACITIES
PRINT CROSS CLASSIFICATION OF LINK COUNTS BY V/C RATIO FROM LAST TWO ASSIGNMENTS
PRINT CROSS CLASSIFICATION OF LINK CAPACITIES BY V/C RATIO FROM LAST TWO ASSIGNMENTS
CLOSE AND REWIND ALL DATA SETS USED
RETURN

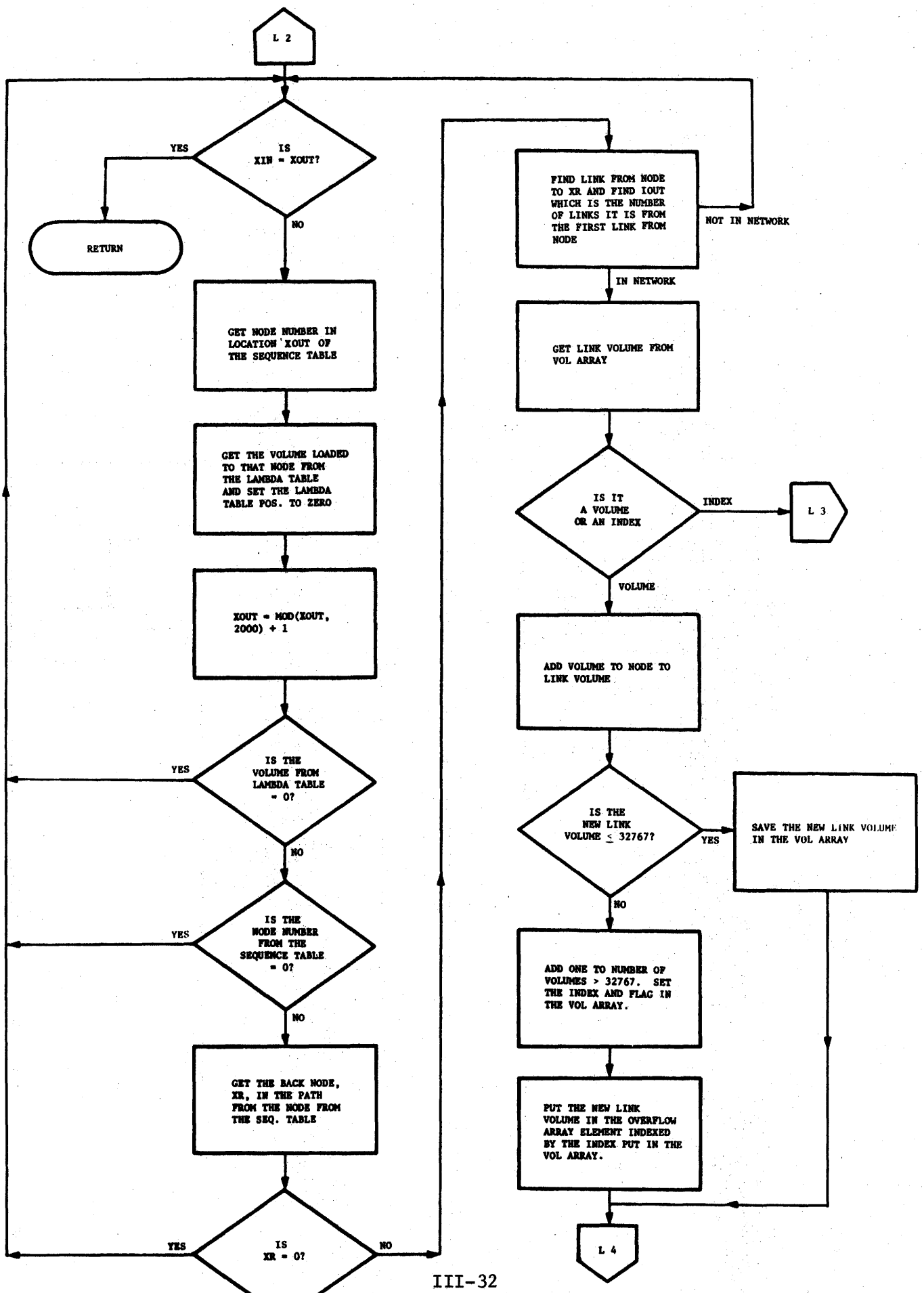

SUBROUTINE
INITL1
IS THE LAST CENTROID NUMBER > 1200?
YES
PRINT ERROR MESSAGE: LSTCEN .GT. 1200, = XXXXX
STOP
NO
IS THE LAST NODE NUMBER > 4000?
YES
PRINT ERROR MESSAGE: NODES .GT. 4000, = XXXXX
STOP
NO
IS THE NUMBER OF ONE-WAY LINKS > 16000?
YES
PRINT ERROR MESSAGE: LSTVOL .GT. 16000, = XXXXX
STOP
NO
ADD CENTROID FLAG TO TURNING MOVEMENT CODES FOR CENTROIDS
REMOVE FREEWAY FLAG FROM TURNING MOVEMENT CODES FOR NODES.
INITIALIZE LINK VOLUMES TO ZERO.
SET UP TURNING MOVEMENT INDEX ARRAY AND FIND NUMBER OF TURNING MOVEMENTS TO SAVE.
IS THE NUMBER OF TURNING MOVEMENTS TO SAVE > 20000?
YES
PRINT ERROR MESSAGE: NUMBER OF TURNING VOLUMES .GT. 20000 = XXXXX
STOP
NO
INITIALIZE TURNING VOLUME ARRAY TO ZERO
RETURN

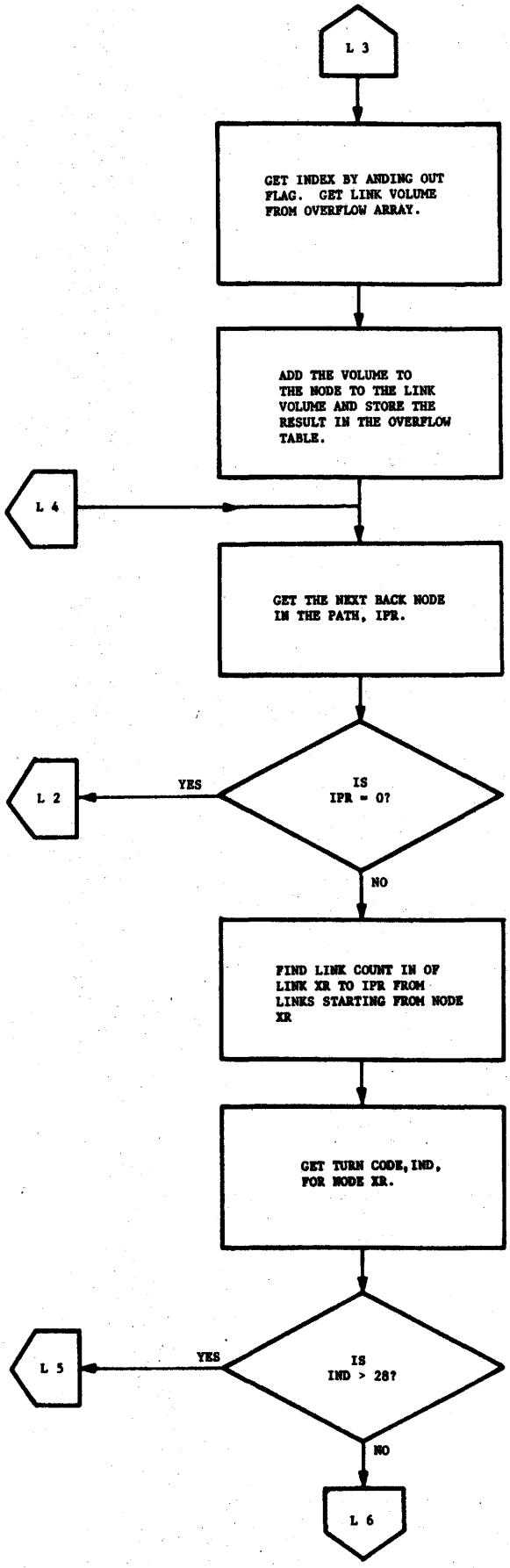

SUBROUTINE
LNKLST
REWIND UNIT NEWNET
READ PARAMETER RECORD FROM UNIT NEWNET TO GET NLD = NUMBER OF ASSIGNMENT.
READ A NODE RECORD. NL = NUMBER OF LINK RECORDS FOR THIS NODE RECORD. I = 1
EOD
REWIND UNIT NETWORK
RETURN
NORMAL READ
READ A LINK RECORD. SET C = GROUND COUNT OR CAPACITY, WHICH EVER WAS USED
IS C = 0?
YES
NO
PRINT ASSIGNMENTS FROM THE ASSIGN SELF-BALANCING AND LINK IMPEDANCES.
I = I + 1
IS I > NL
YES
NO

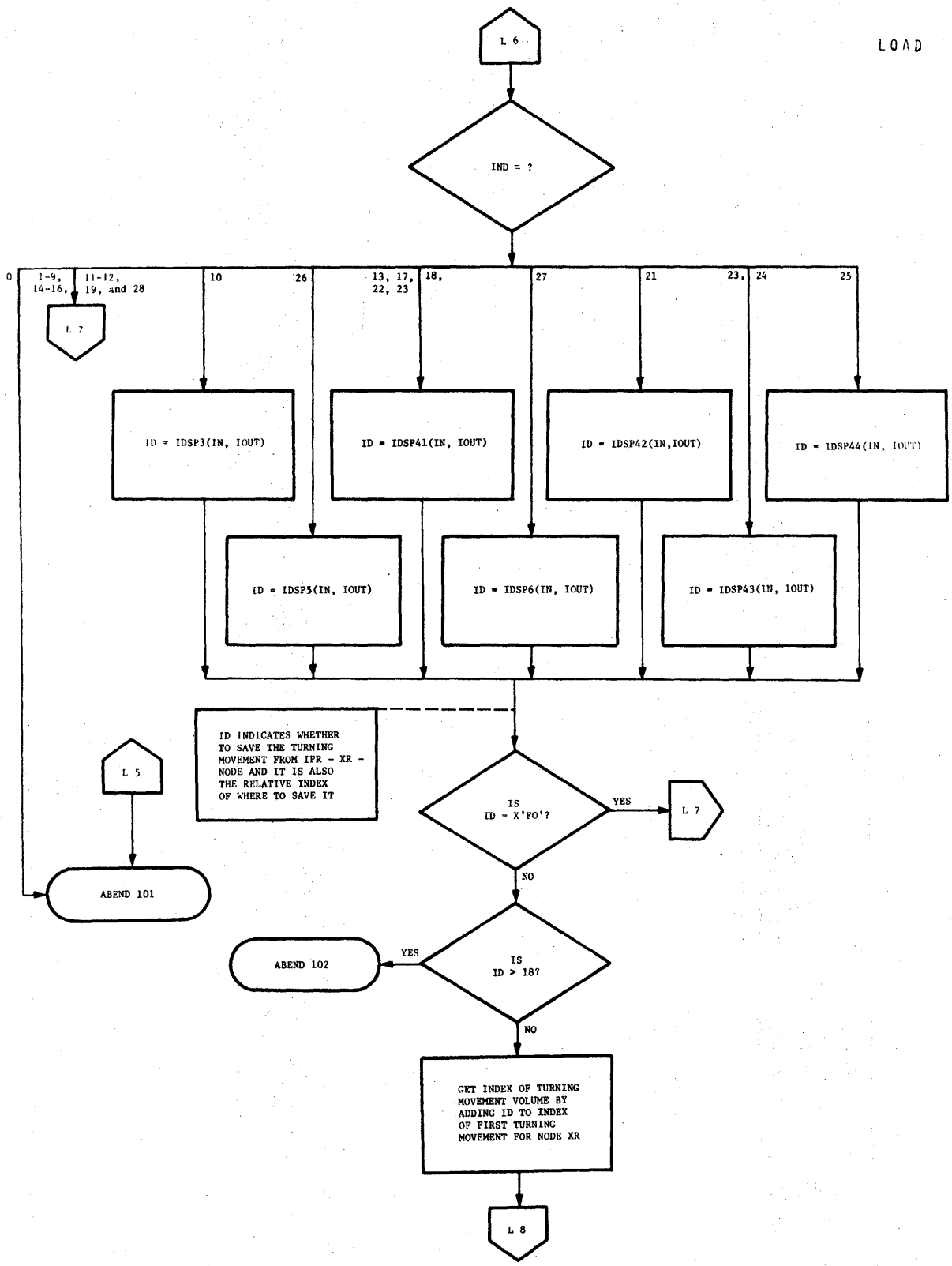


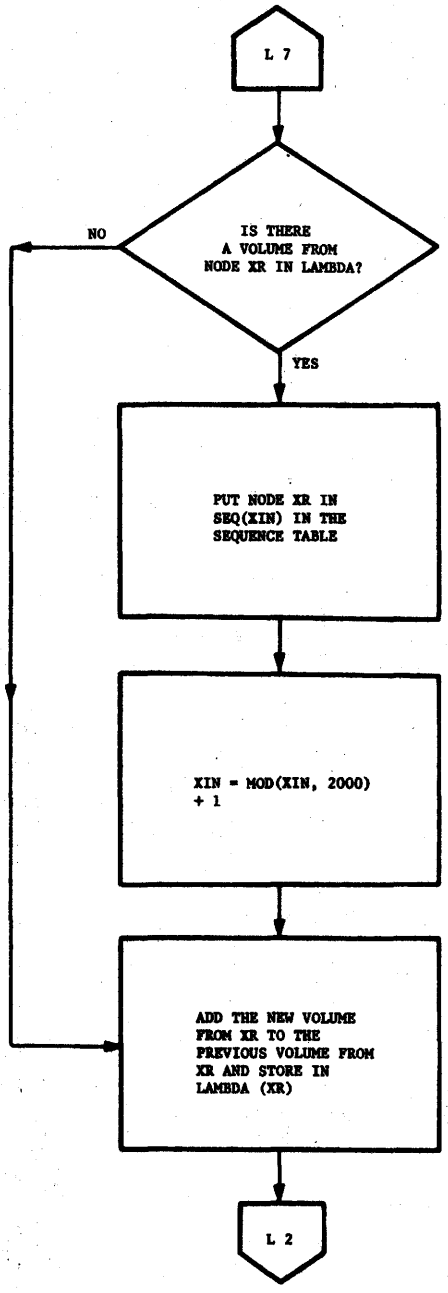
*ASSEMBLY LANGUAGE

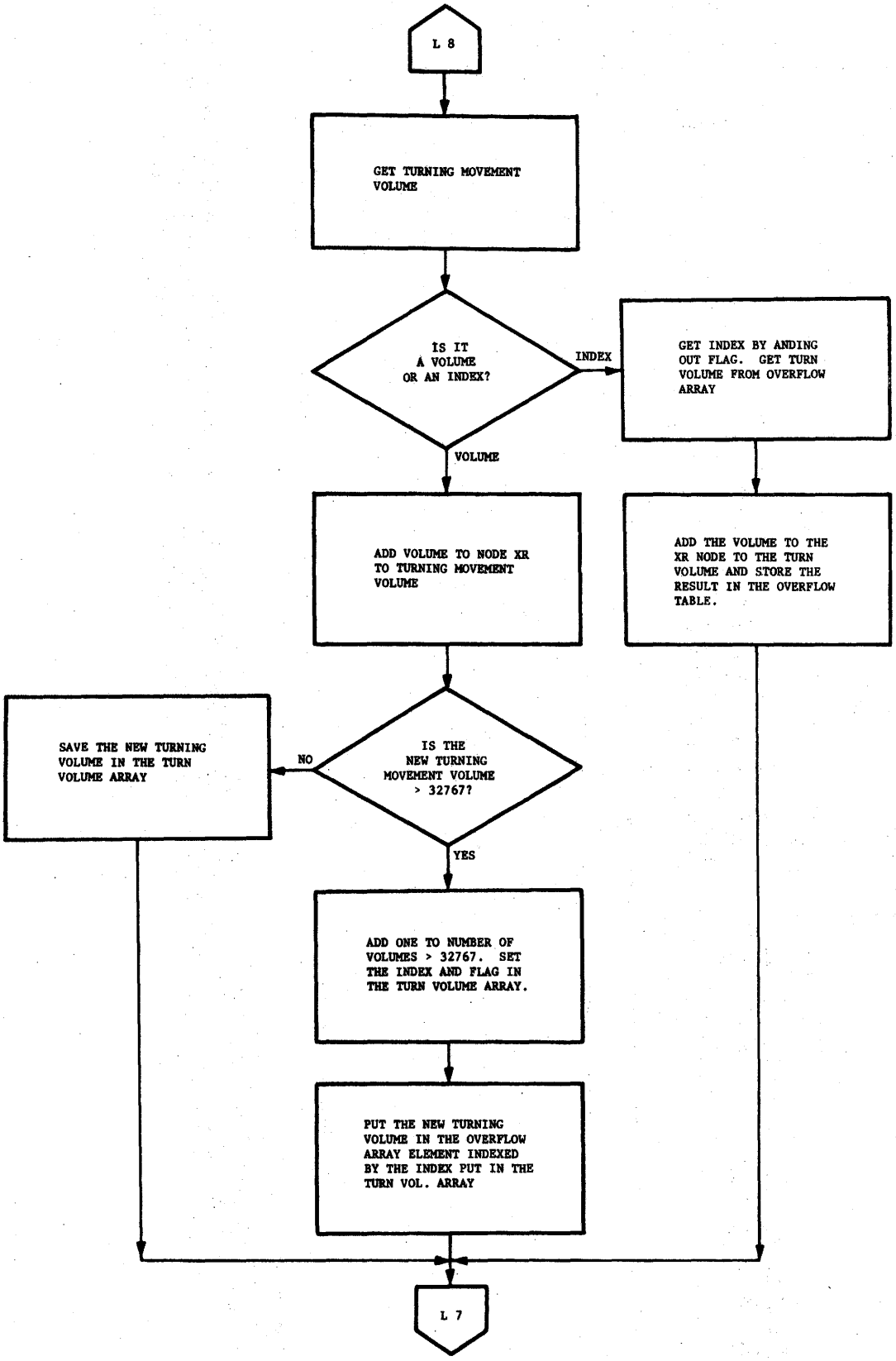

L 1
I = 1
GET I'TH INTERCHANGE ITEM FROM LAST VOLUME RECORD READ AND SEPARATE INTO VOLUME AND DESTI-NATION CENTROID
SET SEQUENCE TABLE ITEM XIN TO THE CENTROID NUMBER
XIN = XIN + 1, SET LAMBDA (CENTROID NUMBER) = THE VOLUME, I = I + 1
IS I > THE NUMBER OF ITEMS IN THIS VOLUME RECORD?
NO
YES
HAS AN END AT FILE BEEN REACHED ON THE TRIP MATRIX DATA SET?
YES
NO
READVL
READ ANOTHER VOLUME RECORD FROM THE TRIP MATRIX DATA SET
DOES THIS VOLUME RECORD HAVE THE SAME ZONE NUMBER?
YES
NO
INITIALIZE OTHER BASE REGISTERS. SET FLAG TO INDICATE LAST TRIP MATRIX RECORD READ HAS NOT BEEN USED.
L 2

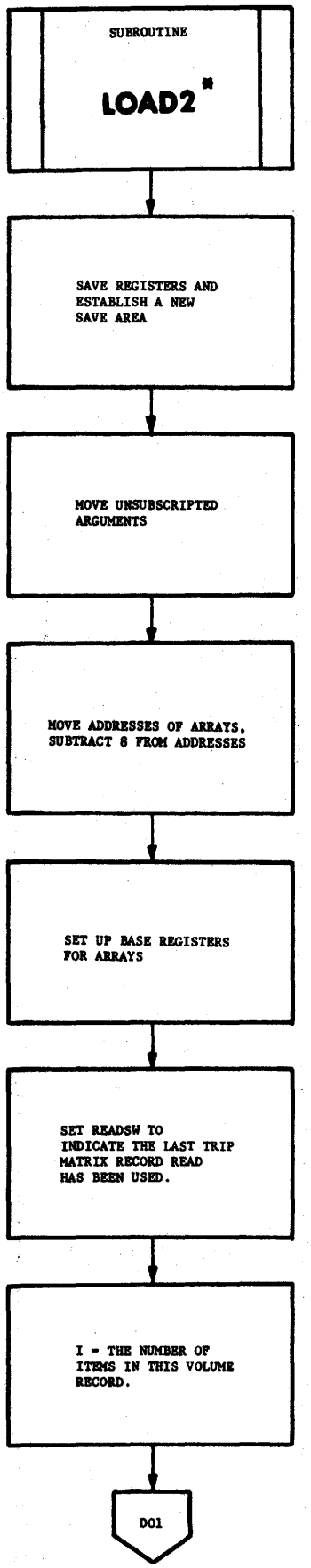

L 2
IS XIN = XOUT?
YES
RETURN
NO
GET NODE NUMBER IN LOCATION XOUT OF THE SEQUENCE TABLE
GET THE VOLUME LOADED TO THAT NODE FROM THE LAMBDA TABLE AND SET THE LAMBDA TABLE POS. TO ZERO
XOUT = MOD(XOUT, 2000) + 1
IS THE VOLUME FROM LAMBDA TABLE = 0?
YES
NO
IS THE NODE NUMBER FROM THE SEQUENCE TABLE = 0?
YES
NO
GET THE BACK NODE, XR, IN THE PATH FROM THE NODE FROM THE SEQ. TABLE
IS XR = 0?
YES
NO
FIND LINK FROM NODE TO XR AND FIND IOUT WHICH IS THE NUMBER OF LINKS IT IS FROM THE FIRST LINK FROM NODE
NOT IN NETWORK
IN NETWORK
GET LINK VOLUME FROM VOL ARRAY
IS IT A VOLUME OR AN INDEX
INDEX
L 3
VOLUME
ADD VOLUME TO NODE TO LINK VOLUME
IS THE NEW LINK VOLUME ≤ 32767?
YES
SAVE THE NEW LINK VOLUME IN THE VOL ARRAY
NO
ADD ONE TO NUMBER OF VOLUMES > 32767. SET THE INDEX AND FLAG IN THE VOL ARRAY.
PUT THE NEW LINK VOLUME IN THE OVERFLOW ARRAY ELEMENT INDEXED BY THE INDEX PUT IN THE VOL ARRAY.
L 4


L 3
GET INDEX BY ANDING OUT FLAG. GET LINK VOLUME FROM OVERFLOW ARRAY.
ADD THE VOLUME TO THE NODE TO THE LINK VOLUME AND STORE THE RESULT IN THE OVERFLOW TABLE.
L 4
GET THE NEXT BACK NODE IN THE PATH, IPR.
IS IPR = 0?
YES
L 2
NO
FIND LINK COUNT IN OF LINK XR TO IPR FROM LINKS STARTING FROM NODE XR
GET TURN CODE, IND, FOR NODE XR.
IS IND > 28?
YES
L 5
NO
L 6

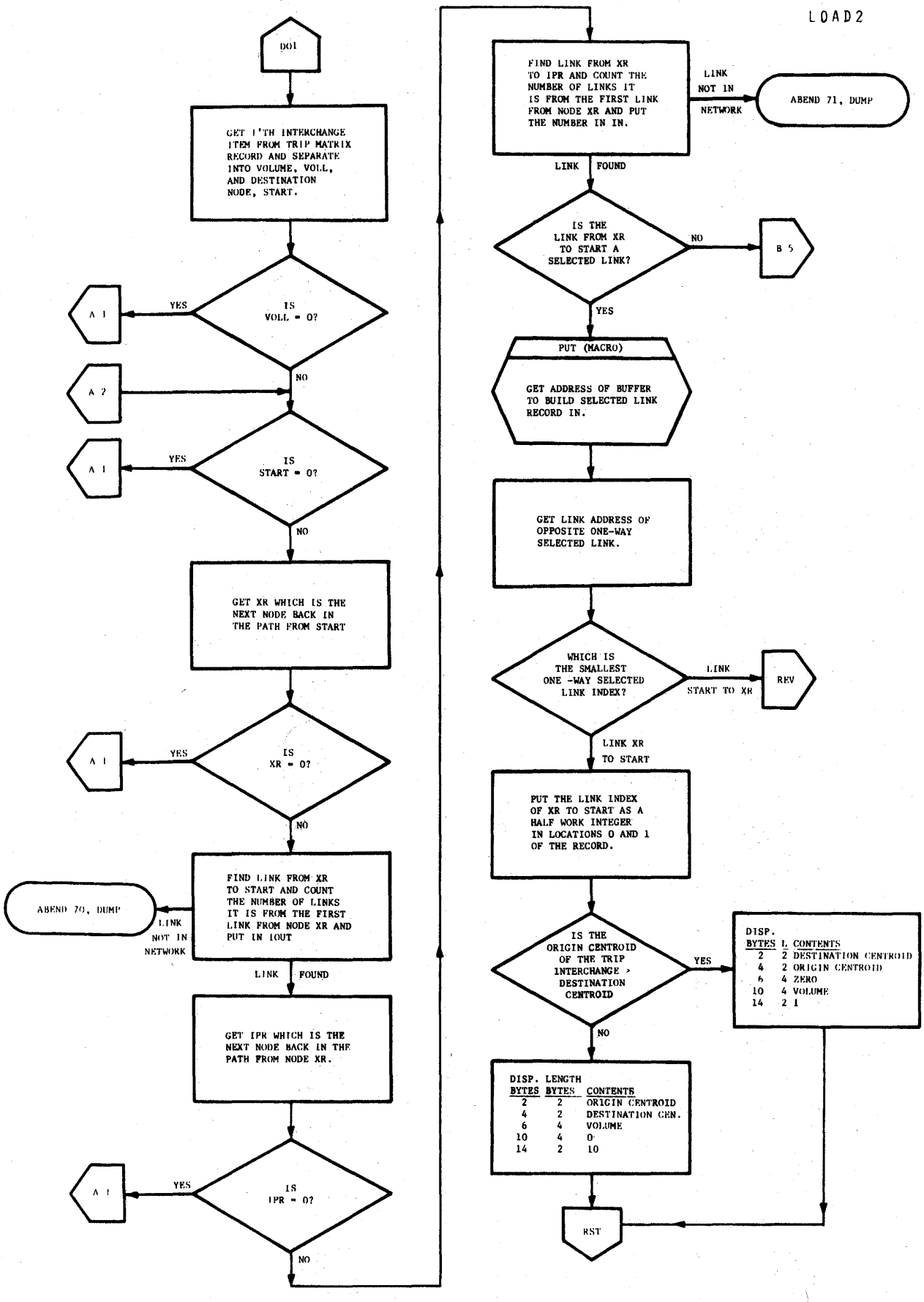

L 6
IND = ?
0
1-9, 14-16,
11-12, 19, and 28
10
26
13, 17, 22, 23
18,
27
21
23,
24
25
L 7
ID = IDSP3(IN, IOUT)
ID = IDSP41(IN, IOUT)
ID = IDSP42(IN,IOUT)
ID = IDSP44(IN, IOUT)
ID = IDSP5(IN, IOUT)
ID = IDSP6(IN, IOUT)
ID = IDSP43(IN, IOUT)
ID INDICATES WHETHER TO SAVE THE TURNING MOVEMENT FROM IPR - XR - NODE AND IT IS ALSO THE RELATIVE INDEX OF WHERE TO SAVE IT
L 5
IS ID = X'FO'?
YES
L 7
NO
ABEND 101
ABEND 102
YES
IS ID > 18?
NO
GET INDEX OF TURNING MOVEMENT VOLUME BY ADDING ID TO INDEX OF FIRST TURNING MOVEMENT FOR NODE XR
L 8

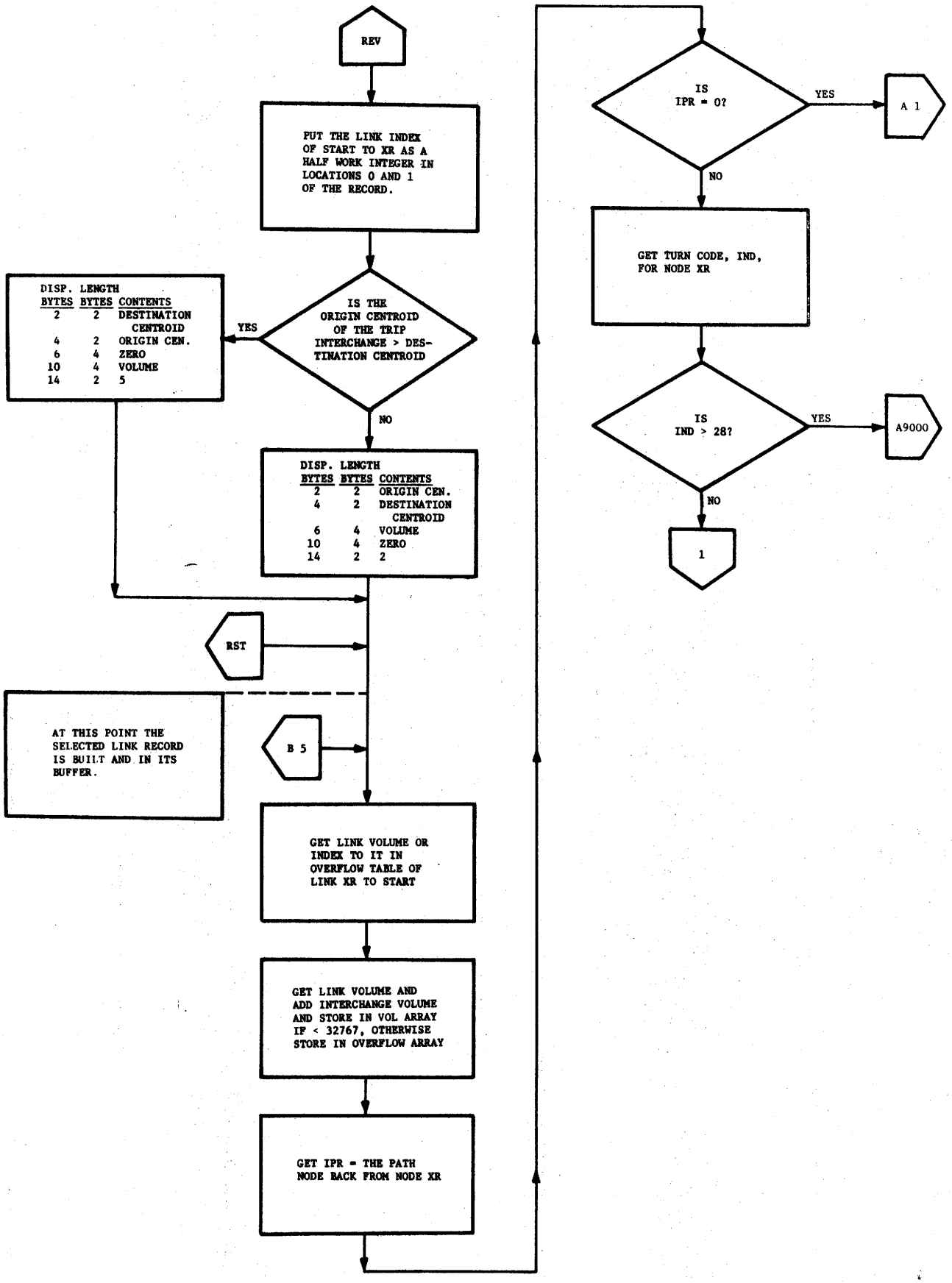

L 7
IS THERE A VOLUME FROM NODE XR IN LAMBDA?
NO
YES
PUT NODE XR IN SEQ(XIN) IN THE SEQUENCE TABLE
XIN = MOD(XIN, 2000) + 1
ADD THE NEW VOLUME FROM XR TO THE PREVIOUS VOLUME FROM XR AND STORE IN LAMBDA (XR)
L 2

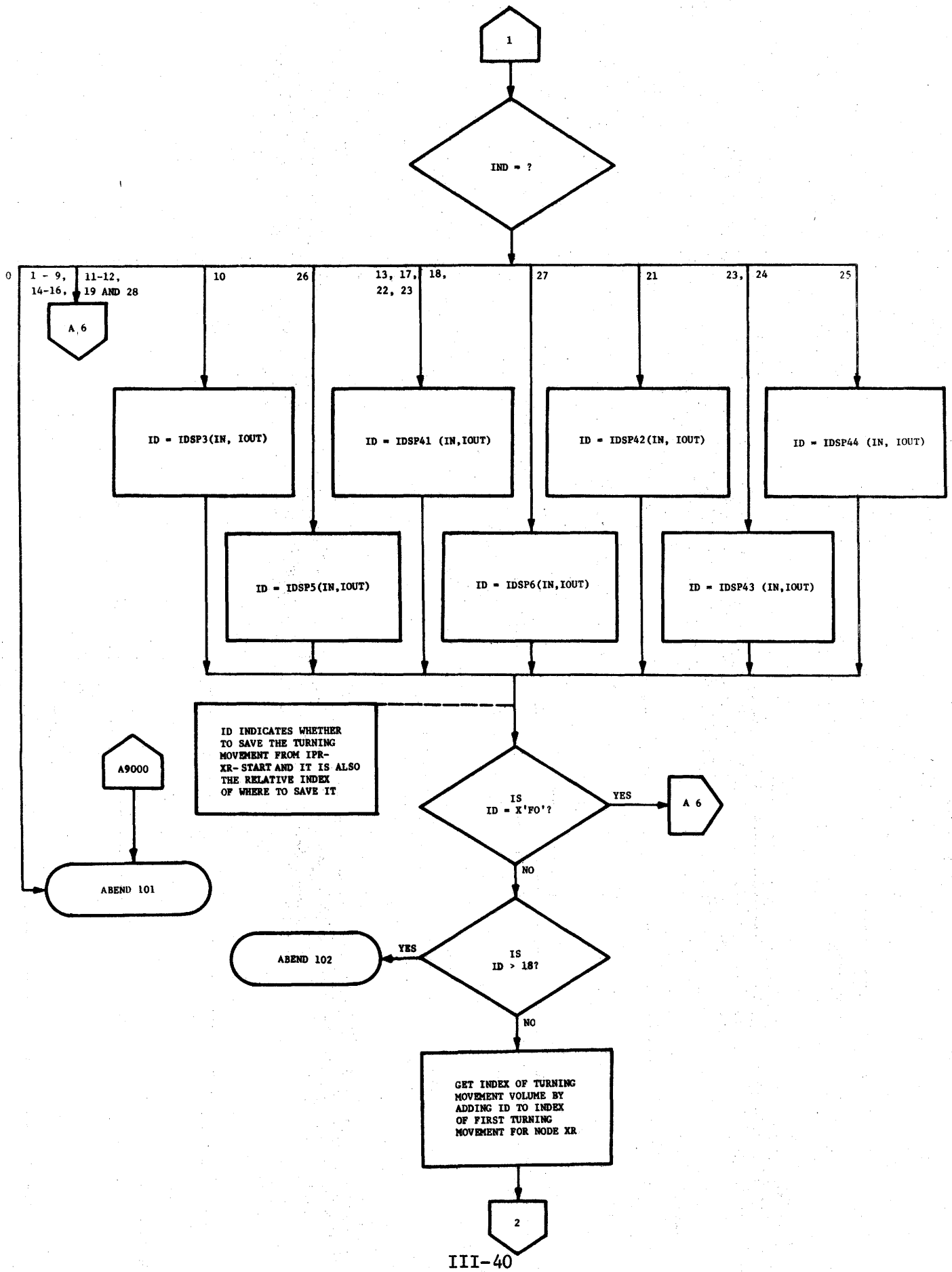

L 8
GET TURNING MOVEMENT VOLUME
IS IT A VOLUME OR AN INDEX?
INDEX
GET INDEX BY ANDING OUT FLAG. GET TURN VOLUME FROM OVERFLOW ARRAY
VOLUME
ADD VOLUME TO NODE XR TO TURNING MOVEMENT VOLUME
ADD THE VOLUME TO THE XR NODE TO THE TURN VOLUME AND STORE THE RESULT IN THE OVERFLOW TABLE.
SAVE THE NEW TURNING VOLUME IN THE TURN VOLUME ARRAY
NO
IS THE NEW TURNING MOVEMENT VOLUME > 32767?
YES
ADD ONE TO NUMBER OF VOLUMES > 32767. SET THE INDEX AND FLAG IN THE TURN VOLUME ARRAY.
PUT THE NEW TURNING VOLUME IN THE OVERFLOW ARRAY ELEMENT INDEXED BY THE INDEX PUT IN THE TURN VOL. ARRAY
L 7

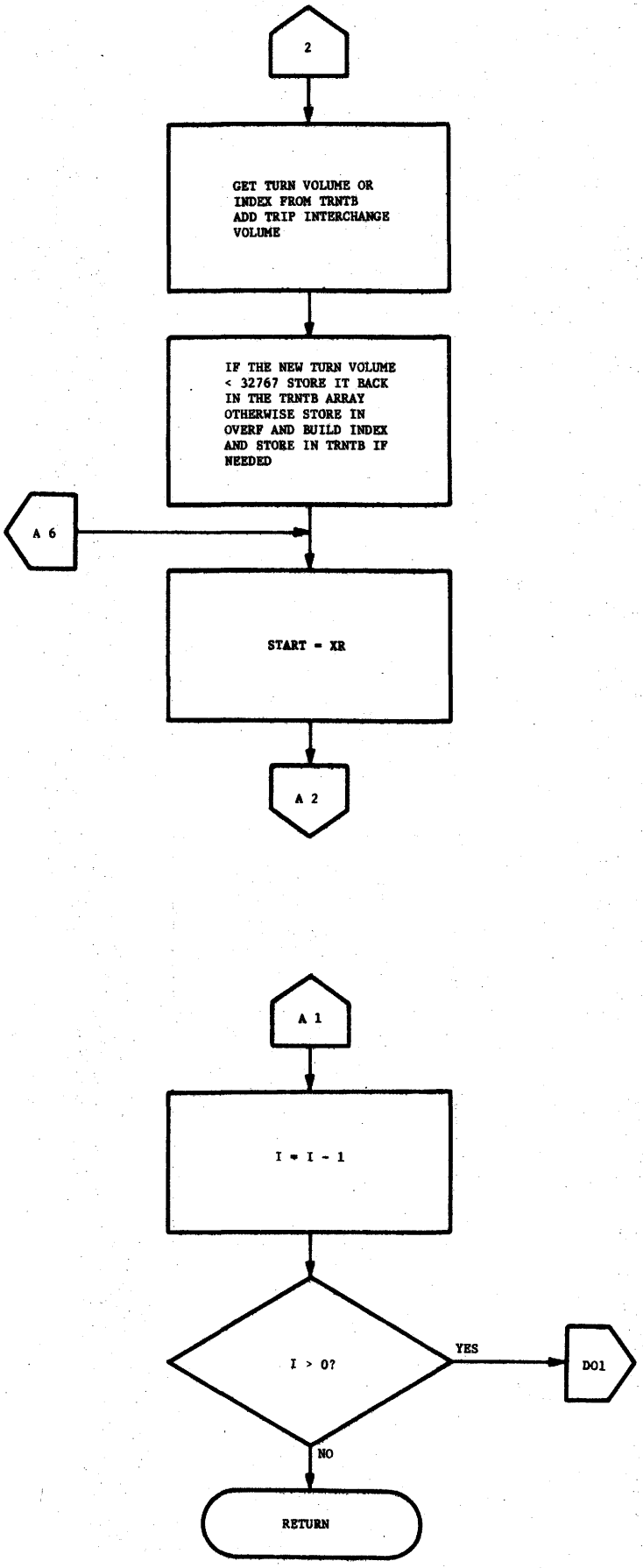


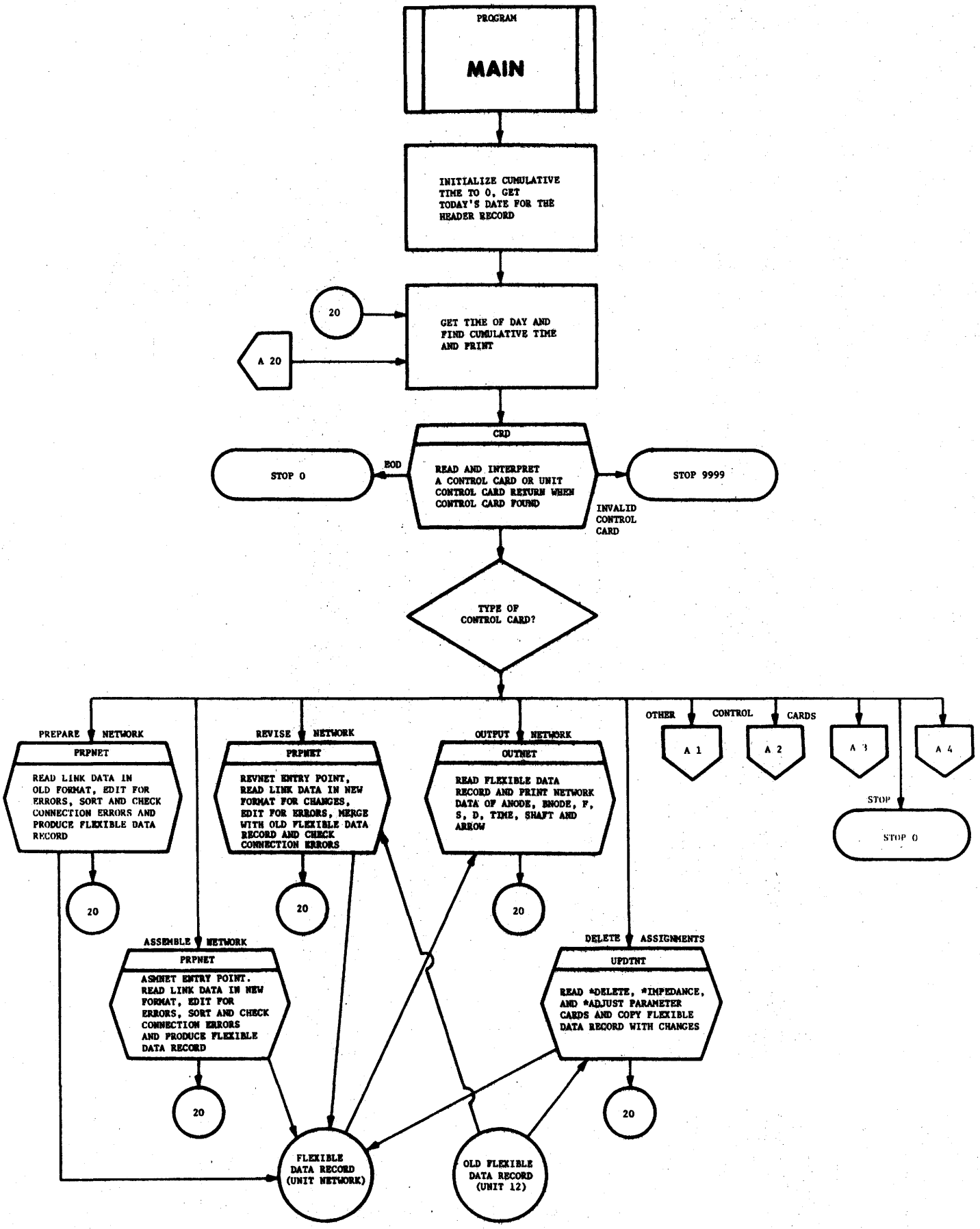
*ASSEMBLY LANGUAGE

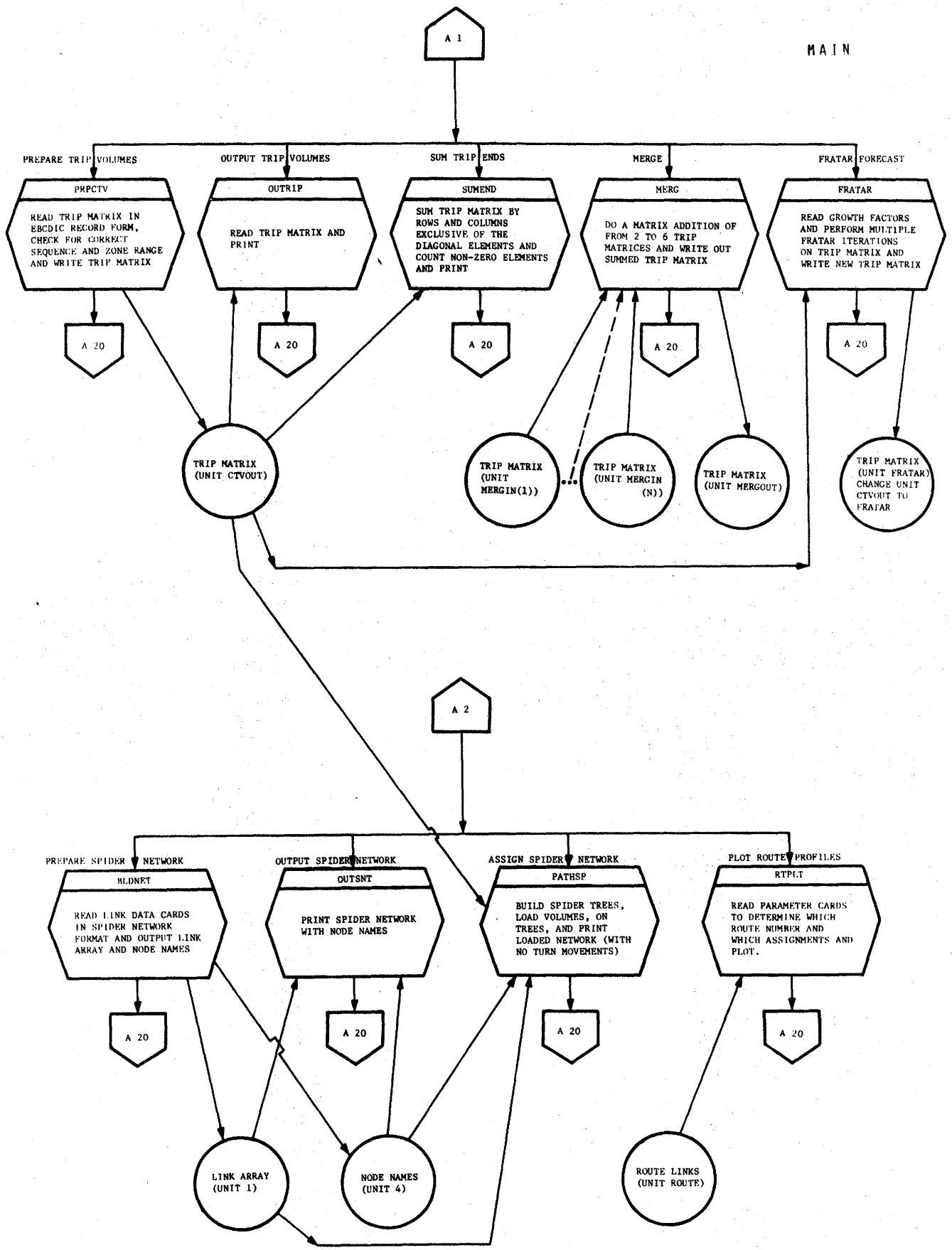

DO1
GET I'TH INTERCHANGE ITEM FROM TRIP MATRIX RECORD AND SEPARATE INTO VOLUME, VOLL, AND DESTINATION NODE, START.
IS VOLL = 0?
YES
A 1
NO
A 2
IS START = 0?
YES
A 1
NO
GET XR WHICH IS THE NEXT NODE BACK IN THE PATH FROM START
IS XR = 0?
YES
A 1
NO
FIND LINK FROM XR TO START AND COUNT THE NUMBER OF LINKS IT IS FROM THE FIRST LINK FROM NODE XR AND PUT IN IOUT
ABEND 70, DUMP
LINK NOT IN NETWORK
LINK FOUND
GET IPR WHICH IS THE NEXT NODE BACK IN THE PATH FROM NODE XR.
IS IPR = 0?
YES
A 1
NO
FIND LINK FROM XR TO IPR AND COUNT THE NUMBER OF LINKS IT IS FROM THE FIRST LINK FROM NODE XR AND PUT THE NUMBER IN IN.
LINK NOT IN NETWORK
ABEND 71, DUMP
LINK FOUND
IS THE LINK FROM XR TO START A SELECTED LINK?
NO
B 5
YES
PUT (MACRO)
GET ADDRESS OF BUFFER TO BUILD SELECTED LINK RECORD IN.
GET LINK ADDRESS OF OPPOSITE ONE-WAY SELECTED LINK.
WHICH IS THE SMALLEST ONE -WAY SELECTED LINK INDEX?
LINK START TO XR
REV
LINK XR TO START
PUT THE LINK INDEX OF XR TO START AS A HALF WORK INTEGER IN LOCATIONS 0 AND 1 OF THE RECORD.
IS THE ORIGIN CENTROID OF THE TRIP INTERCHANGE > DESTINATION CENTROID
YES
DISP. BYTES L CONTENTS
2 2 DESTINATION CENTROID
4 2 ORIGIN CENTROID
6 4 ZERO
10 4 VOLUME
14 2 1
NO
DISP. BYTES LENGTH BYTES CONTENTS
2 2 ORIGIN CENTROID
4 2 DESTINATION CEN.
6 4 VOLUME
10 4 0
14 2 10
RST

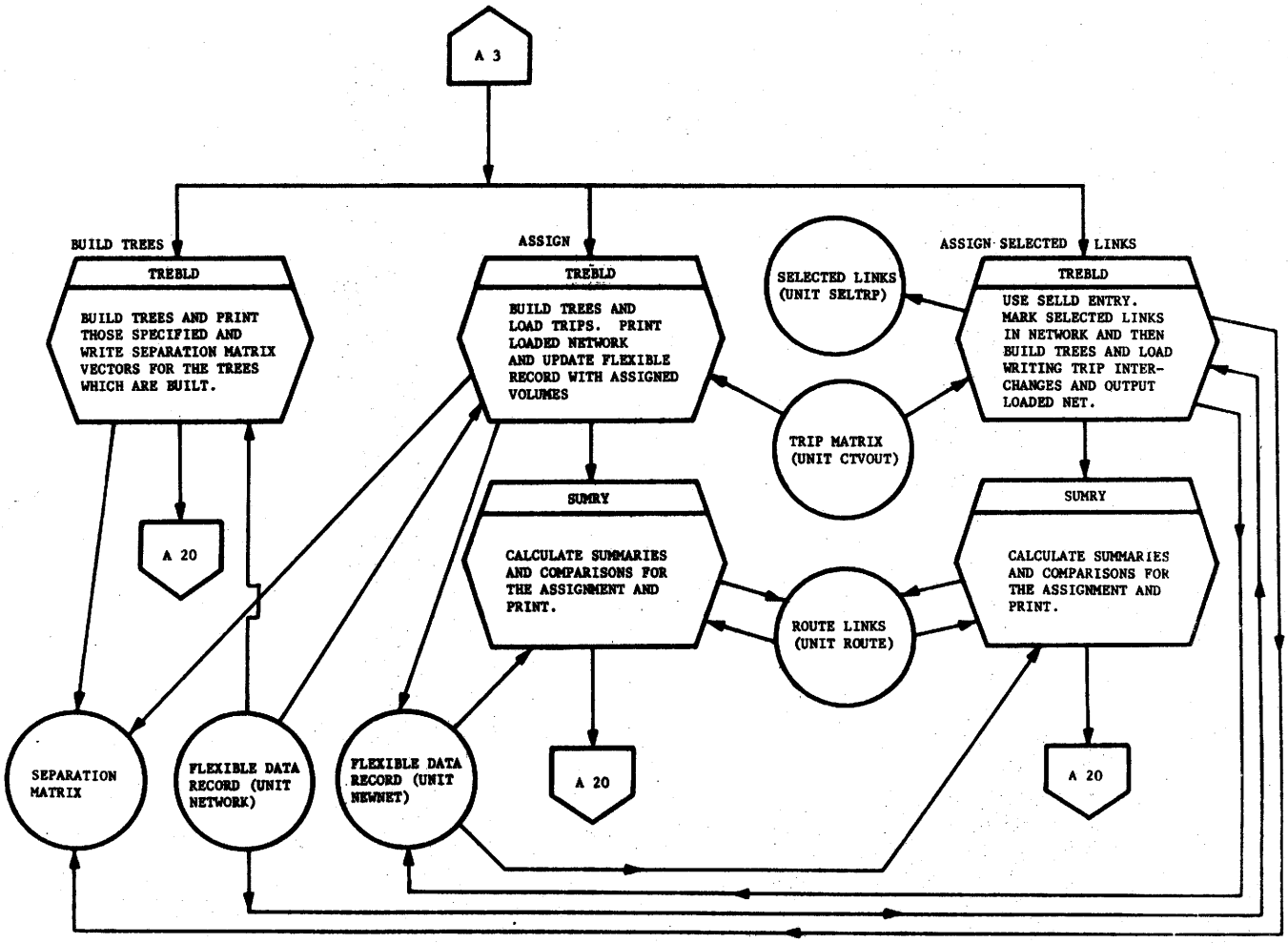

REV
PUT THE LINK INDEX OF START TO XR AS A HALF WORK INTEGER IN LOCATIONS 0 AND 1 OF THE RECORD.
IS THE ORIGIN CENTROID OF THE TRIP INTERCHANGE > DESTINATION CENTROID
YES
NO
DISP. BYTES LENGTH BYTES CONTENTS
2 2 DESTINATION CENTROID
4 2 ORIGIN CEN.
6 4 ZERO
10 4 VOLUME
14 2 5
DISP. BYTES LENGTH BYTES CONTENTS
2 2 ORIGIN CEN.
4 2 DESTINATION CENTROID
6 4 VOLUME
10 4 ZERO
14 2 2
RST
B 5
AT THIS POINT THE SELECTED LINK RECORD IS BUILT AND IN ITS BUFFER.
GET LINK VOLUME OR INDEX TO IT IN OVERFLOW TABLE OF LINK XR TO START
GET LINK VOLUME AND ADD INTERCHANGE VOLUME AND STORE IN VOL ARRAY IF < 32767, OTHERWISE STORE IN OVERFLOW ARRAY
GET IPR = THE PATH NODE BACK FROM NODE XR
IS IPR = 0?
YES
A 1
NO
GET TURN CODE, IND, FOR NODE XR
IS IND > 28?
YES
A9000
NO
1

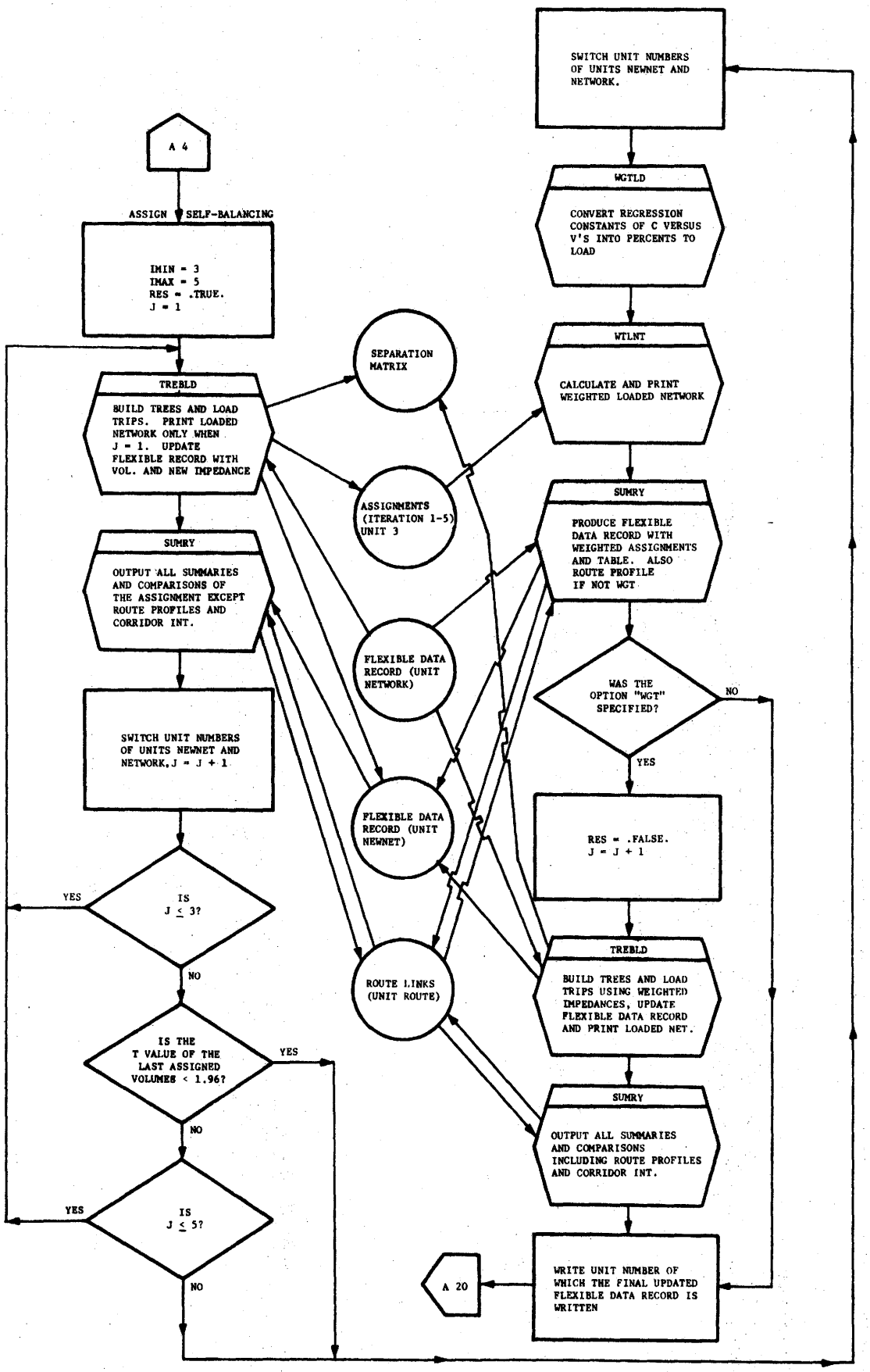

1
IND = ?
0
1 - 9, 14-16,
11-12, 19 AND 28
10
26
13, 17, 22, 23
18,
27
21
23,
24
25
A 6
ID = IDSP3(IN, IOUT)
ID = IDSP41 (IN,IOUT)
ID = IDSP42(IN, IOUT)
ID = IDSP44 (IN, IOUT)
ID = IDSP5(IN,IOUT)
ID = IDSP6(IN,IOUT)
ID = IDSP43 (IN,IOUT)
ID INDICATES WHETHER TO SAVE THE TURNING MOVEMENT FROM IPR-XR- START AND IT IS ALSO THE RELATIVE INDEX OF WHERE TO SAVE IT
A9000
IS ID = X'FO'?
YES
A 6
NO
ABEND 101
ABEND 102
YES
IS ID > 18?
NO
GET INDEX OF TURNING MOVEMENT VOLUME BY ADDING ID TO INDEX OF FIRST TURNING MOVEMENT FOR NODE XR
2

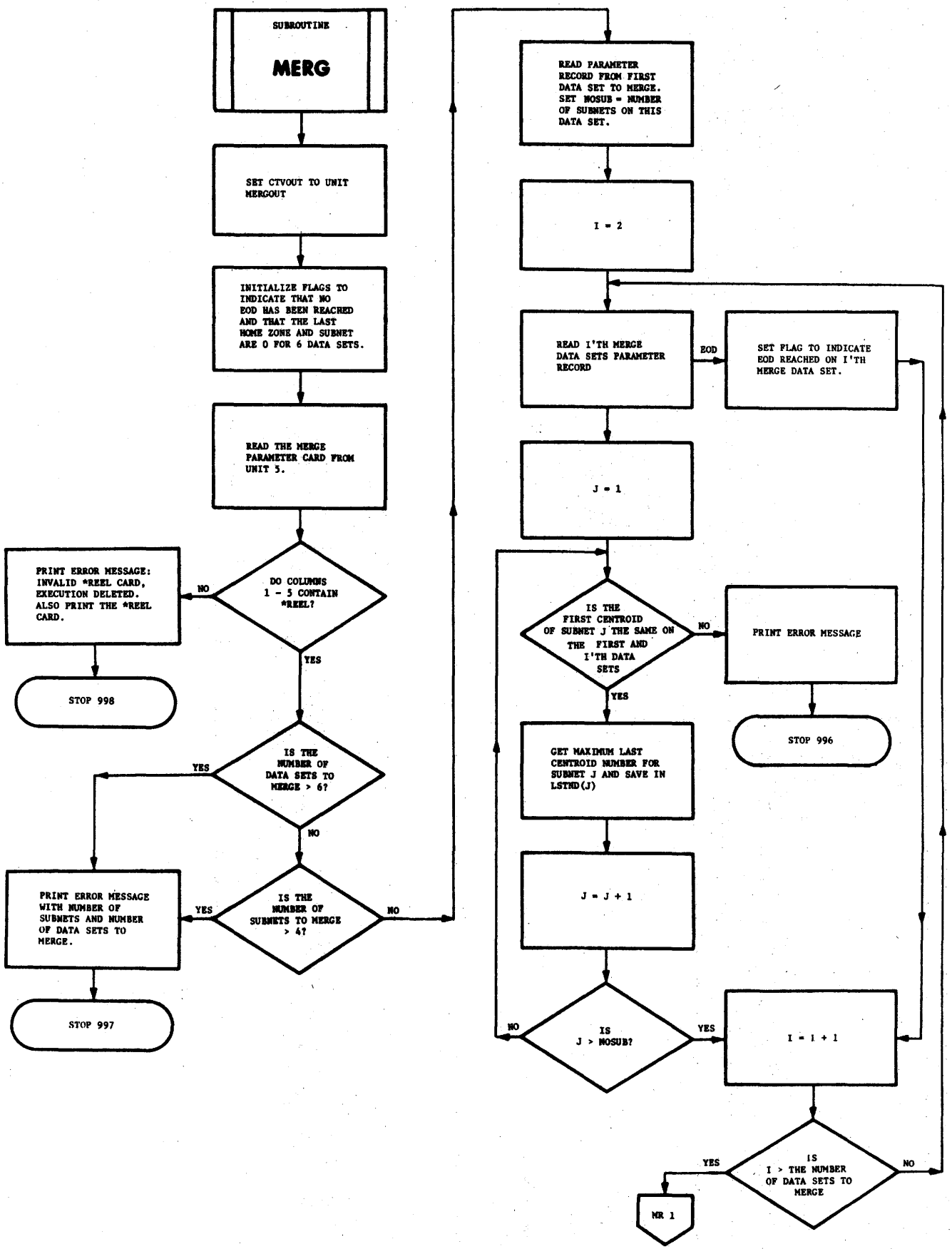

2
GET TURN VOLUME OR INDEX FROM TRNTB ADD TRIP INTERCHANGE VOLUME
IF THE NEW TURN VOLUME < 32767 STORE IT BACK IN THE TRNTB ARRAY OTHERWISE STORE IN OVERF AND BUILD INDEX AND STORE IN TRNTB IF NEEDED
A 6
START = XR
A 2
A 1
I = I - 1
I > 0?
YES
DO1
NO
RETURN

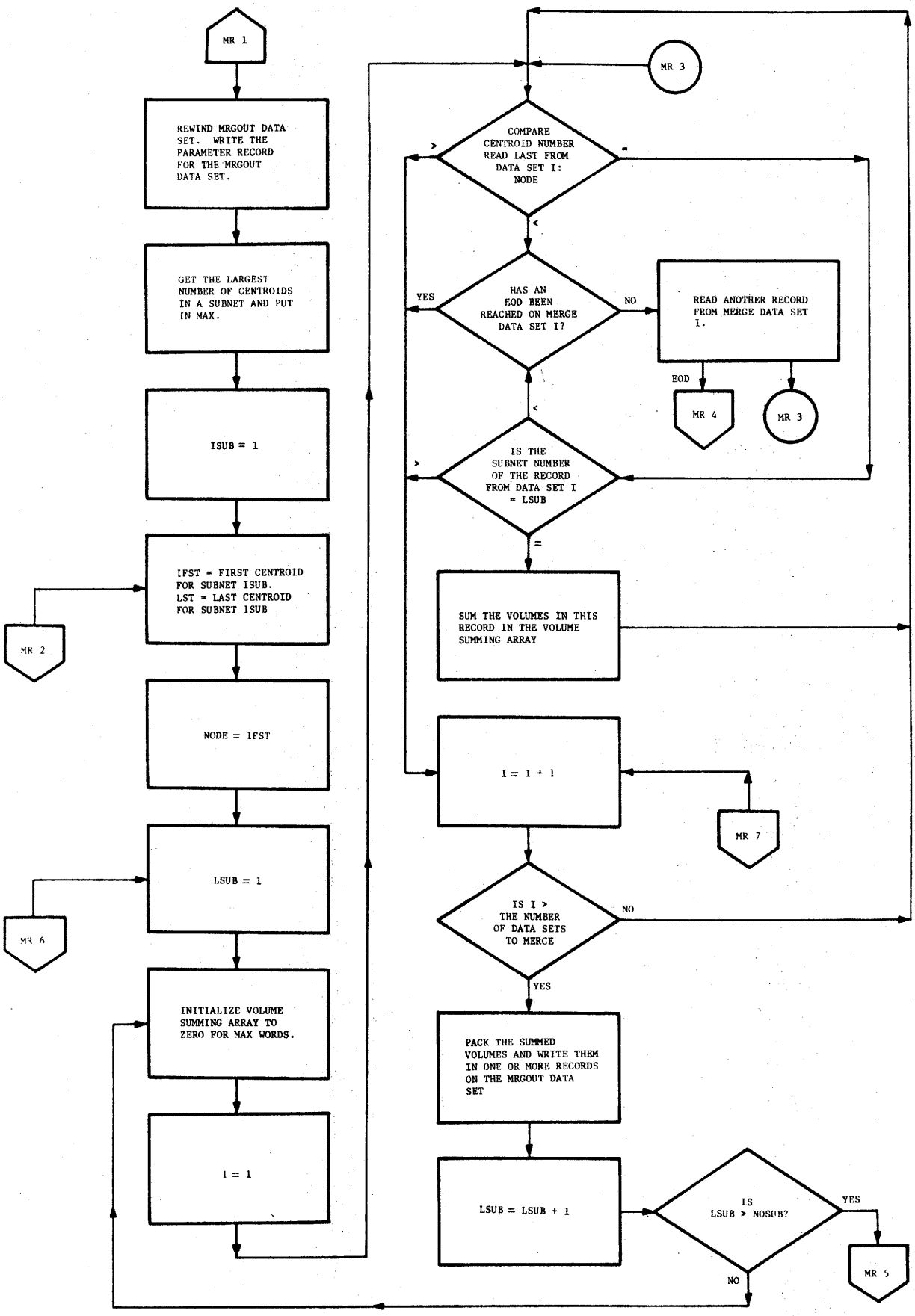

PROGRAM
MAIN
INITIALIZE CUMULATIVE TIME TO 0, GET TODAY'S DATE FOR THE HEADER RECORD
20
A 20
GET TIME OF DAY AND FIND CUMULATIVE TIME AND PRINT
CRD
STOP 0
EOD
READ AND INTERPRET A CONTROL CARD OR UNIT CONTROL CARD RETURN WHEN CONTROL CARD FOUND
STOP 9999
INVALID CONTROL CARD
TYPE OF CONTROL CARD?
PREPARE NETWORK
PRPNET
READ LINK DATA IN OLD FORMAT, EDIT FOR ERRORS, SORT AND CHECK CONNECTION ERRORS AND PRODUCE FLEXIBLE DATA RECORD
REVISE NETWORK
PRPNET
REVNET ENTRY POINT, READ LINK DATA IN NEW FORMAT FOR CHANGES, EDIT FOR ERRORS, MERGE WITH OLD FLEXIBLE DATA RECORD AND CHECK CONNECTION ERRORS
OUTPUT NETWORK
OUTNET
READ FLEXIBLE DATA RECORD AND PRINT NETWORK DATA OF ANODE, BNODE, F, S, D, TIME, SHAFT AND ARROW
OTHER CONTROL CARDS
A 1
A 2
A 3
A 4
STOP
STOP 0
20
20
20
ASSEMBLE NETWORK
PRPNET
ASMNET ENTRY POINT. READ LINK DATA IN NEW FORMAT, EDIT FOR ERRORS, SORT AND CHECK CONNECTION ERRORS AND PRODUCE FLEXIBLE DATA RECORD
DELETE ASSIGNMENTS
UPDTNT
READ *DELETE, *IMPEDANCE, AND *ADJUST PARAMETER CARDS AND COPY FLEXIBLE DATA RECORD WITH CHANGES
20
20
FLEXIBLE DATA RECORD (UNIT NETWORK)
OLD FLEXIBLE DATA RECORD (UNIT 12)

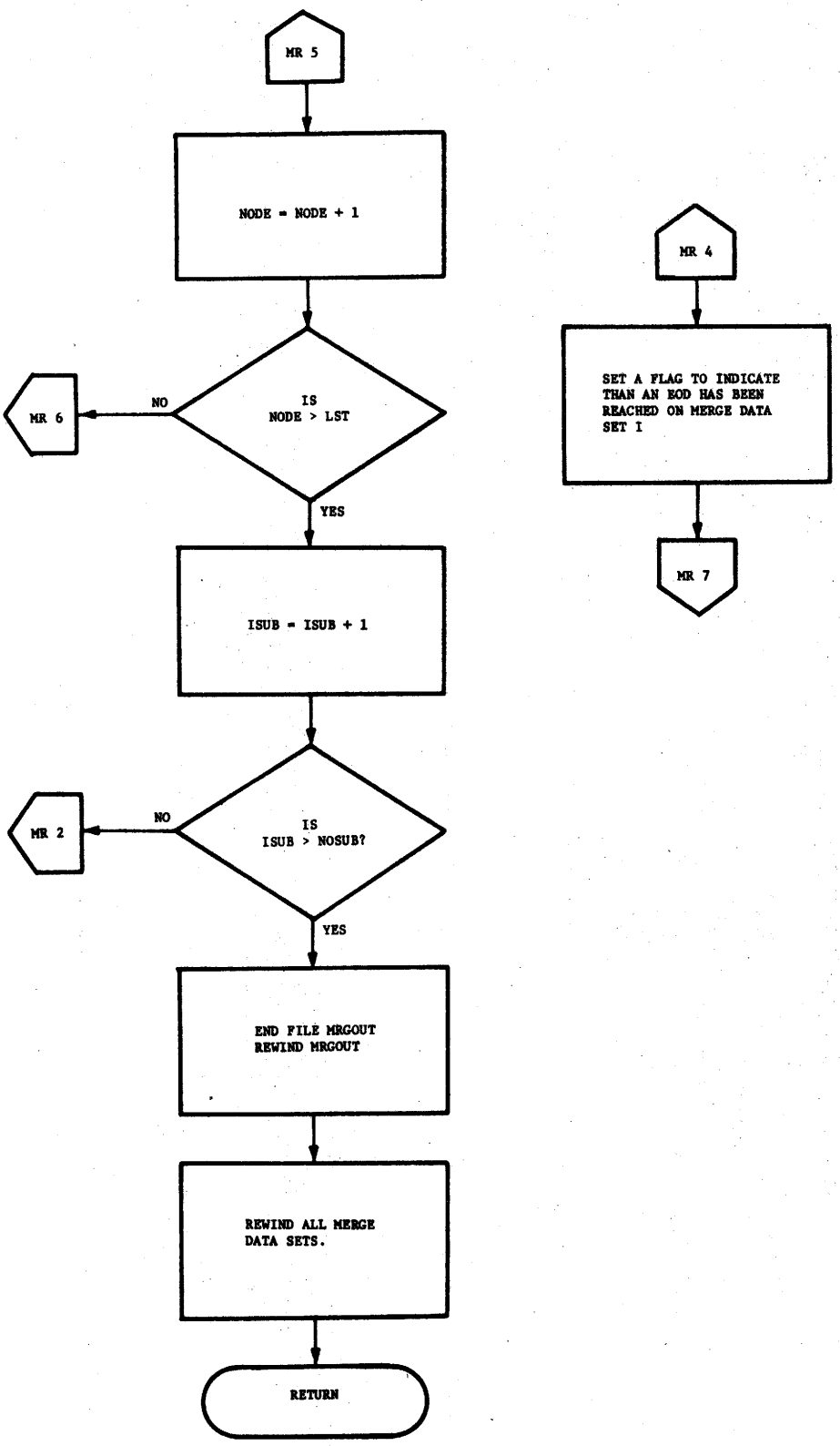

A 1
MAIN
PREPARE TRIP VOLUMES
OUTPUT TRIP VOLUMES
SUM TRIP ENDS
MERGE
FRATAR FORECAST
PRPCTV
READ TRIP MATRIX IN EBCDIC RECORD FORM, CHECK FOR CORRECT SEQUENCE AND ZONE RANGE AND WRITE TRIP MATRIX
OUTRIP
READ TRIP MATRIX AND PRINT
SUMEND
SUM TRIP MATRIX BY ROWS AND COLUMNS EXCLUSIVE OF THE DIAGONAL ELEMENTS AND COUNT NON-ZERO ELEMENTS AND PRINT
MERG
DO A MATRIX ADDITION OF FROM 2 TO 6 TRIP MATRICES AND WRITE OUT SUMMED TRIP MATRIX
FRATAR
READ GROWTH FACTORS AND PERFORM MULTIPLE FRATAR ITERATIONS ON TRIP MATRIX AND WRITE NEW TRIP MATRIX
A 20
A 20
A 20
A 20
A 20
TRIP MATRIX (UNIT CTVOUT)
TRIP MATRIX (UNIT MERGIN(1))
TRIP MATRIX (UNIT MERGIN (N))
TRIP MATRIX (UNIT MERGOUT)
TRIP MATRIX (UNIT FRATAR) CHANGE UNIT CTVOUT TO FRATAR
A 2
PREPARE SPIDER NETWORK
OUTPUT SPIDER NETWORK
ASSIGN SPIDER NETWORK
PLOT ROUTE PROFILES
BLDNET
READ LINK DATA CARDS IN SPIDER NETWORK FORMAT AND OUTPUT LINK ARRAY AND NODE NAMES
OUTSNT
PRINT SPIDER NETWORK WITH NODE NAMES
PATHSP
BUILD SPIDER TREES, LOAD VOLUMES, ON TREES, AND PRINT LOADED NETWORK (WITH NO TURN MOVEMENTS)
RTPLT
READ PARAMETER CARDS TO DETERMINE WHICH ROUTE NUMBER AND WHICH ASSIGNMENTS AND PLOT.
A 20
A 20
A 20
A 20
LINK ARRAY (UNIT 1)
NODE NAMES (UNIT 4)
ROUTE LINKS (UNIT ROUTE)

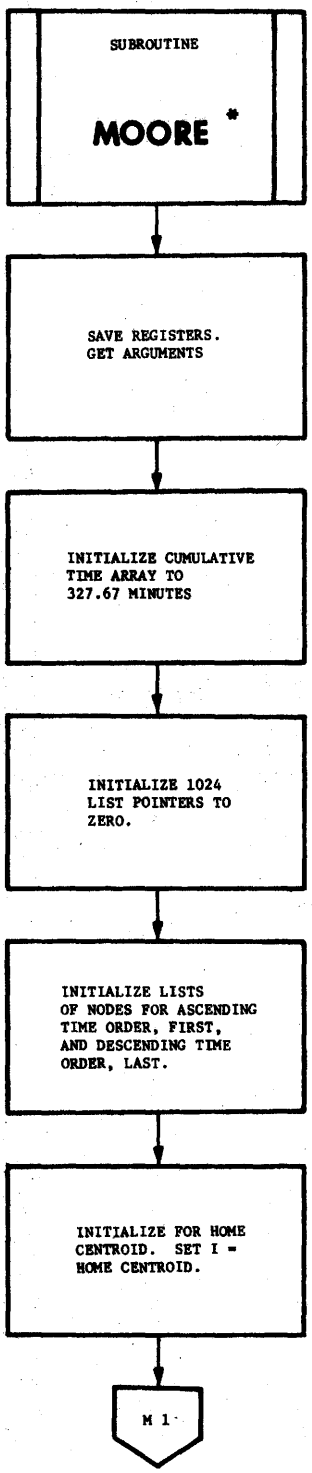

A 3
BUILD TREES
TREBLD
BUILD TREES AND PRINT THOSE SPECIFIED AND WRITE SEPARATION MATRIX VECTORS FOR THE TREES WHICH ARE BUILT.
A 20
ASSIGN
TREBLD
BUILD TREES AND LOAD TRIPS. PRINT LOADED NETWORK AND UPDATE FLEXIBLE RECORD WITH ASSIGNED VOLUMES
SUMRY
CALCULATE SUMMARIES AND COMPARISONS FOR THE ASSIGNMENT AND PRINT.
A 20
SELECTED LINKS (UNIT SELTRP)
TRIP MATRIX (UNIT CTVOUT)
ROUTE LINKS (UNIT ROUTE)
ASSIGN SELECTED LINKS
TREBLD
USE SELLD ENTRY. MARK SELECTED LINKS IN NETWORK AND THEN BUILD TREES AND LOAD WRITING TRIP INTER-CHANGES AND OUTPUT LOADED NET.
SUMRY
CALCULATE SUMMARIES AND COMPARISONS FOR THE ASSIGNMENT AND PRINT.
A 20
SEPARATION MATRIX
FLEXIBLE DATA RECORD (UNIT NETWORK)
FLEXIBLE DATA RECORD (UNIT NEWNET)


A 4
ASSIGN SELF-BALANCING
IMIN = 3
IMAX = 5
RES = .TRUE.
J = 1
TREBLD
BUILD TREES AND LOAD TRIPS. PRINT LOADED NETWORK ONLY WHEN J = 1. UPDATE FLEXIBLE RECORD WITH VOL. AND NEW IMPEDANCE
SEPARATION MATRIX
ASSIGNMENTS (ITERATION 1-5) UNIT 3
SUMRY
OUTPUT ALL SUMMARIES AND COMPARISONS OF THE ASSIGNMENT EXCEPT ROUTE PROFILES AND CORRIDOR INT.
FLEXIBLE DATA RECORD (UNIT NETWORK)
SWITCH UNIT NUMBERS OF UNITS NEWNET AND NETWORK, J = J + 1
FLEXIBLE DATA RECORD (UNIT NEWNET)
YES
IS J ≤ 3?
NO
ROUTE LINKS (UNIT ROUTE)
IS THE T VALUE OF THE LAST ASSIGNED VOLUMES < 1.96?
YES
NO
YES
IS J ≤ 5?
NO
SWITCH UNIT NUMBERS OF UNITS NEWNET AND NETWORK.
WGTLD
CONVERT REGRESSION CONSTANTS OF C VERSUS V'S INTO PERCENTS TO LOAD
WTLNT
CALCULATE AND PRINT WEIGHTED LOADED NETWORK
SUMRY
PRODUCE FLEXIBLE DATA RECORD WITH WEIGHTED ASSIGNMENTS AND TABLE. ALSO ROUTE PROFILE IF NOT WGT
WAS THE OPTION "WGT" SPECIFIED?
NO
YES
RES = .FALSE.
J = J + 1
TREBLD
BUILD TREES AND LOAD TRIPS USING WEIGHTED IMPEDANCES, UPDATE FLEXIBLE DATA RECORD AND PRINT LOADED NET.
SUMRY
OUTPUT ALL SUMMARIES AND COMPARISONS INCLUDING ROUTE PROFILES AND CORRIDOR INT.
WRITE UNIT NUMBER OF WHICH THE FINAL UPDATED FLEXIBLE DATA RECORD IS WRITTEN
A 20

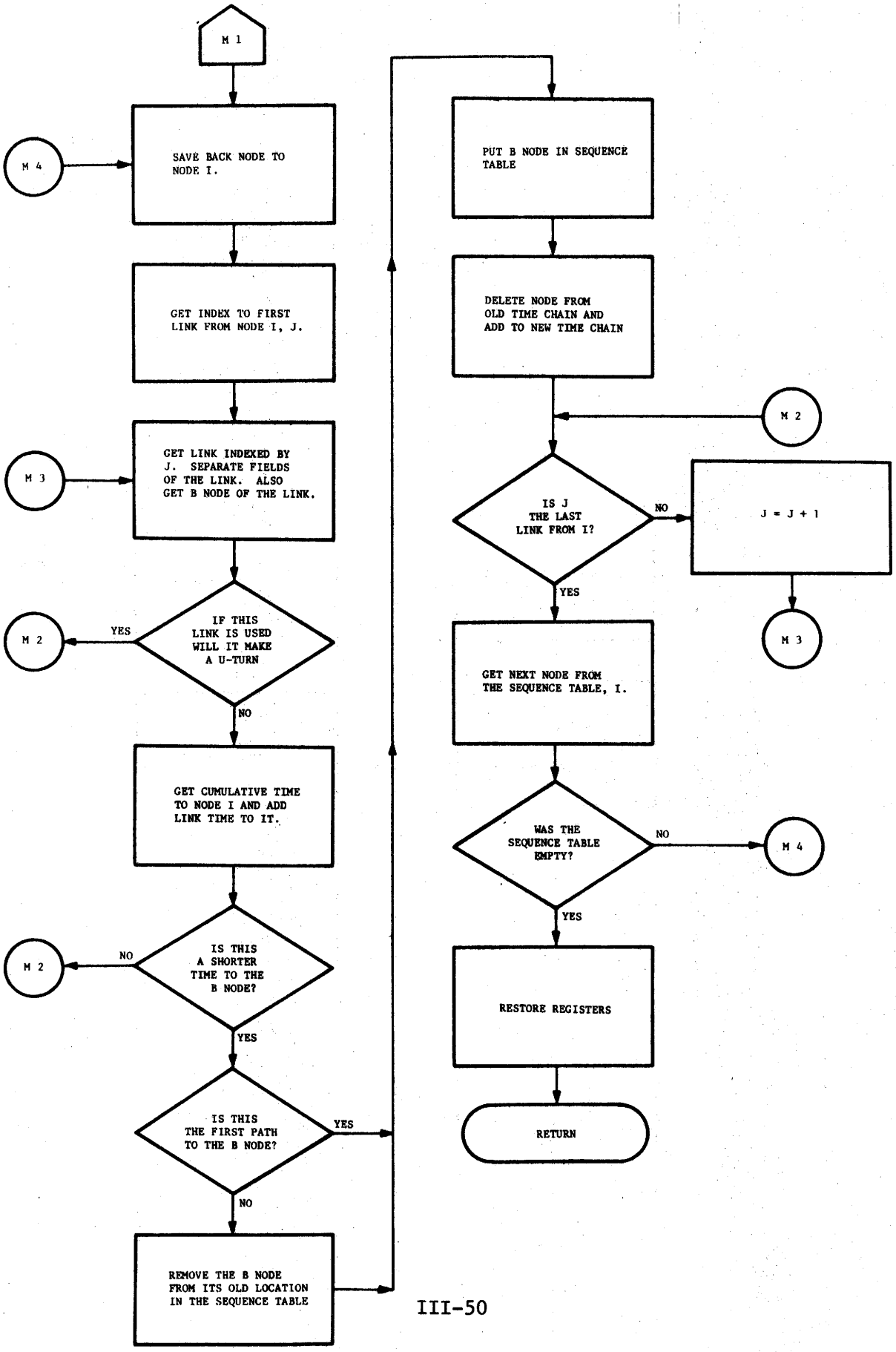

SUBROUTINE
MERG
SET CTVOUT TO UNIT MERGOUT
INITIALIZE FLAGS TO INDICATE THAT NO EOD HAS BEEN REACHED AND THAT THE LAST HOME ZONE AND SUBNET ARE 0 FOR 6 DATA SETS.
READ THE MERGE PARAMETER CARD FROM UNIT 5.
DO COLUMNS 1 - 5 CONTAIN *REEL?
NO
PRINT ERROR MESSAGE: INVALID *REEL CARD, EXECUTION DELETED. ALSO PRINT THE *REEL CARD.
STOP 998
YES
IS THE NUMBER OF DATA SETS TO MERGE > 6?
YES
PRINT ERROR MESSAGE WITH NUMBER OF SUBNETS AND NUMBER OF DATA SETS TO MERGE.
STOP 997
NO
IS THE NUMBER OF SUBNETS TO MERGE > 4?
YES
NO
READ PARAMETER RECORD FROM FIRST DATA SET TO MERGE. SET NOSUB = NUMBER OF SUBNETS ON THIS DATA SET.
I = 2
READ I'TH MERGE DATA SETS PARAMETER RECORD
EOD
SET FLAG TO INDICATE EOD REACHED ON I'TH MERGE DATA SET.
J = 1
IS THE FIRST CENTROID OF SUBNET J THE SAME ON THE FIRST AND I'TH DATA SETS
NO
PRINT ERROR MESSAGE
STOP 996
YES
GET MAXIMUM LAST CENTROID NUMBER FOR SUBNET J AND SAVE IN LSTMD(J)
J = J + 1
IS J > NOSUB?
NO
YES
I = I + 1
IS I > THE NUMBER OF DATA SETS TO MERGE
YES
NO
MR 1

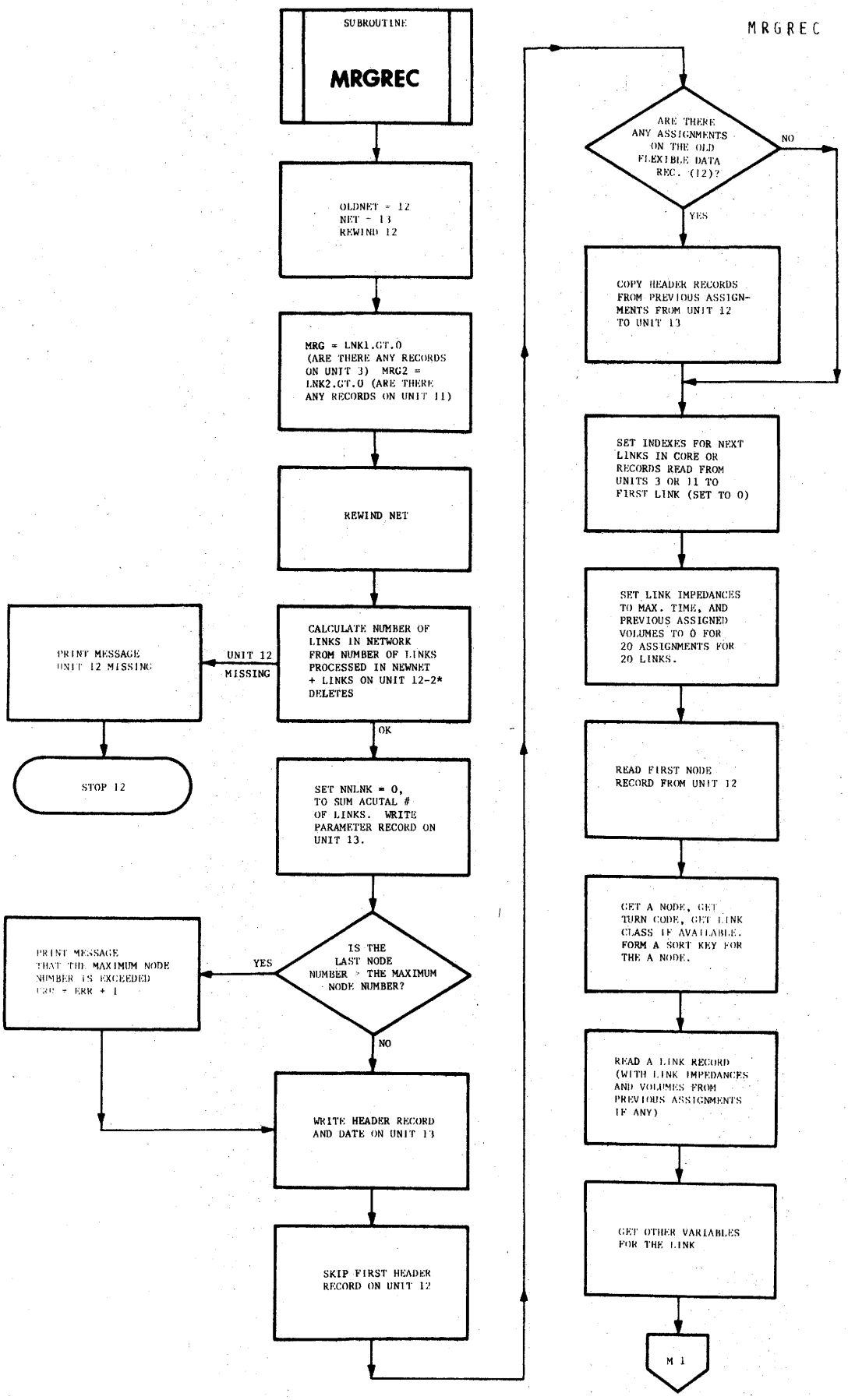

MR 1
REWIND MRGOUT DATA SET. WRITE THE PARAMETER RECORD FOR THE MRGOUT DATA SET.
GET THE LARGEST NUMBER OF CENTROIDS IN A SUBNET AND PUT IN MAX.
ISUB = 1
IFST = FIRST CENTROID FOR SUBNET ISUB. LST = LAST CENTROID FOR SUBNET ISUB
MR 2
NODE = IFST
LSUB = 1
MR 6
INITIALIZE VOLUME SUMMING ARRAY TO ZERO FOR MAX WORDS.
I = 1
MR 3
COMPARE CENTROID NUMBER READ LAST FROM DATA SET I: NODE
>
=
<
HAS AN EOD BEEN REACHED ON MERGE DATA SET I?
YES
NO
READ ANOTHER RECORD FROM MERGE DATA SET I.
EOD
MR 4
MR 3
IS THE SUBNET NUMBER OF THE RECORD FROM DATA SET I = LSUB
SUM THE VOLUMES IN THIS RECORD IN THE VOLUME SUMMING ARRAY
I = I + 1
MR 7
IS I > THE NUMBER OF DATA SETS TO MERGE
NO
YES
PACK THE SUMMED VOLUMES AND WRITE THEM IN ONE OR MORE RECORDS ON THE MRGOUT DATA SET
LSUB = LSUB + 1
IS LSUB > NOSUB?
YES
NO
MR 5

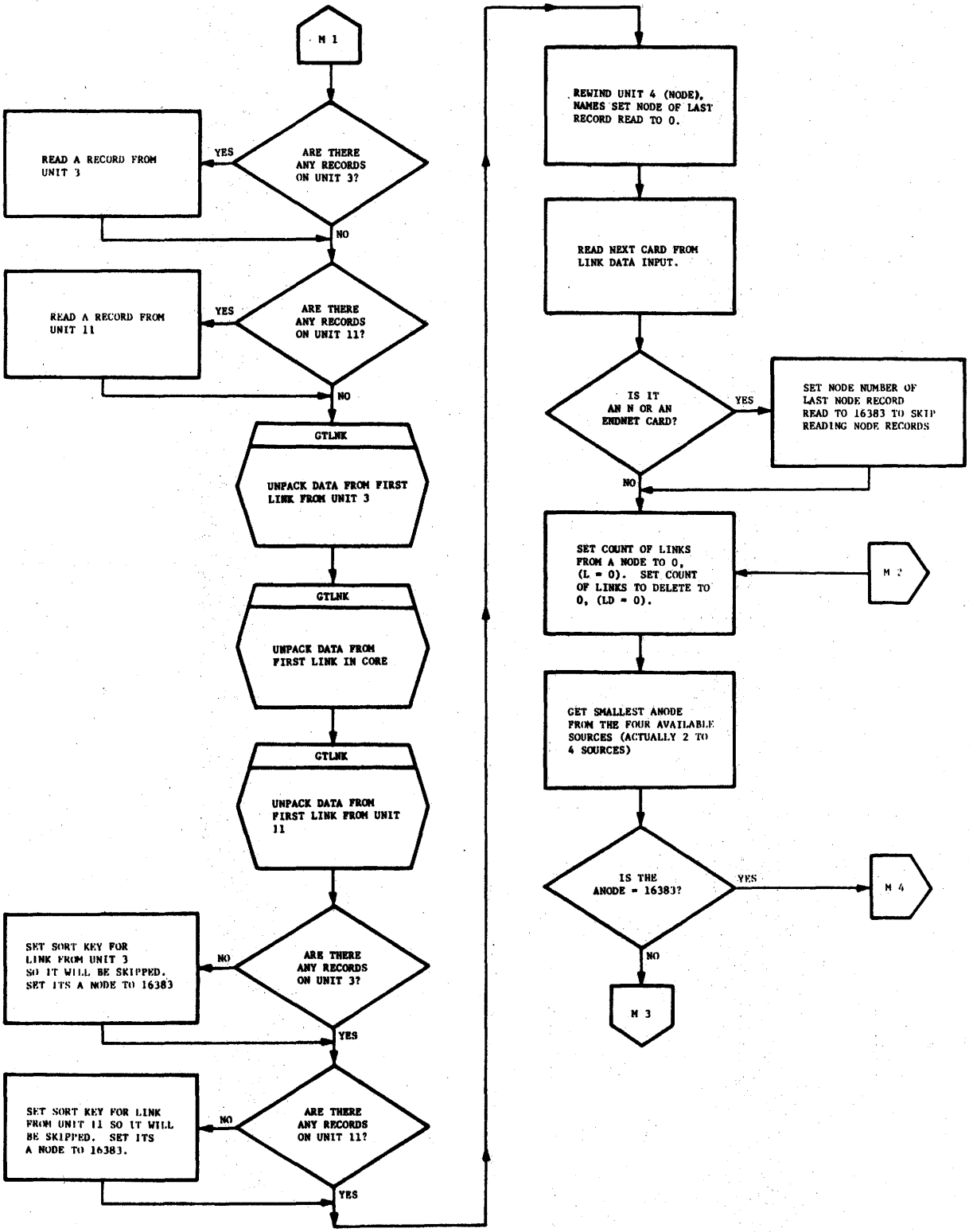

MR 5
NODE = NODE + 1
IS
NODE > LST
NO
MR 6
YES
ISUB = ISUB + 1
IS
ISUB > NOSUB?
NO
MR 2
YES
END FILE MRGOUT
REWIND MRGOUT
REWIND ALL MERGE
DATA SETS.
RETURN
MR 4
SET A FLAG TO INDICATE
THAN AN EOD HAS BEEN
REACHED ON MERGE DATA
SET I
MR 7

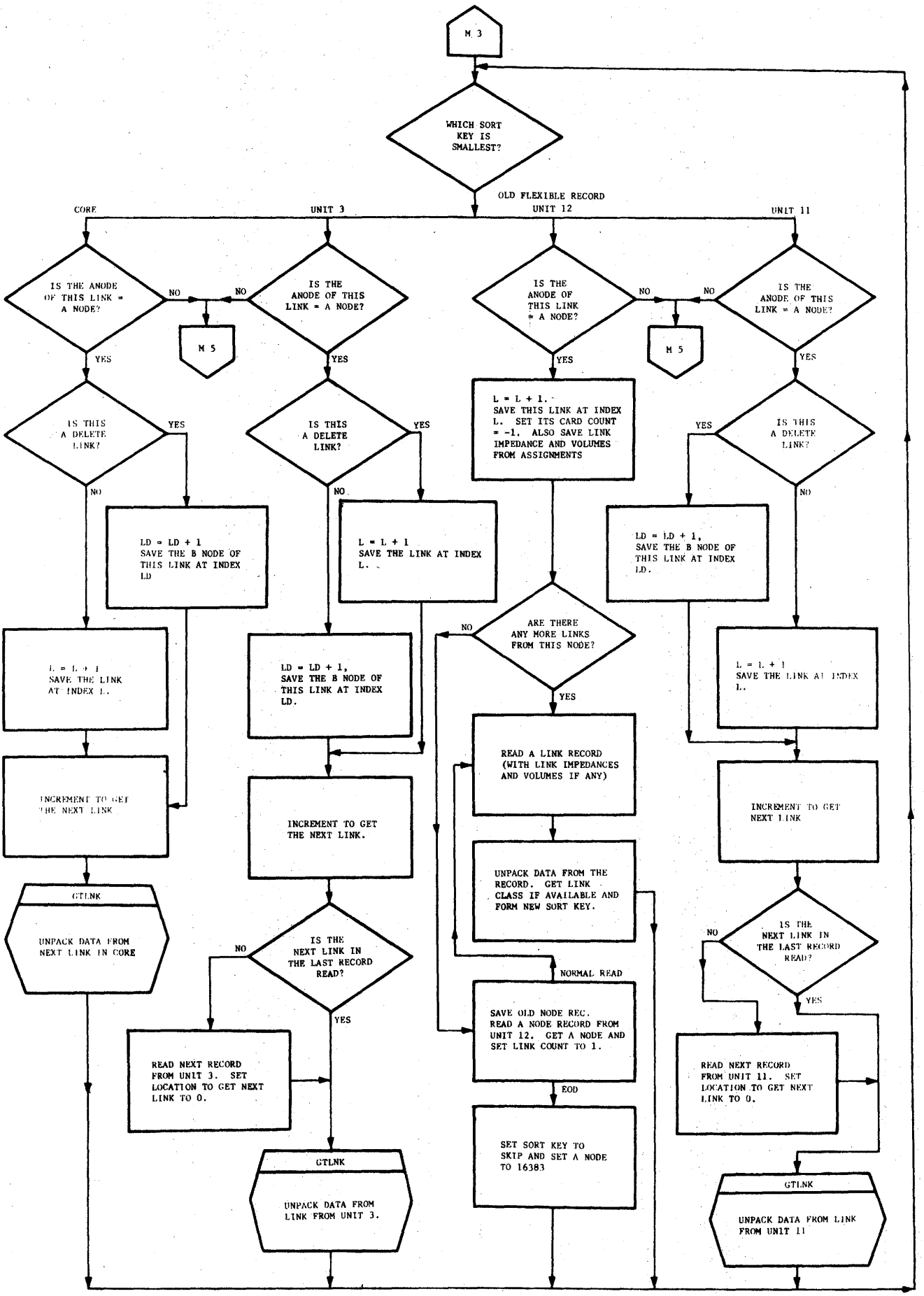


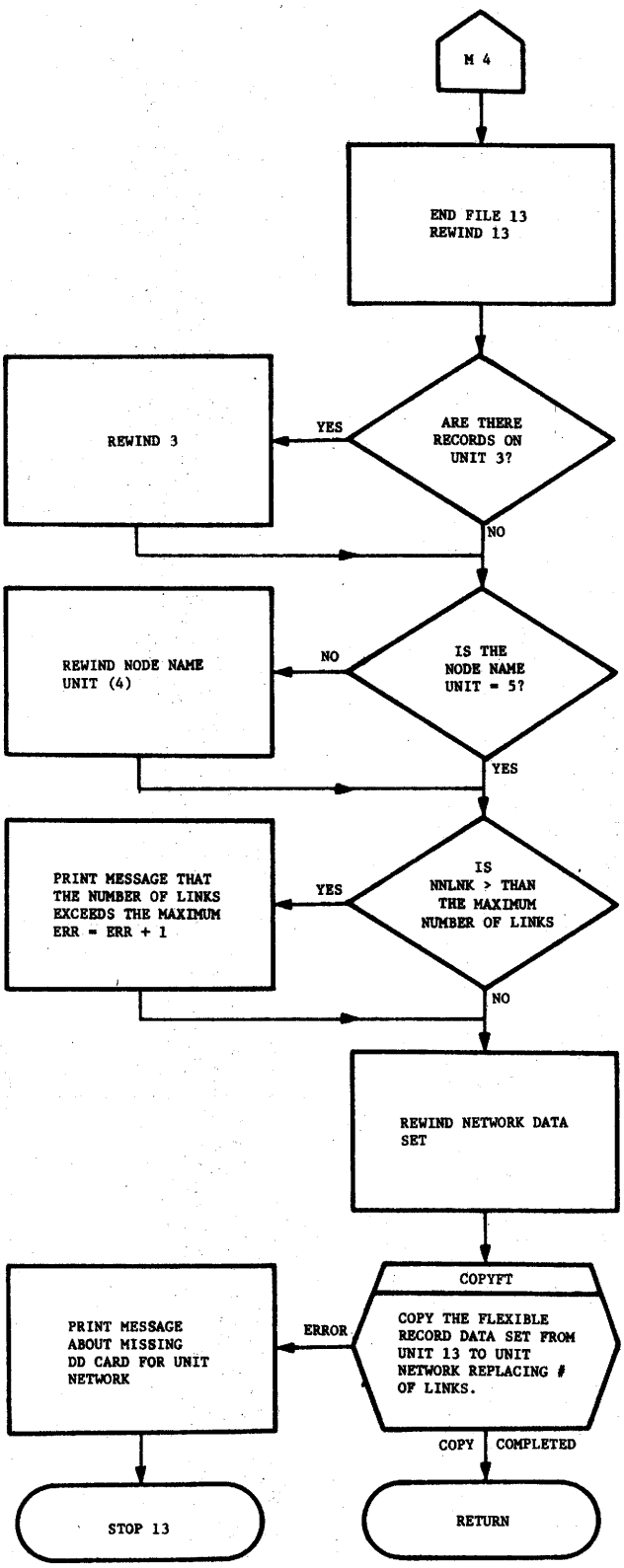
*ASSEMBLY LANGUAGE

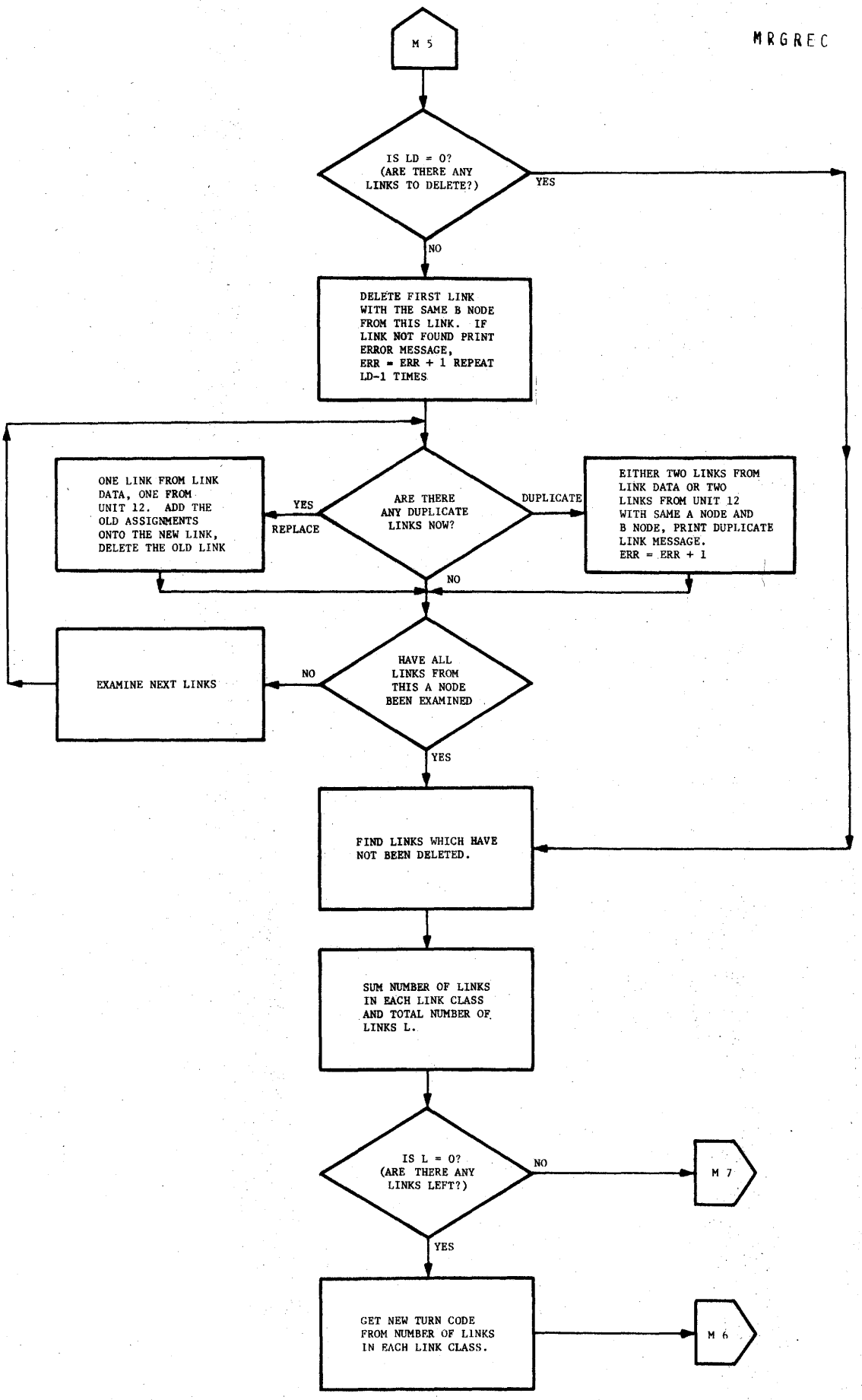

M 1
M 4
SAVE BACK NODE TO NODE I.
GET INDEX TO FIRST LINK FROM NODE I, J.
M 3
GET LINK INDEXED BY J. SEPARATE FIELDS OF THE LINK. ALSO GET B NODE OF THE LINK.
M 2
YES
IF THIS LINK IS USED WILL IT MAKE A U-TURN
NO
GET CUMULATIVE TIME TO NODE I AND ADD LINK TIME TO IT.
M 2
NO
IS THIS A SHORTER TIME TO THE B NODE?
YES
IS THIS THE FIRST PATH TO THE B NODE?
YES
NO
REMOVE THE B NODE FROM ITS OLD LOCATION IN THE SEQUENCE TABLE
PUT B NODE IN SEQUENCE TABLE
DELETE NODE FROM OLD TIME CHAIN AND ADD TO NEW TIME CHAIN
M 2
IS J THE LAST LINK FROM I?
NO
J = J + 1
M 3
YES
GET NEXT NODE FROM THE SEQUENCE TABLE, I.
WAS THE SEQUENCE TABLE EMPTY?
NO
M 4
YES
RESTORE REGISTERS
RETURN

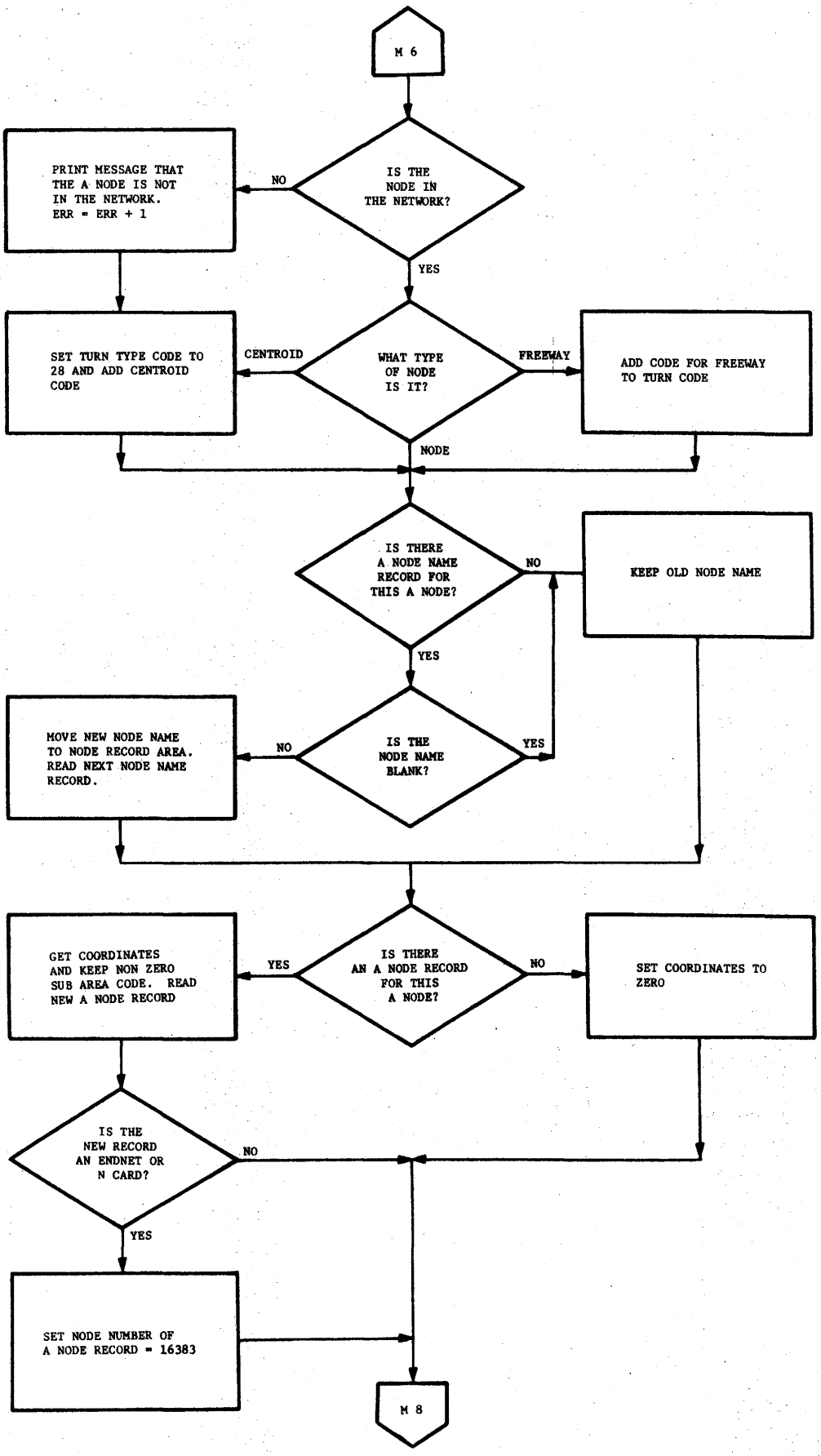

SUBROUTINE
MRGREC
OLDNET = 12
NET = 13
REWIND 12
MRG = LNK1.GT.0 (ARE THERE ANY RECORDS ON UNIT 3) MRG2 = LNK2.GT.0 (ARE THERE ANY RECORDS ON UNIT 11)
REWIND NET
CALCULATE NUMBER OF LINKS IN NETWORK FROM NUMBER OF LINKS PROCESSED IN NEWNET + LINKS ON UNIT 12-2* DELETES
UNIT 12 MISSING
PRINT MESSAGE UNIT 12 MISSING
STOP 12
OK
SET NNLNK = 0, TO SUM ACUTAL # OF LINKS. WRITE PARAMETER RECORD ON UNIT 13.
IS THE LAST NODE NUMBER > THE MAXIMUM NODE NUMBER?
YES
PRINT MESSAGE THAT THE MAXIMUM NODE NUMBER IS EXCEEDED ERR = ERR + 1
NO
WRITE HEADER RECORD AND DATE ON UNIT 13
SKIP FIRST HEADER RECORD ON UNIT 12
ARE THERE ANY ASSIGNMENTS ON THE OLD FLEXIBLE DATA REC. (12)?
NO
YES
COPY HEADER RECORDS FROM PREVIOUS ASSIGN-MENTS FROM UNIT 12 TO UNIT 13
SET INDEXES FOR NEXT LINKS IN CORE OR RECORDS READ FROM UNITS 3 OR 11 TO FIRST LINK (SET TO 0)
SET LINK IMPEDANCES TO MAX. TIME, AND PREVIOUS ASSIGNED VOLUMES TO 0 FOR 20 ASSIGNMENTS FOR 20 LINKS.
READ FIRST NODE RECORD FROM UNIT 12
GET A NODE, GET TURN CODE, GET LINK CLASS IF AVAILABLE. FORM A SORT KEY FOR THE A NODE.
READ A LINK RECORD (WITH LINK IMPEDANCES AND VOLUMES FROM PREVIOUS ASSIGNMENTS IF ANY)
GET OTHER VARIABLES FOR THE LINK
M 1


M 1
ARE THERE ANY RECORDS ON UNIT 3?
YES
READ A RECORD FROM UNIT 3
NO
ARE THERE ANY RECORDS ON UNIT 11?
YES
READ A RECORD FROM UNIT 11
NO
GTLNK
UNPACK DATA FROM FIRST LINK FROM UNIT 3
GTLNK
UNPACK DATA FROM FIRST LINK IN CORE
GTLNK
UNPACK DATA FROM FIRST LINK FROM UNIT 11
ARE THERE ANY RECORDS ON UNIT 3?
NO
SET SORT KEY FOR LINK FROM UNIT 3 SO IT WILL BE SKIPPED. SET ITS A NODE TO 16383
YES
ARE THERE ANY RECORDS ON UNIT 11?
NO
SET SORT KEY FOR LINK FROM UNIT 11 SO IT WILL BE SKIPPED. SET ITS A NODE TO 16383.
YES
REWIND UNIT 4 (NODE), NAMES SET NODE OF LAST RECORD READ TO 0.
READ NEXT CARD FROM LINK DATA INPUT.
IS IT AN N OR AN ENDNET CARD?
YES
SET NODE NUMBER OF LAST NODE RECORD READ TO 16383 TO SKIP READING NODE RECORDS
NO
SET COUNT OF LINKS FROM A NODE TO 0, (L = 0). SET COUNT OF LINKS TO DELETE TO 0, (LD = 0).
M 2
GET SMALLEST ANODE FROM THE FOUR AVAILABLE SOURCES (ACTUALLY 2 TO 4 SOURCES)
IS THE ANODE = 16383?
YES
M 4
NO
M 3

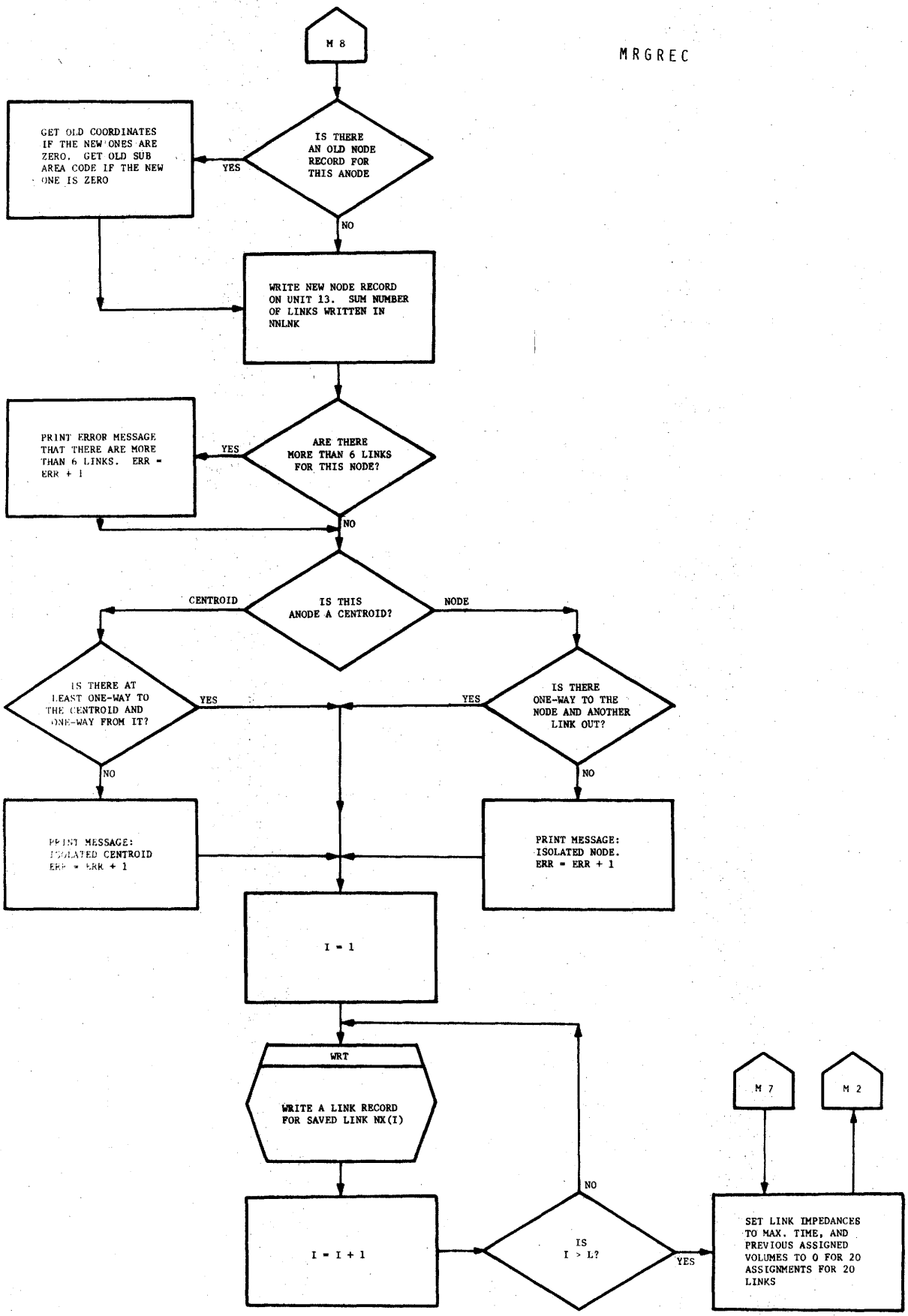

M 3
WHICH SORT KEY IS SMALLEST?
OLD FLEXIBLE RECORD
CORE
UNIT 3
UNIT 12
UNIT 11
IS THE ANODE OF THIS LINK = A NODE?
NO
M 5
YES
IS THIS A DELETE LINK?
LD = LD + 1 SAVE THE B NODE OF THIS LINK AT INDEX LD
L = L + 1 SAVE THE LINK AT INDEX L.
L = L + 1, SAVE THE B NODE OF THIS LINK AT INDEX LD.
L = L + 1 SAVE THIS LINK AT INDEX L. SET ITS CARD COUNT = -1. ALSO SAVE LINK IMPEDANCE AND VOLUMES FROM ASSIGNMENTS
ARE THERE ANY MORE LINKS FROM THIS NODE?
READ A LINK RECORD (WITH LINK IMPEDANCES AND VOLUMES IF ANY)
UNPACK DATA FROM THE RECORD. GET LINK CLASS IF AVAILABLE AND FORM NEW SORT KEY.
NORMAL READ
SAVE OLD NODE REC. READ A NODE RECORD FROM UNIT 12. GET A NODE AND SET LINK COUNT TO 1.
EOD
SET SORT KEY TO SKIP AND SET A NODE TO 16383
L = L + 1 SAVE THE LINK AT INDEX L.
INCREMENT TO GET THE NEXT LINK
GTLNK
UNPACK DATA FROM NEXT LINK IN CORE
IS THE NEXT LINK IN THE LAST RECORD READ?
READ NEXT RECORD FROM UNIT 3. SET LOCATION TO GET NEXT LINK TO 0.
UNPACK DATA FROM LINK FROM UNIT 3.
READ NEXT RECORD FROM UNIT 11. SET LOCATION TO GET NEXT LINK TO 0.
UNPACK DATA FROM LINK FROM UNIT 11

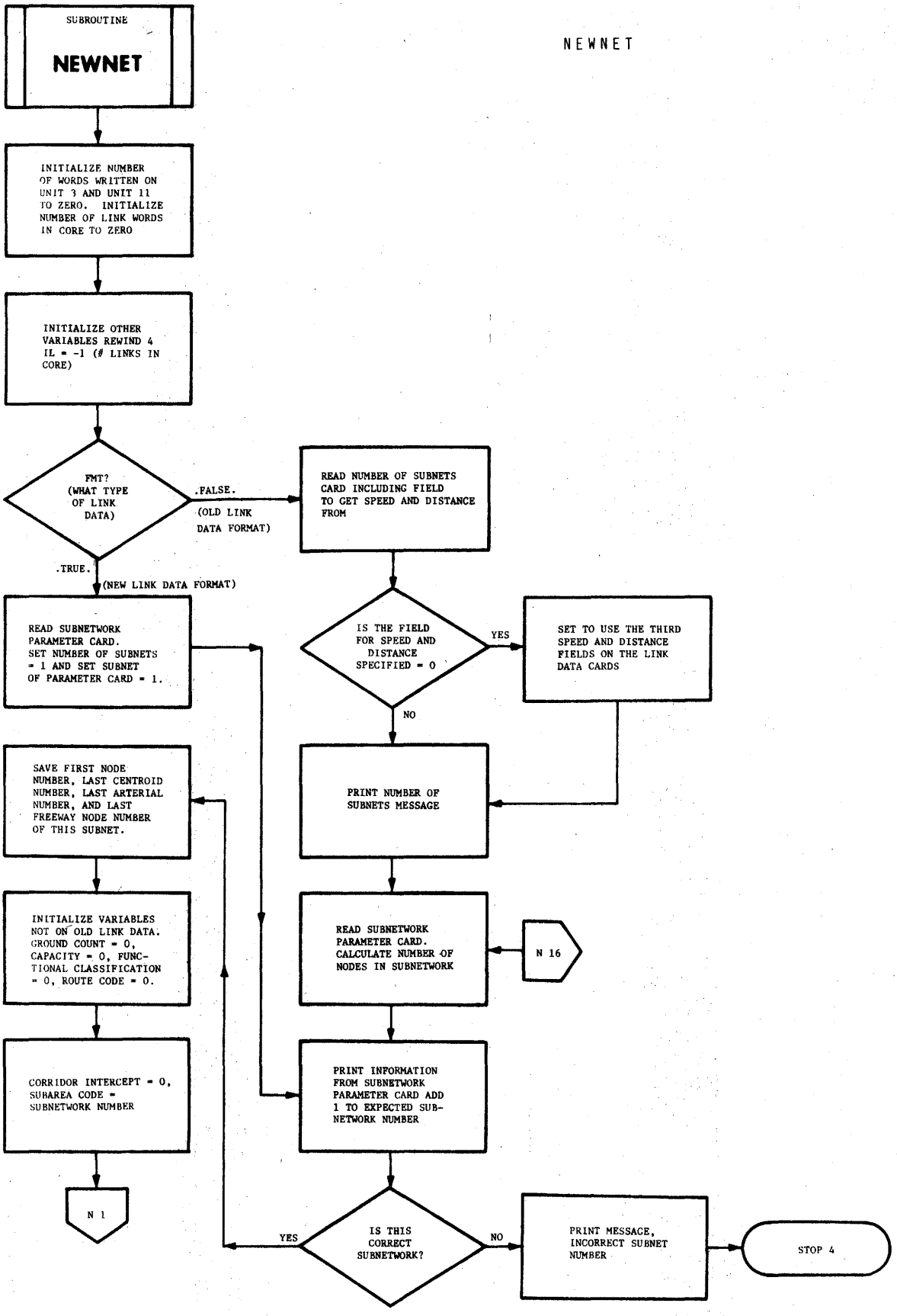

M 4
END FILE 13
REWIND 13
ARE THERE RECORDS ON UNIT 3?
YES
REWIND 3
NO
IS THE NODE NAME UNIT = 5?
NO
REWIND NODE NAME UNIT (4)
YES
IS NNLNK > THAN THE MAXIMUM NUMBER OF LINKS
YES
PRINT MESSAGE THAT THE NUMBER OF LINKS EXCEEDS THE MAXIMUM ERR = ERR + 1
NO
REWIND NETWORK DATA SET
COPYFT
COPY THE FLEXIBLE RECORD DATA SET FROM UNIT 13 TO UNIT NETWORK REPLACING # OF LINKS.
ERROR
PRINT MESSAGE ABOUT MISSING DD CARD FOR UNIT NETWORK
COPY COMPLETED
STOP 13
RETURN

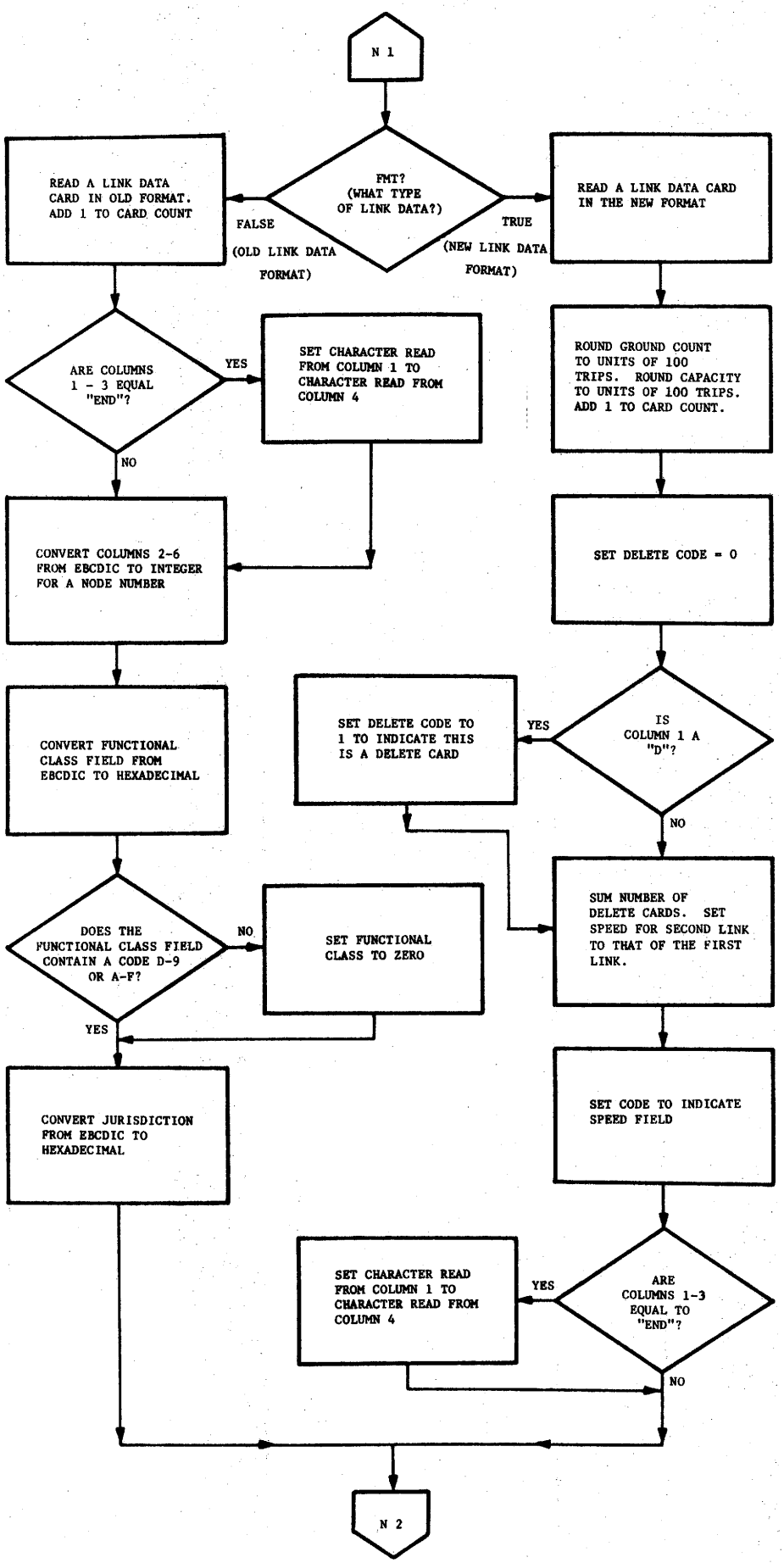

M 5
IS LD = 0?
(ARE THERE ANY
LINKS TO DELETE?)
YES
NO
DELETE FIRST LINK
WITH THE SAME B NODE
FROM THIS LINK. IF
LINK NOT FOUND PRINT
ERROR MESSAGE,
ERR = ERR + 1 REPEAT
LD-1 TIMES
ONE LINK FROM LINK
DATA, ONE FROM
UNIT 12. ADD THE
OLD ASSIGNMENTS
ONTO THE NEW LINK,
DELETE THE OLD LINK
YES
REPLACE
ARE THERE
ANY DUPLICATE
LINKS NOW?
DUPLICATE
EITHER TWO LINKS FROM
LINK DATA OR TWO
LINKS FROM UNIT 12
WITH SAME A NODE AND
B NODE, PRINT DUPLICATE
LINK MESSAGE.
ERR = ERR + 1
NO
EXAMINE NEXT LINKS
NO
HAVE ALL
LINKS FROM
THIS A NODE
BEEN EXAMINED
YES
FIND LINKS WHICH HAVE
NOT BEEN DELETED.
SUM NUMBER OF LINKS
IN EACH LINK CLASS
AND TOTAL NUMBER OF
LINKS L.
IS L = 0?
(ARE THERE ANY
LINKS LEFT?)
NO
M 7
YES
GET NEW TURN CODE
FROM NUMBER OF LINKS
IN EACH LINK CLASS.
M 6

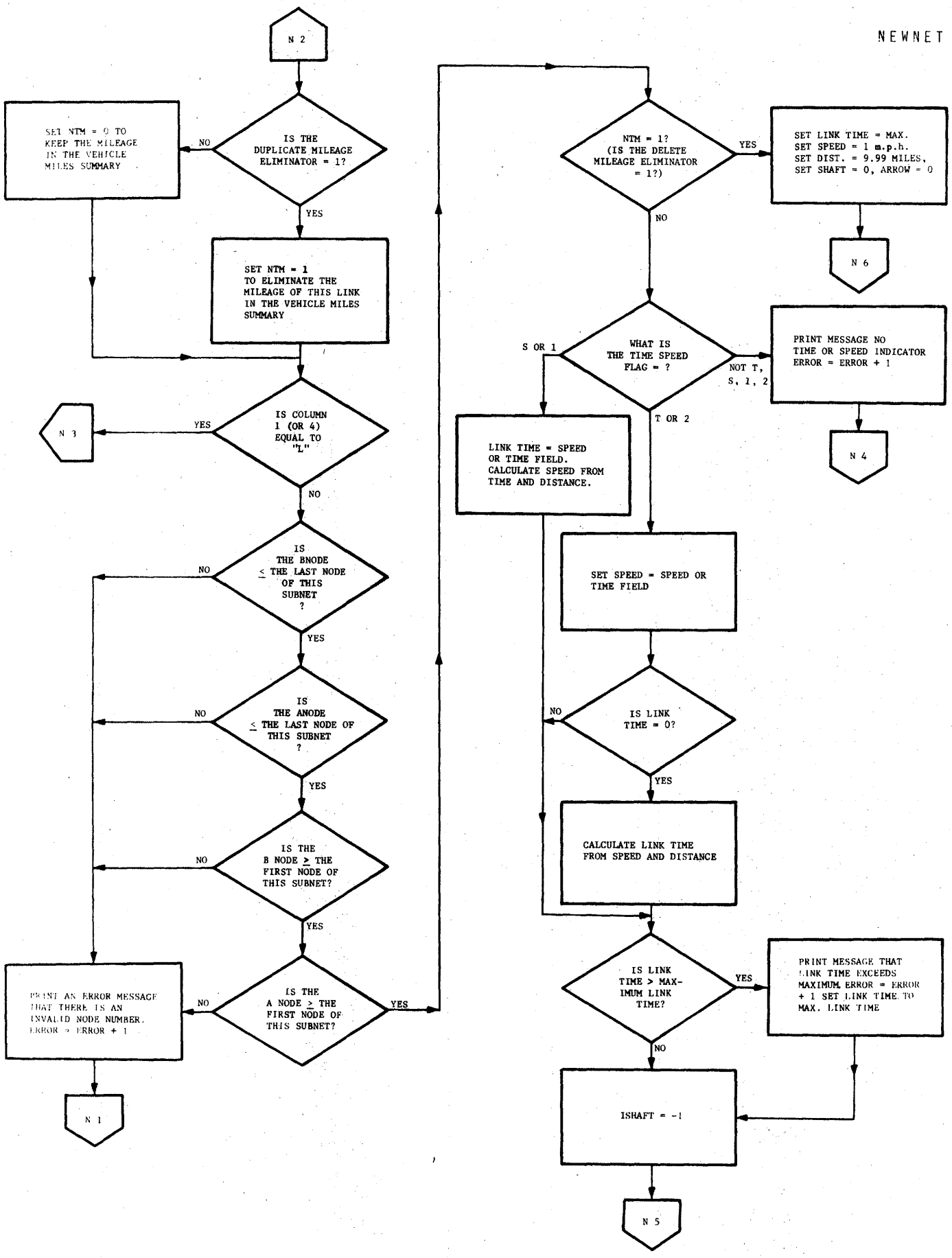

M 6
IS THE NODE IN THE NETWORK?
NO
PRINT MESSAGE THAT THE A NODE IS NOT IN THE NETWORK. ERR = ERR + 1
YES
WHAT TYPE OF NODE IS IT?
CENTROID
SET TURN TYPE CODE TO 28 AND ADD CENTROID CODE
FREEWAY
ADD CODE FOR FREEWAY TO TURN CODE
NODE
IS THERE A NODE NAME RECORD FOR THIS A NODE?
NO
KEEP OLD NODE NAME
YES
IS THE NODE NAME BLANK?
NO
MOVE NEW NODE NAME TO NODE RECORD AREA. READ NEXT NODE NAME RECORD.
YES
IS THERE AN A NODE RECORD FOR THIS A NODE?
YES
GET COORDINATES AND KEEP NON ZERO SUB AREA CODE. READ NEW A NODE RECORD
NO
SET COORDINATES TO ZERO
IS THE NEW RECORD AN ENDNET OR N CARD?
NO
YES
SET NODE NUMBER OF A NODE RECORD = 16383
M 8

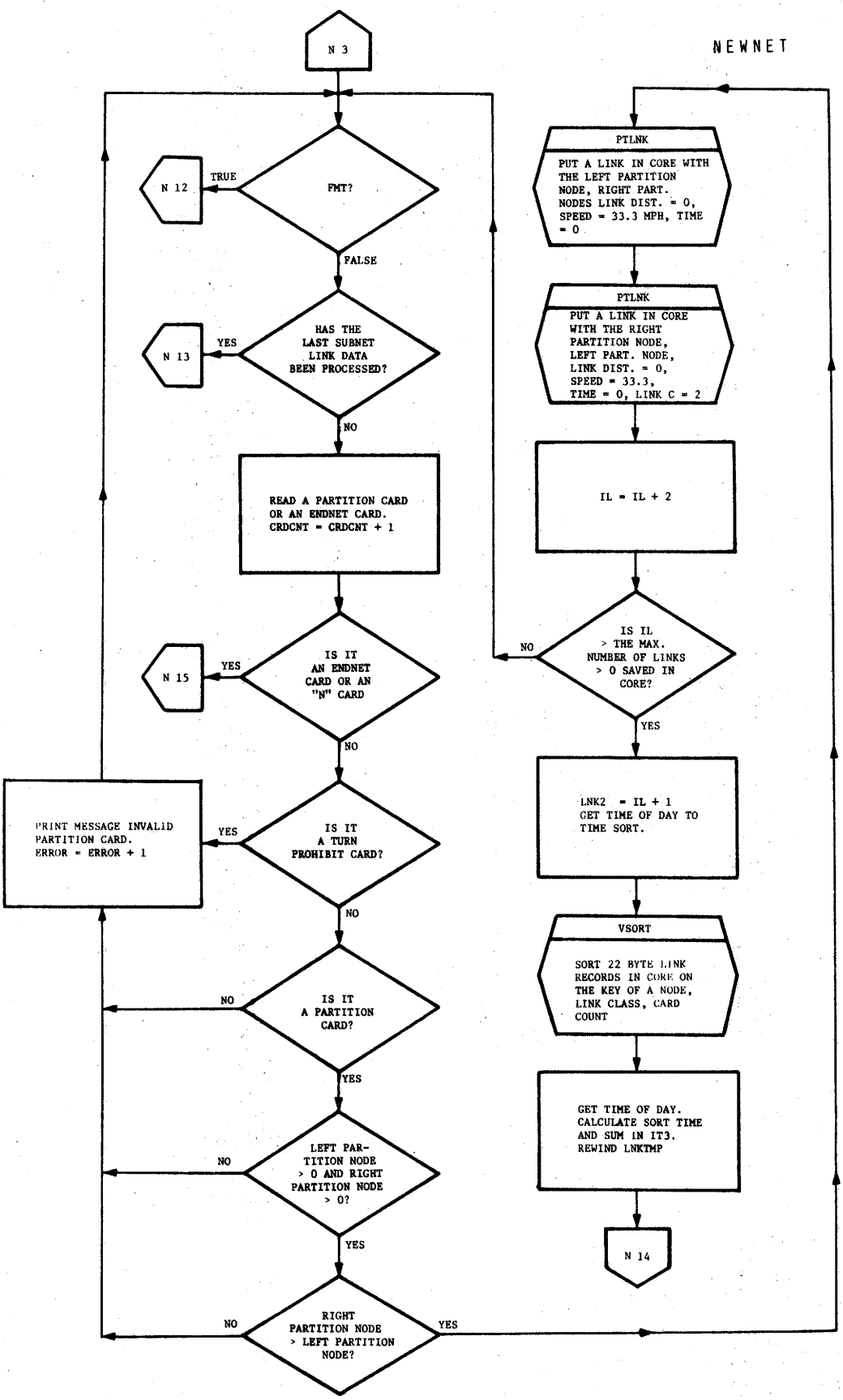
MRGREC

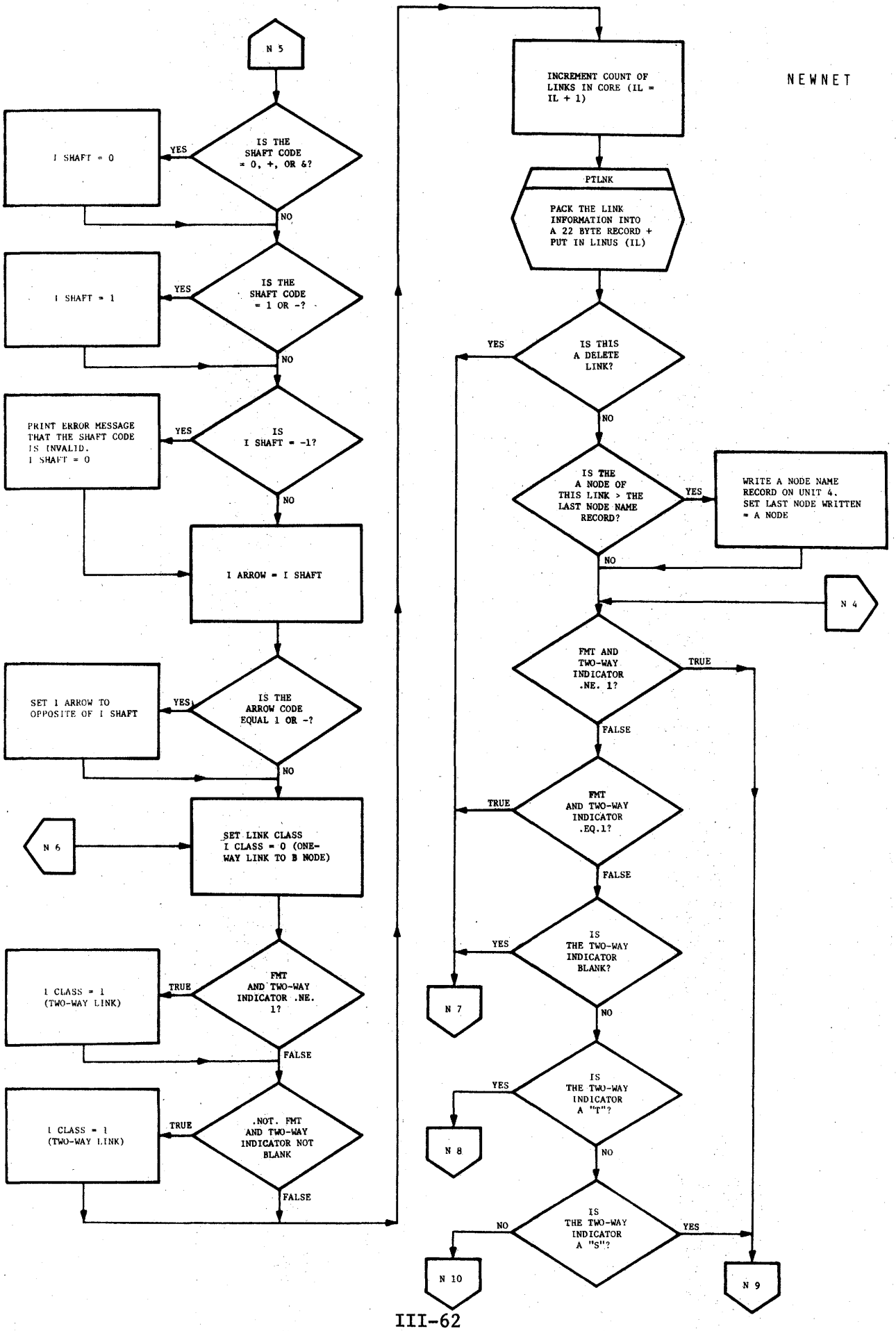

M 8
IS THERE AN OLD NODE RECORD FOR THIS ANODE
YES
GET OLD COORDINATES IF THE NEW ONES ARE ZERO. GET OLD SUB AREA CODE IF THE NEW ONE IS ZERO
NO
WRITE NEW NODE RECORD ON UNIT 13. SUM NUMBER OF LINKS WRITTEN IN NNLNK
ARE THERE MORE THAN 6 LINKS FOR THIS NODE?
YES
PRINT ERROR MESSAGE THAT THERE ARE MORE THAN 6 LINKS. ERR = ERR + 1
NO
IS THIS ANODE A CENTROID?
CENTROID
NODE
IS THERE AT LEAST ONE-WAY TO THE CENTROID AND ONE-WAY FROM IT?
YES
NO
PRINT MESSAGE: ISOLATED CENTROID ERR = ERR + 1
IS THERE ONE-WAY TO THE NODE AND ANOTHER LINK OUT?
YES
NO
PRINT MESSAGE: ISOLATED NODE. ERR = ERR + 1
I = 1
WRT
WRITE A LINK RECORD FOR SAVED LINK NX(I)
I = I + 1
IS I > L?
NO
YES
M 7
M 2
SET LINK IMPEDANCES TO MAX. TIME, AND PREVIOUS ASSIGNED VOLUMES TO 0 FOR 20 ASSIGNMENTS FOR 20 LINKS

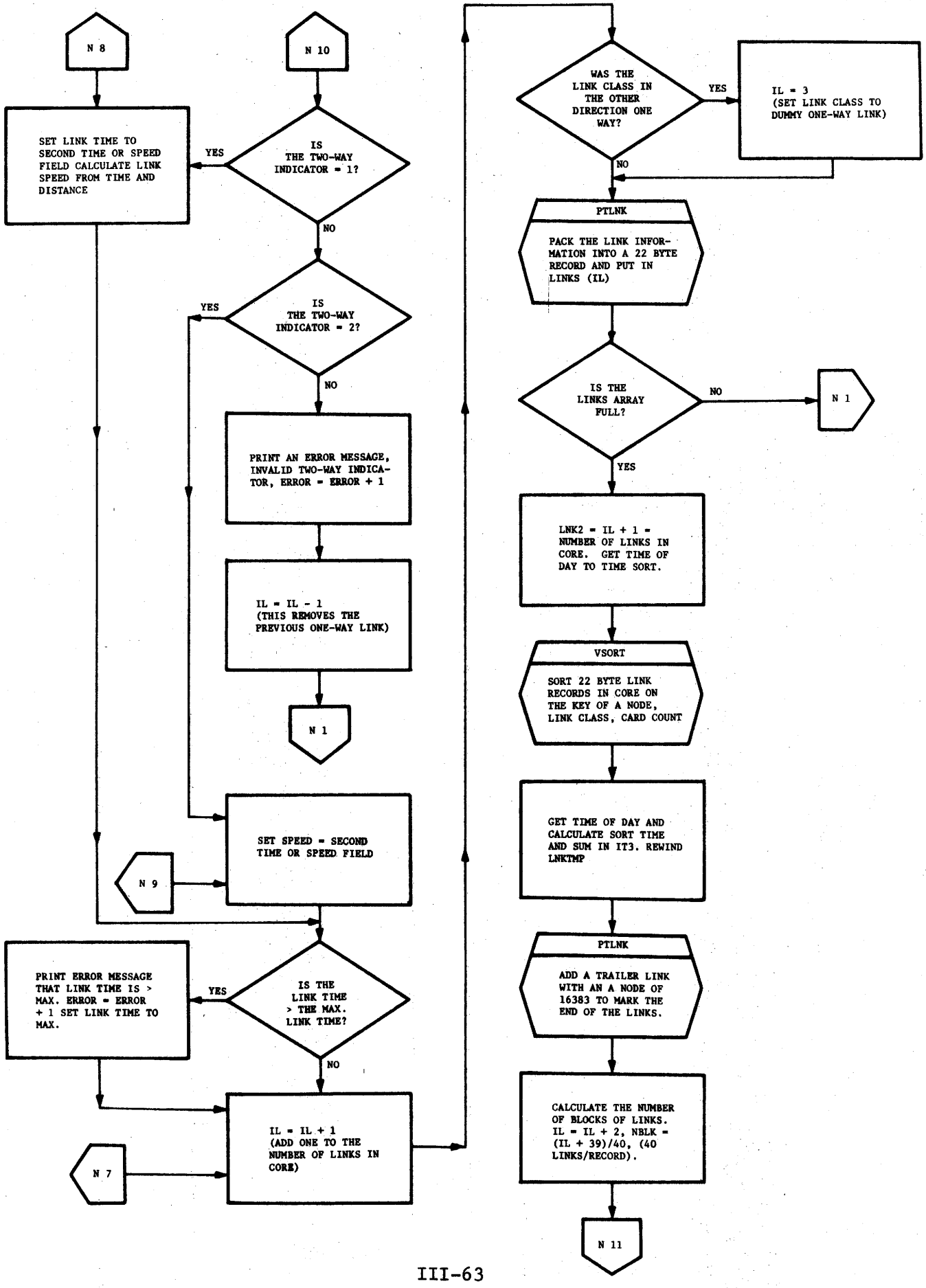

SUBROUTINE
NEWNET
INITIALIZE NUMBER OF WORDS WRITTEN ON UNIT 3 AND UNIT 11 TO ZERO. INITIALIZE NUMBER OF LINK WORDS IN CORE TO ZERO
INITIALIZE OTHER VARIABLES REWIND 4 IL = -1 (# LINKS IN CORE)
FMT? (WHAT TYPE OF LINK DATA)
.FALSE.
(OLD LINK DATA FORMAT)
READ NUMBER OF SUBNETS CARD INCLUDING FIELD TO GET SPEED AND DISTANCE FROM
.TRUE.
(NEW LINK DATA FORMAT)
READ SUBNETWORK PARAMETER CARD. SET NUMBER OF SUBNETS = 1 AND SET SUBNET OF PARAMETER CARD = 1.
IS THE FIELD FOR SPEED AND DISTANCE SPECIFIED = 0
YES
SET TO USE THE THIRD SPEED AND DISTANCE FIELDS ON THE LINK DATA CARDS
NO
SAVE FIRST NODE NUMBER, LAST CENTROID NUMBER, LAST ARTERIAL NUMBER, AND LAST FREEWAY NODE NUMBER OF THIS SUBNET.
PRINT NUMBER OF SUBNETS MESSAGE
INITIALIZE VARIABLES NOT ON OLD LINK DATA. GROUND COUNT = 0, CAPACITY = 0, FUNCTIONAL CLASSIFICATION = 0, ROUTE CODE = 0.
READ SUBNETWORK PARAMETER CARD. CALCULATE NUMBER OF NODES IN SUBNETWORK
N 16
CORRIDOR INTERCEPT = 0, SUBAREA CODE = SUBNETWORK NUMBER
PRINT INFORMATION FROM SUBNETWORK PARAMETER CARD ADD 1 TO EXPECTED SUBNETWORK NUMBER
N 1
YES
IS THIS CORRECT SUBNETWORK?
NO
PRINT MESSAGE, INCORRECT SUBNET NUMBER
STOP 4

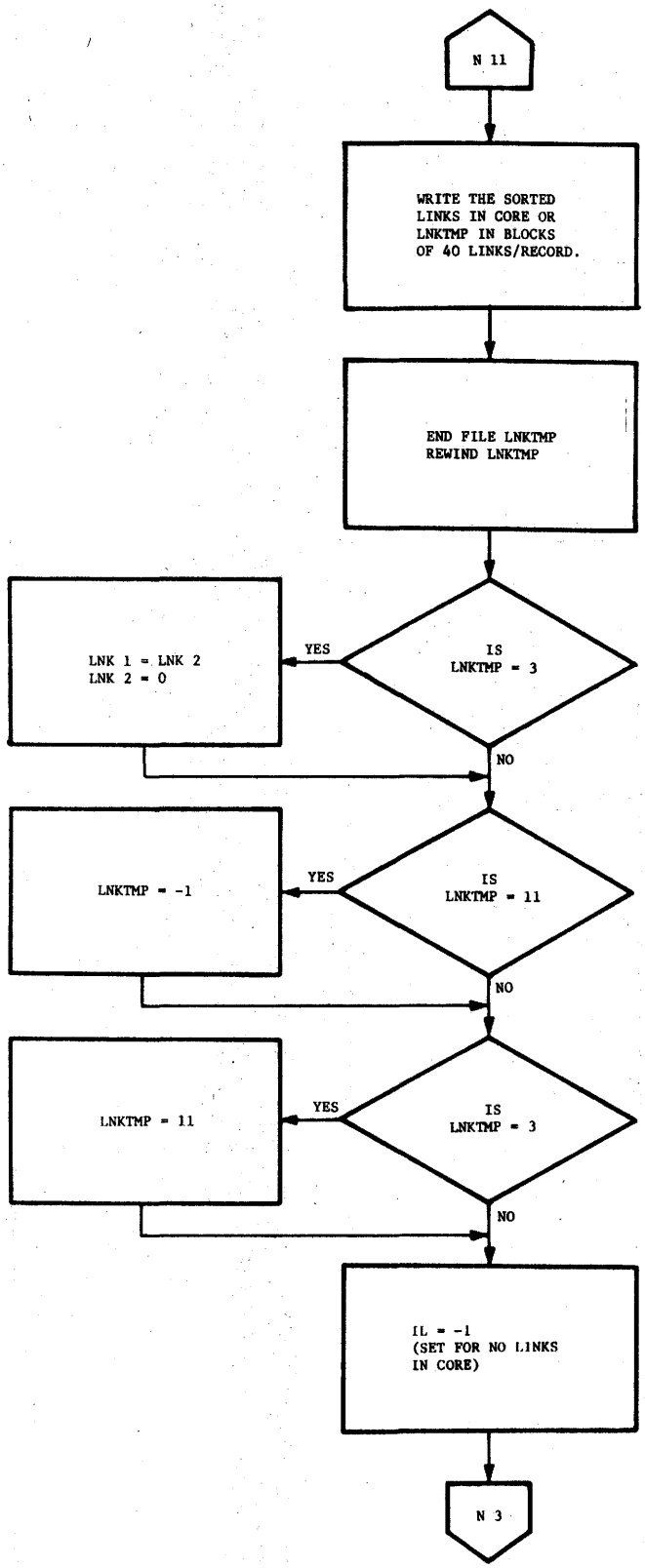

N 1
FMT?
(WHAT TYPE
OF LINK DATA?)
FALSE
(OLD LINK DATA
FORMAT)
TRUE
(NEW LINK DATA
FORMAT)
READ A LINK DATA
CARD IN OLD FORMAT.
ADD 1 TO CARD COUNT
READ A LINK DATA CARD
IN THE NEW FORMAT
ARE COLUMNS
1 - 3 EQUAL
"END"?
YES
SET CHARACTER READ
FROM COLUMN 1 TO
CHARACTER READ FROM
COLUMN 4
NO
ROUND GROUND COUNT
TO UNITS OF 100
TRIPS. ROUND CAPACITY
TO UNITS OF 100 TRIPS.
ADD 1 TO CARD COUNT.
CONVERT COLUMNS 2-6
FROM EBCDIC TO INTEGER
FOR A NODE NUMBER
SET DELETE CODE = 0
CONVERT FUNCTIONAL
CLASS FIELD FROM
EBCDIC TO HEXADECIMAL
SET DELETE CODE TO
1 TO INDICATE THIS
IS A DELETE CARD
YES
IS
COLUMN 1 A
"D"?
NO
DOES THE
FUNCTIONAL CLASS FIELD
CONTAIN A CODE D-9
OR A-F?
NO
SET FUNCTIONAL
CLASS TO ZERO
SUM NUMBER OF
DELETE CARDS. SET
SPEED FOR SECOND LINK
TO THAT OF THE FIRST
LINK.
YES
CONVERT JURISDICTION
FROM EBCDIC TO
HEXADECIMAL
SET CODE TO INDICATE
SPEED FIELD
SET CHARACTER READ
FROM COLUMN 1 TO
CHARACTER READ FROM
COLUMN 4
YES
ARE
COLUMNS 1-3
EQUAL TO
"END"?
NO
N 2

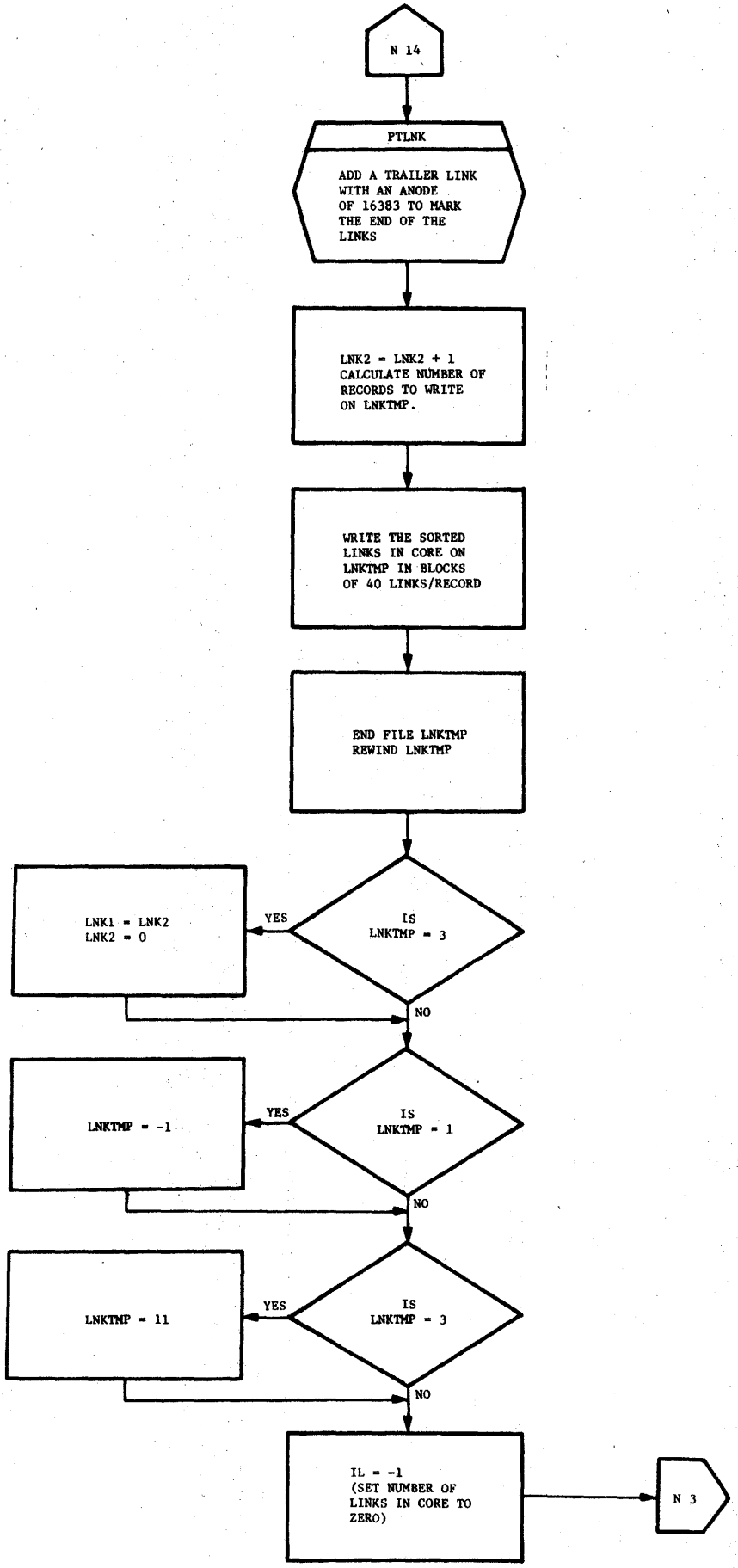

N 2
IS THE DUPLICATE MILEAGE ELIMINATOR = 1?
NO
YES
SET NTM = 0 TO KEEP THE MILEAGE IN THE VEHICLE MILES SUMMARY
SET NTM = 1 TO ELIMINATE THE MILEAGE OF THIS LINK IN THE VEHICLE MILES SUMMARY
IS COLUMN 1 (OR 4) EQUAL TO "L"
YES
N 3
NO
IS THE BNODE ≤ THE LAST NODE OF THIS SUBNET ?
NO
YES
IS THE ANODE ≤ THE LAST NODE OF THIS SUBNET ?
NO
YES
IS THE B NODE ≥ THE FIRST NODE OF THIS SUBNET?
NO
YES
IS THE A NODE ≥ THE FIRST NODE OF THIS SUBNET?
NO
YES
PRINT AN ERROR MESSAGE THAT THERE IS AN INVALID NODE NUMBER. ERROR = ERROR + 1
N 1
NTM = 1? (IS THE DELETE MILEAGE ELIMINATOR = 1?)
YES
NO
SET LINK TIME = MAX. SET SPEED = 1 m.p.h. SET DIST. = 9.99 MILES, SET SHAFT = 0, ARROW = 0
N 6
WHAT IS THE TIME SPEED FLAG = ?
S OR 1
T OR 2
NOT T, S, 1, 2
PRINT MESSAGE NO TIME OR SPEED INDICATOR ERROR = ERROR + 1
N 4
LINK TIME = SPEED OR TIME FIELD. CALCULATE SPEED FROM TIME AND DISTANCE.
SET SPEED = SPEED OR TIME FIELD
IS LINK TIME = 0?
NO
YES
CALCULATE LINK TIME FROM SPEED AND DISTANCE
IS LINK TIME > MAXIMUM LINK TIME?
YES
NO
PRINT MESSAGE THAT LINK TIME EXCEEDS MAXIMUM. ERROR = ERROR + 1 SET LINK TIME TO MAX. LINK TIME
ISHAFT = -1
N 5

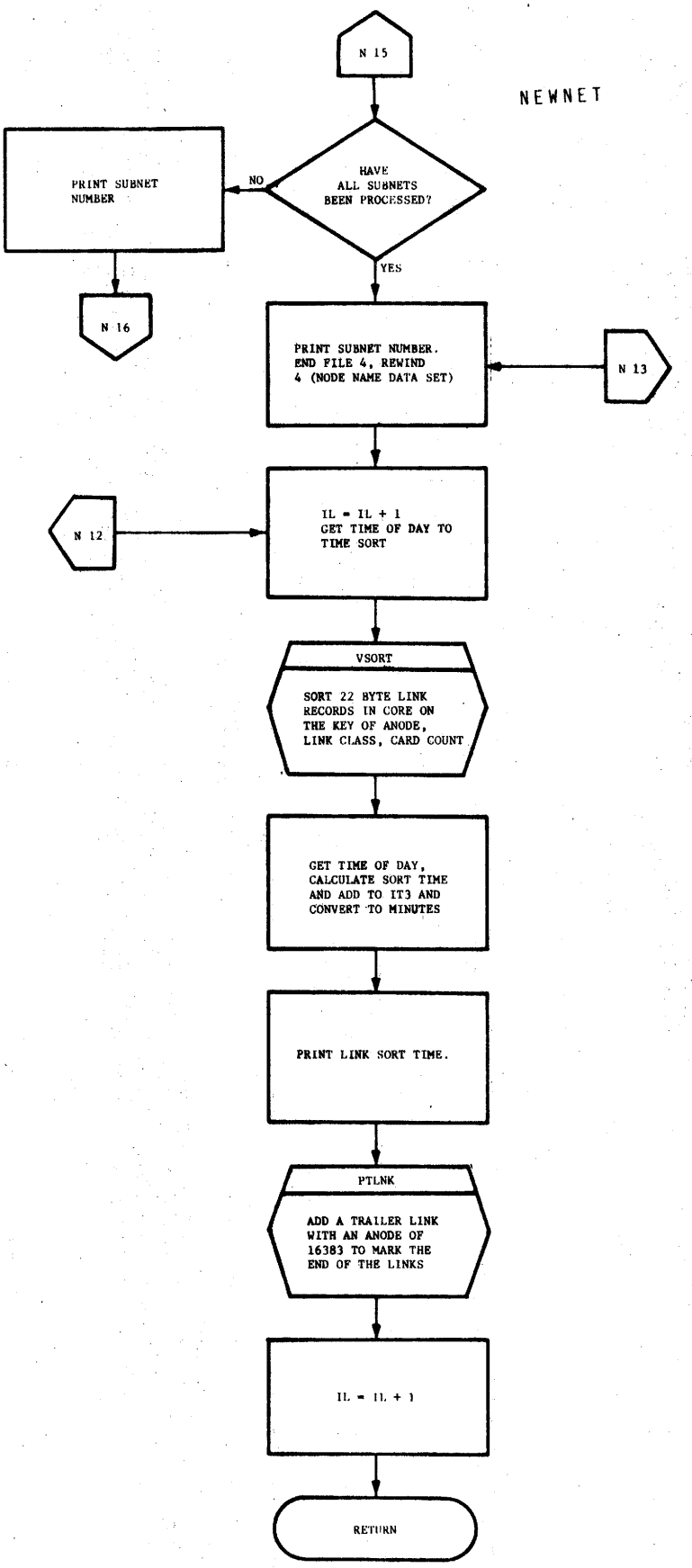

NEWNET
N 3
FMT?
TRUE
N 12
FALSE
HAS THE LAST SUBNET LINK DATA BEEN PROCESSED?
YES
N 13
NO
READ A PARTITION CARD OR AN ENDNET CARD. CRDCNT = CRDCNT + 1
IS IT AN ENDNET CARD OR AN "N" CARD
YES
N 15
NO
IS IT A TURN PROHIBIT CARD?
YES
PRINT MESSAGE INVALID PARTITION CARD. ERROR = ERROR + 1
NO
IS IT A PARTITION CARD?
NO
YES
LEFT PARTITION NODE > 0 AND RIGHT PARTITION NODE > 0?
NO
YES
RIGHT PARTITION NODE > LEFT PARTITION NODE?
NO
YES
PTLNK
PUT A LINK IN CORE WITH THE LEFT PARTITION NODE, RIGHT PART. NODES LINK DIST. = 0, SPEED = 33.3 MPH, TIME = 0
PTLNK
PUT A LINK IN CORE WITH THE RIGHT PARTITION NODE, LEFT PART. NODE, LINK DIST. = 0, SPEED = 33.3, TIME = 0, LINK C = 2
IL = IL + 2
IS IL > THE MAX. NUMBER OF LINKS > 0 SAVED IN CORE?
NO
YES
LNK2 = IL + 1
GET TIME OF DAY TO TIME SORT.
VSORT
SORT 22 BYTE LINK RECORDS IN CORE ON THE KEY OF A NODE, LINK CLASS, CARD COUNT
GET TIME OF DAY. CALCULATE SORT TIME AND SUM IN IT3. REWIND LNKTMP
N 14

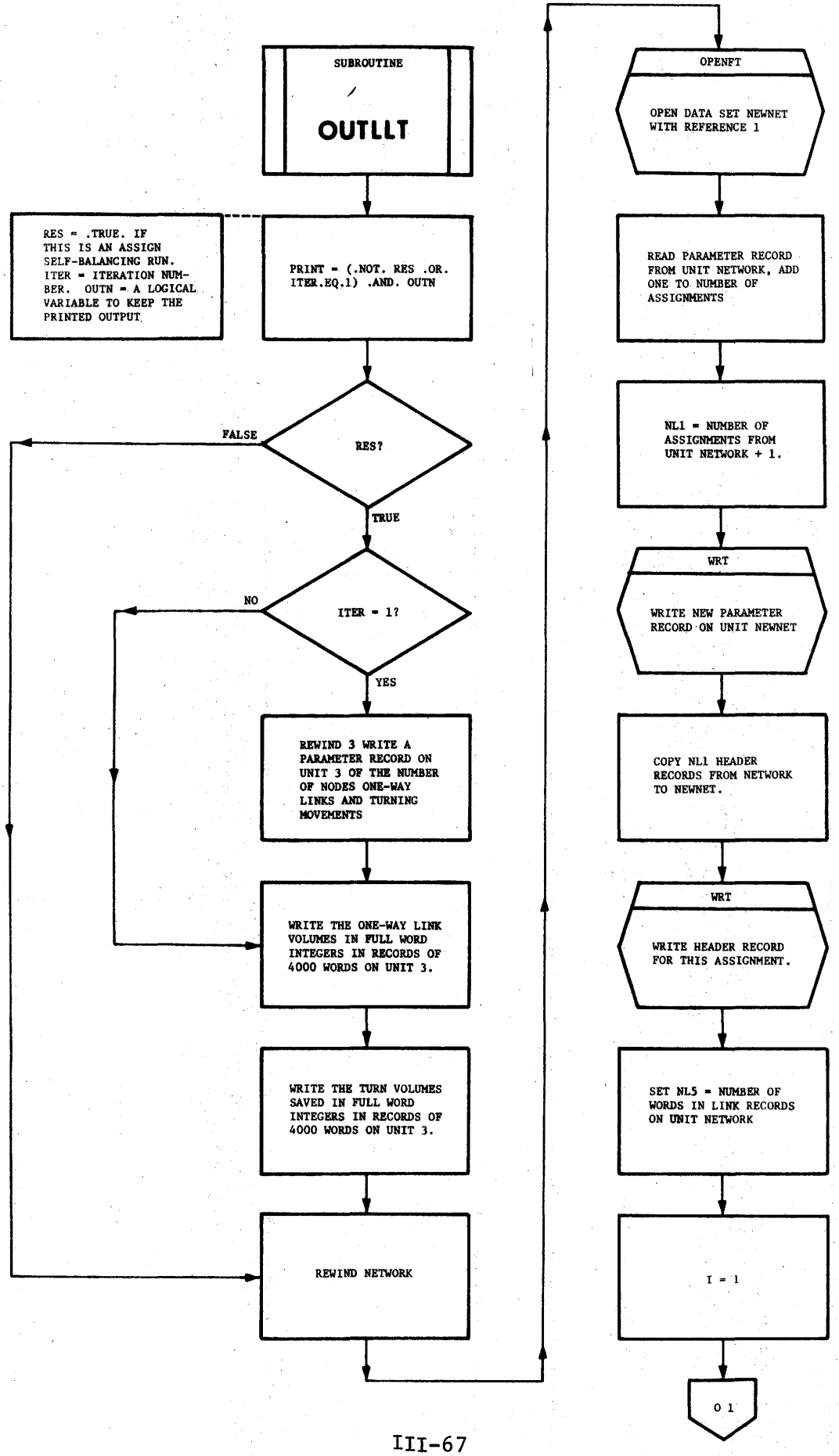

N 5
IS THE SHAFT CODE = 0, +, OR &?
YES
I SHAFT = 0
NO
IS THE SHAFT CODE = 1 OR -?
YES
I SHAFT = 1
NO
IS I SHAFT = -1?
YES
PRINT ERROR MESSAGE THAT THE SHAFT CODE IS INVALID. I SHAFT = 0
NO
I ARROW = I SHAFT
IS THE ARROW CODE EQUAL 1 OR -?
YES
SET I ARROW TO OPPOSITE OF I SHAFT
NO
N 6
SET LINK CLASS I CLASS = 0 (ONE-WAY LINK TO B NODE)
FMT AND TWO-WAY INDICATOR .NE. 1?
TRUE
I CLASS = 1 (TWO-WAY LINK)
FALSE
.NOT. FMT AND TWO-WAY INDICATOR NOT BLANK
TRUE
I CLASS = 1 (TWO-WAY LINK)
FALSE
INCREMENT COUNT OF LINKS IN CORE (IL = IL + 1)
PTLNK
PACK THE LINK INFORMATION INTO A 22 BYTE RECORD + PUT IN LINUS (IL)
IS THIS A DELETE LINK?
YES
NO
IS THE A NODE OF THIS LINK > THE LAST NODE NAME RECORD?
YES
WRITE A NODE NAME RECORD ON UNIT 4. SET LAST NODE WRITTEN = A NODE
NO
N 4
FMT AND TWO-WAY INDICATOR .NE. 1?
TRUE
FALSE
FMT AND TWO-WAY INDICATOR .EQ.1?
TRUE
FALSE
IS THE TWO-WAY INDICATOR BLANK?
YES
N 7
NO
IS THE TWO-WAY INDICATOR A "T"?
YES
N 8
NO
IS THE TWO-WAY INDICATOR A "S"?
NO
N 10
YES
N 9

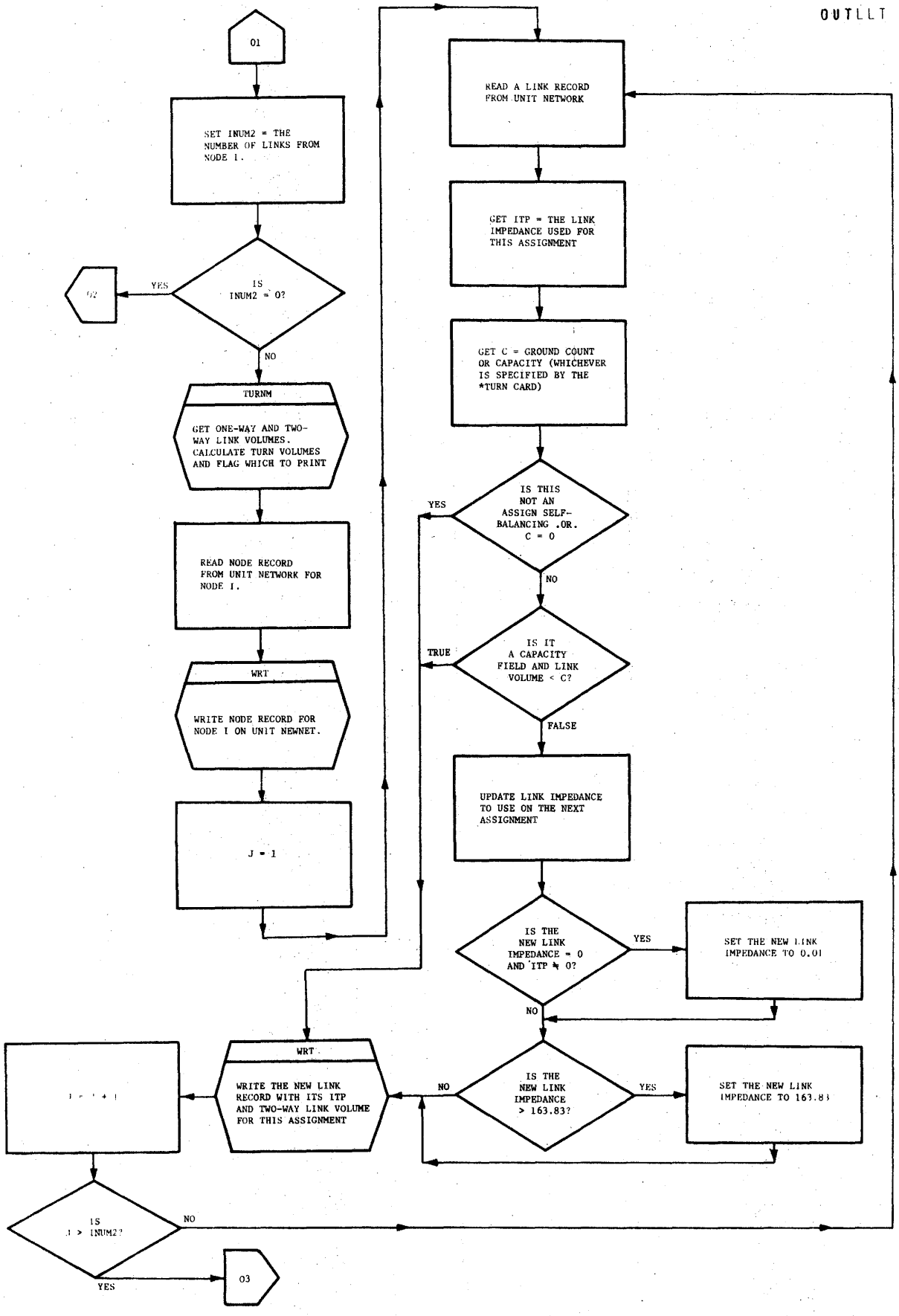

N 8
N 10
SET LINK TIME TO SECOND TIME OR SPEED FIELD CALCULATE LINK SPEED FROM TIME AND DISTANCE
IS THE TWO-WAY INDICATOR = 1?
YES
NO
IS THE TWO-WAY INDICATOR = 2?
YES
NO
PRINT AN ERROR MESSAGE, INVALID TWO-WAY INDICATOR, ERROR = ERROR + 1
IL = IL - 1 (THIS REMOVES THE PREVIOUS ONE-WAY LINK)
N 1
SET SPEED = SECOND TIME OR SPEED FIELD
N 9
PRINT ERROR MESSAGE THAT LINK TIME IS > MAX. ERROR = ERROR + 1 SET LINK TIME TO MAX.
IS THE LINK TIME > THE MAX. LINK TIME?
YES
NO
IL = IL + 1 (ADD ONE TO THE NUMBER OF LINKS IN CORE)
N 7
WAS THE LINK CLASS IN THE OTHER DIRECTION ONE WAY?
YES
IL = 3 (SET LINK CLASS TO DUMMY ONE-WAY LINK)
NO
PTLNK
PACK THE LINK INFORMATION INTO A 22 BYTE RECORD AND PUT IN LINKS (IL)
IS THE LINKS ARRAY FULL?
NO
N 1
YES
LNK2 = IL + 1 = NUMBER OF LINKS IN CORE. GET TIME OF DAY TO TIME SORT.
VSORT
SORT 22 BYTE LINK RECORDS IN CORE ON THE KEY OF A NODE, LINK CLASS, CARD COUNT
GET TIME OF DAY AND CALCULATE SORT TIME AND SUM IN IT3. REWIND LNKTMP
PTLNK
ADD A TRAILER LINK WITH AN A NODE OF 16383 TO MARK THE END OF THE LINKS.
CALCULATE THE NUMBER OF BLOCKS OF LINKS. IL = IL + 2, NBLK = (IL + 39)/40, (40 LINKS/RECORD).
N 11

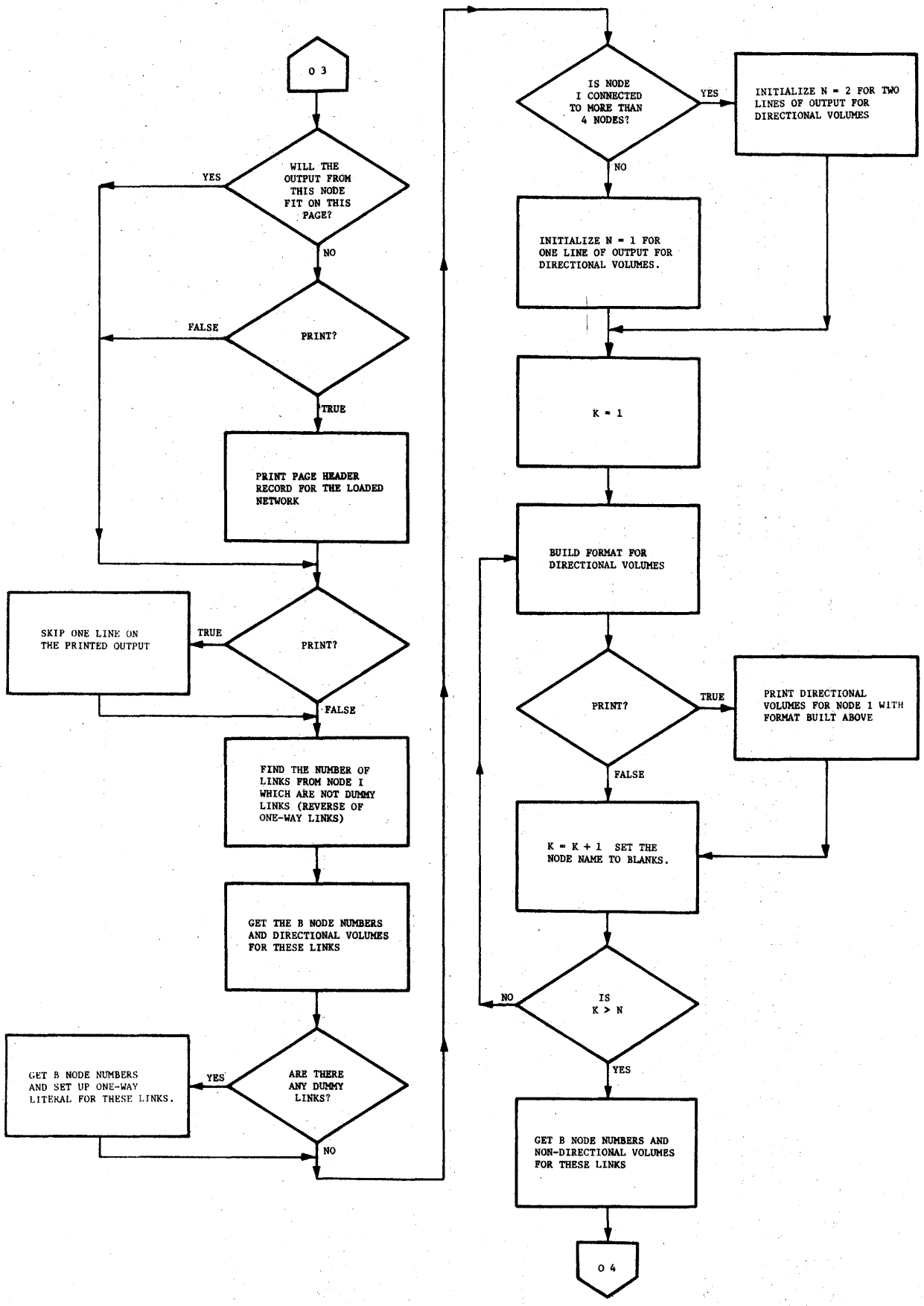

N 11
WRITE THE SORTED LINKS IN CORE OR LNKTMP IN BLOCKS OF 40 LINKS/RECORD.
END FILE LNKTMP
REWIND LNKTMP
IS LNKTMP = 3
YES
LNK 1 = LNK 2
LNK 2 = 0
NO
IS LNKTMP = 11
YES
LNKTMP = -1
NO
IS LNKTMP = 3
YES
LNKTMP = 11
NO
IL = -1
(SET FOR NO LINKS IN CORE)
N 3

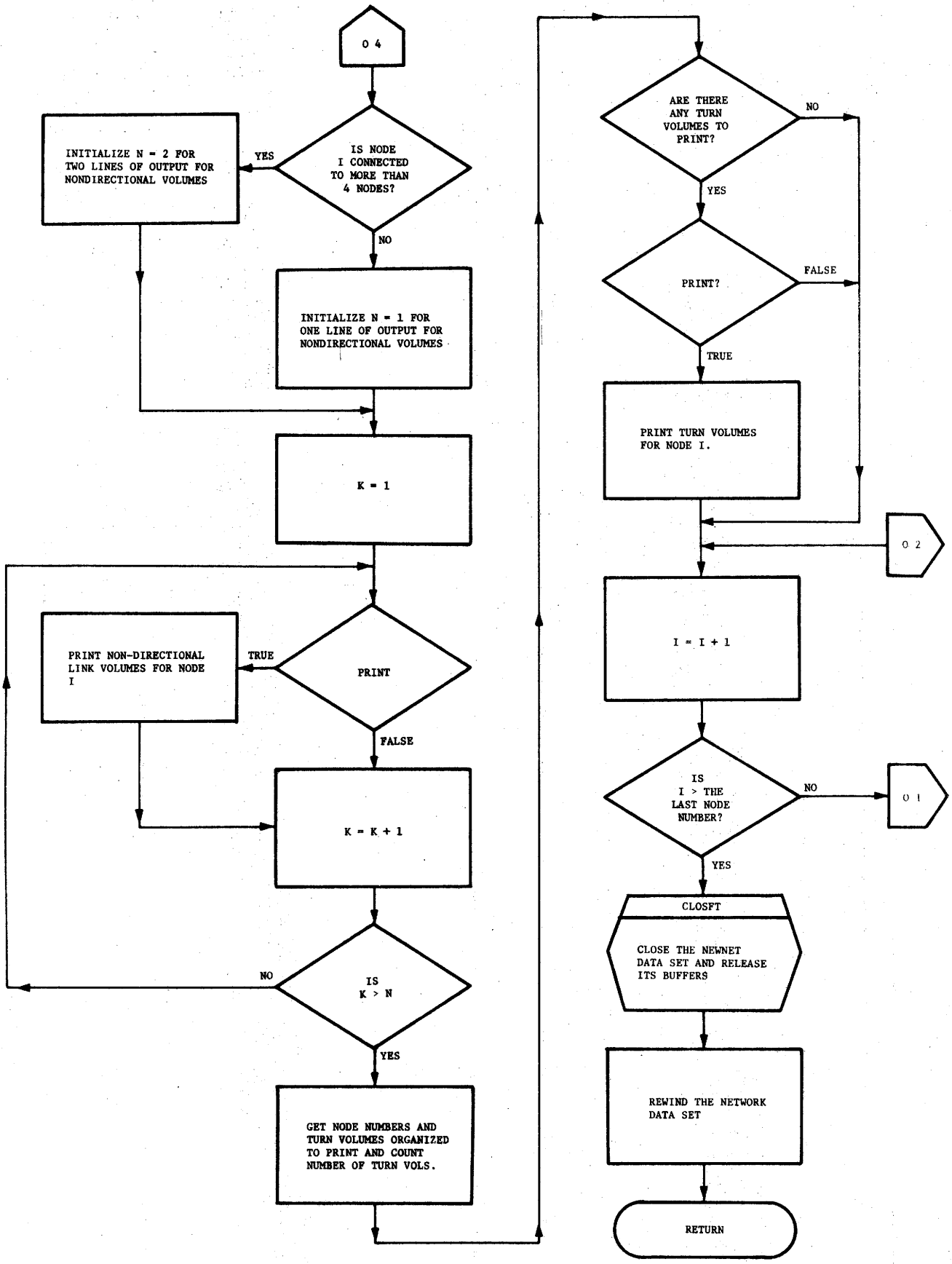

N 14
PTLNK
ADD A TRAILER LINK WITH AN ANODE OF 16383 TO MARK THE END OF THE LINKS
LNK2 = LNK2 + 1
CALCULATE NUMBER OF RECORDS TO WRITE ON LNKTMP.
WRITE THE SORTED LINKS IN CORE ON LNKTMP IN BLOCKS OF 40 LINKS/RECORD
END FILE LNKTMP
REWIND LNKTMP
IS LNKTMP = 3
YES
LNK1 = LNK2
LNK2 = 0
NO
IS LNKTMP = 1
YES
LNKTMP = -1
NO
IS LNKTMP = 3
YES
LNKTMP = 11
NO
IL = -1
(SET NUMBER OF LINKS IN CORE TO ZERO)
N 3

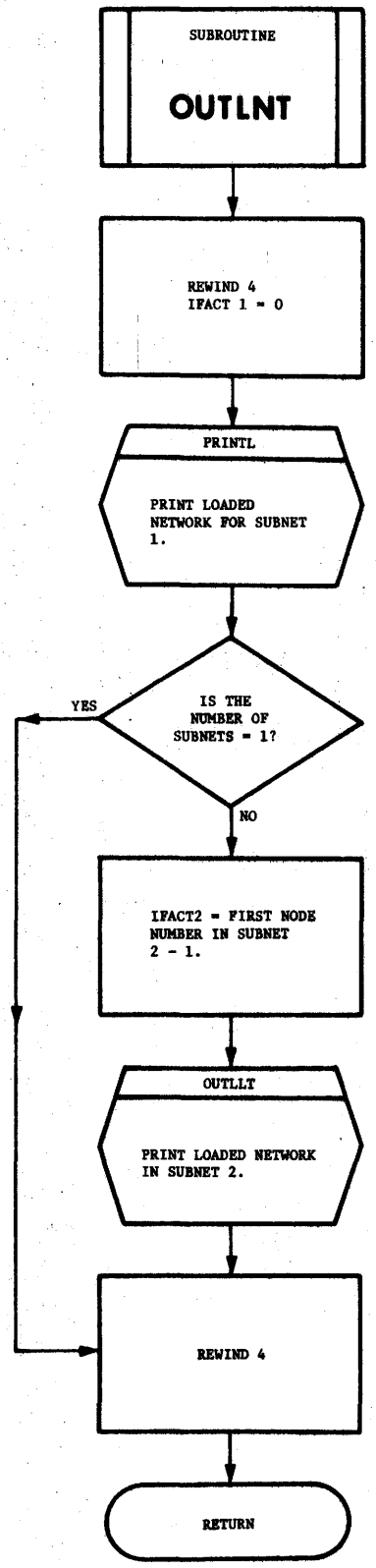

N 15
NEWNET
HAVE ALL SUBNETS BEEN PROCESSED?
NO
PRINT SUBNET NUMBER
YES
N 16
PRINT SUBNET NUMBER. END FILE 4, REWIND 4 (NODE NAME DATA SET)
N 13
N 12
IL = IL + 1
GET TIME OF DAY TO TIME SORT
VSORT
SORT 22 BYTE LINK RECORDS IN CORE ON THE KEY OF ANODE, LINK CLASS, CARD COUNT
GET TIME OF DAY, CALCULATE SORT TIME AND ADD TO IT3 AND CONVERT TO MINUTES
PRINT LINK SORT TIME.
PTLNK
ADD A TRAILER LINK WITH AN ANODE OF 16383 TO MARK THE END OF THE LINKS
IL = IL + 1
RETURN

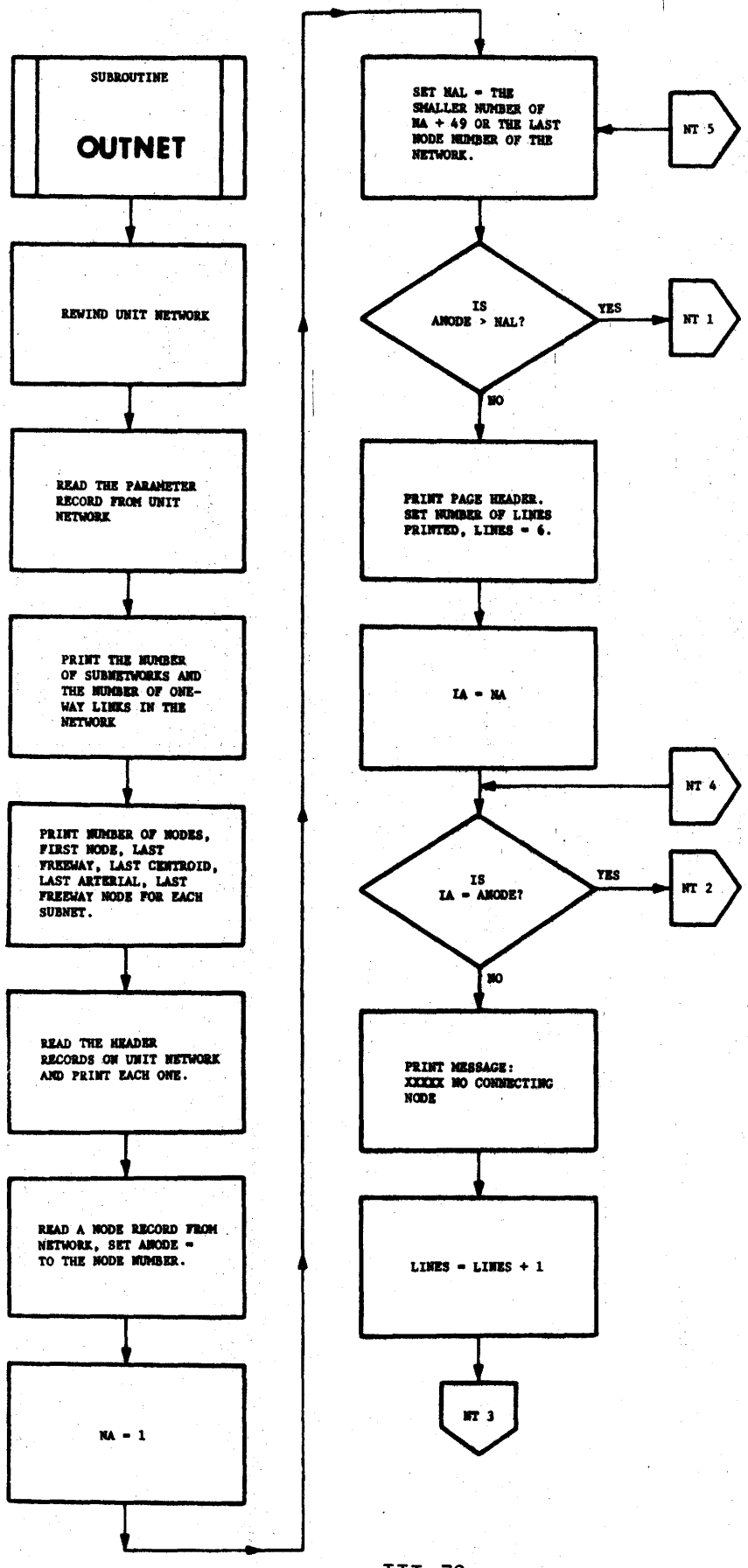

SUBROUTINE
OUTLLT
RES = .TRUE. IF THIS IS AN ASSIGN SELF-BALANCING RUN. ITER = ITERATION NUMBER. OUTN = A LOGICAL VARIABLE TO KEEP THE PRINTED OUTPUT
PRINT = (.NOT. RES .OR. ITER.EQ.1) .AND. OUTN
RES?
FALSE
TRUE
ITER = 1?
NO
YES
REWIND 3 WRITE A PARAMETER RECORD ON UNIT 3 OF THE NUMBER OF NODES ONE-WAY LINKS AND TURNING MOVEMENTS
WRITE THE ONE-WAY LINK VOLUMES IN FULL WORD INTEGERS IN RECORDS OF 4000 WORDS ON UNIT 3.
WRITE THE TURN VOLUMES SAVED IN FULL WORD INTEGERS IN RECORDS OF 4000 WORDS ON UNIT 3.
REWIND NETWORK
OPENFT
OPEN DATA SET NEWNET WITH REFERENCE 1
READ PARAMETER RECORD FROM UNIT NETWORK, ADD ONE TO NUMBER OF ASSIGNMENTS
NL1 = NUMBER OF ASSIGNMENTS FROM UNIT NETWORK + 1.
WRT
WRITE NEW PARAMETER RECORD ON UNIT NEWNET
COPY NL1 HEADER RECORDS FROM NETWORK TO NEWNET.
WRT
WRITE HEADER RECORD FOR THIS ASSIGNMENT.
SET NL5 = NUMBER OF WORDS IN LINK RECORDS ON UNIT NETWORK
I = 1
O 1

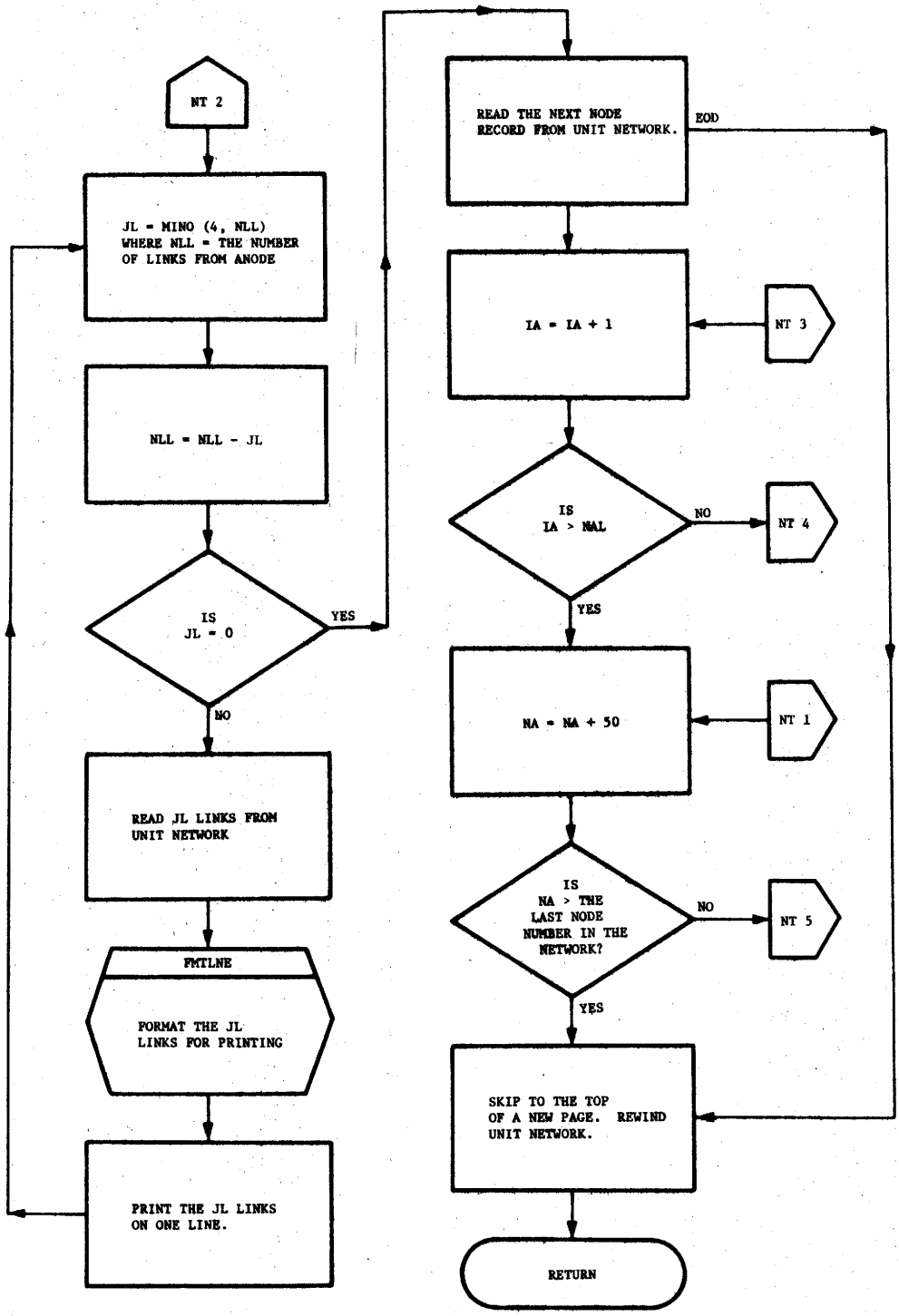

01
SET INUM2 = THE NUMBER OF LINKS FROM NODE I.
IS INUM2 = 0?
YES
02
NO
TURNM
GET ONE-WAY AND TWO-WAY LINK VOLUMES. CALCULATE TURN VOLUMES AND FLAG WHICH TO PRINT
READ NODE RECORD FROM UNIT NETWORK FOR NODE I.
WRT
WRITE NODE RECORD FOR NODE I ON UNIT NEWNET.
J = 1
READ A LINK RECORD FROM UNIT NETWORK
GET ITP = THE LINK IMPEDANCE USED FOR THIS ASSIGNMENT
GET C = GROUND COUNT OR CAPACITY (WHICHEVER IS SPECIFIED BY THE *TURN CARD)
IS THIS NOT AN ASSIGN SELF-BALANCING .OR. C = 0
YES
NO
IS IT A CAPACITY FIELD AND LINK VOLUME < C?
TRUE
FALSE
UPDATE LINK IMPEDANCE TO USE ON THE NEXT ASSIGNMENT
IS THE NEW LINK IMPEDANCE = 0 AND ITP ≠ 0?
YES
SET THE NEW LINK IMPEDANCE TO 0.01
NO
IS THE NEW LINK IMPEDANCE > 163.83?
NO
YES
SET THE NEW LINK IMPEDANCE TO 163.83
WRT
WRITE THE NEW LINK RECORD WITH ITS ITP AND TWO-WAY LINK VOLUME FOR THIS ASSIGNMENT
J = J + 1
IS J > INUM2?
NO
YES
03

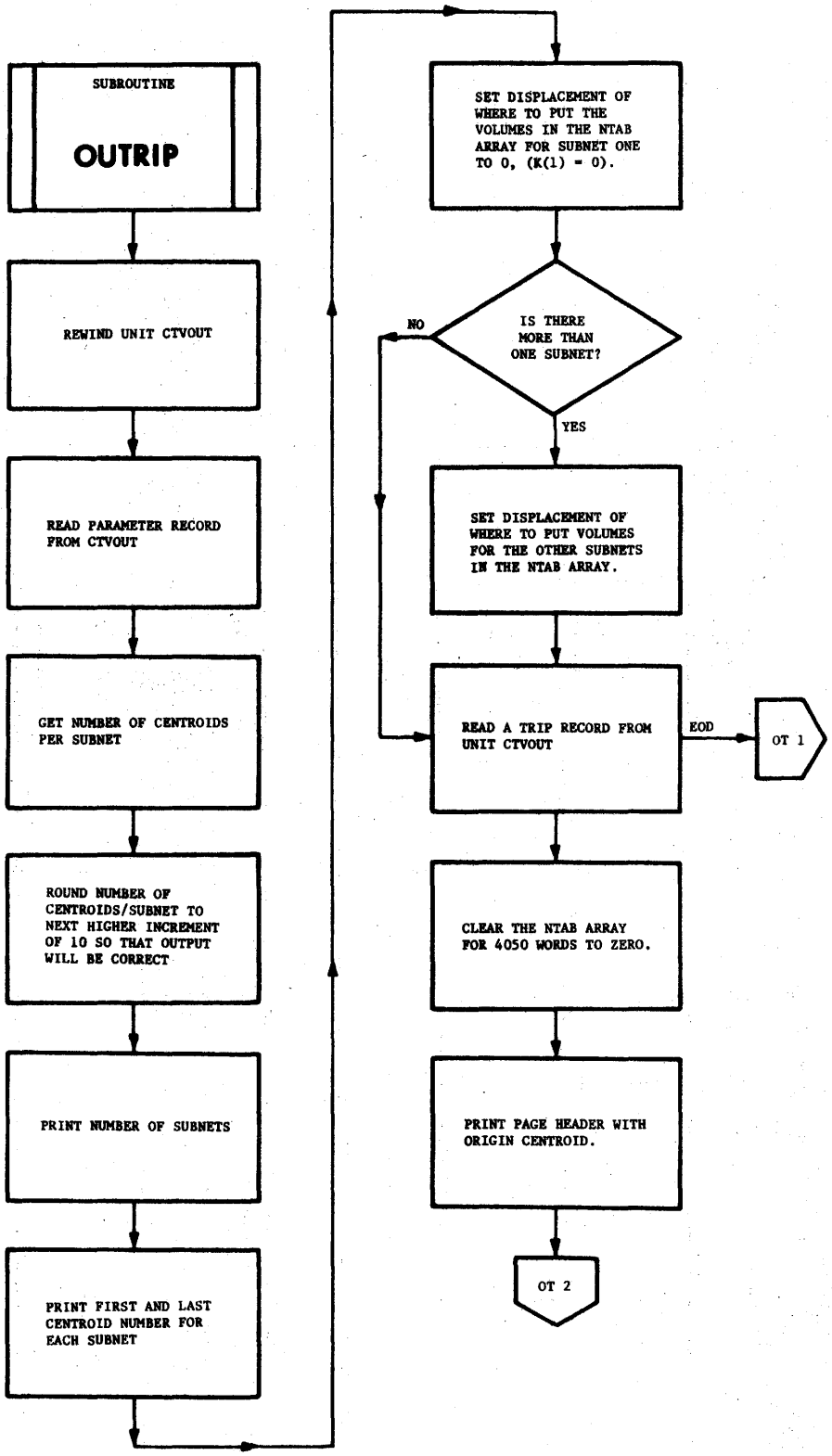

0 3
WILL THE OUTPUT FROM THIS NODE FIT ON THIS PAGE?
YES
NO
PRINT?
FALSE
TRUE
PRINT PAGE HEADER RECORD FOR THE LOADED NETWORK
SKIP ONE LINE ON THE PRINTED OUTPUT
TRUE
PRINT?
FALSE
FIND THE NUMBER OF LINKS FROM NODE I WHICH ARE NOT DUMMY LINKS (REVERSE OF ONE-WAY LINKS)
GET THE B NODE NUMBERS AND DIRECTIONAL VOLUMES FOR THESE LINKS
GET B NODE NUMBERS AND SET UP ONE-WAY LITERAL FOR THESE LINKS.
YES
ARE THERE ANY DUMMY LINKS?
NO
IS NODE I CONNECTED TO MORE THAN 4 NODES?
YES
INITIALIZE N = 2 FOR TWO LINES OF OUTPUT FOR DIRECTIONAL VOLUMES
NO
INITIALIZE N = 1 FOR ONE LINE OF OUTPUT FOR DIRECTIONAL VOLUMES.
K = 1
BUILD FORMAT FOR DIRECTIONAL VOLUMES
PRINT?
TRUE
PRINT DIRECTIONAL VOLUMES FOR NODE 1 WITH FORMAT BUILT ABOVE
FALSE
K = K + 1 SET THE NODE NAME TO BLANKS.
IS K > N
NO
YES
GET B NODE NUMBERS AND NON-DIRECTIONAL VOLUMES FOR THESE LINKS
0 4

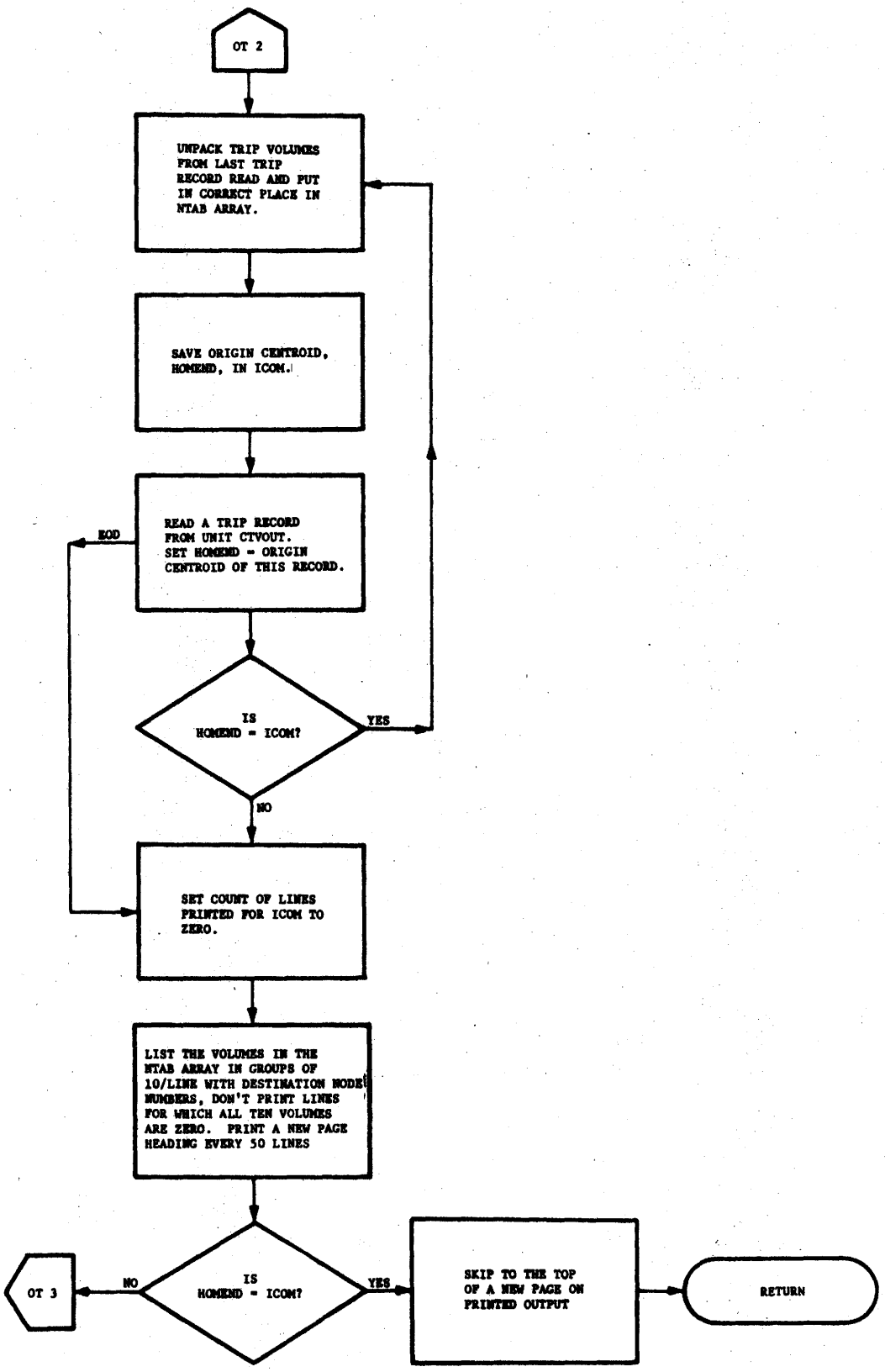

0 4
IS NODE I CONNECTED TO MORE THAN 4 NODES?
YES
INITIALIZE N = 2 FOR TWO LINES OF OUTPUT FOR NONDIRECTIONAL VOLUMES
NO
INITIALIZE N = 1 FOR ONE LINE OF OUTPUT FOR NONDIRECTIONAL VOLUMES
K = 1
PRINT
TRUE
PRINT NON-DIRECTIONAL LINK VOLUMES FOR NODE I
FALSE
K = K + 1
IS K > N
NO
YES
GET NODE NUMBERS AND TURN VOLUMES ORGANIZED TO PRINT AND COUNT NUMBER OF TURN VOLS.
ARE THERE ANY TURN VOLUMES TO PRINT?
NO
YES
PRINT?
FALSE
TRUE
PRINT TURN VOLUMES FOR NODE I.
0 2
I = I + 1
IS I > THE LAST NODE NUMBER?
NO
0 1
YES
CLOSFT
CLOSE THE NEWNET DATA SET AND RELEASE ITS BUFFERS
REWIND THE NETWORK DATA SET
RETURN

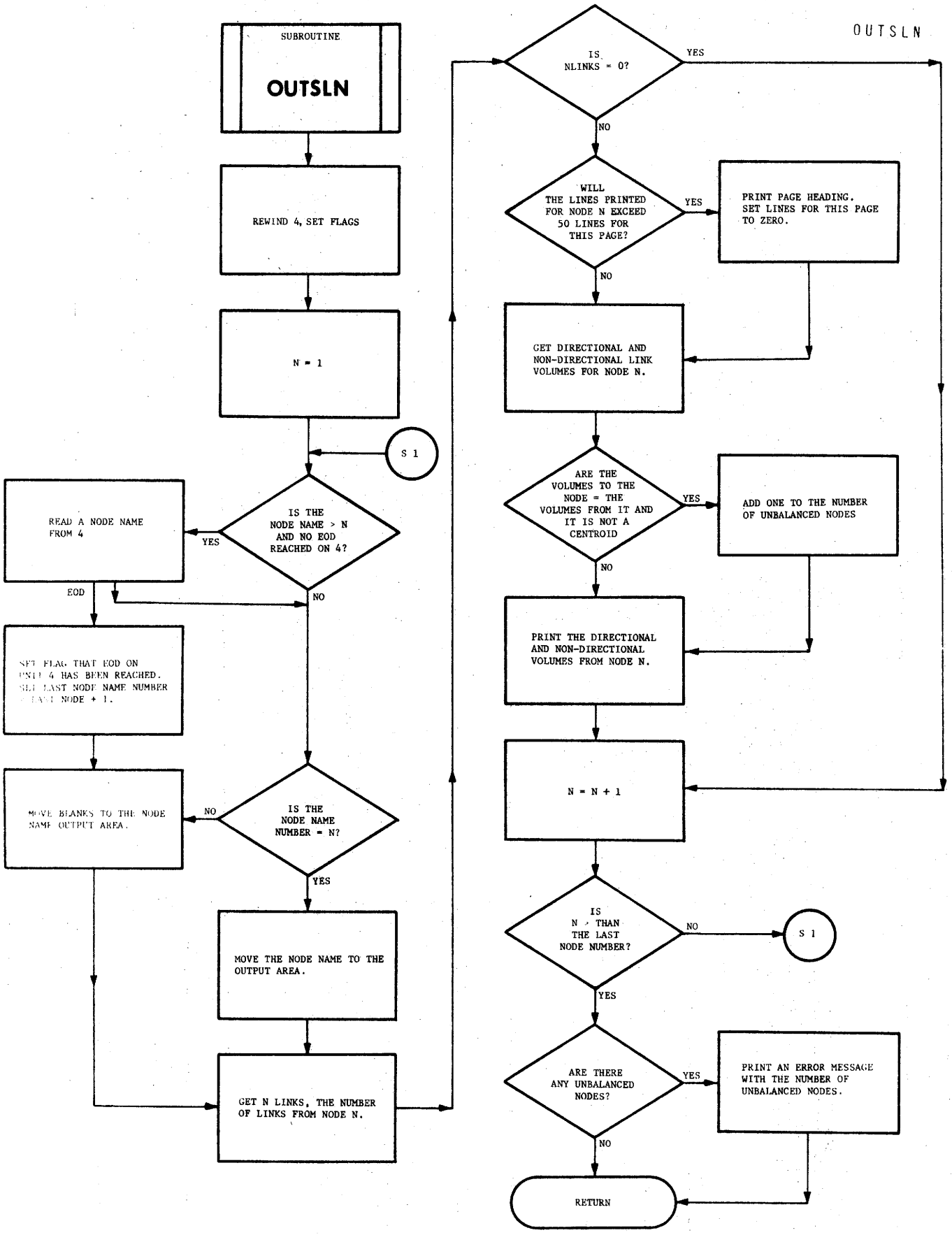

SUBROUTINE
OUTLNT
REWIND 4
IFACT 1 = 0
PRINTL
PRINT LOADED NETWORK FOR SUBNET 1.
IS THE NUMBER OF SUBNETS = 1?
YES
NO
IFACT2 = FIRST NODE NUMBER IN SUBNET 2 - 1.
OUTLLT
PRINT LOADED NETWORK IN SUBNET 2.
REWIND 4
RETURN

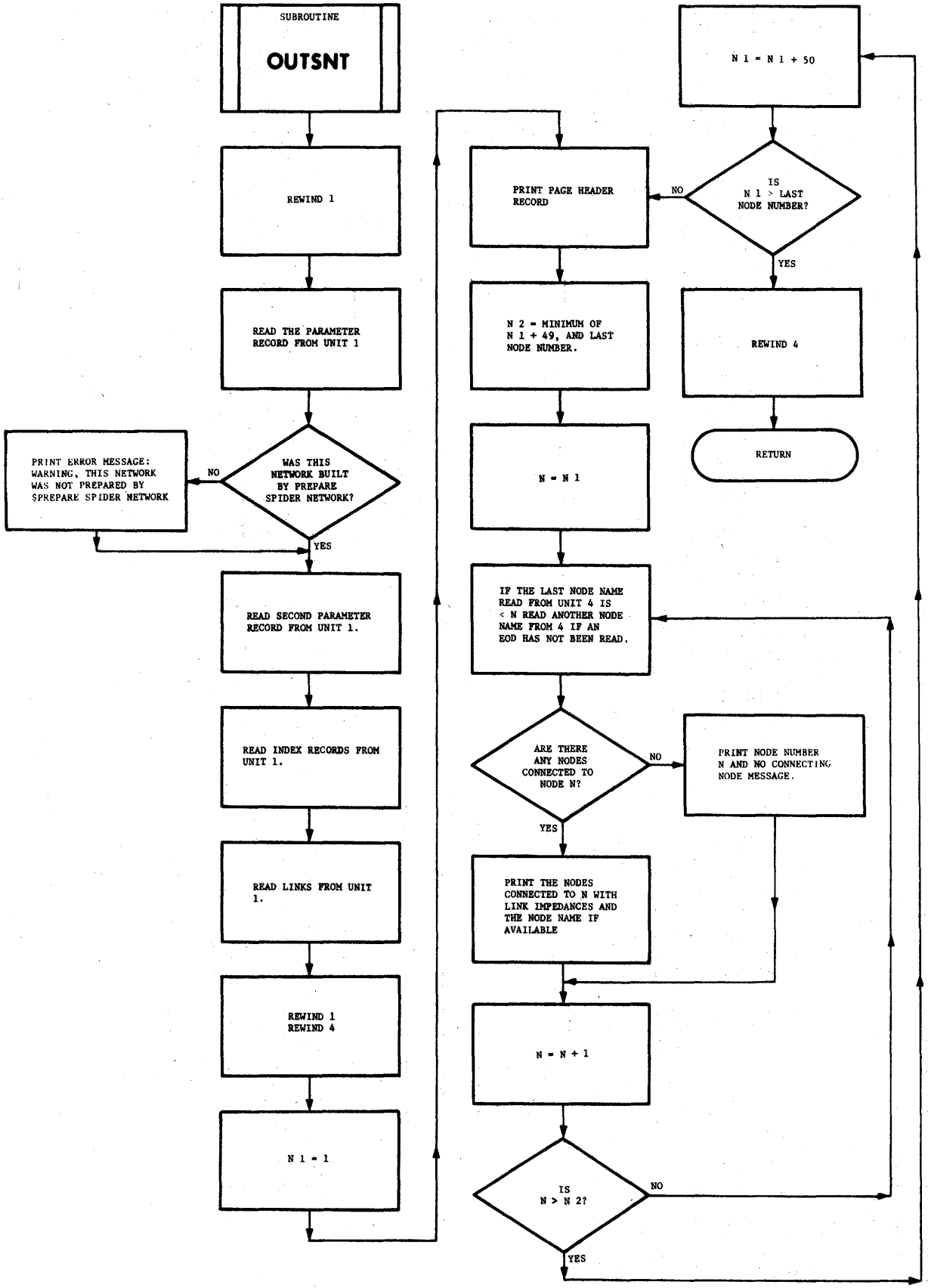

SUBROUTINE
OUTNET
REWIND UNIT NETWORK
READ THE PARAMETER RECORD FROM UNIT NETWORK
PRINT THE NUMBER OF SUBNETWORKS AND THE NUMBER OF ONE-WAY LINKS IN THE NETWORK
PRINT NUMBER OF NODES, FIRST NODE, LAST FREEWAY, LAST CENTROID, LAST ARTERIAL, LAST FREEWAY NODE FOR EACH SUBNET.
READ THE HEADER RECORDS ON UNIT NETWORK AND PRINT EACH ONE.
READ A NODE RECORD FROM NETWORK, SET ANODE = TO THE NODE NUMBER.
NA = 1
SET NAL = THE SMALLER NUMBER OF NA + 49 OR THE LAST NODE NUMBER OF THE NETWORK.
NT 5
IS ANODE > NAL?
YES
NT 1
NO
PRINT PAGE HEADER. SET NUMBER OF LINES PRINTED, LINES = 6.
IA = NA
NT 4
IS IA = ANODE?
YES
NT 2
NO
PRINT MESSAGE: XXXXX NO CONNECTING NODE
LINES = LINES + 1
NT 3

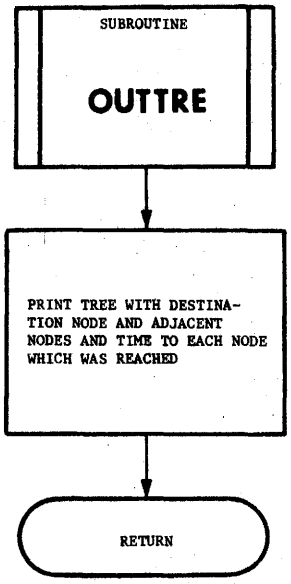

NT 2
JL = MINO (4, NLL)
WHERE NLL = THE NUMBER
OF LINKS FROM ANODE
NLL = NLL - JL
IS
JL = 0
YES
NO
READ JL LINKS FROM
UNIT NETWORK
FMTLNE
FORMAT THE JL
LINKS FOR PRINTING
PRINT THE JL LINKS
ON ONE LINE.
READ THE NEXT NODE
RECORD FROM UNIT NETWORK.
EOD
IA = IA + 1
NT 3
IS
IA > NAL
NO
NT 4
YES
NA = NA + 50
NT 1
IS
NA > THE
LAST NODE
NUMBER IN THE
NETWORK?
NO
NT 5
YES
SKIP TO THE TOP
OF A NEW PAGE. REWIND
UNIT NETWORK.
RETURN

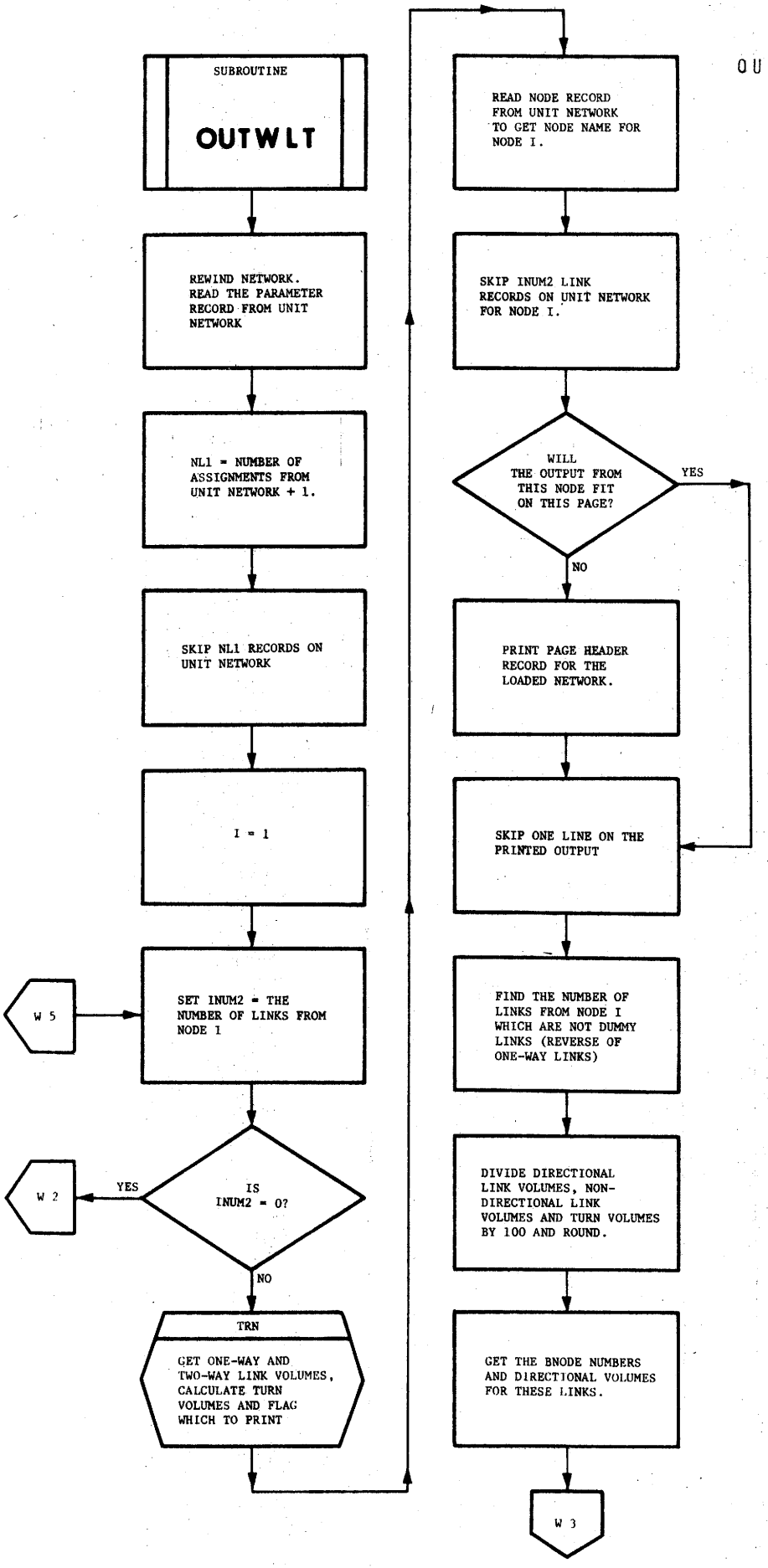

SUBROUTINE
OUTRIP
REWIND UNIT CTVOUT
READ PARAMETER RECORD FROM CTVOUT
GET NUMBER OF CENTROIDS PER SUBNET
ROUND NUMBER OF CENTROIDS/SUBNET TO NEXT HIGHER INCREMENT OF 10 SO THAT OUTPUT WILL BE CORRECT
PRINT NUMBER OF SUBNETS
PRINT FIRST AND LAST CENTROID NUMBER FOR EACH SUBNET
SET DISPLACEMENT OF WHERE TO PUT THE VOLUMES IN THE NTAB ARRAY FOR SUBNET ONE TO 0, (K(1) = 0).
IS THERE MORE THAN ONE SUBNET?
NO
YES
SET DISPLACEMENT OF WHERE TO PUT VOLUMES FOR THE OTHER SUBNETS IN THE NTAB ARRAY.
READ A TRIP RECORD FROM UNIT CTVOUT
EOD
OT 1
CLEAR THE NTAB ARRAY FOR 4050 WORDS TO ZERO.
PRINT PAGE HEADER WITH ORIGIN CENTROID.
OT 2

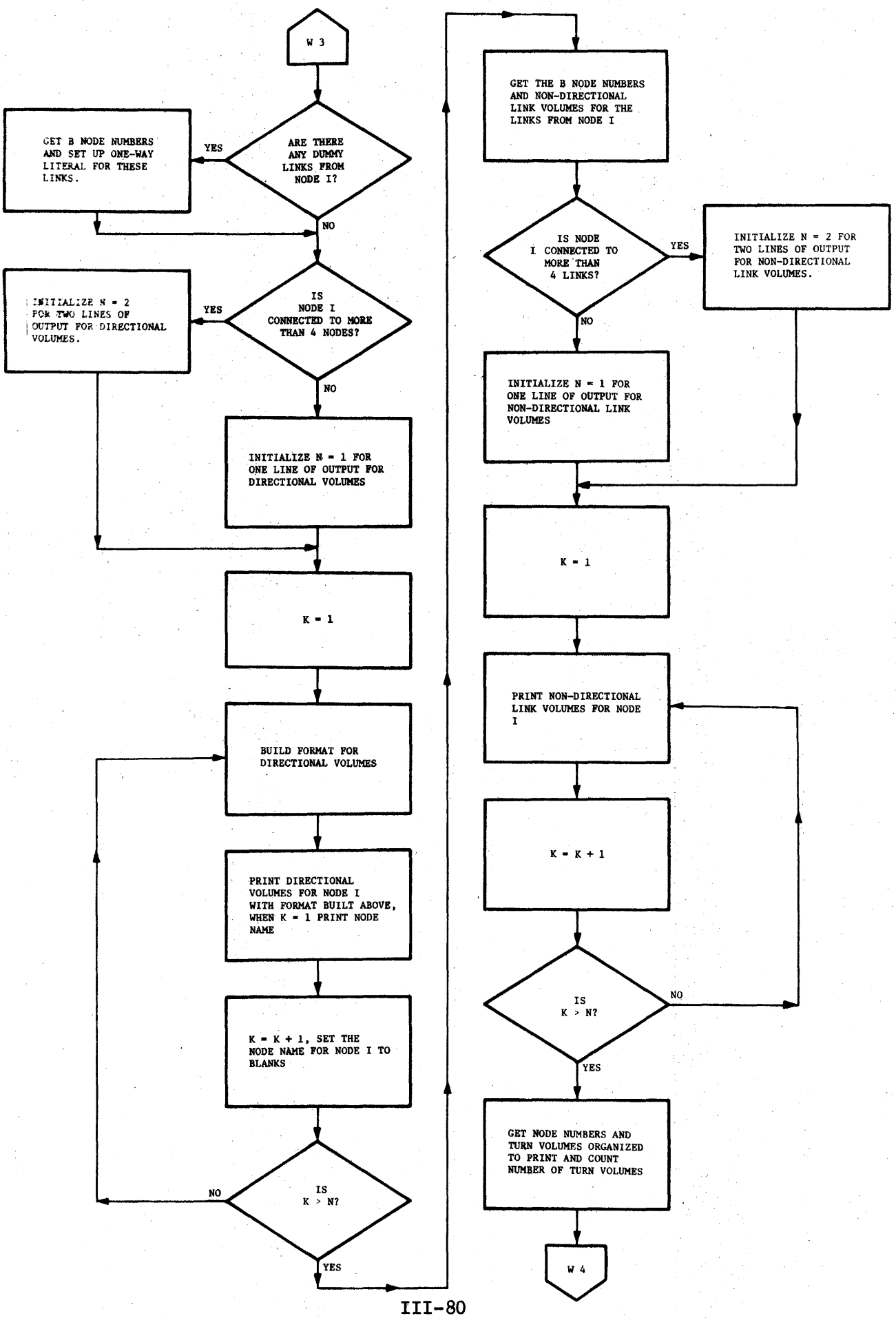

OT 2
UNPACK TRIP VOLUMES FROM LAST TRIP RECORD READ AND PUT IN CORRECT PLACE IN NTAB ARRAY.
SAVE ORIGIN CENTROID, HOMEND, IN ICOM.
READ A TRIP RECORD FROM UNIT CTVOUT. SET HOMEND = ORIGIN CENTROID OF THIS RECORD.
EOD
IS HOMEND = ICOM?
YES
NO
SET COUNT OF LINES PRINTED FOR ICOM TO ZERO.
LIST THE VOLUMES IN THE NTAB ARRAY IN GROUPS OF 10/LINE WITH DESTINATION NODE NUMBERS, DON'T PRINT LINES FOR WHICH ALL TEN VOLUMES ARE ZERO. PRINT A NEW PAGE HEADING EVERY 50 LINES
IS HOMEND = ICOM?
NO
YES
OT 3
SKIP TO THE TOP OF A NEW PAGE ON PRINTED OUTPUT
RETURN

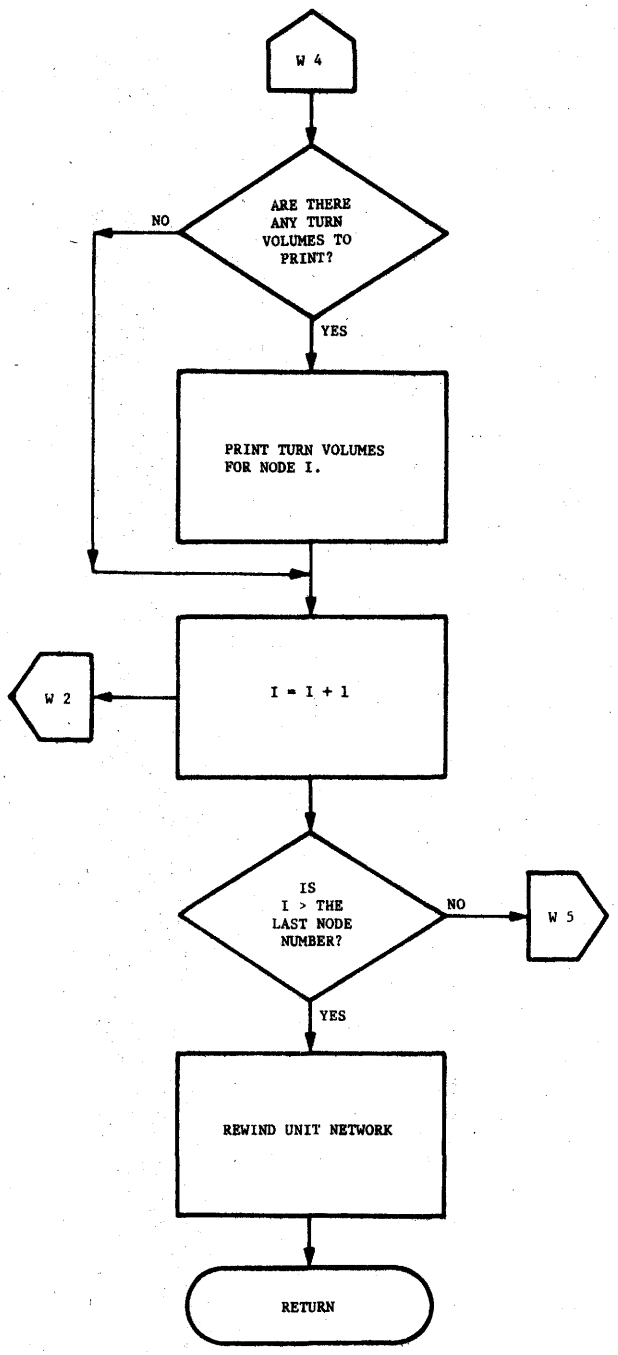

SUBROUTINE
OUTSLN
REWIND 4, SET FLAGS
N = 1
S 1
READ A NODE NAME FROM 4
IS THE NODE NAME > N AND NO EOD REACHED ON 4?
YES
NO
EOD
SET FLAG THAT EOD ON UNIT 4 HAS BEEN REACHED. SET LAST NODE NAME NUMBER = LAST NODE + 1.
MOVE BLANKS TO THE NODE NAME OUTPUT AREA.
IS THE NODE NAME NUMBER = N?
NO
YES
MOVE THE NODE NAME TO THE OUTPUT AREA.
GET N LINKS, THE NUMBER OF LINKS FROM NODE N.
IS NLINKS = 0?
YES
NO
WILL THE LINES PRINTED FOR NODE N EXCEED 50 LINES FOR THIS PAGE?
YES
PRINT PAGE HEADING. SET LINES FOR THIS PAGE TO ZERO.
NO
GET DIRECTIONAL AND NON-DIRECTIONAL LINK VOLUMES FOR NODE N.
ARE THE VOLUMES TO THE NODE = THE VOLUMES FROM IT AND IT IS NOT A CENTROID
YES
ADD ONE TO THE NUMBER OF UNBALANCED NODES
NO
PRINT THE DIRECTIONAL AND NON-DIRECTIONAL VOLUMES FROM NODE N.
N = N + 1
IS N > THAN THE LAST NODE NUMBER?
NO
S 1
YES
ARE THERE ANY UNBALANCED NODES?
YES
PRINT AN ERROR MESSAGE WITH THE NUMBER OF UNBALANCED NODES.
NO
RETURN

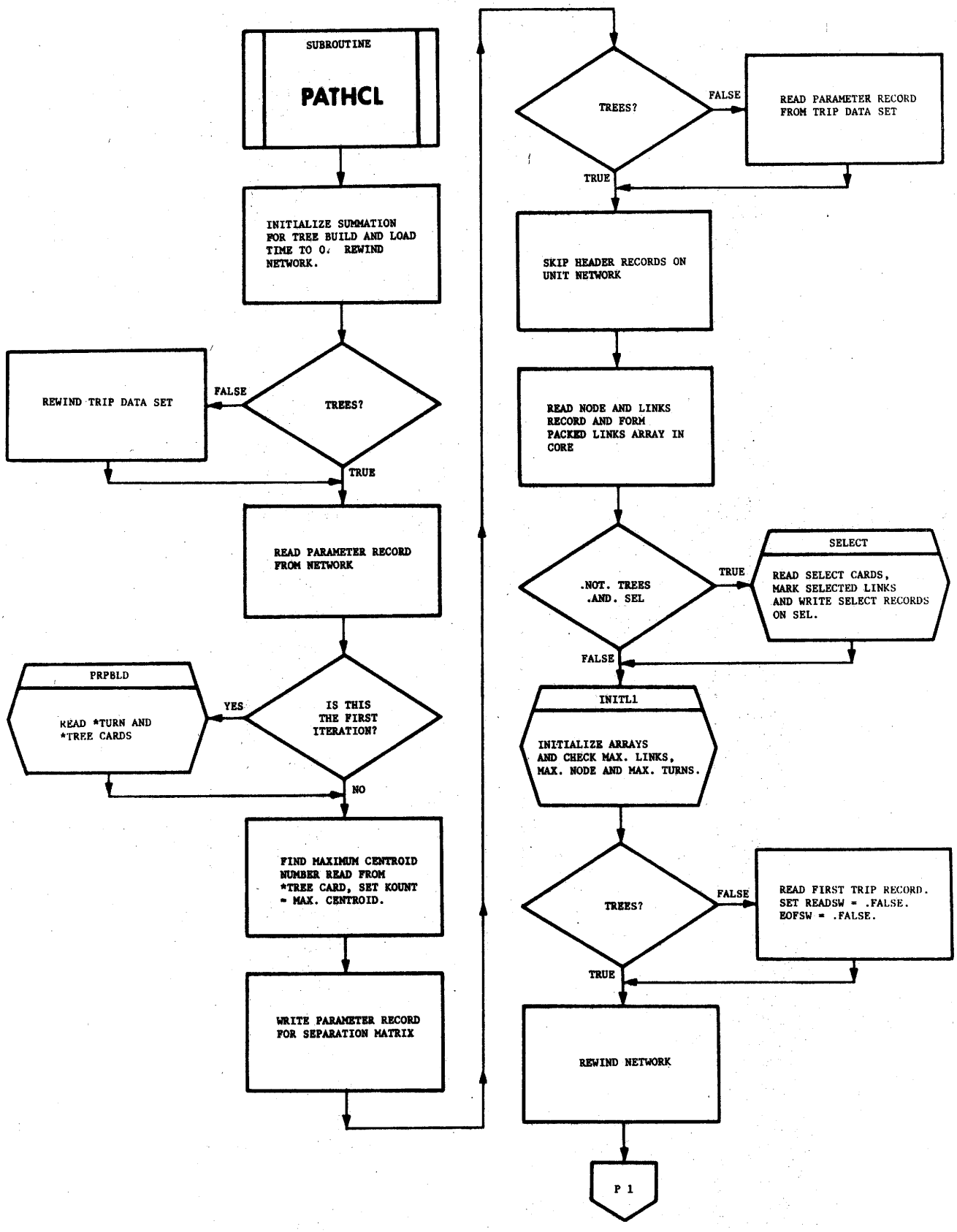

SUBROUTINE
OUTSNT
REWIND 1
READ THE PARAMETER RECORD FROM UNIT 1
WAS THIS NETWORK BUILT BY PREPARE SPIDER NETWORK?
NO
PRINT ERROR MESSAGE: WARNING, THIS NETWORK WAS NOT PREPARED BY $PREPARE SPIDER NETWORK
YES
READ SECOND PARAMETER RECORD FROM UNIT 1.
READ INDEX RECORDS FROM UNIT 1.
READ LINKS FROM UNIT 1.
REWIND 1
REWIND 4
N 1 = 1
PRINT PAGE HEADER RECORD
N 2 = MINIMUM OF N 1 + 49, AND LAST NODE NUMBER.
N = N 1
IF THE LAST NODE NAME READ FROM UNIT 4 IS < N READ ANOTHER NODE NAME FROM 4 IF AN EOD HAS NOT BEEN READ.
ARE THERE ANY NODES CONNECTED TO NODE N?
NO
PRINT NODE NUMBER N AND NO CONNECTING NODE MESSAGE.
YES
PRINT THE NODES CONNECTED TO N WITH LINK IMPEDANCES AND THE NODE NAME IF AVAILABLE
N = N + 1
IS N > N 2?
NO
YES
N 1 = N 1 + 50
IS N 1 > LAST NODE NUMBER?
NO
YES
REWIND 4
RETURN

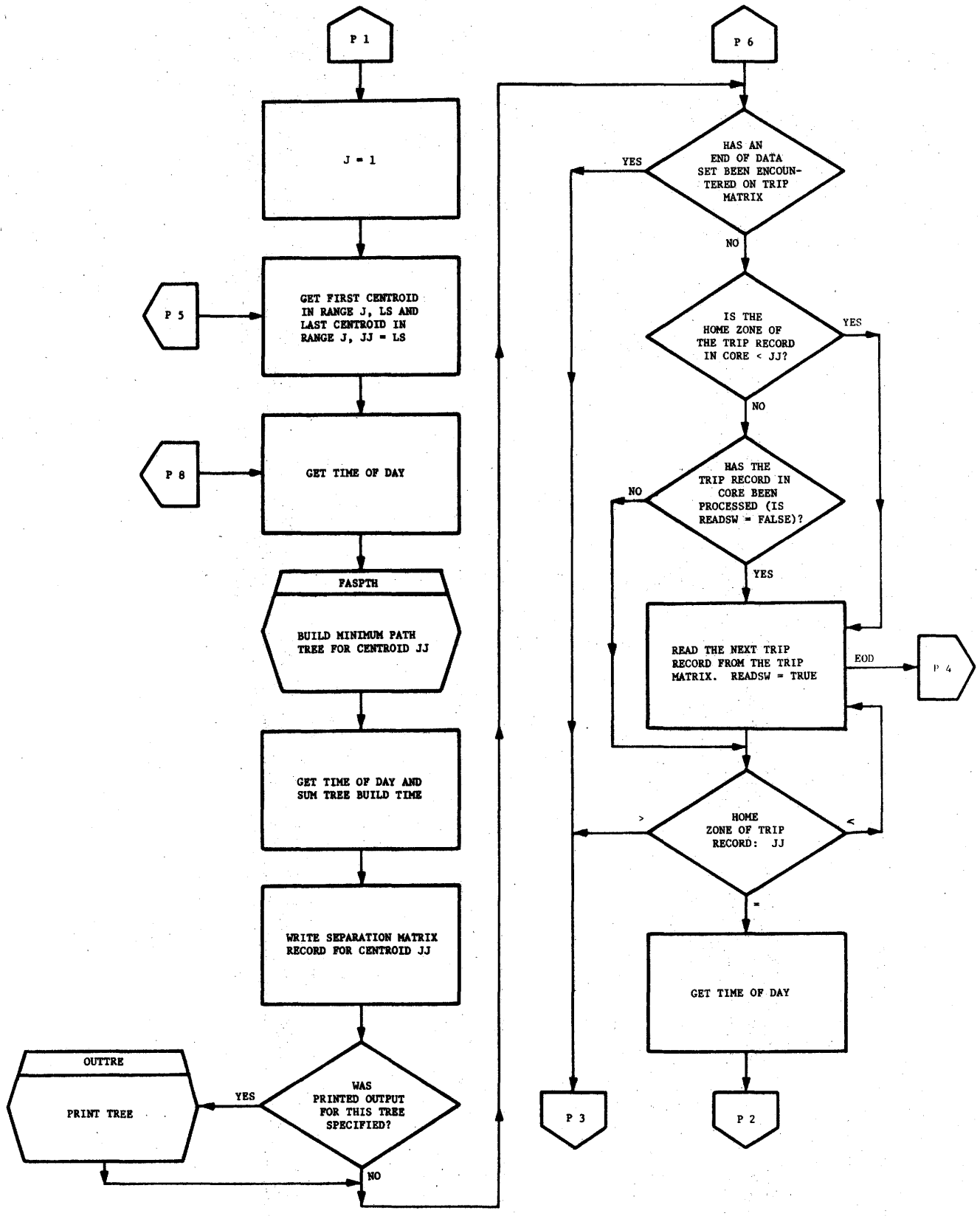

SUBROUTINE
OUTTRE
PRINT TREE WITH DESTINATION NODE AND ADJACENT NODES AND TIME TO EACH NODE WHICH WAS REACHED
RETURN

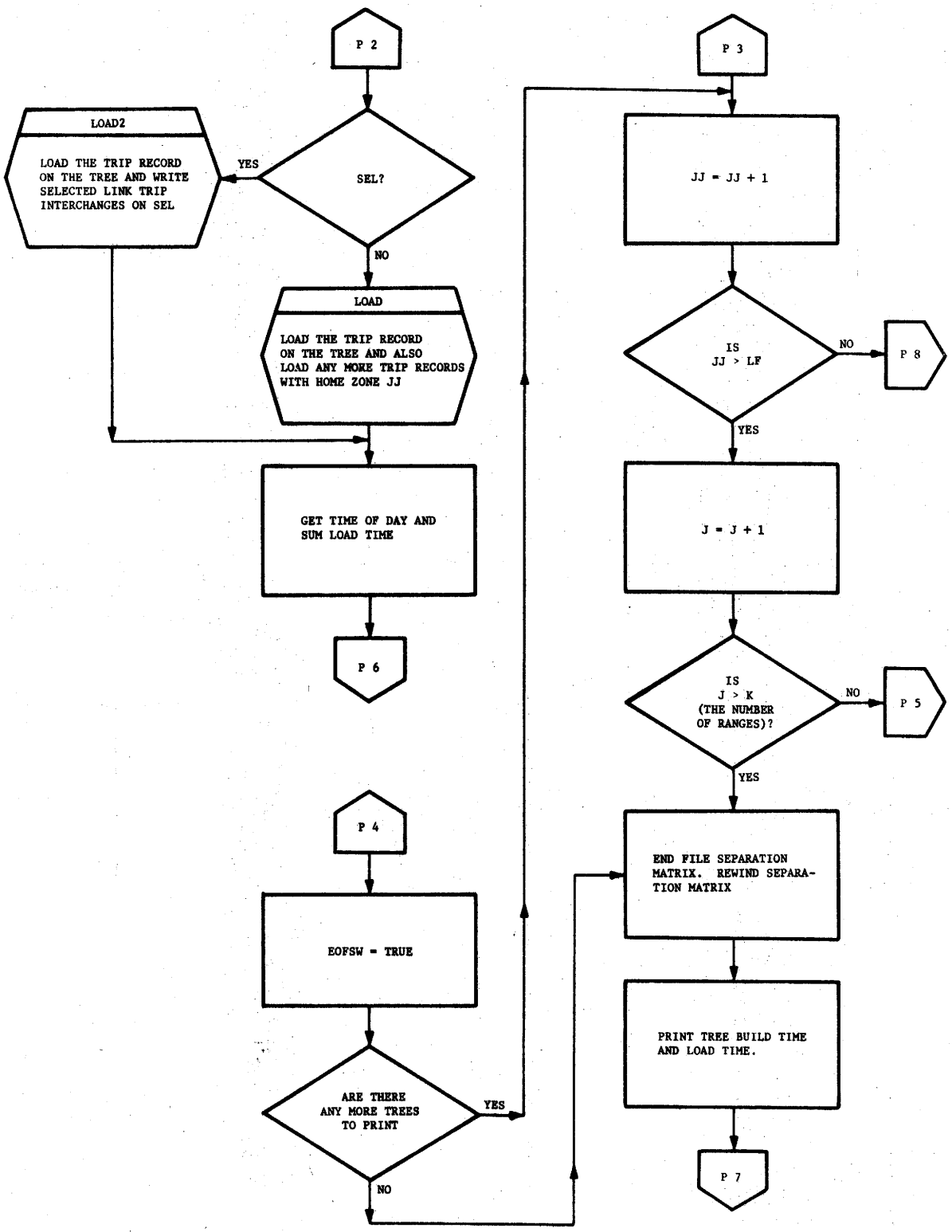

SUBROUTINE
OUTWLT
REWIND NETWORK. READ THE PARAMETER RECORD FROM UNIT NETWORK
NL1 = NUMBER OF ASSIGNMENTS FROM UNIT NETWORK + 1.
SKIP NL1 RECORDS ON UNIT NETWORK
I = 1
W 5
SET INUM2 = THE NUMBER OF LINKS FROM NODE 1
W 2
YES
IS INUM2 = 0?
NO
TRN
GET ONE-WAY AND TWO-WAY LINK VOLUMES, CALCULATE TURN VOLUMES AND FLAG WHICH TO PRINT
READ NODE RECORD FROM UNIT NETWORK TO GET NODE NAME FOR NODE I.
SKIP INUM2 LINK RECORDS ON UNIT NETWORK FOR NODE I.
WILL THE OUTPUT FROM THIS NODE FIT ON THIS PAGE?
YES
NO
PRINT PAGE HEADER RECORD FOR THE LOADED NETWORK.
SKIP ONE LINE ON THE PRINTED OUTPUT
FIND THE NUMBER OF LINKS FROM NODE I WHICH ARE NOT DUMMY LINKS (REVERSE OF ONE-WAY LINKS)
DIVIDE DIRECTIONAL LINK VOLUMES, NON-DIRECTIONAL LINK VOLUMES AND TURN VOLUMES BY 100 AND ROUND.
GET THE BNODE NUMBERS AND DIRECTIONAL VOLUMES FOR THESE LINKS.
W 3

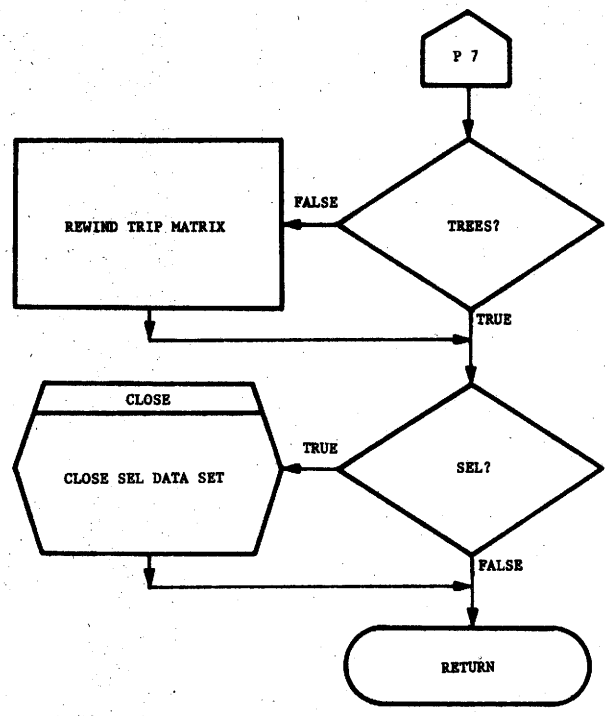

W 3
ARE THERE ANY DUMMY LINKS FROM NODE I?
YES
GET B NODE NUMBERS AND SET UP ONE-WAY LITERAL FOR THESE LINKS.
NO
IS NODE I CONNECTED TO MORE THAN 4 NODES?
YES
INITIALIZE N = 2 FOR TWO LINES OF OUTPUT FOR DIRECTIONAL VOLUMES.
NO
INITIALIZE N = 1 FOR ONE LINE OF OUTPUT FOR DIRECTIONAL VOLUMES
K = 1
BUILD FORMAT FOR DIRECTIONAL VOLUMES
PRINT DIRECTIONAL VOLUMES FOR NODE I WITH FORMAT BUILT ABOVE, WHEN K = 1 PRINT NODE NAME
K = K + 1, SET THE NODE NAME FOR NODE I TO BLANKS
IS K > N?
NO
YES
GET THE B NODE NUMBERS AND NON-DIRECTIONAL LINK VOLUMES FOR THE LINKS FROM NODE I
IS NODE I CONNECTED TO MORE THAN 4 LINKS?
YES
INITIALIZE N = 2 FOR TWO LINES OF OUTPUT FOR NON-DIRECTIONAL LINK VOLUMES.
NO
INITIALIZE N = 1 FOR ONE LINE OF OUTPUT FOR NON-DIRECTIONAL LINK VOLUMES
K = 1
PRINT NON-DIRECTIONAL LINK VOLUMES FOR NODE I
K = K + 1
IS K > N?
NO
YES
GET NODE NUMBERS AND TURN VOLUMES ORGANIZED TO PRINT AND COUNT NUMBER OF TURN VOLUMES
W 4

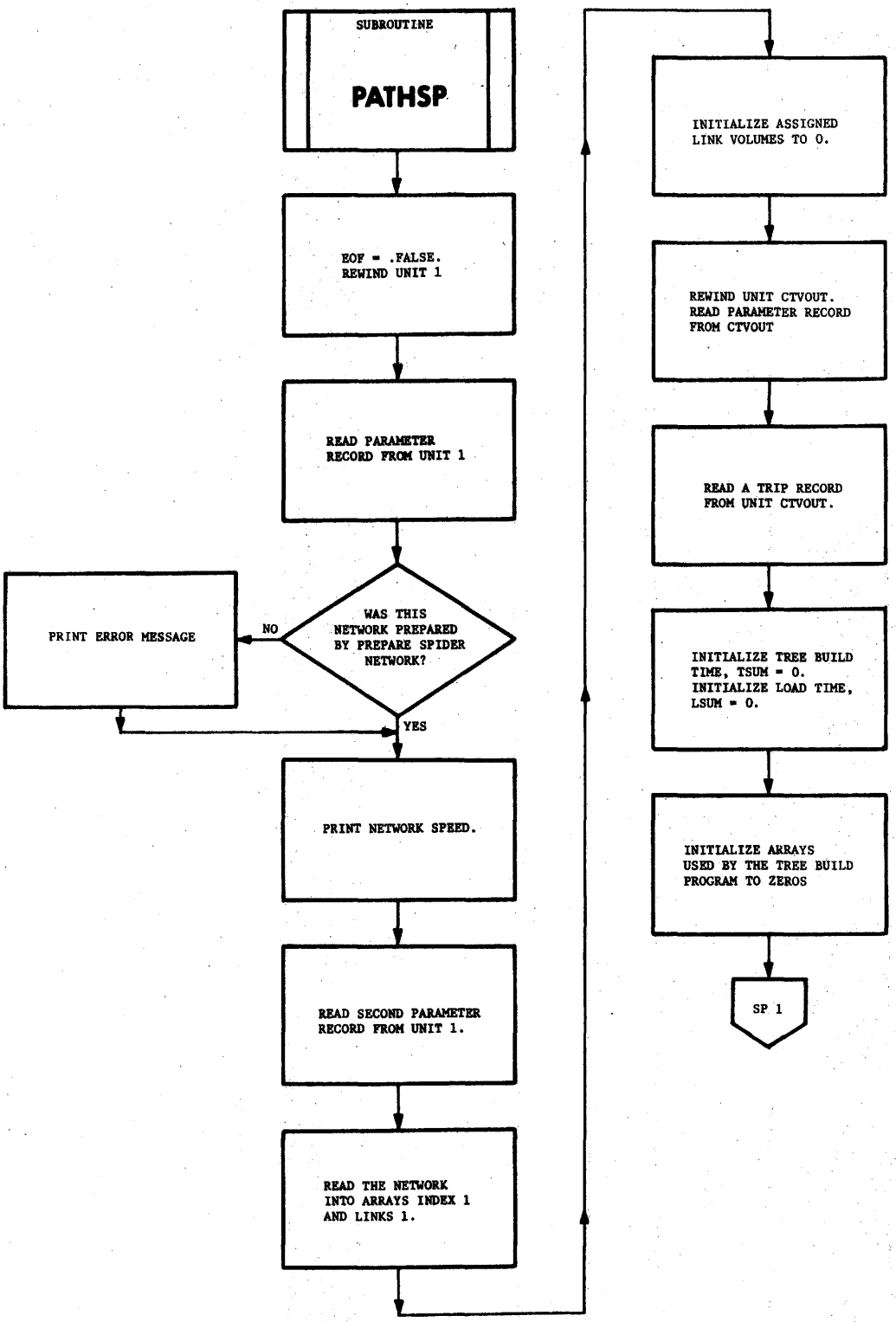

W 4
ARE THERE ANY TURN VOLUMES TO PRINT?
NO
YES
PRINT TURN VOLUMES FOR NODE I.
I = I + 1
W 2
IS I > THE LAST NODE NUMBER?
NO
W 5
YES
REWIND UNIT NETWORK
RETURN


SUBROUTINE
PATHCL
INITIALIZE SUMMATION FOR TREE BUILD AND LOAD TIME TO 0. REWIND NETWORK.
TREES?
FALSE
REWIND TRIP DATA SET
TRUE
READ PARAMETER RECORD FROM NETWORK
IS THIS THE FIRST ITERATION?
YES
PRPBLD
READ *TURN AND *TREE CARDS
NO
FIND MAXIMUM CENTROID NUMBER READ FROM *TREE CARD, SET KOUNT = MAX. CENTROID.
WRITE PARAMETER RECORD FOR SEPARATION MATRIX
TREES?
FALSE
READ PARAMETER RECORD FROM TRIP DATA SET
TRUE
SKIP HEADER RECORDS ON UNIT NETWORK
READ NODE AND LINKS RECORD AND FORM PACKED LINKS ARRAY IN CORE
.NOT. TREES .AND. SEL
TRUE
SELECT
READ SELECT CARDS, MARK SELECTED LINKS AND WRITE SELECT RECORDS ON SEL.
FALSE
INITL1
INITIALIZE ARRAYS AND CHECK MAX. LINKS, MAX. NODE AND MAX. TURNS.
TREES?
FALSE
READ FIRST TRIP RECORD. SET READSW = .FALSE. EOFSW = .FALSE.
TRUE
REWIND NETWORK
P 1

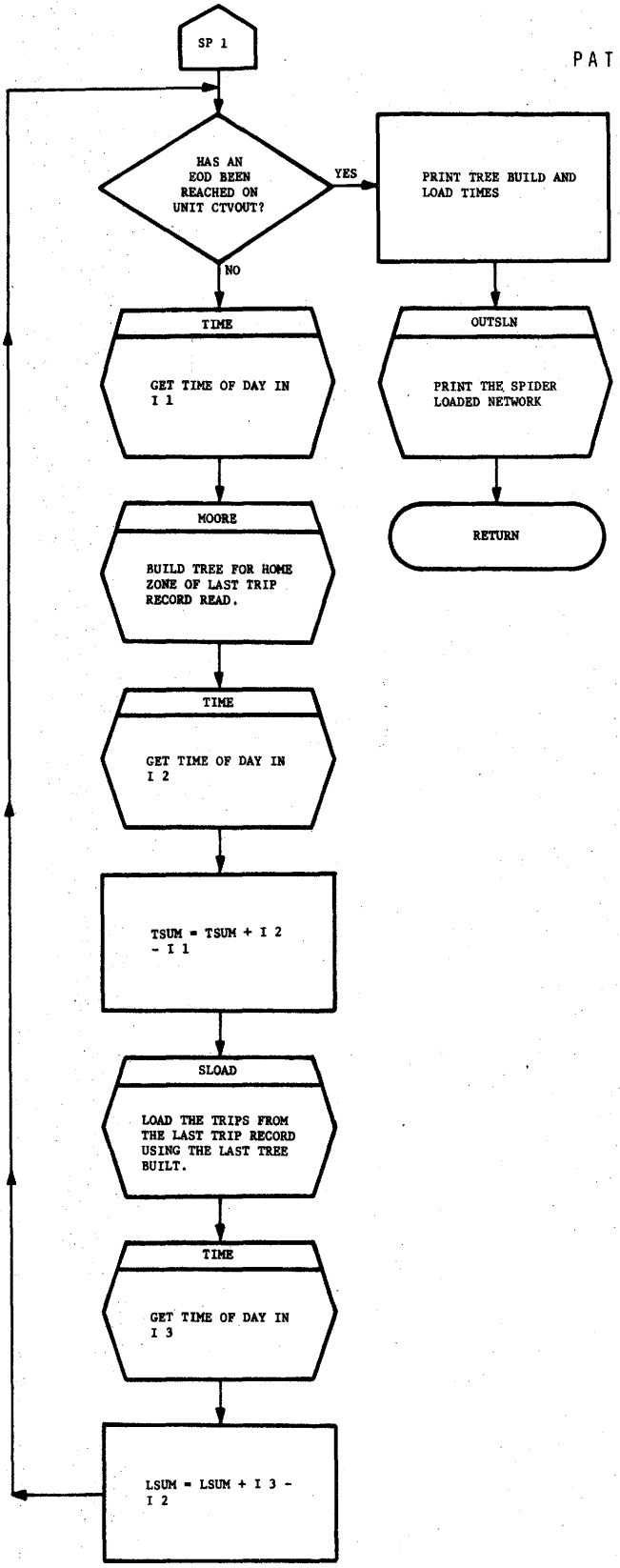

P 1
J = 1
P 5
GET FIRST CENTROID IN RANGE J, LS AND LAST CENTROID IN RANGE J, JJ = LS
P 8
GET TIME OF DAY
FASPTH
BUILD MINIMUM PATH TREE FOR CENTROID JJ
GET TIME OF DAY AND SUM TREE BUILD TIME
WRITE SEPARATION MATRIX RECORD FOR CENTROID JJ
WAS PRINTED OUTPUT FOR THIS TREE SPECIFIED?
YES
OUTTRE
PRINT TREE
NO
P 6
HAS AN END OF DATA SET BEEN ENCOUNTERED ON TRIP MATRIX
YES
NO
IS THE HOME ZONE OF THE TRIP RECORD IN CORE < JJ?
YES
NO
HAS THE TRIP RECORD IN CORE BEEN PROCESSED (IS READSW = FALSE)?
NO
YES
READ THE NEXT TRIP RECORD FROM THE TRIP MATRIX. READSW = TRUE
EOD
P 4
HOME ZONE OF TRIP RECORD: JJ
>
<
=
GET TIME OF DAY
P 3
P 2

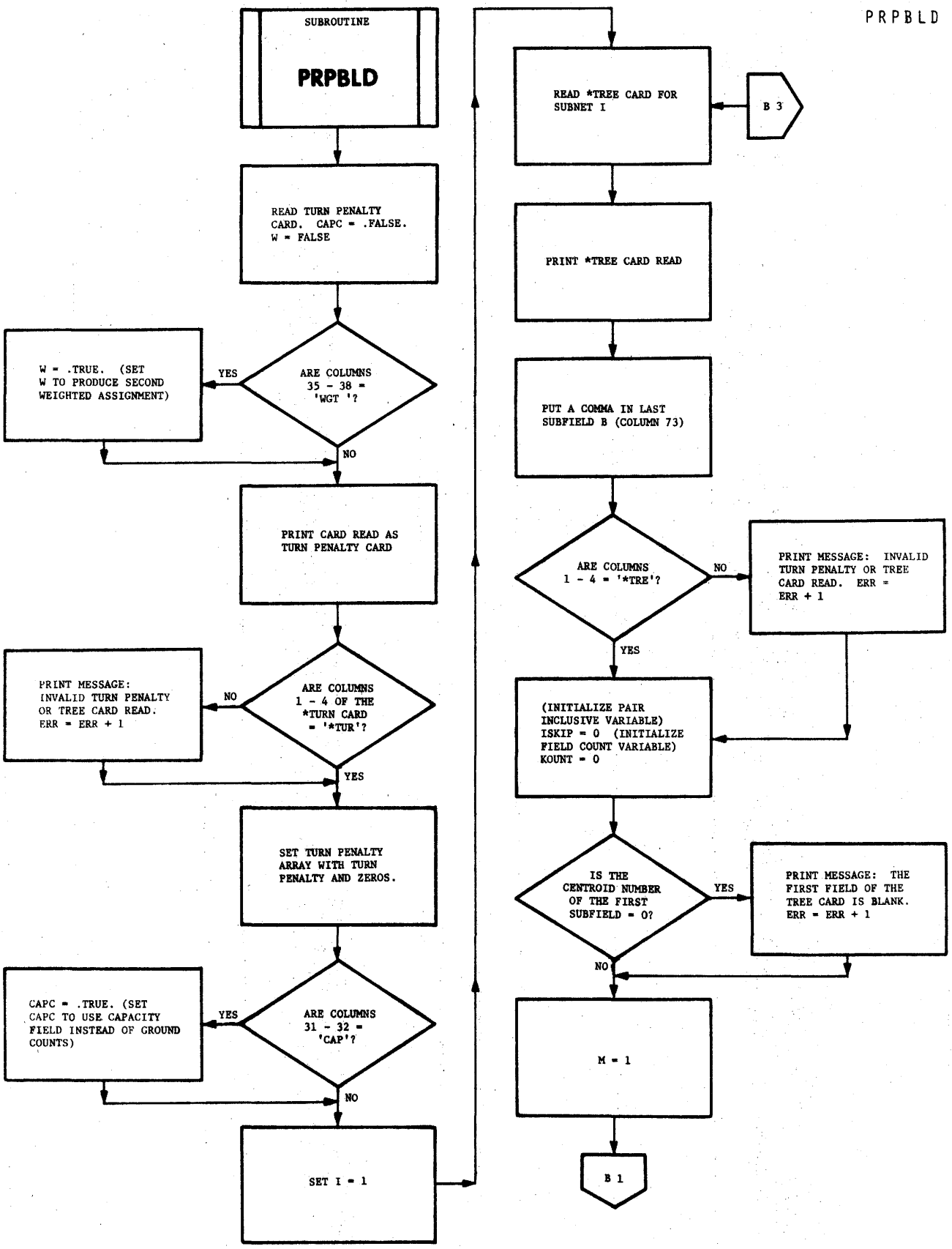

P 2
LOAD2
LOAD THE TRIP RECORD ON THE TREE AND WRITE SELECTED LINK TRIP INTERCHANGES ON SEL
YES
SEL?
NO
LOAD
LOAD THE TRIP RECORD ON THE TREE AND ALSO LOAD ANY MORE TRIP RECORDS WITH HOME ZONE JJ
GET TIME OF DAY AND SUM LOAD TIME
P 6
P 3
JJ = JJ + 1
IS JJ > LF
NO
P 8
YES
J = J + 1
IS J > K (THE NUMBER OF RANGES)?
NO
P 5
YES
END FILE SEPARATION MATRIX. REWIND SEPARATION MATRIX
PRINT TREE BUILD TIME AND LOAD TIME.
P 7
P 4
EOFSW = TRUE
ARE THERE ANY MORE TREES TO PRINT
YES
NO

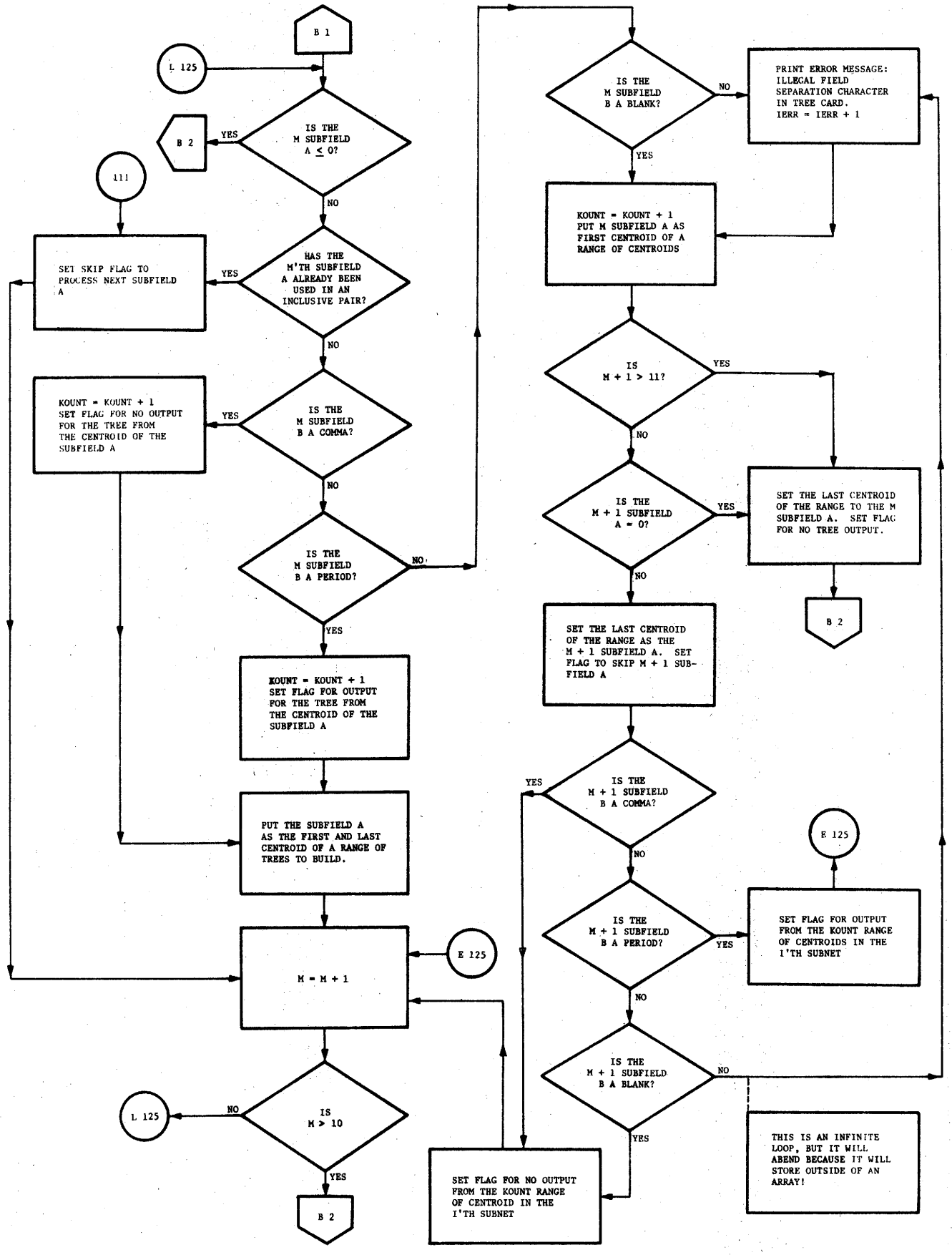

P 7
TREES?
FALSE
REWIND TRIP MATRIX
TRUE
SEL?
TRUE
CLOSE
CLOSE SEL DATA SET
FALSE
RETURN

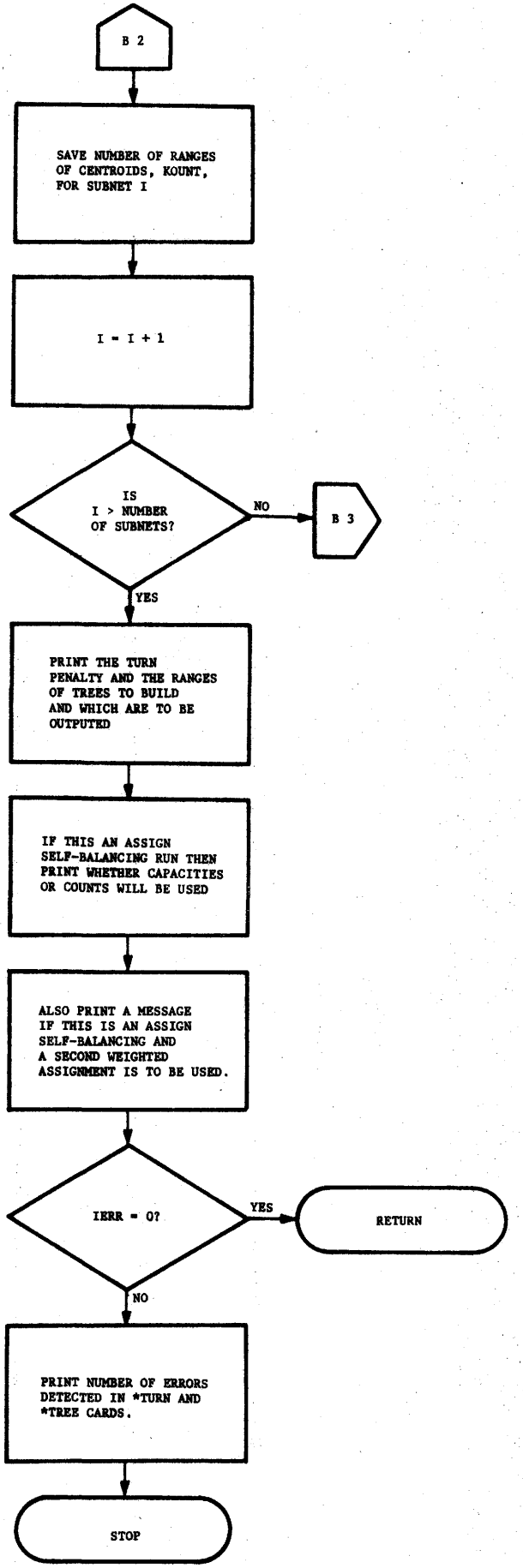

SUBROUTINE
PATHSP
EOF = .FALSE.
REWIND UNIT 1
READ PARAMETER
RECORD FROM UNIT 1
WAS THIS
NETWORK PREPARED
BY PREPARE SPIDER
NETWORK?
NO
PRINT ERROR MESSAGE
YES
PRINT NETWORK SPEED.
READ SECOND PARAMETER
RECORD FROM UNIT 1.
READ THE NETWORK
INTO ARRAYS INDEX 1
AND LINKS 1.
INITIALIZE ASSIGNED
LINK VOLUMES TO 0.
REWIND UNIT CTVOUT.
READ PARAMETER RECORD
FROM CTVOUT
READ A TRIP RECORD
FROM UNIT CTVOUT.
INITIALIZE TREE BUILD
TIME, TSUM = 0.
INITIALIZE LOAD TIME,
LSUM = 0.
INITIALIZE ARRAYS
USED BY THE TREE BUILD
PROGRAM TO ZEROS
SP 1

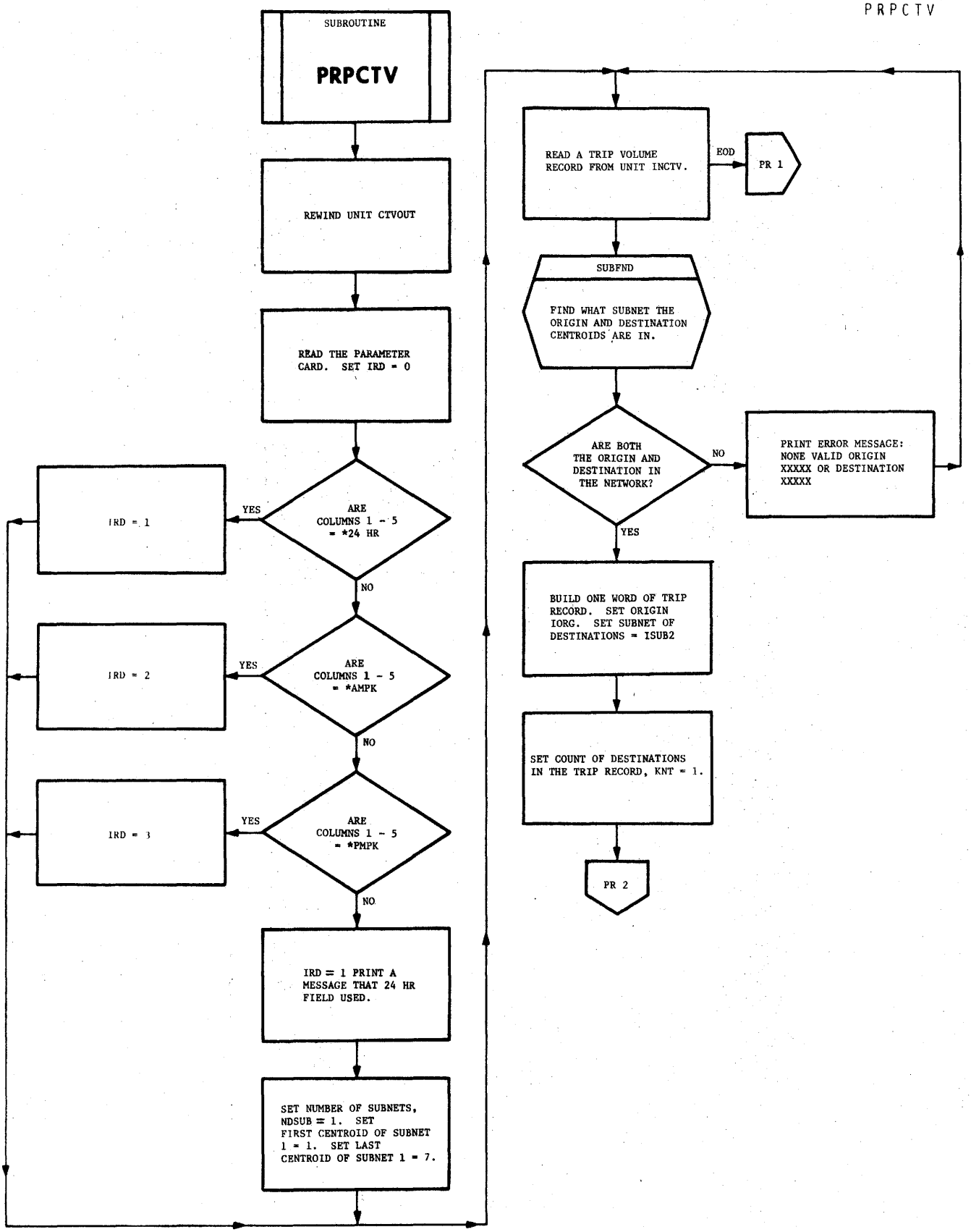
PATHSP

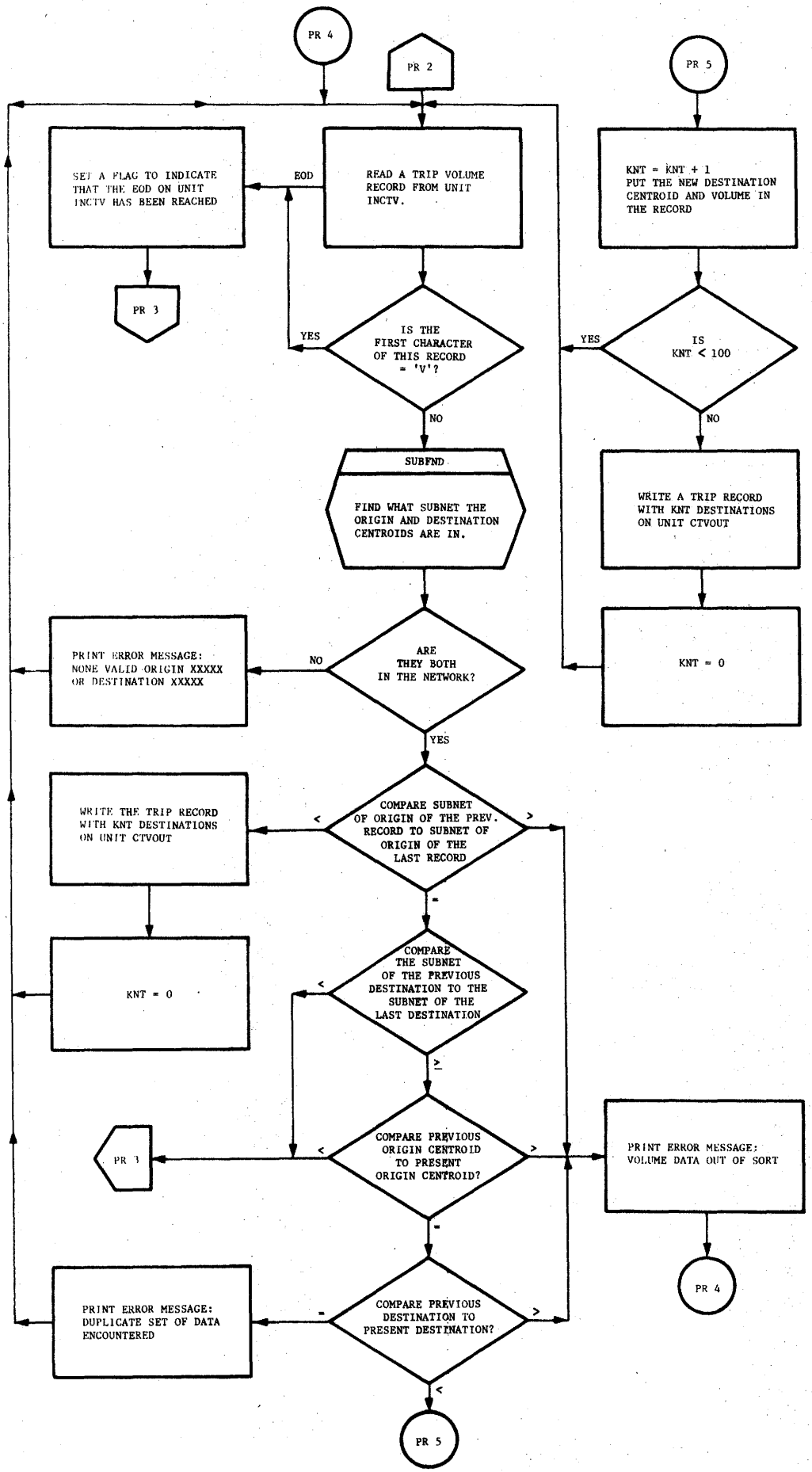

SP 1
HAS AN EOD BEEN REACHED ON UNIT CTVOUT?
YES
NO
PRINT TREE BUILD AND LOAD TIMES
TIME
GET TIME OF DAY IN I 1
OUTSLN
PRINT THE SPIDER LOADED NETWORK
RETURN
MOORE
BUILD TREE FOR HOME ZONE OF LAST TRIP RECORD READ.
TIME
GET TIME OF DAY IN I 2
TSUM = TSUM + I 2 - I 1
SLOAD
LOAD THE TRIPS FROM THE LAST TRIP RECORD USING THE LAST TREE BUILT.
TIME
GET TIME OF DAY IN I 3
LSUM = LSUM + I 3 - I 2

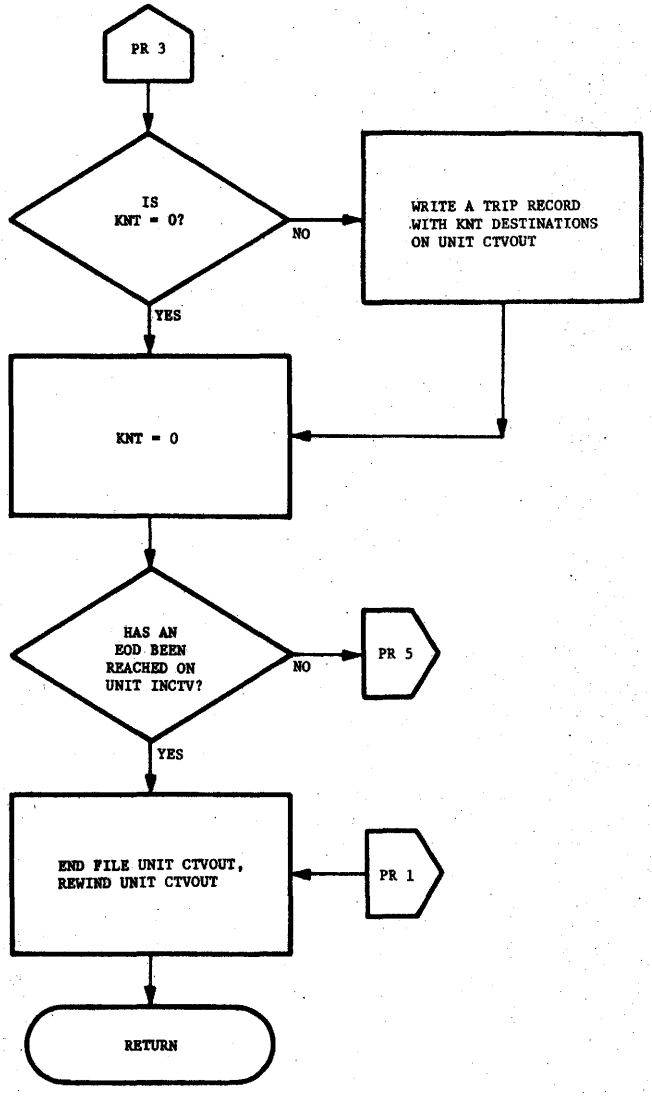

SUBROUTINE
PRPBLD
READ TURN PENALTY CARD. CAPC = .FALSE. W = FALSE
ARE COLUMNS 35 - 38 = 'WGT '?
YES
W = .TRUE. (SET W TO PRODUCE SECOND WEIGHTED ASSIGNMENT)
NO
PRINT CARD READ AS TURN PENALTY CARD
ARE COLUMNS 1 - 4 OF THE *TURN CARD = '*TUR'?
NO
PRINT MESSAGE: INVALID TURN PENALTY OR TREE CARD READ. ERR = ERR + 1
YES
SET TURN PENALTY ARRAY WITH TURN PENALTY AND ZEROS.
ARE COLUMNS 31 - 32 = 'CAP'?
YES
CAPC = .TRUE. (SET CAPC TO USE CAPACITY FIELD INSTEAD OF GROUND COUNTS)
NO
SET I = 1
READ *TREE CARD FOR SUBNET I
B 3
PRINT *TREE CARD READ
PUT A COMMA IN LAST SUBFIELD B (COLUMN 73)
ARE COLUMNS 1 - 4 = '*TRE'?
NO
PRINT MESSAGE: INVALID TURN PENALTY OR TREE CARD READ. ERR = ERR + 1
YES
(INITIALIZE PAIR INCLUSIVE VARIABLE) ISKIP = 0 (INITIALIZE FIELD COUNT VARIABLE) KOUNT = 0
IS THE CENTROID NUMBER OF THE FIRST SUBFIELD = 0?
YES
PRINT MESSAGE: THE FIRST FIELD OF THE TREE CARD IS BLANK. ERR = ERR + 1
NO
M = 1
B 1

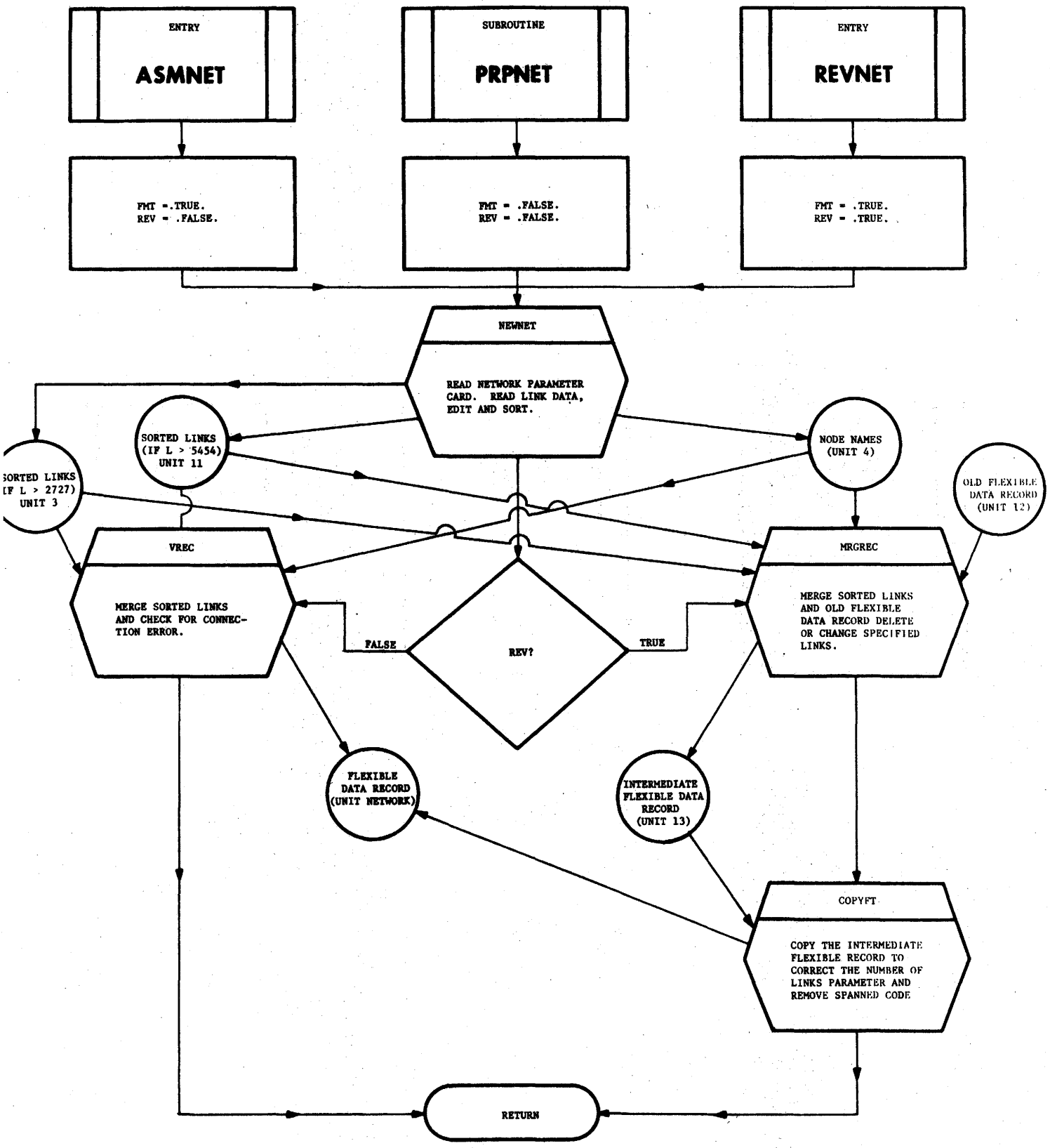

B 1
L 125
IS THE M SUBFIELD A ≤ 0?
YES
B 2
NO
111
SET SKIP FLAG TO PROCESS NEXT SUBFIELD A
HAS THE M'TH SUBFIELD A ALREADY BEEN USED IN AN INCLUSIVE PAIR?
YES
NO
KOUNT = KOUNT + 1 SET FLAG FOR NO OUTPUT FOR THE TREE FROM THE CENTROID OF THE SUBFIELD A
IS THE M SUBFIELD B A COMMA?
YES
NO
IS THE M SUBFIELD B A PERIOD?
NO
YES
KOUNT = KOUNT + 1 SET FLAG FOR OUTPUT FOR THE TREE FROM THE CENTROID OF THE SUBFIELD A
PUT THE SUBFIELD A AS THE FIRST AND LAST CENTROID OF A RANGE OF TREES TO BUILD.
M = M + 1
E 125
IS M > 10
NO
L 125
YES
B 2
IS THE M SUBFIELD B A BLANK?
NO
PRINT ERROR MESSAGE: ILLEGAL FIELD SEPARATION CHARACTER IN TREE CARD. IERR = IERR + 1
YES
KOUNT = KOUNT + 1 PUT M SUBFIELD A AS FIRST CENTROID OF A RANGE OF CENTROIDS
IS M + 1 > 11?
YES
NO
IS THE M + 1 SUBFIELD A = 0?
YES
SET THE LAST CENTROID OF THE RANGE TO THE M SUBFIELD A. SET FLAG FOR NO TREE OUTPUT.
B 2
NO
SET THE LAST CENTROID OF THE RANGE AS THE M + 1 SUBFIELD A. SET FLAG TO SKIP M + 1 SUBFIELD A
IS THE M + 1 SUBFIELD B A COMMA?
YES
NO
E 125
IS THE M + 1 SUBFIELD B A PERIOD?
YES
SET FLAG FOR OUTPUT FROM THE KOUNT RANGE OF CENTROIDS IN THE I'TH SUBNET
NO
IS THE M + 1 SUBFIELD B A BLANK?
NO
YES
THIS IS AN INFINITE LOOP, BUT IT WILL ABEND BECAUSE IT WILL STORE OUTSIDE OF AN ARRAY!
SET FLAG FOR NO OUTPUT FROM THE KOUNT RANGE OF CENTROID IN THE I'TH SUBNET

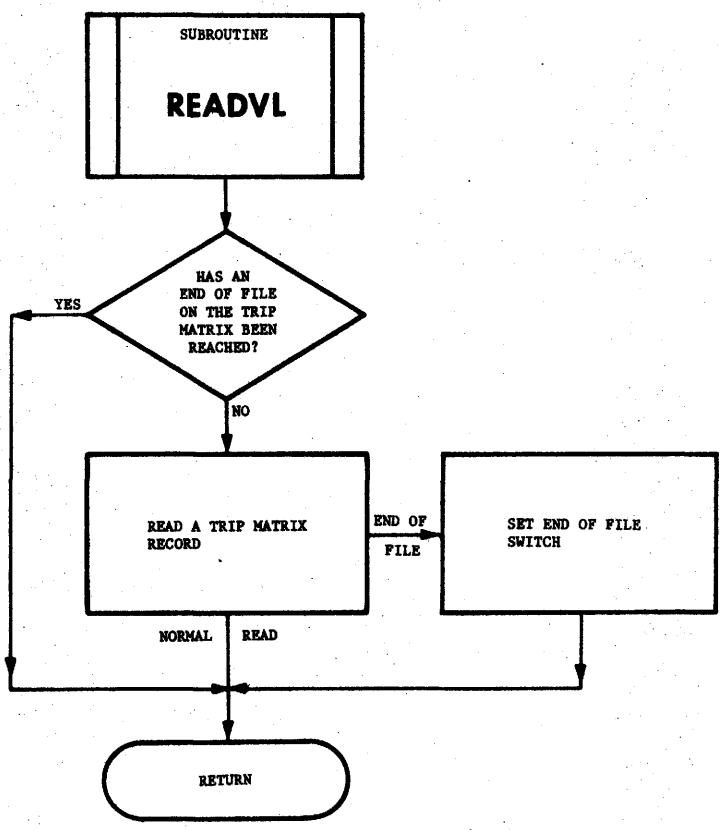

B 2
SAVE NUMBER OF RANGES OF CENTROIDS, KOUNT, FOR SUBNET I
I = I + 1
IS I > NUMBER OF SUBNETS?
NO
B 3
YES
PRINT THE TURN PENALTY AND THE RANGES OF TREES TO BUILD AND WHICH ARE TO BE OUTPUTED
IF THIS AN ASSIGN SELF-BALANCING RUN THEN PRINT WHETHER CAPACITIES OR COUNTS WILL BE USED
ALSO PRINT A MESSAGE IF THIS IS AN ASSIGN SELF-BALANCING AND A SECOND WEIGHTED ASSIGNMENT IS TO BE USED.
IERR = 0?
YES
RETURN
NO
PRINT NUMBER OF ERRORS DETECTED IN *TURN AND *TREE CARDS.
STOP

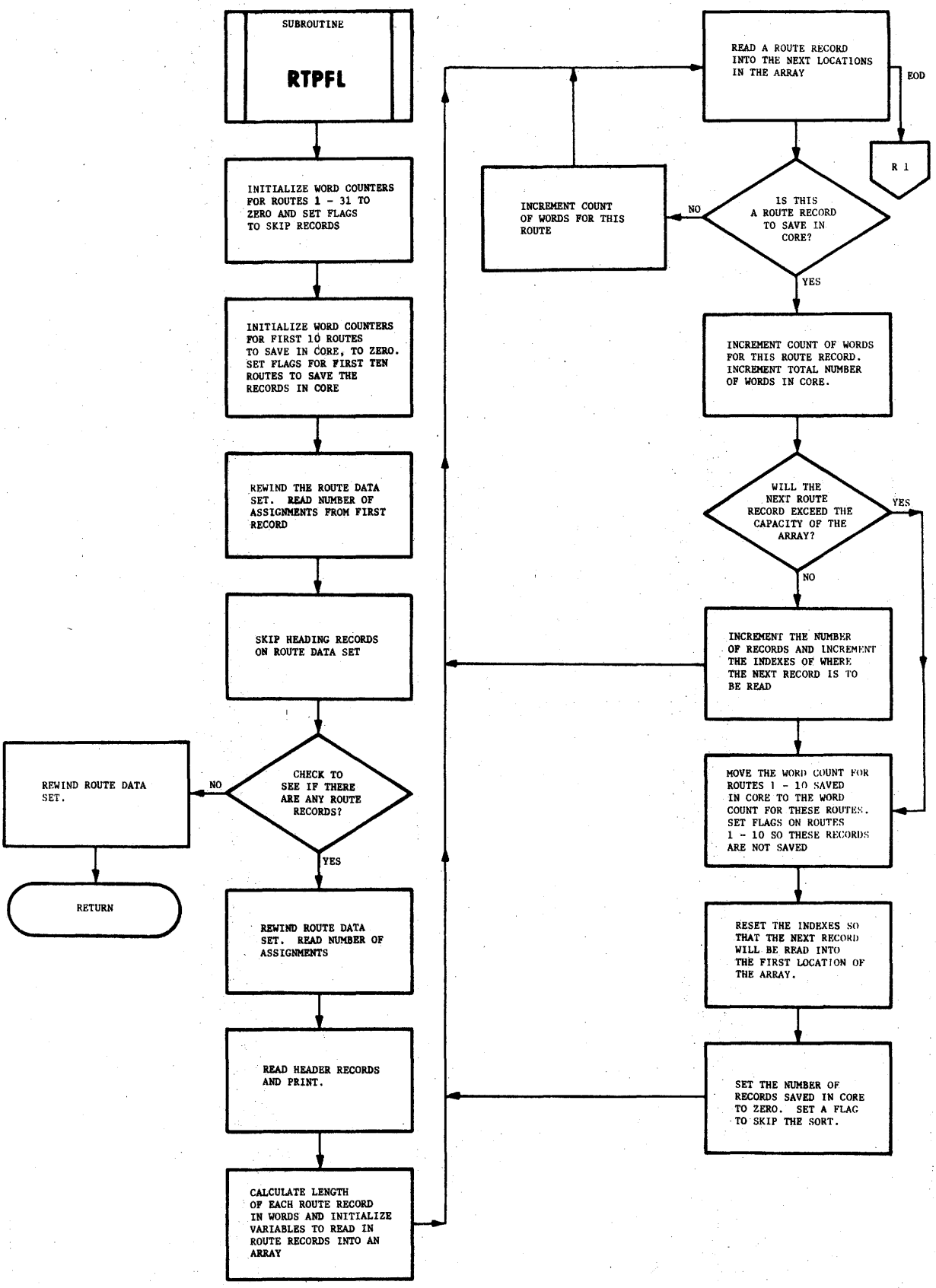

SUBROUTINE
PRPCTV
REWIND UNIT CTVOUT
READ THE PARAMETER CARD. SET IRD = 0
ARE COLUMNS 1 - 5 = *24 HR
YES
IRD = 1
NO
ARE COLUMNS 1 - 5 = *AMPK
YES
IRD = 2
NO
ARE COLUMNS 1 - 5 = *PMPK
YES
IRD = 3
NO
IRD = 1 PRINT A MESSAGE THAT 24 HR FIELD USED.
SET NUMBER OF SUBNETS, NDSUB = 1. SET FIRST CENTROID OF SUBNET 1 = 1. SET LAST CENTROID OF SUBNET 1 = 7.
READ A TRIP VOLUME RECORD FROM UNIT INCTV.
EOD
PR 1
SUBFND
FIND WHAT SUBNET THE ORIGIN AND DESTINATION CENTROIDS ARE IN.
ARE BOTH THE ORIGIN AND DESTINATION IN THE NETWORK?
NO
PRINT ERROR MESSAGE: NONE VALID ORIGIN XXXXX OR DESTINATION XXXXX
YES
BUILD ONE WORD OF TRIP RECORD. SET ORIGIN IORG. SET SUBNET OF DESTINATIONS = ISUB2
SET COUNT OF DESTINATIONS IN THE TRIP RECORD, KNT = 1.
PR 2

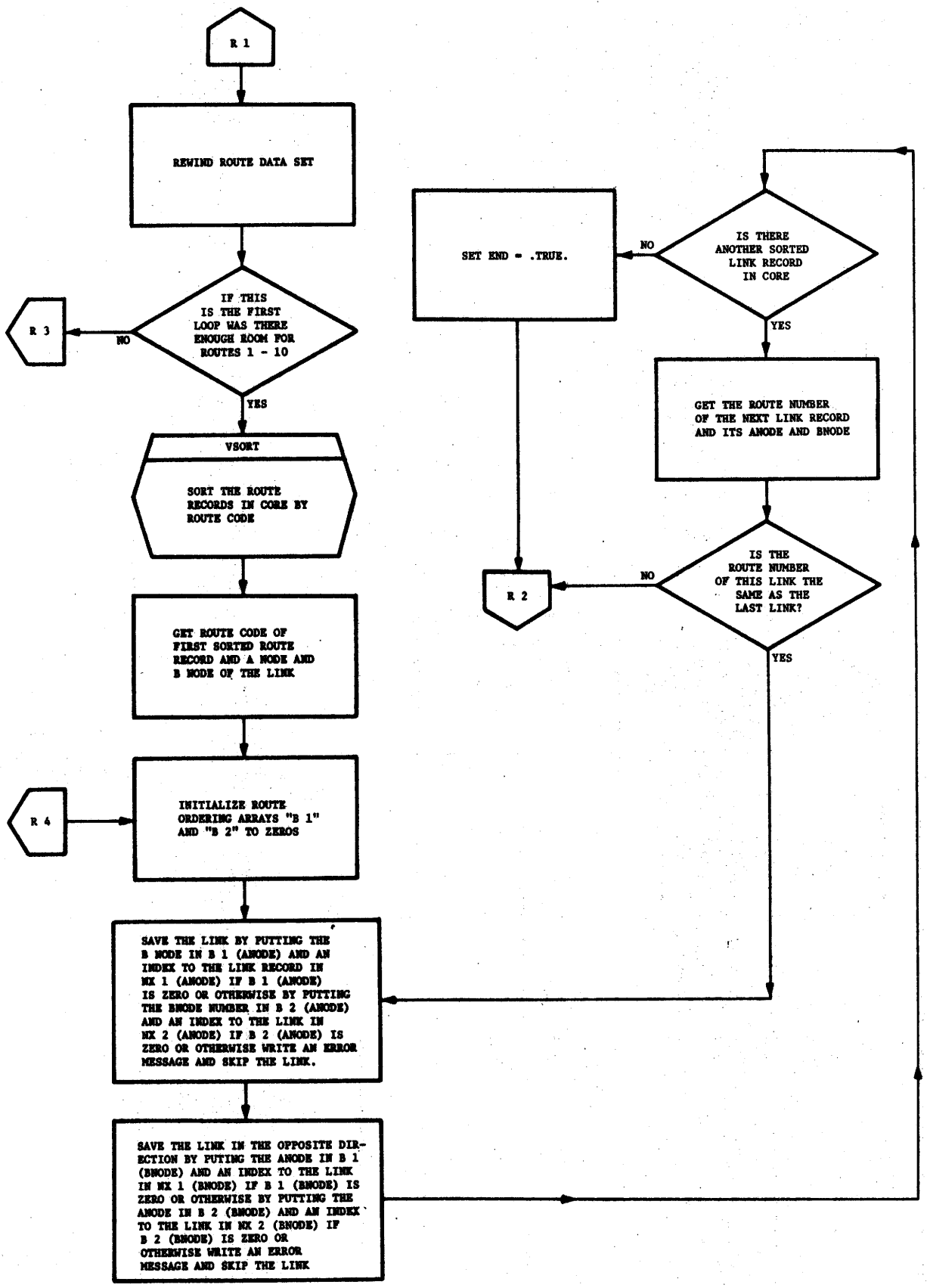

PR 4
PR 2
PR 5
SET A FLAG TO INDICATE THAT THE EOD ON UNIT INCTV HAS BEEN REACHED
EOD
READ A TRIP VOLUME RECORD FROM UNIT INCTV.
KNT = KNT + 1
PUT THE NEW DESTINATION CENTROID AND VOLUME IN THE RECORD
PR 3
YES
IS THE FIRST CHARACTER OF THIS RECORD = 'V'?
YES
IS KNT < 100
NO
NO
SUBFND
FIND WHAT SUBNET THE ORIGIN AND DESTINATION CENTROIDS ARE IN.
WRITE A TRIP RECORD WITH KNT DESTINATIONS ON UNIT CTVOUT
PRINT ERROR MESSAGE: NONE VALID ORIGIN XXXXX OR DESTINATION XXXXX
NO
ARE THEY BOTH IN THE NETWORK?
KNT = 0
YES
WRITE THE TRIP RECORD WITH KNT DESTINATIONS ON UNIT CTVOUT
<
COMPARE SUBNET OF ORIGIN OF THE PREV. RECORD TO SUBNET OF ORIGIN OF THE LAST RECORD
>
=
KNT = 0
COMPARE THE SUBNET OF THE PREVIOUS DESTINATION TO THE SUBNET OF THE LAST DESTINATION
<
≥
PR 3
COMPARE PREVIOUS ORIGIN CENTROID TO PRESENT ORIGIN CENTROID?
PRINT ERROR MESSAGE: VOLUME DATA OUT OF SORT
PR 4
=
PRINT ERROR MESSAGE: DUPLICATE SET OF DATA ENCOUNTERED
COMPARE PREVIOUS DESTINATION TO PRESENT DESTINATION?
PR 5

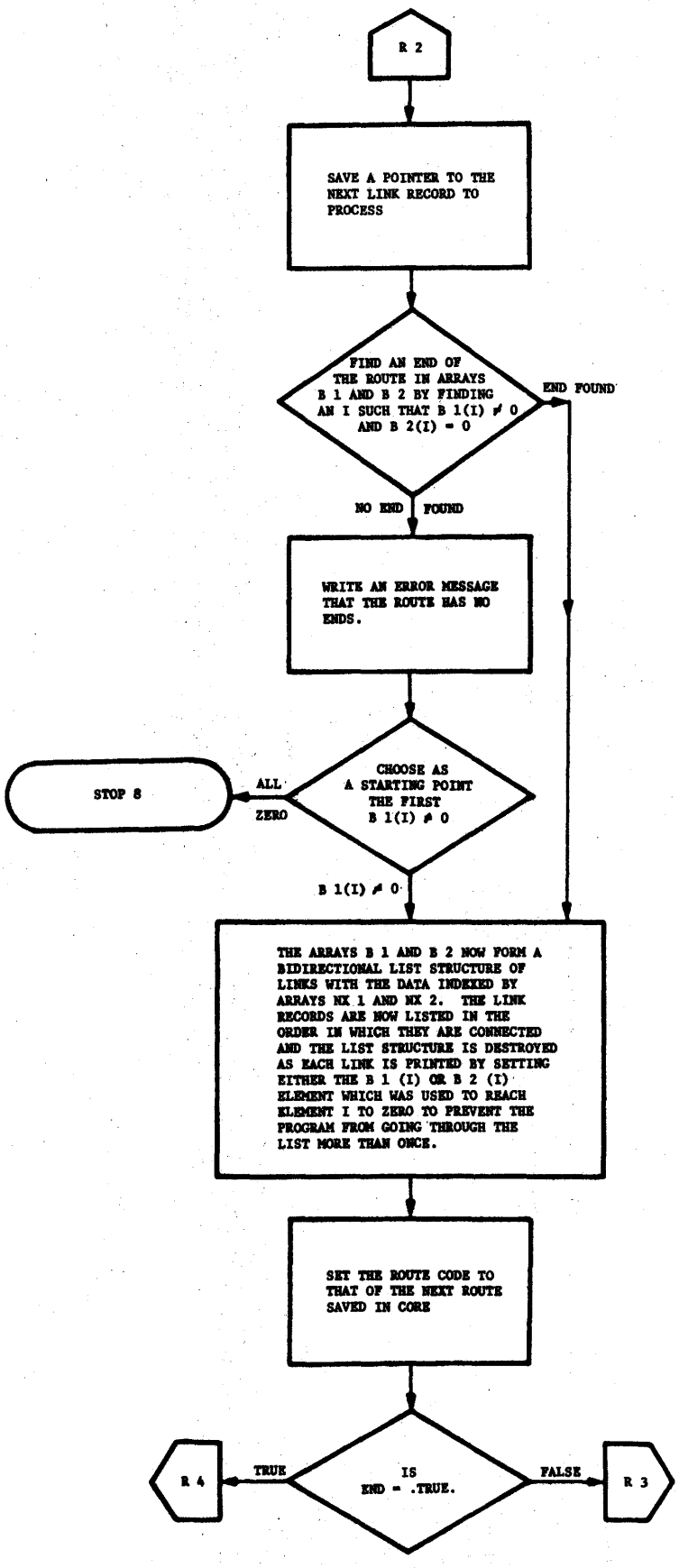

PR 3
IS
KNT = 0?
NO
WRITE A TRIP RECORD
WITH KNT DESTINATIONS
ON UNIT CTVOUT
YES
KNT = 0
HAS AN
EOD BEEN
REACHED ON
UNIT INCTV?
NO
PR 5
YES
END FILE UNIT CTVOUT,
REWIND UNIT CTVOUT
PR 1
RETURN

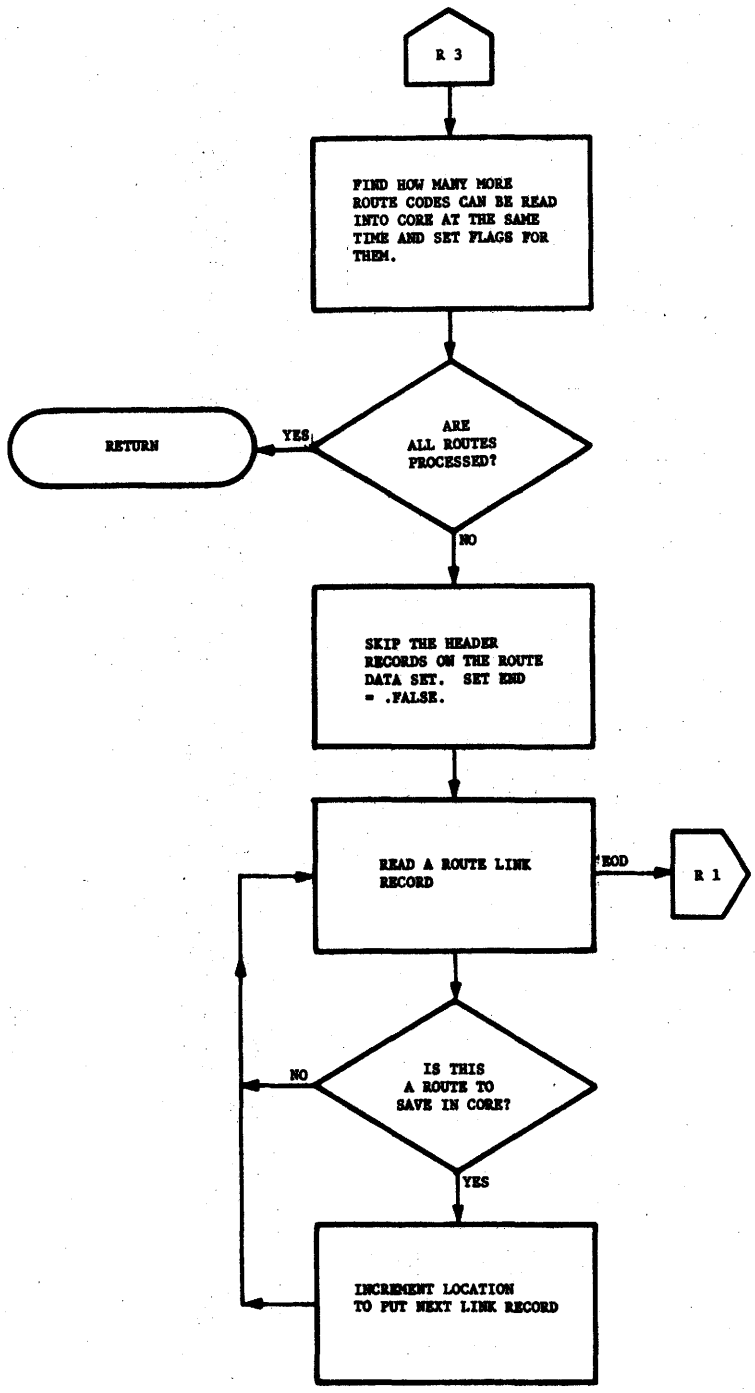

ENTRY
ASMNET
SUBROUTINE
PRPNET
ENTRY
REVNET
FMT =.TRUE.
REV = .FALSE.
FMT = .FALSE.
REV = .FALSE.
FMT = .TRUE.
REV = .TRUE.
NEWNET
READ NETWORK PARAMETER CARD. READ LINK DATA, EDIT AND SORT.
SORTED LINKS (IF L > 5454) UNIT 11
NODE NAMES (UNIT 4)
SORTED LINKS (IF L > 2727) UNIT 3
OLD FLEXIBLE DATA RECORD (UNIT 12)
VREC
MERGE SORTED LINKS AND CHECK FOR CONNEC-TION ERROR.
MRGREC
MERGE SORTED LINKS AND OLD FLEXIBLE DATA RECORD DELETE OR CHANGE SPECIFIED LINKS.
FALSE
REV?
TRUE
FLEXIBLE DATA RECORD (UNIT NETWORK)
INTERMEDIATE FLEXIBLE DATA RECORD (UNIT 13)
COPYFT
COPY THE INTERMEDIATE FLEXIBLE RECORD TO CORRECT THE NUMBER OF LINKS PARAMETER AND REMOVE SPANNED CODE
RETURN

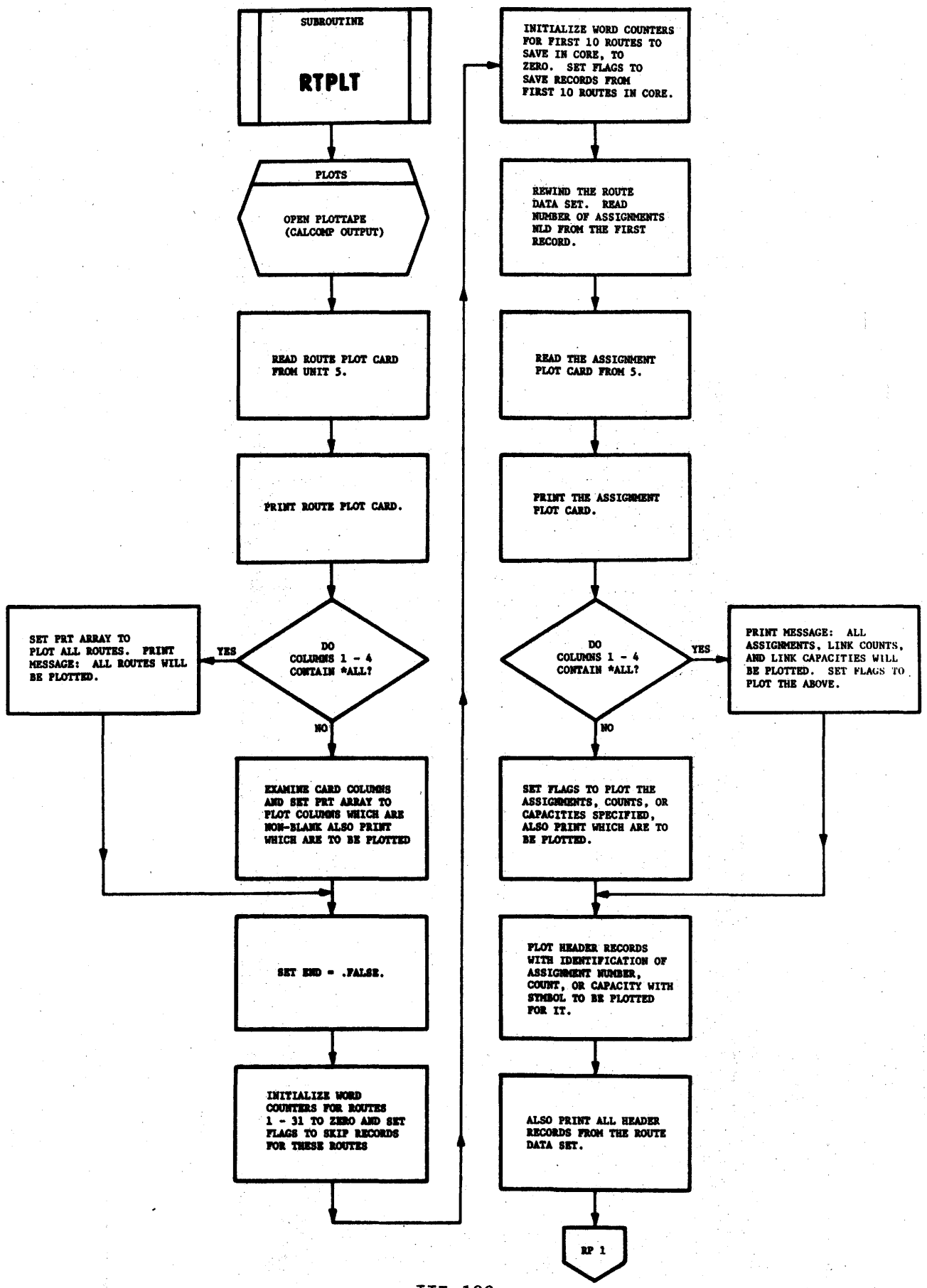

SUBROUTINE
READVL
HAS AN END OF FILE ON THE TRIP MATRIX BEEN REACHED?
YES
NO
READ A TRIP MATRIX RECORD
END OF FILE
SET END OF FILE SWITCH
NORMAL READ
RETURN

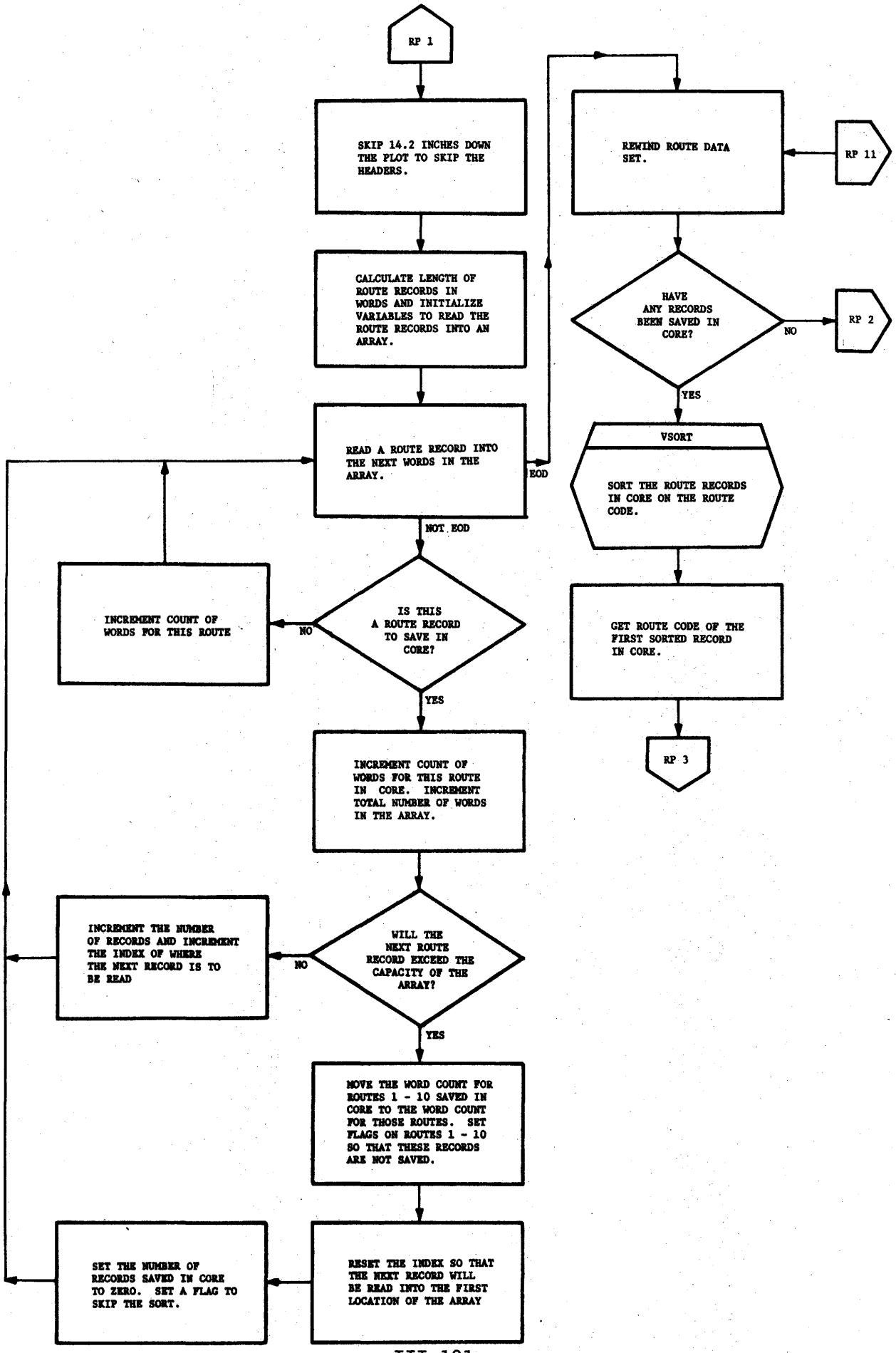

SUBROUTINE
RTPFL
INITIALIZE WORD COUNTERS FOR ROUTES 1 - 31 TO ZERO AND SET FLAGS TO SKIP RECORDS
INITIALIZE WORD COUNTERS FOR FIRST 10 ROUTES TO SAVE IN CORE, TO ZERO. SET FLAGS FOR FIRST TEN ROUTES TO SAVE THE RECORDS IN CORE
REWIND THE ROUTE DATA SET. READ NUMBER OF ASSIGNMENTS FROM FIRST RECORD
SKIP HEADING RECORDS ON ROUTE DATA SET
CHECK TO SEE IF THERE ARE ANY ROUTE RECORDS?
NO
REWIND ROUTE DATA SET.
RETURN
YES
REWIND ROUTE DATA SET. READ NUMBER OF ASSIGNMENTS
READ HEADER RECORDS AND PRINT.
CALCULATE LENGTH OF EACH ROUTE RECORD IN WORDS AND INITIALIZE VARIABLES TO READ IN ROUTE RECORDS INTO AN ARRAY
READ A ROUTE RECORD INTO THE NEXT LOCATIONS IN THE ARRAY
EOD
R 1
IS THIS A ROUTE RECORD TO SAVE IN CORE?
NO
INCREMENT COUNT OF WORDS FOR THIS ROUTE
YES
INCREMENT COUNT OF WORDS FOR THIS ROUTE RECORD. INCREMENT TOTAL NUMBER OF WORDS IN CORE.
WILL THE NEXT ROUTE RECORD EXCEED THE CAPACITY OF THE ARRAY?
YES
NO
INCREMENT THE NUMBER OF RECORDS AND INCREMENT THE INDEXES OF WHERE THE NEXT RECORD IS TO BE READ
MOVE THE WORD COUNT FOR ROUTES 1 - 10 SAVED IN CORE TO THE WORD COUNT FOR THESE ROUTES. SET FLAGS ON ROUTES 1 - 10 SO THESE RECORDS ARE NOT SAVED
RESET THE INDEXES SO THAT THE NEXT RECORD WILL BE READ INTO THE FIRST LOCATION OF THE ARRAY.
SET THE NUMBER OF RECORDS SAVED IN CORE TO ZERO. SET A FLAG TO SKIP THE SORT.

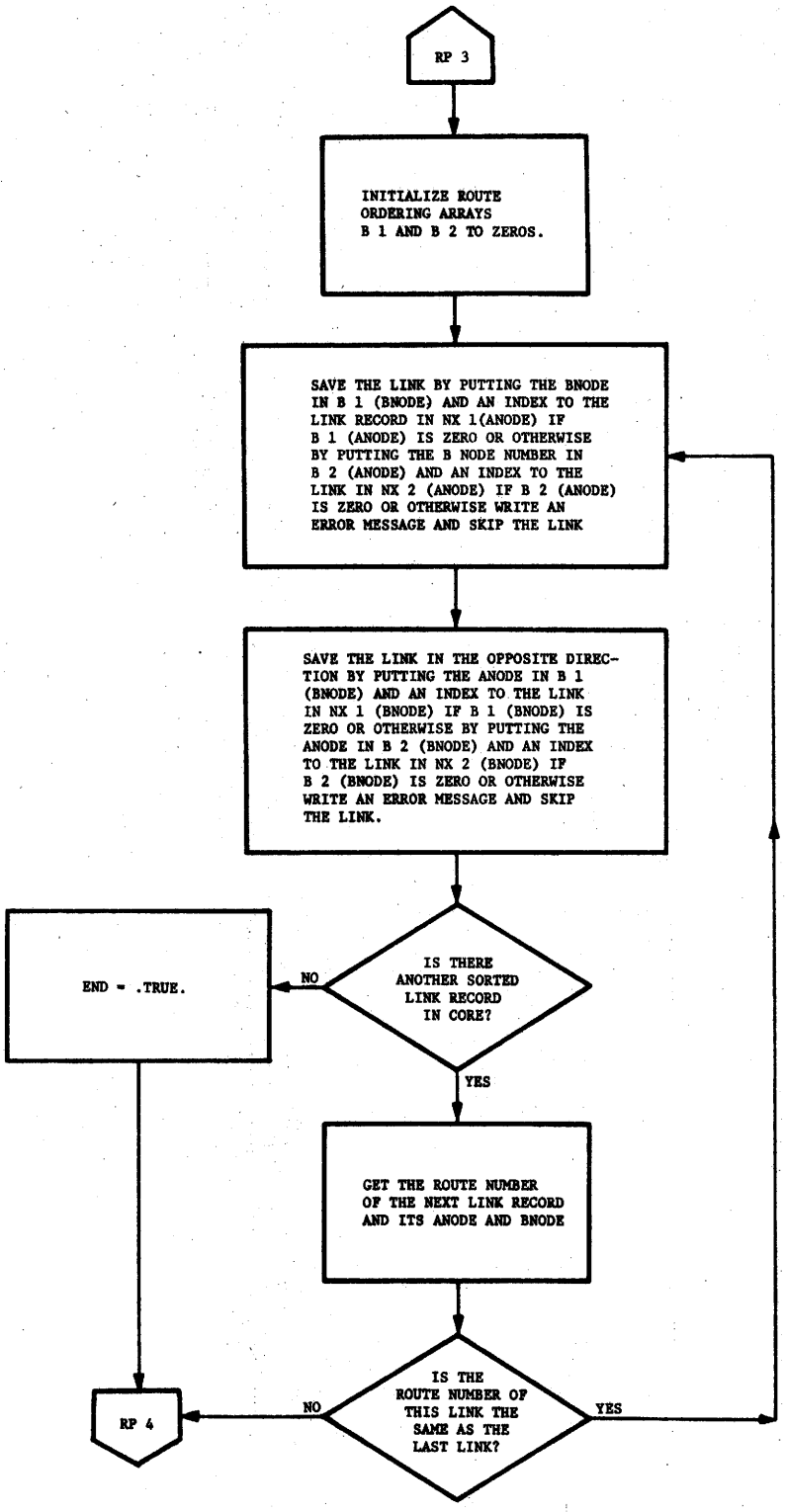

R 1
REWIND ROUTE DATA SET
IF THIS IS THE FIRST LOOP WAS THERE ENOUGH ROOM FOR ROUTES 1 - 10
NO
R 3
YES
VSORT
SORT THE ROUTE RECORDS IN CORE BY ROUTE CODE
GET ROUTE CODE OF FIRST SORTED ROUTE RECORD AND A NODE AND B NODE OF THE LINK
R 4
INITIALIZE ROUTE ORDERING ARRAYS "B 1" AND "B 2" TO ZEROS
SAVE THE LINK BY PUTTING THE B NODE IN B 1 (ANODE) AND AN INDEX TO THE LINK RECORD IN NX 1 (ANODE) IF B 1 (ANODE) IS ZERO OR OTHERWISE BY PUTTING THE BNODE NUMBER IN B 2 (ANODE) AND AN INDEX TO THE LINK IN NX 2 (ANODE) IF B 2 (ANODE) IS ZERO OR OTHERWISE WRITE AN ERROR MESSAGE AND SKIP THE LINK.
SAVE THE LINK IN THE OPPOSITE DIRECTION BY PUTTING THE ANODE IN B 1 (BNODE) AND AN INDEX TO THE LINK IN NX 1 (BNODE) IF B 1 (BNODE) IS ZERO OR OTHERWISE BY PUTTING THE ANODE IN B 2 (BNODE) AND AN INDEX TO THE LINK IN NX 2 (BNODE) IF B 2 (BNODE) IS ZERO OR OTHERWISE WRITE AN ERROR MESSAGE AND SKIP THE LINK
IS THE ROUTE NUMBER OF THIS LINK THE SAME AS THE LAST LINK?
NO
R 2
YES
GET THE ROUTE NUMBER OF THE NEXT LINK RECORD AND ITS ANODE AND BNODE
IS THERE ANOTHER SORTED LINK RECORD IN CORE
NO
SET END = .TRUE.
YES


R 2
SAVE A POINTER TO THE NEXT LINK RECORD TO PROCESS
FIND AN END OF THE ROUTE IN ARRAYS B 1 AND B 2 BY FINDING AN I SUCH THAT B 1(I) ≠ 0 AND B 2(I) = 0
END FOUND
NO END FOUND
WRITE AN ERROR MESSAGE THAT THE ROUTE HAS NO ENDS.
CHOOSE AS A STARTING POINT THE FIRST B 1(I) ≠ 0
ALL ZERO
STOP 8
B 1(I) ≠ 0
THE ARRAYS B 1 AND B 2 NOW FORM A BIDIRECTIONAL LIST STRUCTURE OF LINKS WITH THE DATA INDEXED BY ARRAYS NX 1 AND NX 2. THE LINK RECORDS ARE NOW LISTED IN THE ORDER IN WHICH THEY ARE CONNECTED AND THE LIST STRUCTURE IS DESTROYED AS EACH LINK IS PRINTED BY SETTING EITHER THE B 1 (I) OR B 2 (I) ELEMENT WHICH WAS USED TO REACH ELEMENT I TO ZERO TO PREVENT THE PROGRAM FROM GOING THROUGH THE LIST MORE THAN ONCE.
SET THE ROUTE CODE TO THAT OF THE NEXT ROUTE SAVED IN CORE
IS END = .TRUE.
TRUE
FALSE
R 4
R 3

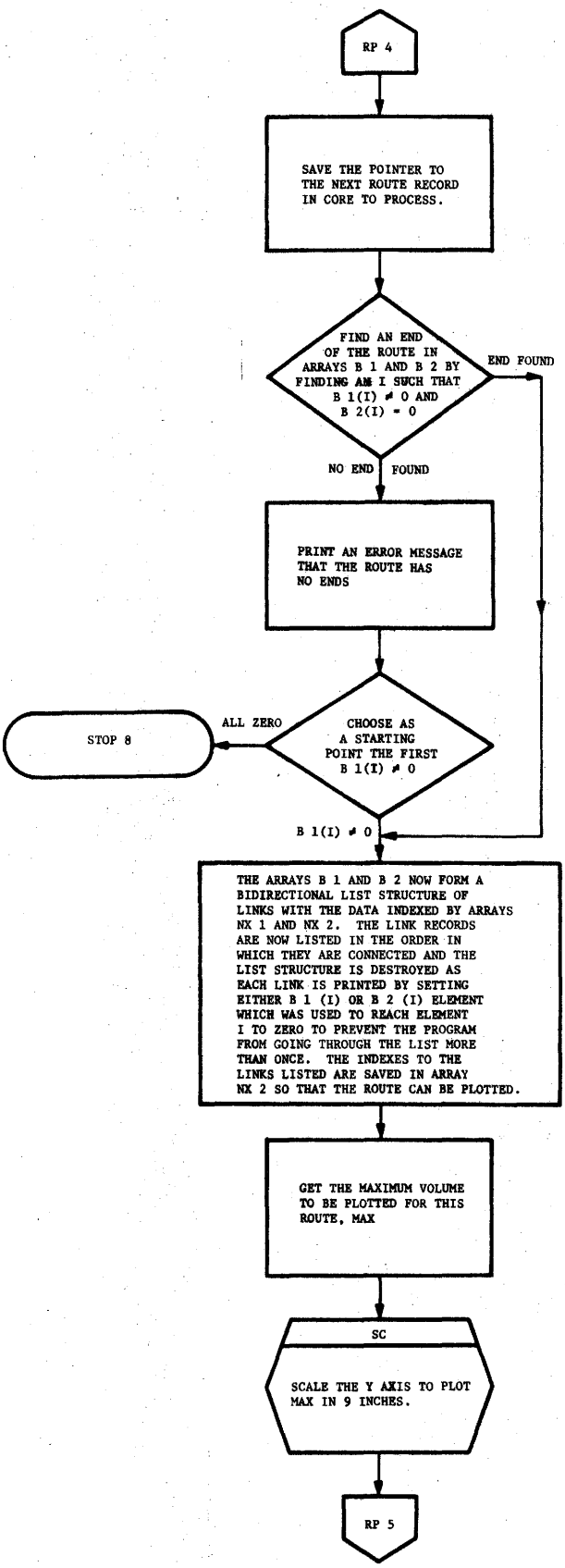

R 3
FIND HOW MANY MORE ROUTE CODES CAN BE READ INTO CORE AT THE SAME TIME AND SET FLAGS FOR THEM.
ARE ALL ROUTES PROCESSED?
YES
RETURN
NO
SKIP THE HEADER RECORDS ON THE ROUTE DATA SET. SET END = .FALSE.
READ A ROUTE LINK RECORD
*EOD
R 1
IS THIS A ROUTE TO SAVE IN CORE?
NO
YES
INCREMENT LOCATION TO PUT NEXT LINK RECORD

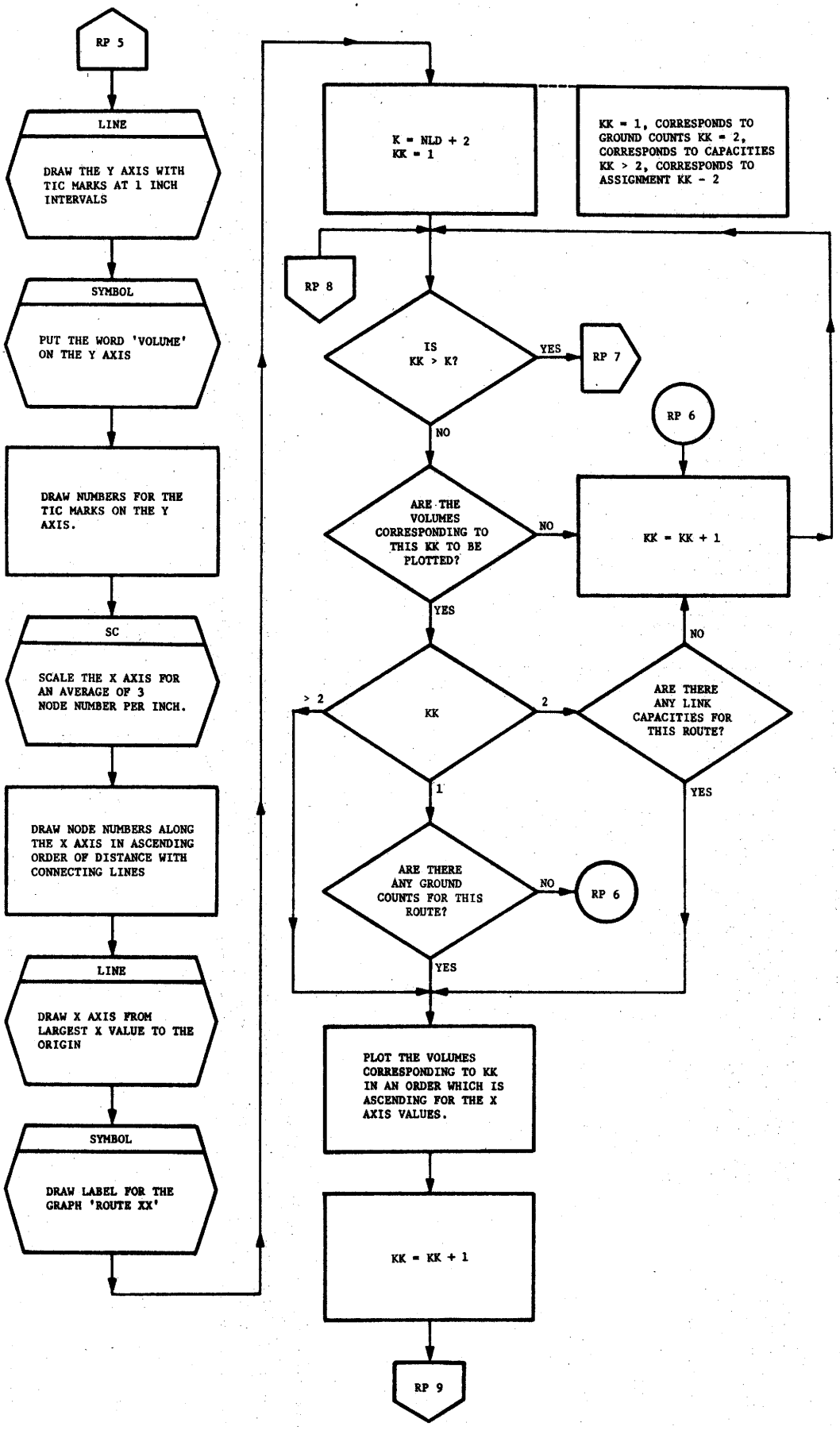

SUBROUTINE
RTPLT
PLOTS
OPEN PLOTTAPE (CALCOMP OUTPUT)
READ ROUTE PLOT CARD FROM UNIT 5.
PRINT ROUTE PLOT CARD.
DO COLUMNS 1 - 4 CONTAIN *ALL?
YES
SET PRT ARRAY TO PLOT ALL ROUTES. PRINT MESSAGE: ALL ROUTES WILL BE PLOTTED.
NO
EXAMINE CARD COLUMNS AND SET PRT ARRAY TO PLOT COLUMNS WHICH ARE NON-BLANK ALSO PRINT WHICH ARE TO BE PLOTTED
SET END = .FALSE.
INITIALIZE WORD COUNTERS FOR ROUTES 1 - 31 TO ZERO AND SET FLAGS TO SKIP RECORDS FOR THESE ROUTES
INITIALIZE WORD COUNTERS FOR FIRST 10 ROUTES TO SAVE IN CORE, TO ZERO. SET FLAGS TO SAVE RECORDS FROM FIRST 10 ROUTES IN CORE.
REWIND THE ROUTE DATA SET. READ NUMBER OF ASSIGNMENTS NLD FROM THE FIRST RECORD.
READ THE ASSIGNMENT PLOT CARD FROM 5.
PRINT THE ASSIGNMENT PLOT CARD.
DO COLUMNS 1 - 4 CONTAIN *ALL?
YES
PRINT MESSAGE: ALL ASSIGNMENTS, LINK COUNTS, AND LINK CAPACITIES WILL BE PLOTTED. SET FLAGS TO PLOT THE ABOVE.
NO
SET FLAGS TO PLOT THE ASSIGNMENTS, COUNTS, OR CAPACITIES SPECIFIED, ALSO PRINT WHICH ARE TO BE PLOTTED.
PLOT HEADER RECORDS WITH IDENTIFICATION OF ASSIGNMENT NUMBER, COUNT, OR CAPACITY WITH SYMBOL TO BE PLOTTED FOR IT.
ALSO PRINT ALL HEADER RECORDS FROM THE ROUTE DATA SET.
RP 1

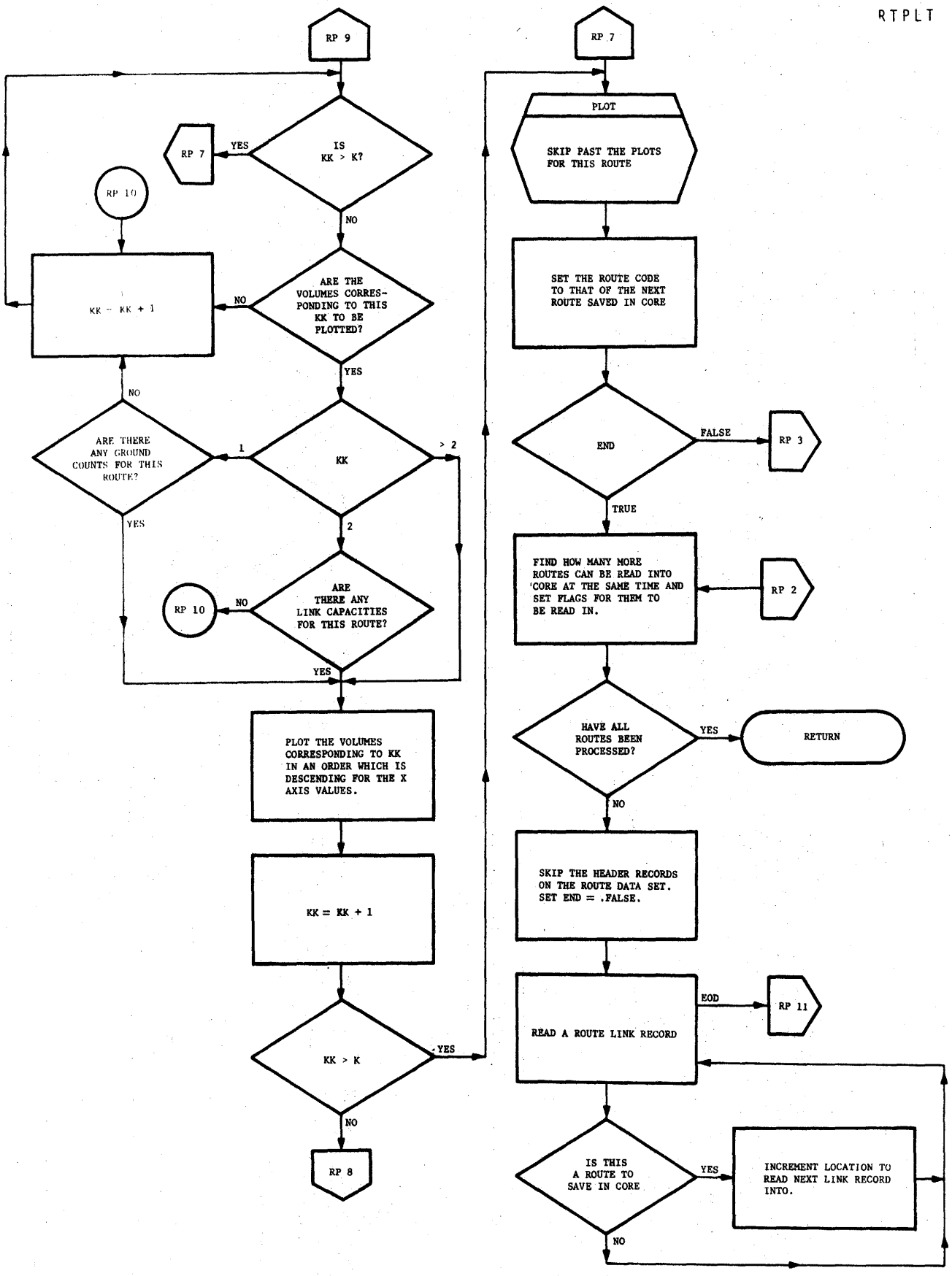

RP 1
SKIP 14.2 INCHES DOWN THE PLOT TO SKIP THE HEADERS.
CALCULATE LENGTH OF ROUTE RECORDS IN WORDS AND INITIALIZE VARIABLES TO READ THE ROUTE RECORDS INTO AN ARRAY.
READ A ROUTE RECORD INTO THE NEXT WORDS IN THE ARRAY.
EOD
NOT EOD
IS THIS A ROUTE RECORD TO SAVE IN CORE?
NO
INCREMENT COUNT OF WORDS FOR THIS ROUTE
YES
INCREMENT COUNT OF WORDS FOR THIS ROUTE IN CORE. INCREMENT TOTAL NUMBER OF WORDS IN THE ARRAY.
WILL THE NEXT ROUTE RECORD EXCEED THE CAPACITY OF THE ARRAY?
NO
INCREMENT THE NUMBER OF RECORDS AND INCREMENT THE INDEX OF WHERE THE NEXT RECORD IS TO BE READ
YES
MOVE THE WORD COUNT FOR ROUTES 1 - 10 SAVED IN CORE TO THE WORD COUNT FOR THOSE ROUTES. SET FLAGS ON ROUTES 1 - 10 SO THAT THESE RECORDS ARE NOT SAVED.
RESET THE INDEX SO THAT THE NEXT RECORD WILL BE READ INTO THE FIRST LOCATION OF THE ARRAY
SET THE NUMBER OF RECORDS SAVED IN CORE TO ZERO. SET A FLAG TO SKIP THE SORT.
REWIND ROUTE DATA SET.
RP 11
HAVE ANY RECORDS BEEN SAVED IN CORE?
NO
RP 2
YES
VSORT
SORT THE ROUTE RECORDS IN CORE ON THE ROUTE CODE.
GET ROUTE CODE OF THE FIRST SORTED RECORD IN CORE.
RP 3

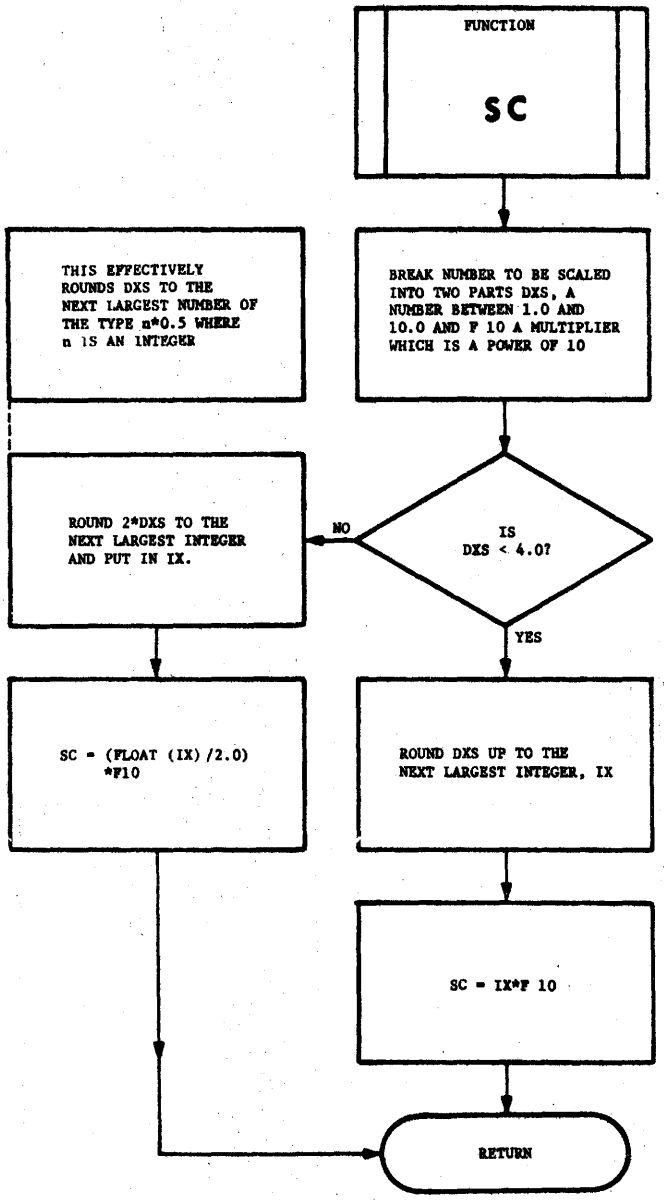

RP 3
INITIALIZE ROUTE ORDERING ARRAYS B 1 AND B 2 TO ZEROS.
SAVE THE LINK BY PUTTING THE BNODE IN B 1 (BNODE) AND AN INDEX TO THE LINK RECORD IN NX 1(ANODE) IF B 1 (ANODE) IS ZERO OR OTHERWISE BY PUTTING THE B NODE NUMBER IN B 2 (ANODE) AND AN INDEX TO THE LINK IN NX 2 (ANODE) IF B 2 (ANODE) IS ZERO OR OTHERWISE WRITE AN ERROR MESSAGE AND SKIP THE LINK
SAVE THE LINK IN THE OPPOSITE DIRECTION BY PUTTING THE ANODE IN B 1 (BNODE) AND AN INDEX TO THE LINK IN NX 1 (BNODE) IF B 1 (BNODE) IS ZERO OR OTHERWISE BY PUTTING THE ANODE IN B 2 (BNODE) AND AN INDEX TO THE LINK IN NX 2 (BNODE) IF B 2 (BNODE) IS ZERO OR OTHERWISE WRITE AN ERROR MESSAGE AND SKIP THE LINK.
IS THERE ANOTHER SORTED LINK RECORD IN CORE?
NO
END = .TRUE.
YES
GET THE ROUTE NUMBER OF THE NEXT LINK RECORD AND ITS ANODE AND BNODE
IS THE ROUTE NUMBER OF THIS LINK THE SAME AS THE LAST LINK?
NO
YES
RP 4

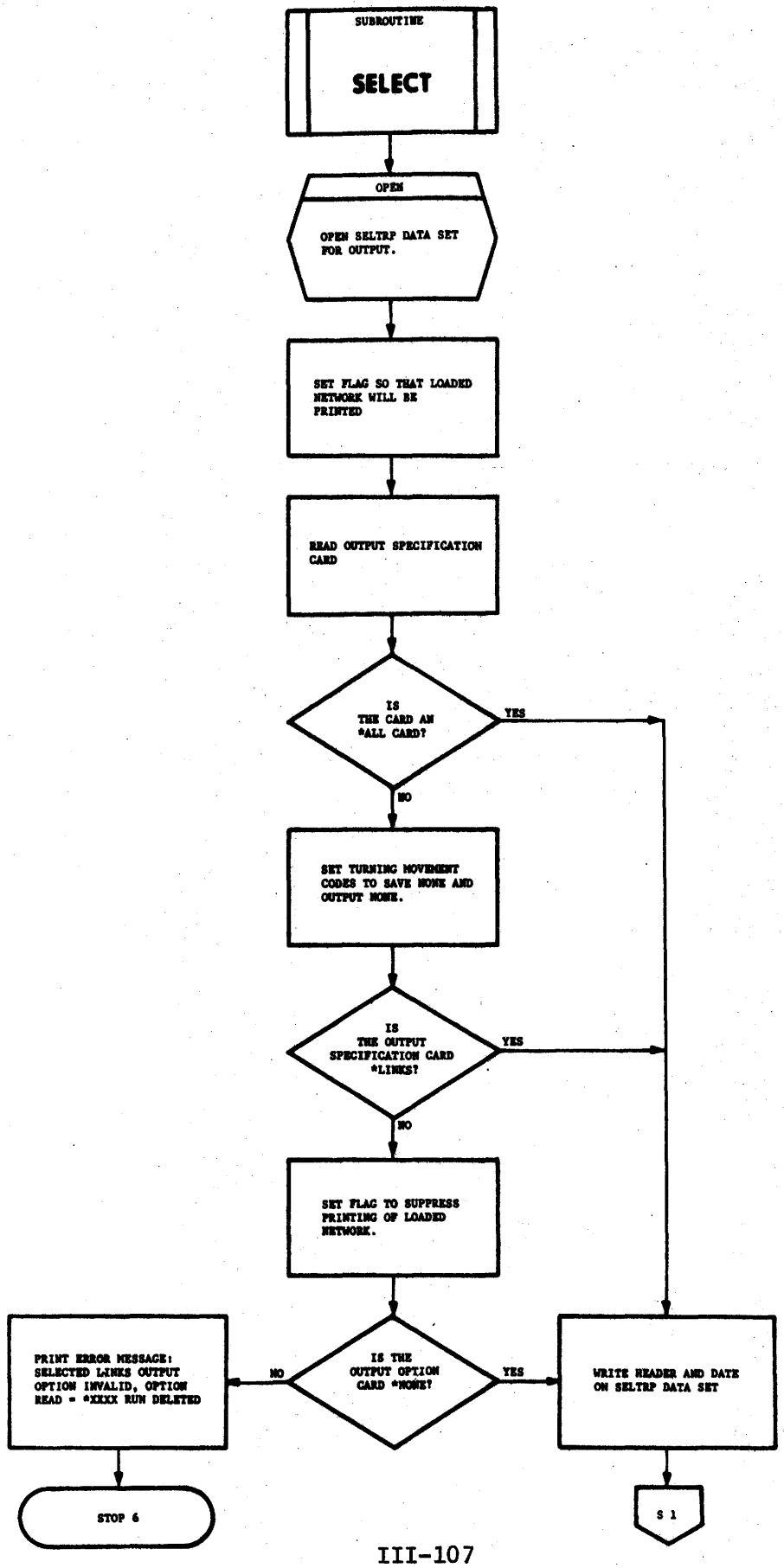
RTPLT

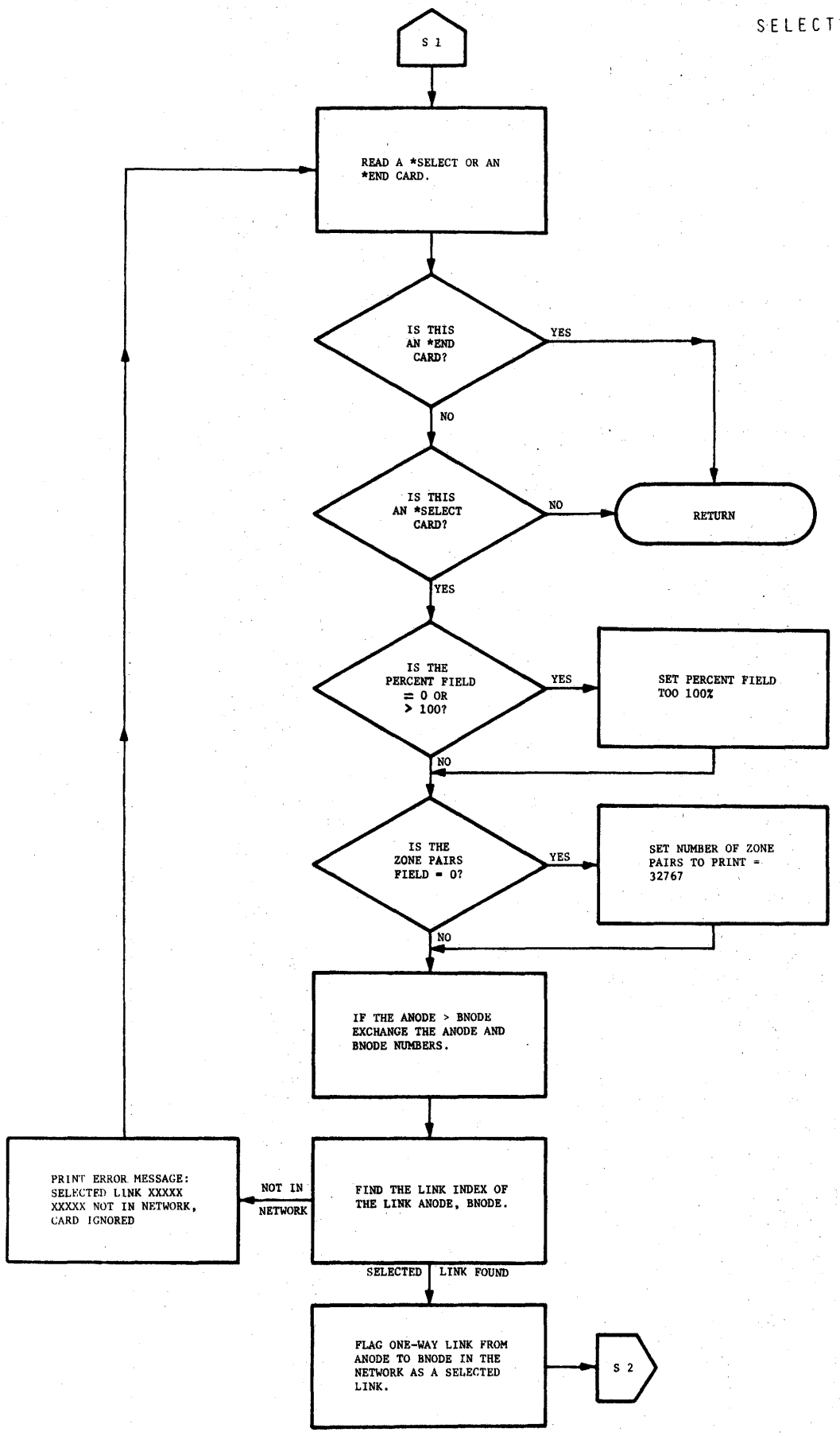

RP 4
SAVE THE POINTER TO THE NEXT ROUTE RECORD IN CORE TO PROCESS.
FIND AN END OF THE ROUTE IN ARRAYS B 1 AND B 2 BY FINDING AN I SUCH THAT B 1(I) ≠ 0 AND B 2(I) = 0
END FOUND
NO END FOUND
PRINT AN ERROR MESSAGE THAT THE ROUTE HAS NO ENDS
CHOOSE AS A STARTING POINT THE FIRST B 1(I) ≠ 0
ALL ZERO
STOP 8
B 1(I) ≠ 0
THE ARRAYS B 1 AND B 2 NOW FORM A BIDIRECTIONAL LIST STRUCTURE OF LINKS WITH THE DATA INDEXED BY ARRAYS NX 1 AND NX 2. THE LINK RECORDS ARE NOW LISTED IN THE ORDER IN WHICH THEY ARE CONNECTED AND THE LIST STRUCTURE IS DESTROYED AS EACH LINK IS PRINTED BY SETTING EITHER B 1 (I) OR B 2 (I) ELEMENT WHICH WAS USED TO REACH ELEMENT I TO ZERO TO PREVENT THE PROGRAM FROM GOING THROUGH THE LIST MORE THAN ONCE. THE INDEXES TO THE LINKS LISTED ARE SAVED IN ARRAY NX 2 SO THAT THE ROUTE CAN BE PLOTTED.
GET THE MAXIMUM VOLUME TO BE PLOTTED FOR THIS ROUTE, MAX
SC
SCALE THE Y AXIS TO PLOT MAX IN 9 INCHES.
RP 5

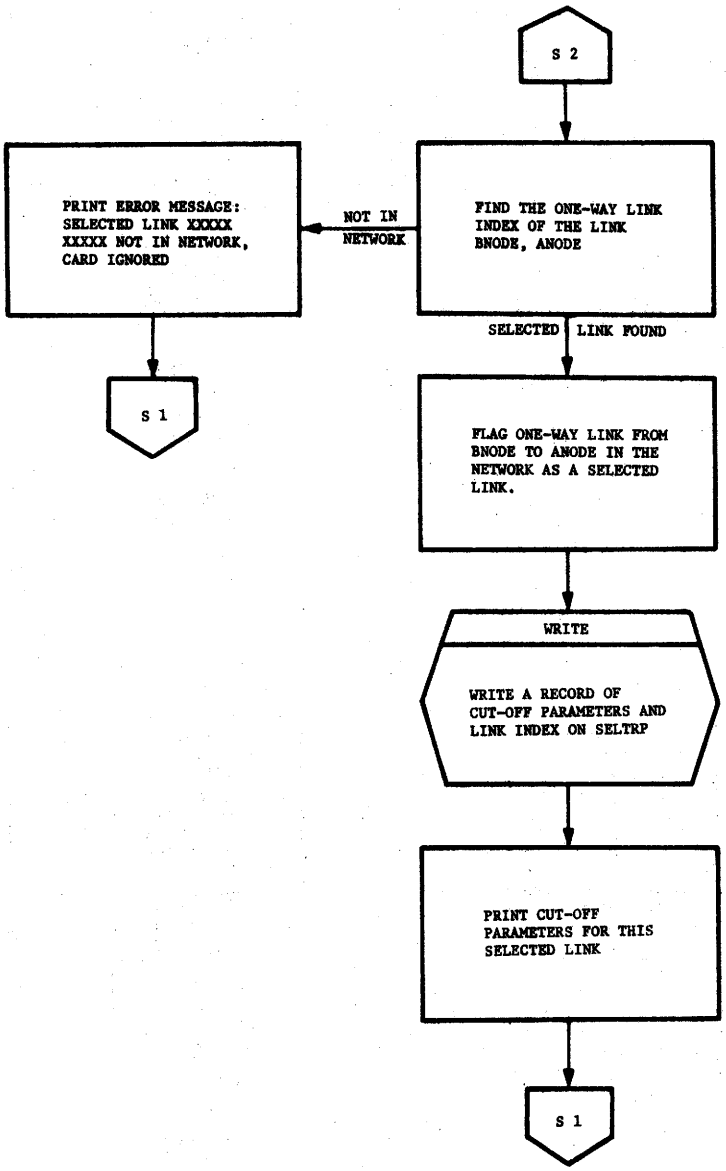

RP 5
LINE
DRAW THE Y AXIS WITH TIC MARKS AT 1 INCH INTERVALS
SYMBOL
PUT THE WORD 'VOLUME' ON THE Y AXIS
DRAW NUMBERS FOR THE TIC MARKS ON THE Y AXIS.
SC
SCALE THE X AXIS FOR AN AVERAGE OF 3 NODE NUMBER PER INCH.
DRAW NODE NUMBERS ALONG THE X AXIS IN ASCENDING ORDER OF DISTANCE WITH CONNECTING LINES
LINE
DRAW X AXIS FROM LARGEST X VALUE TO THE ORIGIN
SYMBOL
DRAW LABEL FOR THE GRAPH 'ROUTE XX'
K = NLD + 2
KK = 1
KK = 1, CORRESPONDS TO GROUND COUNTS KK = 2, CORRESPONDS TO CAPACITIES KK > 2, CORRESPONDS TO ASSIGNMENT KK - 2
RP 8
IS KK > K?
YES
RP 7
NO
RP 6
ARE THE VOLUMES CORRESPONDING TO THIS KK TO BE PLOTTED?
NO
KK = KK + 1
YES
NO
KK
> 2
2
ARE THERE ANY LINK CAPACITIES FOR THIS ROUTE?
1
YES
ARE THERE ANY GROUND COUNTS FOR THIS ROUTE?
NO
RP 6
YES
PLOT THE VOLUMES CORRESPONDING TO KK IN AN ORDER WHICH IS ASCENDING FOR THE X AXIS VALUES.
KK = KK + 1
RP 9

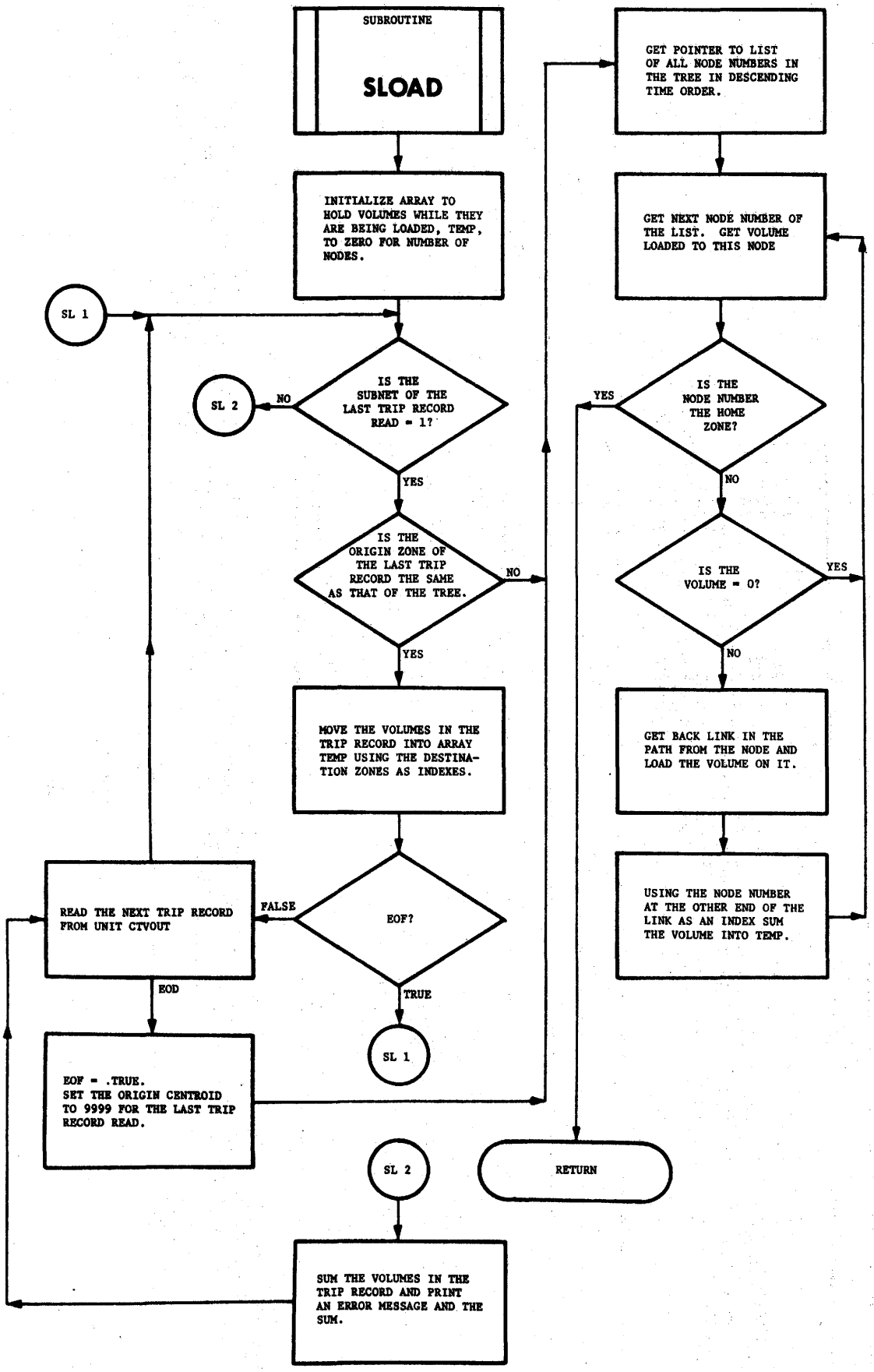

RP 9
IS KK > K?
YES
RP 7
NO
RP 10
ARE THE VOLUMES CORRES-PONDING TO THIS KK TO BE PLOTTED?
NO
KK = KK + 1
YES
NO
ARE THERE ANY GROUND COUNTS FOR THIS ROUTE?
1
KK
> 2
YES
2
RP 10
NO
ARE THERE ANY LINK CAPACITIES FOR THIS ROUTE?
YES
PLOT THE VOLUMES CORRESPONDING TO KK IN AN ORDER WHICH IS DESCENDING FOR THE X AXIS VALUES.
KK = KK + 1
KK > K
YES
NO
RP 8
RP 7
PLOT
SKIP PAST THE PLOTS FOR THIS ROUTE
SET THE ROUTE CODE TO THAT OF THE NEXT ROUTE SAVED IN CORE
END
FALSE
RP 3
TRUE
FIND HOW MANY MORE ROUTES CAN BE READ INTO CORE AT THE SAME TIME AND SET FLAGS FOR THEM TO BE READ IN.
RP 2
HAVE ALL ROUTES BEEN PROCESSED?
YES
RETURN
NO
SKIP THE HEADER RECORDS ON THE ROUTE DATA SET. SET END = .FALSE.
READ A ROUTE LINK RECORD
EOD
RP 11
IS THIS A ROUTE TO SAVE IN CORE
YES
INCREMENT LOCATION TO READ NEXT LINK RECORD INTO.
NO

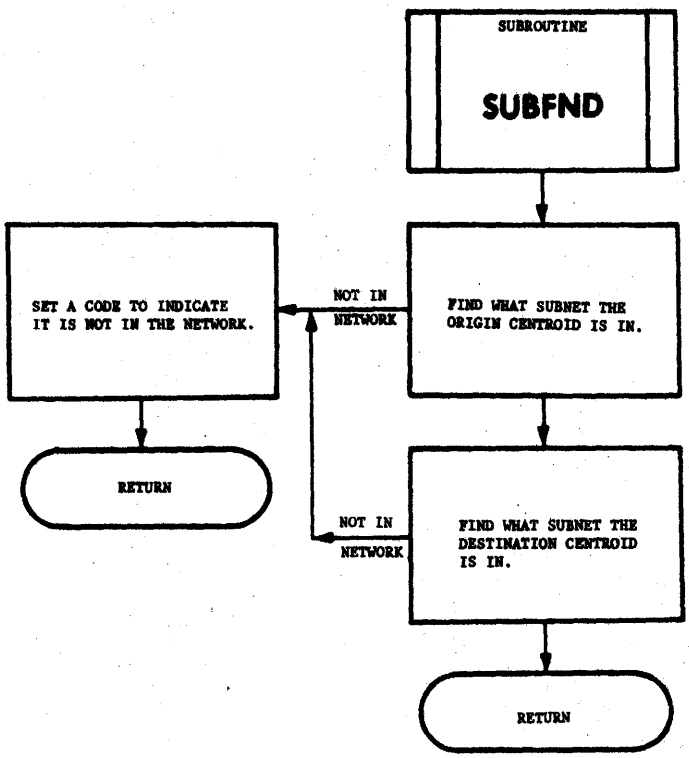

FUNCTION
SC
BREAK NUMBER TO BE SCALED INTO TWO PARTS DXS, A NUMBER BETWEEN 1.0 AND 10.0 AND F 10 A MULTIPLIER WHICH IS A POWER OF 10
IS DXS < 4.0?
NO
YES
THIS EFFECTIVELY ROUNDS DXS TO THE NEXT LARGEST NUMBER OF THE TYPE n*0.5 WHERE n IS AN INTEGER
ROUND 2*DXS TO THE NEXT LARGEST INTEGER AND PUT IN IX.
SC = (FLOAT (IX) /2.0) *F10
ROUND DXS UP TO THE NEXT LARGEST INTEGER, IX
SC = IX*F 10
RETURN

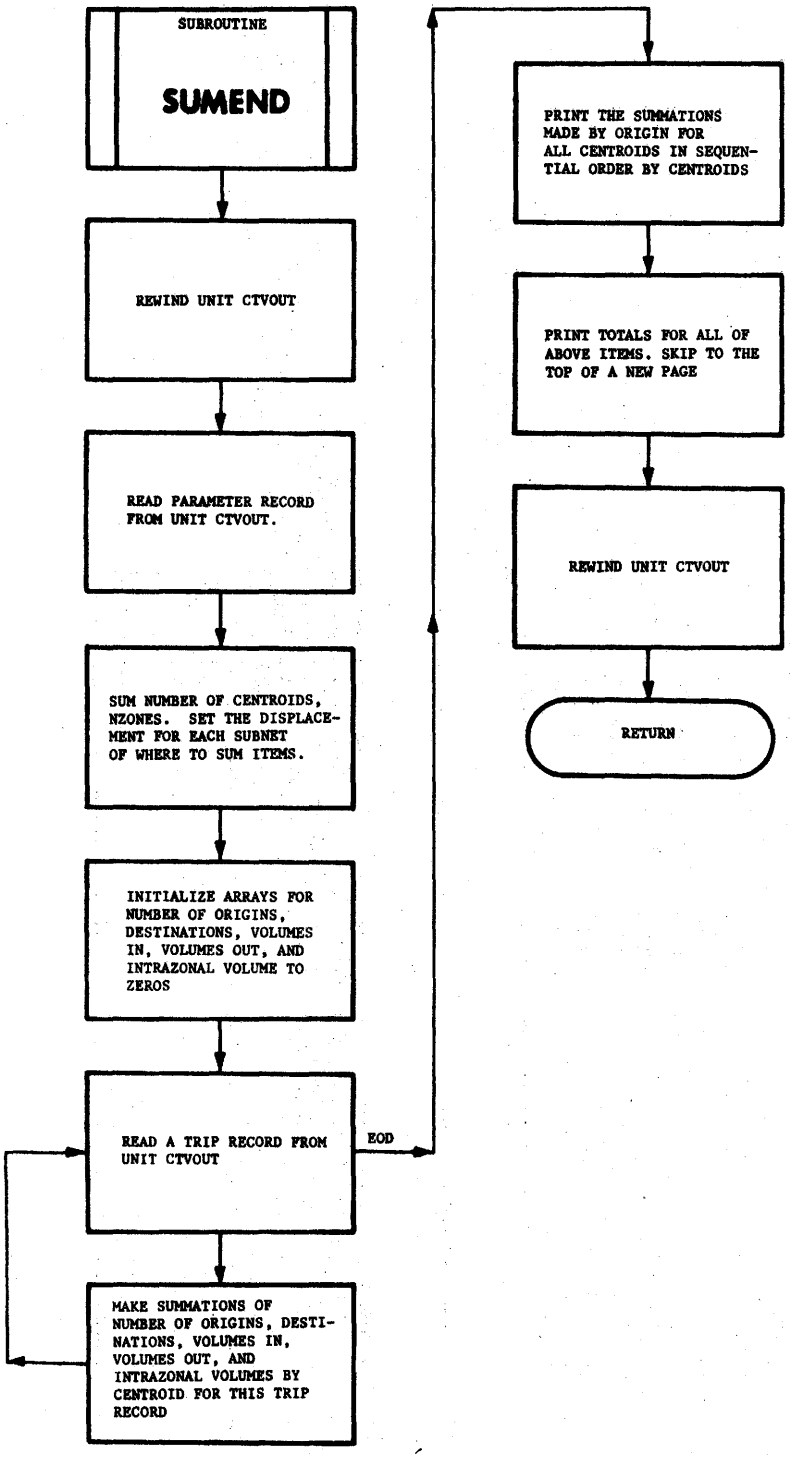

SUBROUTINE
SELECT
OPEN
OPEN SELTRP DATA SET FOR OUTPUT.
SET FLAG SO THAT LOADED NETWORK WILL BE PRINTED
READ OUTPUT SPECIFICATION CARD
IS THE CARD AN *ALL CARD?
YES
NO
SET TURNING MOVEMENT CODES TO SAVE NONE AND OUTPUT NONE.
IS THE OUTPUT SPECIFICATION CARD *LINKS?
YES
NO
SET FLAG TO SUPPRESS PRINTING OF LOADED NETWORK.
IS THE OUTPUT OPTION CARD *NONE?
NO
YES
PRINT ERROR MESSAGE: SELECTED LINKS OUTPUT OPTION INVALID, OPTION READ = *XXXX RUN DELETED
WRITE HEADER AND DATE ON SELTRP DATA SET
STOP 6
S 1

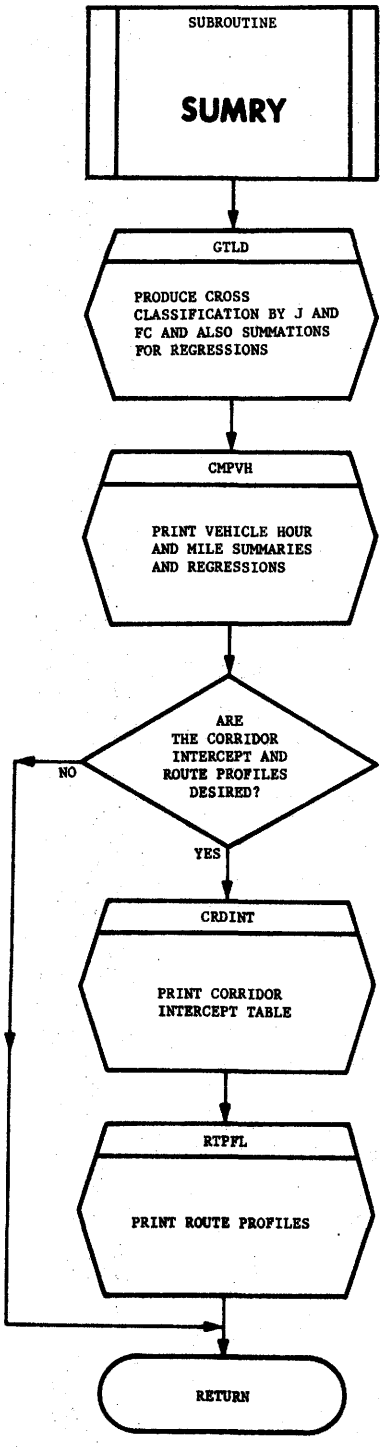

S 1
READ A *SELECT OR AN *END CARD.
IS THIS AN *END CARD?
YES
NO
IS THIS AN *SELECT CARD?
NO
RETURN
YES
IS THE PERCENT FIELD = 0 OR > 100?
YES
SET PERCENT FIELD TOO 100%
NO
IS THE ZONE PAIRS FIELD = 0?
YES
SET NUMBER OF ZONE PAIRS TO PRINT = 32767
NO
IF THE ANODE > BNODE EXCHANGE THE ANODE AND BNODE NUMBERS.
FIND THE LINK INDEX OF THE LINK ANODE, BNODE.
NOT IN NETWORK
PRINT ERROR MESSAGE: SELECTED LINK XXXXX XXXXX NOT IN NETWORK, CARD IGNORED
SELECTED LINK FOUND
FLAG ONE-WAY LINK FROM ANODE TO BNODE IN THE NETWORK AS A SELECTED LINK.
S 2


S 2
FIND THE ONE-WAY LINK INDEX OF THE LINK BNODE, ANODE
NOT IN NETWORK
PRINT ERROR MESSAGE: SELECTED LINK XXXXX XXXXX NOT IN NETWORK, CARD IGNORED
S 1
SELECTED LINK FOUND
FLAG ONE-WAY LINK FROM BNODE TO ANODE IN THE NETWORK AS A SELECTED LINK.
WRITE
WRITE A RECORD OF CUT-OFF PARAMETERS AND LINK INDEX ON SELTRP
PRINT CUT-OFF PARAMETERS FOR THIS SELECTED LINK
S 1

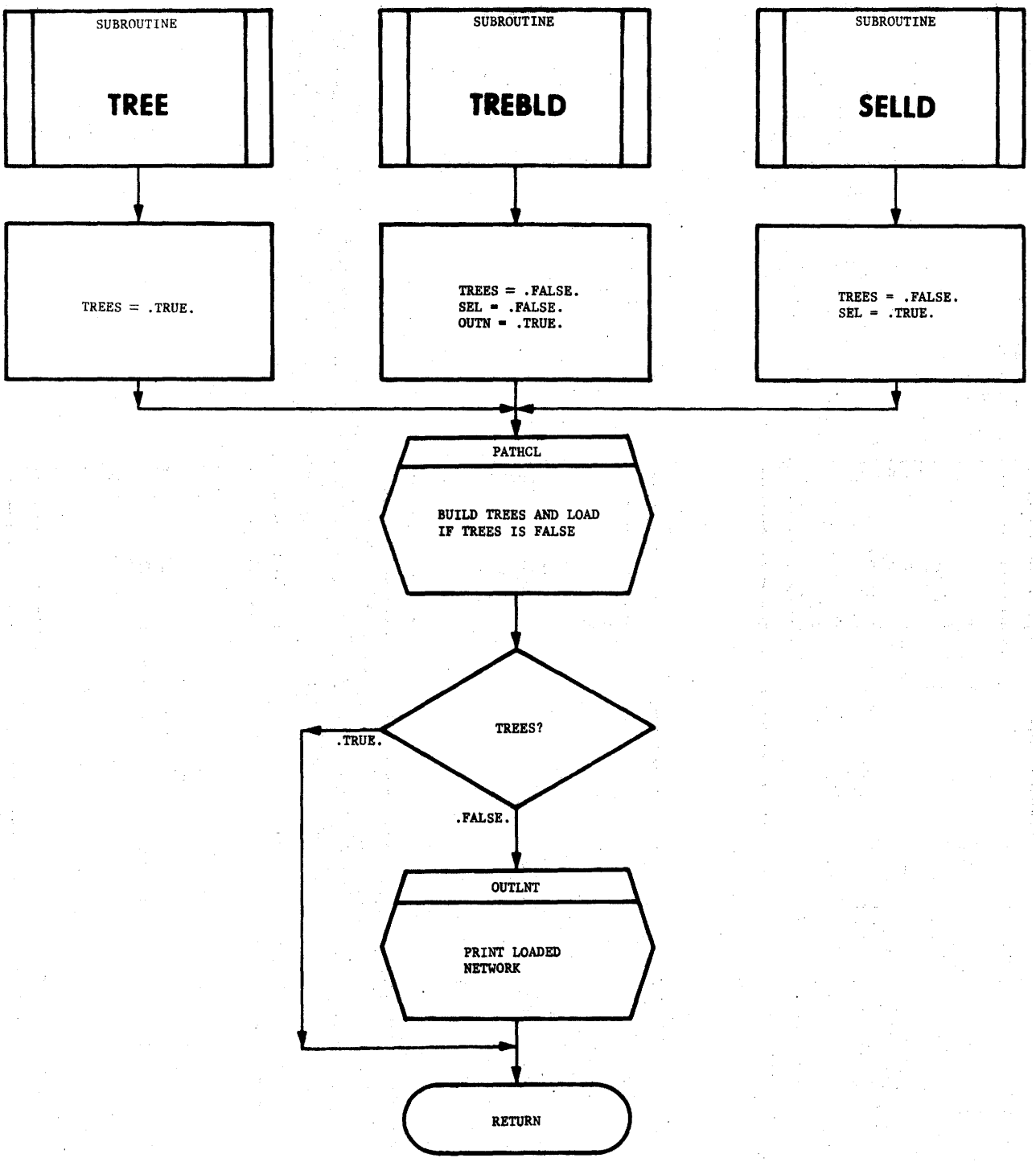

SUBROUTINE
SLOAD
INITIALIZE ARRAY TO HOLD VOLUMES WHILE THEY ARE BEING LOADED, TEMP, TO ZERO FOR NUMBER OF NODES.
SL 1
IS THE SUBNET OF THE LAST TRIP RECORD READ = 1?
NO
SL 2
YES
IS THE ORIGIN ZONE OF THE LAST TRIP RECORD THE SAME AS THAT OF THE TREE.
NO
YES
MOVE THE VOLUMES IN THE TRIP RECORD INTO ARRAY TEMP USING THE DESTINATION ZONES AS INDEXES.
READ THE NEXT TRIP RECORD FROM UNIT CTVOUT
FALSE
EOF?
TRUE
EOD
SL 1
EOF = .TRUE.
SET THE ORIGIN CENTROID TO 9999 FOR THE LAST TRIP RECORD READ.
SL 2
SUM THE VOLUMES IN THE TRIP RECORD AND PRINT AN ERROR MESSAGE AND THE SUM.
GET POINTER TO LIST OF ALL NODE NUMBERS IN THE TREE IN DESCENDING TIME ORDER.
GET NEXT NODE NUMBER OF THE LIST. GET VOLUME LOADED TO THIS NODE
IS THE NODE NUMBER THE HOME ZONE?
YES
NO
IS THE VOLUME = 0?
YES
NO
GET BACK LINK IN THE PATH FROM THE NODE AND LOAD THE VOLUME ON IT.
USING THE NODE NUMBER AT THE OTHER END OF THE LINK AS AN INDEX SUM THE VOLUME INTO TEMP.
RETURN

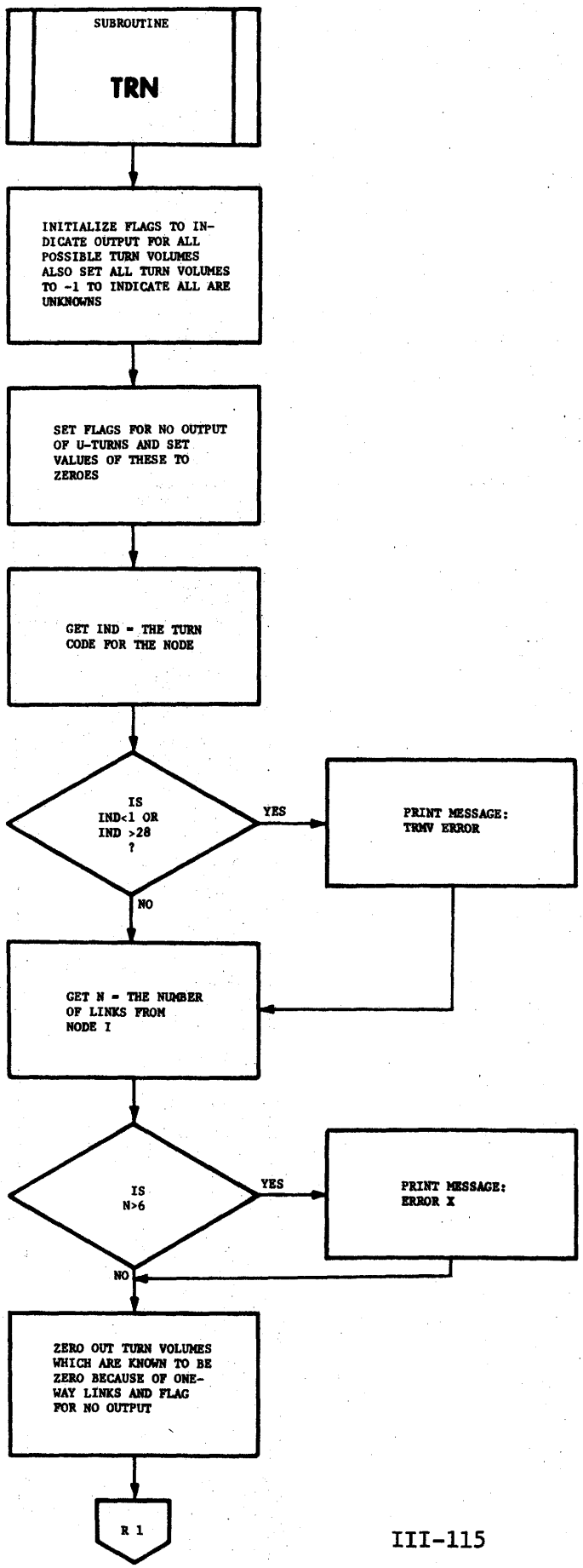

SUBROUTINE
SUBFND
FIND WHAT SUBNET THE ORIGIN CENTROID IS IN.
NOT IN NETWORK
SET A CODE TO INDICATE IT IS NOT IN THE NETWORK.
RETURN
FIND WHAT SUBNET THE DESTINATION CENTROID IS IN.
NOT IN NETWORK
RETURN

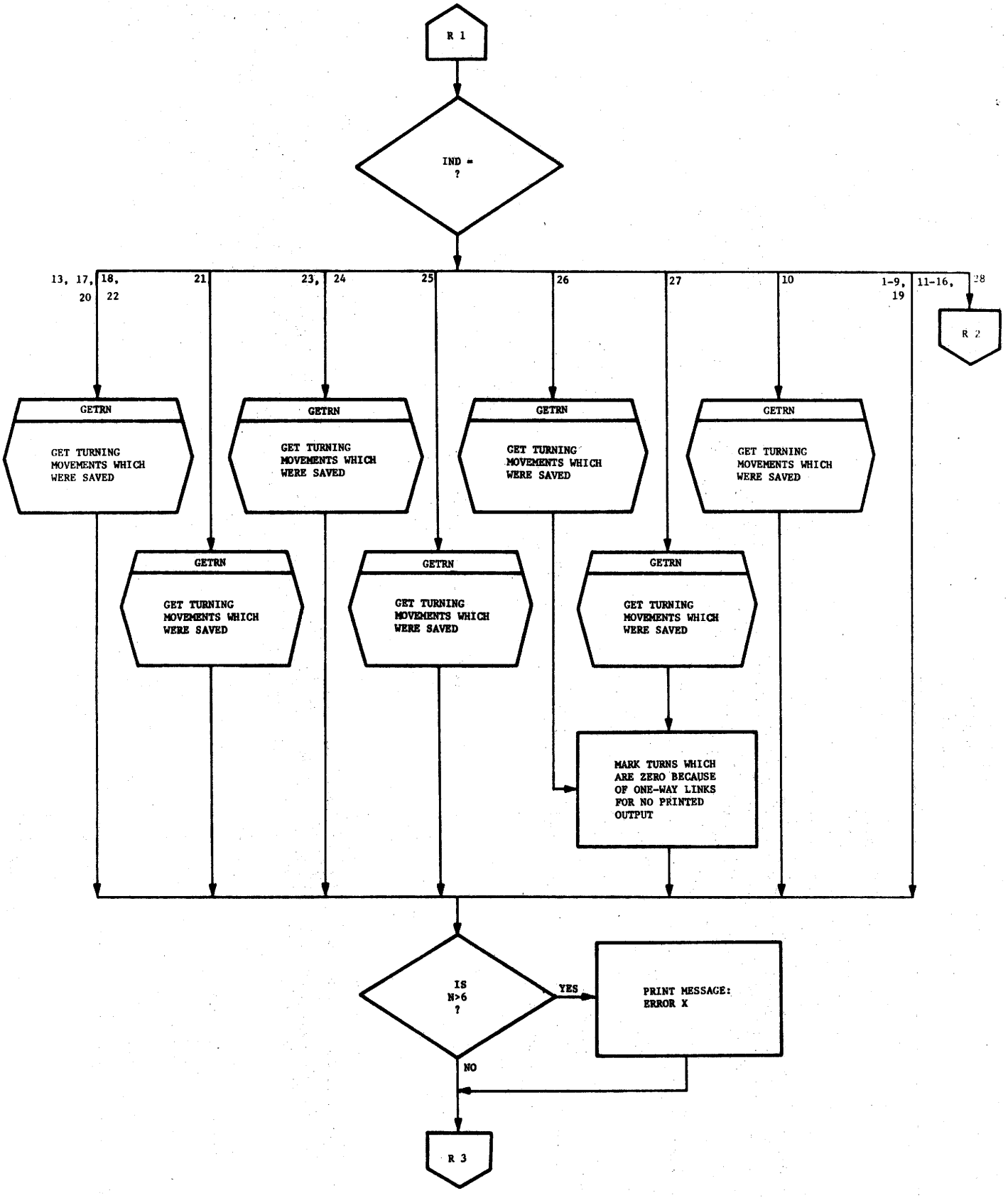

SUBROUTINE
SUMEND
REWIND UNIT CTVOUT
READ PARAMETER RECORD FROM UNIT CTVOUT.
SUM NUMBER OF CENTROIDS, NZONES. SET THE DISPLACEMENT FOR EACH SUBNET OF WHERE TO SUM ITEMS.
INITIALIZE ARRAYS FOR NUMBER OF ORIGINS, DESTINATIONS, VOLUMES IN, VOLUMES OUT, AND INTRAZONAL VOLUME TO ZEROS
READ A TRIP RECORD FROM UNIT CTVOUT
EOD
MAKE SUMMATIONS OF NUMBER OF ORIGINS, DESTINATIONS, VOLUMES IN, VOLUMES OUT, AND INTRAZONAL VOLUMES BY CENTROID FOR THIS TRIP RECORD
PRINT THE SUMMATIONS MADE BY ORIGIN FOR ALL CENTROIDS IN SEQUENTIAL ORDER BY CENTROIDS
PRINT TOTALS FOR ALL OF ABOVE ITEMS. SKIP TO THE TOP OF A NEW PAGE
REWIND UNIT CTVOUT
RETURN

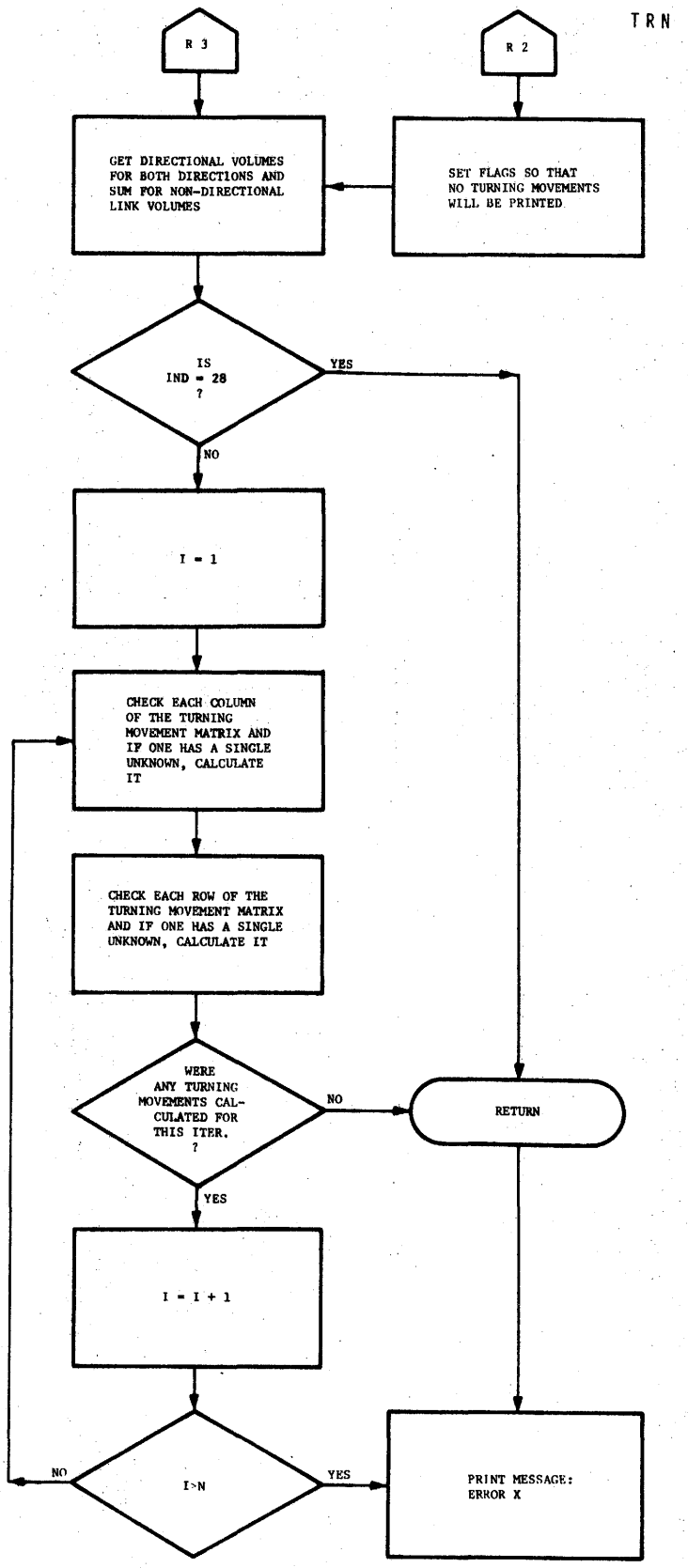

SUBROUTINE
SUMRY
GTLD
PRODUCE CROSS CLASSIFICATION BY J AND FC AND ALSO SUMMATIONS FOR REGRESSIONS
CMPVH
PRINT VEHICLE HOUR AND MILE SUMMARIES AND REGRESSIONS
ARE THE CORRIDOR INTERCEPT AND ROUTE PROFILES DESIRED?
NO
YES
CRDINT
PRINT CORRIDOR INTERCEPT TABLE
RTPFL
PRINT ROUTE PROFILES
RETURN

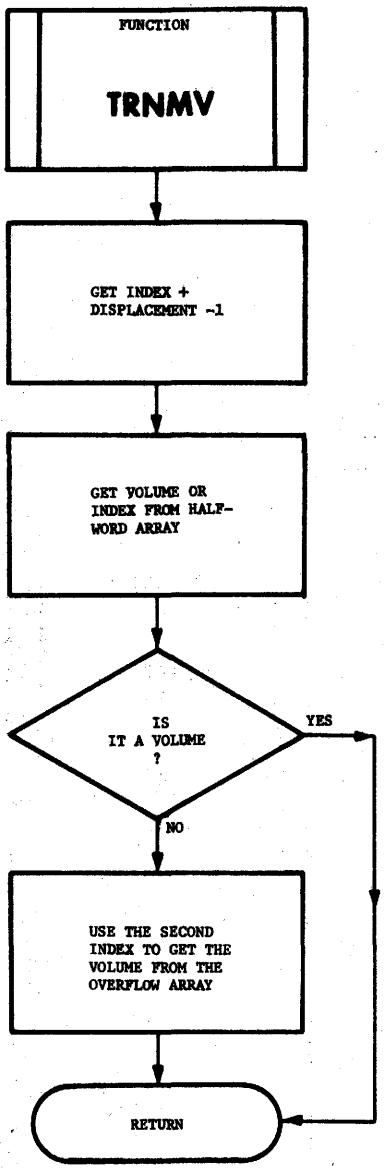
TREE
TREBLD
SELLD

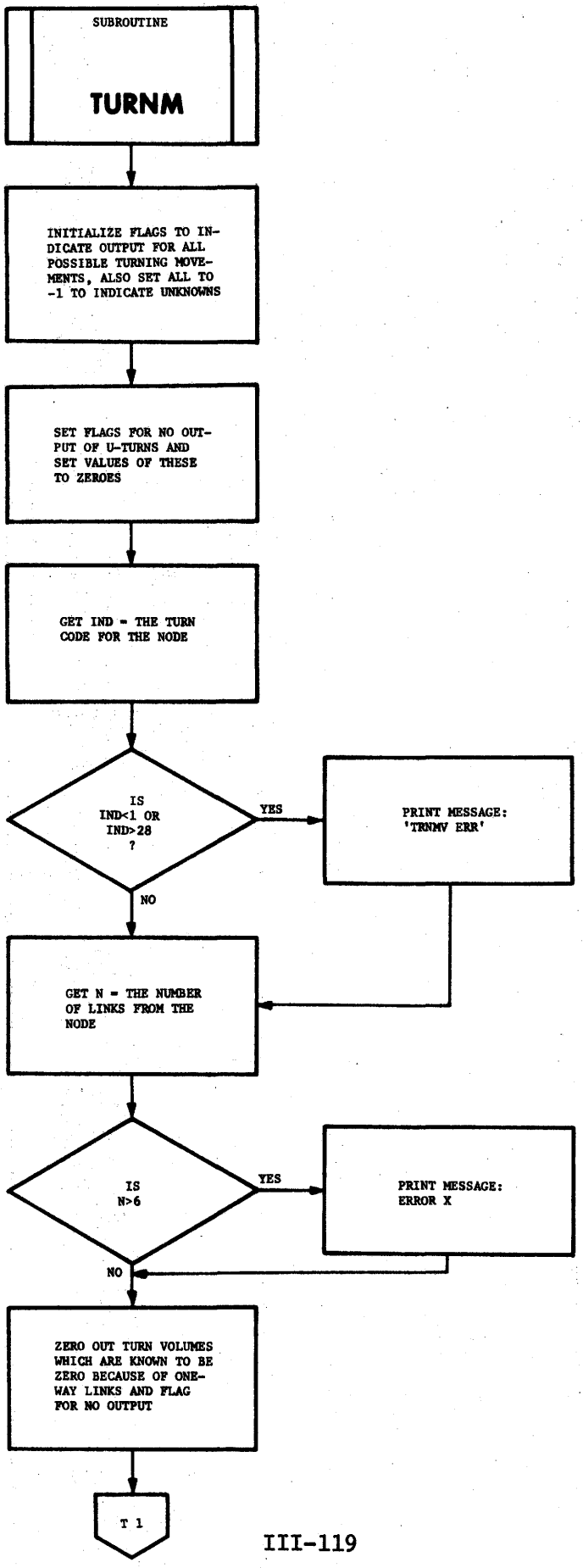


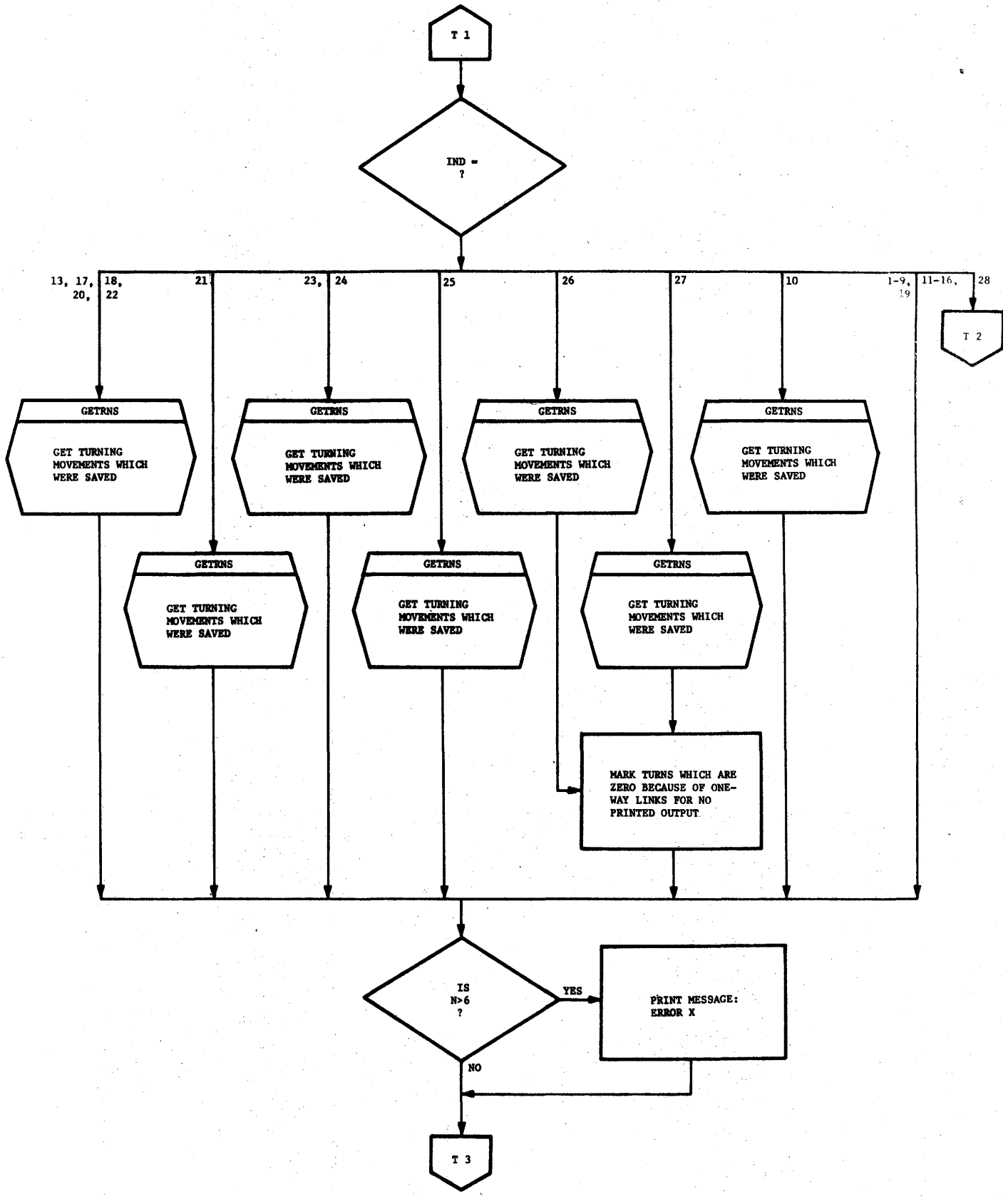

SUBROUTINE
TRN
INITIALIZE FLAGS TO INDICATE OUTPUT FOR ALL POSSIBLE TURN VOLUMES ALSO SET ALL TURN VOLUMES TO -1 TO INDICATE ALL ARE UNKNOWNS
SET FLAGS FOR NO OUTPUT OF U-TURNS AND SET VALUES OF THESE TO ZEROES
GET IND = THE TURN CODE FOR THE NODE
IS IND<1 OR IND >28 ?
YES
PRINT MESSAGE: TRMV ERROR
NO
GET N = THE NUMBER OF LINKS FROM NODE I
IS N>6
YES
PRINT MESSAGE: ERROR X
NO
ZERO OUT TURN VOLUMES WHICH ARE KNOWN TO BE ZERO BECAUSE OF ONE-WAY LINKS AND FLAG FOR NO OUTPUT
R 1

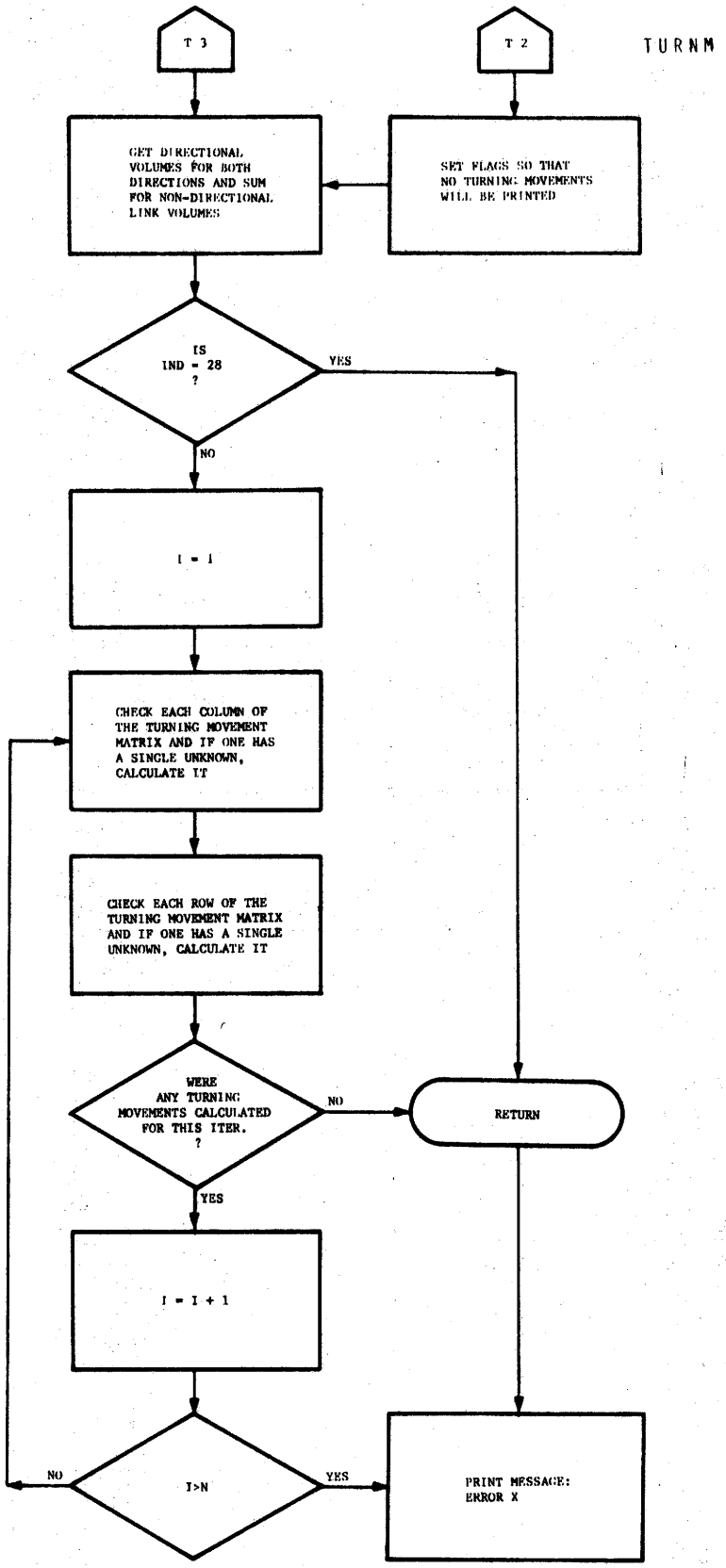

R 1
IND =
?
13, 17, 20
18, 22
21
23,
24
25
26
27
10
1-9, 19
11-16,
28
R 2
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
MARK TURNS WHICH ARE ZERO BECAUSE OF ONE-WAY LINKS FOR NO PRINTED OUTPUT
IS N>6 ?
YES
PRINT MESSAGE: ERROR X
NO
R 3

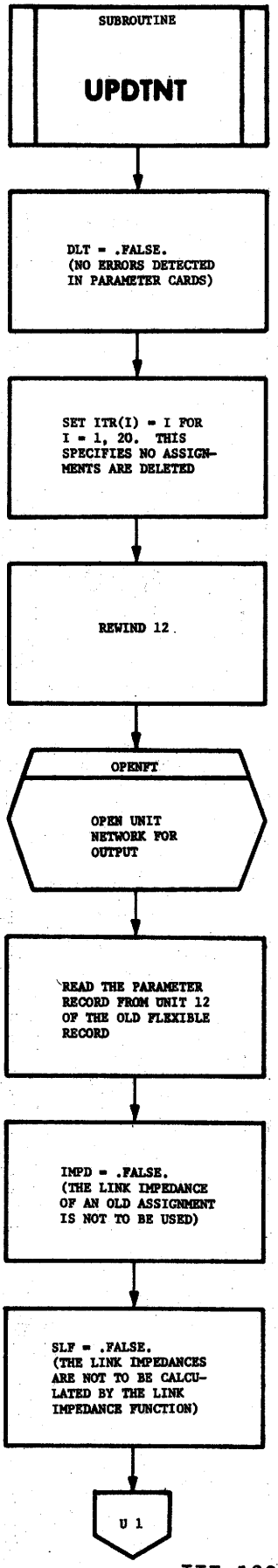

TRN
R 3
R 2
GET DIRECTIONAL VOLUMES FOR BOTH DIRECTIONS AND SUM FOR NON-DIRECTIONAL LINK VOLUMES
SET FLAGS SO THAT NO TURNING MOVEMENTS WILL BE PRINTED
IS IND = 28 ?
YES
NO
I = 1
CHECK EACH COLUMN OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
CHECK EACH ROW OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
WERE ANY TURNING MOVEMENTS CALCULATED FOR THIS ITER. ?
NO
RETURN
YES
I = I + 1
I>N
NO
YES
PRINT MESSAGE: ERROR X

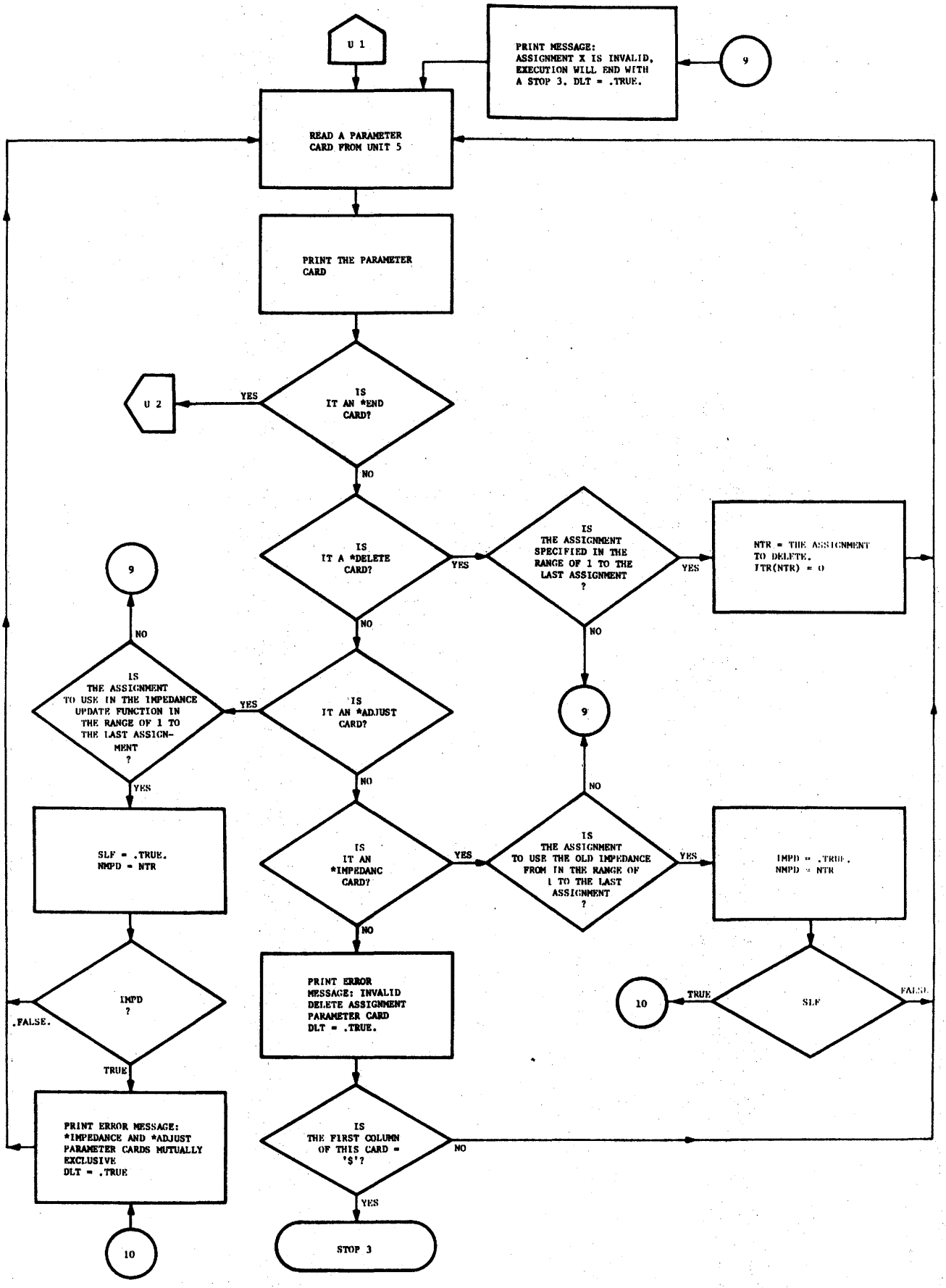

FUNCTION
TRNMV
GET INDEX + DISPLACEMENT -1
GET VOLUME OR INDEX FROM HALF-WORD ARRAY
IS IT A VOLUME ?
YES
NO
USE THE SECOND INDEX TO GET THE VOLUME FROM THE OVERFLOW ARRAY
RETURN

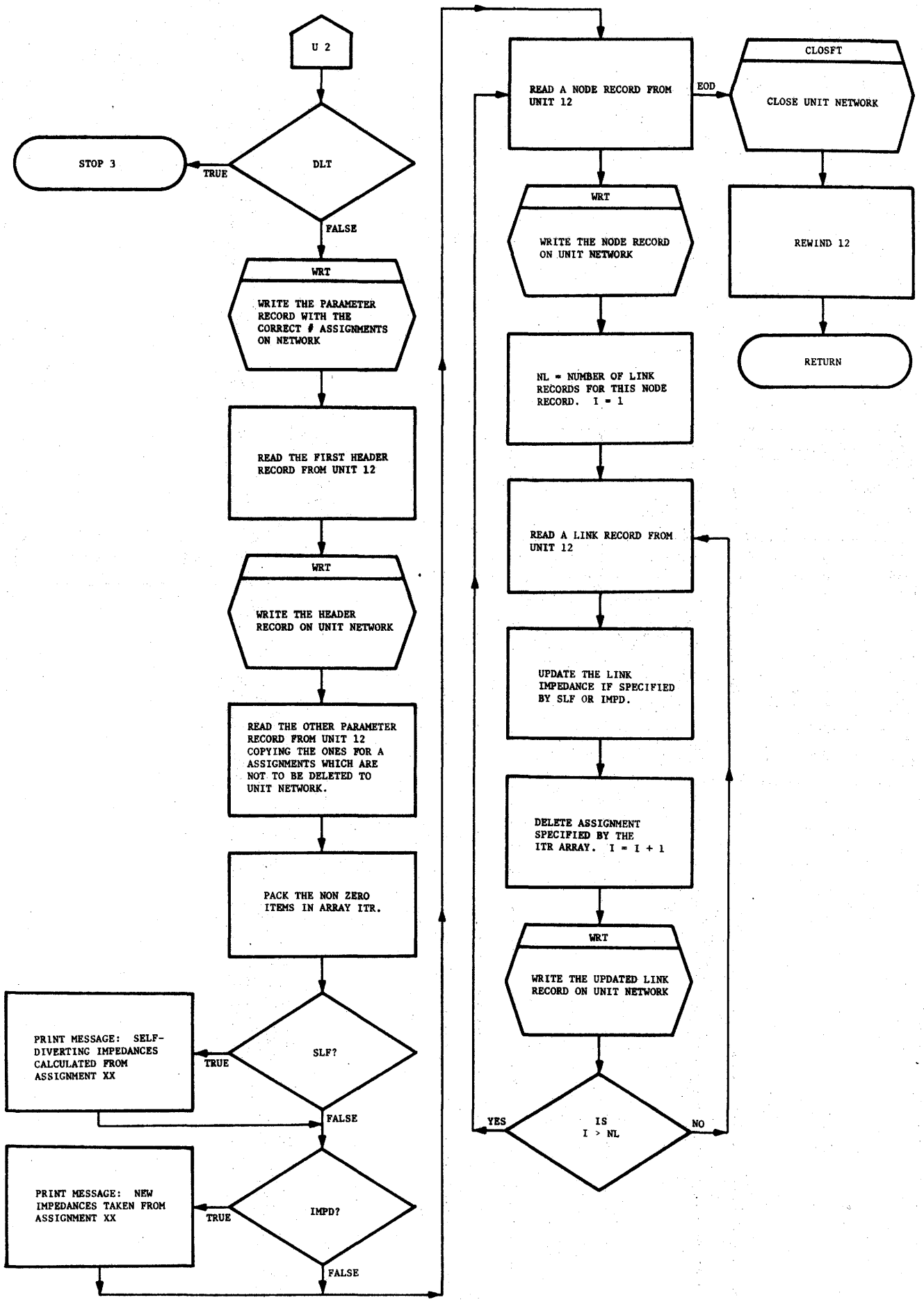

SUBROUTINE
TURNM
INITIALIZE FLAGS TO INDICATE OUTPUT FOR ALL POSSIBLE TURNING MOVEMENTS, ALSO SET ALL TO -1 TO INDICATE UNKNOWNS
SET FLAGS FOR NO OUTPUT OF U-TURNS AND SET VALUES OF THESE TO ZEROES
GET IND = THE TURN CODE FOR THE NODE
IS IND<1 OR IND>28 ?
YES
PRINT MESSAGE: 'TRNMV ERR'
NO
GET N = THE NUMBER OF LINKS FROM THE NODE
IS N>6
YES
PRINT MESSAGE: ERROR X
NO
ZERO OUT TURN VOLUMES WHICH ARE KNOWN TO BE ZERO BECAUSE OF ONE-WAY LINKS AND FLAG FOR NO OUTPUT
T 1

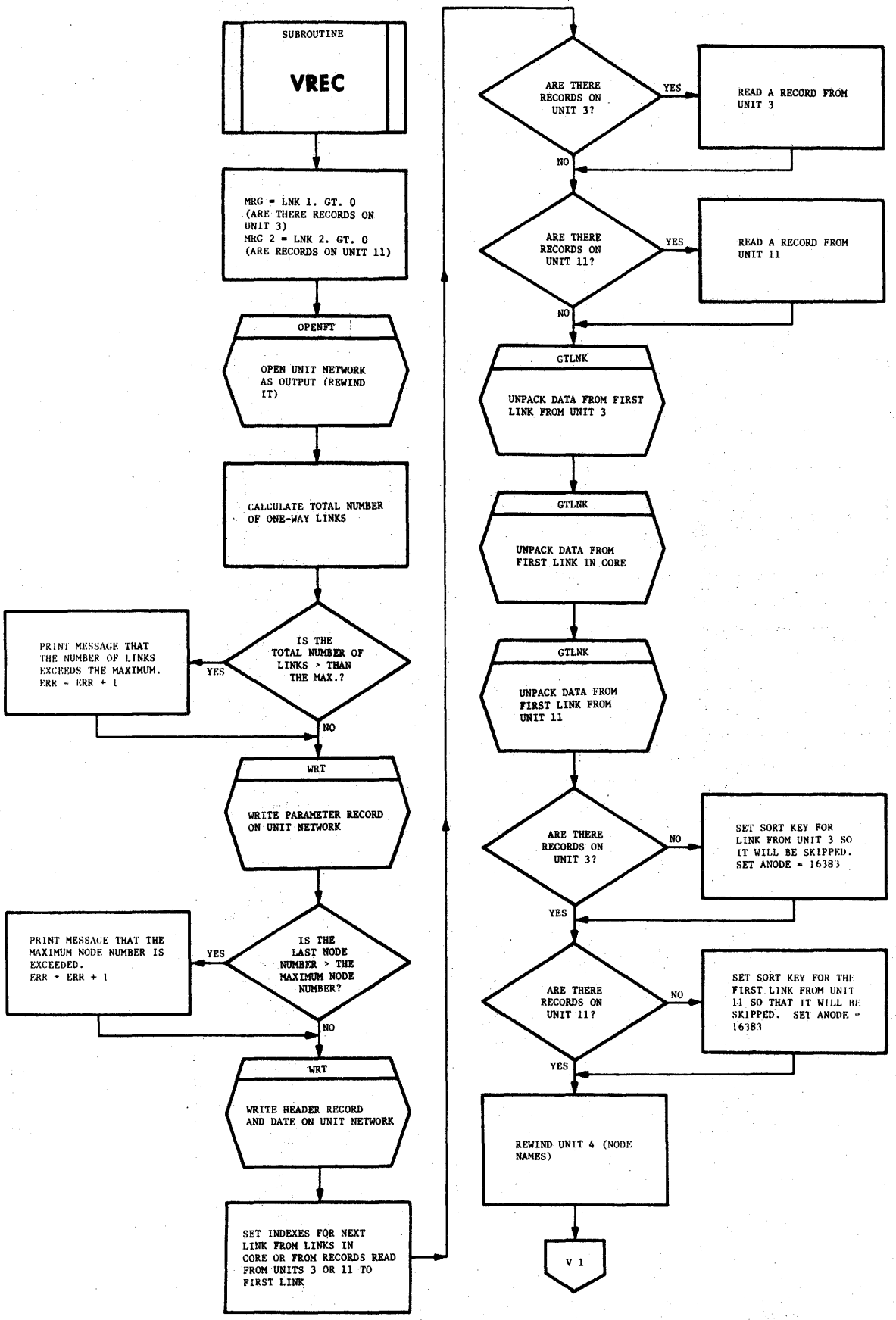

T 1
IND = ?
13, 17, 20,
18, 22
21
23, 24
25
26
27
10
1-9, 19
11-16,
28
T 2
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
MARK TURNS WHICH ARE ZERO BECAUSE OF ONE-WAY LINKS FOR NO PRINTED OUTPUT
IS N>6 ?
YES
PRINT MESSAGE: ERROR X
NO
T 3

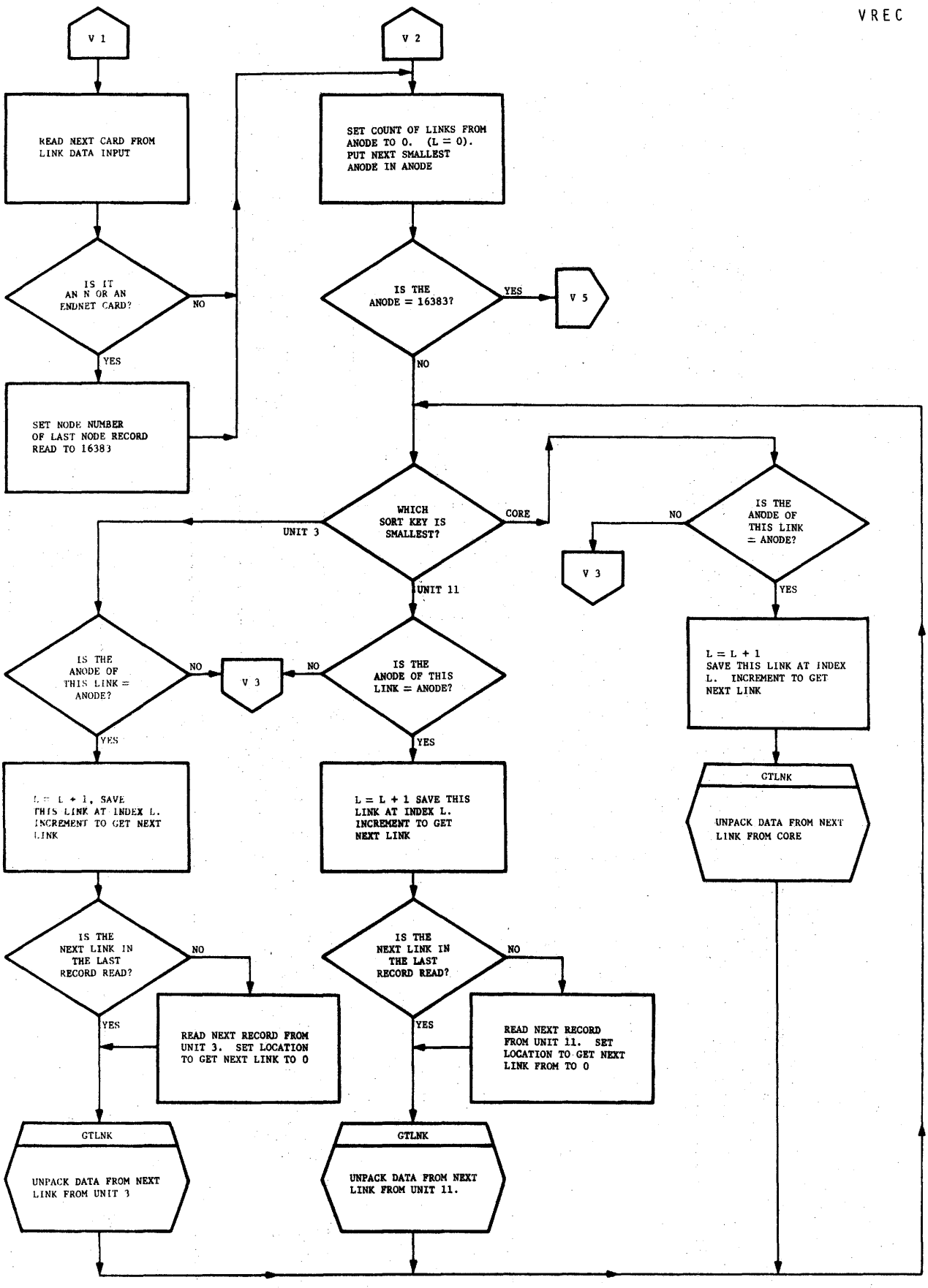

T 3
T 2
TURNM
GET DIRECTIONAL VOLUMES FOR BOTH DIRECTIONS AND SUM FOR NON-DIRECTIONAL LINK VOLUMES
SET FLAGS SO THAT NO TURNING MOVEMENTS WILL BE PRINTED
IS IND = 28 ?
YES
NO
I = 1
CHECK EACH COLUMN OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
CHECK EACH ROW OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
WERE ANY TURNING MOVEMENTS CALCULATED FOR THIS ITER. ?
NO
RETURN
YES
I = I + 1
I>N
NO
YES
PRINT MESSAGE: ERROR X

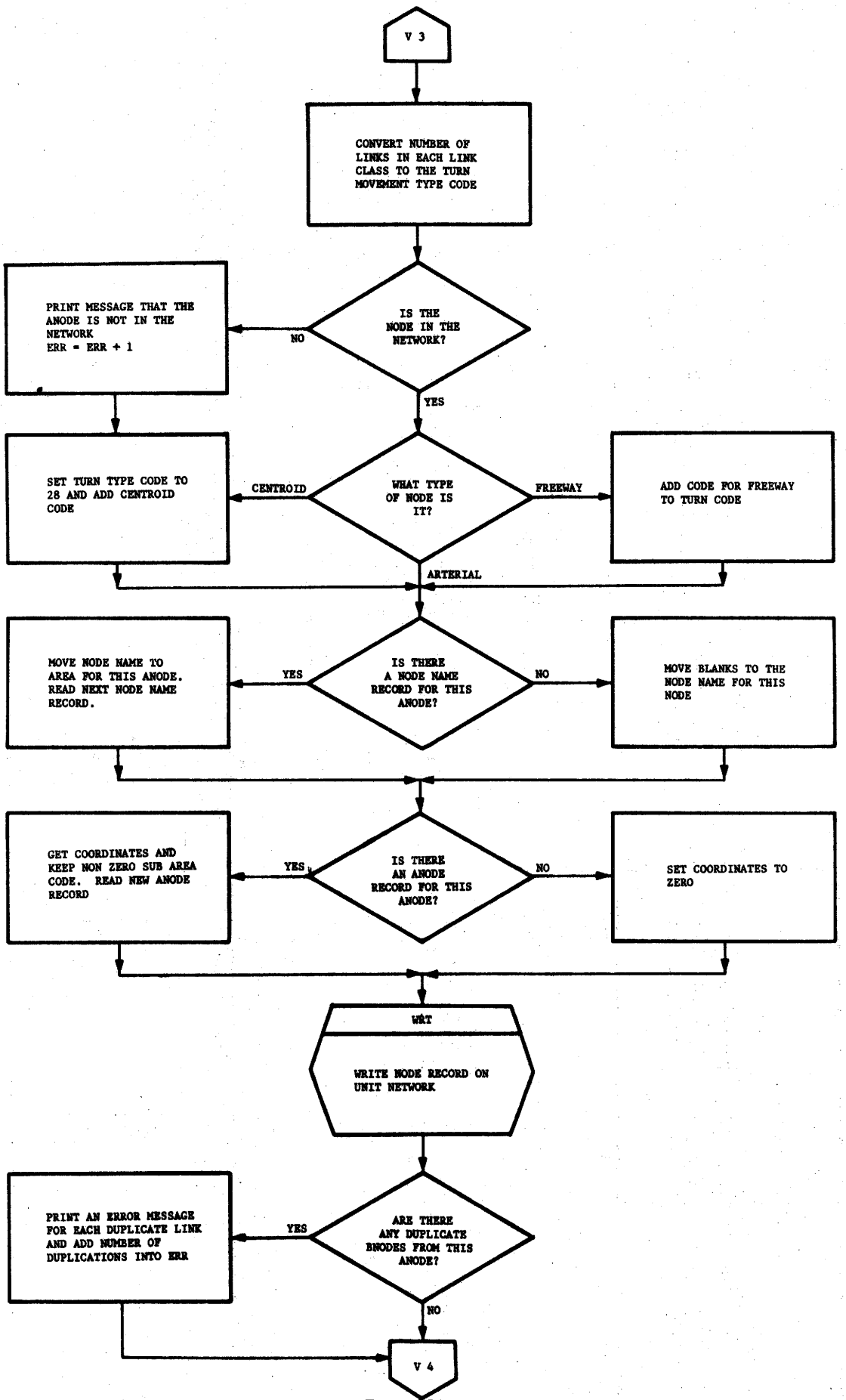

SUBROUTINE
UPDTNT
DLT = .FALSE.
(NO ERRORS DETECTED
IN PARAMETER CARDS)
SET ITR(I) = I FOR
I = 1, 20. THIS
SPECIFIES NO ASSIGN-
MENTS ARE DELETED
REWIND 12
OPENFT
OPEN UNIT
NETWORK FOR
OUTPUT
READ THE PARAMETER
RECORD FROM UNIT 12
OF THE OLD FLEXIBLE
RECORD
IMPD = .FALSE.
(THE LINK IMPEDANCE
OF AN OLD ASSIGNMENT
IS NOT TO BE USED)
SLF = .FALSE.
(THE LINK IMPEDANCES
ARE NOT TO BE CALCU-
LATED BY THE LINK
IMPEDANCE FUNCTION)
U 1

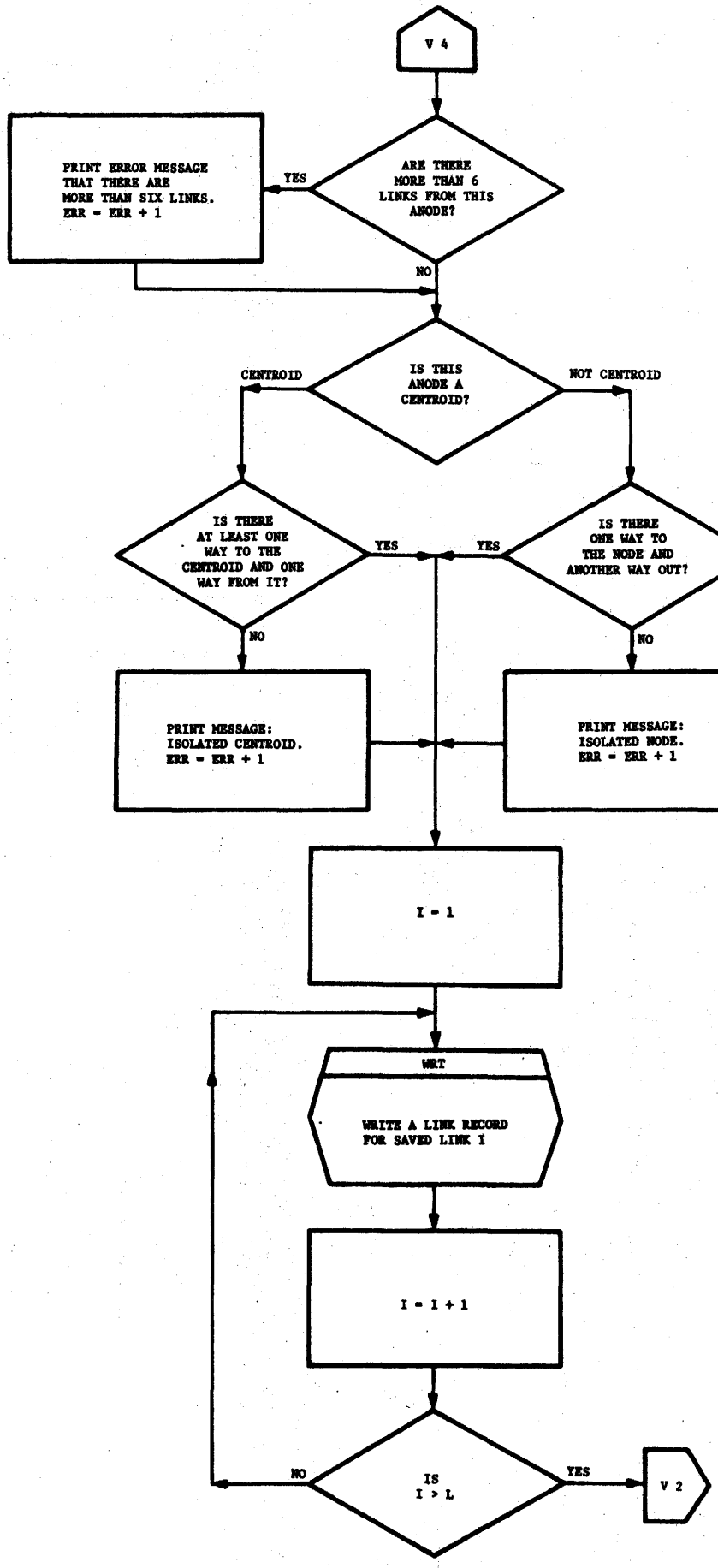

U 1
PRINT MESSAGE: ASSIGNMENT X IS INVALID, EXECUTION WILL END WITH A STOP 3. DLT = .TRUE.
9
READ A PARAMETER CARD FROM UNIT 5
PRINT THE PARAMETER CARD
IS IT AN *END CARD?
YES
U 2
NO
IS IT A *DELETE CARD?
YES
IS THE ASSIGNMENT SPECIFIED IN THE RANGE OF 1 TO THE LAST ASSIGNMENT ?
YES
NTR = THE ASSIGNMENT TO DELETE. ITR(NTR) = 0
NO
9
NO
IS IT AN *ADJUST CARD?
YES
IS THE ASSIGNMENT TO USE IN THE IMPEDANCE UPDATE FUNCTION IN THE RANGE OF 1 TO THE LAST ASSIGN-MENT ?
NO
9
YES
SLF = .TRUE. NMPD = NTR
NO
IS IT AN *IMPEDANC CARD?
YES
IS THE ASSIGNMENT TO USE THE OLD IMPEDANCE FROM IN THE RANGE OF 1 TO THE LAST ASSIGNMENT ?
YES
IMPD = .TRUE. NMPD = NTR
NO
PRINT ERROR MESSAGE: INVALID DELETE ASSIGNMENT PARAMETER CARD DLT = .TRUE.
IMPD ?
.FALSE.
TRUE
PRINT ERROR MESSAGE: *IMPEDANCE AND *ADJUST PARAMETER CARDS MUTUALLY EXCLUSIVE DLT = .TRUE
10
10
TRUE
SLF
FALSE
IS THE FIRST COLUMN OF THIS CARD = '$'?
NO
YES
STOP 3

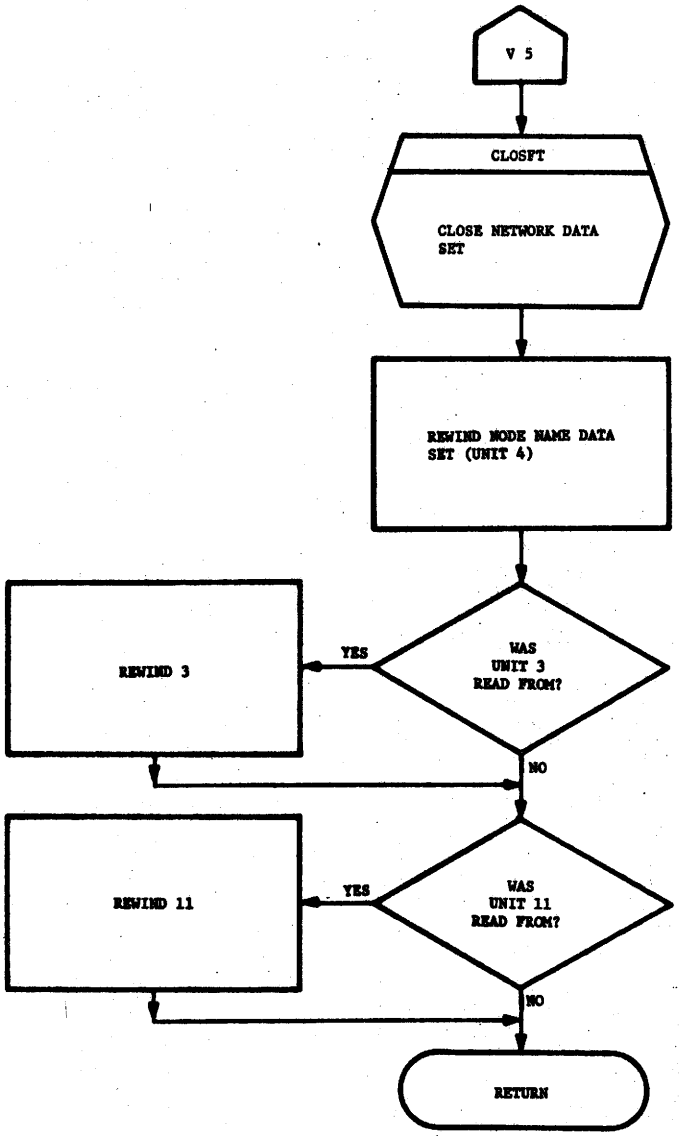

U 2
DLT
TRUE
STOP 3
FALSE
WRT
WRITE THE PARAMETER RECORD WITH THE CORRECT # ASSIGNMENTS ON NETWORK
READ THE FIRST HEADER RECORD FROM UNIT 12
WRT
WRITE THE HEADER RECORD ON UNIT NETWORK
READ THE OTHER PARAMETER RECORD FROM UNIT 12 COPYING THE ONES FOR A ASSIGNMENTS WHICH ARE NOT TO BE DELETED TO UNIT NETWORK.
PACK THE NON ZERO ITEMS IN ARRAY ITR.
SLF?
TRUE
PRINT MESSAGE: SELF-DIVERTING IMPEDANCES CALCULATED FROM ASSIGNMENT XX
FALSE
IMPD?
TRUE
PRINT MESSAGE: NEW IMPEDANCES TAKEN FROM ASSIGNMENT XX
FALSE
READ A NODE RECORD FROM UNIT 12
EOD
CLOSFT
CLOSE UNIT NETWORK
REWIND 12
RETURN
WRT
WRITE THE NODE RECORD ON UNIT NETWORK
NL = NUMBER OF LINK RECORDS FOR THIS NODE RECORD. I = 1
READ A LINK RECORD FROM UNIT 12
UPDATE THE LINK IMPEDANCE IF SPECIFIED BY SLF OR IMPD.
DELETE ASSIGNMENT SPECIFIED BY THE ITR ARRAY. I = I + 1
WRT
WRITE THE UPDATED LINK RECORD ON UNIT NETWORK
IS I > NL
YES
NO

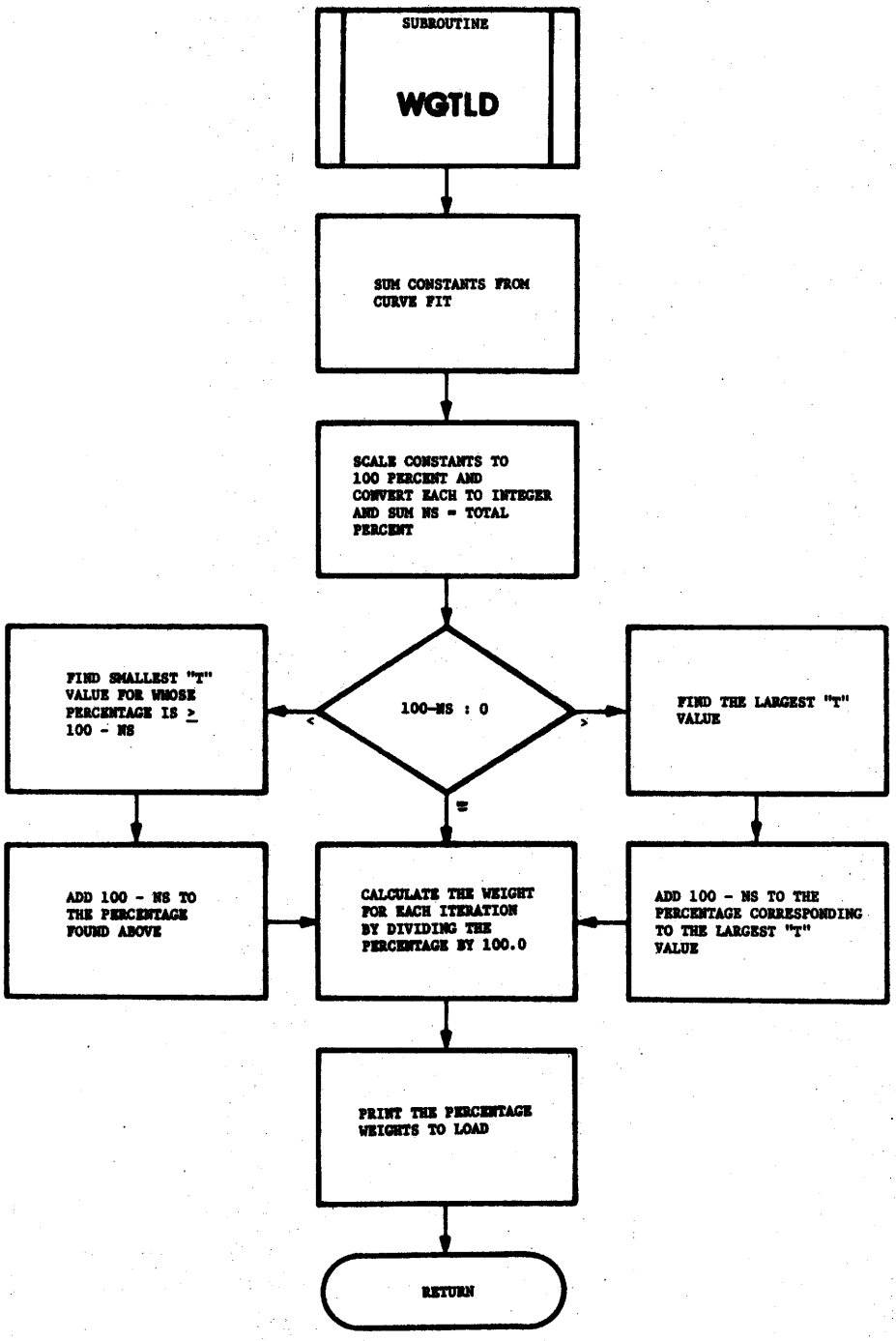

SUBROUTINE
VREC
MRG = LNK 1. GT. 0
(ARE THERE RECORDS ON
UNIT 3)
MRG 2 = LNK 2. GT. 0
(ARE RECORDS ON UNIT 11)
OPENFT
OPEN UNIT NETWORK
AS OUTPUT (REWIND
IT)
CALCULATE TOTAL NUMBER
OF ONE-WAY LINKS
IS THE
TOTAL NUMBER OF
LINKS > THAN
THE MAX.?
YES
PRINT MESSAGE THAT
THE NUMBER OF LINKS
EXCEEDS THE MAXIMUM.
ERR = ERR + 1
NO
WRT
WRITE PARAMETER RECORD
ON UNIT NETWORK
IS THE
LAST NODE
NUMBER > THE
MAXIMUM NODE
NUMBER?
YES
PRINT MESSAGE THAT THE
MAXIMUM NODE NUMBER IS
EXCEEDED.
ERR = ERR + 1
NO
WRT
WRITE HEADER RECORD
AND DATE ON UNIT NETWORK
SET INDEXES FOR NEXT
LINK FROM LINKS IN
CORE OR FROM RECORDS READ
FROM UNITS 3 OR 11 TO
FIRST LINK
ARE THERE
RECORDS ON
UNIT 3?
YES
READ A RECORD FROM
UNIT 3
NO
ARE THERE
RECORDS ON
UNIT 11?
YES
READ A RECORD FROM
UNIT 11
NO
GTLNK
UNPACK DATA FROM FIRST
LINK FROM UNIT 3
GTLNK
UNPACK DATA FROM
FIRST LINK IN CORE
GTLNK
UNPACK DATA FROM
FIRST LINK FROM
UNIT 11
ARE THERE
RECORDS ON
UNIT 3?
NO
SET SORT KEY FOR
LINK FROM UNIT 3 SO
IT WILL BE SKIPPED.
SET ANODE = 16383
YES
ARE THERE
RECORDS ON
UNIT 11?
NO
SET SORT KEY FOR THE
FIRST LINK FROM UNIT
11 SO THAT IT WILL BE
SKIPPED. SET ANODE =
16383
YES
REWIND UNIT 4 (NODE
NAMES)
V 1

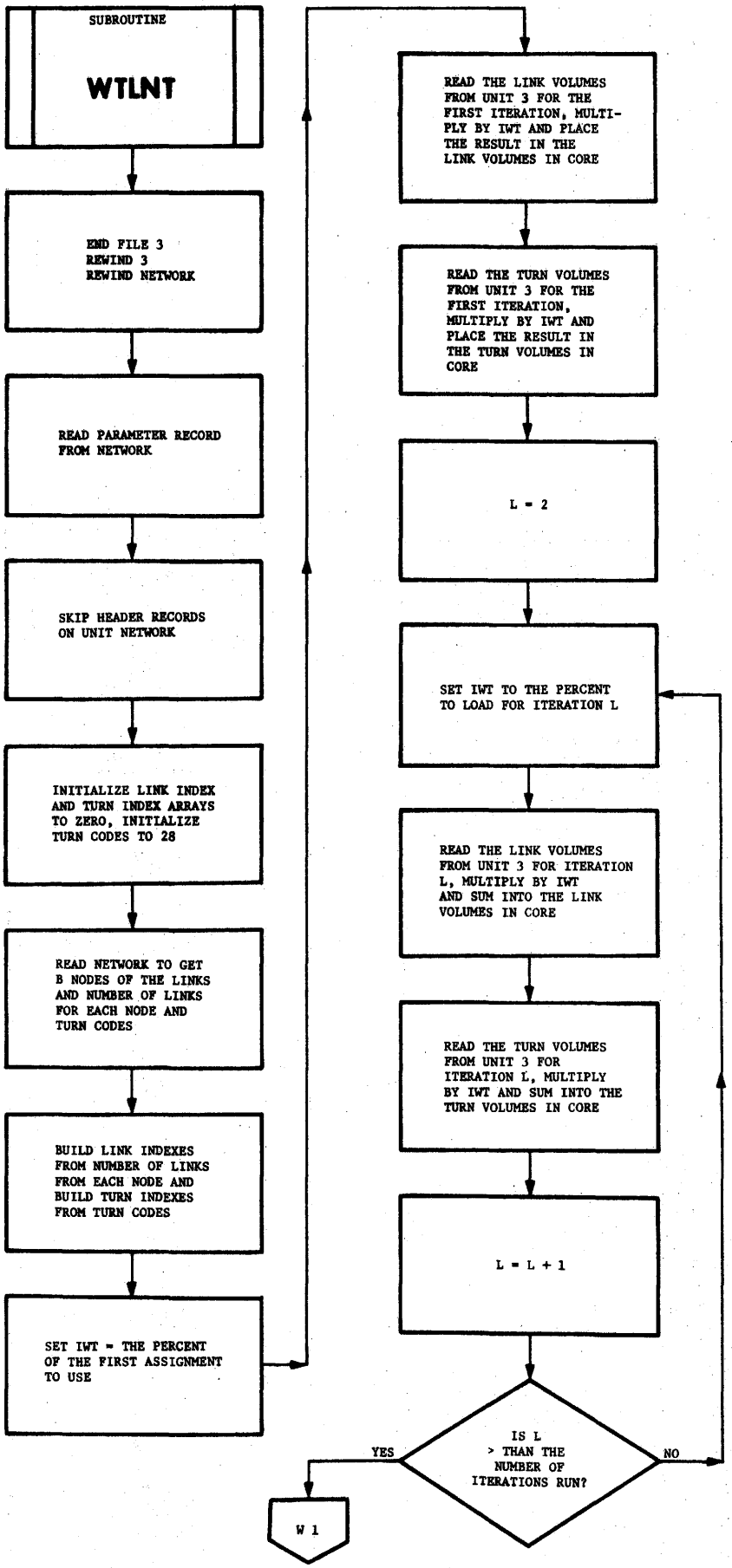

V 1
V 2
READ NEXT CARD FROM LINK DATA INPUT
SET COUNT OF LINKS FROM ANODE TO 0. (L = 0). PUT NEXT SMALLEST ANODE IN ANODE
IS IT AN N OR AN ENDNET CARD?
NO
YES
IS THE ANODE = 16383?
YES
V 5
NO
SET NODE NUMBER OF LAST NODE RECORD READ TO 16383
WHICH SORT KEY IS SMALLEST?
UNIT 3
CORE
UNIT 11
IS THE ANODE OF THIS LINK = ANODE?
NO
V 3
YES
IS THE ANODE OF THIS LINK = ANODE?
NO
V 3
YES
L = L + 1 SAVE THIS LINK AT INDEX L. INCREMENT TO GET NEXT LINK
L = L + 1 SAVE THIS LINK AT INDEX L. INCREMENT TO GET NEXT LINK
L = L + 1 SAVE THIS LINK AT INDEX L. INCREMENT TO GET NEXT LINK
GTLNK
UNPACK DATA FROM NEXT LINK FROM CORE
IS THE NEXT LINK IN THE LAST RECORD READ?
NO
YES
READ NEXT RECORD FROM UNIT 3. SET LOCATION TO GET NEXT LINK TO 0
IS THE NEXT LINK IN THE LAST RECORD READ?
NO
YES
READ NEXT RECORD FROM UNIT 11. SET LOCATION TO GET NEXT LINK FROM TO 0
GTLNK
UNPACK DATA FROM NEXT LINK FROM UNIT 3
GTLNK
UNPACK DATA FROM NEXT LINK FROM UNIT 11.

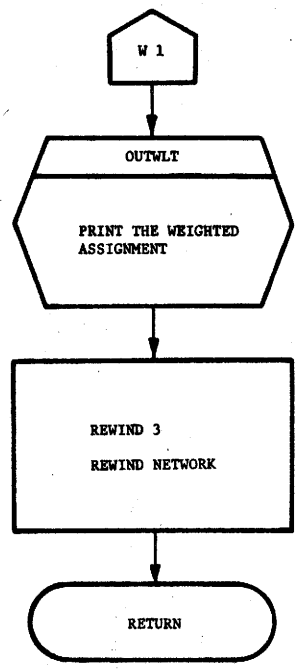

V 3
CONVERT NUMBER OF LINKS IN EACH LINK CLASS TO THE TURN MOVEMENT TYPE CODE
IS THE NODE IN THE NETWORK?
NO
PRINT MESSAGE THAT THE ANODE IS NOT IN THE NETWORK
ERR = ERR + 1
YES
WHAT TYPE OF NODE IS IT?
CENTROID
SET TURN TYPE CODE TO 28 AND ADD CENTROID CODE
FREEWAY
ADD CODE FOR FREEWAY TO TURN CODE
ARTERIAL
IS THERE A NODE NAME RECORD FOR THIS ANODE?
YES
MOVE NODE NAME TO AREA FOR THIS ANODE. READ NEXT NODE NAME RECORD.
NO
MOVE BLANKS TO THE NODE NAME FOR THIS NODE
IS THERE AN ANODE RECORD FOR THIS ANODE?
YES
GET COORDINATES AND KEEP NON ZERO SUB AREA CODE. READ NEW ANODE RECORD
NO
SET COORDINATES TO ZERO
WRT
WRITE NODE RECORD ON UNIT NETWORK
ARE THERE ANY DUPLICATE BNODES FROM THIS ANODE?
YES
PRINT AN ERROR MESSAGE FOR EACH DUPLICATE LINK AND ADD NUMBER OF DUPLICATIONS INTO ERR
NO
V 4


V 4
ARE THERE MORE THAN 6 LINKS FROM THIS ANODE?
YES
PRINT ERROR MESSAGE THAT THERE ARE MORE THAN SIX LINKS. ERR = ERR + 1
NO
IS THIS ANODE A CENTROID?
CENTROID
NOT CENTROID
IS THERE AT LEAST ONE WAY TO THE CENTROID AND ONE WAY FROM IT?
YES
IS THERE ONE WAY TO THE NODE AND ANOTHER WAY OUT?
YES
NO
NO
PRINT MESSAGE: ISOLATED CENTROID. ERR = ERR + 1
PRINT MESSAGE: ISOLATED NODE. ERR = ERR + 1
I = 1
WRT
WRITE A LINK RECORD FOR SAVED LINK I
I = I + 1
IS I > L
NO
YES
V 2


V 5
CLOSFT
CLOSE NETWORK DATA SET
REWIND NODE NAME DATA SET (UNIT 4)
WAS UNIT 3 READ FROM?
YES
REWIND 3
NO
WAS UNIT 11 READ FROM?
YES
REWIND 11
NO
RETURN


SUBROUTINE
WGTLD
SUM CONSTANTS FROM CURVE FIT
SCALE CONSTANTS TO 100 PERCENT AND CONVERT EACH TO INTEGER AND SUM NS = TOTAL PERCENT
100-NS : 0
<
=
>
FIND SMALLEST "T" VALUE FOR WHOSE PERCENTAGE IS ≥ 100 - NS
FIND THE LARGEST "T" VALUE
ADD 100 - NS TO THE PERCENTAGE FOUND ABOVE
CALCULATE THE WEIGHT FOR EACH ITERATION BY DIVIDING THE PERCENTAGE BY 100.0
ADD 100 - NS TO THE PERCENTAGE CORRESPONDING TO THE LARGEST "T" VALUE
PRINT THE PERCENTAGE WEIGHTS TO LOAD
RETURN


SUBROUTINE
WTLNT
END FILE 3
REWIND 3
REWIND NETWORK
READ PARAMETER RECORD FROM NETWORK
SKIP HEADER RECORDS ON UNIT NETWORK
INITIALIZE LINK INDEX AND TURN INDEX ARRAYS TO ZERO, INITIALIZE TURN CODES TO 28
READ NETWORK TO GET B NODES OF THE LINKS AND NUMBER OF LINKS FOR EACH NODE AND TURN CODES
BUILD LINK INDEXES FROM NUMBER OF LINKS FROM EACH NODE AND BUILD TURN INDEXES FROM TURN CODES
SET IWT = THE PERCENT OF THE FIRST ASSIGNMENT TO USE
READ THE LINK VOLUMES FROM UNIT 3 FOR THE FIRST ITERATION, MULTIPLY BY IWT AND PLACE THE RESULT IN THE LINK VOLUMES IN CORE
READ THE TURN VOLUMES FROM UNIT 3 FOR THE FIRST ITERATION, MULTIPLY BY IWT AND PLACE THE RESULT IN THE TURN VOLUMES IN CORE
L = 2
SET IWT TO THE PERCENT TO LOAD FOR ITERATION L
READ THE LINK VOLUMES FROM UNIT 3 FOR ITERATION L, MULTIPLY BY IWT AND SUM INTO THE LINK VOLUMES IN CORE
READ THE TURN VOLUMES FROM UNIT 3 FOR ITERATION L, MULTIPLY BY IWT AND SUM INTO THE TURN VOLUMES IN CORE
L = L + 1
IS L > THAN THE NUMBER OF ITERATIONS RUN?
YES
NO
W 1


W 1
OUTWLT
PRINT THE WEIGHTED ASSIGNMENT
REWIND 3
REWIND NETWORK
RETURN

S I G N I F I C A N T V A R I A B L E S

A N D A R R A Y S

LABELED COMMON

DESCRIPTIONS OF SIGNIFICANT
VARIABLES AND ARRAYS

LABELED COMMON

Twelve labeled common control sections are contained in the Texas Small Network Package. These labeled commons serve several important functions. Their primary function is, of course, to provide a convenient media for passing various variables and arrays between subroutines. They are also used to save certain variables and arrays as various subroutines are overlayed. They have also been used in a few instances to allign half-word arrays on full-word boundaries. Table 5 provides a cross reference of the labeled common control sections and the program control sections with which they are associated.

TABLE 5: CROSS REFERENCE OF LABELED COMMON CONTROL SECTIONS AND PROGRAM CONTROL SECTIONS

	LABELED COMMON											
PROGRAMS	ALLIGN	CAPRES	CAPREP	CD	DELETE	FILES	GROUP1	HEADR	OUTDCB	SDATE	STOP	VOLTP
ALCP		X						X				
BLDNET						X						
BLOCK DATA		X				X		X		X		
CLOSFT									X			
CRD						X		X		X		
CRDINT				X		X		X				
FRATAR						X						
GTLD		X		X		X		X			X	
LNKLST		X				X		X				
MAIN		X				X		X				
MERG						X						
MRGREC	X				X	X		X				
NEWNET					X	X						
OPENFT									X			
OUTLLT		X				X		X				
OUTLNT			X					X				
OUTNET						X		X				
OUTRIP						X		X				
OUTSLN								X				
OUTSNT								X				
OUTTRE								X				
OUTWLT						X		X				
PATHCL		X	X			X						
PATHSP						X	X					
PRPBLD		X										
PRPCTV						X						X
READVL						X						
RTPFL						X					X	
RTPLT						X						
SELECT								X				
SLOAD						X						
SUBFND												X
SUMEND						X		X				
SUMRY		X		X							X	
TREBLD			X									
UPDTNT		X				X						
VREC					X	X		X				
WGTLD		X						X				
WRT									X			
WTLNT						X						

DESCRIPTIONS OF VARIABLES AND ARRAYS

The purpose of the section is to provide information concerning the significant variables and arrays used in the package. For convenience, this information has been summarized in tables by subroutine. The programmer may, therefore, when reviewing the flowcharts and program listings of a given subroutine, refer to the table(s) summarizing the significant variables and/or arrays used in the subroutine. The tables summarizing the significant variables and arrays used in various subroutines, arranged in alphabetical order by the subroutine name, are as follows:

SUBROUTINE ALCP

In the following description the C field will be used to represent either the link COUNT field when it is used or the link CAPACITY field when it is used in ASSIGN SELF-BALANCING.

Variable	Contents
FN	The number of links used in the curve fit (the number of links with a nonzero C field which are not centroied connectors).
M	The number of iterations run in the ASSIGN SELF-BALANCING run at this point.
SY	The sum of the C fields except for centroid connectors.
SYY	The sum of the C fields squared except for the centroid connectors.

Control Variable	Value	Meaning
CNVRG	False	The ASSIGN SELF-BALANCING run should continue unless it has run the maximum number of iterations.
CNVRG	True	The ASSIGN SELF-BALANCING run should not run another iteration if it has run the minimum iterations.

Array	Contents
SX	The sum of the non-directional assigned link volumes for links with nonzero C fields except for centroid connectors for iterations 1 through M.
XY	The sum of the products of the non-directional assigned link volumes with the C fields except for centroid connectors for iterations 1 through M.
XX	The sum of the non-directional assigned link volumes squared for the links with nonzero C fields except for centroid connectors for iterations 1 through M.

SUBROUTINE BLDNET

Control Variables	Value	Meaning
FORMAT	False	Use the link card format written in the manual with from 1 to 4 nondirectional links per card.
FORMAT	True	Use the link card format that is used for PREPARE NETWORK.
EOF	False	An end of data set has not been reached on unit INLNK.
EOF	True	An end of data set has been reached on unit INLNK.

Variable	Contents
MILAGE	The sum of the milage of all the link data cards in units of 0.01 miles.
IZLINK	The number of nondirectional links with a zero link impedance in the network.

Array	Contents
SPEEDS	The number of nondirectional links with link speeds between 0 mph and 100 mph in increments of 1 mph.
KOUNT(I)	The number of links from node I in array LINKS.
INDEX(I)	The index in array LINKS where the links from node I are stored (or where they will be stored if there are any).
LINKS	Each element of this array is a structure of data items called a link.

Structure of an Element in array LINKS

Displacement Bits	Length Bits	Contents
0	1	Last link code (0 if not last links, 1 if last link from the A node).
1	19	Link impedance in 0.01 minute units.
20	12	B node of the link.

SUBROUTINE CMPVH

Variable	Contents
LSTJ	The largest jurisdiction number in the network.
NLD	The number of assignments on unit NEWNET.

Control Variable	Value	Meaning
NLD	1	Don't print the comparison of the last two assignments.
NLD	2 or greater	Print the comparison of the last two assignments.

Array	Contents
VMI (J,L)	Vehicle miles cross classified by jurisdiction + 1 used as the first index and three link classes second index. The three link classes are centroid connectors, arterials, and freeway links.
VHR (J, L)	Vehicle hours cross classified the same as VMI.
MI (J, L)	Network miles cross classified the same as VMI.
VM (J, F)	Vehicle miles cross classified by jurisdiction + 1 used as the first index and functional class + 1 used as the second index.
M (J, F)	Network miles cross classified the same as VM.
VMC (J, F)	Vehicle miles for links with a nonzero count field cross classified the same as VM.
MC (J, F)	Network miles for the links with a nonzero count field cross classified the same as VM.
VMCC (J, F)	Vehicle miles for links with a nonzero capacity field cross classified the same as VM.
MCC (J, F)	Network miles for the links with a nonzero capacity field cross classified the same as VM.

Array	Contents
FC (F)	The number of links with functional class + 1 used as index F in the network.
FN (R, J)	J = 1: Number of links with nonzero link counts by route; J = 2: Number of links with nonzero link capacities by route; J = 3: Number of links in the network by route.
SY (R, J)	J = 1: Sum of link counts by route code; J = 2: Sum of link capacities by route code; J = 3: Sum of nondirectional link volume from the previous assignment by route.
SYY (R, J)	J = 1: Sum of link counts squared by route code; J = 2: Sum of link capacities squared by route code; J = 3: Sum of nondirectional link volumes from the previous assignment squared by route code.
SX (R, J)	J = 1: Sum of nondirectional link volumes for this assignment for those links which have a nonzero count by route; J = 2: Sum of nondirectional link volumes for this assignment for those links which have a nonzero link capacity by route; J = 3: Sum of nondirectional link volumes for this assignment by route.
SXX (R, J)	J = 1: Sum of nondirectional link volumes squared for this assignment for those links which have a nonzero count by route; J = 2: Sum of nondirectional link volumes squared for this assignment for those links which have a nonzero link capacity by route; J = 3: Sum of nondirectional link volumes squared for this assignment by route code.
SXY (R, J)	J = 1: Sum of nondirectional link volumes from this assignment multiplied by link count by route; J = 2: Sum of nondirectional link volumes from this assignment multiplied by link capacity by routes; J = 3: Sum of nondirectional link volumes from this assignment multiplied by nondirectional link volumes from the previous assignment by route.
H1	The header record and date from the previous assignment.
H2	The header record and date from the last assignment.
HN	The header record and date of when the network was built.

SUBROUTINE CRD

Control Variable	Value	Meaning
I	1	$PREPARE NETWORK control card read.
I	2	$OUTPUT NETWORK control card read.
I	3	$PREPARE TRIP VOLUMES control card read.
I	4	$OUTPUT TRIP VOLUMES control card read.
I	5	$SUM TRIP ENDS control card read.
I	6	$ASSIGN control card read.
I	7	$BUILD TREES control card read.
I	8	$STOP control card read.
I	9	$ASSIGN SELECTED LINKS control card read.
I	10	$FRATAR FORECAST control card read.
I	11	$MERGE control card read.
I	12	$PREPARE SPIDER NETWORK control card read.
I	13	$OUTPUT SPIDER NETWORK control card read.
I	14	$ASSIGN SPIDER NETWORK control card read.
I	15	$ASSEMBLE NETWORK control card read.
I	16	$REVISE NETWORK control card read.
I	17	$ASSIGN SELF-DIVERTING or $ASSIGN SELF-BALANCING control card read.
I	18	$DELETE ASSIGNMENTS control card read.
I	19	$PLOT ROUTE PROFILES control card read.

Variable	Contents
INLNK	Variable unit number INLNK
INCTV	Variable unit number CTVIN
IVOL	Variable unit number CTVOUT
IFRAT	Variable unit number FRATAR
MRGOUT	Variable unit number MRGOUT
NET	Variable unit number NETWORK
NNET	Variable unit number NEWNET
MSEP	Variable unit number SEPARAT
IRTPFL	Variable unit number ROUTE

Array	Contents
MERGIN	Variable unit numbers for the six MERGIN units.
HEADER	The header which is printed on output.
DATE	The date that the program started executing.
RNAME	The 19 control card names.

SUBROUTINE CRDINT

Control Variable	Value	Meaning
SUM	False	Print header records from unit NETWORK.
SUM	True	Print header records from unit NEWNET.

Variable	Contents
NLD	The number of assignments which are on unit NETWORK if SUM is false or on unit NEWNET if SUM is true.

Array	Contents
LINK	A structure with a length of 16 + 4NLD bytes per record, the records are corridor intercept links.
LK	The same array as LINK except this is in half words.

Corridor Intercept Record

Bytes Displacement	Bytes Length	Contents
0	2	Corridor intercept
2	2	Anode of the link
4	2	Bnode of the link
6	2	Route code of the link
8	2	Functional class code of the link
10	2	Link speed
12	2	Count field of the link in units of 100 trips.
14	2	Capacity field of the link in units of 100 trips.
16	4	Nondirectional assigned volume for the first assignment.
	.	
	.	
	.	
12+4NLD	4	Nondirectional assigned volume for the last assignment.

SUBROUTINE FASPTH

Variable	Contents
NONDS	The number of nodes in the network.
LSART4	The last arterial node number times 4.

Array	Contents
TRNPTY	Turn penalty array, contains 0, TP, TP, 0 where TP is the turn penalty in units of 0.01 minutes.
INDEX (I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS.
LINKS	This array contains a link in each word, the links are structures which contain 5 data items.
LAMBDA (I)	This array contains the cummulative times to reach node I in units of 0.01 minutes.
IPATH (I)	This array is a structure, element I contains the next node in the path back from node I, the turn code, and a flag which indicates whether the node is in the sequence table or is a centroid.
ISEQ	This is the sequence table, it contains all of the node numbers of the active sites of where the tree is being built.

Links Structure

Displacement Bits	Length Bits	Contents
0	1	Last link flag (0 if not last link, 1 if last link or dummy one-way link).
1	1	Shaft code
2	1	Arrow code
3	3	Unused
6	14	Link impedance in units of 0.01 minutes.
20	12	Bnode of the link

IPATH array structure

Displacement Bits	Length Bits	Contents
0	1	Sequence entered flag (0 if not entered and if not a centroid, 1 if entered in the sequence table or a centroid).
1	7	Turn code
8	24	Path node

SUBROUTINE FRATAR

Variable	Contents
ITER	Number of Fratar iterations that have been run
A1	Input trip matrix unit number
A2	Output trip matrix unit number (A1 and A2 are switched at the end of each iteration)
A0	Unit CTVOUT
NOSUB	Number of subnets

Array	Contents
TSUM (I,J)	I = subnet number, J = the relative zone in the subnet, T sum is the trip generations or the production volume plus the attraction volume for each zone for the input trip matrix.
ESUM (I,J)	TSUM (I,J)* GFAC (I,J)/100 = the expected production + attraction volume.
GFAC (I,J)	Growth factor, the factor multiplied by the trip generations which is the desired future trip generations.
LFAC (I,J)	Is the trip generations produced by the last growth factors.
ITEST	Growth factor frequency table for the last iteration run.
VOL	Used to read the trip volumes from the input trip matrix and write them on the output trip matrix.
FCEN	First centroid in each subnet.
LCEN	Last centroid in each subnet.

SUBROUTINE GTLD

Control Variable	Value	Meaning
SUM	False	Don't produce a weighted assignment.
SUM	True	Produce a weighted assignment from weighted impedances and write a new flexible record data set for it.

Variable	Contents
NLD	The number of assignments which are on unit NETWORK.
ITER	The number of iterations run for ASSIGN SELF-BALANCING.
JMAX	The maximum jurisdiction number in the network.

Array	Contents
VMI (J, L)	Vehicle miles cross classified by jurisdiction + 1 used as the first index and three link classes second index. The three link classes are centroid connectors, arterials, and freeway links.
VHR (J, L)	Vehicle hours cross classified the same as VMI.
MI (J, L)	Network miles cross classified the same as VMI.
VM (J, F)	Vehicle miles cross classified by jurisdiction + 1 used as the first index and functional class + 1 used as the second index.
M (J, F)	Network miles cross classified the same as VM.
VMC (J, F)	Vehicle miles for links with a nonzero count field cross classified the same as VM.
MC (J, F)	Network miles for the links with a nonzero count field cross classified the same as VM.
VMCC (J, F)	Vehicle miles for links with a nonzero capacity field cross classified the same as VM.
MCC (J, F)	Network miles for the links with a nonzero capacity field cross classified the same as VM.

Array	Contents
FC (F)	The number of links with functional class + 1 used as index F in the network.
FN (R, J)	J = 1: Number of links with nonzero link counts by route; J = 2: Number of links with nonzero link capacities by route; J = 3: Number of links in the network by route.
SY (R, J)	J = 1: Sum of link counts by route code; J = 2: Sum of link capacities by route code; J = 3: Sum of nondirectional link volume from the previous assignment by route.
SYY (R, J)	J = 1: Sum of link counts squared by route code; J = 2: Sum of link capacities squared by route code; J = 3: Sum of nondirectional link volumes from the previous assignment squared by route code.
SX (R, J)	J = 1: Sum of nondirectional link volumes for this assignment for those links which have a nonzero count by route; J = 2: Sum of nondirectional link volumes for this assignment for those links which have a nonzero link capacity by route; J = 3: Sum of nondirectional link volumes for this assignment by route.
SXX (R, J)	J = 1: Sum of nondirectional link volumes squared for this assignment for those links which have a nonzero count by route; J = 2: Sum of nondirectional link volumes squared for this assignment for those links which have a nonzero link capacity by route; J = 3: Sum of nondirectional link volumes squared for this assignment by route code.
SXY (R, J)	J = 1: Sum of nondirectional link volumes from this assignment multiplied by link count by route; J = 2: Sum of nondirectional link volumes from this assignment multiplied by link capacity by routes; J = 3: Sum of nondirectional link volumes from this assignment multiplied by nondirectional link volumes from the previous assignment by route.
H1	The header record and date from the previous assignment.
H2	The header record and date from the last assignment.
HN	The header record and date of when the network was built.
WGT(J)	This array contains the weights in percentages to use on each iteration when SUM is true.

The following arrays and variables are summed for links with a nonzero count (or capacity) field. The *TURN card is used to specify whether the count or capacity field is used. It should also be noted that the following arrays and variables are not summed for centroid connectors.

Array	Contents
SX2(J)	Sum of the nondirectional link volumes for iteration J.
XY(J)	Sum of the nondirectional link volumes multiplied by the count (or capacity) field for iteration J.
XX(J, K)	Sum of the nondirectional link volume for iteration J multiplied by the nondirectional link volume for iteration K.

Variable	Contents
SY2	The sum of the count (or capacity) fields.
SYY2	The sum of the count (or capacity) fields squared.
FN2	The number of nonzero count (or capacity) fields for links which are not centroid connectors.

SUBROUTINE LNKLST

Variable	Contents
NA	The number of iterations run in an ASSIGN SELF-BALANCING run plus one if a weighted assignment has been produced.
NET	The Fortran unit on which the last assigned Flexible Record is written.

SUBROUTINES LOAD AND LOAD2

Control Variables	Value	Meaning
READSW	False	The last record of trip volumes read has been loaded.
READSW	True	The last record of trip volumes read has not been loaded.
EOFSW	False	An end of data set has not been reached on unit CTVOUT.
EOFSW	True	An end of data set has been reached on unit CTVOUT.

Variable	Contents
IV	Number of volume items in the last trip record read.
IFACT	First zone number minus 1.
LHOM	Origin zone of the last trip record read.
LNET	Origin subnet of LHOM (should be 1 for the Small Network Package).
NODES*	Last node number of the network.

Array	Contents
INDEX (I)	This array contains the Fortran type index for node I of where the links from node I start in array links.
LINKS	The same as array LINKS in subroutine FASPTH.
BUF	This array is a structure where each word of the array is an item containing the trip movement volume in the first 18 bits as an unsigned binary integer, and the destination zone number in the last 14 bits as an unsigned binary integer.

*This is the variable NODES1 in subroutine LOAD2

Array	Contents
VOL (I)	This is a half word array which has the same dimension as array LINKS and element I contains either the assigned directional link volume for link LINKS (I) or the index of where it is in array OVERF. The first bit of a VOL element is a flag bit, if it is zero, then the next 15 bits are on unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.
TRNTB (I)	This is a half word array which is either used to store turn volumes or indexes to where they are stored. The flag bit is the same as for array VOL and the next 15 bits are also treated the same as for array VOL.
XRTRN (J)	This is a half word array which contains unsigned 16 bit integers which are indexes into array TRNTB where the turn volumes for node J are stored.
PATH	This array is the same as array IPATH in subroutine FASPTH.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater tha 32767.

SUBROUTINE MRGREC

Variable	Contents
IL	This is the number of link records in array LINKS.
NX	This is the number of links written on unit 3.
LNK2	This is the number of links written on unit 11.
MAXTIM	This the maximum link time in 0.01 minute units.
MAXLNK	This is the maximum number of one-way links for a network.
MAXNDS	This is the maximum number of nodes for a network.
NOSUB	This is the number of subnets the network is in.

Arrays

Array	Length	Contents Contents
FSTN	4	First node of each subnet.
LSTC	4	Last centroid of each subnet.
LSTF	4	Last freeway of each subnet.
LSTA	4	Last arterial node of each subnet.
ARRAY	30004	Contains the sorted packed links array described in NEWNET.

Control Variables	Value	Action Implied	Location Where Set
FMT	False	Use old link data format	PRPNET, ASMNET, or REVNET
FMT	True	Use new link data format	PRPNET, ASMNET, or REVNET
LNKTMP	3	Write first sorted links on unit 3	Initialization of NEWNET
LNKTMP	11	Write second sorted links on unit 11	Set to 11 after sorted links are written on 3
LNKTMP	-1	If the sorted links area is filled up three times there are too many links and an attempt to write on unit -1 will be made	Set to -1 after sorted links are written on unit 11
ERROR	Number of Error detected in subroutines NEWNET, VREC, and MRGREC		

Array LINKS

Array LINKS is the array in which oneway internal link records are accumulated and sorted. These records are 22 bytes long and are stored by subroutine PTLNK and referenced by subroutine GTLNK. The format for these 22 byte records is as follows:

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
0	0	0	14	Anode number
1	6	0	2	Link class code 0 = two-way 1 = one-way out 2&3 = dummy link
2	0	0	15	Link data card count
3	7	0	1	Not mileage code 0 = Use in Vehicle Mile Summary 1 = Do not use in Vehicle Mile Summary
4	0	0	14	Bnode number
5	6	0	14	Count field in units of 100 trips
7	4	0	4	Jurisdiction code in hexadecimal
8	0	0	4	Functional class code in hexadecimal
8	3	0	7	Subarea code
9	3	0	14	Link Capacity in units of 100 trips
11	1	0	7	Speed in units of tenths of a mile per hour
12	0	0	10	Link distance in units of $\frac{1}{100}$ of a mile
13	2	0	7	Corridor intersect code
14	1	0	5	Route number
14	6	0	1	Shaft code, 0 = one direction 1 = other direction
14	7	0	1	Arrow code, 0 = one direction 1 = other direction
15	0	1	0	Unused
16	0	0	6	Link Impedance field, in units of $\frac{1}{100}$ minutes
16	6	0	1	Link delete code 0 = keep link 1 = delete link from updated Flexible Data Record
16	7	4	1	Unused

SUBROUTINE OUTLLT

Control Variable	Contents	Meaning
PRINT	False	Don't print the loaded network.
PRINT	True	Print the loaded network.
OUTN	False	Don't print the loaded network.
OUTN	True	Print the loaded network if variable RES is false or ITR is equal to 1.
RES	False	This is not an ASSIGN SELF-BALANCING iteration.
RES	True	This is an ASSIGN SELF-BALANCING iteration.
CAP	False	The COUNT field is used in an ASSIGN SELF-BALANCING RUN.
CAP	True	The Capacity field is used in an ASSIGN SELF-BALANCING run.

Variable	Contents
IOVER	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.
IPATH(I)	This array is a structure, element I contains the next node in the path back from node I, the turn code, and a flag which indicates whether the node is in the sequence table or is a centroid.
INDEX(I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS.
NODE	This array contains a link in each word, the links are structures which contain 5 data items.
ITR(I)	This is a half word array which is either used to store turn volumes or indexes to where they are stored. The flag bit is the same as for array VOL and the next 15 bits are also treated the same as for array VOL.
IXR(J)	This is a half word array which contains unsigned 16 bit integers which are indexes into array ITR where the turn volumes for node J are stored.

Array	Contents
VOL(I)	This is a half word array which has the same length as array LINKS and element I contains either the assigned directional link volume for link LINKS(I) or the index of where it is in array OVERF. The first bit of a VOL element is a flag bit, if it is zero, then the next 15 bits are on unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.

SUBROUTINE OUTNET

Variable	Contents
L	The Fortran unit number of the Flexible Data Record unit NETWORK.
LINES	The number of lines printed on the page being printed.

SUBROUTINE OUTSLN

Control Variable	Contents	Meaning
EOF	False	An end of data set has not been reached on unit 4.
EOF	True	An end of data set has been reached on unit 4.
FLG	False	All nodes which were not centroids had the same number of trips entering the node and leaving the node.
FLG	True	One or more nodes which were not centroids had a different number of trips entering than leaving the node.

Array	Contents
INDEX	This array contains the Fortran type index of where the links from node I start in array LINKS.
VOL(I)	This is a half word array which has the same dimension as array LINKS and element I contains either the assigned directional link volume for link LINKS(I) or the index of where it is in array OVERF. The first bit of a VOL element is a flag bit, if it is zero, then the next 15 bits are an unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.
LINKS	Each element of this array is a structure of data items called a link.

Structure of an Element in array LINKS

Displacement in Bits	Length in Bits	Contents
0	1	Last link code (0 if not last link; 1 if last link from the Anode).
1	19	Link impedance in 0.01 minute units.
20	12	Bnode of the link.

SUBROUTINE OUTSNT

Control Variable	Contents	Meaning
EOF	False	An end of data set has not been reached on unit 4.
EOF	True	An end of data set has been reached on unit 4.
SPIDER	'SPDR'	The data set on unit 1 was prepared by the PREPARE SPIDER NETWORK program.

Array	Contents
INDEX	This array contains the Fortran type index of where the links from node I start in array LINKS.
LINKS	Each element of this array is structure of data items called a link.

Structure of an Element in LINKS Array

Displacement in Bits	Length in Bits	Contents
0	1	Last link code (0 if not last link; 1 if last link from A node)
1	19	Link impedance in hundredths of a minute
20	12	B node of the link.

SUBROUTINE OUTWLT

Variable	Contents
NONDS	Last node number
IFACT	First centroid number minus 1.
NET	Is the Fortran unit number which contains a Flexible Record data set.

Array	Contents
INDEX(I)	This array contains the Fortran type index indicating the location where the links from node I start in array NODE.
NODE(I)	Each element of this array is a link. The first bit of each half word is the last link flag. If this bit is a 1, then this link is either the last link from the Anode or a dummy oneway link. The next 15 bits contain the Bnode of the link.
IPATH(I)	The Ith element of this array contains the turn code for node I as a half word integer.
VOL(I)	The Ith element of this array contains the directional weighted link volume multiplied by 100 for link NODE(I).
ITR(I)	Each element of ITR contains a directional weighted turn volume multiplied by 100. The turn volumes for node J begin at the index of IXR(J) and the number of turn volumes for node J are determined by the turn code IPATH(J).
IXR(J)	This is a half word array which contain unsigned 16 bit integers which are indices into array ITR where the turn volumes for node J are stored.

SUBROUTINE PATHCL

Variable	Contents
VOLF	Unit CTVOUT number.
NETD	Unit NETWORK number.

Control Variable	Contents	Meaning
READSW	False	The last record of trip volumes read has been loaded.
READSW	True	The last record of trip volumes read has not been loaded.
EOFSW	False	An end of data set has not been reached on unit CTVOUT.
EOFSW	True	An end of data set has been reached on unit CTVOUT.

Array	Contents
LAMBD1(I)	Used as the cumulative time to node I in subroutine FASPTH and OUTTRE and as a scratch array in subroutine LOAD.
SEQ	Used as a scratch array in subroutines FASPTH, LOAD, and LOAD2 (not used in PATHCL).
TRNTB1(I)	This is a half word array which is either used to store turn volumes or indexes to where they are stored. The flag bit is the same as for array VOL and the next 15 bits are also treated the same as for array VOL.
XR1TRN(J)	This is a half word array which contains unsigned 16 bit integers which are indexes into array TRNTB1 where the turn volumes for node J are stored.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.
VOL1(I)	This is a half word array which has the same dimension as array LINKS1 and element I contains either the assigned directional link volume for link LINKS1(I) or the index of

Array	Contents
VOL1(I) cont.	where it is in array OVERF. The first bit of a VOL1 element is a flag bit, if it is zero, then the next 15 bits are an unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.
INDEX1(I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS1.
LINKS1	This array contains a link in each word, the links are structures which contain 5 data items.
PATH1(I)	This array is a structure, element I contains the next node in the path back from node I, the turn code, and a flag which indicates whether the node is in the sequence table or is a centroid.

Links Structure

Displacement Bits	Length Bits	Contents
0	1	Last link flag (0 if not last link, 1 if last link or dummy oneway link).
1	1	Shaft code
2	1	Arrow code
3	3	Unused
6	14	Link impedance in units of 0.01 minutes.
20	12	Bnode of the link

SUBROUTINE PATHSP

Control Variable		Contents
VOLF	Univ CTVOUT	Unit CTVOUT number.

Array	Contents
INDEX1(I)	This array contains the Fortran type index for node I of where the links from node I start in array links.
BACK(I)	This array contains the path of the last tree built. For node I the contents of BACK(I) contain the previous node in the path from the origin node to node I.
LAMBDA(I)	This is a scratch array used by subroutines MOORE and SLOAD.
SUCC	This is a scratch array used by subroutine MOORE.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.
PRED	A list of nodes in descending cumulative time order in which the nodes were reached in the last tree built.
VOL(I)	This is a half word array which has the same dimension as array LINKS1 and element I contains either the assigned directional link volume for link LINKS1(I) or the index of where it is in array OVERF. The first bit of a VOL element is a flag bit, if it is zero, then the next 15 bits are an unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.
LINKS1	Each element of this array is a structure of data items called a link.

Structure of an Element in array LINKS1

Displacement in Bits	Length in Bits	Contents
0	1	Last link code (0 if not last link; 1 if last link from the A node).
1	19	Link impedance in 0.01 minute units.
20	12	B node of the link.

SUBROUTINE PRPBLD

Control Variable	Contents	Meaning
RES	False	This is not an ASSIGN SELF-BALANCING run.
RES	True	This is an ASSIGN SELF-BALANCING run.
CAPC	False	The count field is to be used by ASSIGN SELF-BALANCING.
CAPC	True	The capacity field is to be used by ASSIGN SELF-BALANCING.
W	False	An assignment using weighted impedances is not to be made.
W	True	An assignment using weighted impedances is to be made in ASSIGN SELF-BALANCING.
OUT (I,J)	False	Don't print the trees with origins between INDX1 (I,J) and INDX2 (I,J).
OUT (I,J)	True	Print the trees with origins between INDX1 (I,J) and INDX2 (I,J).

Variable	Contents
NOSUB	The number of subnetworks.
COUNT (I)	The number of ranges of trees to build in subnet I.
INDX1 (I,J)	The beginning of a range of trees to build in subnet I.
INDX2 (I,J)	The end of a range of trees to build in subnet I.

SUBROUTINE PRPNET

Logical Variables

Variable Name	Set	Action Implied	Where Tested
FMT	False	Use old link data format	NEWNET, VREC
REV	False	This is not a REVISE NETWORK run	PRPNET

Maximum Value Variables

Variable Name	Value	Meaning
MAXLK2	5455	This is the maximum number of oneway links in core.
MAXNDS	4000	This is the maximum last node number.
MAXLNK	16000	This is the maximum number of oneway links for the network.
MAXTIM	16383	This is the maximum link time in hundredths of a minute (i.e., 163.83 minutes).

Arrays

Name	Length	Contains
FSTN	4	First node of each subnet
LSTC	4	Last centroid of each subnet
LSTF	4	Last freeway node of each subnet
LSTA	4	Last arterial node of each subnet
ARRAY	3004	Contains the packed links array described as array LINKS in subroutine NEWNET.

When entry point ASMNET is used, the logical variables FMT and REV are set as follows:

Variable Name	Value Set	Action Implied	Where Tested
FMT	True	Use new link data format	NEWNET, VREC
REV	False	This is not a REVISE NETWORK run	PRPNET

When entry point REVNET is used, the logical variables FMT and REV are set as follows:

Variable Name	Value Set	Action Implied	Where Tested
FMT	True	Use new link data format	NEWNET, MRGREC
REV	True	This is a REVISE NETWORK run	PRPNET

SUBROUTINES RTPFL AND RTPLT

Control Variable	Contents	Meaning
END	False	There was enough room in array F for the first 10 routes.
END	True	There was not enough room in array F for the first 10 routes.
RTS (I)	False	Don't save the records read for route I in array F.
RTS (I)	True	Save the records read for route I in array F.

Variable	Contents
NRD	The number of words in array F used by one route record.
NWORDS	The length of array F in words.
NLD	The number of assignments on the NEWNET data set.

Array	Contents
B1 (I)	If B1 (I) is not zero, then there is a link for route RT2 between node I and node B1 (I).
B2 (I)	If B2 (I) is not zero, then there is a link for route RT2 between node I and node B2 (I).
NX1 (I)	NX1 (I) is the index into array F of where the record for the link represented by B1 (I) is stored.
NX2 (I)	NX2 (I) is the index into array F of where the record for the link represented by B2 (I) is stored.
F (I)	This is a full word array used to store a group of words and half words which are a single record for a link.
H (I)	This is a half word array equivalenced to array F.

Array	Contents
RTT (I)	Contains either the number of route records for route I or zero if the records are in array F or have been printed.
RT10 (I)	Contains the number of route records for route I for the first ten routes.

A route record has the following order of items and is stored in array F in the same order:

Displacement in bytes	Length in bytes	Contents
0	2	Route code
2	2	Anode number
4	2	Bnode number
6	2	link functional classification
8	2	link distance in 1/100 miles
10	2	link speed in tenths of a mile/hour
12	2	link count/100
14	2	link capacity/100
16	4	link nondirectional assigned volume for first assignment
⋮	4	
12+4NLD	4	link nondirectional assigned volume for the last assignment

SUBROUTINE SELECT

Control Variable	Contents	Meaning
OUT	True	no errors found in SELECT cards.
OUT	False	errors found in SELECT cards.

Array	Contents
INDEX (I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS.
LINKS	This array contains a link in each word, the links are structures which contain 5 data items.

Links Structure

Displacement Bits	Length Bits	Contents
0	1	Last link flag (0 if not last link, 1 if last link or dummy one-way link).
1	1	Shaft code
2	1	Arrow code
3	1	Selected link code (1 if selected link)
4	2	Unused
6	14	Link impedance in units of 0.01 minutes.
20	12	Bnode of the link

SUBROUTINE SLOAD

Control Variable	Contents	Meaning
EOF	False	The end of data set on unit CTVOUT has not been reached.
EOF	True	The end of data set on unit CTVOUT has been reached.

Variable	Contents
NODES	Last node number.
LHOM	Origin node of last trip record read.
COUNT	Number of trip items in last trip record read.
IOVR	Number of words used in the OVERF array, (number of directional volume greater than 32767).

Array	Contents
INDEX	The same as array INDEX in subroutine BLDNET.
LINKS	The same as array LINKS in subroutine BLDNET.
VOL	The same as array VOL in subroutine LOAD.
PRED	A list of nodes in descending cumulative time order in which the nodes were reached in the last tree built.
OVERF	The same as array OVERF in subroutine LOAD.
BACK (I)	This array contains the path of the last tree built. For node I the contents of BACK (I) contain the previous node in the path from the origin node to node I.

SUBROUTINE SUMEND

Array	Contents
IORG (I)	The sum of all trip volumes with the origin I except for the intrazonal volume for I.
IDEST (I)	The sum of all trip volumes with the destination I except for the intrazonal volume for I.
IIN (I)	The number of nonzero trip volumes with destination I.
IOUT (I)	The number of nonzero trip volumes with origin I.
INTRA (I)	Intrazonal volume for zone I.
ISUB (I)	Number of zones in subnet I.
IFSTND (I)	The first zone in subnet I.
LSTND (I)	Last zone in subnet I.

Variable	Contents
NOSUB	Number of subnets

SUBROUTINE TREBLD

Control Variable	Contents	Meaning
TREES	True	Build trees, but don't load the network or print the loaded network.
TREES	False	Build trees and load trips.
SEL	False	Don't read select cards and don't write the selected links data set.
SEL	True	Read select cards and write unit SELTRP.
OUTN	True	Print the loaded network.

Subroutine Entry Point	TREES	SEL	OUTN
TREBLD	False	False	True
TREE	True	False	-
SELLD	False	True	-

SUBROUTINE TRN

Control Array	Contents	Meaning
TL (I,J)	False	Don't print turn movement TM (I,J).
TL (I,J)	True	Print turn movement TM (I,J)
TM (I,J)	-1	The turning movement TM (I,J) is unknown.
TM (I,J)	≥ 0	TM (I,J) is a turning movement volume.

Variable	Contents
NODE	Node number to get directional volumes for and calculate turn movements for.
IND	Turn code for NODE (the turn codes are explained in the Other Information section).
N	Number of nodes connected to NODE.

Array	Contents
TM (I,J)	Turn movement between the Ith node and the Jth node connected to NODE.
NDIR (I)	Nondirectional link volumes for the links connected to NODE.
IDIR (I)	Directional link volumes for the links connected to NODE.
CH (I)	Directional link volumes for the links going in the direction of the nodes connected to NODE to NODE.
LINKS (I)	This array contains links which contain the Bnode in bits 1 thru 15 of the half word and a last link or dummy link indicator in bit 0.
TRNTB	This array contains the turn volumes saved, they are indexed by array XRTRN.

Array	Contents
TRNCD (I)	TRNCD (I) contains the turn code for node I.
VOL (I)	VOL (I) contains the directional link volumes for LINKS (I).
KC (IND)	A table indexed by the turn code which has the number of one-way links out from NODE.
KR (IND)	A table indexed by the turn code which has the number of one-way links into NODE.
INDEX (I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS.
XRTRN (J)	This is a half word array which contains unsized 16 bit integers which are indices into array TRNTB where the turn volumes for node J are stored.

The following arrays are used to place the turning movements which have been saved in ARRAY TM before the other turning movements are calculated. When a location in the following tables is not negative, the following action is taken: TM (I,J) = TRNTB (XRTRN (NODE) + IDSPXX (I,J)). If the IDSPXX (I,J) position is negative, a zero is placed in TM (I,J). The XX part of the IDSPXX array above varies.

Array	Used for turn code
IDSP3	10
IDSP41	13, 17, 18, 20, 22
IDSP42	21
IDSP43	23, 24
IDSP44	25
IDSP5	26
IDSP6	27

SUBROUTINE TURNM

Control Array	Contents	Meaning
TL (I,J)	False	Don't print turn movement TM (I,J).
TL (I,J)	True	Print turn movement TM (I,J).
TM (I,J)	-1	The turning movement TM (I,J) is unknown.
TM (I,J)	≥ 0	TM (I,J) is a turning movement volume.

Variable	Contents
NODE	Node number to get directional volumes for and calculate turn movements for.
IND	Turn code for NODE (the turn codes are explained in the Other Information section).
N	Number of nodes connected to NODE.

Array	Contents
TM (I,J)	Turn movement between the Ith node and the Jth node connected to NODE.
NDIR (I)	Nondirectional link volumes for the links connected to NODE.
IDIR (I)	Directional link volumes for the links connected to NODE.
CH (I)	Directional link volumes for the links going in the direction of the nodes connected to NODE to NODE.
KC (IND)	A table indexed by the turn code which has the number of one-way links out from NODE.
KR (IND)	A table indexed by the turn code which has the number of one-way links into NODE.
IPATH (I)	This array is a structure, element I contains the next node in the path back from node I, the turn code, and a flag which indicates whether the node is in the sequence table or is a centroid.

Array	Contents
INDEX (I)	This array contains the Fortran type index indicating the location where the links from node I begin in array LINKS.
LINKS	This array contains a link in each word, the links are structures which contain 5 data items.
VOL (I)	This is a half word array which has the same dimension as array LINKS and element I contains either the assigned directional link volumes for link LINKS (I) or the index of where it is in array OVERF. The first bit of a VOL element is a flag bit, if it is zero, then the next 15 bits are on unsigned binary integer which is a link volume. If the flag bit is 1, then the next 15 bits are an unsigned binary integer which is an index into array OVERF where the link volume is stored.
TRNTB (I)	This is a half word array which is either used to store turn volumes or indexes to where they are stored. The flag bit is the same as for array VOL and the next 15 bits are also treated the same as for array VOL.
XRTRN (J)	This is a half word array which contains unsigned 16 bit integers which are indexes into array TRNTB where the turn volumes for node J are stored.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.

IPATH array structure

Displacement Bits	Length Bits	Contents
0	1	Sequence entered flag (0 if not entered and if not a centroid, 1 if entered in the sequence table or a centroid).
1	7	Turn code
8	24	Path node

Links Structure

Displacement Bits	Length Bits	Contents
0	1	Last link flag (0 if not last link, 1 if last link or dummy one-way link).
1	1	Shaft code
2	1	Arrow code
3	3	Unused
6	14	Link impedance in units of 0.01 minutes.
20	12	Bnode of the link

The following arrays are used to place the turning movements which have been saved in array TM before the other turning movements are calculated. When a location in the following table is not negative, the following action is taken: TM (I,J) = TRNTB (XRTRN(NODE) + IDSPXX(I,J)). If the half word from TRNTB is negative, then the lower 15 bits are used as an index into the OVERF array to get the turn volume. If the IDSPXX(I,J) position is negative, a zero is placed in TM (I,J). The XX part of the IDSPXX array above varies.

Array	Used for turn codes
IDSP3	10
IDSP41	13, 17, 18, 20, 22
IDSP42	21
IDSP43	23, 24
IDSP44	25
IDSP5	26
IDSP6	27

SUBROUTINE UPDTNT

Control Variable	Contents	Meaning
DLT	False	There are no errors in the parameter cards read.
DLT	True	There are one or more errors in the parameter cards read for DELETE ASSIGNMENTS. The program will continue reading control cards but it will end execution with a STOP 3 when the next card with a $ character is column 1 or and *END card is read.
IMPD	False	An *IMPEDANCE parameter card has not been read.
IMPD	True	An *IMPEDANCE parameter card has been read.
SLF	False	An *ADJUST parameter card has not been read.
SLF	True	An *ADJUST parameter card has been read.

Variable	Contents
NMPD	The assignment number of the assignment which is to be the new link impedance if IMPD is true or from which the impedance update function using the count field is to be used to calculate a new set of link impedances.

SUBROUTINE VREC

Variable	Contents
IL	This is the number of link records in array LINKS.
LNK1	This is the number of links written on unit 3.
LNK2	This is the number of links written on unit 11.
MAXTIM	This is the maximum link time in 0.01 minute units.
MAXLNK	This is the maximum number of one-way links for a network.
MAXNDS	This is the maximum number of nodes for a network.
NOSUB	This is the number of subnets the network is in.
ERR	This is the number of errors found in processing the link data

Arrays

Array	Length	Contents
FSTND	4	First node of each subnet.
LSTCEN	4	Last centroid of each subnet.
LSTFWY	4	Last freeway of each subnet.
LSTART	4	Last arterial node of each subnet.
LINKS	30004	Contains the sorted packed links array described in NEWNET.
ARRAY	220	Contains one record from unit 3 of 40 packed links.
ARRAY2	220	Contains one record from unit 11 of 40 packed links.

SUBROUTINE WTLNT

Variable	Contents
ITER	Number of iterations run for ASSIGN SELF-BALANCING

Array	Contents
INDEX (I)	This array contains the Fortran type index of where the links from node I start in array BNODE.
BNODE (I)	Each element of this array is a link. The first bit of each half word is the last link flag. If this bit is a 1, then this link is either the last link from the Anode or a dummy one-way link. The next 15 bits contain the Bnode of the link.
TRNCD (I)	TRNCD (I) contains the turn code for node I as a half word integer.
VOL (I)	The element of VOL (I) contains the directional weighted link volume multiplied by 100 for link BNODE (I).
TRN (I)	Each element of TRN contains a directional weighted turn volume multiplied by 100. The turn volumes for node J begin at the index of XRTRN (J) and the number of turn volumes for node J are determined by the turn code TRNCD (J).
XRTRN(J)	This is a half word array which contains unsigned 16 bit integers which are indices into array TRN where the turn volumes for node J are stored.

D A T A S E T S A N D
D A T A S E T F O R M A T S

DATA SETS

DATA SET FORMATS

OUTPUT SELECTED LINKS

DATA SETS

Two categories of data sets are associated with the Texas Small Network Package: relocatable data sets and fixed data sets. The unit numbers associated with relocatable data sets may be changed either by the use of unit control cards or, in some instances, by the execution of some programs such as ASSIGN SELF-BALANCING. A cross reference of the data sets with associated programs is given in Table 6.

DATA SET FORMATS

There are twelve basic formats associated with data sets used by the package. These twelve format types are:

FORMAT TYPE	FORMAT TYPE CODE
Trip Volumes Data Set	B
Flexible Record Data Set	F
Separation Matrix Data Set	I
Selected Interchanges Data Set	L
Node Names Data Set	N
Calcomp Plot Tape	P
Route Data Set	R
Spider Network Data Set	S
Trip Matrix Data Set	T
Scratch Node Names Data Set	X
Scratch Packed Links Data Set	Y
Scratch Multiple Assignments Data Set	Z

The format type codes (indicated above) are used in the cross reference contained in Table 7 to indicate the format types used with each data set

TABLE 6: CROSS REFERENCE OF DATA SETS WITH ASSOCIATED PROGRAMS

Data Set Identification	Relocatable Data Sets										Fixed Data Sets							
	INLNK	CTVIN	CTVOUT	FRATAR	MERGOUT	MERGIN	NETWORK	ROUTE	NEWNET	SEPARAT	Scratch	Scratch	Scratch	Scratch	Network	Scratch	SELTRP	PLOTTAPE
(Default) Unit Number	5	10	8	16	**	**	1	25	9	20	3	4	17	11	12	13	***	***
PREPARE NETWORK	I						0				I/0	I/0		I/0				
ASSEMBLE NETWORK	I						0				I/0	I/0		I/0				
REVISE NETWORK	I						0				I/0	I/0		I/0	I	I/0		
OUTPUT NETWORK							I											
DELETE ASSIGNMENTS							0								I			
PREPARE TRIP VOLUMES		I	0															
OUTPUT TRIP VOLUMES			I															
BUILD TREES							I			0								
ASSIGN			I				I	I/0	0	0								
ASSIGN SELF-BALANCING			I				I/0	I/0	I/0	0	I/0							
ASSIGN SELECTED LINKS			I				I	I/0	0	0							0	
PLOT ROUTE PROFILES								I										0
FRATAR FORECAST****			I	I/0									I/0					
SUM TRIP ENDS			I															
MERGE					0	I												
PREPARE SPIDER NETWORK	I						0*					0						
OUTPUT SPIDER NETWORK							I*					I						
ASSIGN SPIDER NETWORK			I				I*					I						

I = Input Data Set

0 = Output Data Set

* For these programs this data set is fixed to unit 1.

** No default option exists for the MERGE program. Appropriate Unit Designation Cards must be provided by the user.

*** Assembly language program reference.

**** The FRATAR FORECAST program sets the CTVOUT unit to the same unit as FRATAR.

Note: Some of the output data sets may be suppressed by use of the DD DUMMY option in the JCL.

TABLE 7: CROSS REFERENCE OF DATA SETS WITH ASSOCIATED PROGRAMS INDICATING THE DATA SET FORMAT TYPES

Data Set Identification	Relocatable Data Sets									Fixed Data Sets							
	CTVIN	CTVOUT	FRATAR	MERGOUT	MERGIN	NETWORK	ROUTE	NEWNET	SEPARAT	Scratch	Scratch	Scratch	Scratch	Network	Scratch	SELTRP	PLOTTAPE
(Default) Unit Number	10	8	16	**	**	1	25	9	20	3	4	17	11	12	13	***	***
PREPARE NETWORK						F				Y	X		Y				
ASSEMBLE NETWORK						F				Y	X		Y				
REVISE NETWORK						F				Y	X		Y	F	F		
OUTPUT NETWORK						F											
DELETE ASSIGNMENTS						F								F			
PREPARE TRIP VOLUMES	B	T															
OUTPUT TRIP VOLUMES		T															
BUILD TREES						F			I								
ASSIGN		T				F	R	F	I								
ASSIGN SELF-BALANCING		T				F	R	F	I	Z							
ASSIGN SELECTED LINKS		T				F	R	F	I							L	
PLOT ROUTE PROFILES							R										P
FRATAR FORECAST****		T	T									T					
SUM TRIP ENDS		T															
MERGE				T	T												
PREPARE SPIDER NETWORK						S*					N						
OUTPUT SPIDER NETWORK						S*					N						
ASSIGN SPIDER NETWORK		T				S*					N						

* For these programs this data set is fixed to unit 1.

** No default option exists for the MERGE program. Appropriate Unit Designation Cards must be provided by the user.

*** Assembly language program reference.

**** The FRATAR FORECAST program sets the CTVOUT unit to the same unit as FRATAR.

Note: Some of the output data sets may be suppressed by use of the DD DUMMY option in the JCL.

and its associated programs. As can be seen from Table 7, some of the data sets have two different formats associated with them depending on the user program option being executed. Likewise, several of the data sets may have the same format as in the case of the trip matrix data set format. In order to determine the format for a given data set, the programmer should:

- Reference Table 7 to determine which of the twelve formats is associated with the data set of interest.
- Refer to the detailed description of the format.

The detailed descriptions of eleven* of the twelve formats are as follows:

*The format for the Calcomp plot tape (format type code: P) has not been included.

TRIP VOLUMES DATA SET
(Format Type Code: B)

Trip Volume Record

Displacement Bytes	Length Bytes	Contents
0	6	Zone of Origin
6	6	Zone of Destination
12	6	24-hour volume
18	6	AM-peak volume
24	6	PM-peak volume

Each field in the record is in EBCDIC and these records must be sorted into ascending order on a key of the first 12 bytes. The records should be in Fixed length or Fixed Blocked format. The minimum length of the records is 18 bytes if the 24-hour volume is used, 24 bytes if the AM-peak volume is used, or 30 bytes if the PM-peak volume is used.

End of Data Set Indicator Record

Displacement Bytes	Length Bytes	Contents
0	1	"V"
1	N - 1	blanks

N is the minimum length for a trip volume record. This record is only required if this data set is on cards and is read from unit 5 and it must follow the last Trip Volume record.

FLEXIBLE RECORD DATA SET

(Format Type Code: F)

Parameter Record (One record)

Bytes Displacement	Length	Contents
0	4	Number of Subnetworks in the Network
4	4	Number of Assignments
8	4	Number of directional links in the Network
12	4	First Centroid in Subnetwork 1
16	4	Last Centroid in Subnetwork 1
20	4	Last Arterial node in Subnetwork 1
24	4	Last Freeway node in Subnetwork 1
.		
.		
.		

(The last four items are repeated once for each subnetwork)

Heading record (One from network preparation and one from each assignment)

Bytes Displacement	Length	Contents
0	80	Heading record in EBCDIC
80	12	Processing date

Anode record (One for each Anode; the records are in sorted order on the Anode number; each Anode record is followed by the Link records which are connected to it.)

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
0	0	2	0	Anode number
2	0	2	0	Number of links connected to this node
4	0	0	1	Centroid flag (One if it is a centroid)
4	1	0	1	Freeway flag (One if it is a Freeway)
4	2	0	6	Turning movement type code
5	0	3	0	Not used
8	0	2	0	X coordinate of Anode
10	0	2	0	Y coordinate of Anode
12	0	2	0	Subarea code of Anode
14	0	20	0	Anode name in EBCDIC

Link Record (There is one link record for each link connected to a node; the link records follow the Anode to which they are connected)

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
0	0	0	1	Last Link from Anode flag
0	1	0	1	Shaft flag 0 = one direction 1 = other direction

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
0	2	0	1	Arrow flag 0 = one direction 1 = other direction
0	3	0	1	Not used
0	4	0	14	Link time in hundredths of a minute
0	18	0	14	Bnode of Link
4	0	0	4	Jurisdiction code of Anode
4	4	0	14	Distance of Link in hundredths of a mile
4	18	0	14	Speed in tenths of a mile/hour
8	0	2	0	Functional class (Codes 0 thru 15)
10	0	2	0	Route number (Codes 0 thru 99)
12	0	2	0	Corridor intercept
14	0	2	0	Duplicate Mileage Eliminator flag (One if link is to be eliminated from mileage summaries)
16	0	2	0	Link Volume
18	0	2	0	Link Capacity
20	0	4	0	Link impedance used on first assignment
24	0	4	0	Nondirectional Link volume from first assignment

.
.
.

(The last two items are repeated for each assignment, the above two are not present on a Flexible Record with no assignments)

SEPARATION MATRIX DATA SET

(Format Type Code: I)

Parameter Record

Byte Displacement	Length in Bytes	Contents
0	4	Number of zones
4	4	Zero
.	.	.
.	.	.
.	.	.
4 (number of zones)-4	4	Zero

Separation Record

Bytes Displacement	Length in Bytes	Contents
0	4	Time to Zone 1
4	4	Time to Zone 2
.	.	.
.	.	.
.	.	.
4 (number of zones)-4	4	Time to the last zone

The time is in hundredths of a minute. If a zone is not reached, its time field will be 16,777,215 hundredths of a minute. The separation records will be in the same order as the trees that are built.

SELECTED INTERCHANGES DATA SET

(Format Type Code: L)

Header Records

Bytes Displacement	Length in Bytes	Contents
0	2	Zeros
2	2	2I + 1
4	8	Columns 8I + 1 to 8I + 7 of the Header Line

There are 12 header records (I = 0, 11); each header record has eight bytes of the header line except the last record which has four bytes of the header line.

Select Record

Bytes Displacement	Length in Bytes	Contents
0	2	Link Index of the Selected Link*
2	2	Zeros
4	2	Percent of Trip Volumes to Print for this Selected Link
6	2	Smallest Node of Selected Link
8	2	Largest Node of Selected Link
10	2	Cut of Volume for Printing
12	2	Number of Trip Interchanges to print

*This is the index of the directional link from the smallest node of this selected link to the largest node of this selected link.

Interchange Record

Bytes Displacement	Length in Bytes	Contents
0	2	Link Index of Selected Link*
2	2	First Zone of the Interchange
4	2	Second Zone of the Interchange
6	4	Number of Trips in the Interchange
10	4	Zeros
14	2	Trip Direction Code

Trip Direction Code	Direction of Interchange	Direction of Trip Through Selected Link
10	First Zone to Second Zone	Small Node number to Large Node number
2	First Zone to Second Zone	Large Node number to Small Node number

Interchange Record

Bytes Displacement	Length of Bytes	Contents
0	2	Link Index of Selected Link*
2	2	First Zone of the Interchange
4	2	Second Zone of the Interchange
6	4	Zeros
10	4	Number of Trips in the Interchange
14	2	Trip Direction Code

Trip Direction Code	Direction of Interchange	Direction of Trip Through Selected Link
1	Second Zone to First Zone	Small Node number to Large Node number
5	Second Zone to First Zone	Large Node number to Small Node number

*These records are written fixed blocked 18 bytes long. They are 18 bytes long so that they can be sorted.

NODE NAMES DATA SET*

(Format Type Code: N)

Node Name Records

Column Displacement	Length in Columns	Contents
0	20	Node Name
20	4	Node Number (4 byte integer)

There is one Node Name Record for each different node name found in the Link Data Cards. The Link Data Cards should be in ascending order on the first node number.

*This data set uses FORTRAN formatted I/O.

ROUTE DATA SET

(Format Type Code: R)

Parameter Record

Displacement Bytes	Length Bytes	Contents
0	4	NLS = the Number of Assignments
4	4* (NLS + 3)	Unused

Header Records

Displacement Bytes	Length Bytes	Contents
0	4	Sort Key = 100* (Assignment number + 1) + J
4	12	Twelve bytes of the header
16	4* NLS	Unused

There are 8 of the Header records for each Header that is on a Flexible Record. The J in the Sort Key of the above records is 1, 4, 7, 10, 13, 16, 19, 22 and is the index of where the three words should be read into the header array in core when they are read. The record where J = 22 contains only two words of the header. The location that would be the third word is filled by 4 bytes of a 0 integer. The assignment number for the header record when the Flexible Record was built is set to 0. The above records are repeated for each assignment.

Route Records

Displacement Bytes	Length Bytes	Contents
0	2	Route Code
2	2	Anode of the Link
4	2	Bnode of the Link
6	2	Functional Class Code
8	2	Distance of the link in 0.01 mile units.
10	2	Speed of the link in 0.1 mile/hour units
12	2	Count field in units of 100 trips
14	2	Capacity in units of 100 trips
16	4	Nondirectional Assigned volume for the first assignment
⋮	4	⋮
12 + NLS*4	4	Nondirectional Assigned volume for the NLS assignment

One Route record is written for each link that has a route code where the Anode is less than the Bnode.

SPIDER NETWORK DATA SET

(Format Type Code: S)

Subnet Record

Byte Displacement	Length in Bytes	Contents
0	4	Number of Subnets (Set to 1)
4	4	Network Speed in miles/hour
8	4	Literal 'SPDR'

Network Parameter Record

Byte Displacement	Length in Bytes	Contents
0	4	Subnet Number (Set to 1)
4	4	Number of Nodes
8	4	First Node (Set to 1)
12	4	1
16	4	Last Node
20	4	Last Node
24	4	0
28	4	0
32	4	Number of oneway links

Index Record

Byte Displacement	Length in Bytes	Contents
0	2	Link index of node N
2	2	Link index of node N + 1
	⋮	
398	2	Link index of node N + 199

There are 200 indices in each record except the last one. The last record contains the number of indices which is the number of nodes taken modulus 200 plus one. N starts at 1 for the first record and is incremented by 200 for each additional record necessary.

Time Link Records

Byte Displacement	Length in Bytes	Contents
0	4	Oneway Link
4	4	Oneway Link
⋮	⋮	⋮
796	4	Oneway Link

The format of a Oneway Link is:

Bit Displacement	Length in Bits	Contents
0	1	Last Link Flag (Contains 1 if it is either the last link from the Anode or if it is indicating a dummy Link to Anode.)
1	1	Shaft flag 0 = one direction (could be East-West) 1 = the other direction (could be North-South)
2	1	Arrow Flag
3	3	Not used
4	14	Link Time in hundredths of a minute
20	14	Bnode of Link

The Anode of the Link must be used as an index into the Index array to get the index where the links from the Anode start in the Time Link Array. If an Anode has no links connected to it then INDEX(ANODE) = INDEX(ANODE + 1). The last Time Link Record may be less than 200 words since it will contain only the remaining links in the network. The Links from one Anode are in the following order: oneway out, twoway, and dummy oneway in. Within each class of oneway links, the links are in the order of the link data cards.

Turn Type Records

Byte Displacement	Length in Bytes	Contents
0	4	For node N, the first two bits are zero, the next six bits contain the turn type code for the node which is set to 28, and the next 24 bits contain zeros.
4	4	For node N + 1 with the above information types.
⋮	⋮	
796	4	For node N + 199 with the above information types.

There is a turn type word for each node from node 1 to the last node in the network. All turn type codes are 28 which indicate no turns are to be saved. This array is broken up into 200 word records as shown above.

TRIP MATRIX DATA SET

(Format Type Code: T)

Header Record

Displacement	Length	Contents
0	4	Number of Subnetworks
4	4	First centroid in Subnet I
8	4	Last centroid in Subnet I

The last two items are repeated for the number of subnets where I = 1,N.

Trip Record

Displacement	Length	Contents
0	4	Origin zone of all interchanges in this record
4	4	Subnet of the origin zone
8	4	N-Number of interchanges in this record (from 1 to 100)
12	4	Interchange item
⋮	⋮	⋮
8+4N	4	Interchange item

The interchange item is an 18 bit interchange volume followed by a 14-bit destination zone number.

The trip records are in sort on the origin zone and the interchange items for each origin are in sort on the destination zones.

SCRATCH NODE NAMES DATA SET

(Format Type Code: X)

Node Name Record

Displacement Bytes	Length Bytes	Contents
0	4	Anode number as a 4 byte integer
4	20	Node name

The node name records are written in ascending order of node numbers.

SCRATCH PACKED LINKS DATA SET

(Format Type Code: Y)

This data set is made up of records which contain 40 link records. Thses 40 link records are in the 22 byte format used in the LINKS array in Logical Division 1. The link records are sorted on the key of Anode, Link class, and Link data card count in ascending order for both Unit 3 and Unit 11 separately. The format for the 22 byte link records is as follows:

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
0	0	0	14	Anode number
1	6	0	2	Link class code 0 = twoway 1 = oneway out 2 & 3 = dummy link
2	0	0	15	Link data card count
3	7	0	1	Mileage code 0 = Use in Vehicle Mileage Summary 1 = Do not use in Vehicle Mileage Summary
4	0	0	14	Bnode number
5	6	0	14	Count field in units of 100 trips
7	4	0	4	Jurisdiction code in hexadecimal
8	0	0	4	Functional class code in hexadecimal
8	4	0	7	Subarea code
9	3	0	14	Link Capacity in units of 100 trips
11	1	0	7	Speed in units of tenths of a mile per hour

Link Record Format (continued)

Displacement		Length		Contents
Bytes	Bits	Bytes	Bits	
12	0	0	10	Link distance in units of $\frac{1}{100}$ of a mile
13	2	0	7	Corridor intersect code
14	1	0	5	Route number
14	6	0	1	Shaft code, 0 = one direction 1 = other direction
14	7	0	1	Arrow code, 0 = one direction 1 = other direction
15	0	1	0	Unused
16	0	0	6	Link Impedance field, in units of $\frac{1}{100}$ minutes
16	6	0	1	Link delete code 0 = keep link 1 = delete link from updated Flexible Data Record
16	7	4	1	Unused

SCRATCH MULTIPLE ASSIGNMENTS DATA SET

(Format Type Code: Z)

Header Record

Displacement Bytes	Length Bytes	Contents
0	4	Last node number
4	4	Number of one-way links
8	4	Number of Turning Movements saved

Links Record

Displacement Bytes	Length Bytes	Contents
0	4	Link Volume I
4	4	Link Volume I + 1
⋮	⋮	⋮
4K - 4	4	Link Volume K

The link records contain from 1 to 4000 directional link volumes each and the link volumes are written out in order of ascending link index.

Turn Volume Records

Displacement Bytes	Length Bytes	Contents
0	4	Turn Volume I
4	4	Turn Volume I + 1
⋮	⋮	⋮
4K - 4	4	Turn Volume K

The turn volume records contain from 1 to 4000 turn volumes and are written in order of ascending turn volume indexes.

The link volume records and turn volume records are repeated for other iterations of an Assign Self-Balancing run.

OUTPUT SELECTED LINKS

The OUTPUT SELECTED LINKS program must be run as a separate job (or as separate job steps). It uses the SELTRP data set built by ASSIGN SELECTED LINKS as input. The program performs two sorts and, thereby, produces two data sets. Both data sets have the same format. The format for these data sets is as follows:

SORTED SELECTED INTERCHANGES DATA SET

This is the data set which comes from the first sort in the OUTPUT SELECTED LINKS job as it is modified by the E 35 exit in the IBM sort using the E 35 assembly language subroutine. It is also the format of the data set which results from the second sort performed in the OUTPUT SELECTED LINKS job.

Header Records

Bytes Displacement	Length in Bytes	Contents
0	2	Zeros
2	2	2I + 1
4	8	Columns 8I + 1 to 8I + 7 of the Header Line

There are 12 header records (I = 0, 11); each header record has eight bytes of the header line except the last record which has four bytes of the header line.

Select Record

Bytes Displacement	Length in Bytes	Contents
0	2	Link Index of Selected Link*
2	2	Smallest node number of the selected link
4	2	Largest node number of the selected link
6	2	32767
8	2	Percent of Trip Volumes to print for this selected Link
10	2	Cut of Volume for Printing
12	2	Number of Trip Interchanges to print

*This is the index of the directional link from the smallest node of this selected link to the largest node of this selected link.

Sum Record

Displacement Bytes	Length in Bytes	Contents
0	2	Link Index of Selected Link
2	4	Zero
6	2	32766
8	2	-1
10	4	Sum of Trip interchange loaded through the Selected Link

Interchange Record

Displacement Bytes	Length in Bytes	Contents
0	2	Link Index of Selected Link
2	2	First Zone of the Interchange
4	2	Second Zone of the Interchange
6	4	Nondirectional link volume between the origin and destination zones
10	4	Directional link volume (direction specified by Trip Direction Code)
14	2	Trip Direction Code (see table on next page)

Trip Direction Code		First Zone to Second Zone Interchange		Second Zone to First Zone Interchange	
Decimal	Binary	Present	Direction of trip through link is small node number to large node number	Present	Direction of trip through link is small node number to large node number
1	0001	No	-	Yes	Yes
2	0010	Yes	Yes	No	-
3	0011	Yes	Yes	Yes	Yes
5	0101	No	-	Yes	No
7	0111	Yes	Yes	Yes	No
10	1010	Yes	No	No	-
11	1011	Yes	No	Yes	Yes
15	1111	Yes	No	Yes	No

O T H E R I N F O R M A T I O N

PRINTED OUTPUT FROM $ASSIGN AND $ASSIGN SELF-BALANCING

TURNING MOVEMENTS

PRINTED OUTPUT FROM $ASSIGN AND $ASSIGN SELF-BALANCING

Nineteen different types of tables may be produced during the execution of $ASSIGN SELF-BALANCING and sixteen different types during the execution of $ASSIGN. However, many of these tables are produced only under certain conditions. In addition, during the $ASSIGN SELF-BALANCING process, many of these tables are produced multiple times: some after each iteration, some after certain iterations, and some only after the last iteration. The following two tables, therefore, provide a summary of the output produced by these two programs under the various conditions:

SUMMARY OF OUTPUT FOR $ASSIGN SELF-BALANCING AND $ASSIGN

OUTPUT	$ASSIGN SELF-BALANCING						$ASSIGN
	First Iteration	Other Iterations	Last Iteration	Weighted Assignment Calculated from Iterations	Weighted Assignment made with Weighted Impedances (optional)	Other Output Produced After Last Assignment	
1. Selected Tables and Summaries*	X	X	X	X	X		X
2. Iteration Weighting-Multiple Regression Analysis	X	X	X				
3. Link Volumes	X			X	X		X
4. Iteration Weights Applied			X				
5. Corridor Intercept Tables						X	X
6. Route Profiles						X	X
7. List of Volumes and Impedances for Updated Links						X	

*see table titled "Tables and Summaries Produced with Each Assignment" on next page.

TABLES AND SUMMARIES PRODUCED WITH EACH ASSIGNMENT

Tables and Summaries	CONDITIONS UNDER WHICH TABLE OR SUMMARY IS PRODUCED				
	Functional Class Code for more than 5% of Links	Functional Class Code for less than 5% of Links	Non-zero Count Field for one or more of the links	Non-zero Capacity Field for one or more of the links	One or More Previous Assignments
1. Cross Classification of V/C Frequencies from Last Two Assignments					X
2. Cross Classification of Link Counts by V/C Ratio from Last Two Assignments			X		X
3. Jurisdiction Summary		X			
4. Jurisdictional/Functional Cross Classification of Assigned Volumes	X				
5. Jurisdictional/Functional Cross Classification of Counted Volumes	X		X		
6. Jurisidctional/Functional Cross Classification of Link Capacities	X			X	
7. Comparison of Assigned Volumes with Counted Volumes			X		
8. Comparison of Assigned Volumes with Link Capacities				X	
9. Comparison of Assigned Volumes (from last assignment) with Assigned Volumes (from assignment before last)					X

TURNING MOVEMENTS

Turning movements are directional volumes which are loaded through a specific triplet of nodes. Turning movements are logically associated with the intersection node. For a node connected to three other nodes the following equations can be written:

$$T_{1,1} + T_{1,2} + T_{1,3} = D_1$$

$$T_{2,1} + T_{2,2} + T_{2,3} = D_2$$

$$T_{3,1} + T_{3,2} + T_{3,3} = D_3$$

$$T_{1,1} + T_{2,1} + T_{3,1} = R_1$$

$$T_{1,2} + T_{2,2} + T_{3,2} = R_2$$

$$T_{1,3} + T_{2,3} + T_{3,3} = R_3$$

Where R_i = the directional link volume from the intersection node to the node of the i^{th} link.

Where D_j = the directional link volume from the node of the j^{th} link to the intersection node.

Where T_{ij} = the turning movement between the node in the i^{th} link and the node in the j^{th} link which are connected to the intersection node.

These equations can also be represented by a matrix with two vectors:

$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	D_1
$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	D_2
$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	D_3
R_1	R_2	R_3	

Because of the way in which trees are built and in which paths are represented in the Texas Small Network Package the turning movements on the diagonal of the matrix which are U-turns are all zero. Also the turning movements in some rows and columns will be zero because of the one-way links. To limit the possible number of cases with one-way links, the links which are connected to each node are connected in the following order: one-way links into the node, two-way links, one-way links out from node.

Putting in zeros for the diagonal elements for a case of three two-way links there are six equations with six unknowns:

0	$T_{1,2}$	$T_{1,3}$	D_1
$T_{2,1}$	0	$T_{2,3}$	D_2
$T_{3,1}$	$T_{3,2}$	0	D_3
R_1	R_2	R_3	

Each equation has two variables in it and one constant. Six equations with six unknowns can be solved if the equations are independent, however these equations are not. If any one of the six turning movements is known the other five can be calculated. The known turning movement will make two equations with only one unknown each which can be calculated and the turning movements which are calculated from these equations will allow other turning movements to be calculated.

The following method is used in calculating turning movements: (1) All locations in the turning movements matrix are set to -1 to represent unknowns; (2) The diagonal elements are set to zeros; (3) If there are any one-way links into the node then the corresponding row of the matrix is set to zero; (4) If there are any one-way links out the corresponding column of the matrix is set to zero; (5) Turning movements which have been saved are placed in the matrix; (6) The directional link volumes are found and become two vectors of constants; (7) The matrix is searched by rows and if a row has only one unknown it is calculated; (9) If there are any unknown turning movements left then steps 7 and 8 are repeated for up to N times where N is the number of nodes connected to the intersection node.

The process for calculating unknown turning movements can be used for a node connected to any number of nodes but the number of turning movements to save if all links are two-way goes up rapidly with the number of links to which a node is connected. Also the number of combinations of one-way links out, two-way links and one-way links in goes up rapidly with the number of links even when these links are sorted into the three link classes and arranged in the above order. For N, the number of nodes to which an intersection node is connected, where the links are all two-way $M = N^2 - 3N + 1$ for $N > 2$ where M is the number of turning movements to save. If U-turns were allowed then $M = N^2 - 2N + 1$.

In the Texas Small Network Package turn codes are set up for all combinations of two-way and one-way links for a node connected to either three or four nodes. Also there is a turn code for a node connected to

either five or six nodes. These turn codes are set up in either the Prepare Network, Assembly Network, or the Revise Network program and they are written on the Flexible Data Record data set. The turn code are described in a table. The turn codes for a node connected to five or six nodes cause enough turning movements to be saved to calculate the other turning movements when all of the links are two-way. This is also more than enough for the cases with one or more one-way links.

The turn codes and their meaning have been defined for the Texas Small Network Package since 1967 but a method for determining which turning movements to save and which to calculate will be outlined here. The easiest way to work with this problem is to represent the turning movements in a matrix form as was done earlier for the case of a node connected to three other nodes. It is convenient to let the row and column positions within the matrix represent the links which contain the node numbers instead of writing subscripts on the variables. Also a "s" will be written if the turning movement is saved, a "c" will be written if it is calculated and a zero will be written in the matrix position if the turning movement is known to be zero either because it is a U-turn or because of a one-way link. Also the two vectors which represent directional link volumes will not be written since these are always saved. To identify each case three one digit integers will be written over each matrix which are the number of two-way links, the number of one-way links in and the number of one-way links out which are connected to the intersection node. The following examples are all of the cases for a node connected to four other nodes for which one or more turning movements must be saved:

0 2 2

0	0	0	0
0	0	0	0
c	c	0	0
s	c	0	0

1 1 2

0	0	0	0
0	0	0	0
c	c	0	0
s	c	c	0

1 2 1

0	0	0	0
c	0	0	0
c	c	0	0
s	c	0	0

2 0 2

0	0	0	0
0	0	0	0
c	c	0	c
s	c	c	0

2 1 1

0	0	0	0
c	0	c	0
c	s	0	0
s	c	c	0

2 2 0

0	c	0	0
c	0	0	0
c	c	0	0
s	c	0	0

3 1 0

0	c	c	0
c	0	c	0
s	c	0	0
s	s	c	0

3 0 1

0	0	0	0
c	0	c	c
s	c	0	c
s	s	c	0

4 0 0

0	c	c	c
c	0	s	c
s	s	0	c
s	s	c	0

TURN CODES

Turn Code	Total Number of Links	T	I	0	Number of Turning Movements to Save	Turn Movements to Save*
1	3	0	0	3	0	
2	3	0	1	2	0	
3	3	0	2	1	0	
4	3	0	3	0	0	
5	3	1	0	2	0	
6	3	1	1	1	0	
7	3	1	2	0	0	
8	3	2	0	1	0	
9	3	2	1	0	0	
10	3	3	0	0	1	3-1
11	4	0	0	4	0	
12	4	0	1	3	0	
13	4	0	2	2	1	4-1
14	4	0	3	1	0	
15	4	0	4	0	0	
16	4	1	0	3	0	
17	4	1	1	2	1	4-1
18	4	1	2	1	1	4-1
19	4	1	3	0	0	
20	4	2	0	2	1	4-1
21	4	2	1	1	2	4-1,3-2
22	4	2	2	0	1	4-1
23	4	3	0	1	3	4-1,4-2,3-1
24	4	3	1	0	3	4-1,4-2,3-1
25	4	4	0	0	5	4-1,4-2,3-1,3-2,2-3
26	5	-	-	-	11	5-1,5-2,5-3,4-1,4-2,4-3, 3-1,3-2,3-4,2-3,2-4
27	6	-	-	-	19	6-1,6-2,6-3,6-4,5-1,5-2, 5-3,5-4,5-1,4-2,4-3,4-5, 3-1,3-2,3-4,3-5,2-3,2-5, 1-4
28	-	-	-	-	0	**

T = number of two-way links connected to the intersection node
I = number of one-way links connected into the intersection node
0 = number of one-way links connected out from the intersection node

*The turning movements to save are listed by the subscript pair in the form i-j which indicate the position of the turning movement in the turning movement matrix.
**Save no turning movements for this node (or centroid) and print no turning movements.

RECENT CHANGES AND MODIFICATIONS