PROGRAM DOCUMENTATION MANUAL

for

THE TEXAS LARGE NETWORK PACKAGE

by

J. D. Benson
Assistant Research Planner

Charles E. Bell
Data Processing Programmer

and

Vergil G. Stover
Study Supervisor

Research Report 167-4

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

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

ABSTRACT

The Texas Large Network Package is a collection of computer programs designed to assign traffic to large transportation networks. The purpose of this manual is to provide data processing personnel with a link between the Operating Manual for the Texas Large Network Package (Research Report 119-2) 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 Large 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 Large Network Package is a collection of computer programs designed to assign traffic to large 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 Large Network Package in addition to the usual programs regarding the assignment of traffic to minimum time paths. 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 the previous assignments.

The Texas Large Network Package is comprised of seventy-seven control sections. The control sections perform the sixteen 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 Large Network Package has been operational on the IBM 360 computer installation of the Texas Highway Department since January, 1969. 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 Large 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 Large 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 Large 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 Large 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 Large Network Package. For example, this section 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 Large Network Package is comprised of seventy-seven 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 eighteen logical divisions as shown in Figure 2 (note that Logical Division 18 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, seven of the logical divisions contain only one control section and two 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 Section	Revision Date	Program Control Section	Revision Date
ALCP	11-10-71	OUTWLT	7-26-71
BLOCK DATA	11-10-71	PARAM	10- 1-70
CLOAD	9-20-71	PATHCL	10- 1-70
CLOSE	9-20-71	PRPBLD	7-26-71
CMPVH	11-10-71	PRPCTV	3-31-71
COPYFT	8-30-71	PRPNET	3-31-71
CRD	3-31-71	PTLNK	10- 1-70
CRDINT	3-31-71	REGRES	3-31-71
E35	*	RTPFL	3-31-71
FMTLNE	10- 1-70	RTPLT	7-30-71
FRATAR	3-31-71	SC	3-31-71
FWTO	10- 1-70	SELECT	8-30-71
GETDAT	10- 1-70	SUBFND	10- 1-70
GETRN	7-26-71	SUMEND	3-31-71
GETRNS	10- 1-70	SUMRY	3-31-71
GETVOL	7-26-71	SVLOAD	7-26-71
GTLD	11-10-71	TIME	10- 1-70
GTVL	7-26-71	TREBLD	10- 1-70
ITOA	10- 1-70	TRN	7-26-71
INITL1	10- 1-70	TRNMV	10- 1-70
LNKLST	7-26-71	TRPCKM	10- 1-70
LOAD	9-20-71	TURNM	10- 1-70
LOPS	8-30-71	UNPKX	10- 1-70
MAIN	7-26-71	UPDTNT	3-31-71
MAIN (for Output Selected Links)	*	VREC	8-30-70
		VSORT	10- 1-70
MERG	10- 1-70	WGT	7-26-71
MOOR	10- 1-70	WGTA	7-26-71
MRGREC	7-30-71	WGTLD	7-26-71
NEWNET	8-30-71	WRT	8-30-71
OPENFT	8-30-71	WSL	10- 1-70
OUTLLT	8-30-71	WTSGLN	7-26-71
OUTNET	8-30-71	Overlay structure	7-26-71
OUTRIP	3-31-71		
OUTTRE	10- 1-70		

Labeled Common Control Sections: ALLIGN, ARRAYS, CD, CAPRES, DELETE, FILES, 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: Large Network Package Overlay Structure

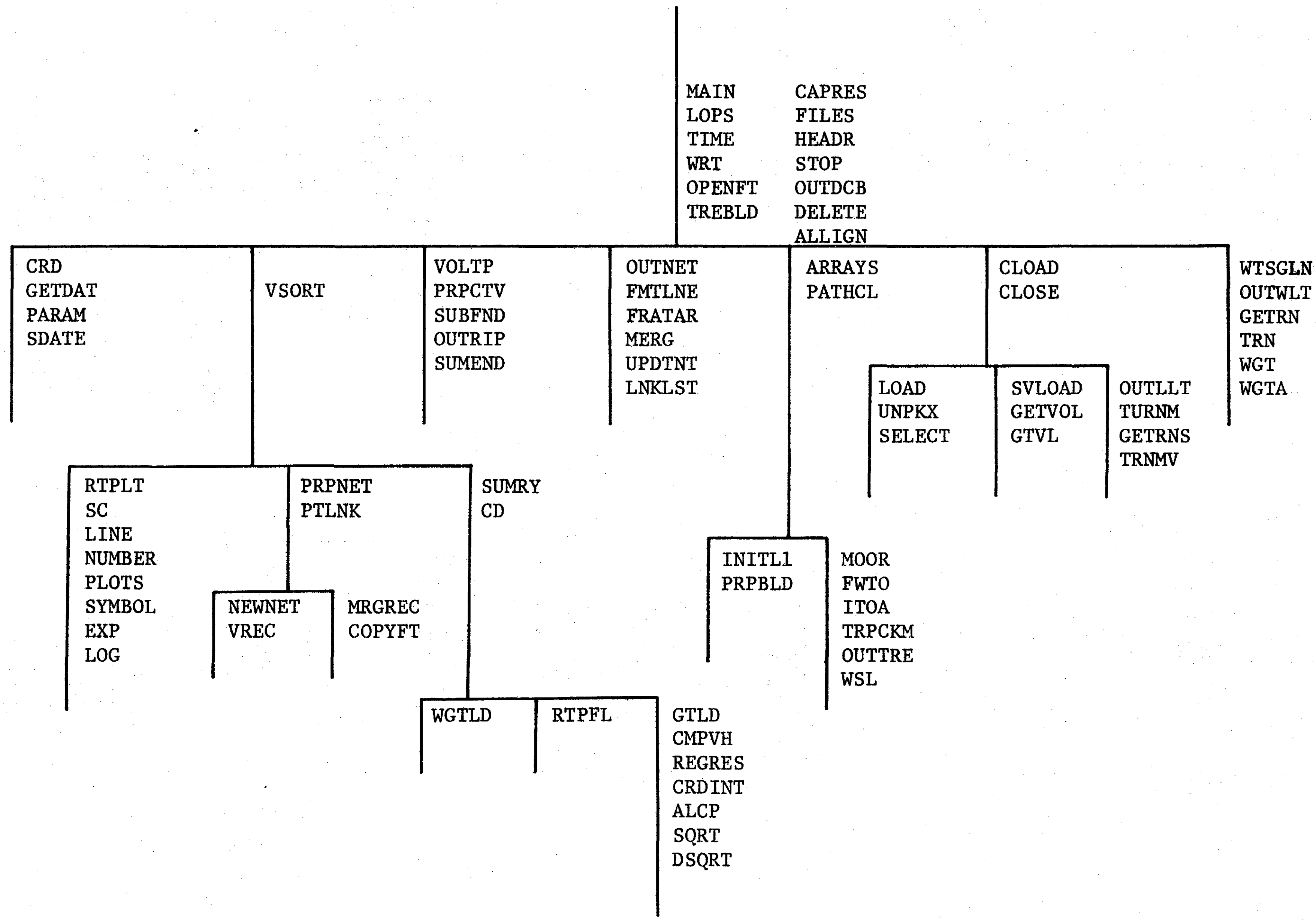
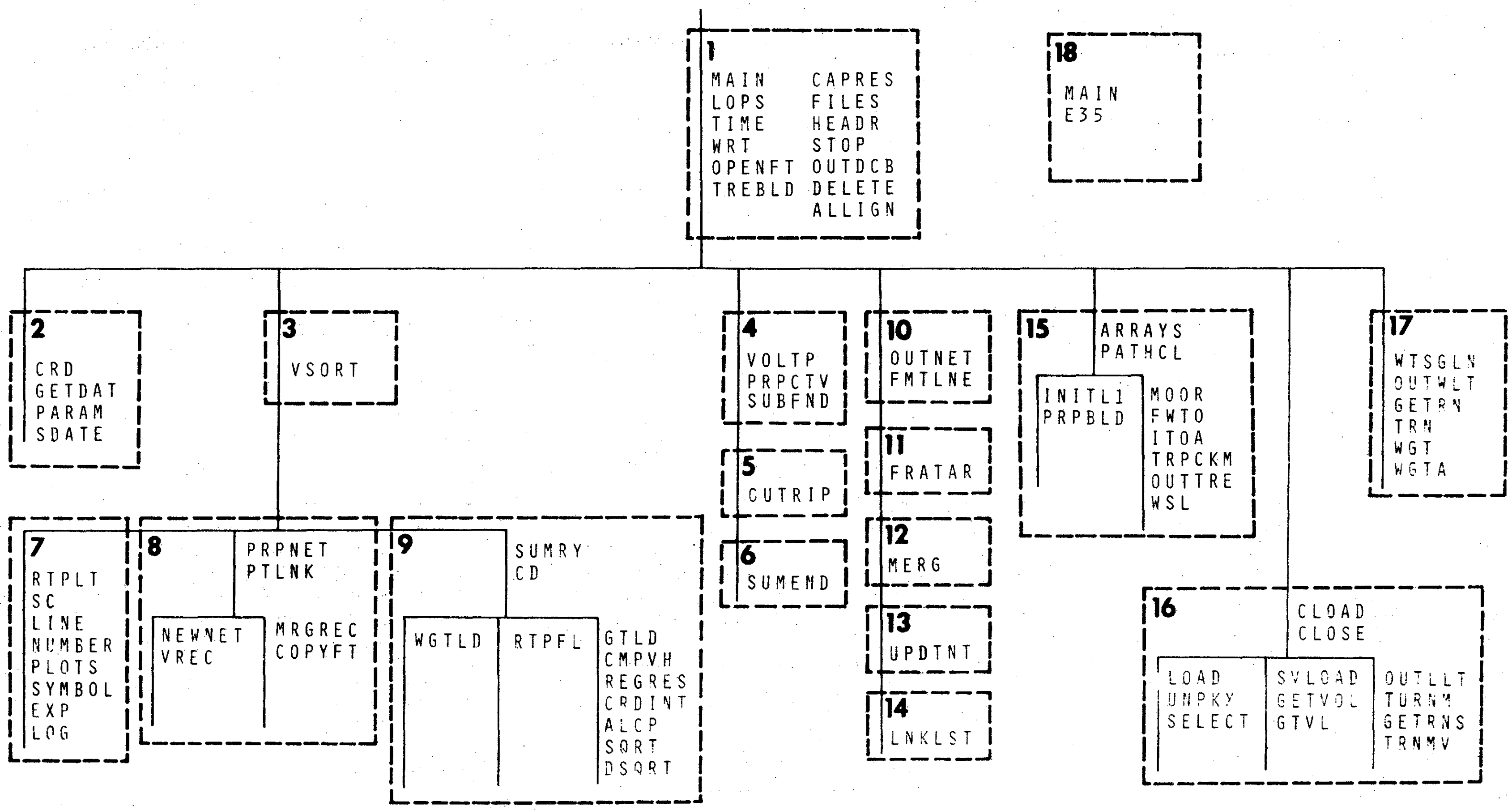


FIGURE 2: LARGE NETWORK PACKAGE LOGICAL DIVISIONS



options described in the operating manual. On the other hand, it may be noted that logical division 16 contains twelve control sections. It is within this division that the network is loaded, and the loaded network is printed.

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. The assembly language programs with multiple control sections are also listed in Table 2.

TABLE 2: PROGRAMS WITH MULTIPLE ENTRY POINTS OR MULTIPLE CONTROL SECTIONS

PROGRAM	OTHER ENTRY POINTS OR CONTROL SECTIONS
CLOAD	LDSEL
GETVOL*	WGT**, WGTA**, GTVL**
ITOA*	ITOAB
LOAD*	CLOSE**, OPEN, WRITE, TRDCB
LOPS*	LGLS, LGRS, LANA, LORA, LEX, LANAD, LANAL, NBYTE, LBYTE, LANAH, SBYTE, IB16, NULLT
MOOR*	MOORE, MINUS
PRPNET	ASMNET, REVNET
PTLNK*	GTLNK
WRT*	OPENFT**, CLOSFT

*Assembly language routine
**Control Section

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

INTRODUCTION

The seventy-seven control sections comprising the Texas Large Network Package have been grouped for the convenience of discussion, into eighteen 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

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 8 → Logical Division 3
Logical Division 8 → Logical Division 1

$REVISE NETWORK

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 8 → Logical Division 3

$OUTPUT NETWORK

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 10

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
$PREPARE NETWORK	X	X	X					X										
$ASSEMBLE NETWORK	X	X	X					X										
$REVISE NETWORK	X	X	X					X										
$OUTPUT NETWORK	X	X								X								
$DELETE ASSIGNMENTS	X	X											X					
$PREPARE TRIP VOLUMES	X	X		X														
$OUTPUT TRIP VOLUMES	X	X			X													
$BUILD TREES	X	X													X			
$LOAD NETWORK	X	X	X						X							X		
$LOAD SELECTED LINKS	X	X	X						X							X		
$ASSIGN SELF-BALANCING	X	X	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						

$DELETE ASSIGNMENTS

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 13 → Logical Division 1

$PREPARE TRIP VOLUMES

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 4

$OUTPUT TRIP VOLUMES

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 5

$BUILD TREES

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 15

$LOAD NETWORK
$LOAD SELECTED LINKS

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 16 → Logical Division 1
Logical Division 1 → Logical Division 9 → Logical Division 3

$ASSIGN SELF-BALANCING

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 15
Logical Division 1 → Logical Division 16 → Logical Division 1
Logical Division 1 → Logical Division 9 → Logical Division 3
Logical Division 1 → Logical Division 17 → Logical Division 1
Logical Division 1 → Logical Division 14

$OUTPUT SELECTED LINKS

Logical Division 18

$PLOT ROUTE PROFILES

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 7 → Logical Division 3

$FRATAR FORECAST

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 11

$SUM TRIP ENDS

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 6

$MERGE

Logical Division 1 → Logical Division 2
Logical Division 1 → Logical Division 12

DESCRIPTIONS OF LOGICAL DIVISIONS

The description of each of the logical divisions in the Texas Large 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. Also, subroutine WRT is called from other Logical Divisions to write unformatted data sets.

Control Sections: MAIN, TIME, CAPRES, FILES, HEADR, LOPS, ALLIGN, STOP, OUTDCB, DELETE, OPENFT, CLOSFT, TREBLD, WRT

Sequence of Subroutines Called

MAIN →
- Logical Division 2 (GETDAT)
- Logical Division 2 (CRD)
- TIME
- (Other Logical Division needed to perform the functions specified by control cards)

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 containing the data control blocks built by subroutine OPENFT.

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 13 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.

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 control section. The DCB is indexed by either a 1 or a 2 which is the argument in the call to CLOSFT.

WRT: This subrouting 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.

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

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 $HEADR 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

Logical Division 1 → GETDAT

Logical Division 1 → CRD → PARAM

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 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 division contains the subroutine VSORT which performs an in-core sort. It is used by Logical Divisions 7, 8, and 9.

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 (7, 8, or 9) ⟶ 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 4

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 5

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 6

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 7

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 LOAD NETWORK, LOAD 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

RTPLT → Logical Division 3 (VSORT)
RTPLT → SC
RTPLT → Calcomp Subroutines (AXIS, LINE, NUMBER, PLOTS, SYMBOL)

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 8

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 6667 link data cards (recall that each link data card produces 2 link records). This program will handle up to 3 groups of approximately 6667 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 ≤ node number ≤ 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 10.23 "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 6667 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 6667 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 and 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 9

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.

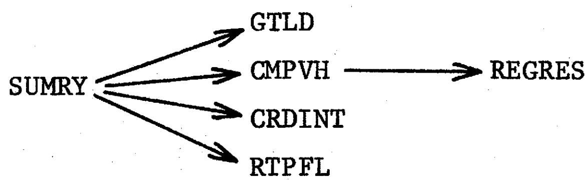
Output: The tables listed in the general section above and Unit ROUTE.

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

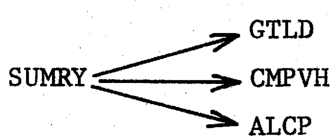
<u>Sequence of Subroutines Called</u>

$LOAD NETWORK and

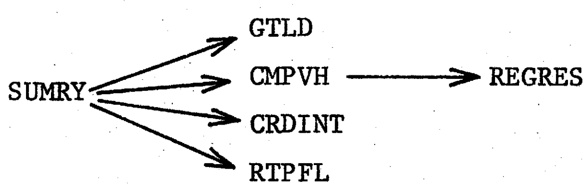
$LOAD SELECTED LINKS



$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)

Logical Division 1 ——→ WGTLD

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 10

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. This 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 11

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 12

General

This division is called by the program MAIN (in Logical Divison 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 13

General

This division is called by Logical Division 1 and uses the WRT subroutine in Logical Division 1. 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 1)

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 Division1) 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 14

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 15

General

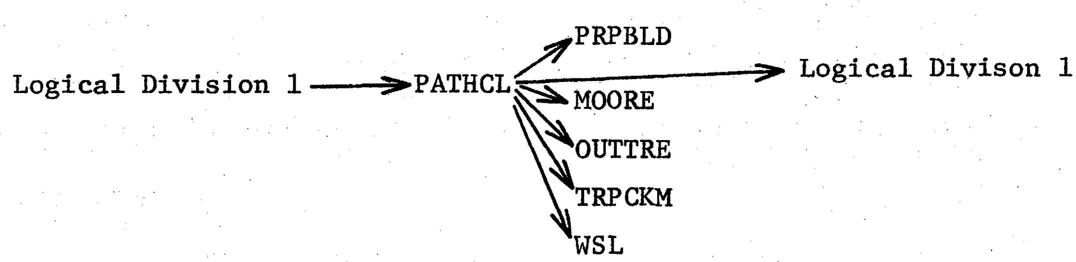
This section performs the $BUILD TREES function. It is also used to perform part of the $ASSIGN SELF-BALANCING function.

Input: *TURN card, *TREE cards, unit NETWORK and possibly unit 49 (the old Paths data set) if the COPY option is on the *TURN card.

Output: Printed trees specified, Paths data set, and unit SEPARAT, the separation matrix.

Control Sections Used: PATHCL, ARRAYS, INITL1, PRPBLD, MOOR, FWTO, ITOA, TRPCKM, OUTTRE, WSL, and Logical Division 1.

Sequence of Programs Called



Descriptions of Individual Control Sections

PATHCL: This is the control subroutine for this division. It defines arrays used by subroutines called from this division. It reads the network into core from unit NETWORK and changes it to the form used by the tree builder subroutine. It controls the building of trees, the printing of trees, the packing of the paths, and writes the Paths data set and the separation matrix data sets.

ARRAYS: This is a labled common which contains most of the storage used by subroutine PATHCL.

INITL1: This subroutine checks to see that all turn type codes read are valid and also checks the number of the nodes in the network. This subroutine is not used.

PRPBLD: This subroutine reads the *TURN card and the *TREE cards which specify the turn penalty and the trees to be built and printed. The COPY parameter is also specified on the *TURN card if it is used.

MOOR: This control section builds one minimum path tree each time it is called. Its entry point is MOORE.

FWTO: This subroutine prints one line on the computer operators console each time it is called. It is called at approximately five-minute intervals during the tree building process and identifies the trees built during that period.

ITOA: This subroutine converts a binary interger to EBCDIC format for printing.

TRPCKM: This subroutine packs an array of path indices from 16 bit integers to ten 3 bit integers per word. The control sections also contains the entry point TEST which checks to see that an array of packed path indices contains no indices of 6.

OUTTRE: This subroutine prints one tree each time it is called.

WSL: This subroutine writes a record of separations for one tree for the centroids only.

LOGICAL DIVISION 16

General

This section performs the loading of trees and printing of the loaded network function for the following user program options:

- $LOAD NETWORK
- $LOAD SELECTED LINKS
- $ASSIGN SELF-BALANCING

Input: Flexible Record data set (unit NETWORK), Paths data set (Unit 50), and parameter cards for LOAD SELECTED LINKS if it is run. The parameter cards for LOAD SELECTED LINKS are *ALL, *LINKS, *NONE, *SEL, and *END.

Output: New Flexible Record data set (unit NEWNET), printed loaded network, and Selected Interchanges data set (DD name SELTRP) if LOAD SELECTED LINKS is run.

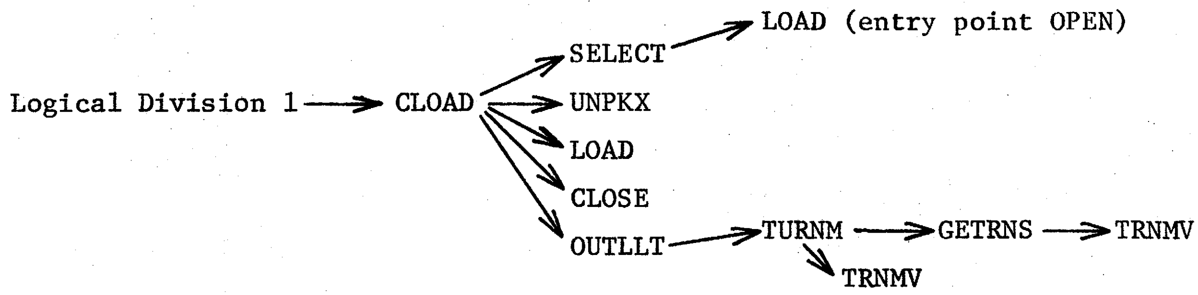
Control Sections Used: CLOAD, CLOSE, LOAD, UNPKX, SELECT, SVLOAD, GETVOL, GTVL, OUTLLT, TURNM, GETRNS, and TRNMV.

Sequence of Programs Called

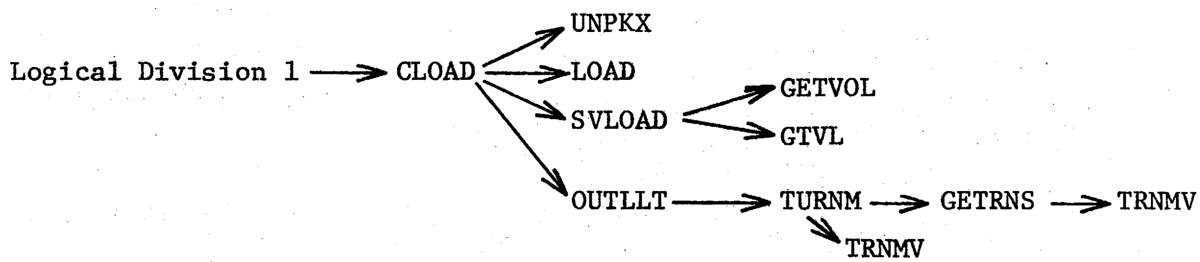
$LOAD NETWORK

```
                                    ↗ UNPKX
Logical Division 1 ──> CLOAD ──> LOAD
                                    ↘ OUTLLT ──> TURNM ──> GETRNS
                                                    ↘ TRNMV    ↘ TRNMV
```

$LOAD SELECTED LINKS



$ASSIGN SELF-BALANCING



Descriptions of Individual Control Sections

CLOAD: This subroutine controls the execution of this logical division. It reads the network from unit NETWORK and modifies this to the format needed for the LOAD subroutine. It initializes the directional link volume array and the turn volume array to zero. It calls subroutine SELECT if it is a LOAD SELECTED LINKS run. It reads the trip matrix and the paths data sets. These are assumed to be in sort on the origin zones. It calls subroutine LOAD to load trips in the network if there is both a tree record and one or more trip records for an origin zone. After the network is loaded CLOAD calls subroutine SVLOAD to save the loaded network on unit 3 if an ASSIGN SELF-BALANCING run is in iteration 1 thru 5. CLOAD then calls subroutine OUTLLT to print the loaded network.

CLOSE: This subroutine closes data set SELTRP.

LOAD: This subroutine loads a trip record by adding each trip interchange volume to all of the directional link volumes in the path connected between the origin and destination zones of the trip interchange. Some turn volumes are also summed in this process. This subroutine also writes a record on unit SELTRP for each selected link crossed in loading each trip interchange volume.

UNPKX: This subroutine unpacks the path indices and places them in half words.

SELECT: This subroutine reads the parameter cards of LOAD SELECTED LINKS. For each *SEL card it writes one record on unit SELTRP and it marks both of the one-way directional links as selected. This subroutine also reads one of the following parameter cards: *ALL, *LINKS or *NONE. If the *LINKS card is read, this subroutine sets all turn codes in core to 28. If the *NONE card is read, a logical variable is set to specify that the loaded network will not be printed.

SVLOAD: This subroutine writes the directional link volumes twice on unit 3 and the turn volumes also on unit 3. It also sets up segment sizes for the network which are small enough so that a segment of the loaded network may be summed in core using the weights calculated for ASSIGN SELF-BALANCING. The first set of link volumes written for each segment are in the same order as the links in the network. Each of the second set of the link volumes is in the reverse direction of the links in the network.

GETVOL: This subroutine gets a group of link volumes between two integer indexes and places them in a full word array.

GTVL: This subroutine gets a group of link volumes which are the reverse direction of those obtained for GETVOL. It also places these link volumes in a full word array.

OUTLLT: This subroutine controls the printing of the loaded network. It prints page headings, calls subroutine TURNM to get the link volumes and turn volumes for a node and formats the directional link volume, non-directional link volumes, and turn volumes.

TURNM: This subroutine gets the directional volumes, nondirectional volumes, and turn volumes saved. It also calculates the other turn volumes and marks which turn volumes should not be printed because of one-way links.

GETRNS: This subroutine gets the turn volumes which were saved and places them in the turn volume matrix.

TRNMV: This subroutine adds two indices together and gets the assigned volumes indexed by the sum from a half word array. If the half word is negative, it is a flag and an index and this index is used to get the actual volume from a full word array.

LOGICAL DIVISION 17

General

This section is used when ASSIGN SELF-BALANCING is run to calculate a weighted assignment.

Input: Unit NETWORK and unit 3.

Output: Unit NEWNET and the printed weighted loaded network.

Control Sections Used: WTSGLN, OUTWLT, GETRN, WGT, and WGTA.

Sequence of Programs Called:

Logical Division 1 ⟶ WTSGLN ⟶ OUTWLT ⟶ TRN ⟶ GETRN
WTSGLN ↘ WGT
WTSGLN ↘ WGTA

Descriptions of Individual Control Sections

WTSGLN: This subroutine reads unit 3 and using the weights for each iteration sums up the weighted directional link volume, reverse directional link volumes, and turn volumes for one segment in core. It then rewinds unit 3 and calls subroutine OUTWLT to print this segment of the loaded network. It repeats the above steps for other segments. The line counter used by subroutine OUTWLT to print page headings is only initialized for the first call to OUTWLT.

OUTWLT: This subroutine prints the loaded network for one segment of the loaded network. It calles subroutine TRN to calculate the turn volumes for one node. It reads the node numbers and the node names from unit NETWORK and it writes the updated Flexible Record with the weighted assignment volumes added on unit NEWNET.

TRN: This subroutine gets the weighted directional volumes the weighted nondirectional volumes, and the weighted turn volumes saved. It also calculates the other weighted turn volumes and marks which turn volumes should not be printed because of one-way links.

GETRN: This subroutine gets the weighted turn volumes which were saved and places them in the turn volume matrix.

WGT: This subroutine multiplies a group of volumes by an integer percent and places the results in another array.

WGTA: This subroutine multiplies a group of volumes by an integer percent and adds the results into another array.

LOGICAL DIVISION 18

General

This division prints the selected links output (i.e., the output from $LOAD 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 E35, and a Fortran program to list the selected links and the trip interchanges loaded through them (i.e., MAIN).

Sequence of Program Execution:

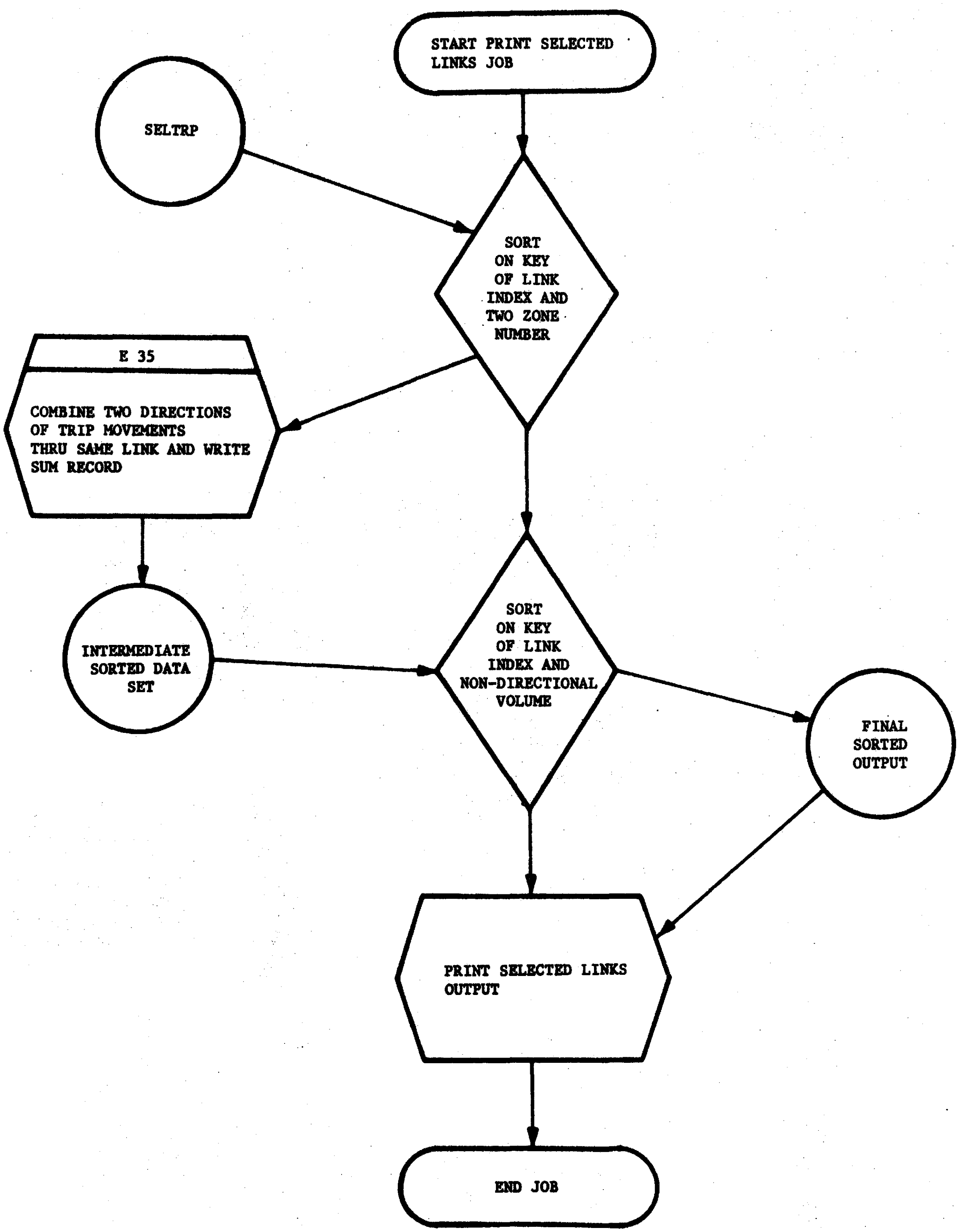
JOB STEP 1	JOB STEP 2	JOB STEP 3
IBM SORT	IBM SORT	MAIN (List Selected Links)
E 35		

Summary of Individual Programs

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

E35: 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 nondirectional zone pair trips through the selected link. The total nondirectional 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 nondirectional 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 nondirectional zone pair volume) and prints the interchanges assigned to each selected link (in descending order of magnitude of the nondirectional 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	ABEND	ALCP	ASMNET	CLOAD	CMPVH	CRD	CRDINT	FRATAR	GETRNS	GTLD	INITL1	LDSEL	LNKLST	MAIN	MERG	MRGREC	NEWNET	OUTLLT	OUTNET	OUTRIP	OUTTRE	OUTWLT	PATHCL	PRPBLD	PRPCTV	PRPNET	READVL	REGRES	REVNET	RTPFL	RTPLT	SC	SELECT	SUMEND	SUMRY	SVLOAD	TREBLD	TRN	TRNMV	TURNM	UPDTNT	VREC	WGTLD	WSL	WTSGLN
ABEND									X																													X		X					
ALCP																																			X										
ALOG10																															X	X													
AMAX1																															X												X		
AMIN1																		X																							X				
ASMNET														X																															
CLOAD														X																															
CLOSE				X								X																																	
CLOSFT										X								X				X																			X	X			
CMPVH																																			X										
COPYFT																X																													
CRD														X																															
CRDINT																																			X										
DSQRT		X																										X																	
FMTLNE																			X																										
FRATAR														X																															
FRXPT																		X																							X				
FRXPR																																X													
FWTO																							X																						
GETDAT														X																															
GETRN																																						X							
GETRNS																																								X					
GETVOL																																				X									
GTLD																																			X										
GTLNK																X																										X			
GTVL																																				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
IB16																		X			X		X										X			X			X	X					
ITOAB																							X																						
LANA				X						X	X	X				X	X	X				X	X										X			X			X	X	X				
LANAH																		X															X							X					
LANAL				X						X		X	X		X	X		X		X	X	X	X											X						X					
LBYTE				X								X																								X				X					
LDSEL														X																															
LGLS										X					X	X		X				X	X		X								X								X	X			
LGRS				X						X	X	X	X		X	X	X	X		X		X	X											X											
LINE																															X														
LNKLST														X																															
LOAD				X								X																																	
LORA				X						X		X			X	X		X				X	X										X								X	X			
MAXO										X					X	X															X										X	X			
MERG														X																															
MINO							X			X						X			X											X	X					X					X	X			X
MOORE																							X																						
MRGREC																													X																
NBYTE																																	X												
NEWNET			X																							X			X																
NUMBER																															X														
OPEN																																	X												

TABLE 4: (continued)

CALLING PROGRAM

PROGRAMS CALLED	ABEND	ALCP	ASMNET	CLOAD	CMPVH	CRD	CRDINT	FRATAR	GETRNS	GTLD	INITL1	LDSEL	LNKLST	MAIN	MERG
OPENFT										X					
OUTLLT				X								X			
OUTNET														X	
OUTRIP														X	
OUTTRE															
OUTWLT															
PARAM						X									
PATHCL														X	
PLOT															
PLOTS															
PRPBLD															
PRPCTV														X	
PRPNET														X	
PTLNK															
REGRES					X										
REVNET														X	
RTPFL															
RTPLT														X	
SBYTE				X								X			
SC															
SELECT				X								X			
SQRT		X													
SUBFND															
SUMEND														X	
SUMRY														X	
SVLOAD				X								X			
SYMBOL															
TEST															
TIME				X								X		X	
TRN															
TRNMV									X						
TRPCKM															
TURNM															
UNPKX				X								X			
UPDTNT														X	
VREC			X												
VSORT							X								
WGT															
WGTA															
WGTLD														X	
WRITE															
WRT										X					
WSL															
WTSGLN														X	

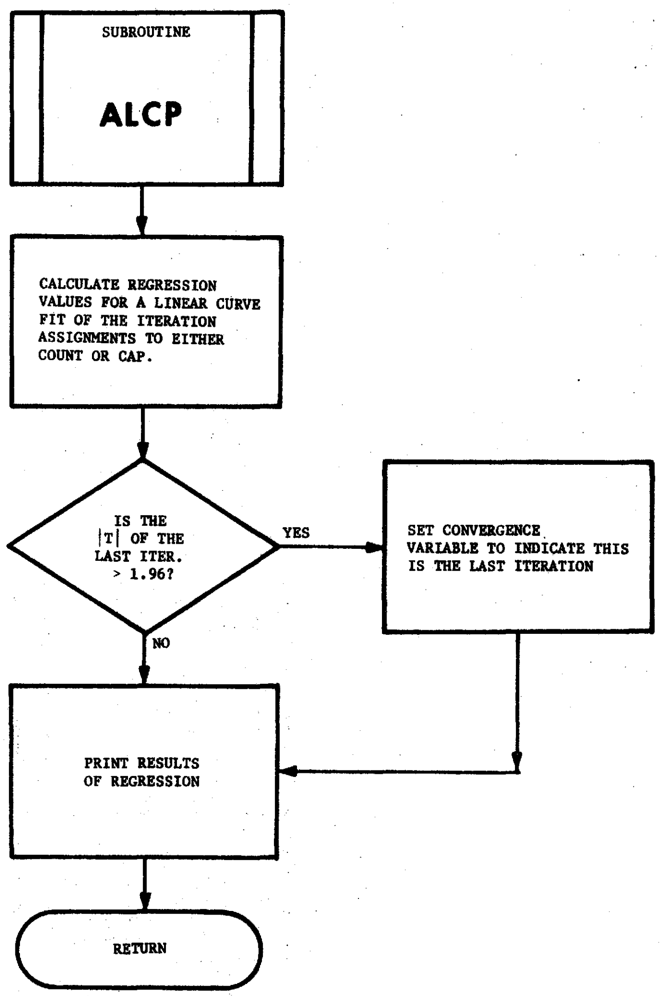
PROGRAMS CALLED	MRGREC	NEWNET	OUTLLT	OUTNET	OUTRIP	OUTTRE	OUTWLT	PATHCL	PRPBLD	PRPCTV	PRPNET	READVL	REGRES	REVNET	RTPFL
OPENFT			X				X								
OUTLLT															
OUTNET															
OUTRIP															
OUTTRE								X							
OUTWLT															
PARAM															
PATHCL															
PLOT															
PLOTS															
PRPBLD								X							
PRPCTV															
PRPNET															
PTLNK		X													
REGRES															
REVNET															
RTPFL															
RTPLT															
SBYTE															
SC															
SELECT															
SQRT															
SUBFND										X					
SUMEND															
SUMRY															
SVLOAD															
SYMBOL															
TEST								X							
TIME		X						X							
TRN							X								
TRNMV															
TRPCKM								X							
TURNM			X												
UNPKX															
UPDTNT															
VREC											X				
VSORT		X													X
WGT															
WGTA															
WGTLD															
WRITE															
WRT			X				X								
WSL								X							
WTSGLN															

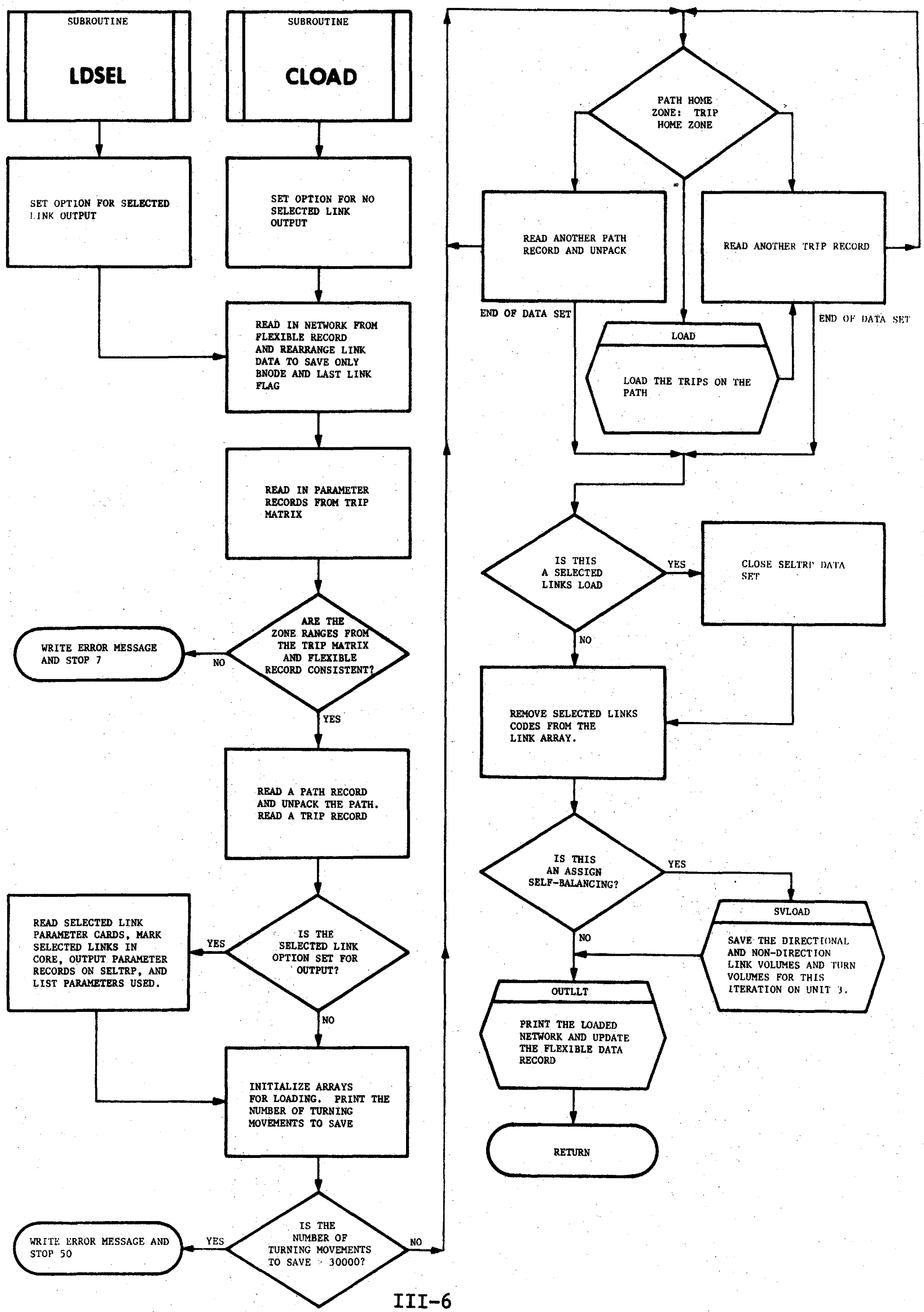
PROGRAMS CALLED	RTPLT	SC	SELECT	SUMEND	SUMRY	SVLOAD	TREBLD	TRN	TRNMV	TURNM	UPDTNT	VREC	WGTLD	WSL	WTSGLN
OPENFT											X	X			
OUTLLT															
OUTNET															
OUTRIP															
OUTTRE															
OUTWLT															X
PARAM															
PATHCL															
PLOT	X														
PLOTS	X														
PRPBLD															
PRPCTV															
PRPNET															
PTLNK															
REGRES															
REVNET															
RTPFL					X										
RTPLT															
SBYTE			X												
SC	X														
SELECT															
SQRT															
SUBFND															
SUMEND															
SUMRY															
SVLOAD															
SYMBOL	X														
TEST															
TIME															
TRN															
TRNMV										X					
TRPCKM															
TURNM															
UNPKX															
UPDTNT															
VREC															
VSORT	X														
WGT															X
WGTA															X
WGTLD															
WRITE			X												
WRT											X	X			
WSL															
WTSGLN															

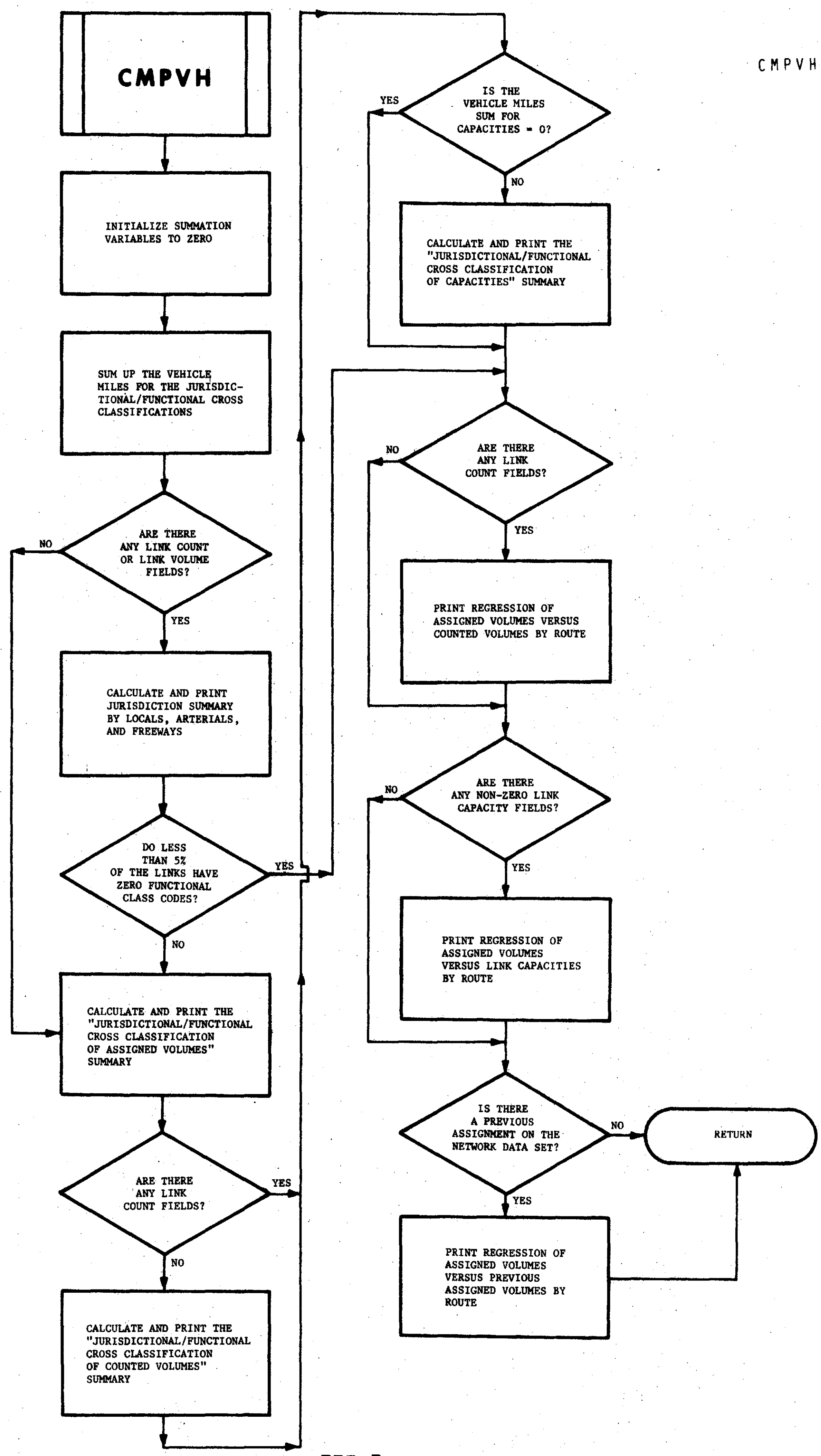
FLOWCHARTS

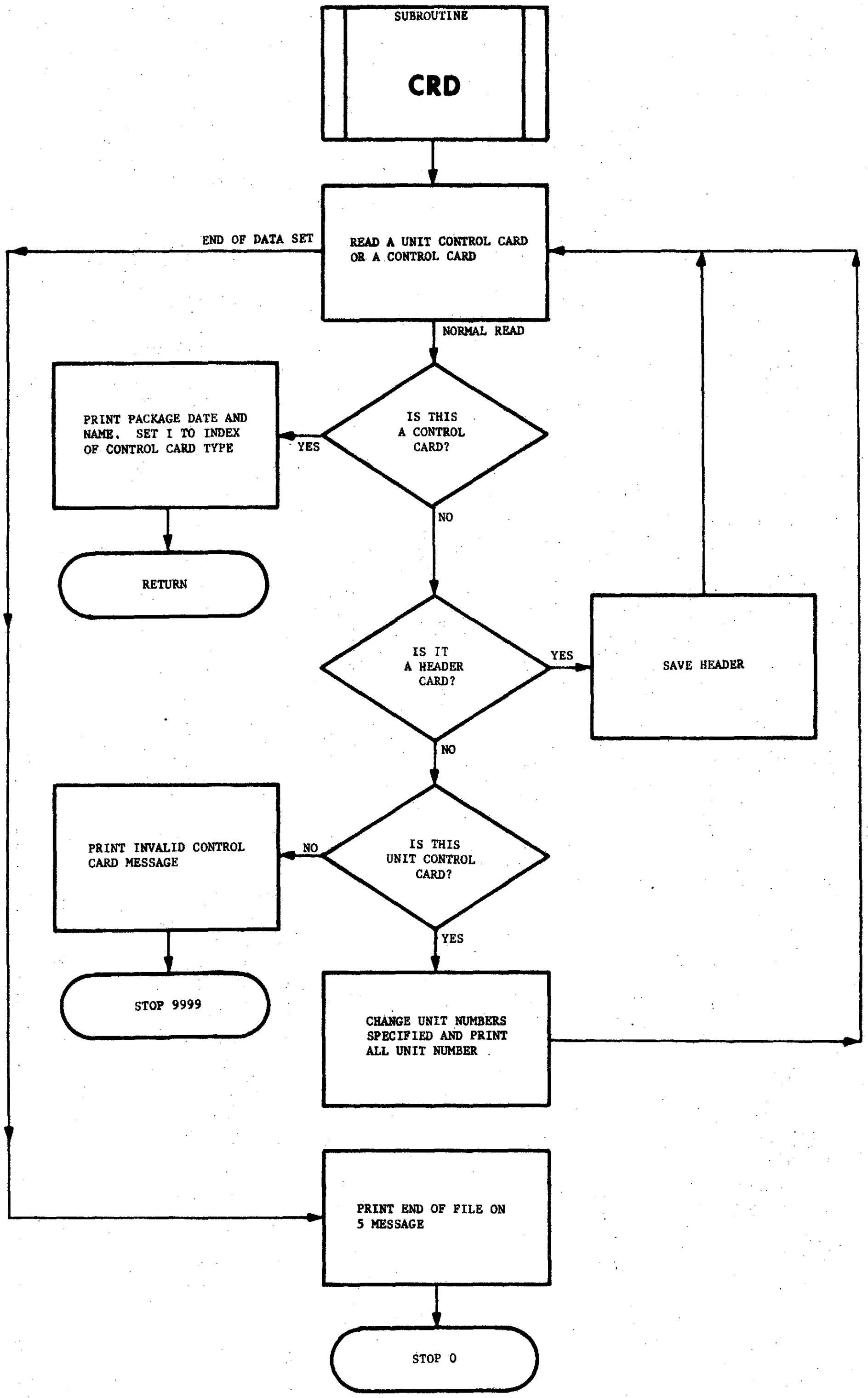
The following are the flowcharts associated with the significant subroutines in the Large Network Package. For convenience, these flowcharts are in alphabetical order.

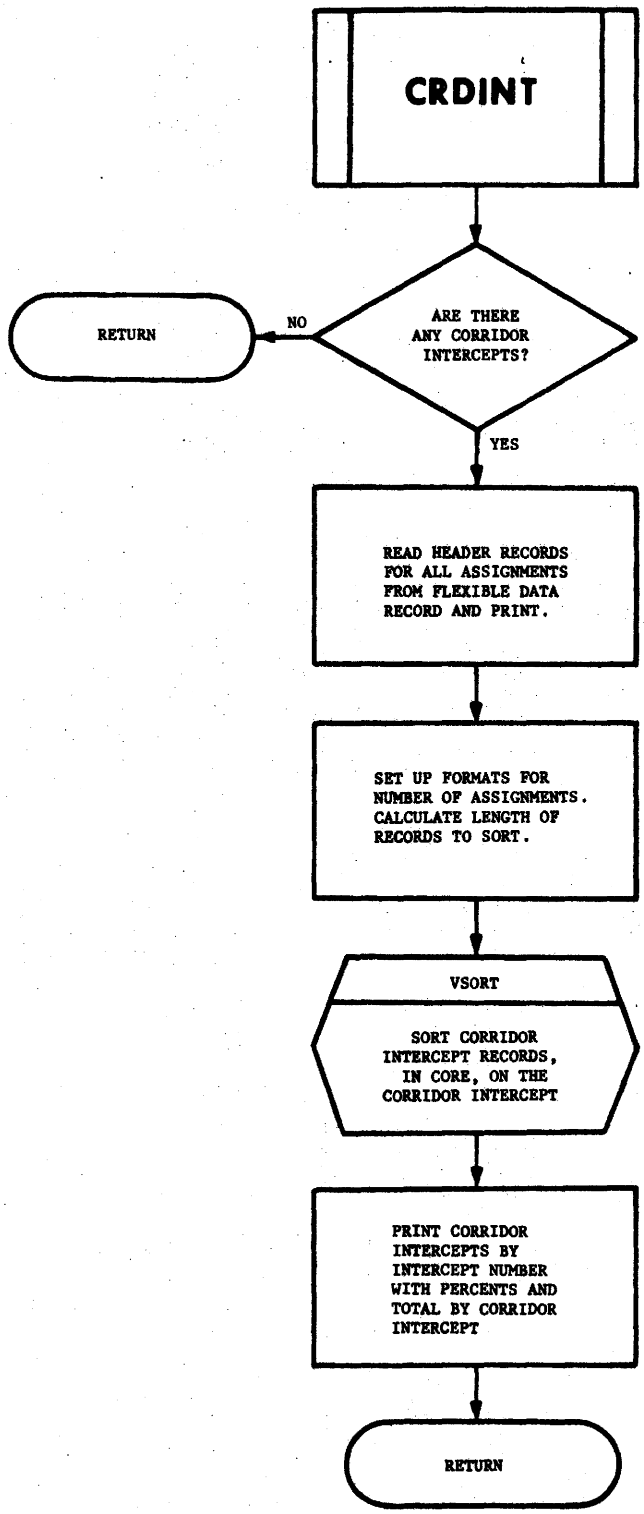
The objective of the flowcharts is to provide the programmer with an overview of the operation of each individual program. The level of detail contained in each flowchart is felt to be minimal for such an understanding. It should also be noted that these flowcharts are intended to be used in conjunction 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
SET CONVERGENCE VARIABLE TO INDICATE THIS IS THE LAST ITERATION
NO
PRINT RESULTS OF REGRESSION
RETURN

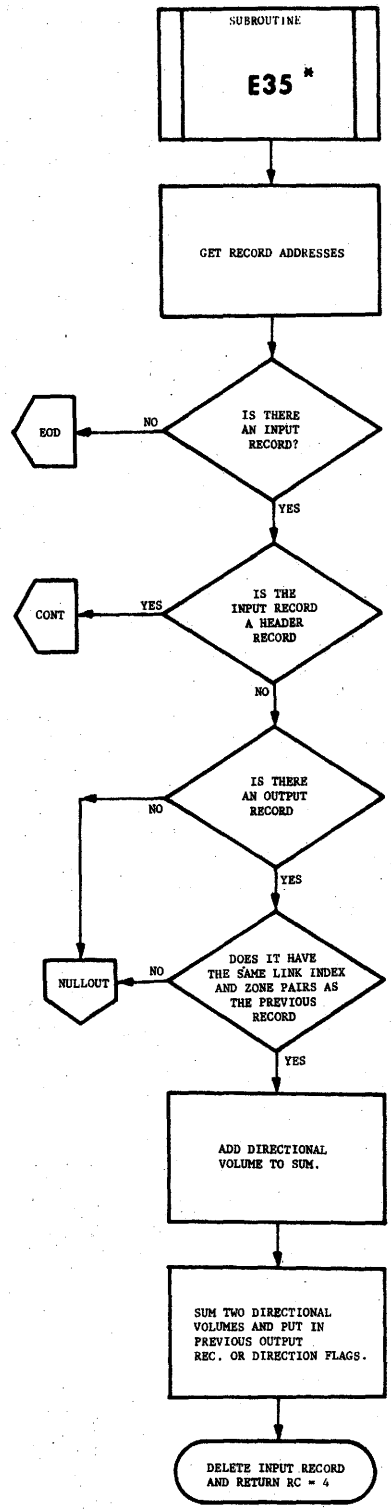

SUBROUTINE
LDSEL
SUBROUTINE
CLOAD
SET OPTION FOR SELECTED LINK OUTPUT
SET OPTION FOR NO SELECTED LINK OUTPUT
READ IN NETWORK FROM FLEXIBLE RECORD AND REARRANGE LINK DATA TO SAVE ONLY BNODE AND LAST LINK FLAG
READ IN PARAMETER RECORDS FROM TRIP MATRIX
ARE THE ZONE RANGES FROM THE TRIP MATRIX AND FLEXIBLE RECORD CONSISTENT?
NO
WRITE ERROR MESSAGE AND STOP 7
YES
READ A PATH RECORD AND UNPACK THE PATH. READ A TRIP RECORD
IS THE SELECTED LINK OPTION SET FOR OUTPUT?
YES
READ SELECTED LINK PARAMETER CARDS, MARK SELECTED LINKS IN CORE, OUTPUT PARAMETER RECORDS ON SELTRP, AND LIST PARAMETERS USED.
NO
INITIALIZE ARRAYS FOR LOADING. PRINT THE NUMBER OF TURNING MOVEMENTS TO SAVE
IS THE NUMBER OF TURNING MOVEMENTS TO SAVE > 30000?
YES
WRITE ERROR MESSAGE AND STOP 50
NO
PATH HOME ZONE: TRIP HOME ZONE
READ ANOTHER PATH RECORD AND UNPACK
READ ANOTHER TRIP RECORD
END OF DATA SET
END OF DATA SET
LOAD
LOAD THE TRIPS ON THE PATH
IS THIS A SELECTED LINKS LOAD
YES
CLOSE SELTRP DATA SET
NO
REMOVE SELECTED LINKS CODES FROM THE LINK ARRAY.
IS THIS AN ASSIGN SELF-BALANCING?
YES
SVLOAD
SAVE THE DIRECTIONAL AND NON-DIRECTION LINK VOLUMES AND TURN VOLUMES FOR THIS ITERATION ON UNIT 3.
NO
OUTLLT
PRINT THE LOADED NETWORK AND UPDATE THE FLEXIBLE DATA RECORD
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
RETURN
YES
PRINT REGRESSION OF ASSIGNED VOLUMES VERSUS PREVIOUS ASSIGNED VOLUMES BY ROUTE

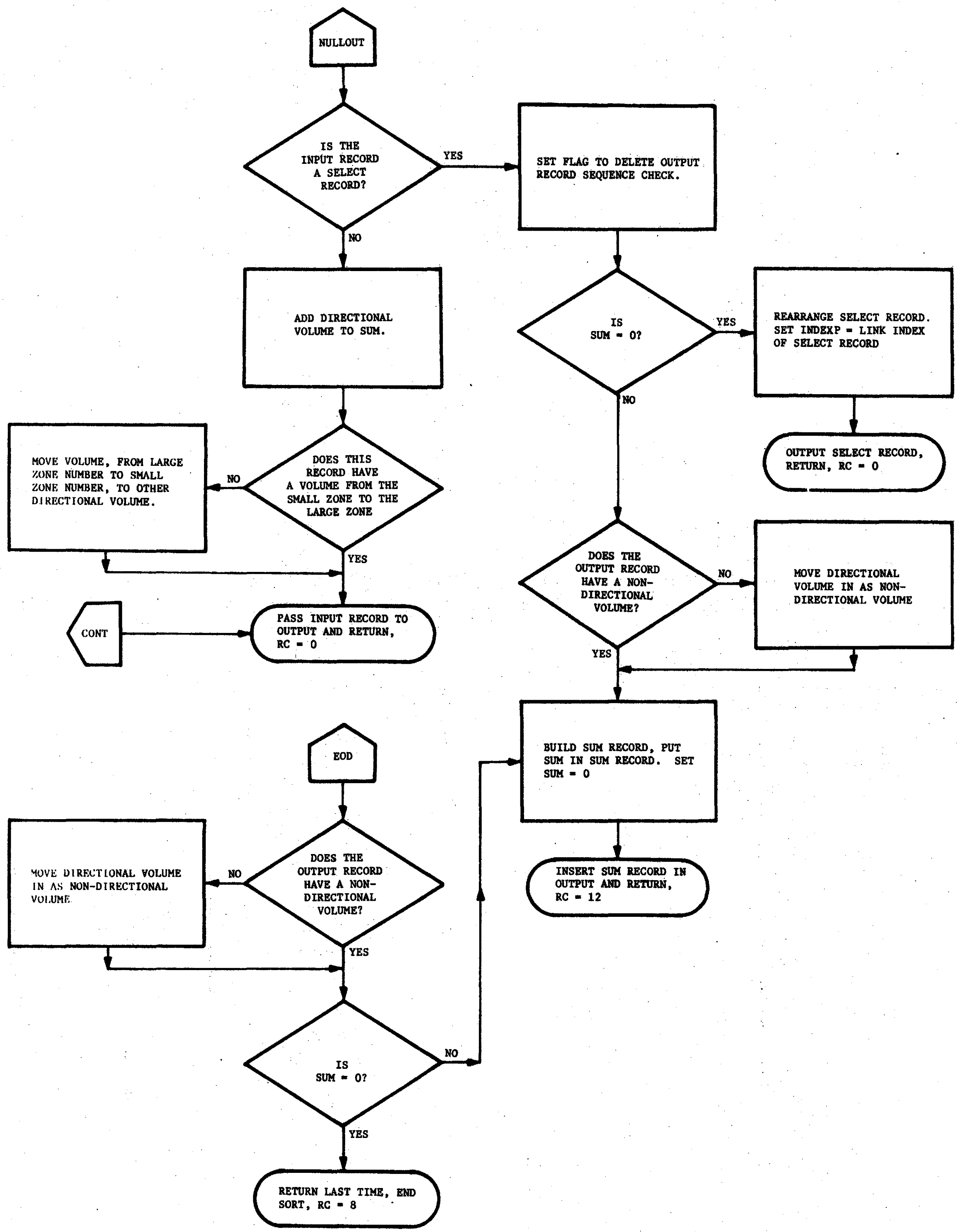

SUBROUTINE
CRD
READ A UNIT CONTROL CARD OR A CONTROL CARD
END OF DATA SET
NORMAL READ
IS THIS A CONTROL CARD?
YES
PRINT PACKAGE DATE AND NAME. SET I TO INDEX OF CONTROL CARD TYPE
RETURN
NO
IS IT A HEADER CARD?
YES
SAVE HEADER
NO
IS THIS UNIT CONTROL CARD?
NO
PRINT INVALID CONTROL CARD MESSAGE
STOP 9999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

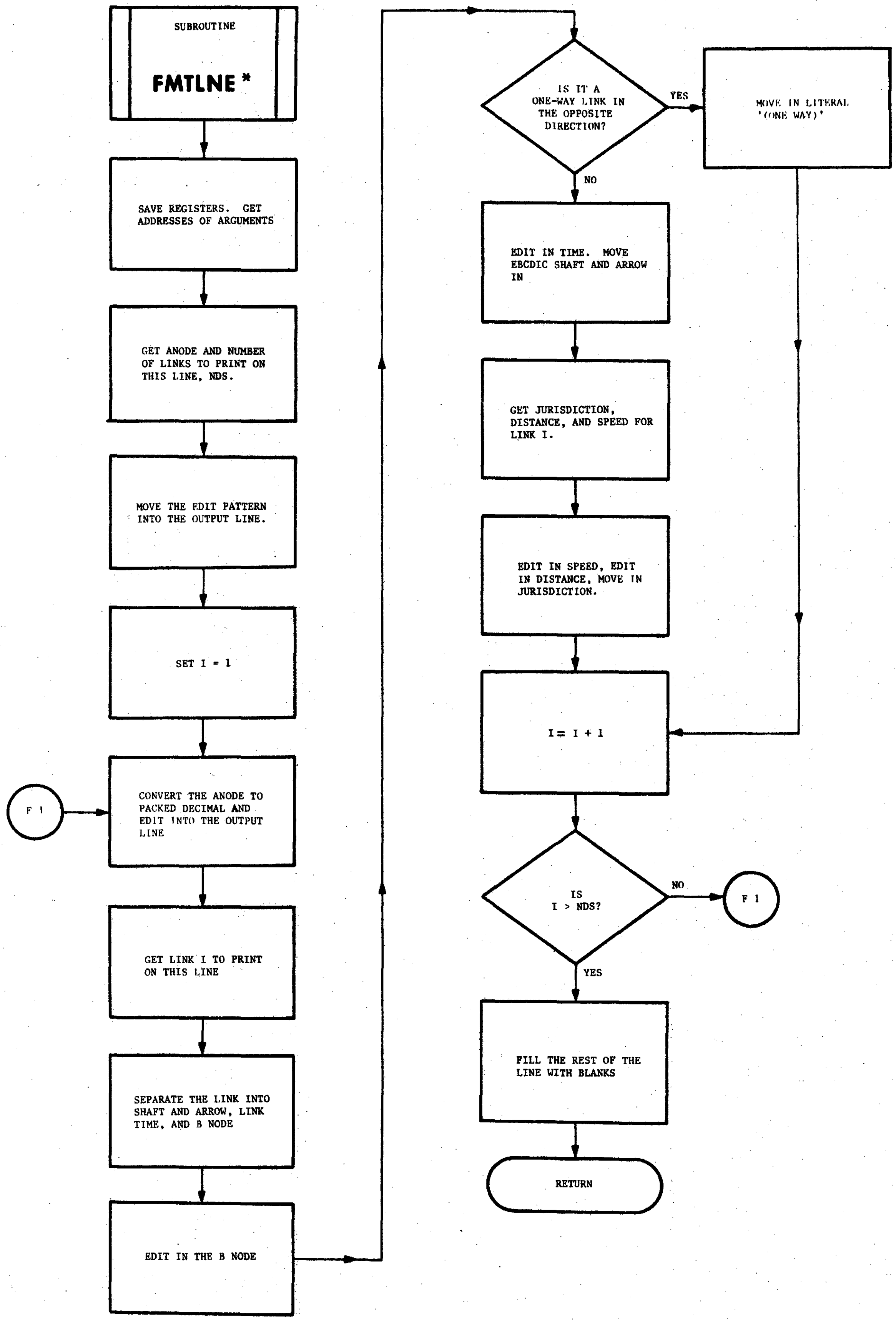

CRDINT
ARE THERE ANY CORRIDOR INTERCEPTS?
NO
RETURN
YES
READ HEADER RECORDS FOR ALL ASSIGNMENTS FROM FLEXIBLE DATA RECORD AND PRINT.
SET UP FORMATS FOR NUMBER OF ASSIGNMENTS. CALCULATE LENGTH OF RECORDS TO SORT.
VSORT
SORT CORRIDOR INTERCEPT RECORDS, IN CORE, ON THE CORRIDOR INTERCEPT
PRINT CORRIDOR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

E 35

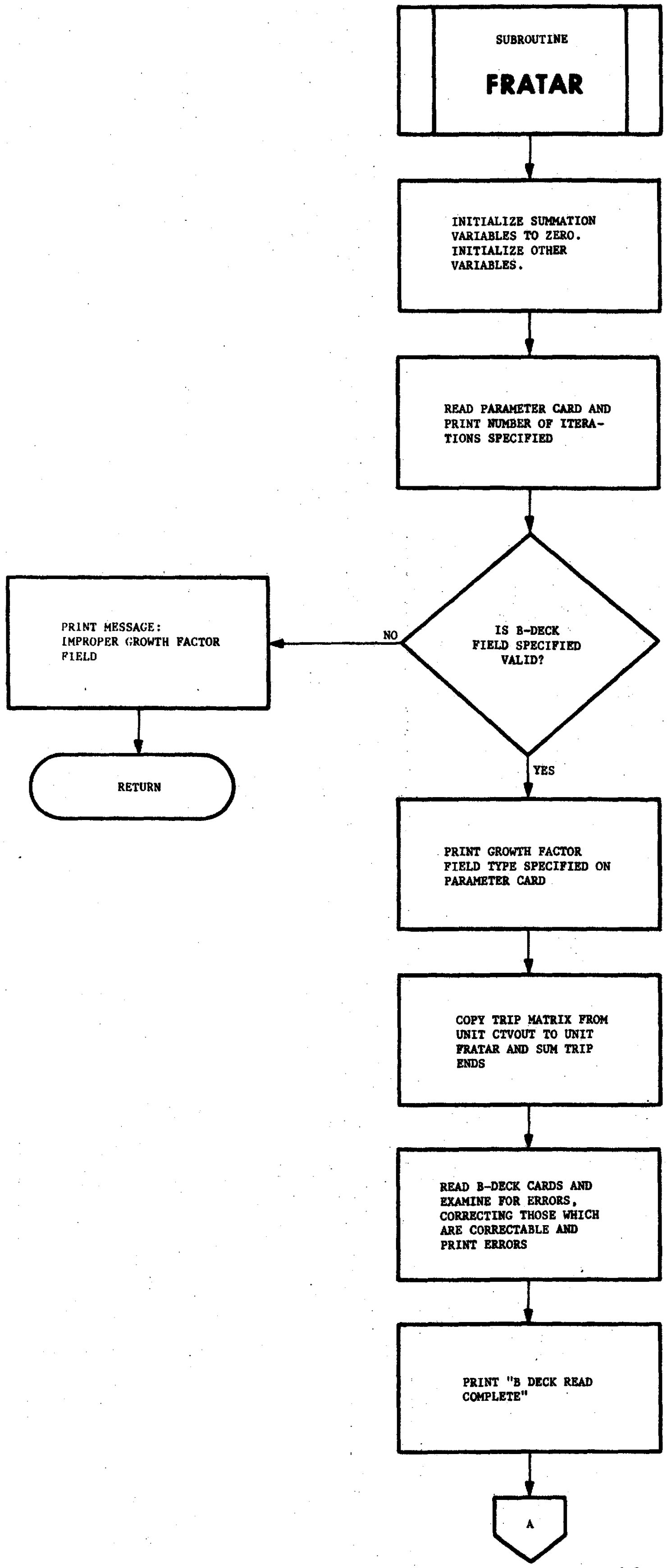


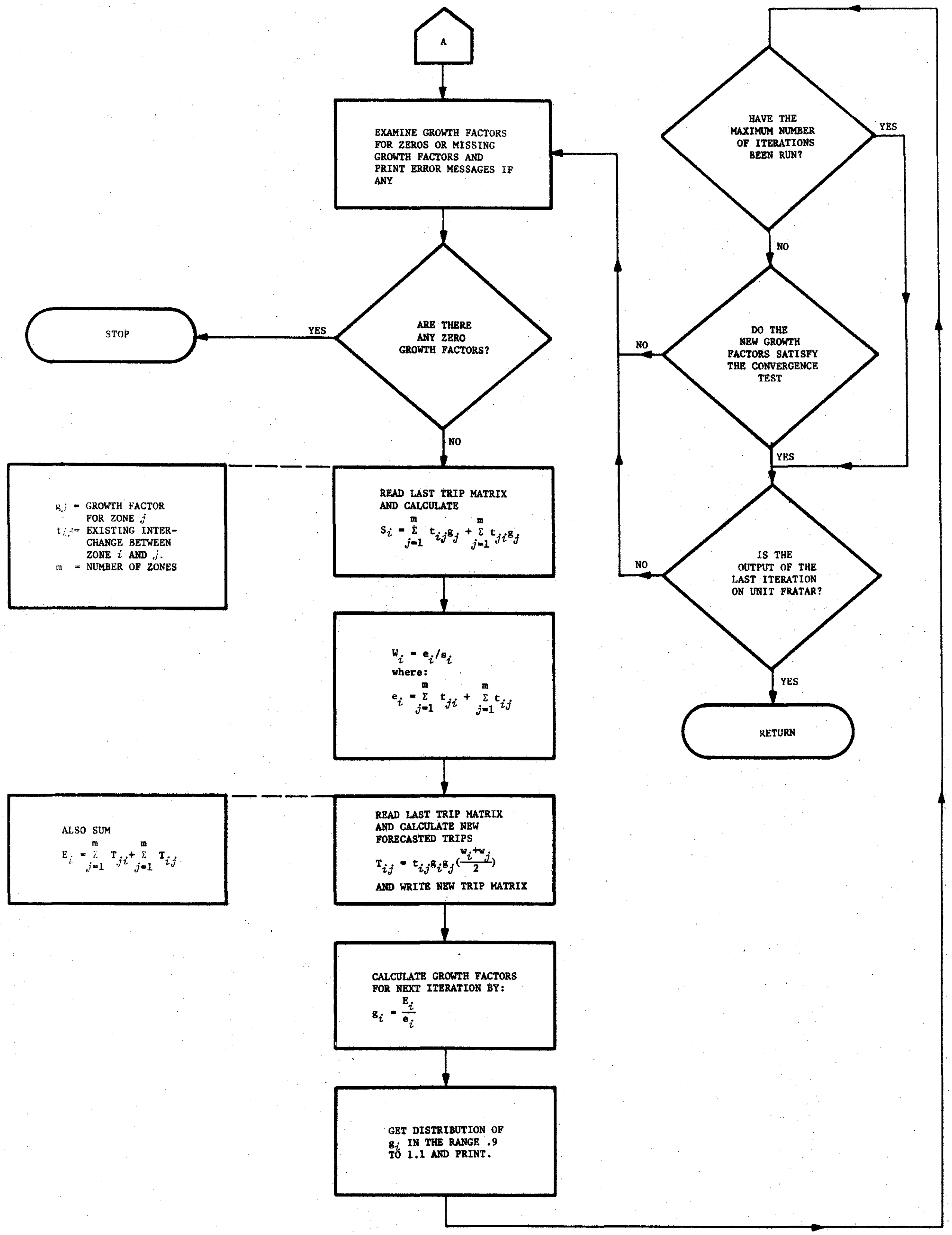
*ASSEMBLY LANGUAGE

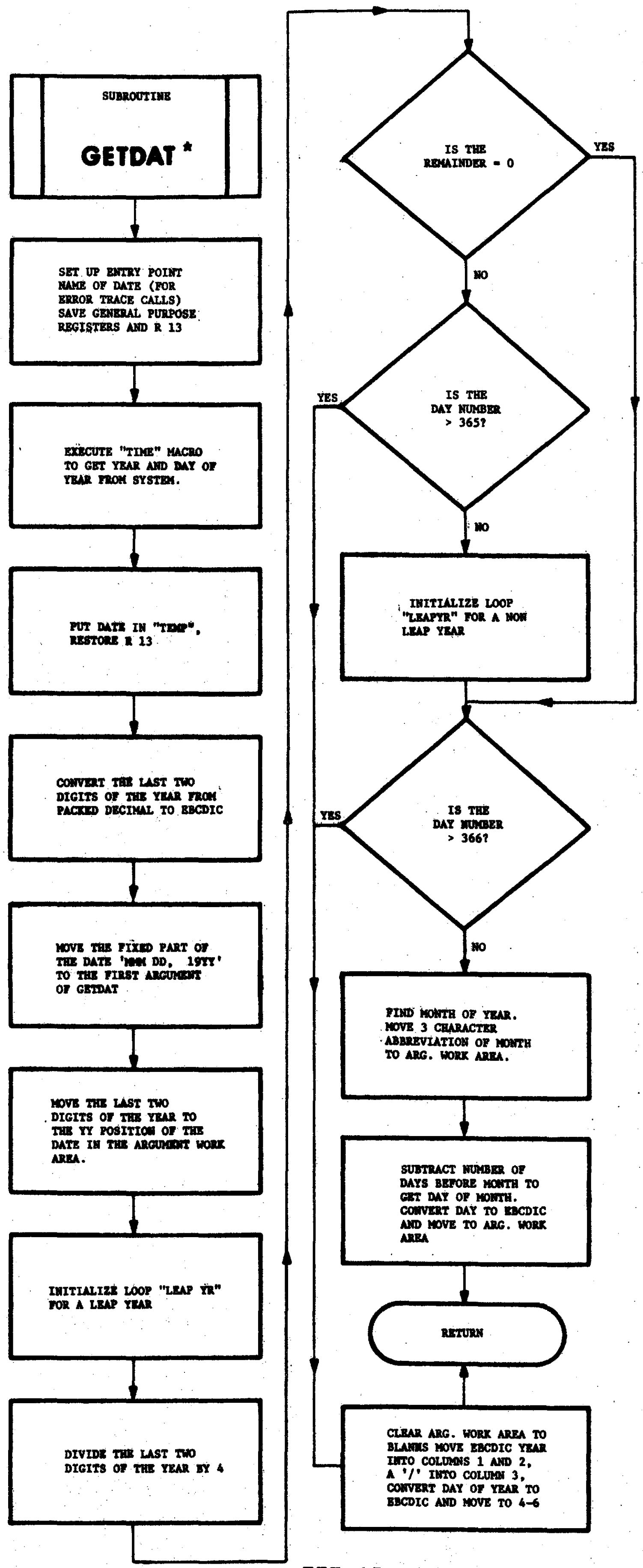

NULLOUT
IS THE INPUT RECORD A SELECT RECORD?
YES
SET FLAG TO DELETE OUTPUT RECORD SEQUENCE CHECK.
NO
ADD DIRECTIONAL VOLUME TO SUM.
IS SUM = 0?
YES
REARRANGE SELECT RECORD. SET INDEXP = LINK INDEX OF SELECT RECORD
OUTPUT SELECT RECORD, RETURN, RC = 0
NO
MOVE VOLUME, FROM LARGE ZONE NUMBER TO SMALL ZONE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



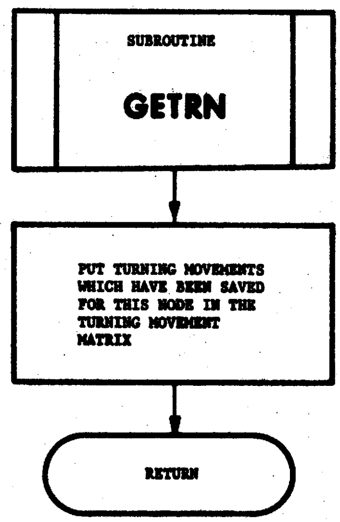
* ASSEMBLY LANGUAGE

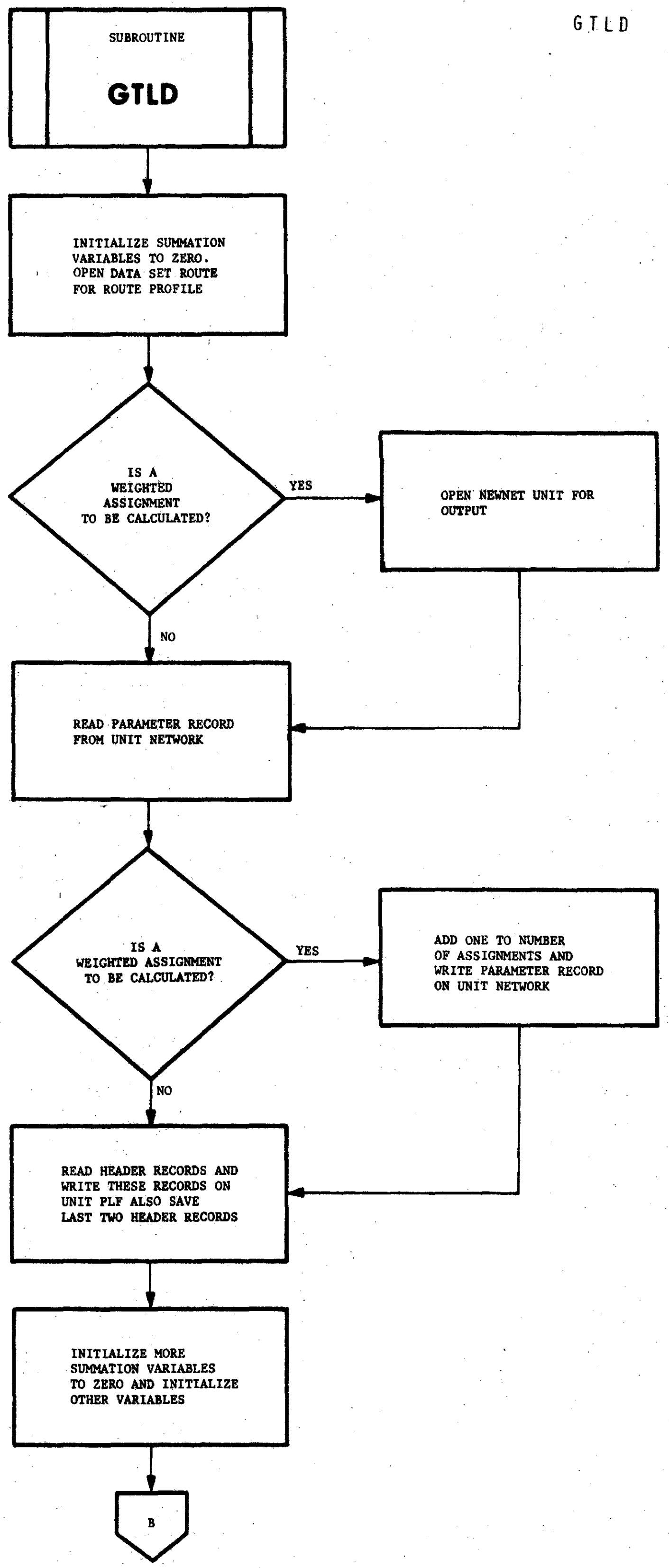

SUBROUTINE
FRATAR
INITIALIZE SUMMATION VARIABLES TO ZERO. INITIALIZE OTHER VARIABLES.
READ PARAMETER CARD AND PRINT NUMBER OF ITERA- TIONS SPECIFIED
IS B-DECK FIELD SPECIFIED VALID?
NO
PRINT MESSAGE: IMPROPER GROWTH FACTOR FIELD
RETURN
YES
PRINT GROWTH FACTOR FIELD TYPE SPECIFIED ON PARAMETER CARD
COPY 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

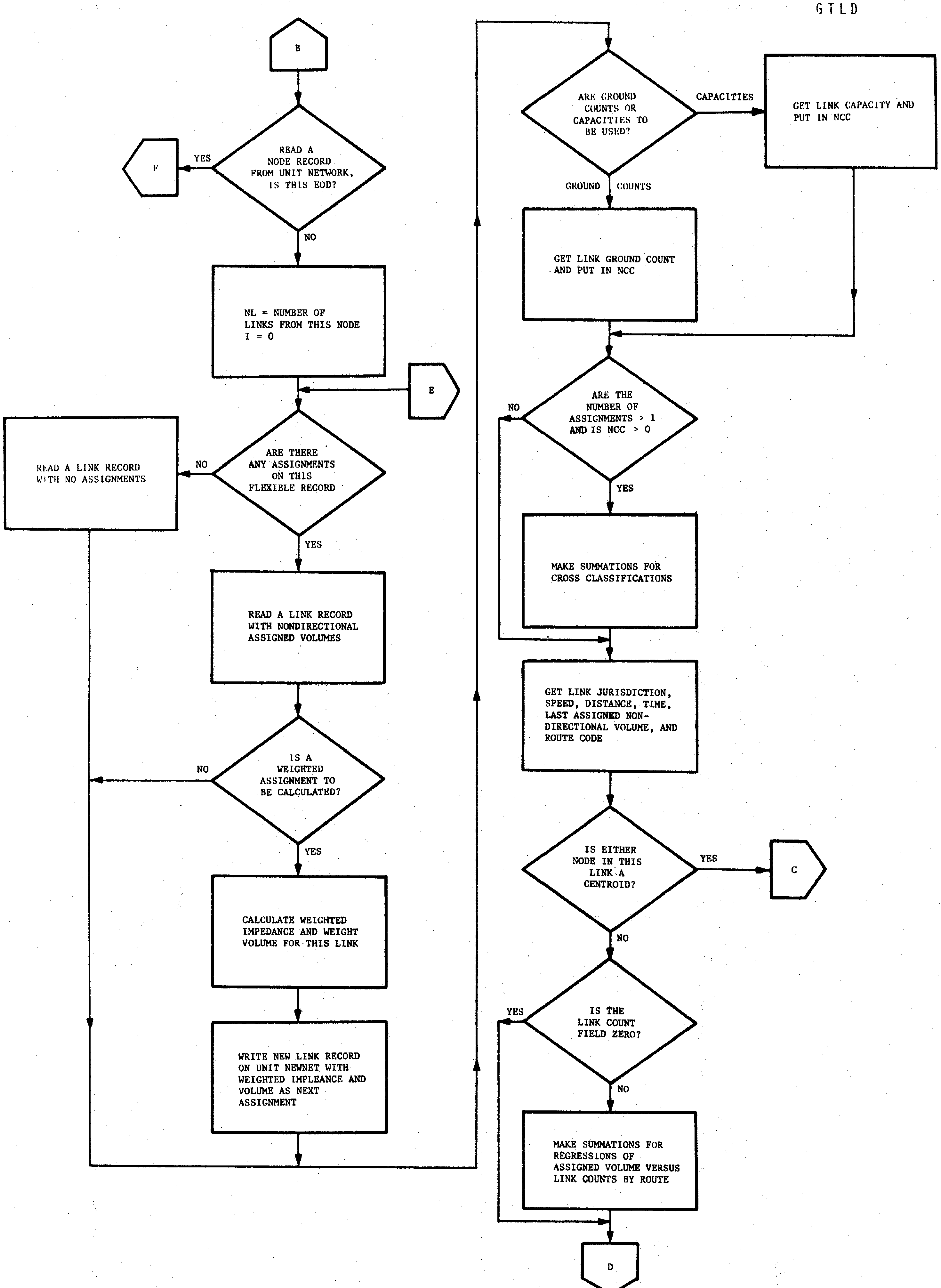

A
EXAMINE GROWTH FACTORS FOR ZEROS OR MISSING GROWTH FACTORS AND PRINT ERROR MESSAGES IF ANY
ARE THERE ANY ZERO GROWTH FACTORS?
YES
STOP
NO
g_j = GROWTH FACTOR FOR ZONE j
t_{ij} = EXISTING INTER-CHANGE BETWEEN ZONE i AND j.
m = NUMBER OF ZONES
READ LAST TRIP MATRIX AND CALCULATE
$S_i = \sum_{j=1}^{m} t_{ij}g_j + \sum_{j=1}^{m} t_{ji}g_j$
$W_i = e_i/s_i$
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

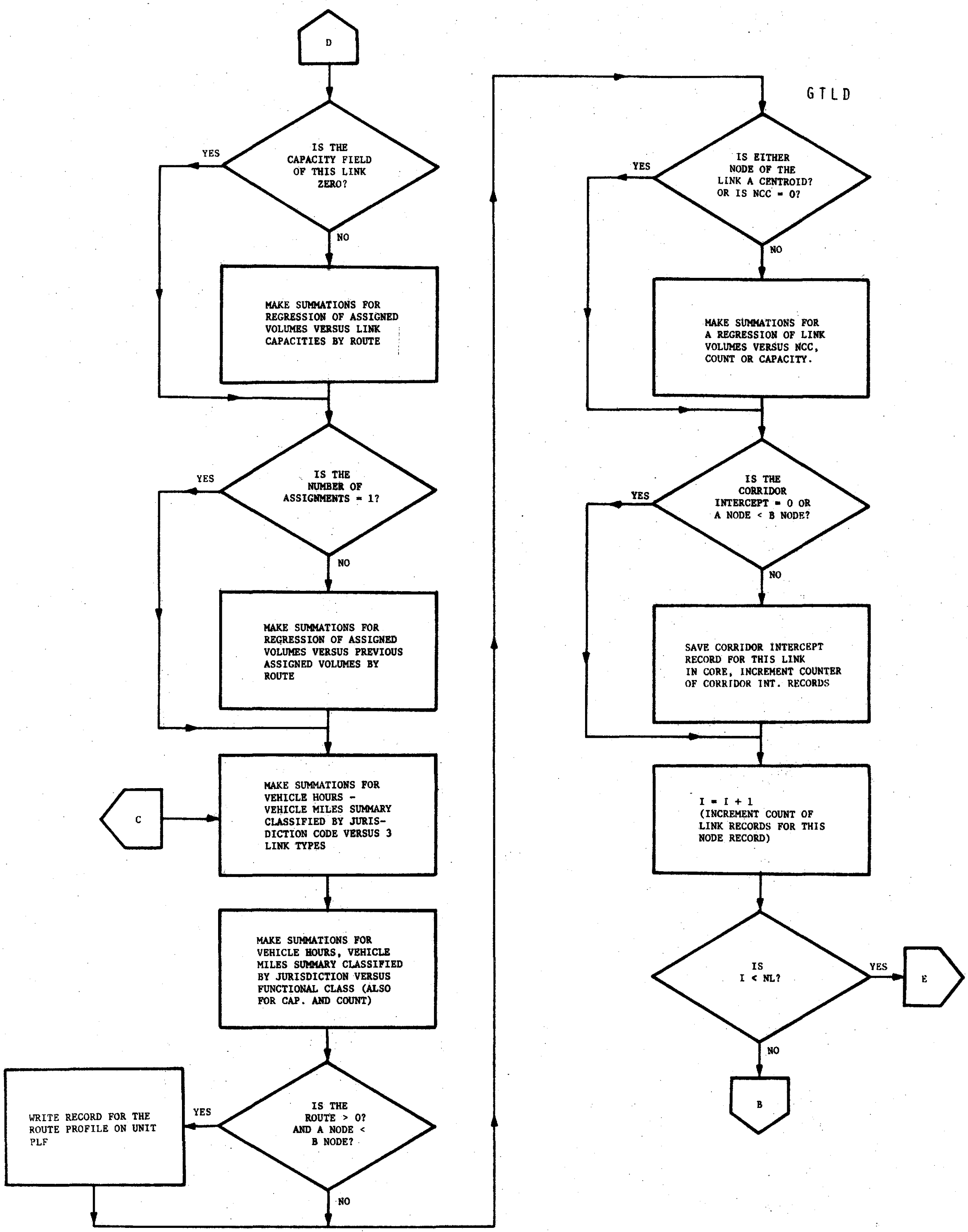


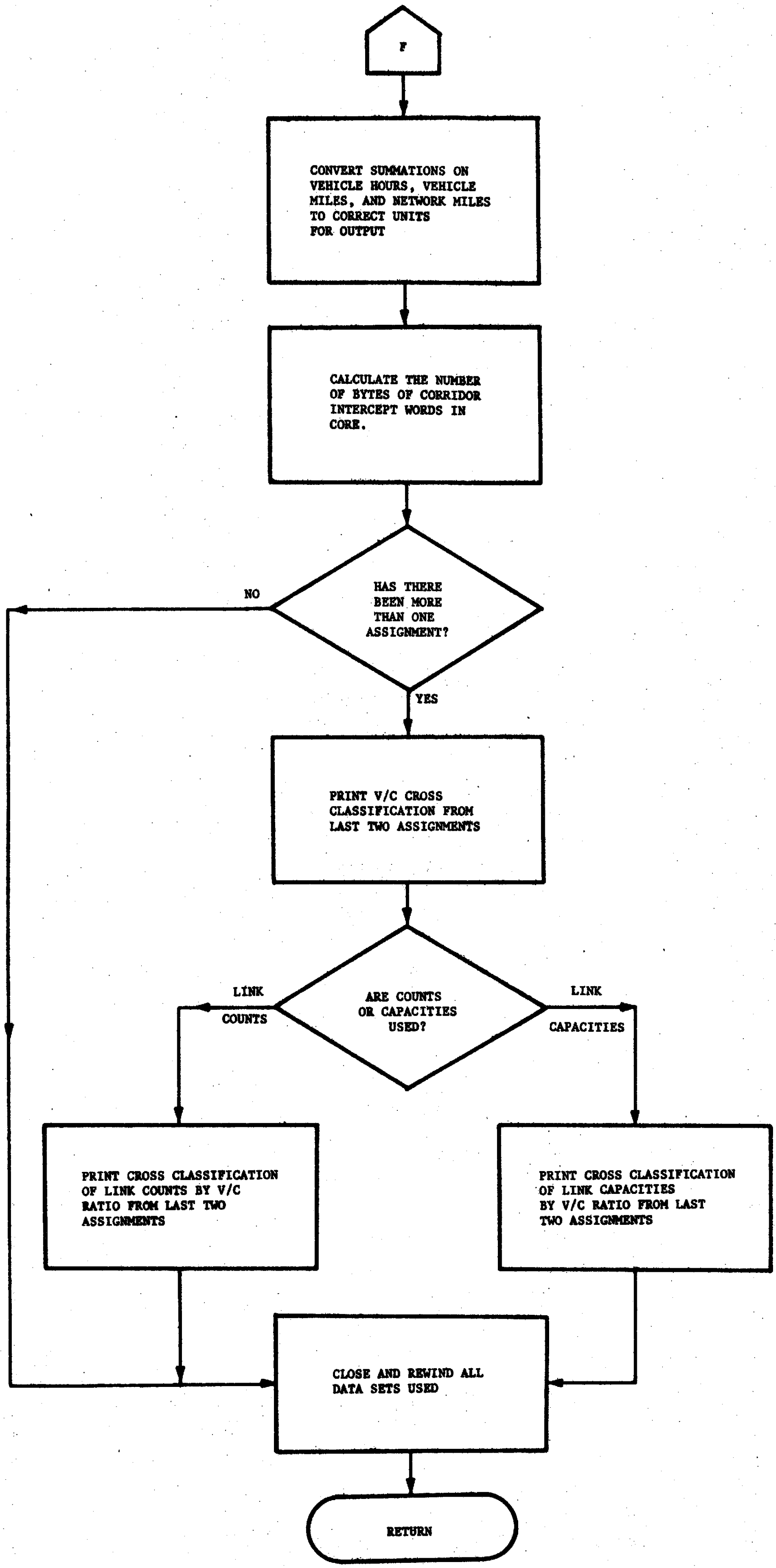
*ASSEMBLY LANGUAGE

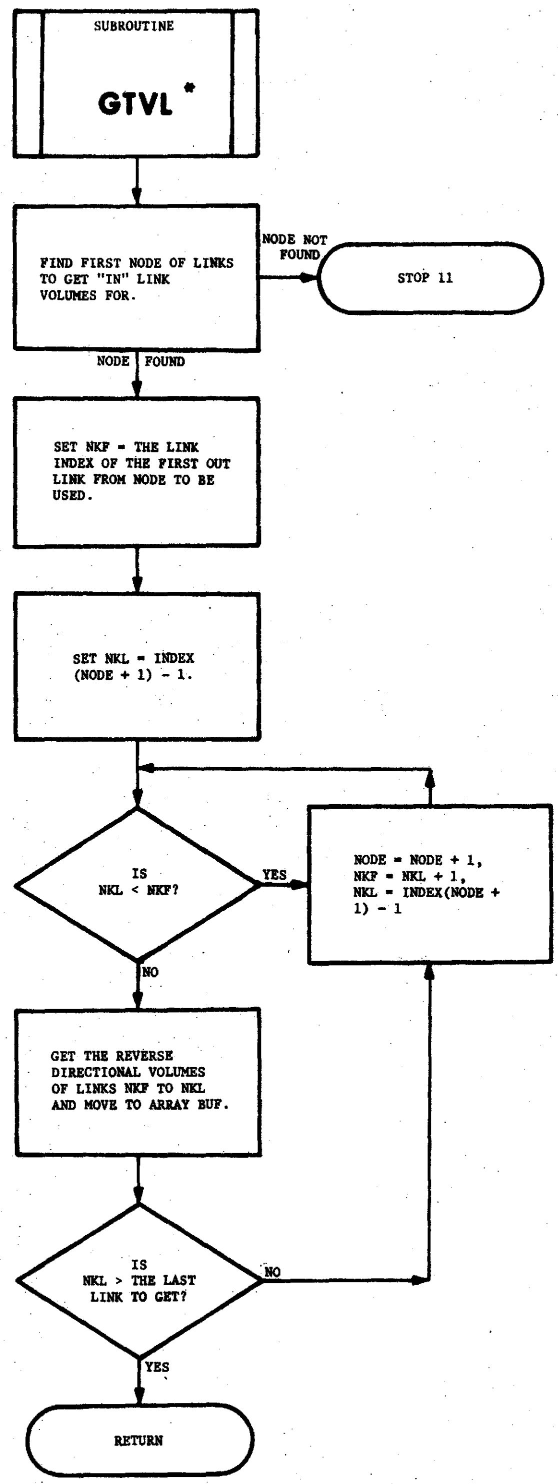

SUBROUTINE
GETRN
PUT TURNING MOVEMENTS WHICH HAVE BEEN SAVED FOR THIS NODE IN THE TURNING MOVEMENT MATRIX
RETURN


SUBROUTINE
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

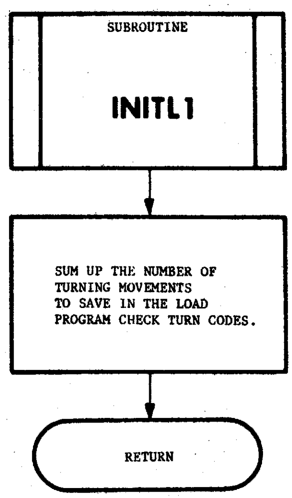

B
READ A NODE RECORD FROM UNIT NETWORK, IS THIS EOD?
YES
F
NO
NL = NUMBER OF LINKS FROM THIS NODE
I = 0
E
ARE THERE ANY ASSIGNMENTS ON THIS FLEXIBLE RECORD
NO
READ A LINK RECORD WITH NO ASSIGNMENTS
YES
READ A LINK RECORD WITH NONDIRECTIONAL ASSIGNED VOLUMES
IS A WEIGHTED ASSIGNMENT TO BE CALCULATED?
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

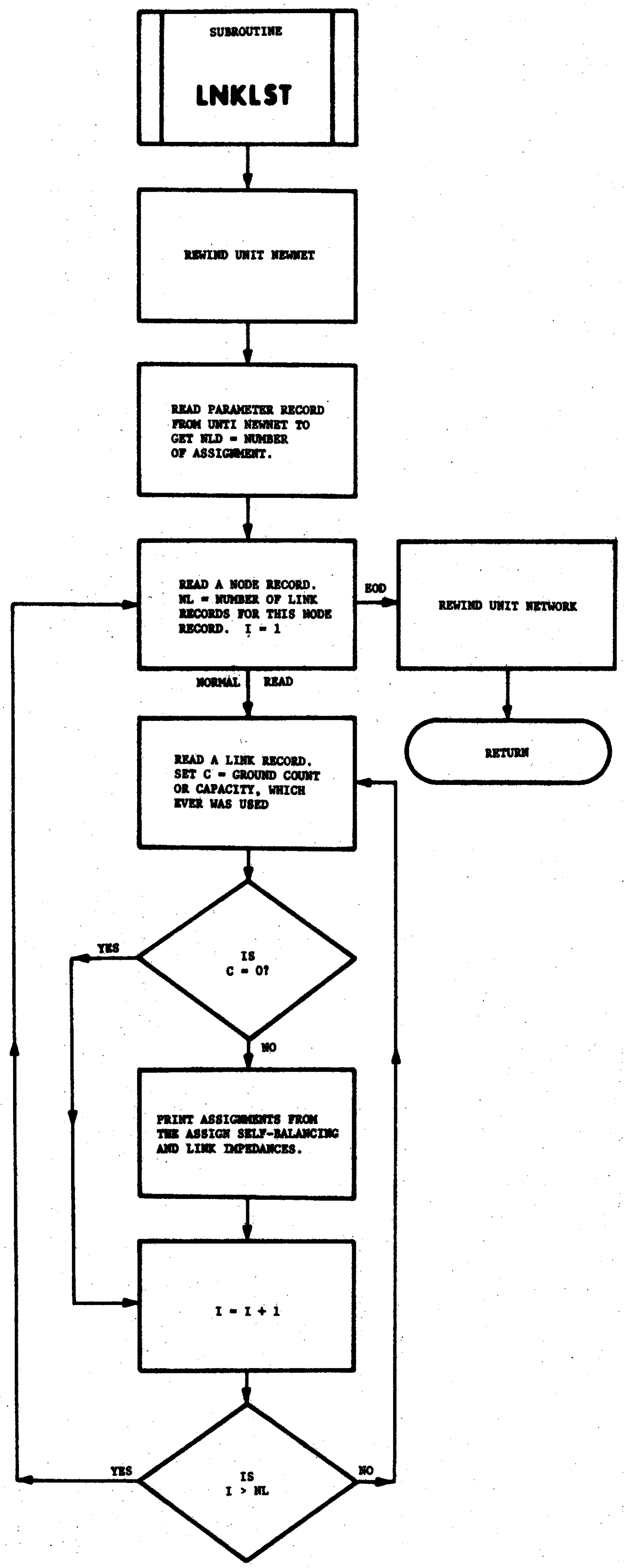

D
IS THE CAPACITY FIELD OF THIS LINK ZERO?
YES
NO
MAKE SUMMATIONS FOR REGRESSION OF ASSIGNED VOLUMES VERSUS LINK CAPACITIES BY ROUTE
IS THE NUMBER OF ASSIGNMENTS = 1?
YES
NO
MAKE SUMMATIONS FOR REGRESSION OF ASSIGNED VOLUMES VERSUS PREVIOUS ASSIGNED VOLUMES BY ROUTE
C
MAKE SUMMATIONS FOR VEHICLE HOURS - 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

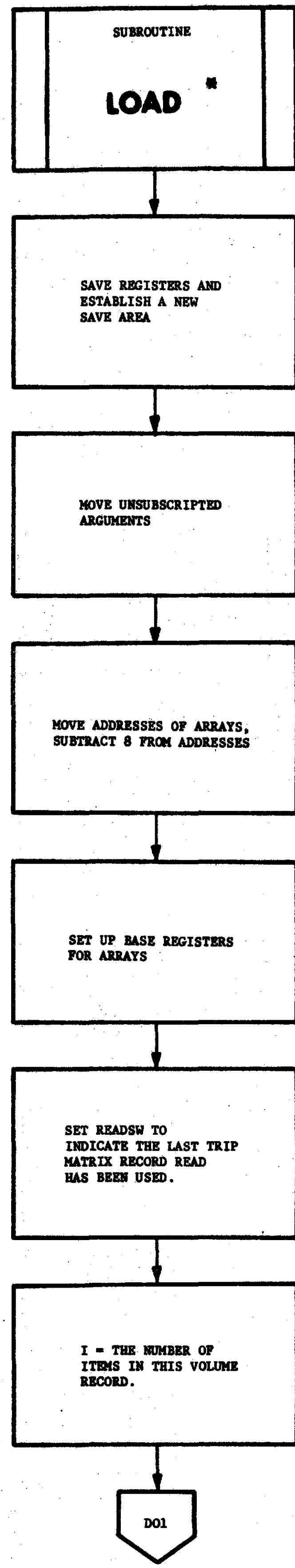

F
CONVERT SUMMATIONS ON VEHICLE HOURS, VEHICLE MILES, AND NETWORK MILES TO CORRECT UNITS FOR OUTPUT
CALCULATE THE NUMBER OF BYTES OF CORRIDOR INTERCEPT WORDS IN CORE.
HAS THERE BEEN MORE THAN ONE ASSIGNMENT?
NO
YES
PRINT V/C CROSS CLASSIFICATION FROM LAST TWO ASSIGNMENTS
ARE COUNTS OR CAPACITIES USED?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



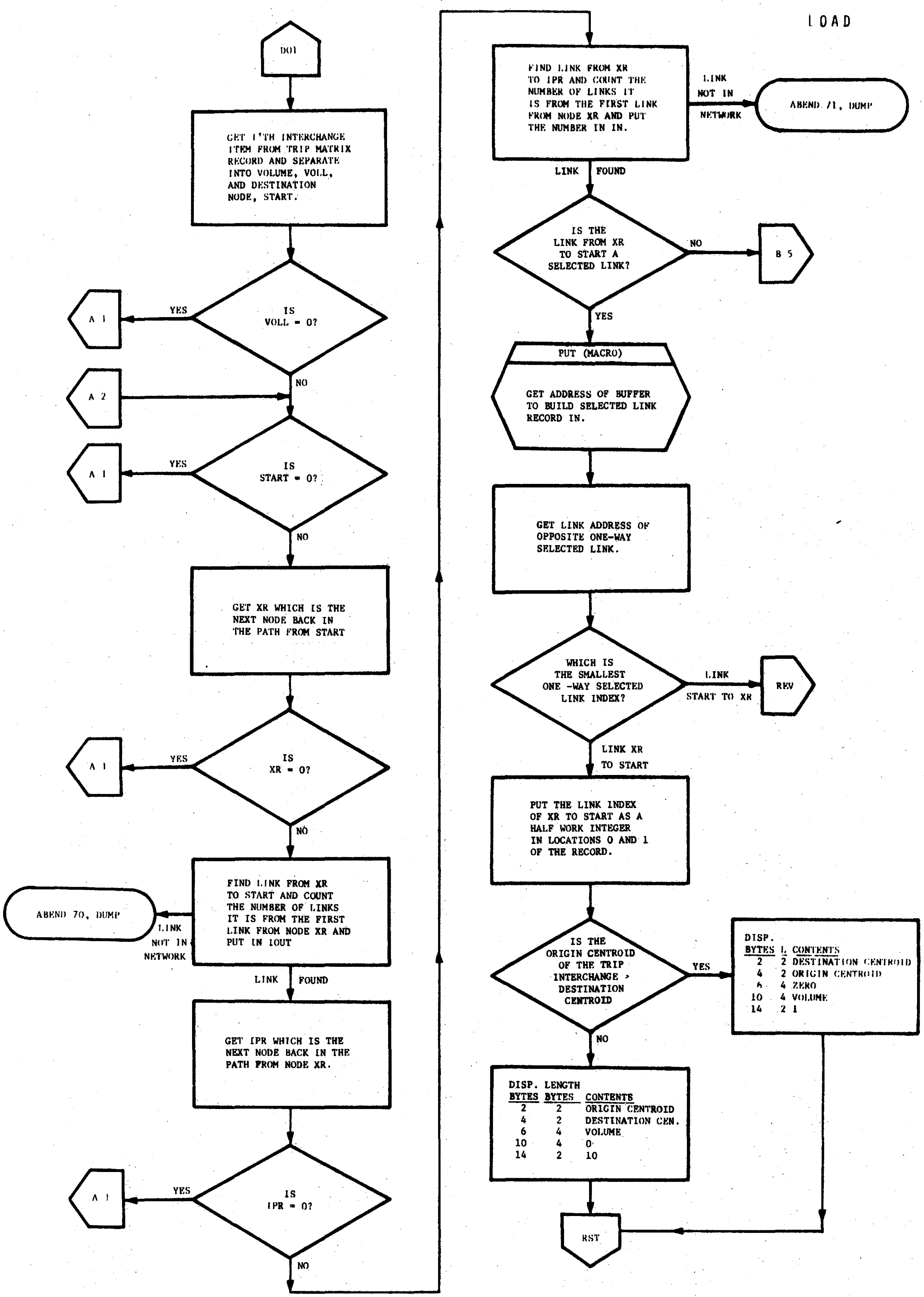
*ASSEMBLY LANGUAGE

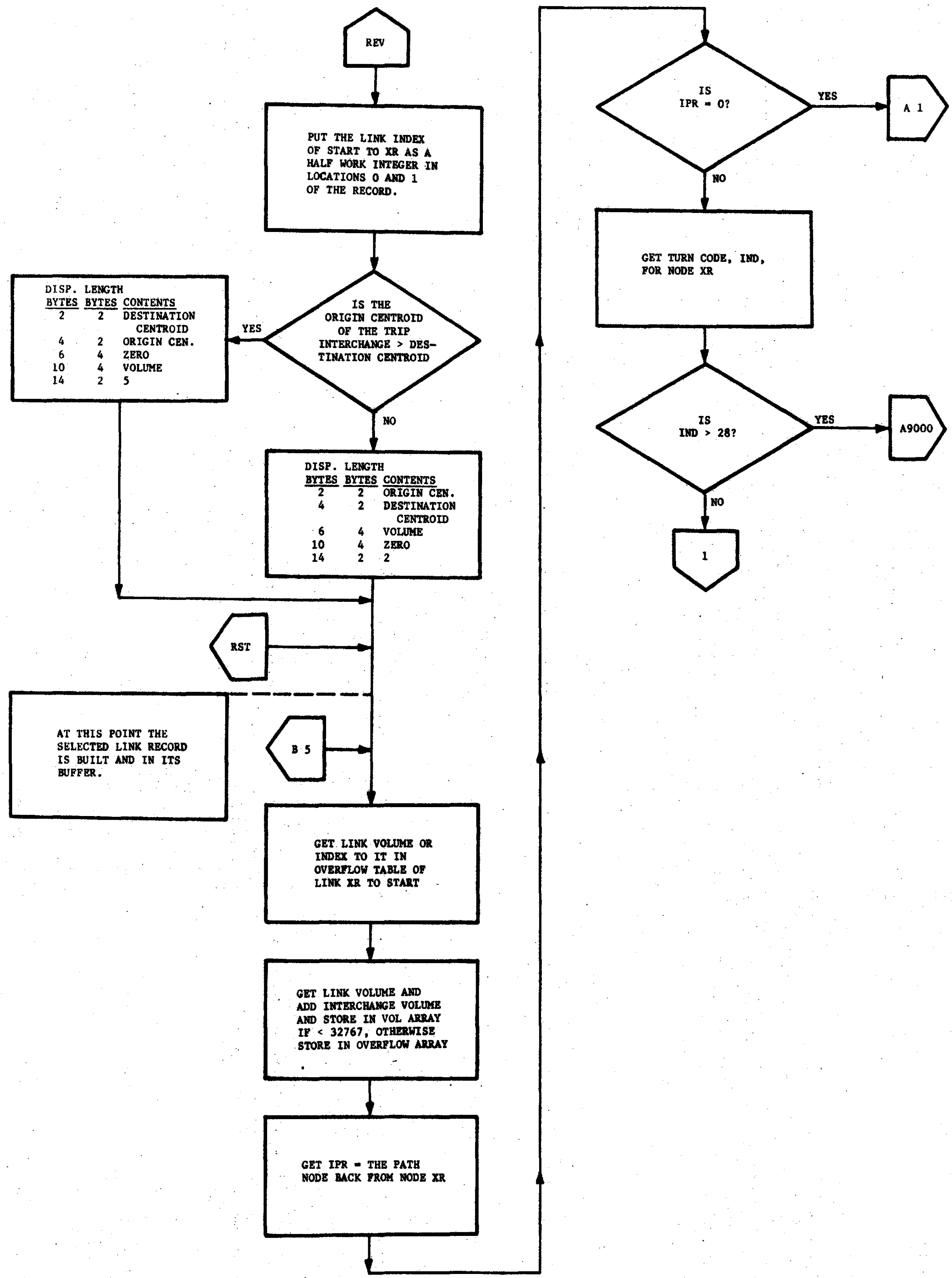

SUBROUTINE
INITL1
SUM UP THE NUMBER OF
TURNING MOVEMENTS
TO SAVE IN THE LOAD
PROGRAM CHECK TURN CODES.
RETURN

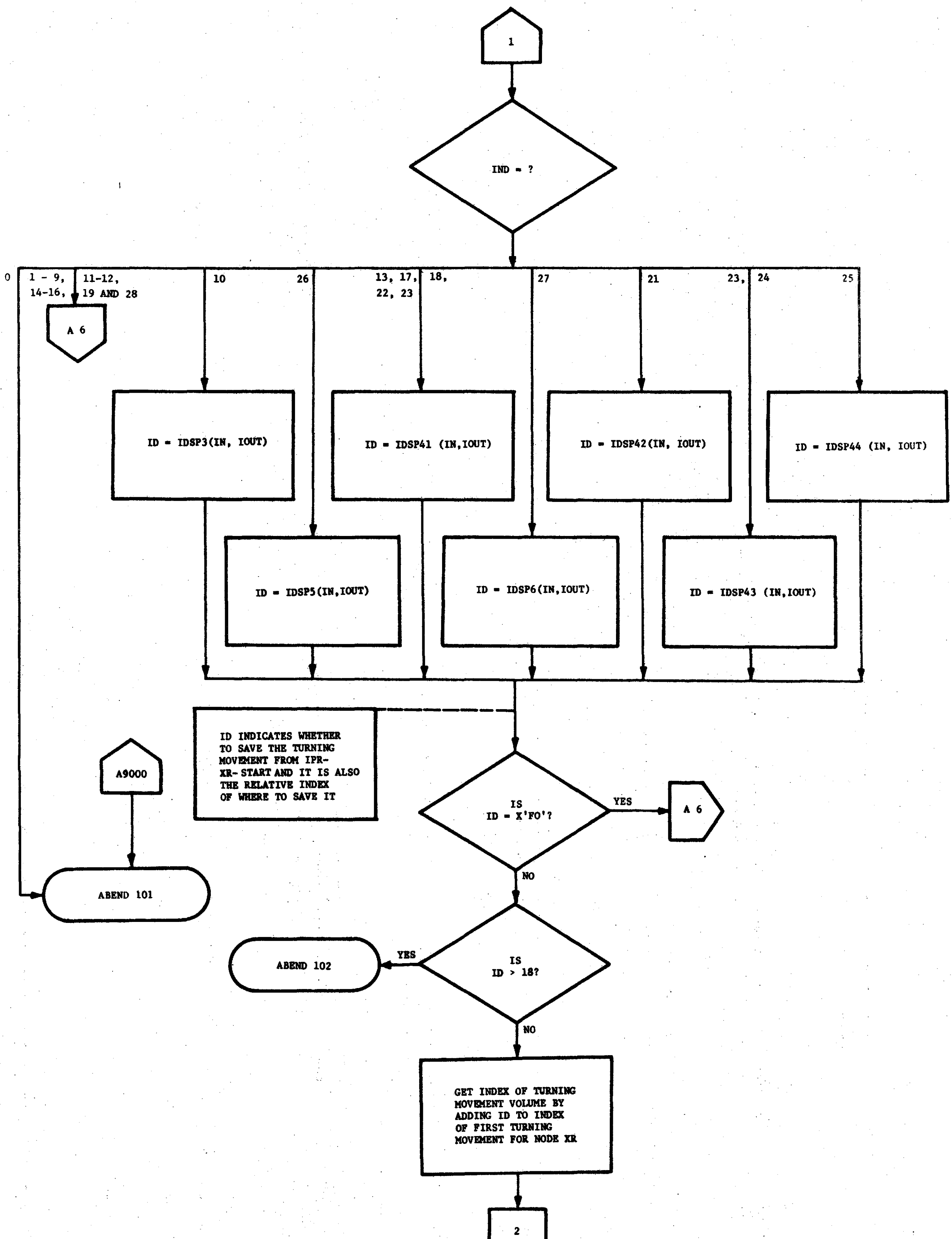

SUBROUTINE
LNKLST
REWIND UNIT NEWNET
READ PARAMETER RECORD FROM UNTI NEWNET TO GET NLD = NUMBER OF ASSIGNMENT.
READ A NODE RECORD. NL = NUMBER OF LINK RECORDS FOR THIS NODE RECORD. I = 1
EOD
REWIND UNIT NETWORK
RETURN
NORMAL READ
READ A LINK RECORD. SET C = GROUND COUNT OR CAPACITY, WHICH EVER WAS USED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

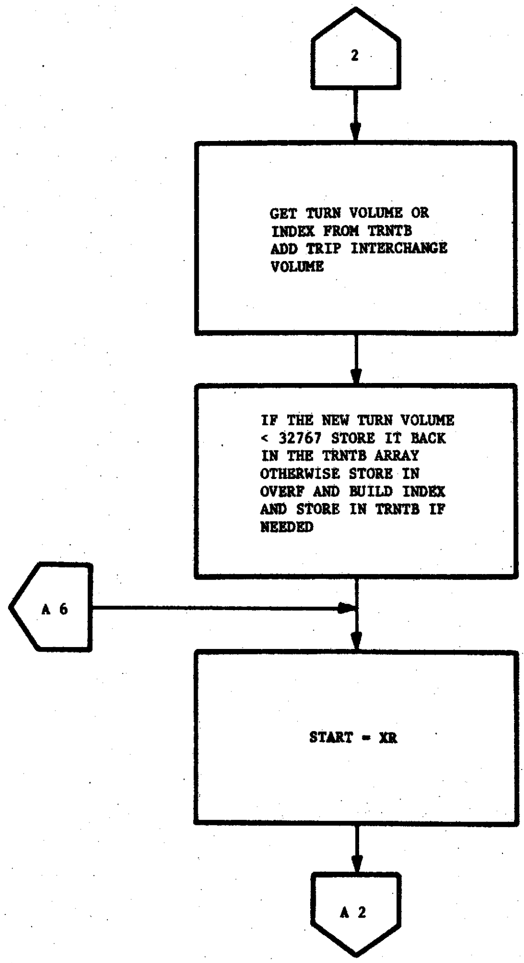


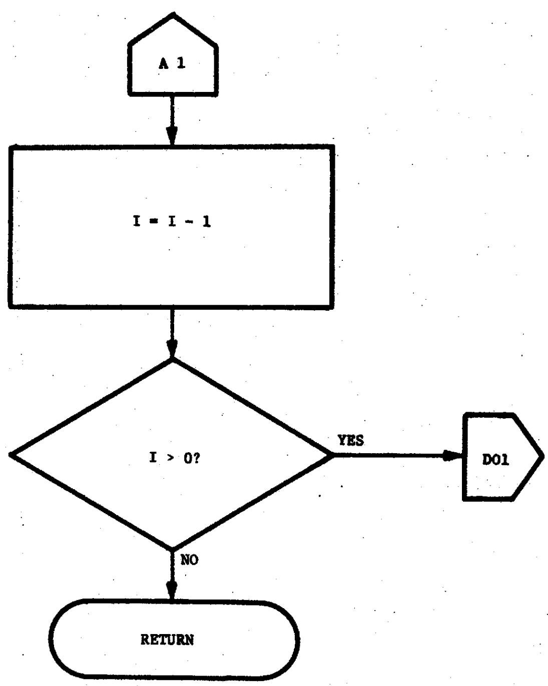
*ASSEMBLY LANGUAGE

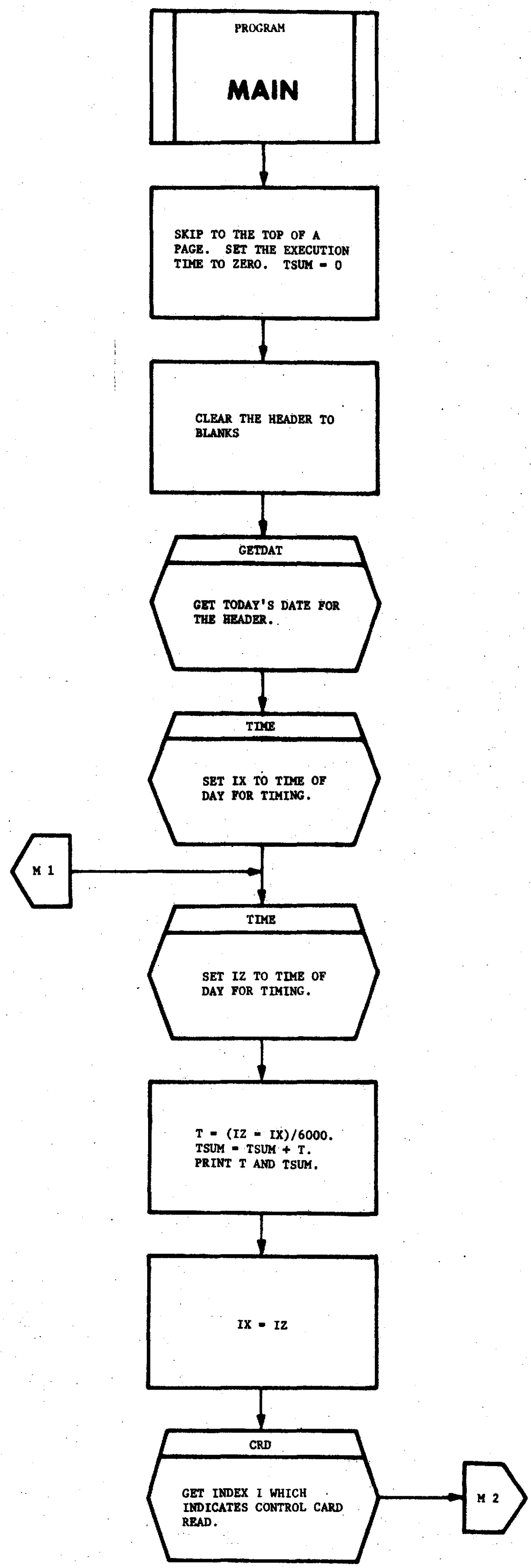

DO1
GET I'TH INTERCHANGE ITEM FROM TRIP MATRIX RECORD AND SEPARATE INTO VOLUME, VOLL, AND DESTINATION NODE, START.
IS VOLL = 0?
YES
A 1
NO
A 2
IS START = 0?
YES
A 1
NO
GET XR WHICH IS THE NEXT NODE BACK IN THE PATH FROM START
IS XR = 0?
YES
A 1
NO
FIND LINK FROM XR TO START AND COUNT THE NUMBER OF LINKS IT IS FROM THE FIRST LINK FROM NODE XR AND PUT IN IOUT
LINK NOT IN NETWORK
ABEND 70, DUMP
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

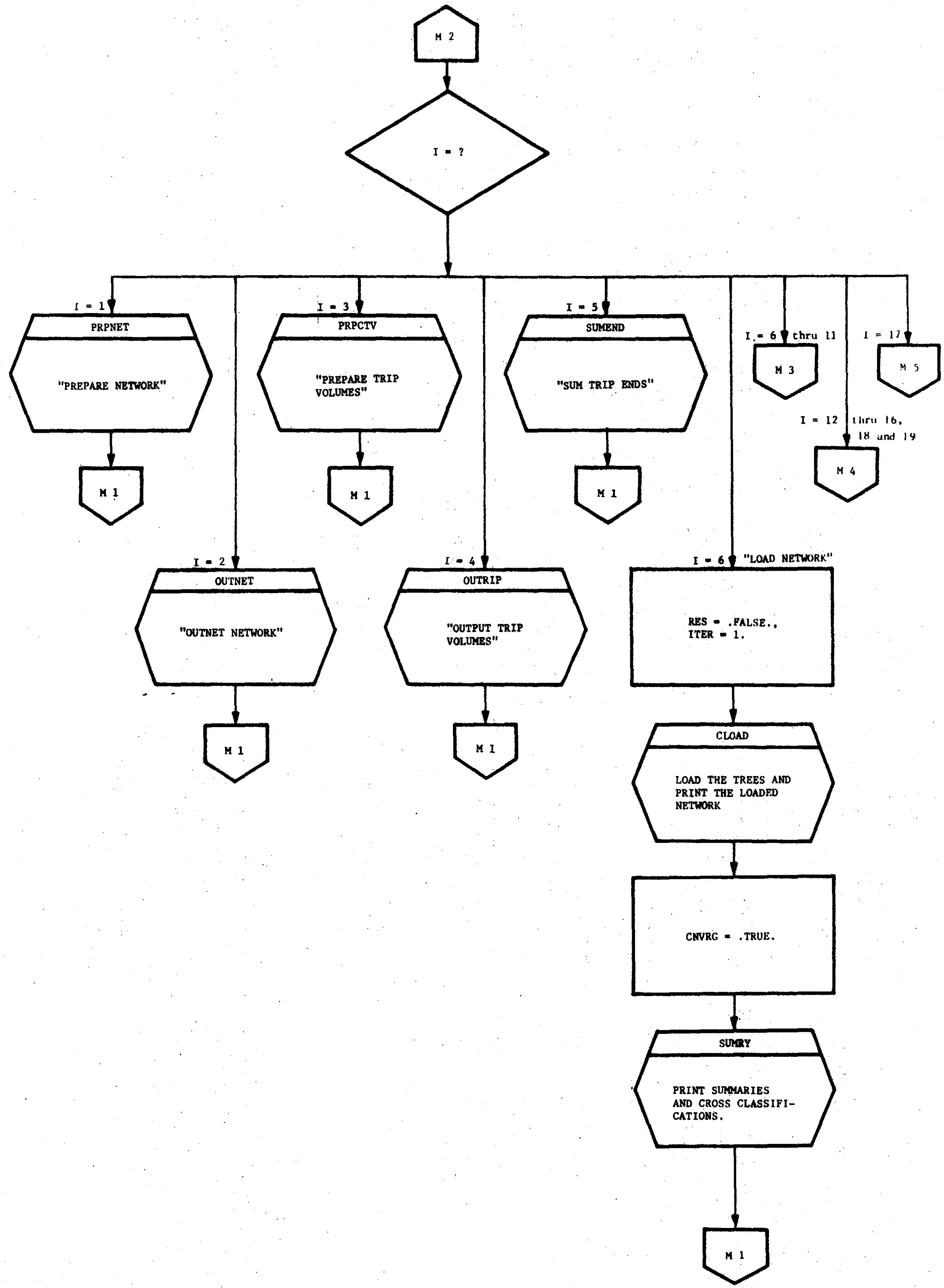

REV
PUT THE LINK INDEX OF START TO XR AS A HALF WORK INTEGER IN LOCATIONS 0 AND 1 OF THE RECORD.
IS THE ORIGIN CENTROID OF THE TRIP INTERCHANGE > DESTINATION CENTROID
YES
DISP. LENGTH
BYTES BYTES CONTENTS
2 2 DESTINATION CENTROID
4 2 ORIGIN CEN.
6 4 ZERO
10 4 VOLUME
14 2 5
NO
DISP. LENGTH
BYTES BYTES CONTENTS
2 2 ORIGIN CEN.
4 2 DESTINATION CENTROID
6 4 VOLUME
10 4 ZERO
14 2 2
RST
AT THIS POINT THE SELECTED LINK RECORD IS BUILT AND IN ITS BUFFER.
B 5
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

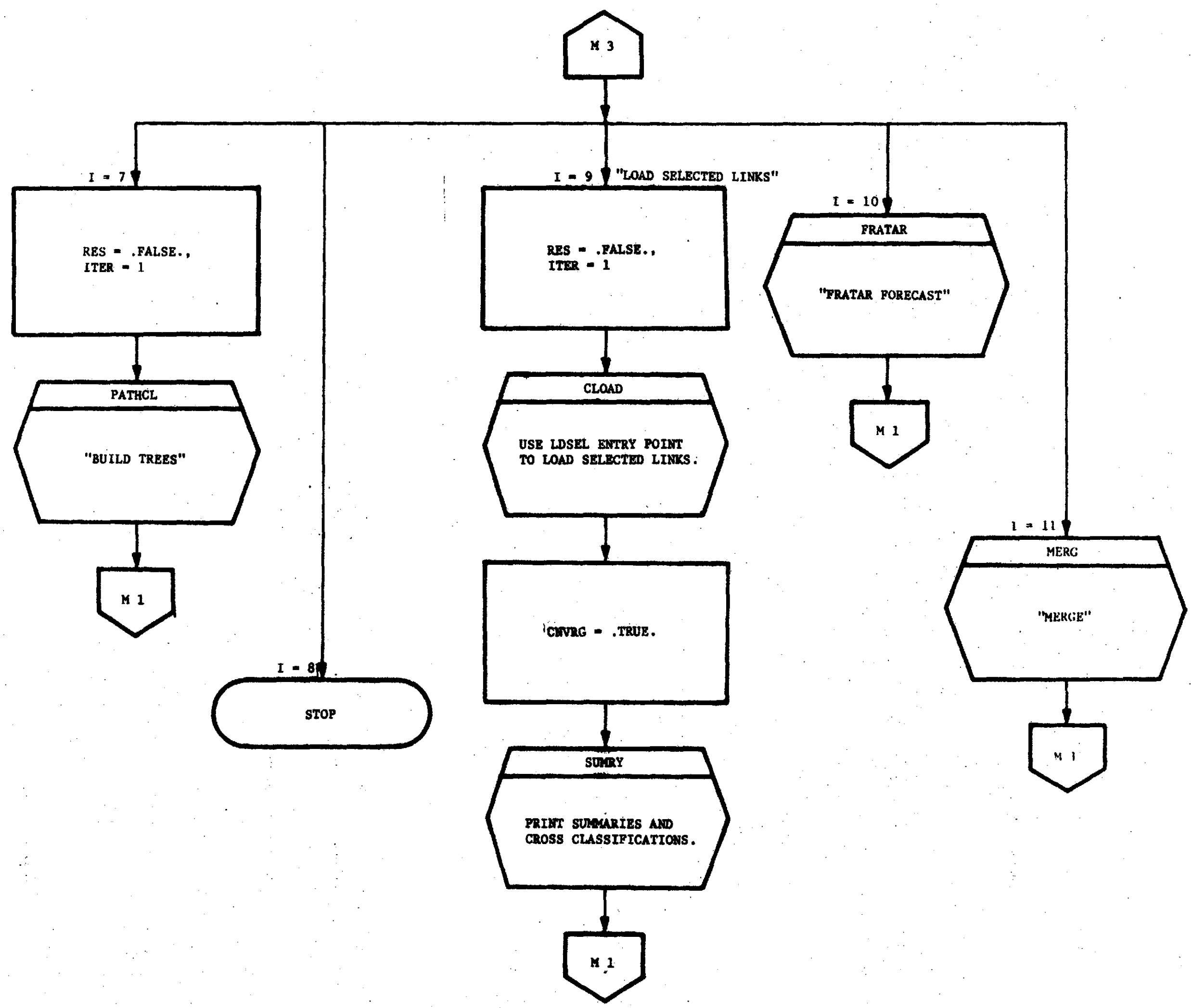

1
IND = ?
0
1 - 9,
14-16,
11-12,
19 AND 28
10
26
13, 17,
22, 23
18,
27
21
23,
24
25
A 6
ID = IDSP3(IN, IOUT)
ID = IDSP41 (IN,IOUT)
ID = IDSP42(IN, IOUT)
ID = IDSP44 (IN, IOUT)
ID = IDSP5(IN,IOUT)
ID = IDSP6(IN,IOUT)
ID = IDSP43 (IN,IOUT)
ID INDICATES WHETHER TO SAVE THE TURNING MOVEMENT FROM IPR-XR-START 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

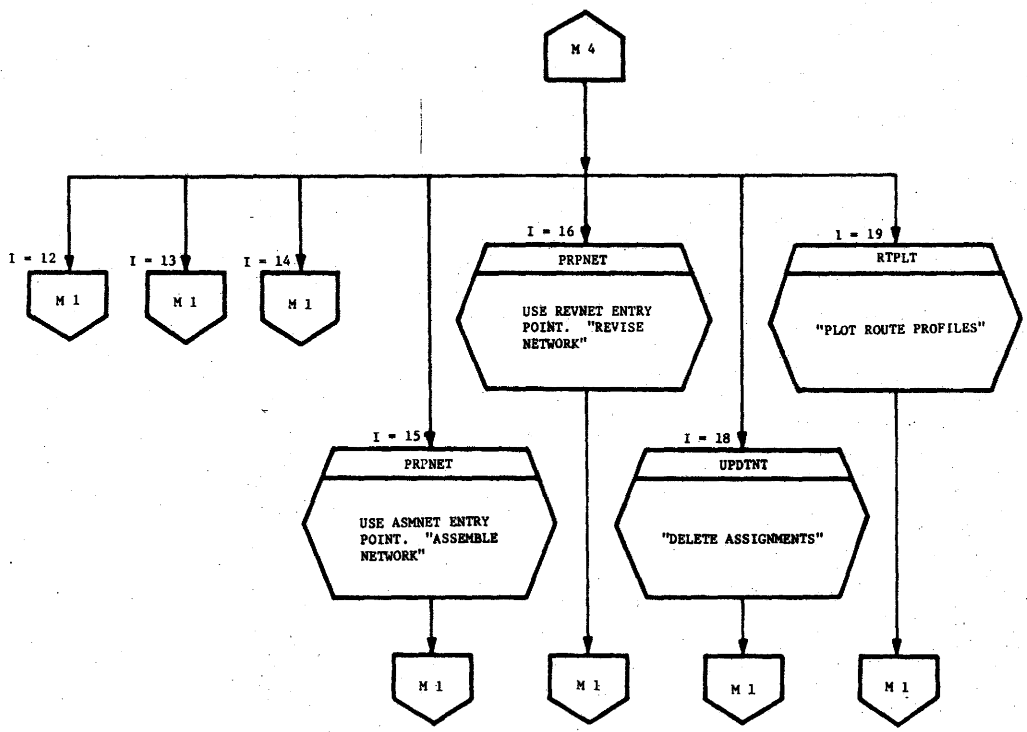

2
GET TURN VOLUME OR INDEX FROM TRNTB ADD TRIP INTERCHANGE VOLUME
IF THE NEW TURN VOLUME < 32767 STORE IT BACK IN THE TRNTB ARRAY OTHERWISE STORE IN OVERF AND BUILD INDEX AND STORE IN TRNTB IF NEEDED
A 6
START = XR
A 2

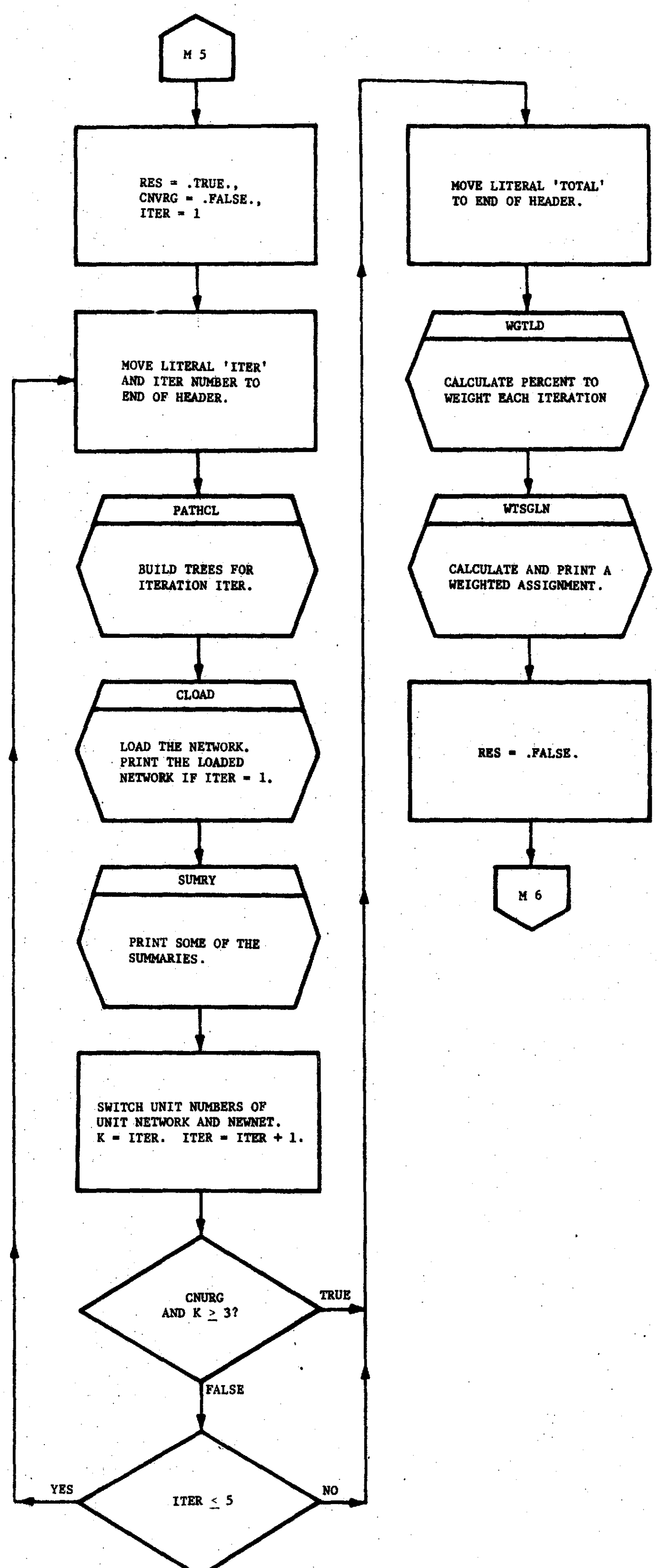

A 1
I = I - 1
I > 0?
YES
D01
NO
RETURN

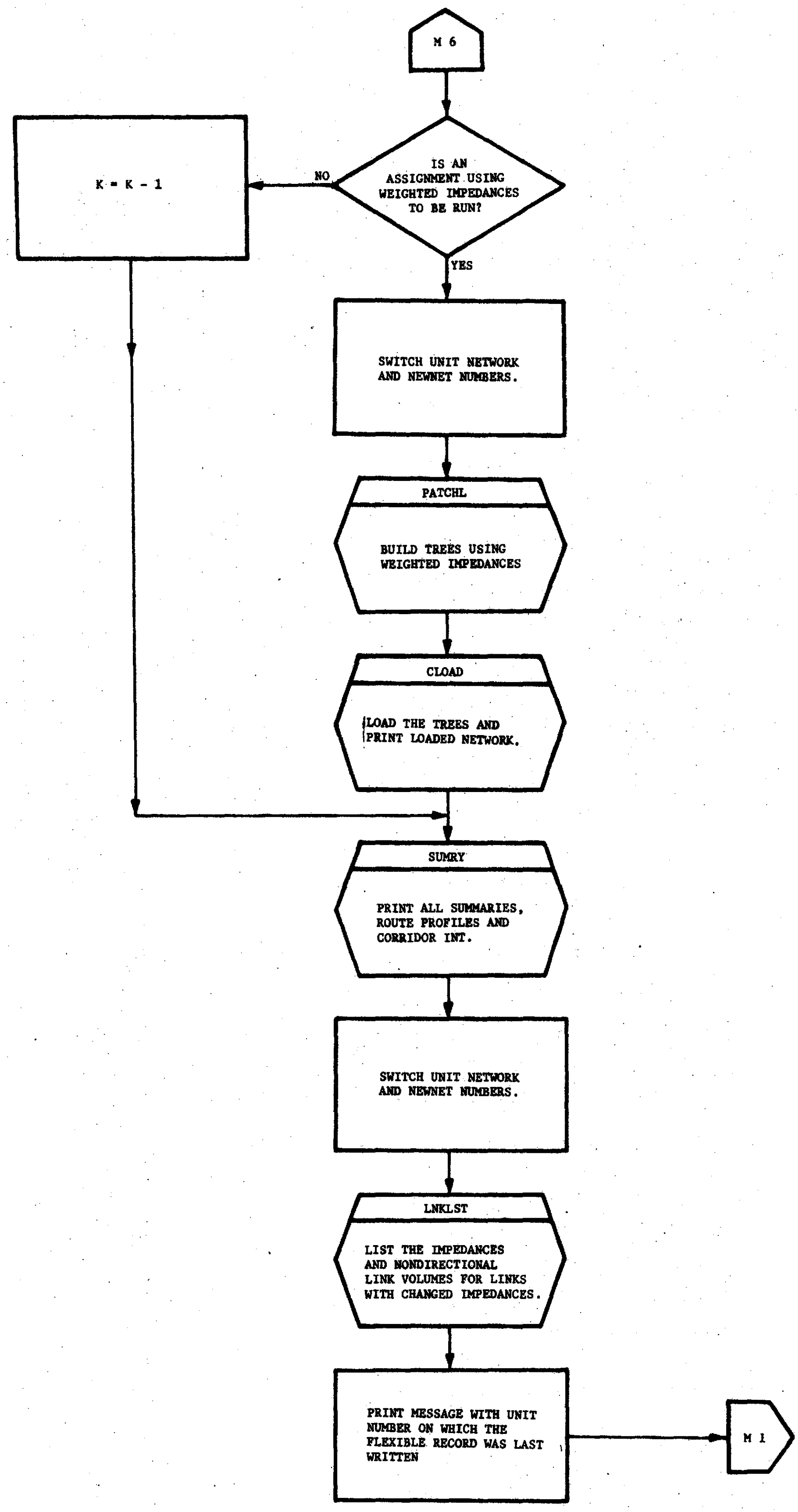

PROGRAM
MAIN
SKIP TO THE TOP OF A PAGE. SET THE EXECUTION TIME TO ZERO. TSUM = 0
CLEAR THE HEADER TO BLANKS
GETDAT
GET TODAY'S DATE FOR THE HEADER.
TIME
SET IX TO TIME OF DAY FOR TIMING.
M 1
TIME
SET IZ TO TIME OF DAY FOR TIMING.
T = (IZ - IX)/6000.
TSUM = TSUM + T.
PRINT T AND TSUM.
IX = IZ
CRD
GET INDEX I WHICH INDICATES CONTROL CARD READ.
M 2

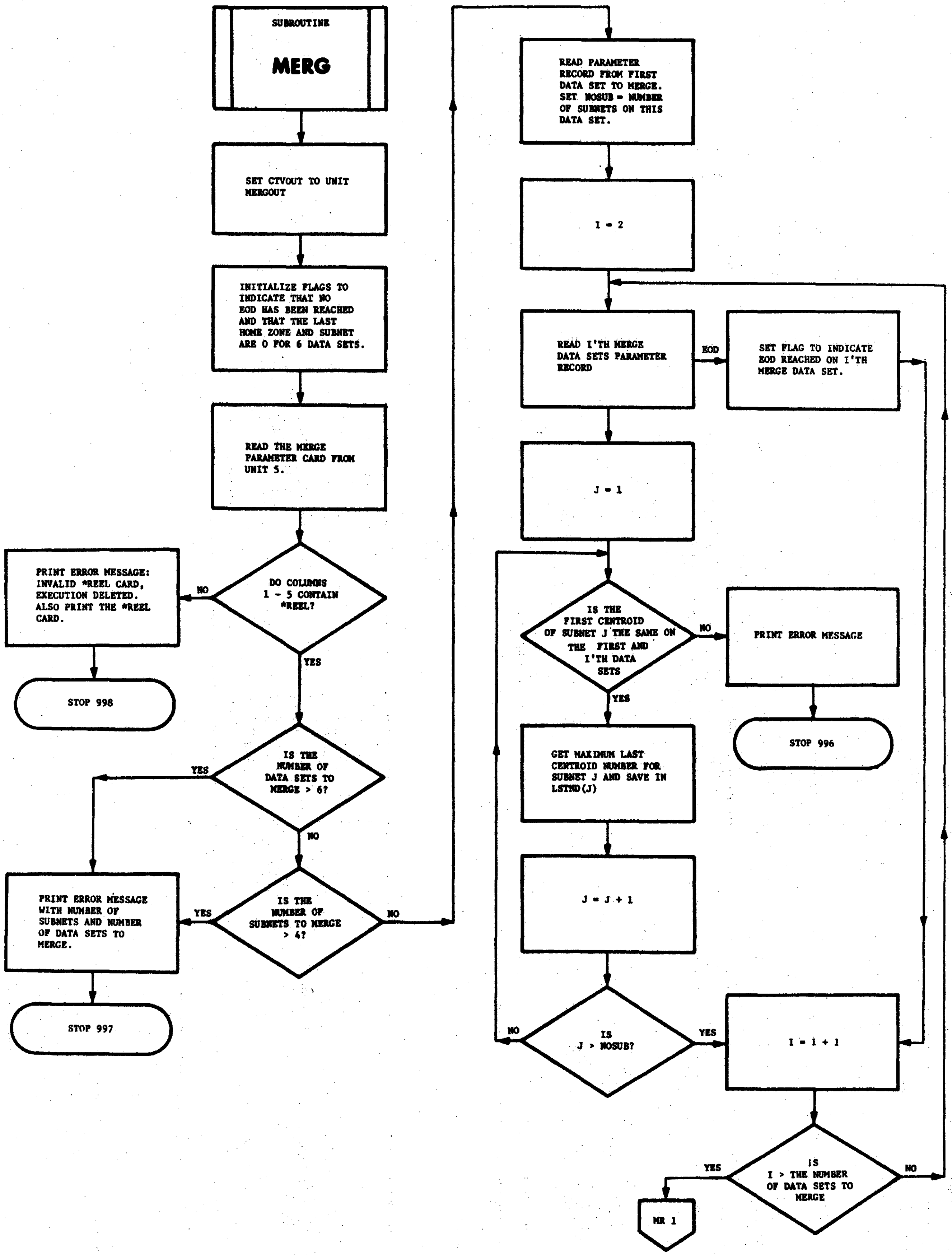

M 2
I = ?
I = 1
PRPNET
"PREPARE NETWORK"
M 1
I = 2
OUTNET
"OUTNET NETWORK"
M 1
I = 3
PRPCTV
"PREPARE TRIP VOLUMES"
M 1
I = 4
OUTRIP
"OUTPUT TRIP VOLUMES"
M 1
I = 5
SUMEND
"SUM TRIP ENDS"
M 1
I = 6 thru 11
M 3
I = 17
M 5
I = 12 thru 16, 18 and 19
M 4
I = 6 "LOAD NETWORK"
RES = .FALSE., ITER = 1.
CLOAD
LOAD THE TREES AND PRINT THE LOADED NETWORK
CNVRG = .TRUE.
SUMRY
PRINT SUMMARIES AND CROSS CLASSIFI- CATIONS.
M 1

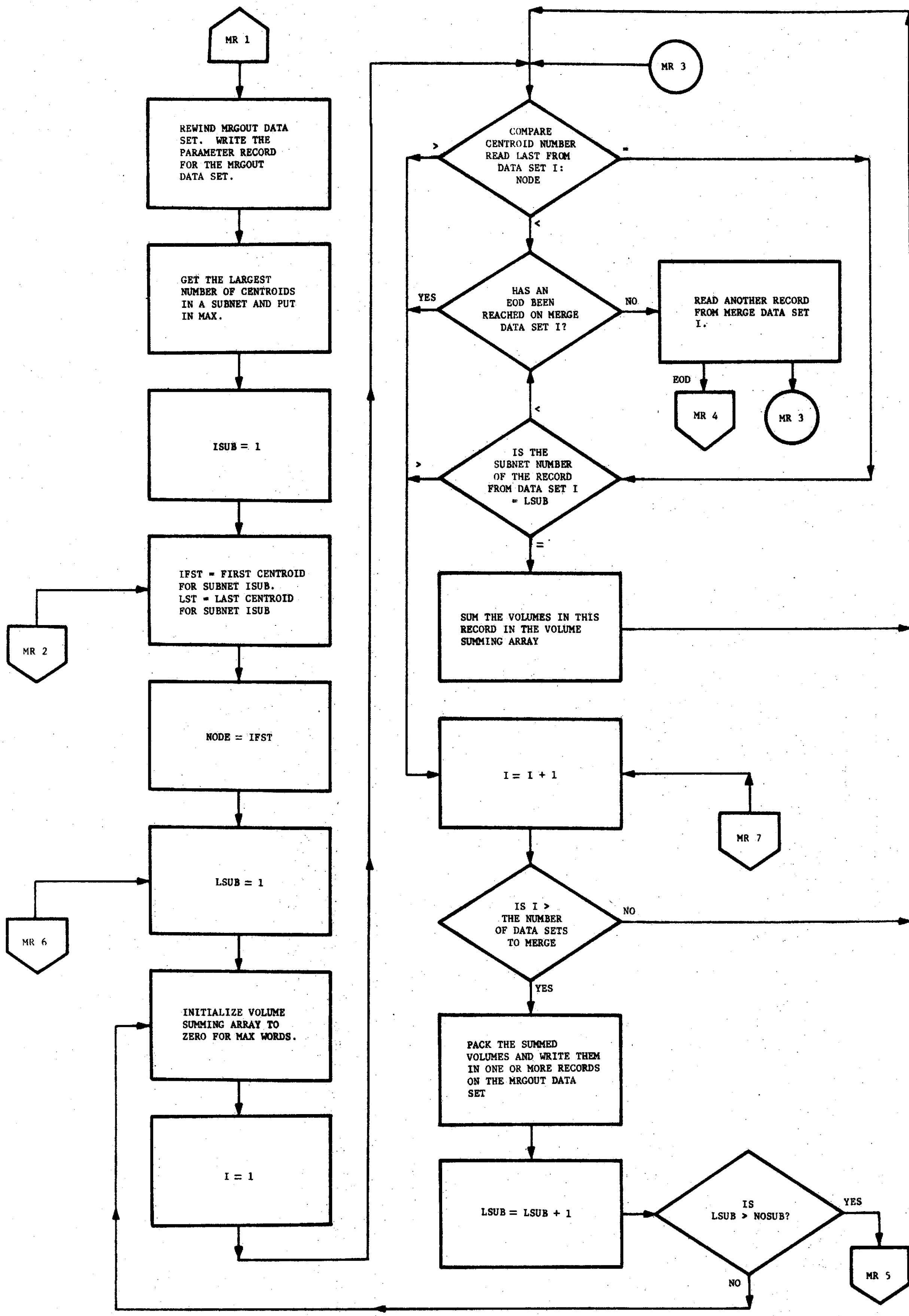

M 3
I = 7
RES = .FALSE.,
ITER = 1
PATHCL
"BUILD TREES"
M 1
I = 8
STOP
I = 9
"LOAD SELECTED LINKS"
RES = .FALSE.,
ITER = 1
CLOAD
USE LDSEL ENTRY POINT
TO LOAD SELECTED LINKS.
CNVRG = .TRUE.
SUMRY
PRINT SUMMARIES AND
CROSS CLASSIFICATIONS.
M 1
I = 10
FRATAR
"FRATAR FORECAST"
M 1
I = 11
MERG
"MERGE"
M 1

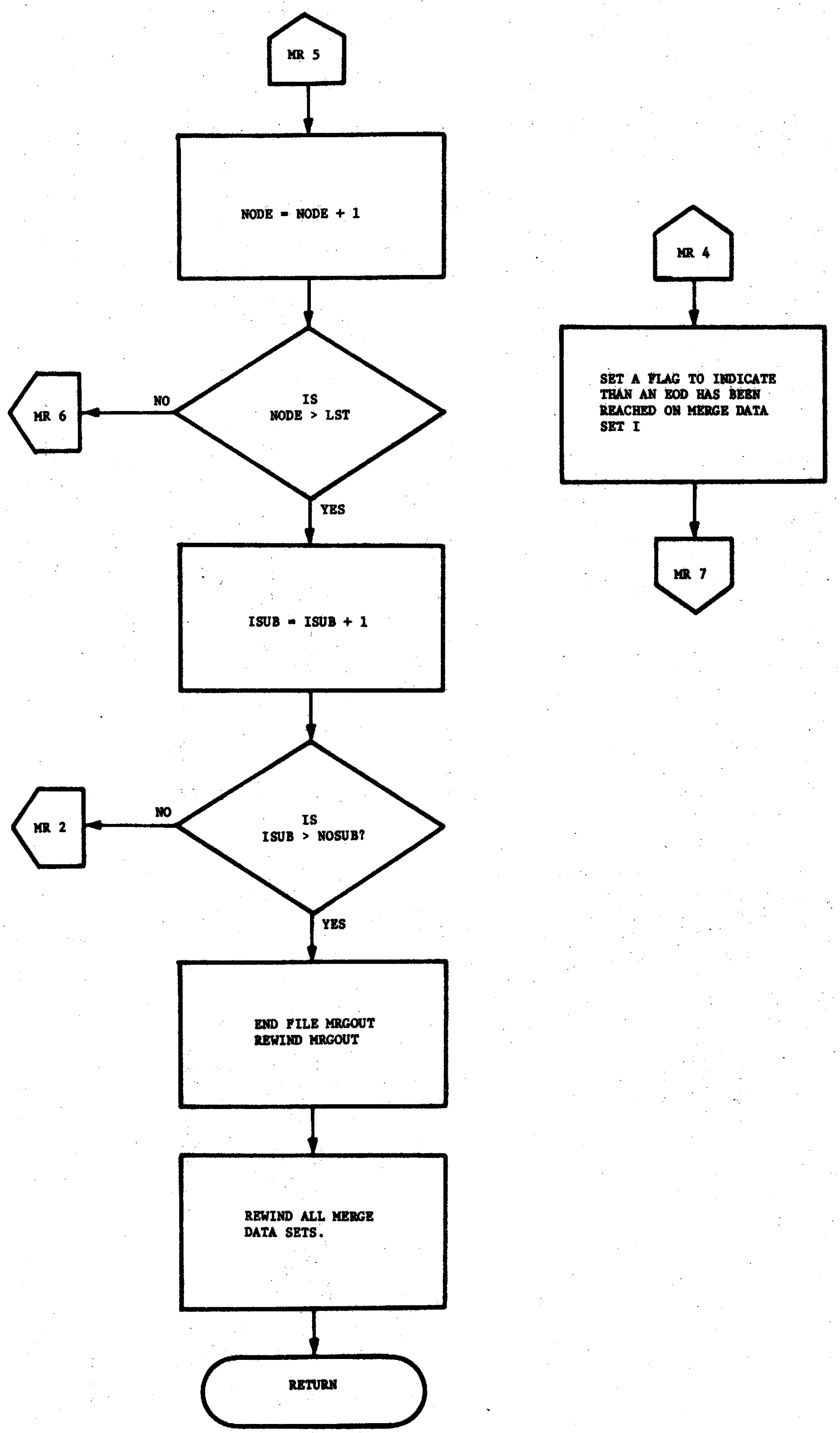

M 4
I = 12
M 1
I = 13
M 1
I = 14
M 1
I = 16
PRPNET
USE REVNET ENTRY POINT. "REVISE NETWORK"
I = 19
RTPLT
"PLOT ROUTE PROFILES"
I = 15
PRPNET
USE ASMNET ENTRY POINT. "ASSEMBLE NETWORK"
I = 18
UPDTNT
"DELETE ASSIGNMENTS"
M 1
M 1
M 1
M 1

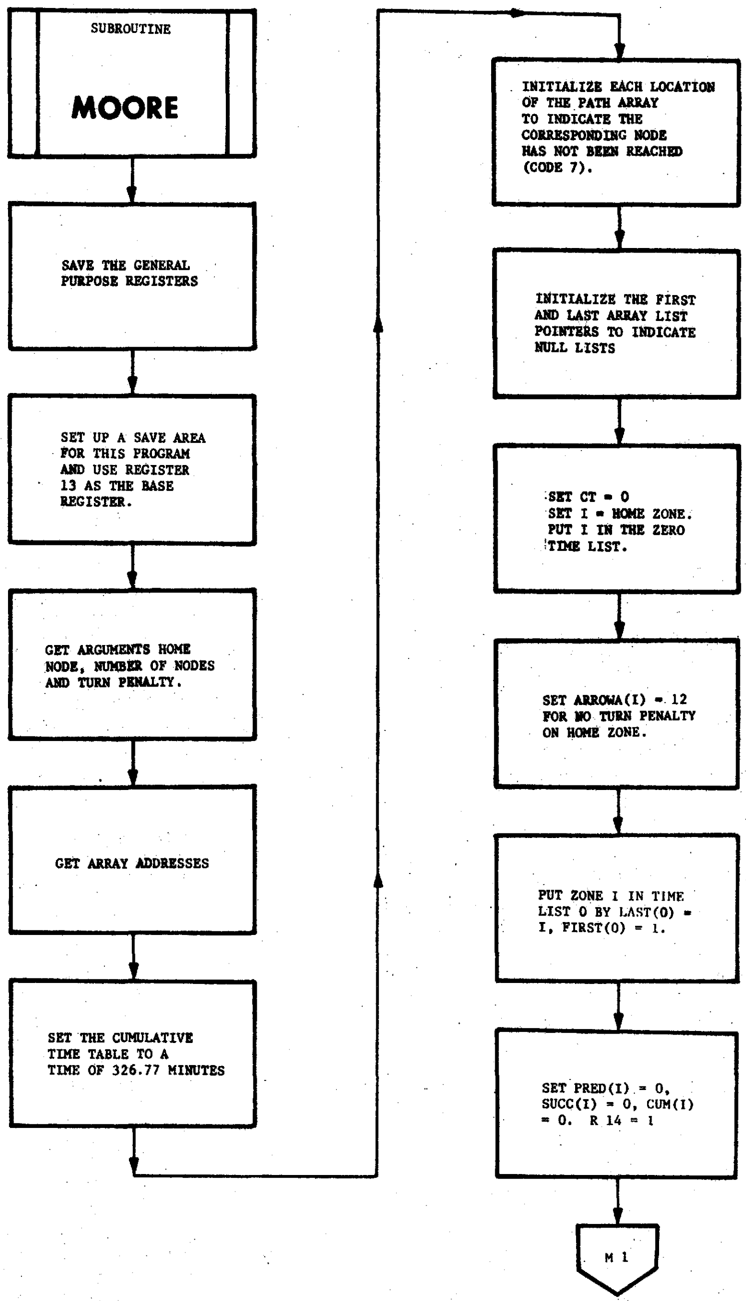

M 5
RES = .TRUE.,
CNVRG = .FALSE.,
ITER = 1
MOVE LITERAL 'ITER'
AND ITER NUMBER TO
END OF HEADER.
PATHCL
BUILD TREES FOR
ITERATION ITER.
CLOAD
LOAD THE NETWORK.
PRINT THE LOADED
NETWORK IF ITER = 1.
SUMRY
PRINT SOME OF THE
SUMMARIES.
SWITCH UNIT NUMBERS OF
UNIT NETWORK AND NEWNET.
K = ITER. ITER = ITER + 1.
CNURG
AND K ≥ 3?
TRUE
FALSE
ITER ≤ 5
YES
NO
MOVE LITERAL 'TOTAL'
TO END OF HEADER.
WGTLD
CALCULATE PERCENT TO
WEIGHT EACH ITERATION
WTSGLN
CALCULATE AND PRINT A
WEIGHTED ASSIGNMENT.
RES = .FALSE.
M 6

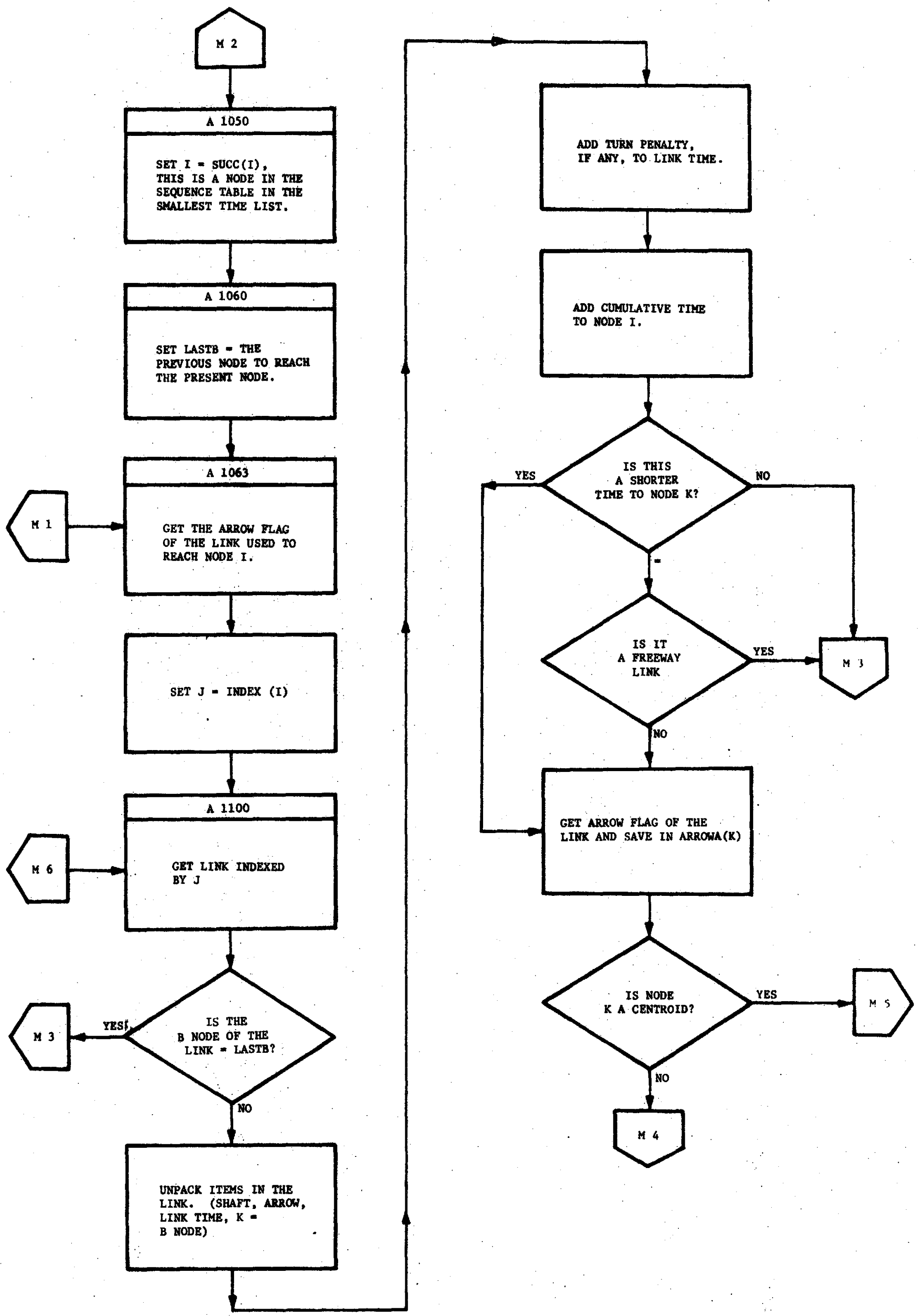

M 6
IS AN ASSIGNMENT USING WEIGHTED IMPEDANCES TO BE RUN?
NO
K = K - 1
YES
SWITCH UNIT NETWORK AND NEWNET NUMBERS.
PATCHL
BUILD TREES USING WEIGHTED IMPEDANCES
CLOAD
LOAD THE TREES AND PRINT LOADED NETWORK.
SUMRY
PRINT ALL SUMMARIES, ROUTE PROFILES AND CORRIDOR INT.
SWITCH UNIT NETWORK AND NEWNET NUMBERS.
LNKLST
LIST THE IMPEDANCES AND NONDIRECTIONAL LINK VOLUMES FOR LINKS WITH CHANGED IMPEDANCES.
PRINT MESSAGE WITH UNIT NUMBER ON WHICH THE FLEXIBLE RECORD WAS LAST WRITTEN
M 1

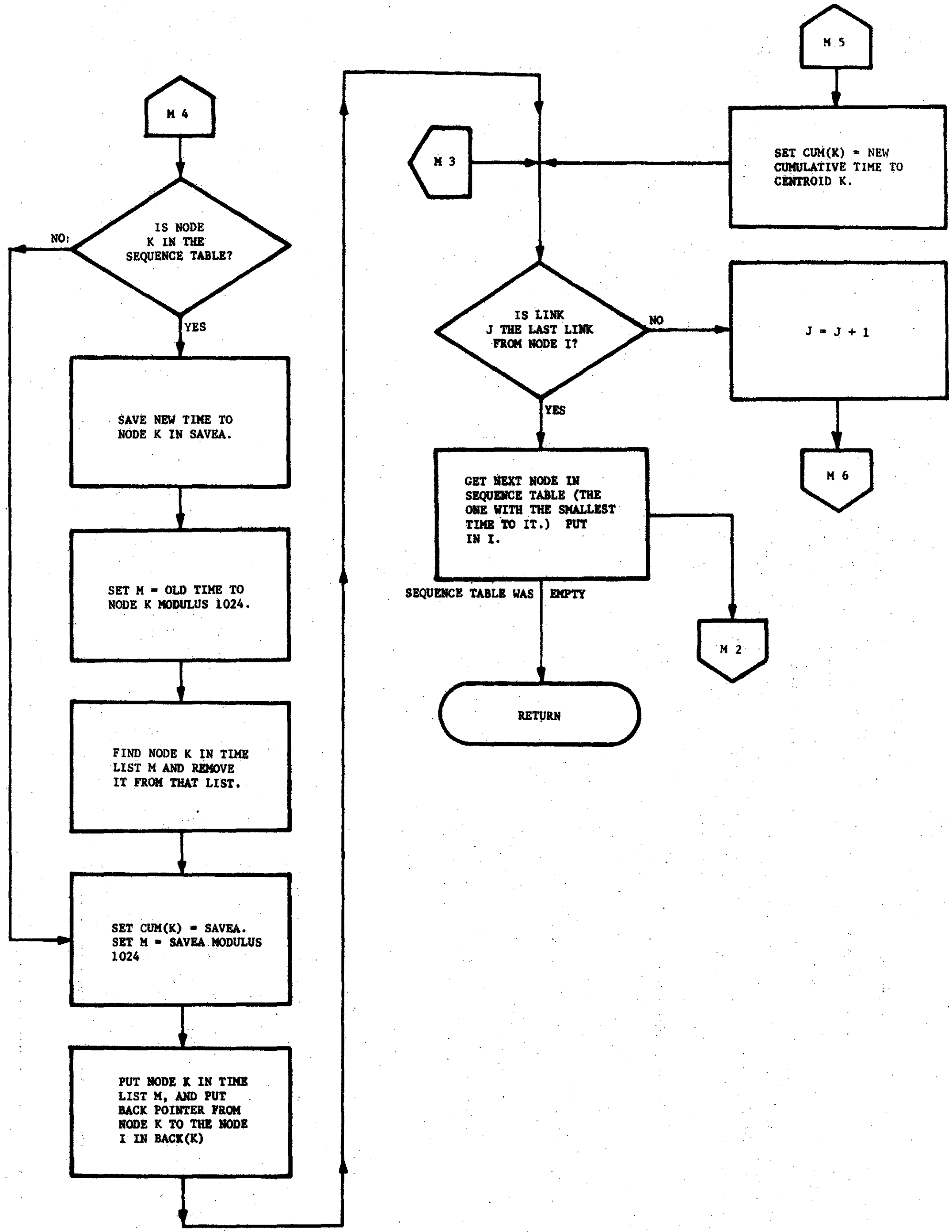

SUBROUTINE
MERG
SET CTVOUT TO UNIT MERGOUT
INITIALIZE FLAGS TO INDICATE THAT NO EOD HAS BEEN REACHED AND THAT THE LAST HOME ZONE AND SUBNET ARE 0 FOR 6 DATA SETS.
READ THE MERGE PARAMETER CARD FROM UNIT 5.
DO COLUMNS 1 - 5 CONTAIN *REEL?
NO
PRINT ERROR MESSAGE: INVALID *REEL CARD, EXECUTION DELETED. ALSO PRINT THE *REEL CARD.
STOP 998
YES
IS THE NUMBER OF DATA SETS TO MERGE > 6?
YES
NO
IS THE NUMBER OF SUBNETS TO MERGE > 4?
YES
PRINT ERROR MESSAGE WITH NUMBER OF SUBNETS AND NUMBER OF DATA SETS TO MERGE.
STOP 997
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 LSTND(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

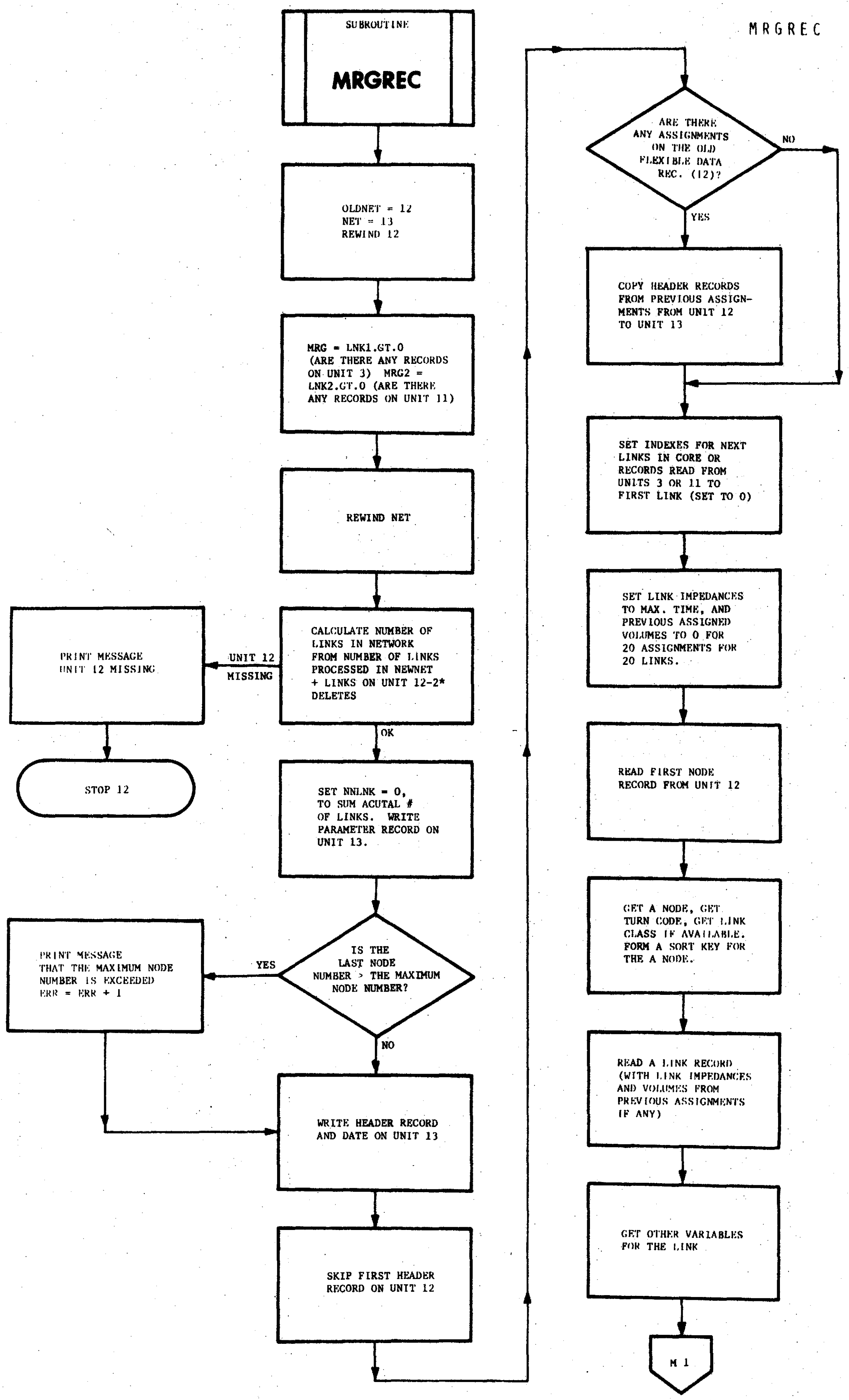

MR 1
REWIND MRGOUT DATA SET. WRITE THE PARAMETER RECORD FOR THE MRGOUT DATA SET.
GET THE LARGEST NUMBER OF CENTROIDS IN A SUBNET AND PUT IN MAX.
ISUB = 1
IFST = FIRST CENTROID FOR SUBNET ISUB. LST = LAST CENTROID FOR SUBNET ISUB
MR 2
NODE = IFST
LSUB = 1
MR 6
INITIALIZE VOLUME SUMMING ARRAY TO ZERO FOR 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

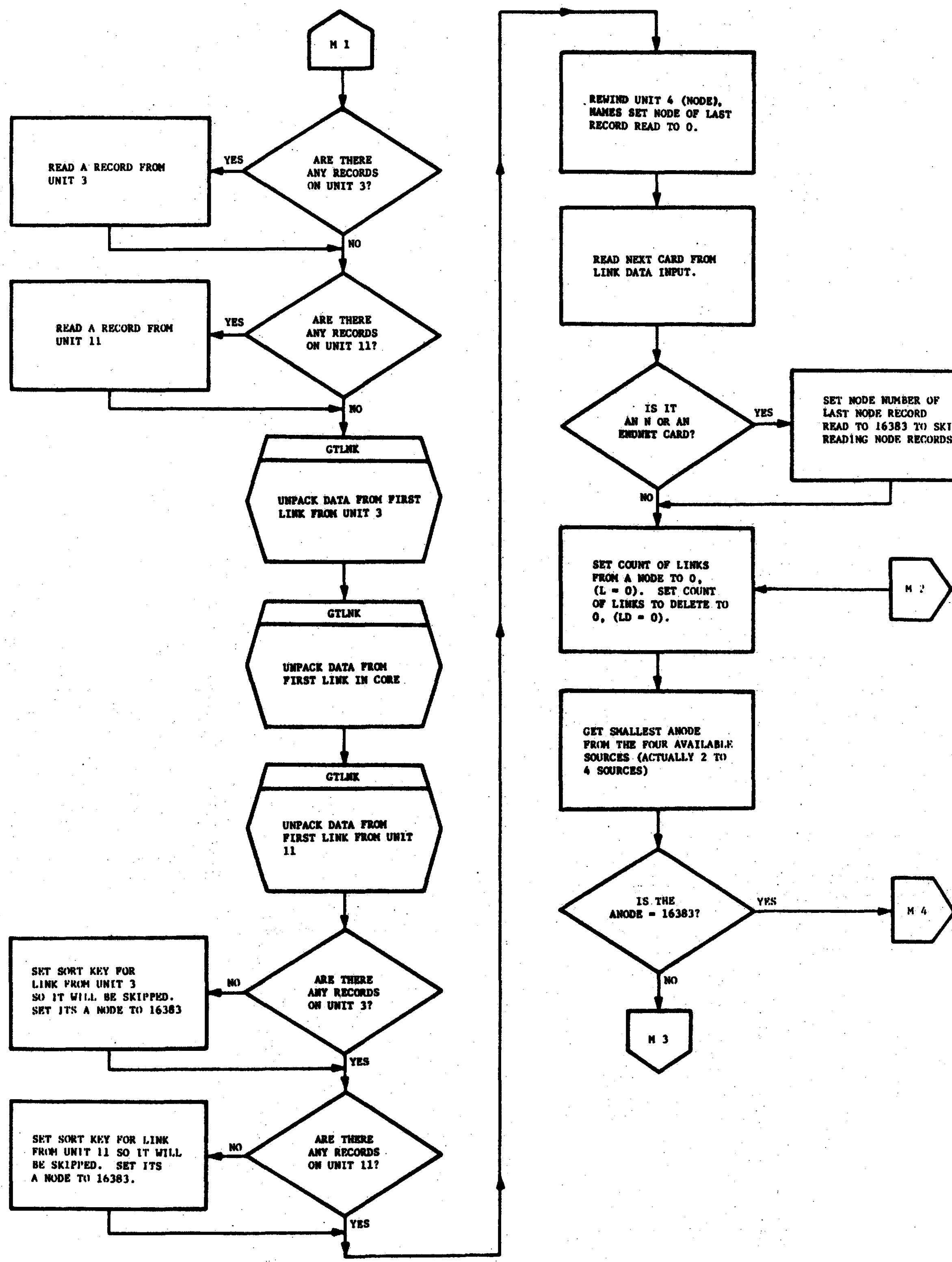

MR 5
NODE = NODE + 1
IS
NODE > LST
NO
MR 6
YES
ISUB = ISUB + 1
IS
ISUB > NOSUB?
NO
MR 2
YES
END FILE MRGOUT
REWIND MRGOUT
REWIND ALL MERGE
DATA SETS.
RETURN
MR 4
SET A FLAG TO INDICATE
THAN AN EOD HAS BEEN
REACHED ON MERGE DATA
SET I
MR 7

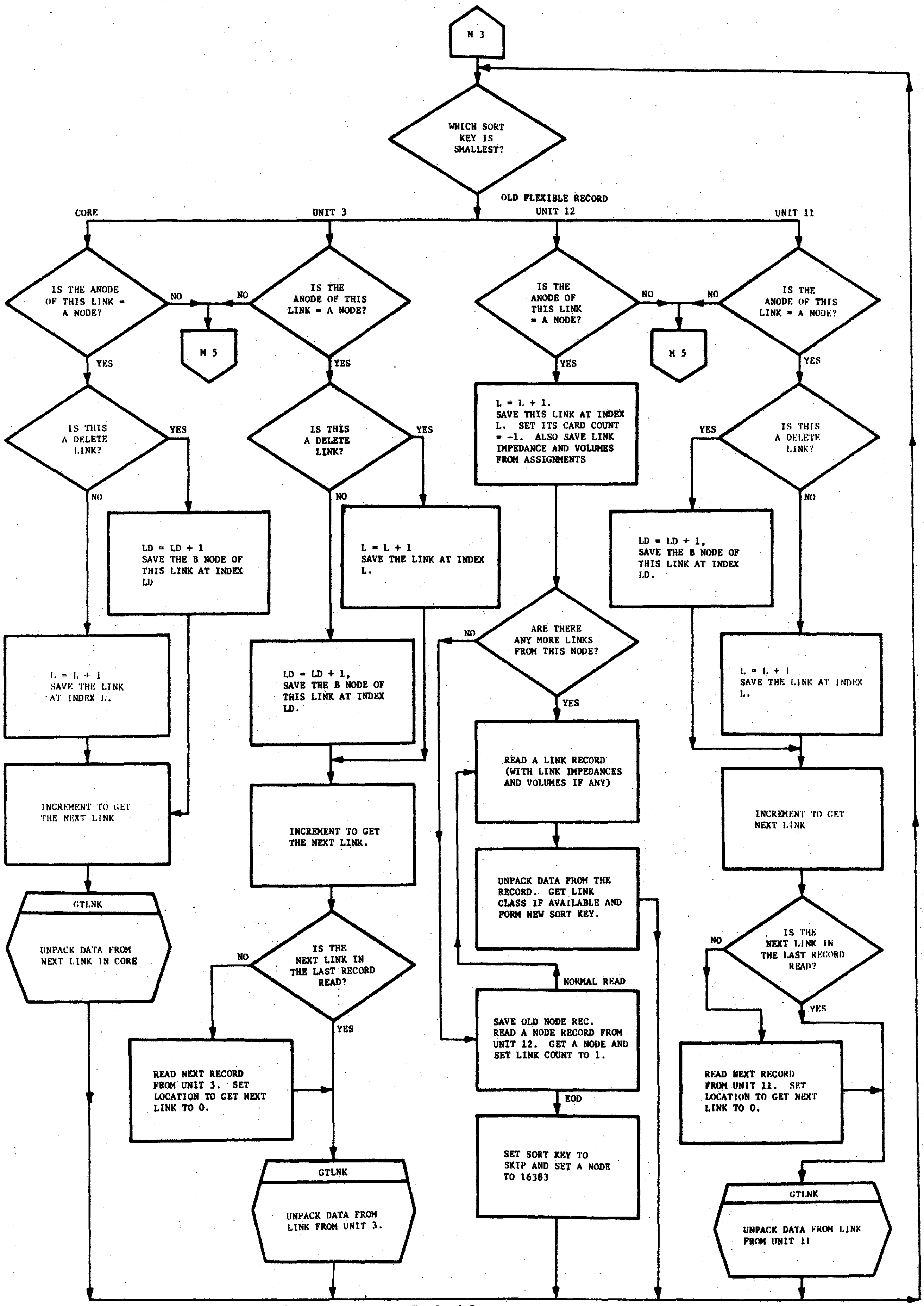

SUBROUTINE
MOORE
SAVE THE GENERAL PURPOSE REGISTERS
SET UP A SAVE AREA FOR THIS PROGRAM AND USE REGISTER 13 AS THE BASE REGISTER.
GET ARGUMENTS HOME NODE, NUMBER OF NODES AND TURN PENALTY.
GET ARRAY ADDRESSES
SET THE CUMULATIVE TIME TABLE TO A TIME OF 326.77 MINUTES
INITIALIZE EACH LOCATION OF THE PATH ARRAY TO INDICATE THE CORRESPONDING NODE HAS NOT BEEN REACHED (CODE 7).
INITIALIZE THE FIRST AND LAST ARRAY LIST POINTERS TO INDICATE NULL LISTS
SET CT = 0 SET I = HOME ZONE. PUT I IN THE ZERO TIME LIST.
SET ARROWA(I) = 12 FOR NO TURN PENALTY ON HOME ZONE.
PUT ZONE I IN TIME LIST 0 BY LAST(0) = I, FIRST(0) = 1.
SET PRED(I) = 0, SUCC(I) = 0, CUM(I) = 0. R 14 = 1
M 1

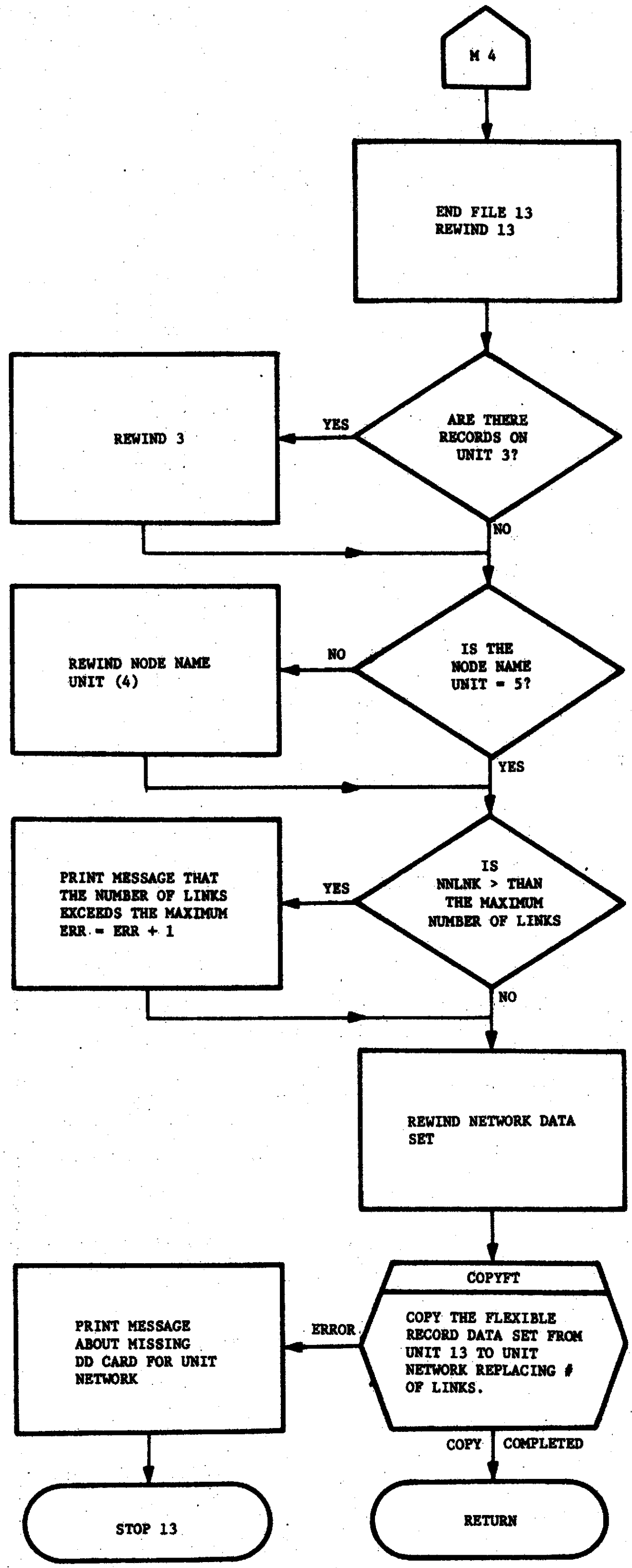

M 2
A 1050
SET I = SUCC(I), THIS IS A NODE IN THE SEQUENCE TABLE IN THE SMALLEST TIME LIST.
A 1060
SET LASTB = THE PREVIOUS NODE TO REACH THE PRESENT NODE.
M 1
A 1063
GET THE ARROW FLAG OF THE LINK USED TO REACH NODE I.
SET J = INDEX (I)
M 6
A 1100
GET LINK INDEXED BY J
M 3
YES
IS THE B NODE OF THE LINK = LASTB?
NO
UNPACK ITEMS IN THE LINK. (SHAFT, ARROW, LINK TIME, K = B NODE)
ADD TURN PENALTY, IF ANY, TO LINK TIME.
ADD CUMULATIVE TIME TO NODE I.
YES
IS THIS A SHORTER TIME TO NODE K?
NO
IS IT A FREEWAY LINK
YES
M 3
NO
GET ARROW FLAG OF THE LINK AND SAVE IN ARROWA(K)
IS NODE K A CENTROID?
YES
M 5
NO
M 4

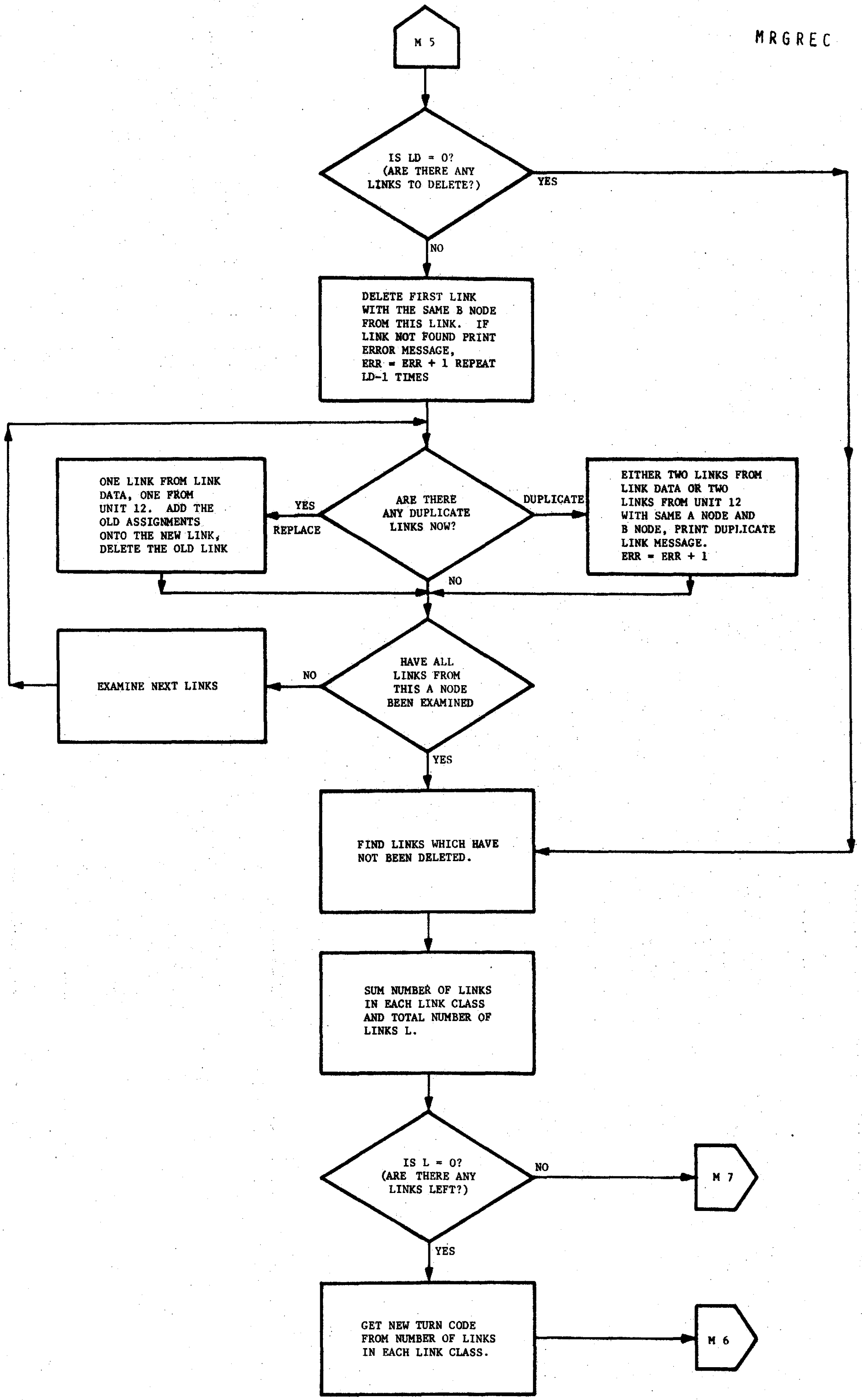

M 4
IS NODE K IN THE SEQUENCE TABLE?
NO
YES
SAVE NEW TIME TO NODE K IN SAVEA.
SET M = OLD TIME TO NODE K MODULUS 1024.
FIND NODE K IN TIME LIST M AND REMOVE IT FROM THAT LIST.
SET CUM(K) = SAVEA. SET M = SAVEA MODULUS 1024
PUT NODE K IN TIME LIST M, AND PUT BACK POINTER FROM NODE K TO THE NODE I IN BACK(K)
M 5
SET CUM(K) = NEW CUMULATIVE TIME TO CENTROID K.
M 3
IS LINK J THE LAST LINK FROM NODE I?
NO
J = J + 1
M 6
YES
GET NEXT NODE IN SEQUENCE TABLE (THE ONE WITH THE SMALLEST TIME TO IT.) PUT IN I.
SEQUENCE TABLE WAS EMPTY
M 2
RETURN

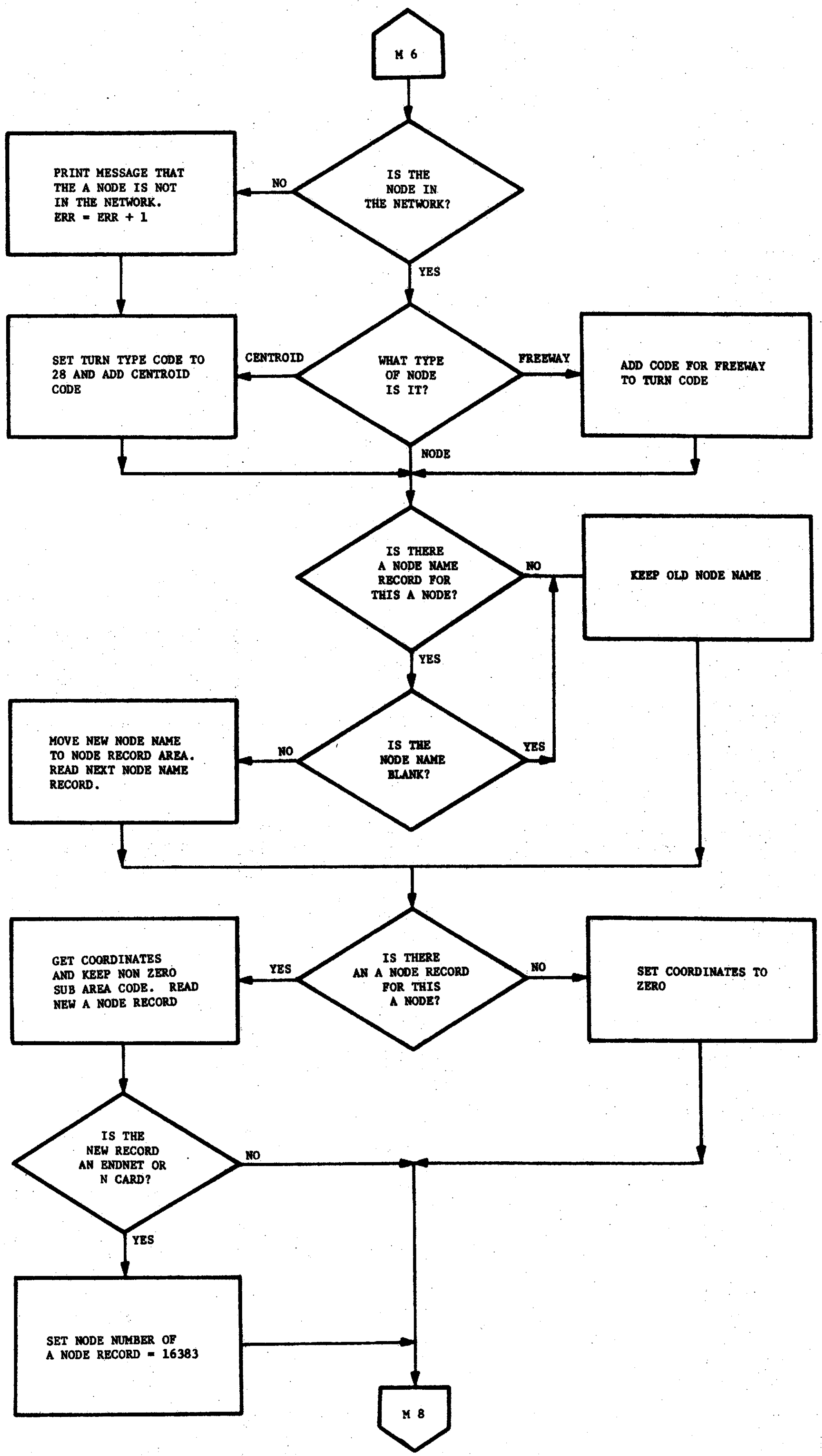

SUBROUTINE
MRGREC
OLDNET = 12
NET = 13
REWIND 12
MRG = LNK1.GT.0
(ARE THERE ANY RECORDS
ON UNIT 3) MRG2 =
LNK2.GT.0 (ARE THERE
ANY RECORDS ON UNIT 11)
REWIND NET
CALCULATE NUMBER OF
LINKS IN NETWORK
FROM NUMBER OF LINKS
PROCESSED IN NEWNET
+ LINKS ON UNIT 12-2*
DELETES
UNIT 12
MISSING
PRINT MESSAGE
UNIT 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

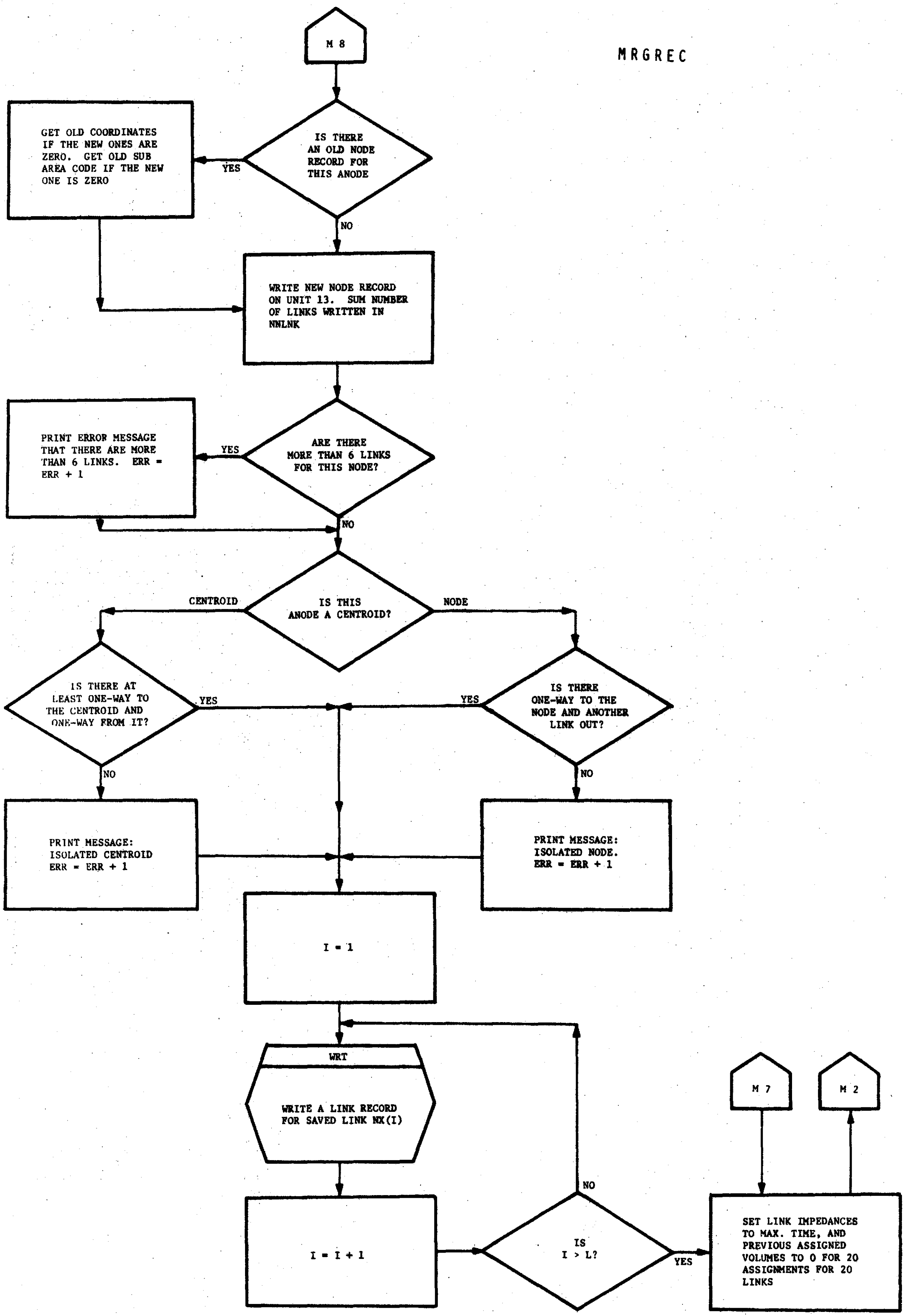

M 1
ARE THERE ANY RECORDS ON UNIT 3?
YES
READ A RECORD FROM UNIT 3
NO
ARE THERE ANY RECORDS ON UNIT 11?
YES
READ A RECORD FROM UNIT 11
NO
GTLNK
UNPACK DATA FROM FIRST LINK FROM UNIT 3
GTLNK
UNPACK DATA FROM FIRST LINK IN CORE
GTLNK
UNPACK DATA FROM FIRST LINK FROM UNIT 11
ARE THERE ANY RECORDS ON UNIT 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

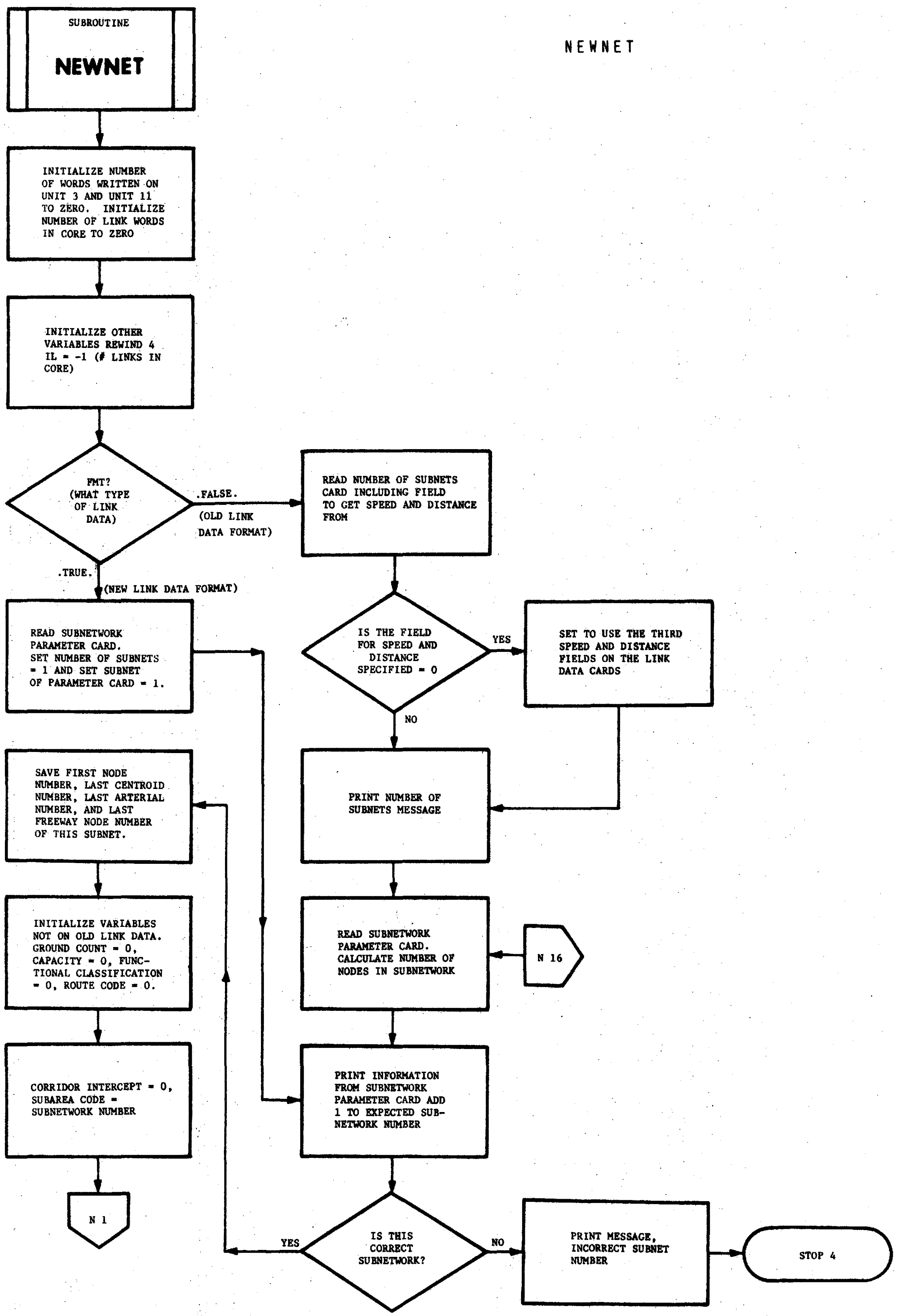

M 3
WHICH SORT KEY IS SMALLEST?
CORE
UNIT 3
OLD FLEXIBLE RECORD UNIT 12
UNIT 11
IS THE ANODE OF THIS LINK = A NODE?
NO
M 5
YES
IS THIS A DELETE LINK?
LD = LD + 1 SAVE THE B NODE OF THIS LINK AT INDEX LD
L = L + 1 SAVE THE LINK AT INDEX L.
INCREMENT TO GET THE NEXT LINK
GTLNK
UNPACK DATA FROM NEXT LINK IN CORE
IS THE ANODE OF THIS LINK = A NODE?
IS THIS A DELETE LINK?
L = L + 1 SAVE THE LINK AT INDEX L.
LD = LD + 1, SAVE THE B NODE OF THIS LINK AT INDEX LD.
INCREMENT TO GET THE NEXT LINK.
IS THE NEXT LINK IN THE LAST RECORD READ?
READ NEXT RECORD FROM UNIT 3. SET LOCATION TO GET NEXT LINK TO 0.
GTLNK
UNPACK DATA FROM LINK FROM UNIT 3.
IS THE ANODE OF THIS LINK = A NODE?
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
IS THE ANODE OF THIS LINK = A NODE?
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.
INCREMENT TO GET NEXT LINK
IS THE NEXT LINK IN THE LAST RECORD READ?
READ NEXT RECORD FROM UNIT 11. SET LOCATION TO GET NEXT LINK TO 0.
GTLNK
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
STOP 13
COPY COMPLETED
RETURN

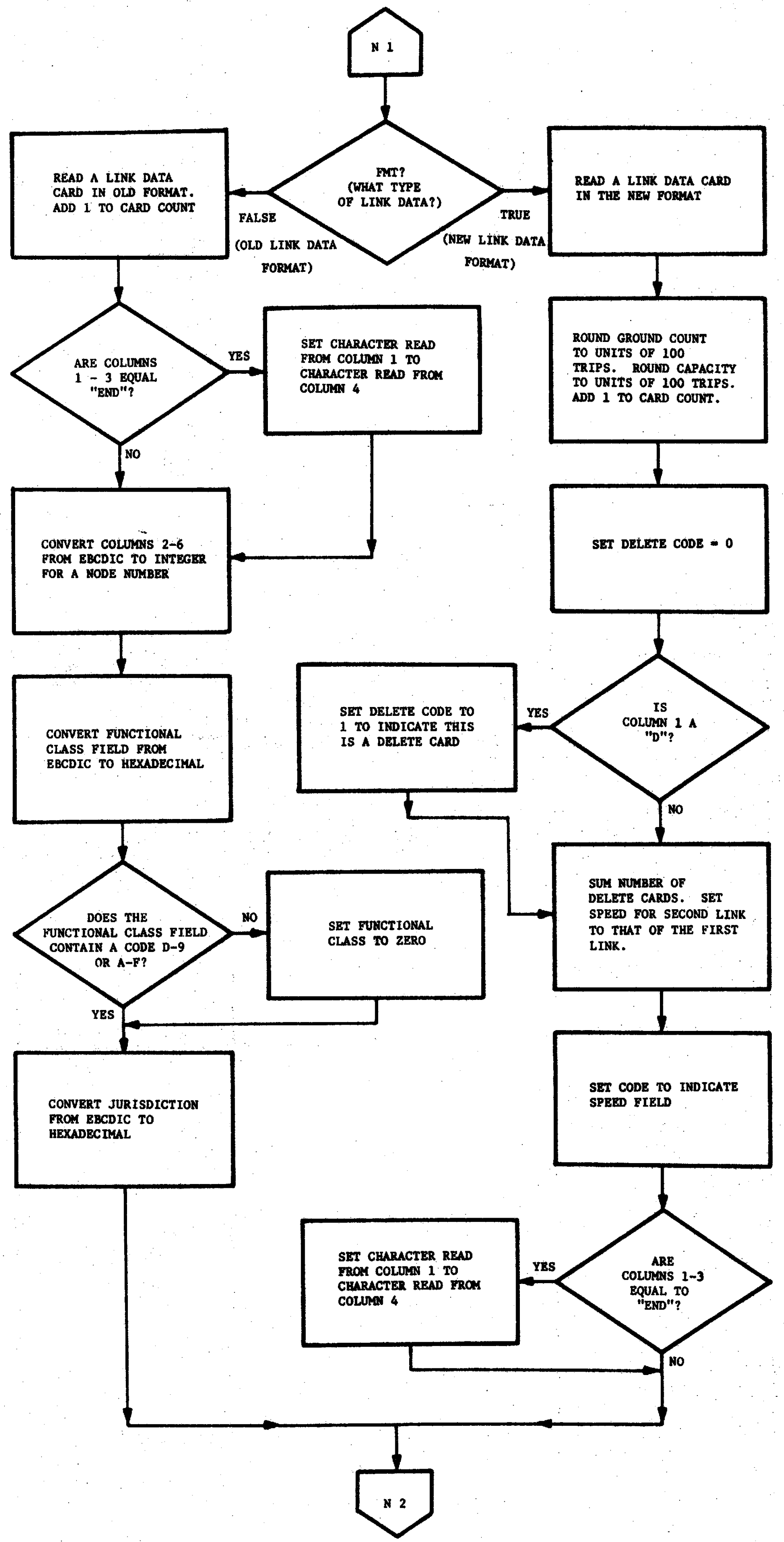

M 5
IS LD = 0?
(ARE THERE ANY
LINKS TO DELETE?)
YES
NO
DELETE FIRST LINK
WITH THE SAME B NODE
FROM THIS LINK. IF
LINK NOT FOUND PRINT
ERROR MESSAGE,
ERR = ERR + 1 REPEAT
LD-1 TIMES
ONE LINK FROM LINK
DATA, ONE FROM
UNIT 12. ADD THE
OLD ASSIGNMENTS
ONTO THE NEW LINK,
DELETE THE OLD LINK
YES
REPLACE
ARE 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

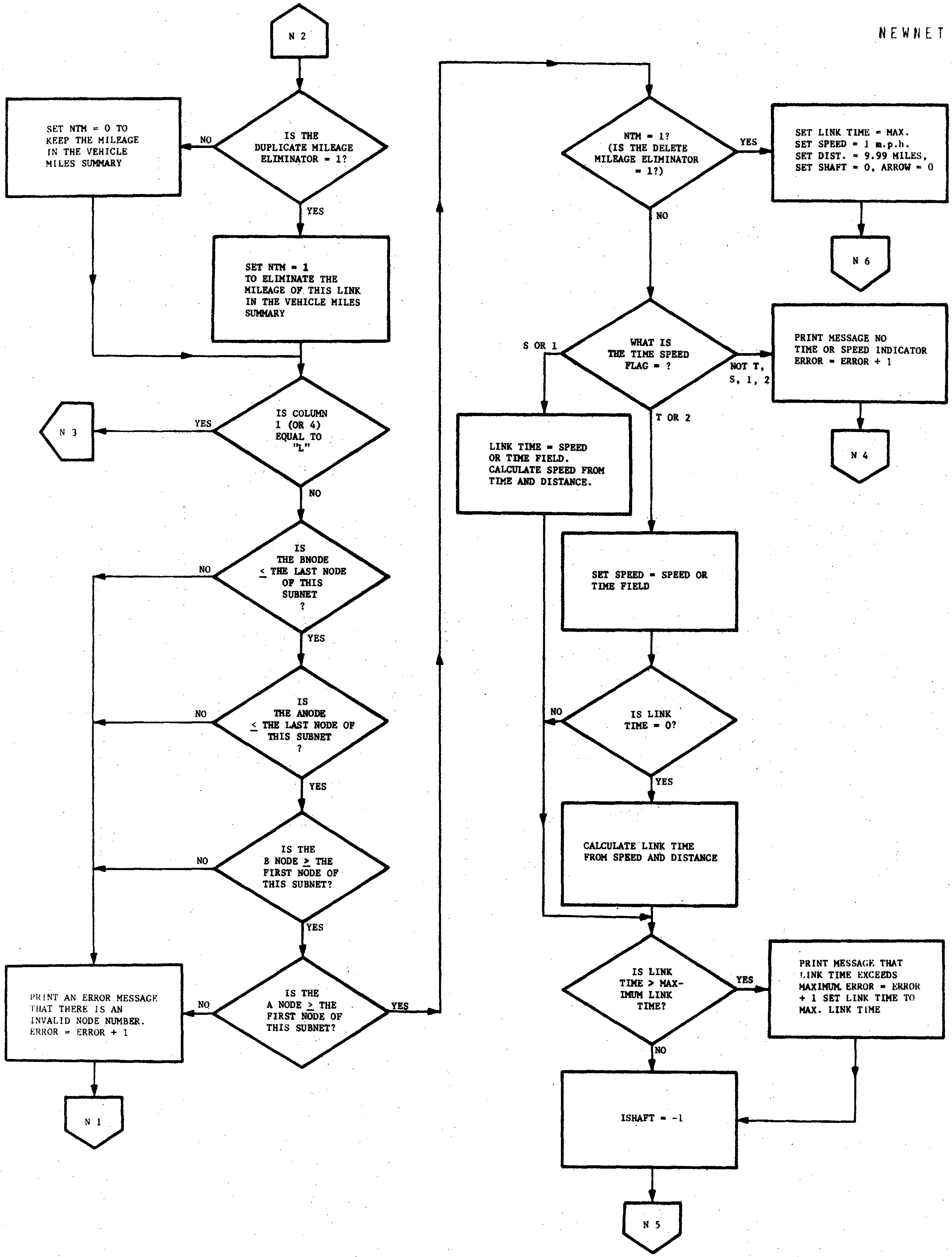

M 6
IS THE NODE IN THE NETWORK?
NO
PRINT MESSAGE THAT THE A NODE IS NOT IN THE NETWORK. ERR = ERR + 1
YES
WHAT TYPE OF NODE IS IT?
CENTROID
SET TURN TYPE CODE TO 28 AND ADD CENTROID CODE
FREEWAY
ADD CODE FOR FREEWAY TO TURN CODE
NODE
IS THERE A NODE NAME RECORD FOR THIS A NODE?
NO
KEEP OLD NODE NAME
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

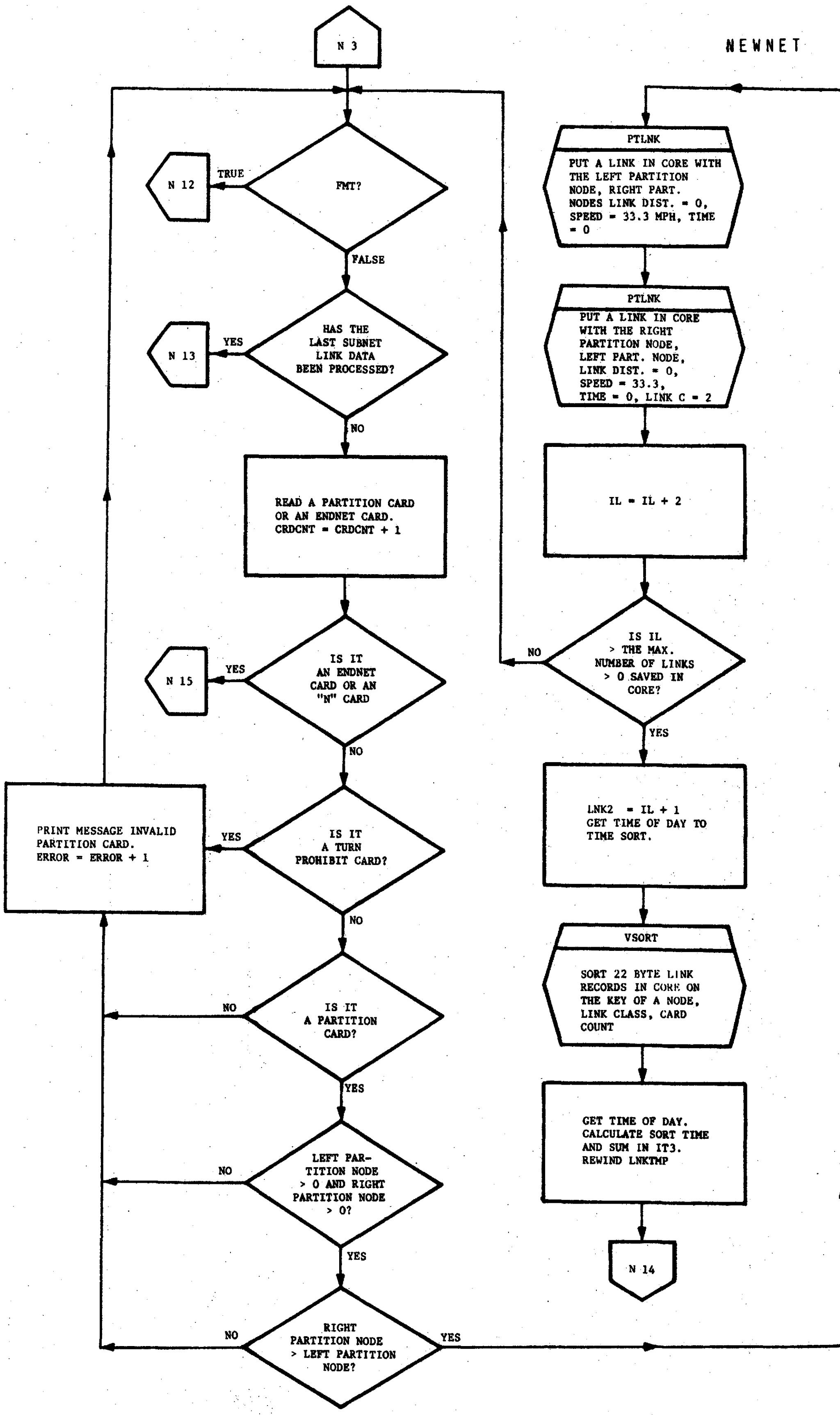

M 8
IS THERE AN OLD NODE RECORD FOR THIS ANODE
YES
NO
GET OLD COORDINATES IF THE NEW ONES ARE ZERO. GET OLD SUB AREA CODE IF THE NEW ONE IS ZERO
WRITE NEW NODE RECORD ON UNIT 13. SUM NUMBER OF LINKS WRITTEN IN NNLNK
ARE THERE MORE THAN 6 LINKS FOR THIS NODE?
YES
NO
PRINT ERROR MESSAGE THAT THERE ARE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

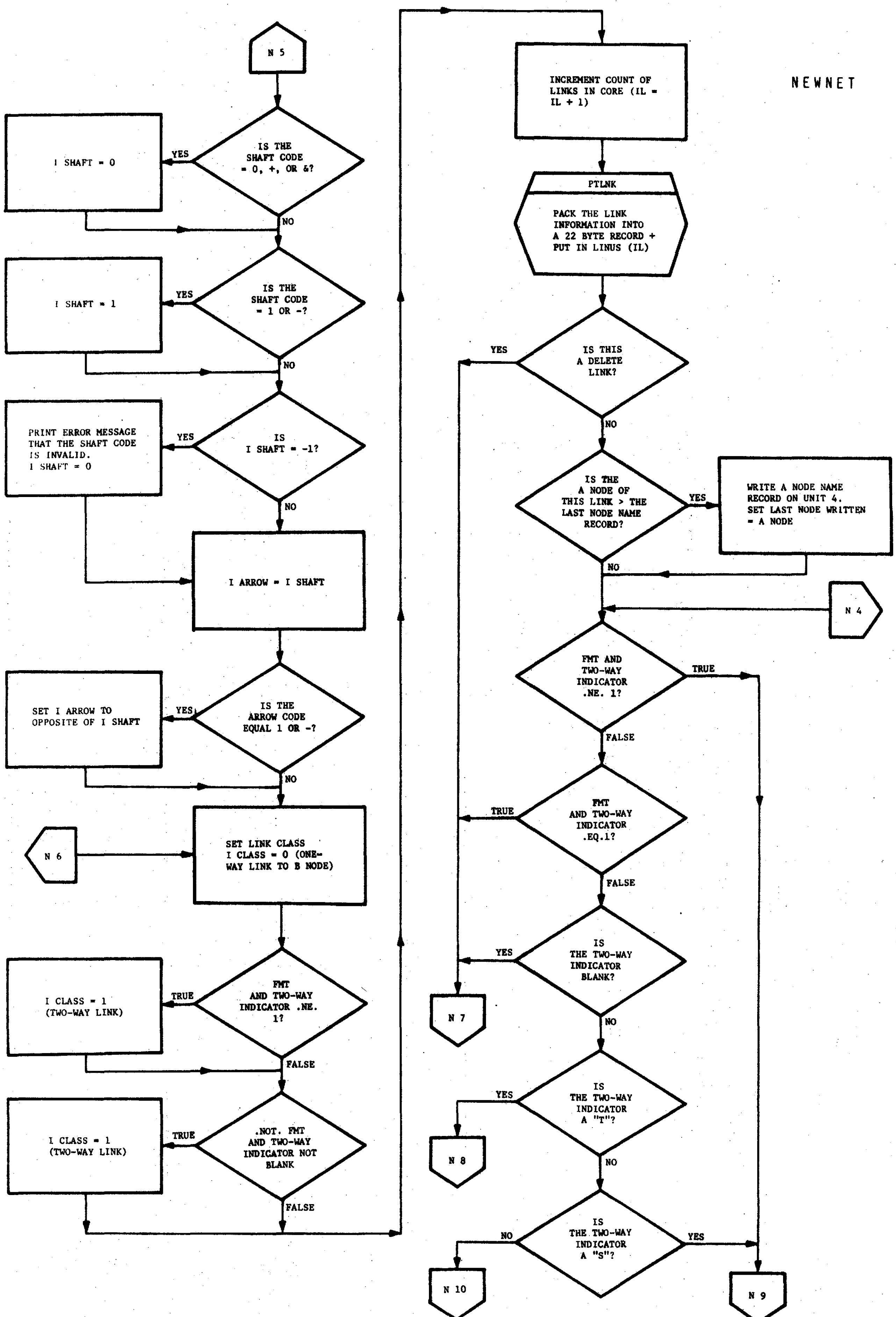


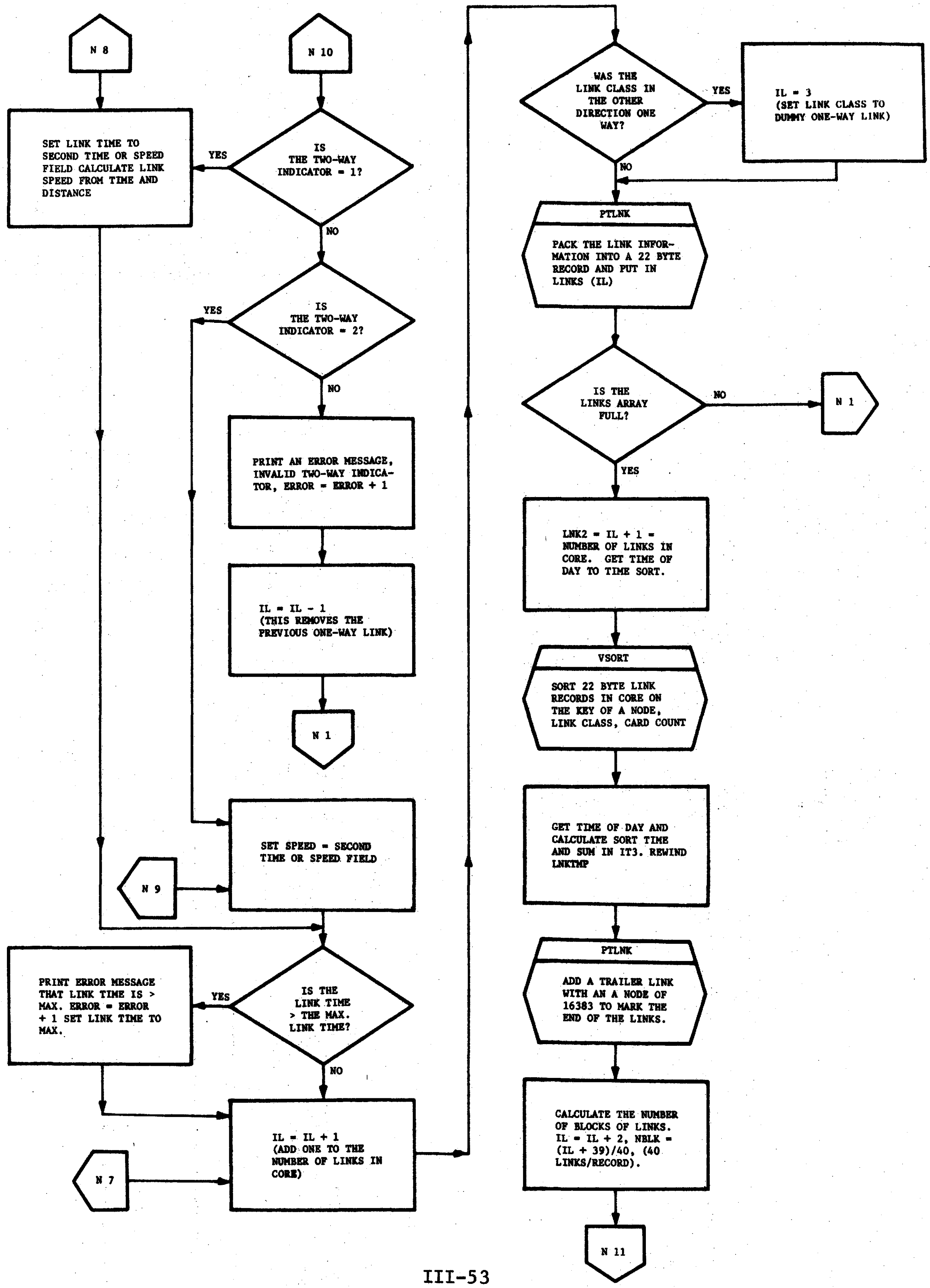
SUBROUTINE
NEWNET
INITIALIZE NUMBER OF WORDS WRITTEN ON UNIT 3 AND UNIT 11 TO ZERO. INITIALIZE NUMBER OF LINK WORDS IN CORE TO ZERO
INITIALIZE OTHER VARIABLES REWIND 4 IL = -1 (# LINKS IN CORE)
FMT? (WHAT TYPE OF LINK DATA)
.FALSE.
(OLD LINK DATA FORMAT)
.TRUE.
(NEW LINK DATA FORMAT)
READ NUMBER OF SUBNETS CARD INCLUDING FIELD TO GET SPEED AND DISTANCE FROM
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

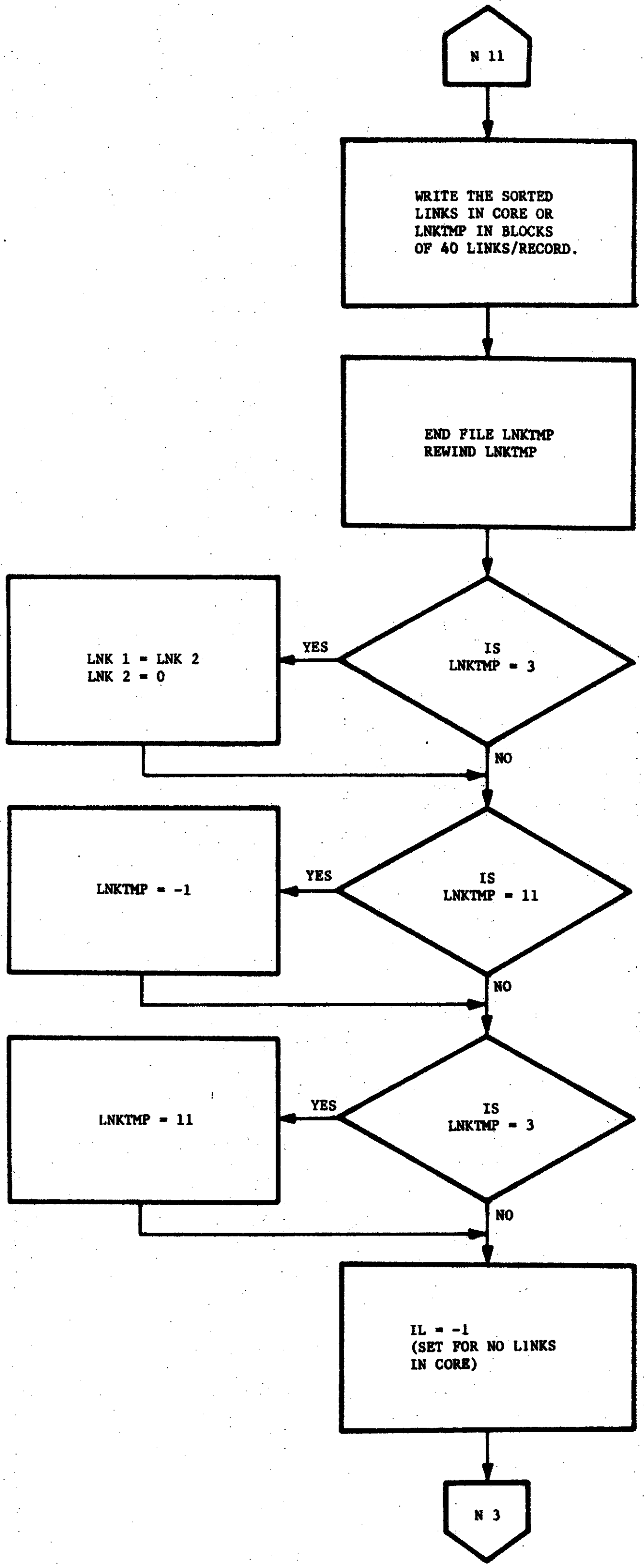

N 1
FMT? (WHAT TYPE OF LINK DATA?)
FALSE
(OLD LINK DATA FORMAT)
TRUE
(NEW LINK DATA FORMAT)
READ A LINK DATA CARD IN OLD FORMAT. ADD 1 TO CARD COUNT
READ A LINK DATA CARD IN THE NEW FORMAT
ARE COLUMNS 1 - 3 EQUAL "END"?
YES
NO
SET CHARACTER READ FROM COLUMN 1 TO CHARACTER READ FROM COLUMN 4
ROUND GROUND 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
SUM NUMBER OF DELETE CARDS. SET SPEED FOR SECOND LINK TO THAT OF THE FIRST LINK.
DOES THE FUNCTIONAL CLASS FIELD CONTAIN A CODE D-9 OR A-F?
NO
SET FUNCTIONAL CLASS TO ZERO
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

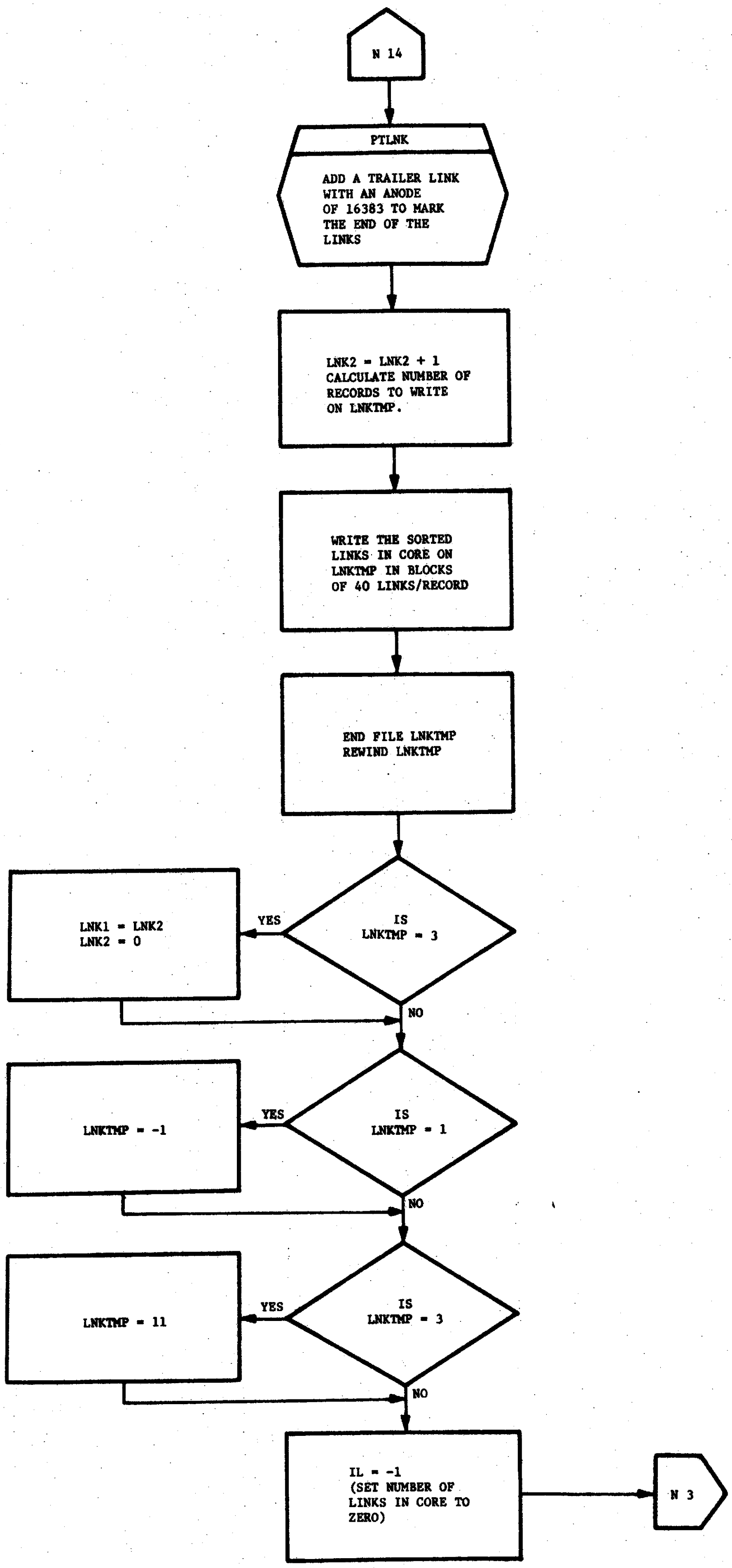

N 2
IS THE DUPLICATE MILEAGE ELIMINATOR = 1?
NO
SET NTM = 0 TO KEEP THE MILEAGE IN THE VEHICLE MILES SUMMARY
YES
SET NTM = 1 TO ELIMINATE THE MILEAGE OF THIS LINK IN THE VEHICLE MILES SUMMARY
IS COLUMN 1 (OR 4) EQUAL TO "L"
YES
N 3
NO
IS THE BNODE ≤ THE LAST NODE OF THIS SUBNET ?
NO
YES
IS THE ANODE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


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

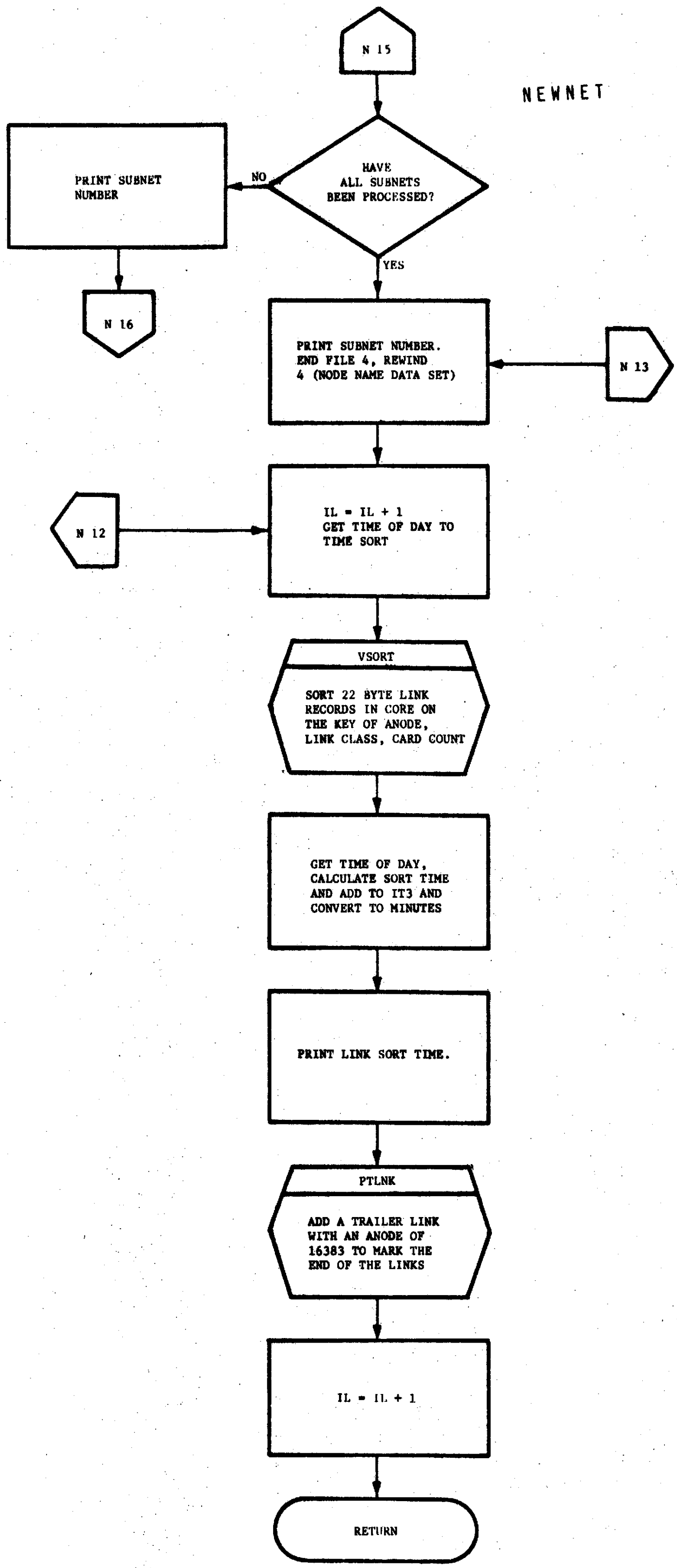

N 5
IS THE SHAFT CODE = 0, +, OR &?
YES
I SHAFT = 0
NO
IS THE SHAFT CODE = 1 OR -?
YES
I SHAFT = 1
NO
IS I SHAFT = -1?
YES
PRINT ERROR MESSAGE THAT THE SHAFT CODE IS INVALID. I SHAFT = 0
NO
I ARROW = I SHAFT
IS THE ARROW CODE EQUAL 1 OR -?
YES
SET I ARROW TO OPPOSITE OF I SHAFT
NO
N 6
SET LINK CLASS 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
YES
N 10
N 9

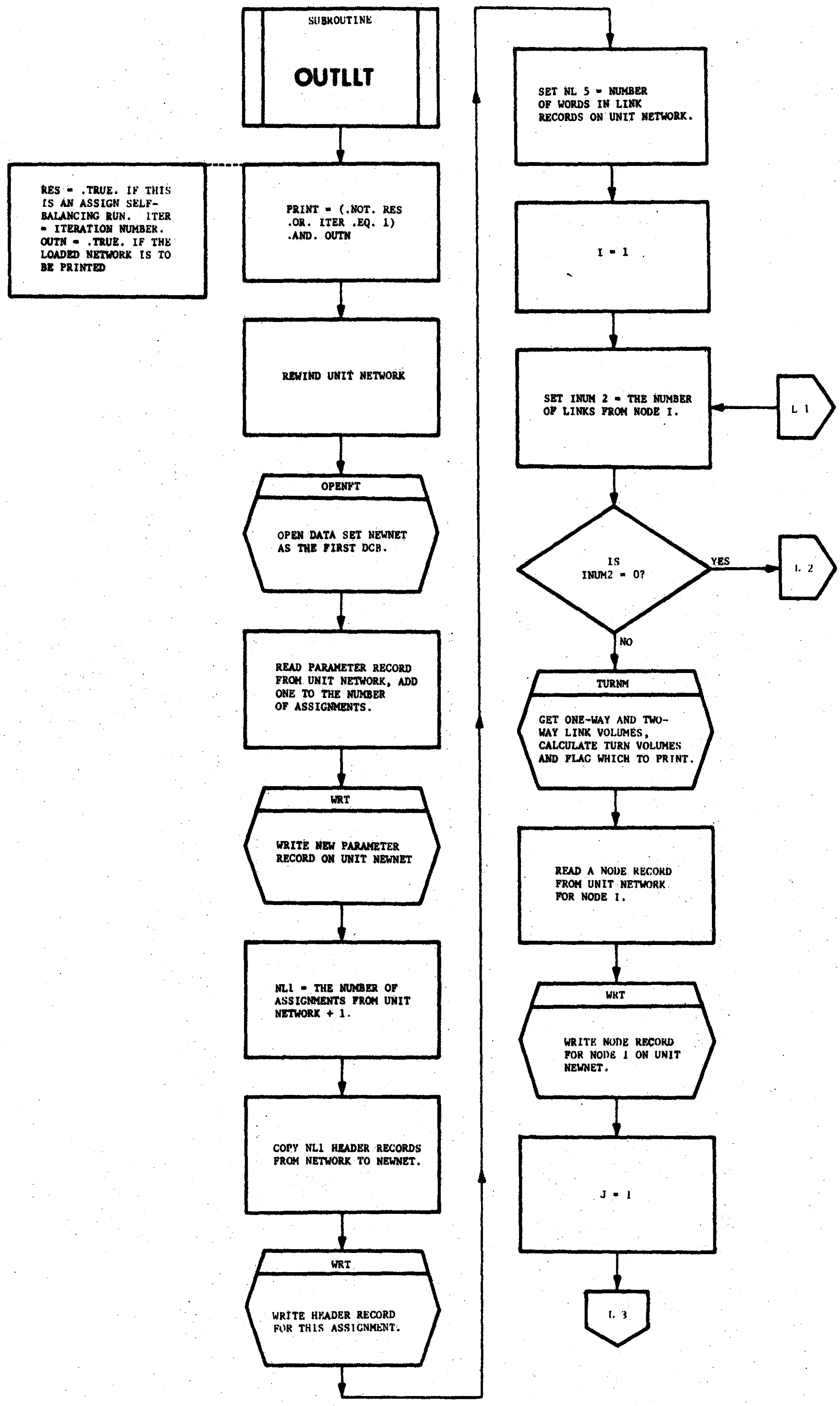

N 8
N 10
SET LINK TIME TO SECOND TIME OR SPEED FIELD CALCULATE LINK SPEED FROM TIME AND DISTANCE
YES
IS THE TWO-WAY INDICATOR = 1?
NO
YES
IS THE TWO-WAY INDICATOR = 2?
NO
PRINT AN ERROR MESSAGE, INVALID TWO-WAY INDICATOR, ERROR = ERROR + 1
IL = IL - 1 (THIS REMOVES THE PREVIOUS ONE-WAY LINK)
N 1
SET SPEED = SECOND TIME OR SPEED FIELD
N 9
PRINT ERROR MESSAGE THAT LINK TIME IS > MAX. ERROR = ERROR + 1 SET LINK TIME TO MAX.
YES
IS THE LINK TIME > THE MAX. LINK TIME?
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

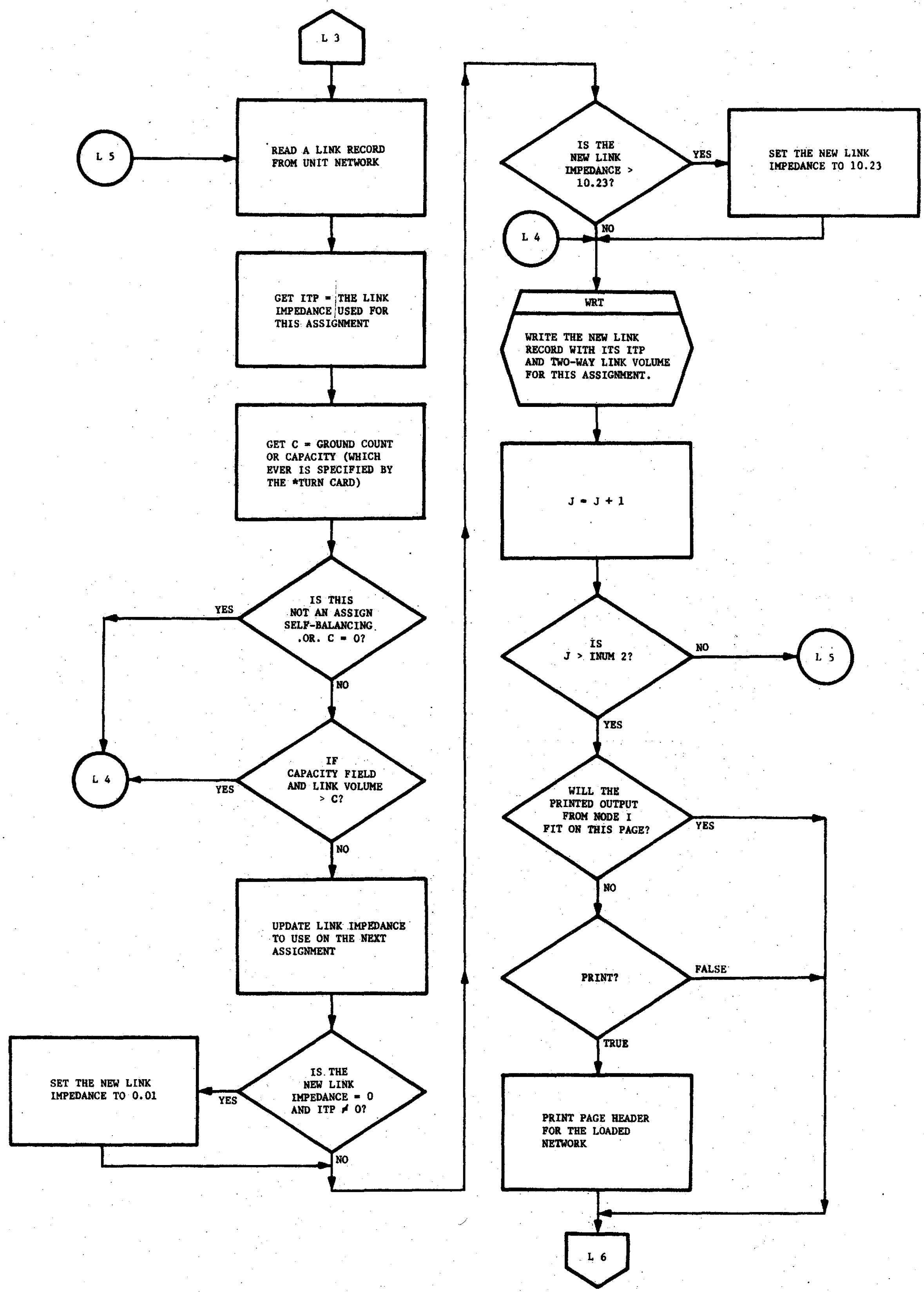

N 11
WRITE THE SORTED LINKS IN CORE OR LNKTMP IN BLOCKS OF 40 LINKS/RECORD.
END FILE LNKTMP
REWIND LNKTMP
IS LNKTMP = 3
YES
LNK 1 = LNK 2
LNK 2 = 0
NO
IS LNKTMP = 11
YES
LNKTMP = -1
NO
IS LNKTMP = 3
YES
LNKTMP = 11
NO
IL = -1
(SET FOR NO LINKS IN CORE)
N 3

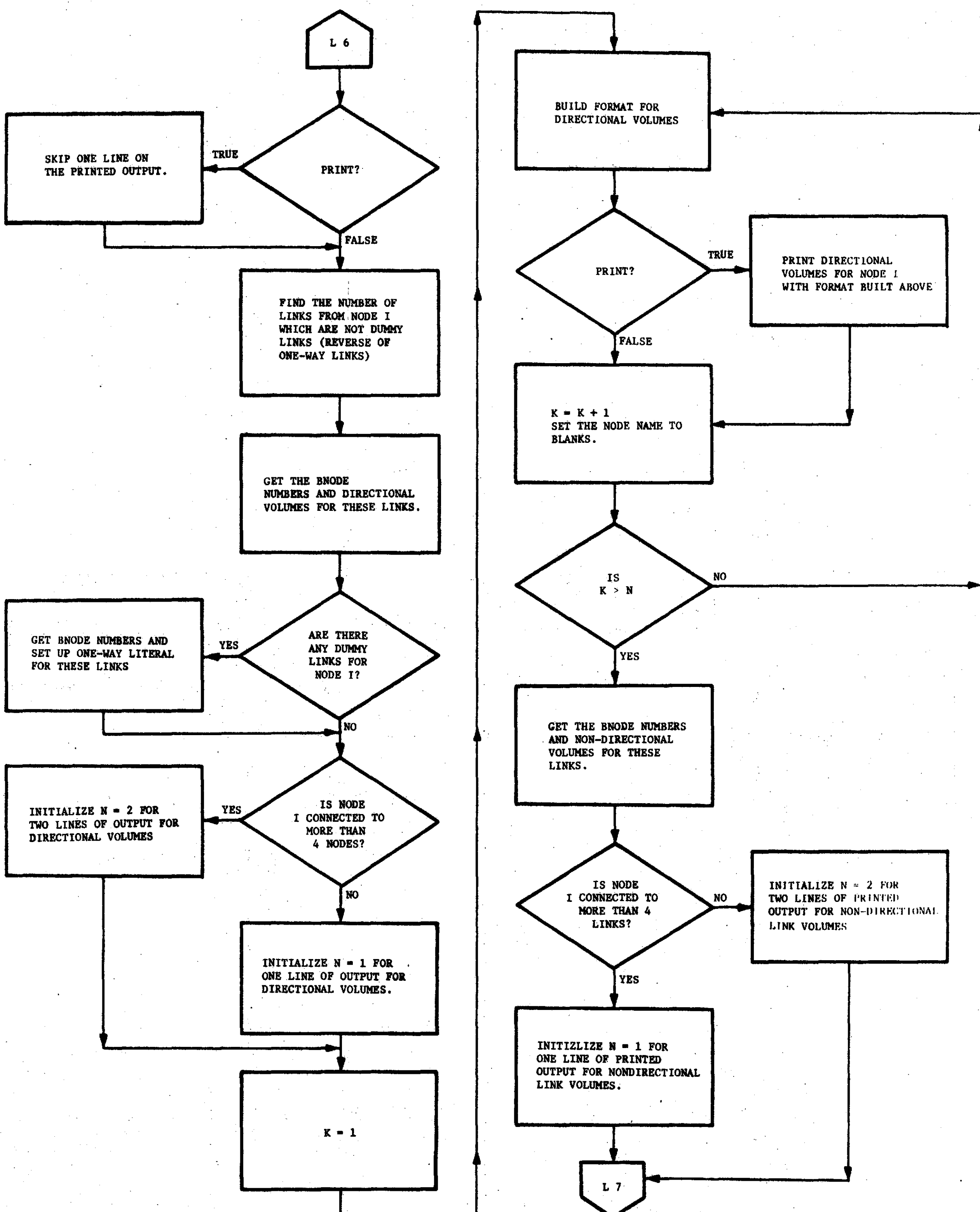

N 14
PTLNK
ADD A TRAILER LINK WITH AN ANODE OF 16383 TO MARK THE END OF THE LINKS
LNK2 = LNK2 + 1
CALCULATE NUMBER OF RECORDS TO WRITE ON LNKTMP.
WRITE THE SORTED LINKS IN CORE ON LNKTMP IN BLOCKS OF 40 LINKS/RECORD
END FILE LNKTMP
REWIND LNKTMP
IS LNKTMP = 3
YES
LNK1 = LNK2
LNK2 = 0
NO
IS LNKTMP = 1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

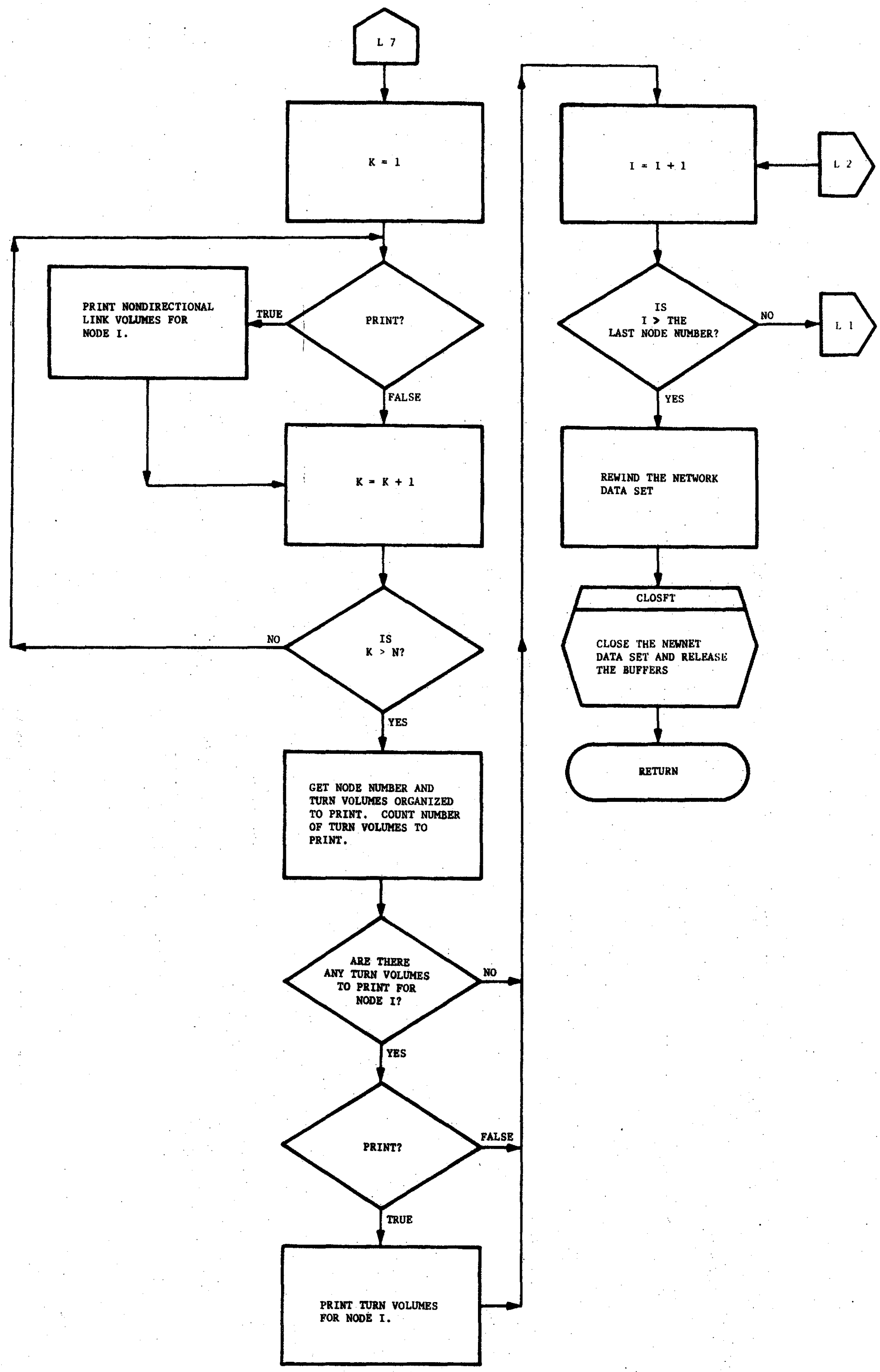
NEWNET

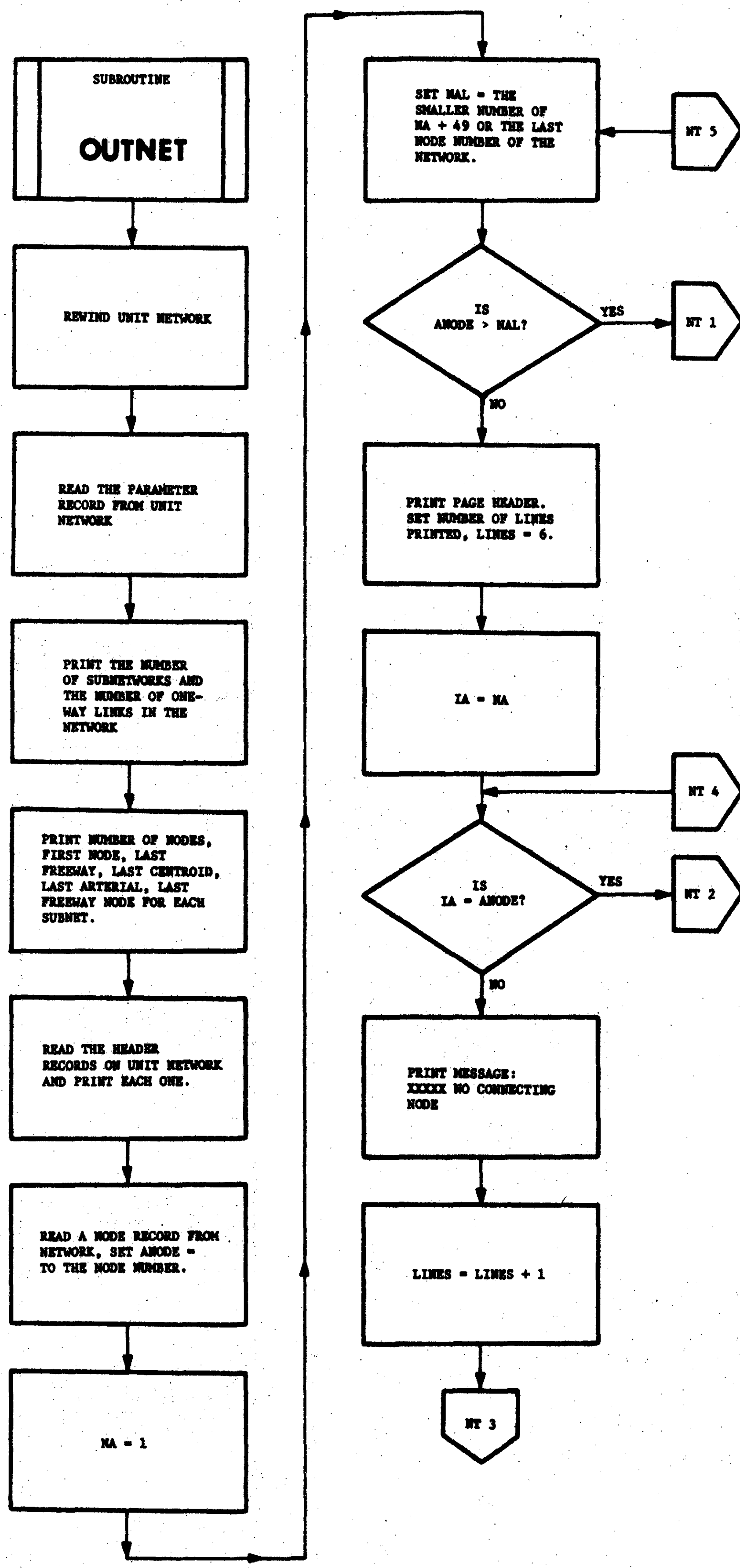

N 15
HAVE ALL SUBNETS BEEN PROCESSED?
NO
PRINT SUBNET NUMBER
N 16
YES
PRINT SUBNET NUMBER. END FILE 4, REWIND 4 (NODE NAME DATA SET)
N 13
N 12
IL = IL + 1 GET TIME OF DAY TO TIME SORT
VSORT
SORT 22 BYTE LINK RECORDS IN CORE ON THE KEY OF ANODE, LINK CLASS, CARD COUNT
GET TIME OF DAY, CALCULATE SORT TIME 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 = .TRUE. IF THE LOADED NETWORK IS TO BE PRINTED
PRINT = (.NOT. RES .OR. ITER .EQ. 1) .AND. OUTN
REWIND UNIT NETWORK
OPENFT
OPEN DATA SET NEWNET AS THE FIRST DCB.
READ PARAMETER RECORD FROM UNIT NETWORK, ADD ONE TO THE NUMBER OF ASSIGNMENTS.
WRT
WRITE NEW PARAMETER RECORD ON UNIT NEWNET
NL1 = THE NUMBER OF ASSIGNMENTS FROM UNIT NETWORK + 1.
COPY NL1 HEADER RECORDS FROM NETWORK TO NEWNET.
WRT
WRITE HEADER RECORD FOR THIS ASSIGNMENT.
SET NL 5 = NUMBER OF WORDS IN LINK RECORDS ON UNIT NETWORK.
I = 1
SET INUM 2 = THE NUMBER OF LINKS FROM NODE I.
L 1
IS INUM2 = 0?
YES
L 2
NO
TURNM
GET ONE-WAY AND TWO-WAY LINK VOLUMES, CALCULATE TURN VOLUMES AND FLAG WHICH TO PRINT.
READ A NODE RECORD FROM UNIT NETWORK FOR NODE I.
WRT
WRITE NODE RECORD FOR NODE I ON UNIT NEWNET.
J = 1
L 3

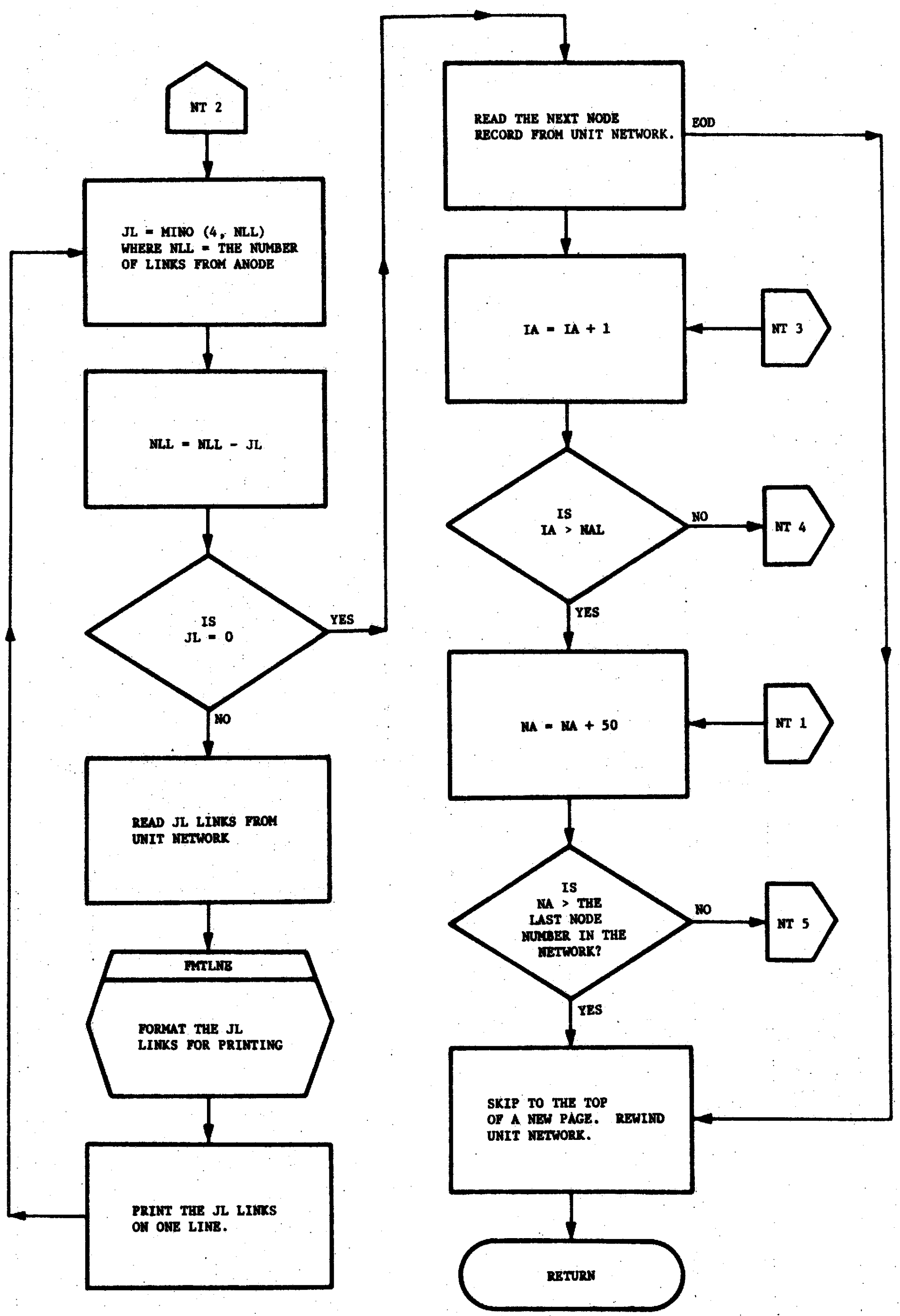

L 3
L 5
READ A LINK RECORD FROM UNIT NETWORK
GET ITP = THE LINK IMPEDANCE USED FOR THIS ASSIGNMENT
GET C = GROUND COUNT OR CAPACITY (WHICH EVER IS SPECIFIED BY THE *TURN CARD)
IS THIS NOT AN ASSIGN SELF-BALANCING .OR. C = 0?
YES
NO
IF CAPACITY FIELD AND LINK VOLUME > C?
YES
L 4
NO
UPDATE LINK IMPEDANCE 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 > 10.23?
YES
SET THE NEW LINK IMPEDANCE TO 10.23
NO
L 4
WRT
WRITE THE NEW LINK RECORD WITH ITS ITP AND TWO-WAY LINK VOLUME FOR THIS ASSIGNMENT.
J = J + 1
IS J > INUM 2?
NO
L 5
YES
WILL THE PRINTED OUTPUT FROM NODE I FIT ON THIS PAGE?
YES
NO
PRINT?
FALSE
TRUE
PRINT PAGE HEADER FOR THE LOADED NETWORK
L 6

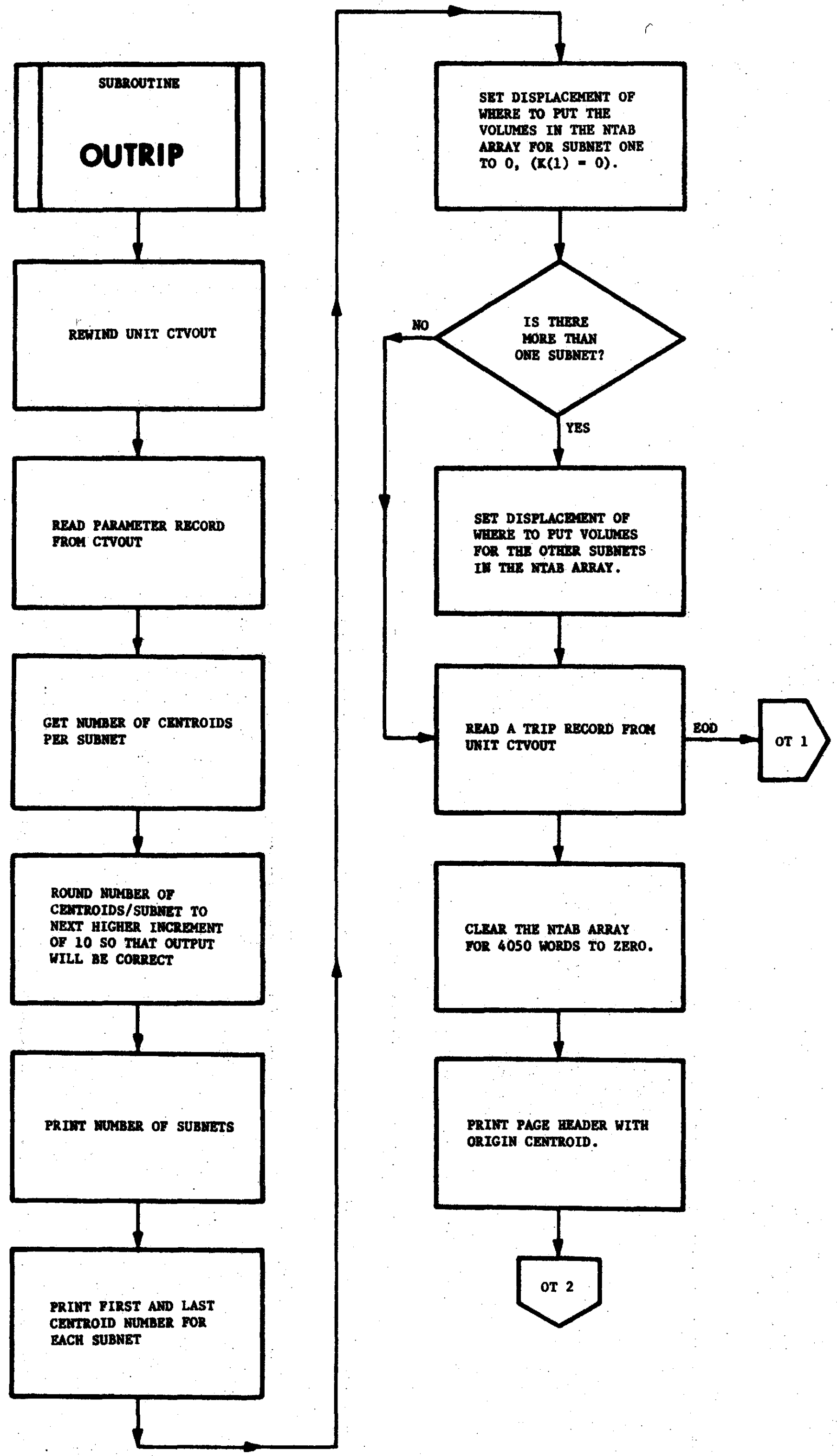

L 6
PRINT?
TRUE
SKIP ONE LINE ON THE PRINTED OUTPUT.
FALSE
FIND THE NUMBER OF LINKS FROM NODE I WHICH ARE NOT DUMMY LINKS (REVERSE OF ONE-WAY LINKS)
GET THE BNODE NUMBERS AND DIRECTIONAL VOLUMES FOR THESE LINKS.
ARE THERE ANY DUMMY LINKS FOR NODE I?
YES
GET BNODE NUMBERS AND SET UP ONE-WAY LITERAL FOR 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?
TRUE
PRINT DIRECTIONAL VOLUMES FOR NODE I WITH FORMAT BUILT ABOVE
FALSE
K = K + 1
SET THE NODE NAME TO BLANKS.
IS K > N
NO
YES
GET THE BNODE NUMBERS AND NON-DIRECTIONAL VOLUMES FOR THESE LINKS.
IS NODE I CONNECTED TO MORE THAN 4 LINKS?
NO
INITIALIZE N = 2 FOR TWO LINES OF PRINTED OUTPUT FOR NON-DIRECTIONAL LINK VOLUMES
YES
INITIZLIZE N = 1 FOR ONE LINE OF PRINTED OUTPUT FOR NONDIRECTIONAL LINK VOLUMES.
L 7

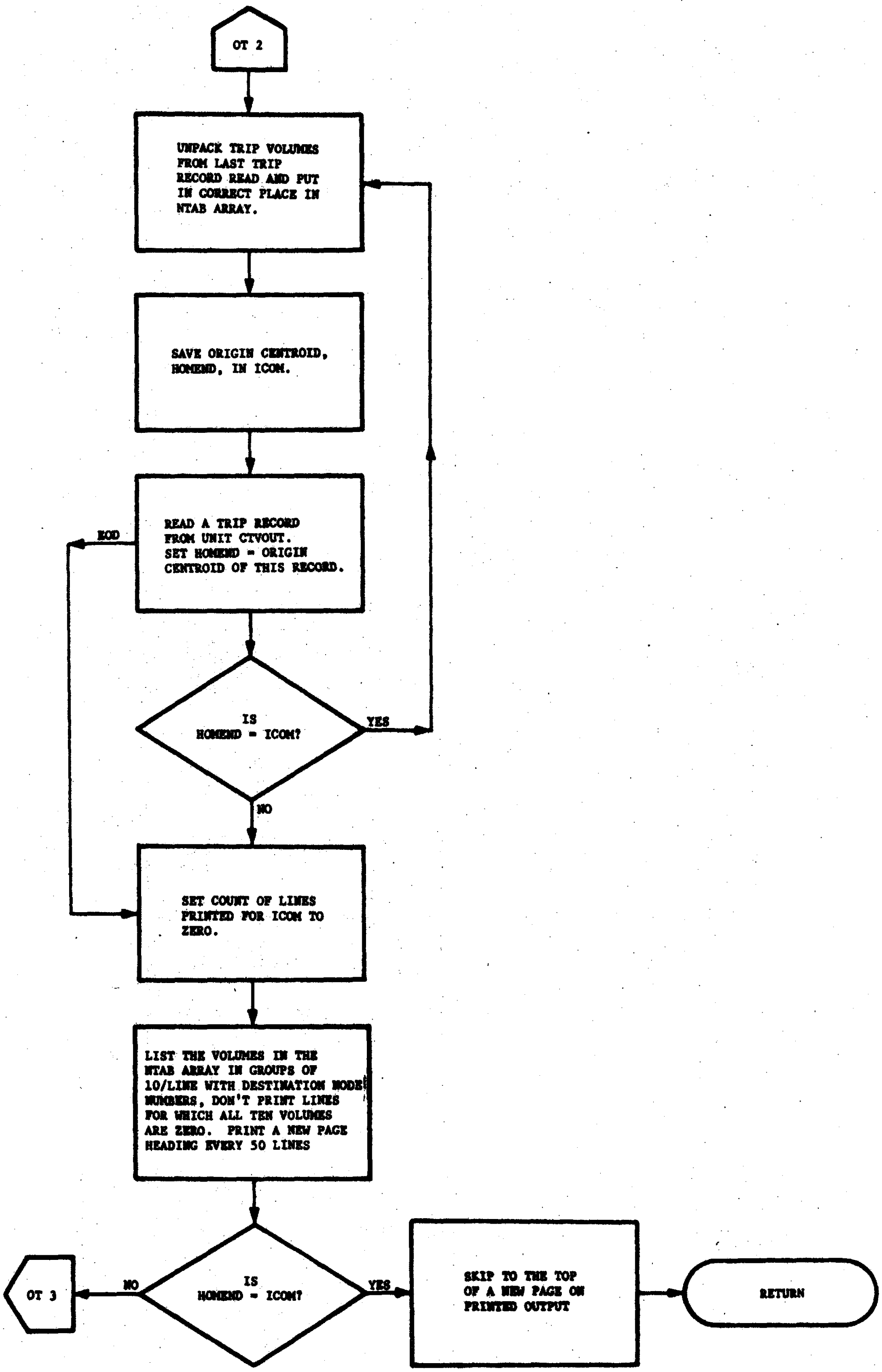

L 7
K = 1
PRINT?
TRUE
PRINT NONDIRECTIONAL LINK VOLUMES FOR NODE I.
FALSE
K = K + 1
IS K > N?
NO
YES
GET NODE NUMBER AND TURN VOLUMES ORGANIZED TO PRINT. COUNT NUMBER OF TURN VOLUMES TO PRINT.
ARE THERE ANY TURN VOLUMES TO PRINT FOR NODE I?
NO
YES
PRINT?
FALSE
TRUE
PRINT TURN VOLUMES FOR NODE I.
I = I + 1
L 2
IS I > THE LAST NODE NUMBER?
NO
L 1
YES
REWIND THE NETWORK DATA SET
CLOSFT
CLOSE THE NEWNET DATA SET AND RELEASE THE BUFFERS
RETURN

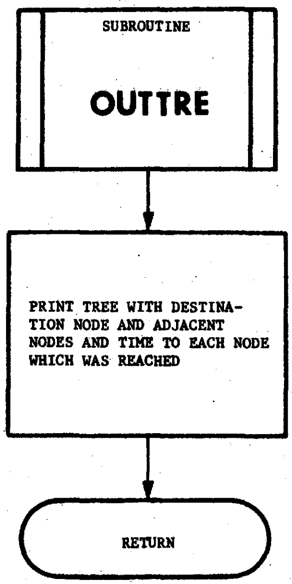


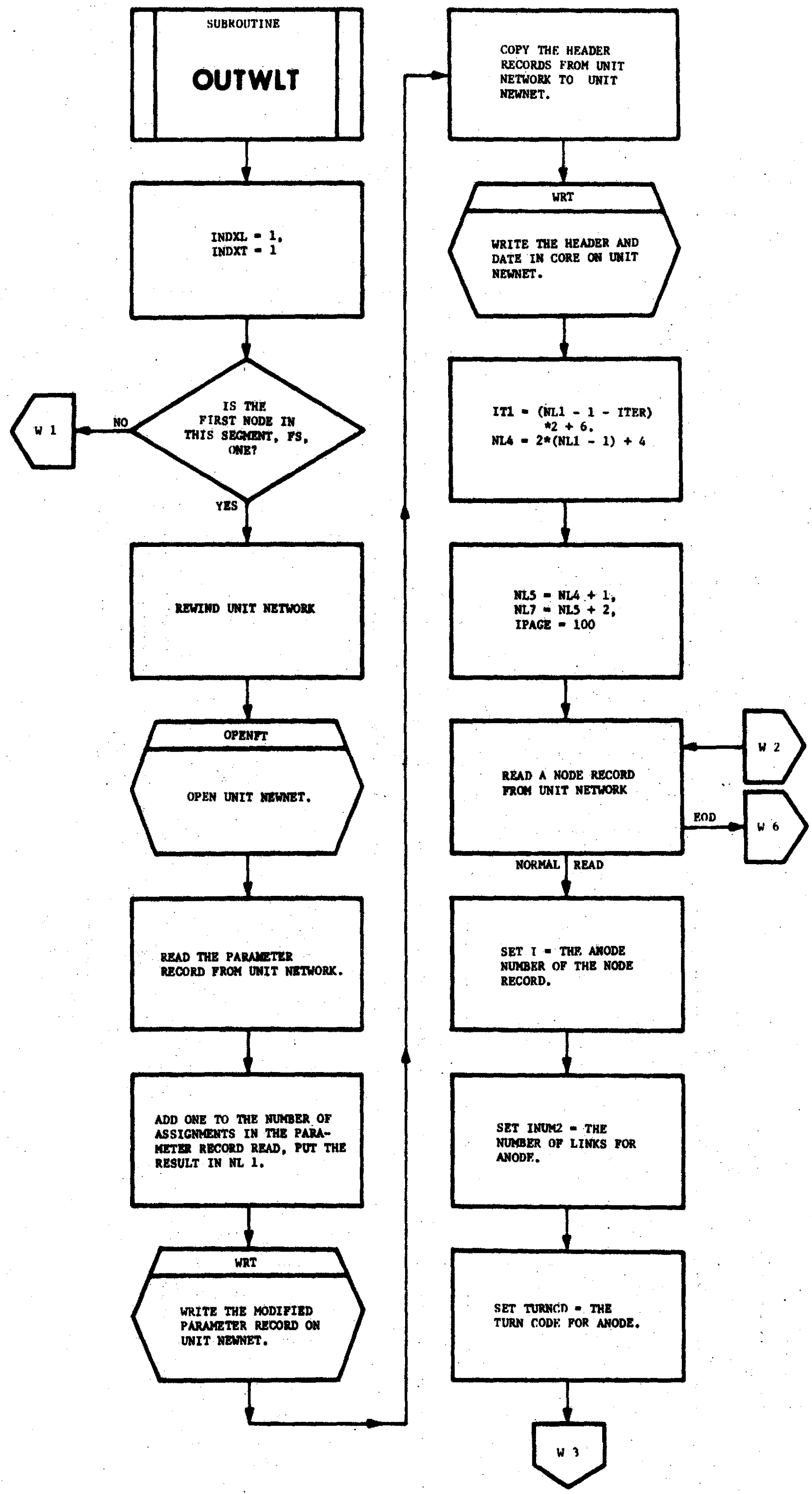
SUBROUTINE
OUTNET
REWIND UNIT NETWORK
READ THE PARAMETER RECORD FROM UNIT NETWORK
PRINT THE NUMBER OF SUBNETWORKS AND THE NUMBER OF ONE-WAY LINKS IN THE NETWORK
PRINT NUMBER OF NODES, FIRST NODE, LAST FREEWAY, LAST CENTROID, LAST ARTERIAL, LAST FREEWAY NODE FOR EACH SUBNET.
READ THE HEADER RECORDS ON 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

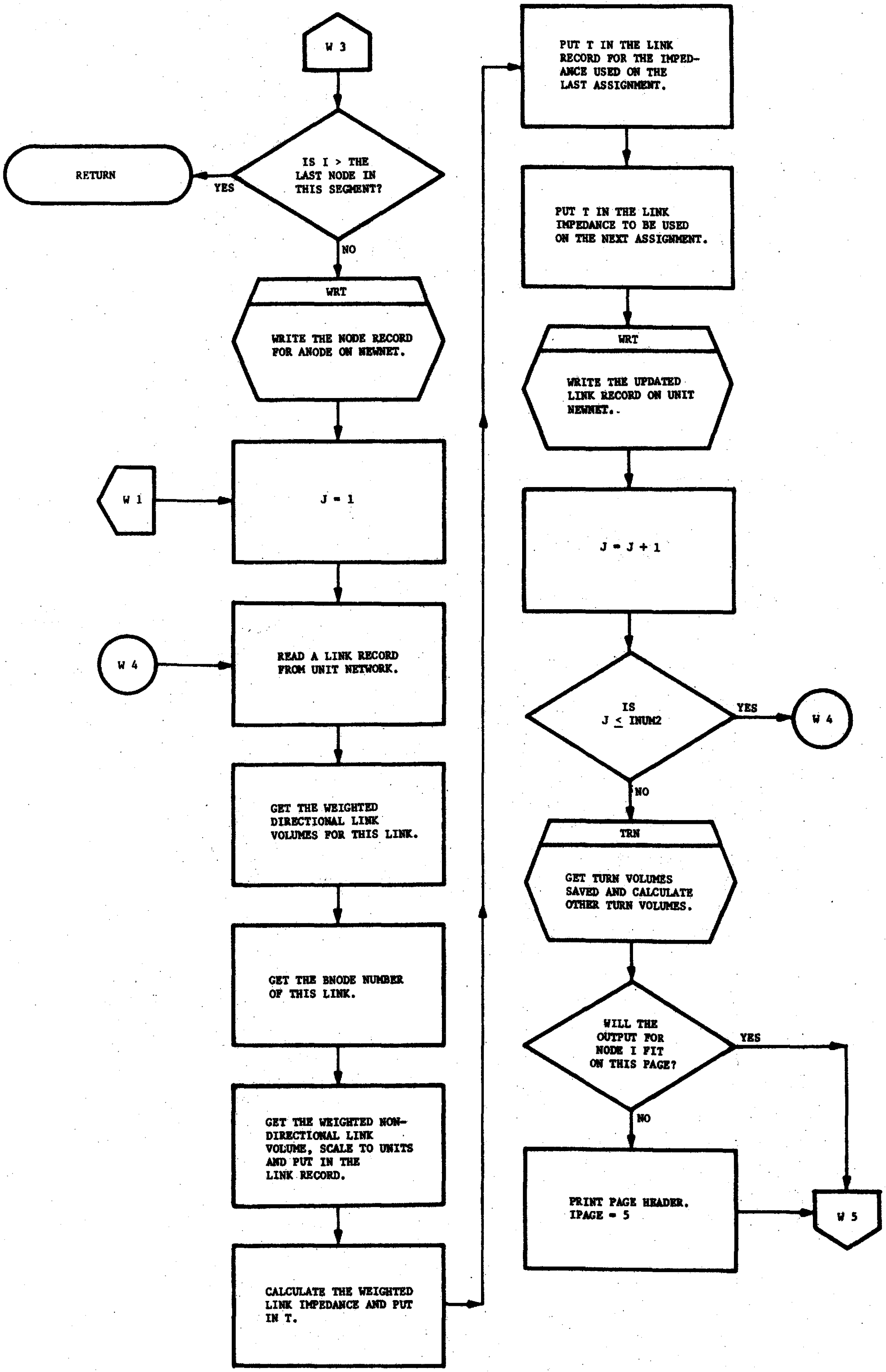

NT 2
JL = MINO (4, NLL)
WHERE NLL = THE NUMBER
OF LINKS FROM ANODE
NLL = NLL - JL
IS
JL = 0
YES
NO
READ JL LINKS FROM
UNIT NETWORK
FMTLNE
FORMAT THE JL
LINKS FOR PRINTING
PRINT THE JL LINKS
ON ONE LINE.
READ THE NEXT NODE
RECORD FROM UNIT NETWORK.
EOD
IA = IA + 1
NT 3
IS
IA > NAL
NO
NT 4
YES
NA = NA + 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

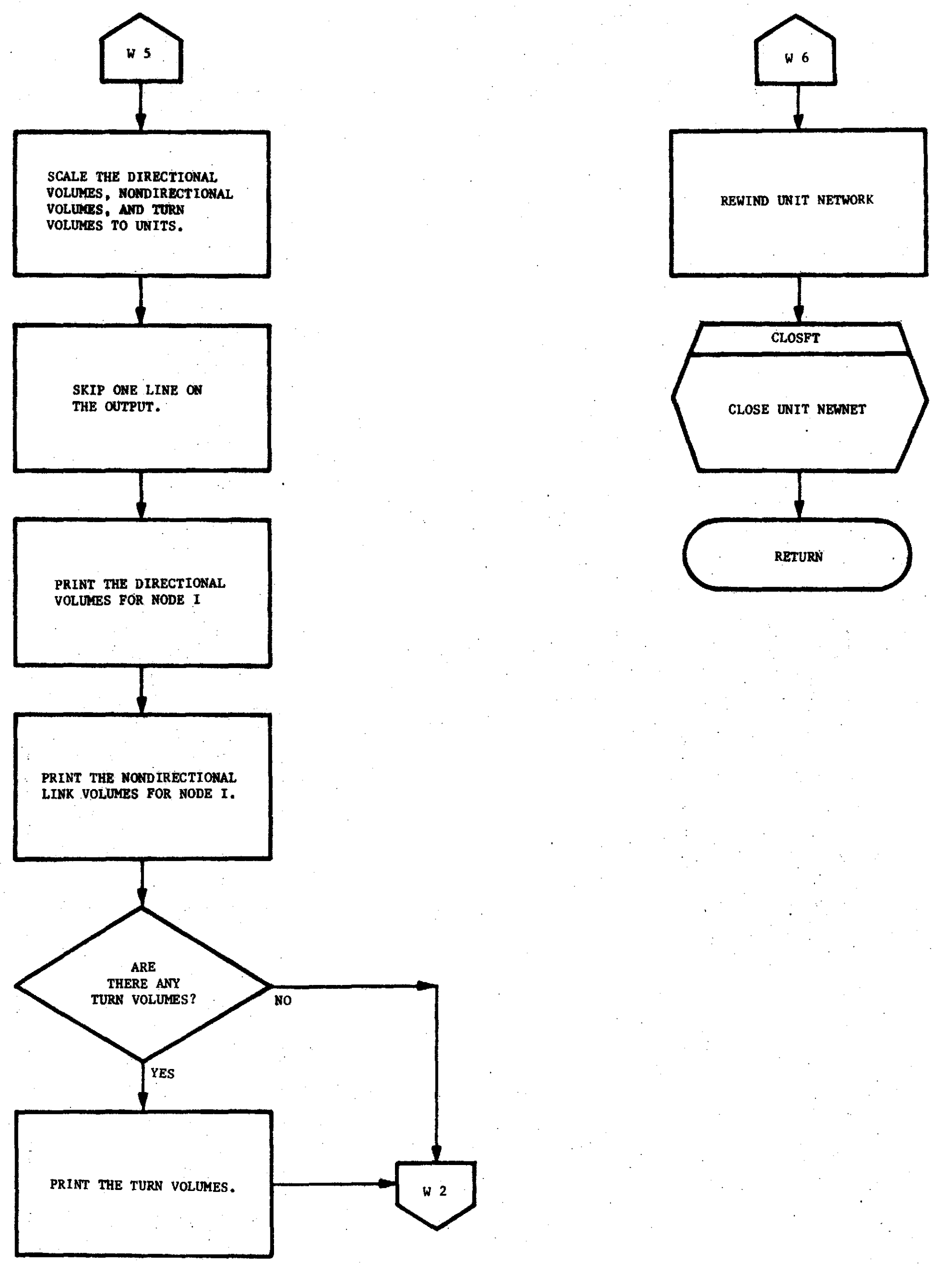

SUBROUTINE
OUTRIP
REWIND UNIT CTVOUT
READ PARAMETER RECORD FROM CTVOUT
GET NUMBER OF CENTROIDS PER SUBNET
ROUND NUMBER OF CENTROIDS/SUBNET TO NEXT HIGHER INCREMENT OF 10 SO THAT OUTPUT WILL BE CORRECT
PRINT NUMBER OF SUBNETS
PRINT FIRST AND LAST CENTROID NUMBER FOR EACH SUBNET
SET DISPLACEMENT OF WHERE 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

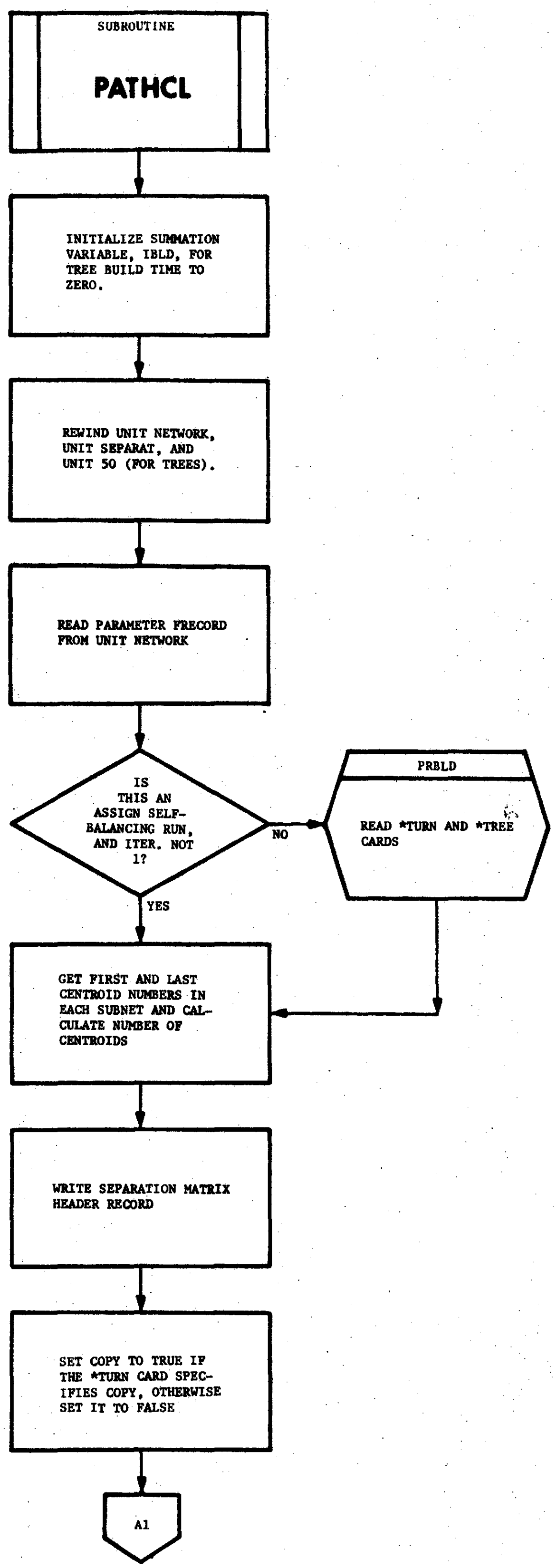

OT 2
UNPACK TRIP VOLUMES FROM LAST TRIP RECORD READ AND PUT IN CORRECT PLACE IN NTAB ARRAY.
SAVE ORIGIN CENTROID, HOMEND, IN ICOM.
READ A TRIP RECORD FROM UNIT CTVOUT. SET HOMEND = ORIGIN CENTROID OF THIS RECORD.
EOD
IS HOMEND = ICOM?
YES
NO
SET COUNT OF LINES PRINTED FOR ICOM TO ZERO.
LIST THE VOLUMES 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

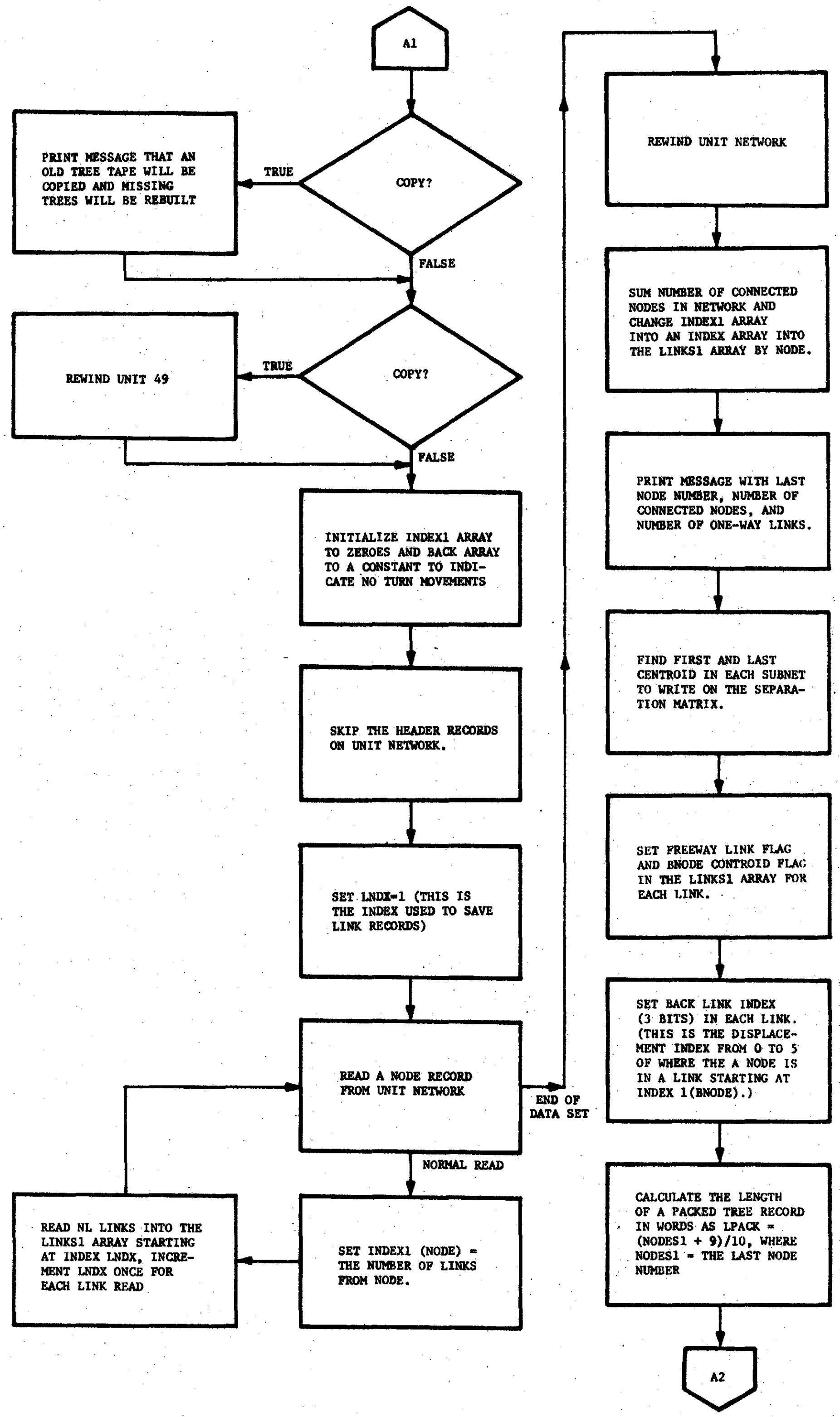

SUBROUTINE
OUTTRE
PRINT TREE WITH DESTINATION NODE AND ADJACENT NODES AND TIME TO EACH NODE WHICH WAS REACHED
RETURN

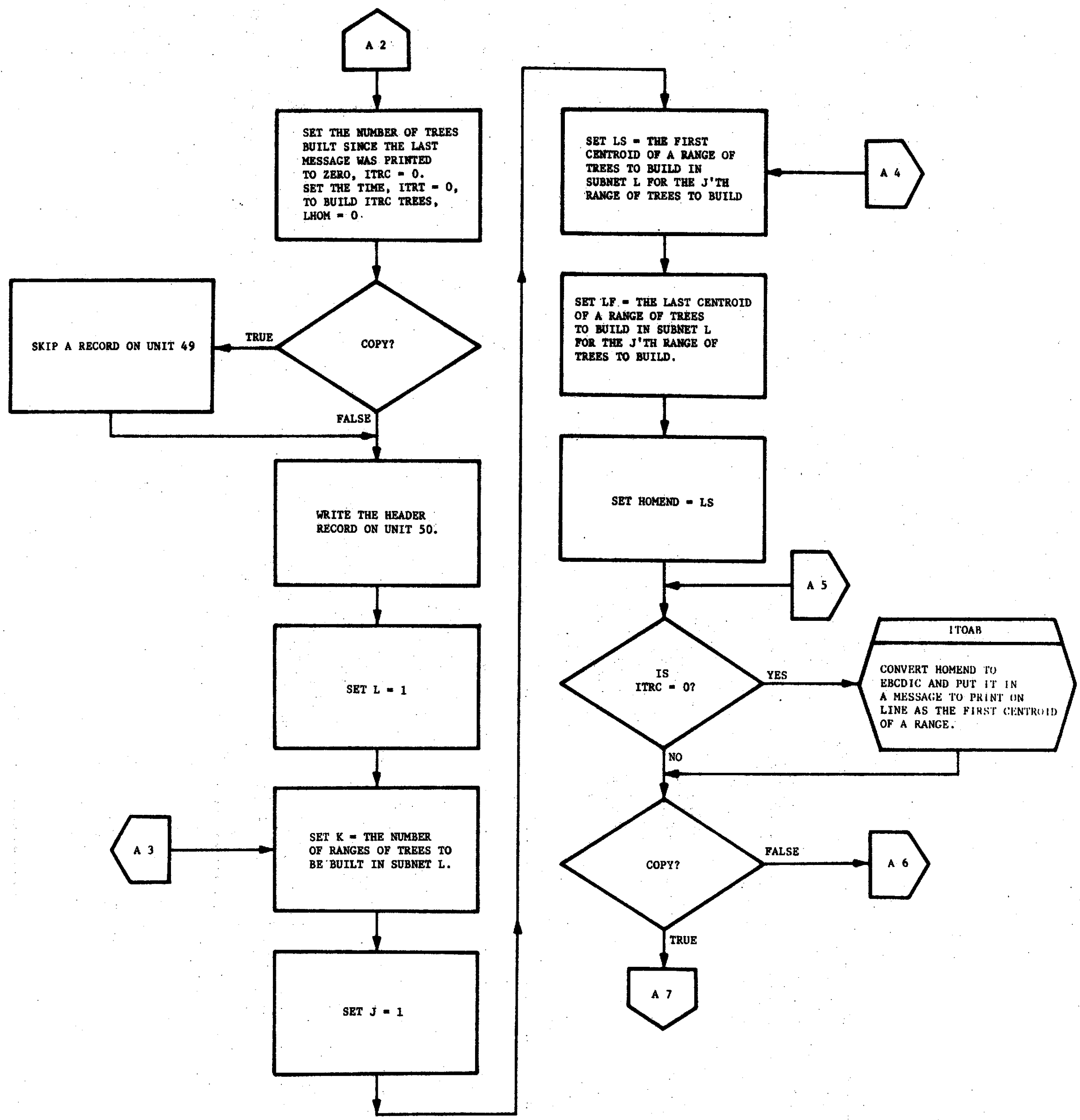

SUBROUTINE
OUTWLT
INDXL = 1,
INDXT = 1
IS THE FIRST NODE IN THIS SEGMENT, FS, ONE?
NO
W 1
YES
REWIND UNIT NETWORK
OPENFT
OPEN UNIT NEWNET.
READ THE PARAMETER RECORD FROM UNIT NETWORK.
ADD ONE TO THE NUMBER OF ASSIGNMENTS IN THE PARAMETER RECORD READ, PUT THE RESULT IN NL 1.
WRT
WRITE THE MODIFIED PARAMETER RECORD ON UNIT NEWNET.
COPY THE HEADER RECORDS FROM UNIT NETWORK TO UNIT NEWNET.
WRT
WRITE THE HEADER AND DATE IN CORE ON UNIT NEWNET.
IT1 = (NL1 - 1 - ITER) *2 + 6,
NL4 = 2*(NL1 - 1) + 4
NL5 = NL4 + 1,
NL7 = NL5 + 2,
IPAGE = 100
READ A NODE RECORD FROM UNIT NETWORK
W 2
EOD
W 6
NORMAL READ
SET I = THE ANODE NUMBER OF THE NODE RECORD.
SET INUM2 = THE NUMBER OF LINKS FOR ANODE.
SET TURNCD = THE TURN CODE FOR ANODE.
W 3

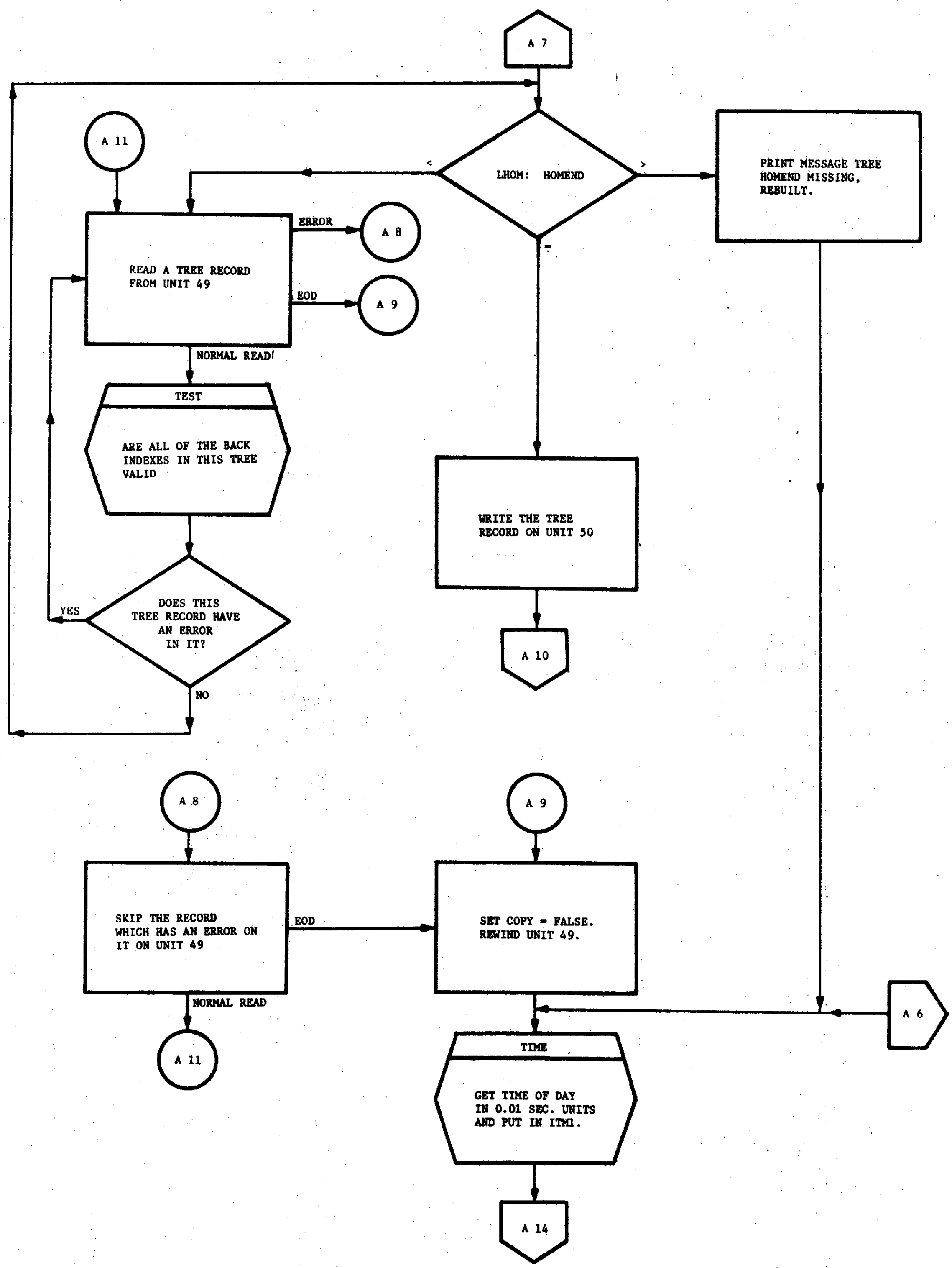

W 3
IS I > THE LAST NODE IN THIS SEGMENT?
YES
RETURN
NO
WRT
WRITE THE NODE RECORD FOR ANODE ON NEWNET.
W 1
J = 1
W 4
READ A LINK RECORD FROM UNIT NETWORK.
GET THE WEIGHTED DIRECTIONAL LINK VOLUMES FOR THIS LINK.
GET THE BNODE NUMBER OF THIS LINK.
GET THE WEIGHTED NON-DIRECTIONAL LINK VOLUME, SCALE TO UNITS AND PUT IN THE LINK RECORD.
CALCULATE THE WEIGHTED LINK IMPEDANCE AND PUT IN T.
PUT T IN THE LINK RECORD FOR THE IMPED-ANCE USED ON THE LAST ASSIGNMENT.
PUT T IN THE LINK IMPEDANCE TO BE USED ON THE NEXT ASSIGNMENT.
WRT
WRITE THE UPDATED LINK RECORD ON UNIT NEWNET.
J = J + 1
IS J ≤ INUM2
YES
W 4
NO
TRN
GET TURN VOLUMES SAVED AND CALCULATE OTHER TURN VOLUMES.
WILL THE OUTPUT FOR NODE I FIT ON THIS PAGE?
YES
NO
PRINT PAGE HEADER. IPAGE = 5
W 5

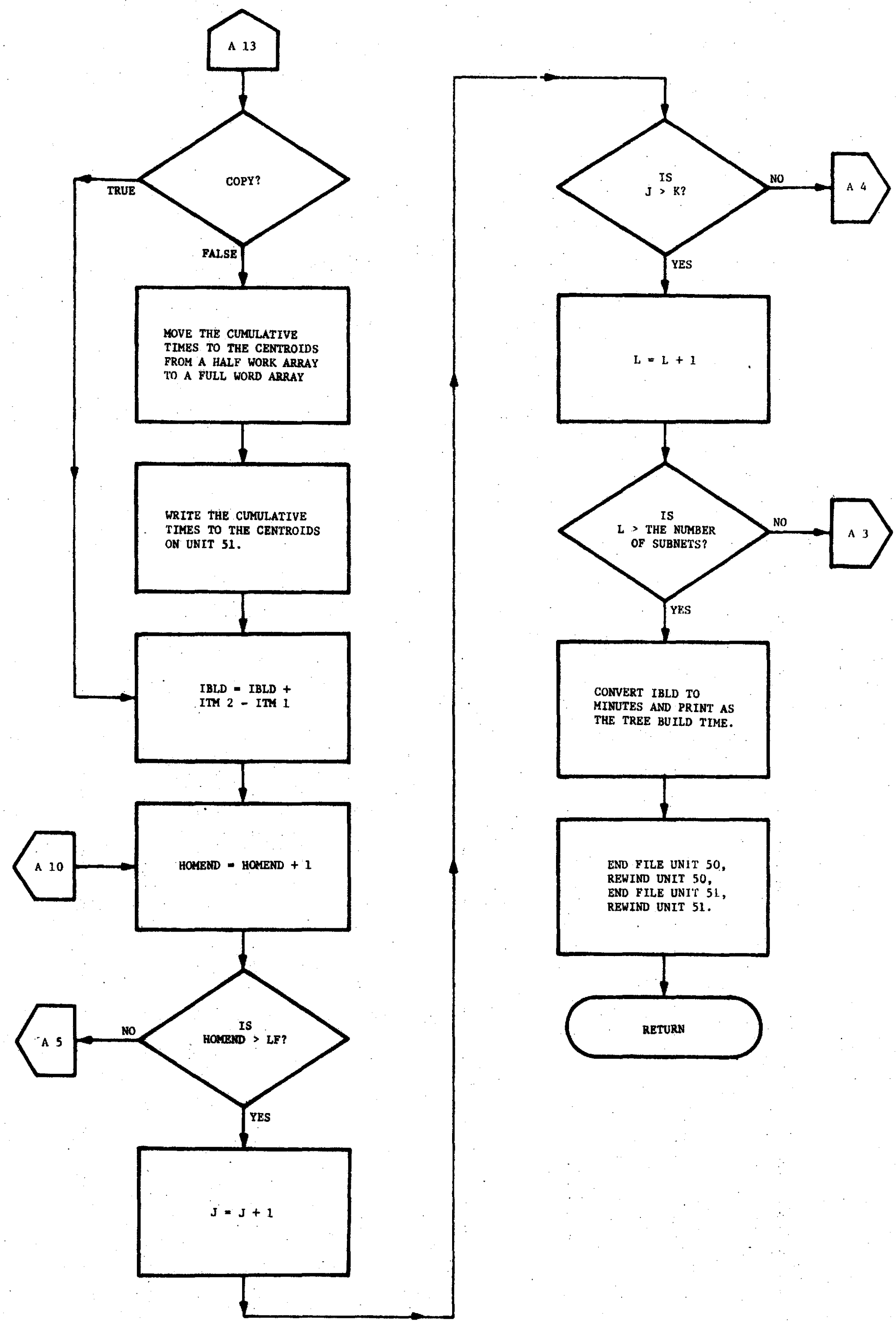

W 5
SCALE THE DIRECTIONAL VOLUMES, NONDIRECTIONAL VOLUMES, AND TURN VOLUMES TO UNITS.
SKIP ONE LINE ON THE OUTPUT.
PRINT THE DIRECTIONAL VOLUMES FOR NODE I
PRINT THE NONDIRECTIONAL LINK VOLUMES FOR NODE I.
ARE THERE ANY TURN VOLUMES?
NO
YES
PRINT THE TURN VOLUMES.
W 2
W 6
REWIND UNIT NETWORK
CLOSFT
CLOSE UNIT NEWNET
RETURN

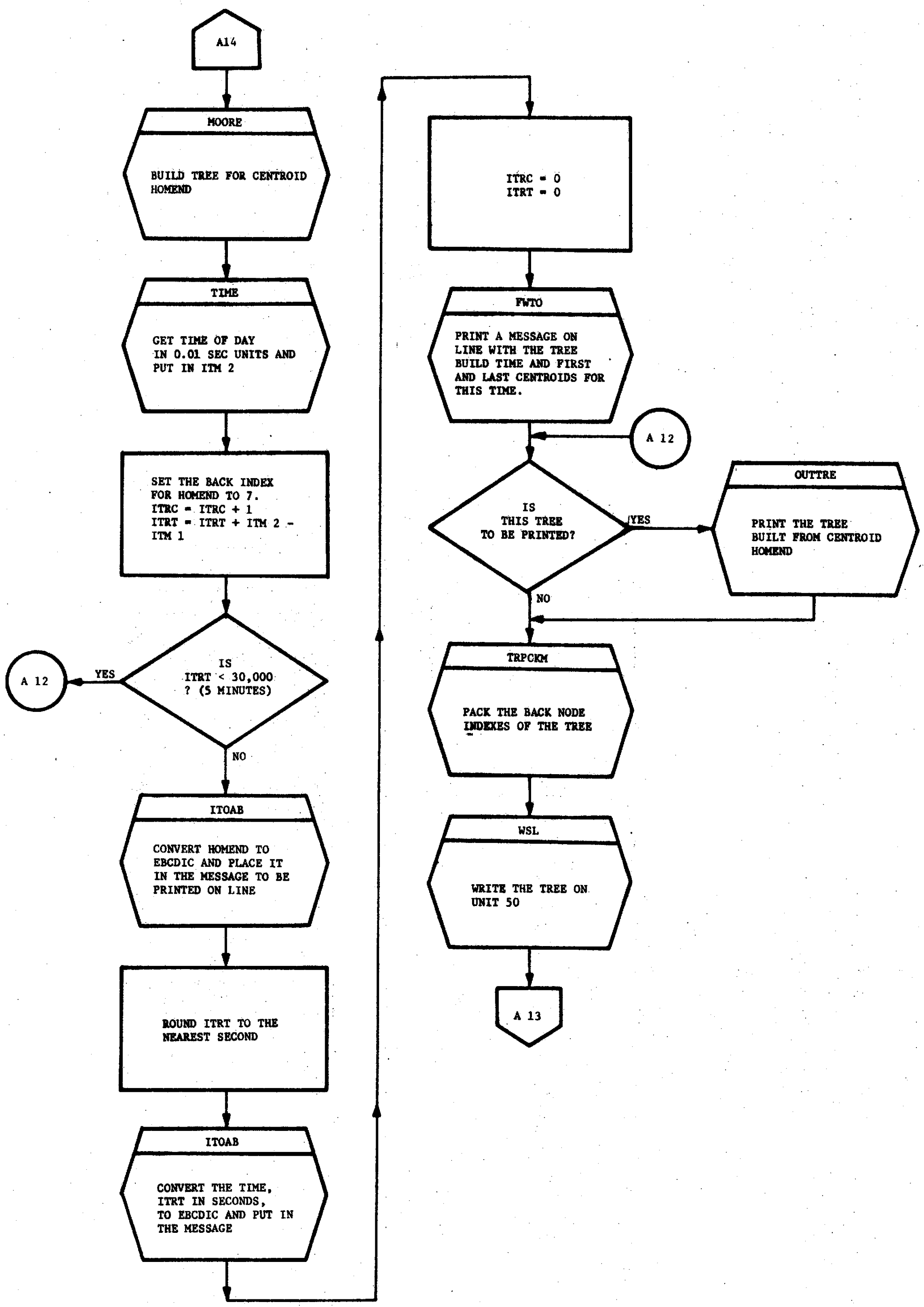

SUBROUTINE
PATHCL
INITIALIZE SUMMATION VARIABLE, IBLD, FOR TREE BUILD TIME TO ZERO.
REWIND UNIT NETWORK, UNIT SEPARAT, AND UNIT 50 (FOR TREES).
READ PARAMETER FRECORD FROM UNIT NETWORK
IS THIS AN ASSIGN SELF-BALANCING RUN, AND ITER. NOT 1?
NO
YES
PRBLD
READ *TURN AND *TREE CARDS
GET FIRST AND LAST CENTROID NUMBERS IN EACH SUBNET AND CALCULATE NUMBER OF CENTROIDS
WRITE SEPARATION MATRIX HEADER RECORD
SET COPY TO TRUE IF THE *TURN CARD SPECIFIES COPY, OTHERWISE SET IT TO FALSE
A1

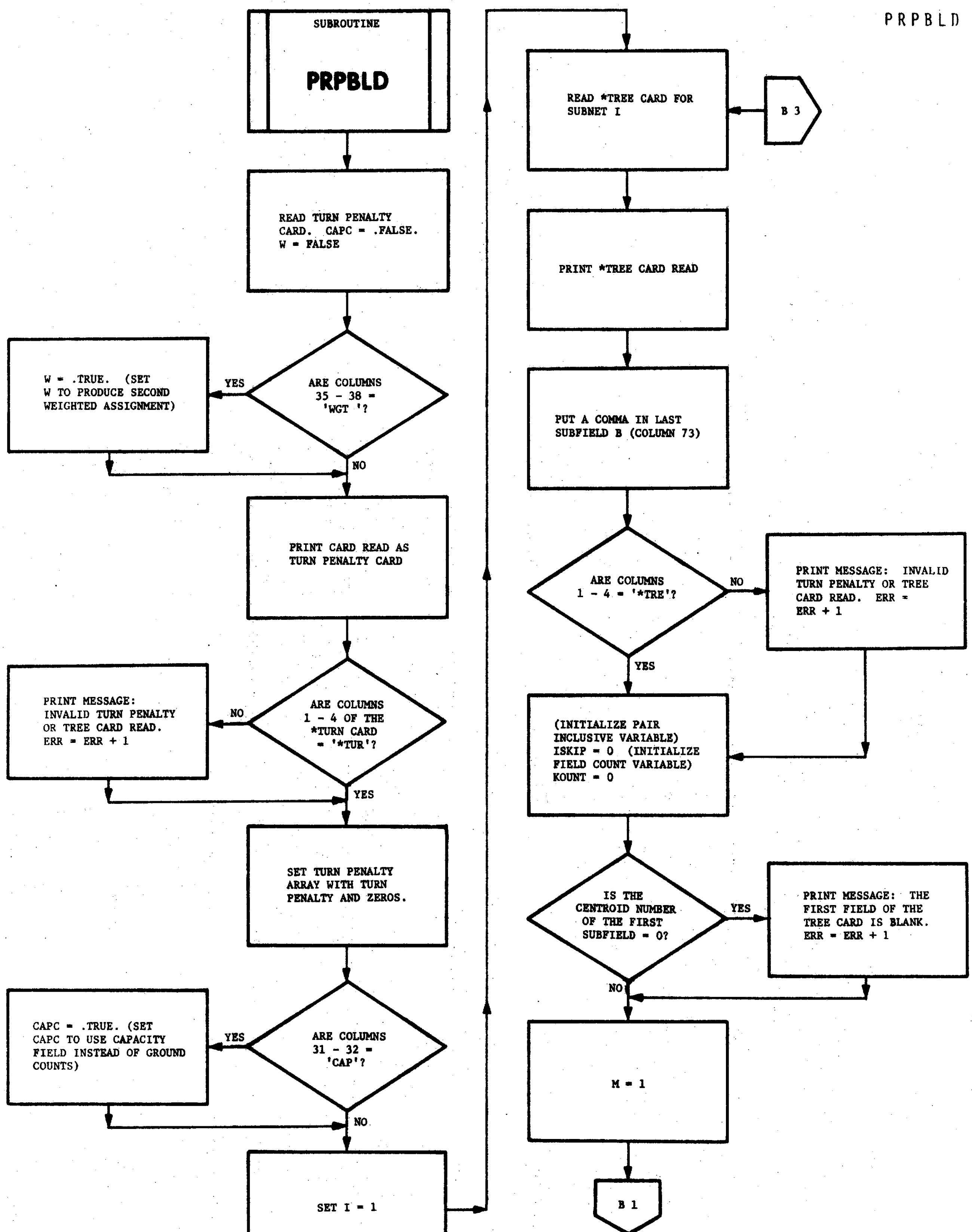

A1
COPY?
TRUE
PRINT MESSAGE THAT AN OLD TREE TAPE WILL BE COPIED AND MISSING TREES WILL BE REBUILT
FALSE
COPY?
TRUE
REWIND UNIT 49
FALSE
INITIALIZE INDEX1 ARRAY TO ZEROES AND BACK ARRAY TO A CONSTANT TO INDICATE NO TURN MOVEMENTS
SKIP THE HEADER RECORDS ON UNIT NETWORK.
SET LNDX=1 (THIS IS THE INDEX USED TO SAVE LINK RECORDS)
READ A NODE RECORD FROM UNIT NETWORK
END OF DATA SET
NORMAL READ
SET INDEX1 (NODE) = THE NUMBER OF LINKS FROM NODE.
READ NL LINKS INTO THE LINKS1 ARRAY STARTING AT INDEX LNDX, INCREMENT LNDX ONCE FOR EACH LINK READ
REWIND UNIT NETWORK
SUM NUMBER OF CONNECTED NODES IN NETWORK AND CHANGE INDEX1 ARRAY INTO AN INDEX ARRAY INTO THE LINKS1 ARRAY BY NODE.
PRINT MESSAGE WITH LAST NODE NUMBER, NUMBER OF CONNECTED NODES, AND NUMBER OF ONE-WAY LINKS.
FIND FIRST AND LAST CENTROID IN EACH SUBNET TO WRITE ON THE SEPARATION MATRIX.
SET FREEWAY LINK FLAG AND BNODE CONTROID FLAG IN THE LINKS1 ARRAY FOR EACH LINK.
SET BACK LINK INDEX (3 BITS) IN EACH LINK. (THIS IS THE DISPLACEMENT INDEX FROM 0 TO 5 OF WHERE THE A NODE IS IN A LINK STARTING AT INDEX 1(BNODE).)
CALCULATE THE LENGTH OF A PACKED TREE RECORD IN WORDS AS LPACK = (NODES1 + 9)/10, WHERE NODES1 = THE LAST NODE NUMBER
A2

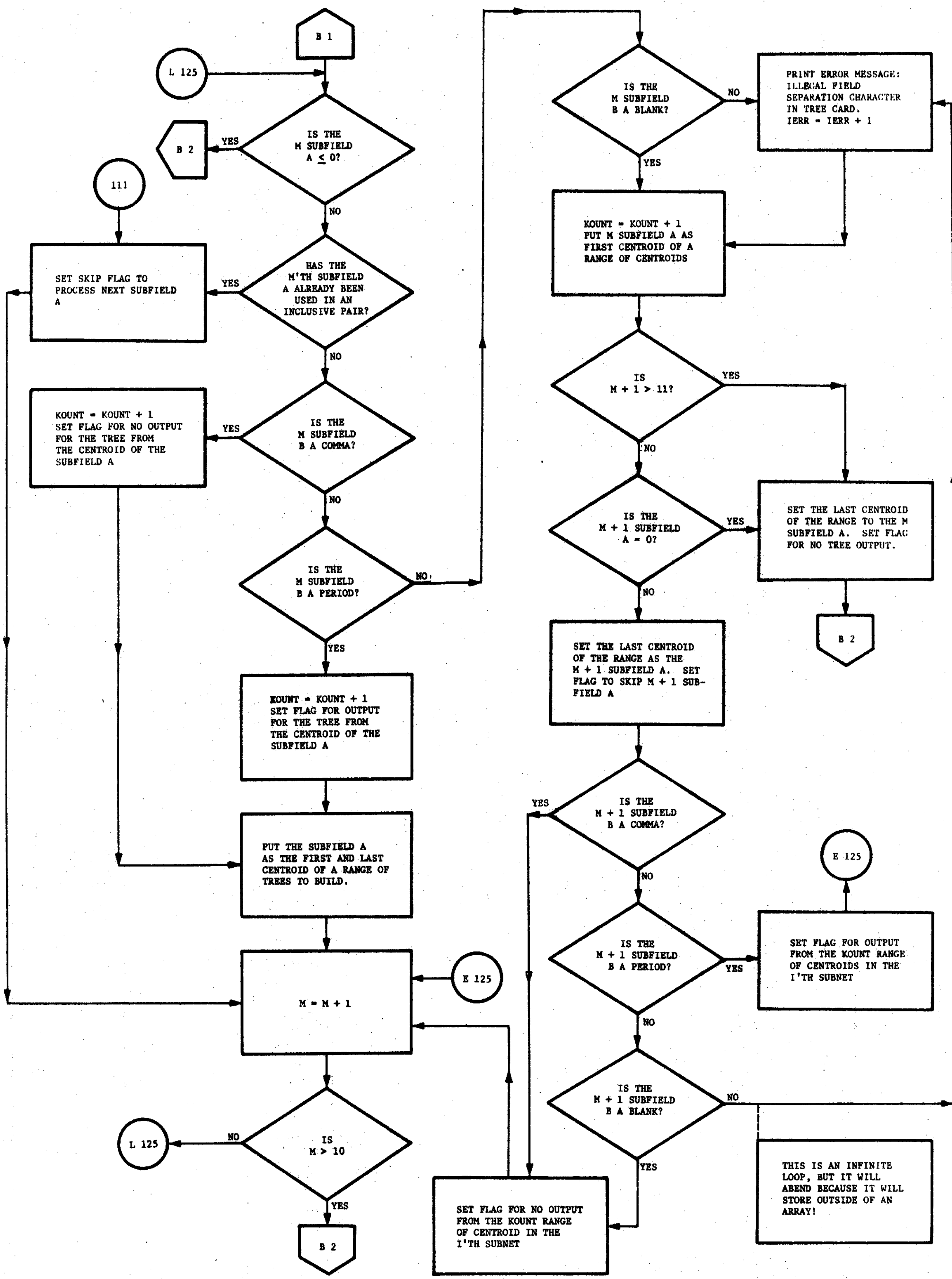

A 2
SET THE NUMBER OF TREES BUILT SINCE THE LAST MESSAGE WAS PRINTED TO ZERO, ITRC = 0. SET THE TIME, ITRT = 0, TO BUILD ITRC TREES, LHOM = 0.
COPY?
TRUE
SKIP A RECORD ON UNIT 49
FALSE
WRITE THE HEADER RECORD ON UNIT 50.
SET L = 1
A 3
SET K = THE NUMBER OF RANGES OF TREES TO BE BUILT IN SUBNET L.
SET J = 1
SET LS = THE FIRST CENTROID OF A RANGE OF TREES TO BUILD IN SUBNET L FOR THE J'TH RANGE OF TREES TO BUILD
A 4
SET LF = THE LAST CENTROID OF A RANGE OF TREES TO BUILD IN SUBNET L FOR THE J'TH RANGE OF TREES TO BUILD.
SET HOMEND = LS
A 5
IS ITRC = 0?
YES
ITOAB
CONVERT HOMEND TO EBCDIC AND PUT IT IN A MESSAGE TO PRINT ON LINE AS THE FIRST CENTROID OF A RANGE.
NO
COPY?
FALSE
A 6
TRUE
A 7


A 7
LHOM: HOMEND
<
>
=
PRINT MESSAGE TREE HOMEND MISSING, REBUILT.
A 11
READ A TREE RECORD FROM UNIT 49
ERROR
A 8
EOD
A 9
NORMAL READ
TEST
ARE ALL OF THE BACK INDEXES IN THIS TREE VALID
DOES THIS TREE RECORD HAVE AN ERROR IN IT?
YES
NO
WRITE THE TREE RECORD ON UNIT 50
A 10
A 8
A 9
SKIP THE RECORD WHICH HAS AN ERROR ON IT ON UNIT 49
EOD
SET COPY = FALSE. REWIND UNIT 49.
NORMAL READ
A 11
A 6
TIME
GET TIME OF DAY IN 0.01 SEC. UNITS AND PUT IN ITM1.
A 14

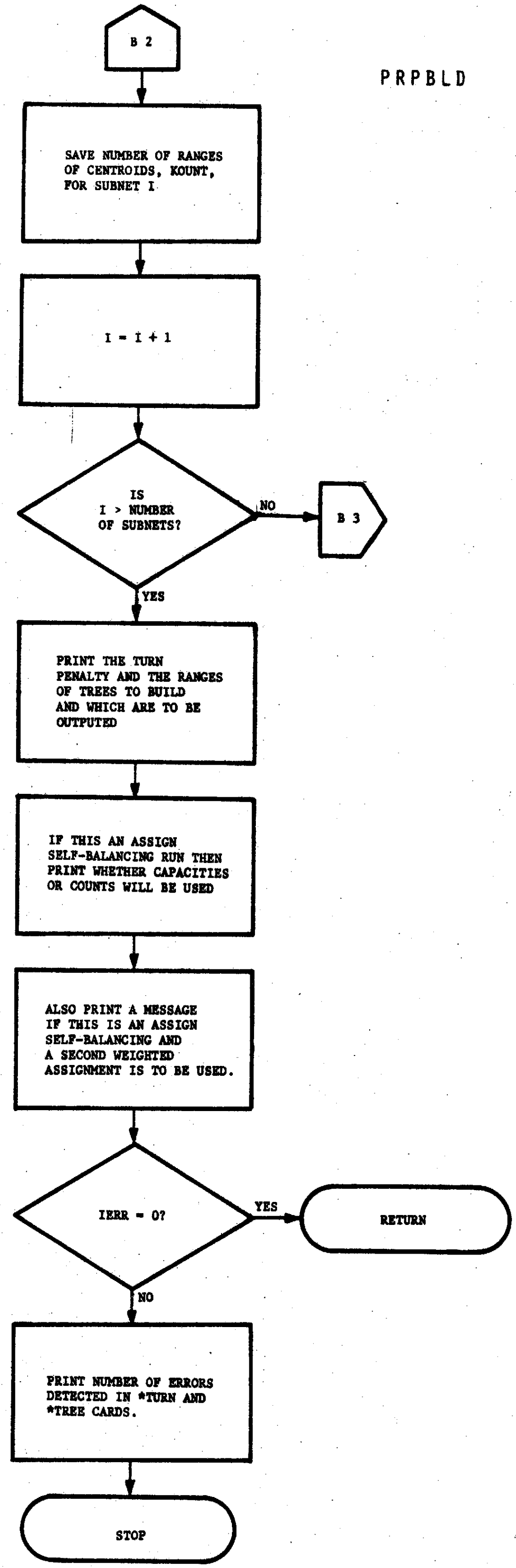

A 13
COPY?
TRUE
FALSE
MOVE THE CUMULATIVE TIMES TO THE CENTROIDS FROM A HALF WORK ARRAY TO A FULL WORD ARRAY
WRITE THE CUMULATIVE TIMES TO THE CENTROIDS ON UNIT 51.
IBLD = IBLD + ITM 2 - ITM 1
A 10
HOMEND = HOMEND + 1
A 5
NO
IS HOMEND > LF?
YES
J = J + 1
IS J > K?
NO
A 4
YES
L = L + 1
IS L > THE NUMBER OF SUBNETS?
NO
A 3
YES
CONVERT IBLD TO MINUTES AND PRINT AS THE TREE BUILD TIME.
END FILE UNIT 50, REWIND UNIT 50, END FILE UNIT 51, REWIND UNIT 51.
RETURN


A14
MOORE
BUILD TREE FOR CENTROID HOMEND
TIME
GET TIME OF DAY IN 0.01 SEC UNITS AND PUT IN ITM 2
SET THE BACK INDEX FOR HOMEND TO 7.
ITRC = ITRC + 1
ITRT = ITRT + ITM 2 - ITM 1
IS ITRT < 30,000 ? (5 MINUTES)
YES
A 12
NO
ITOAB
CONVERT HOMEND TO EBCDIC AND PLACE IT IN THE MESSAGE TO BE PRINTED ON LINE
ROUND ITRT TO THE NEAREST SECOND
ITOAB
CONVERT THE TIME, ITRT IN SECONDS, TO EBCDIC AND PUT IN THE MESSAGE
ITRC = 0
ITRT = 0
FWTO
PRINT A MESSAGE ON LINE WITH THE TREE BUILD TIME AND FIRST AND LAST CENTROIDS FOR THIS TIME.
A 12
IS THIS TREE TO BE PRINTED?
YES
OUTTRE
PRINT THE TREE BUILT FROM CENTROID HOMEND
NO
TRPCKM
PACK THE BACK NODE INDEXES OF THE TREE
WSL
WRITE THE TREE ON UNIT 50
A 13

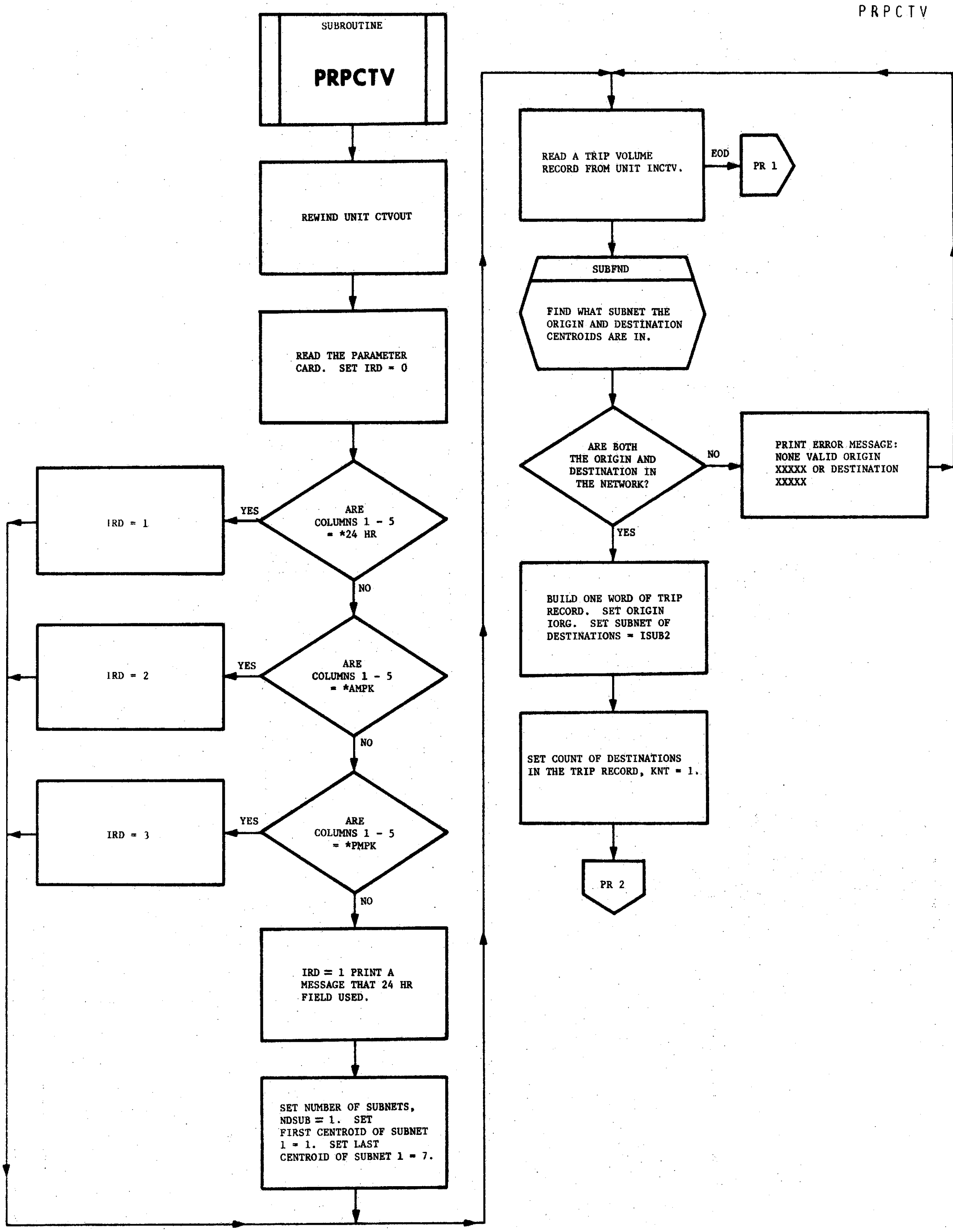

SUBROUTINE
PRPBLD
READ TURN PENALTY CARD. CAPC = .FALSE. W = FALSE
ARE COLUMNS 35 - 38 = 'WGT '?
YES
W = .TRUE. (SET W TO PRODUCE SECOND WEIGHTED ASSIGNMENT)
NO
PRINT CARD READ AS TURN PENALTY CARD
ARE COLUMNS 1 - 4 OF THE *TURN CARD = '*TUR'?
NO
PRINT MESSAGE: INVALID TURN PENALTY OR TREE CARD READ. 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

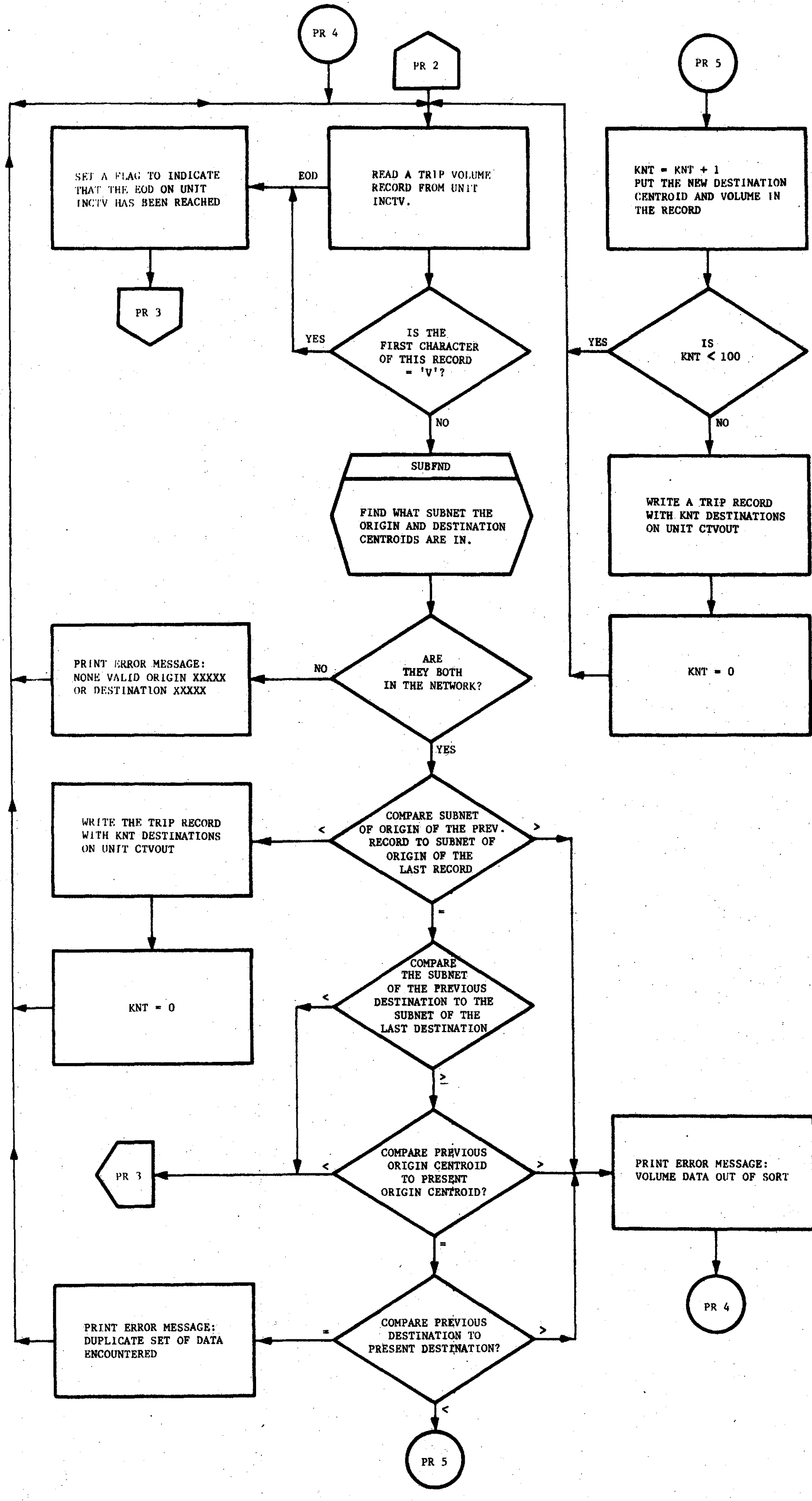

B 1
L 125
IS THE M SUBFIELD A ≤ 0?
YES
B 2
NO
111
HAS THE M'TH SUBFIELD A ALREADY BEEN USED IN AN INCLUSIVE PAIR?
SET SKIP FLAG TO PROCESS NEXT SUBFIELD A
IS THE M SUBFIELD B A COMMA?
KOUNT = KOUNT + 1 SET FLAG FOR NO OUTPUT FOR THE TREE FROM THE CENTROID OF THE SUBFIELD A
IS THE M SUBFIELD B A PERIOD?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

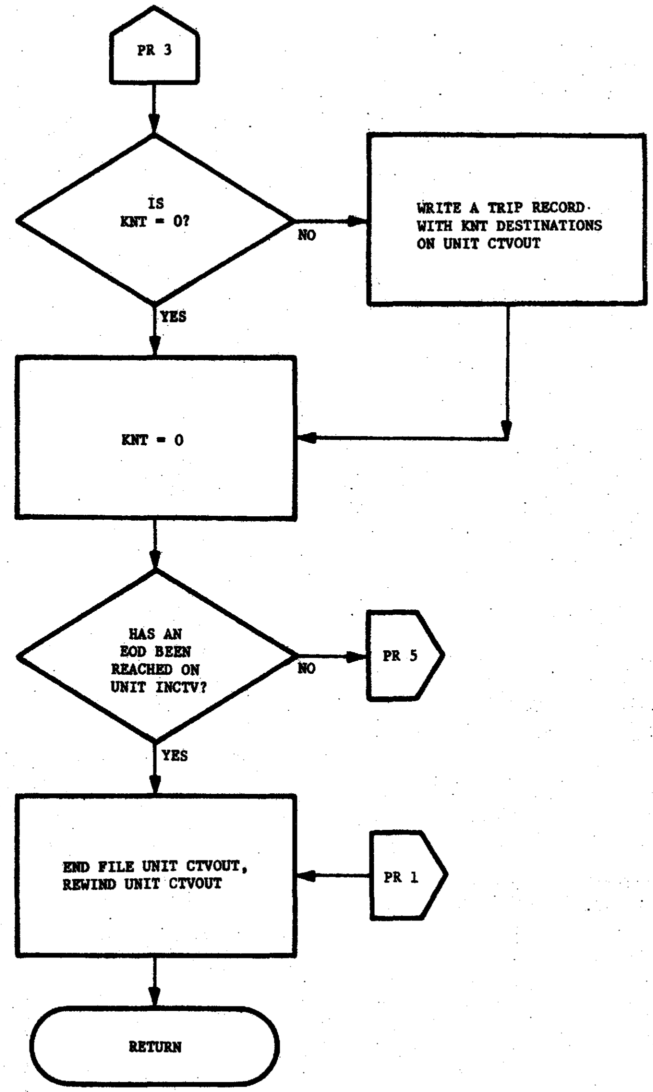
PRPBLD

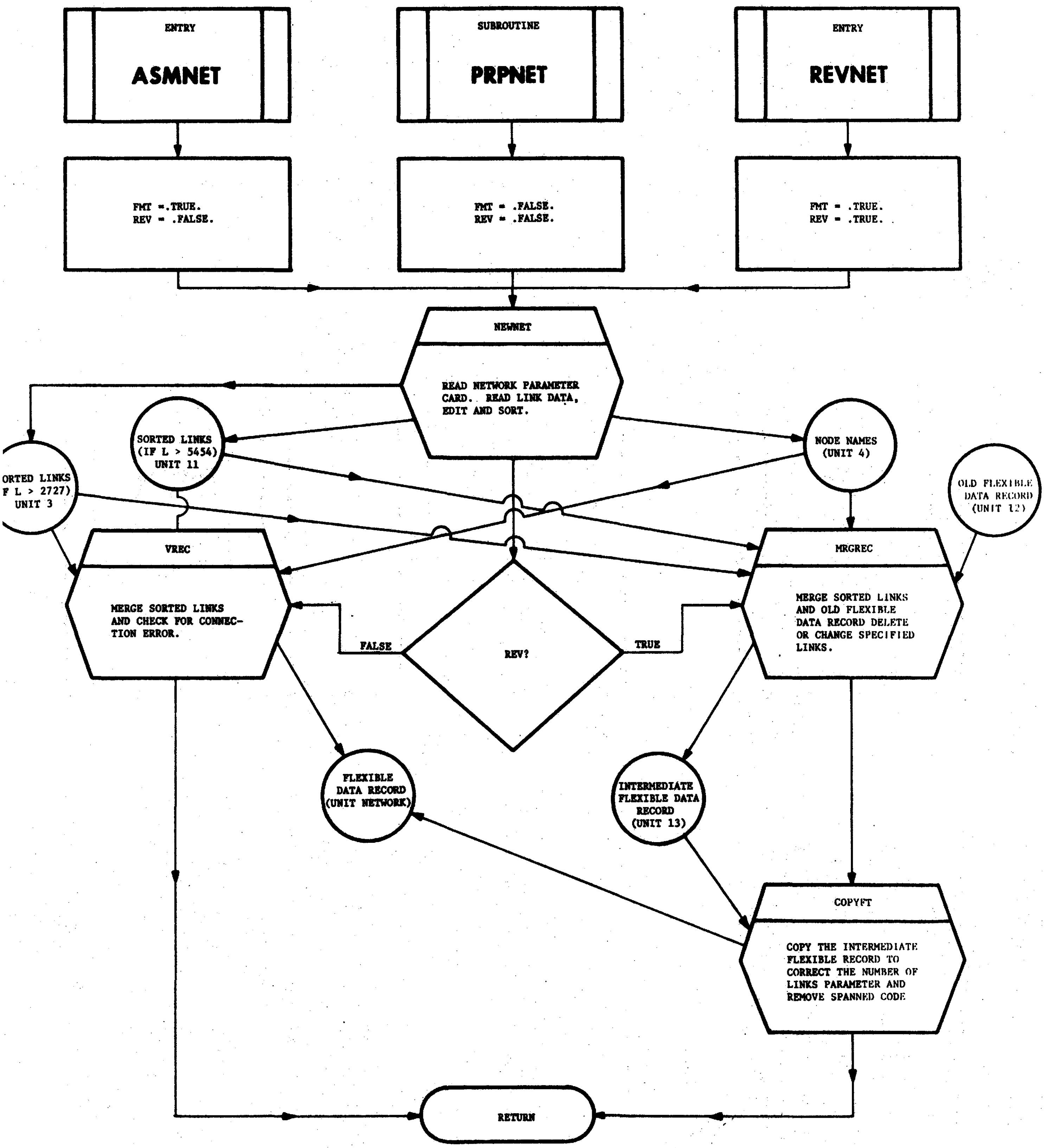


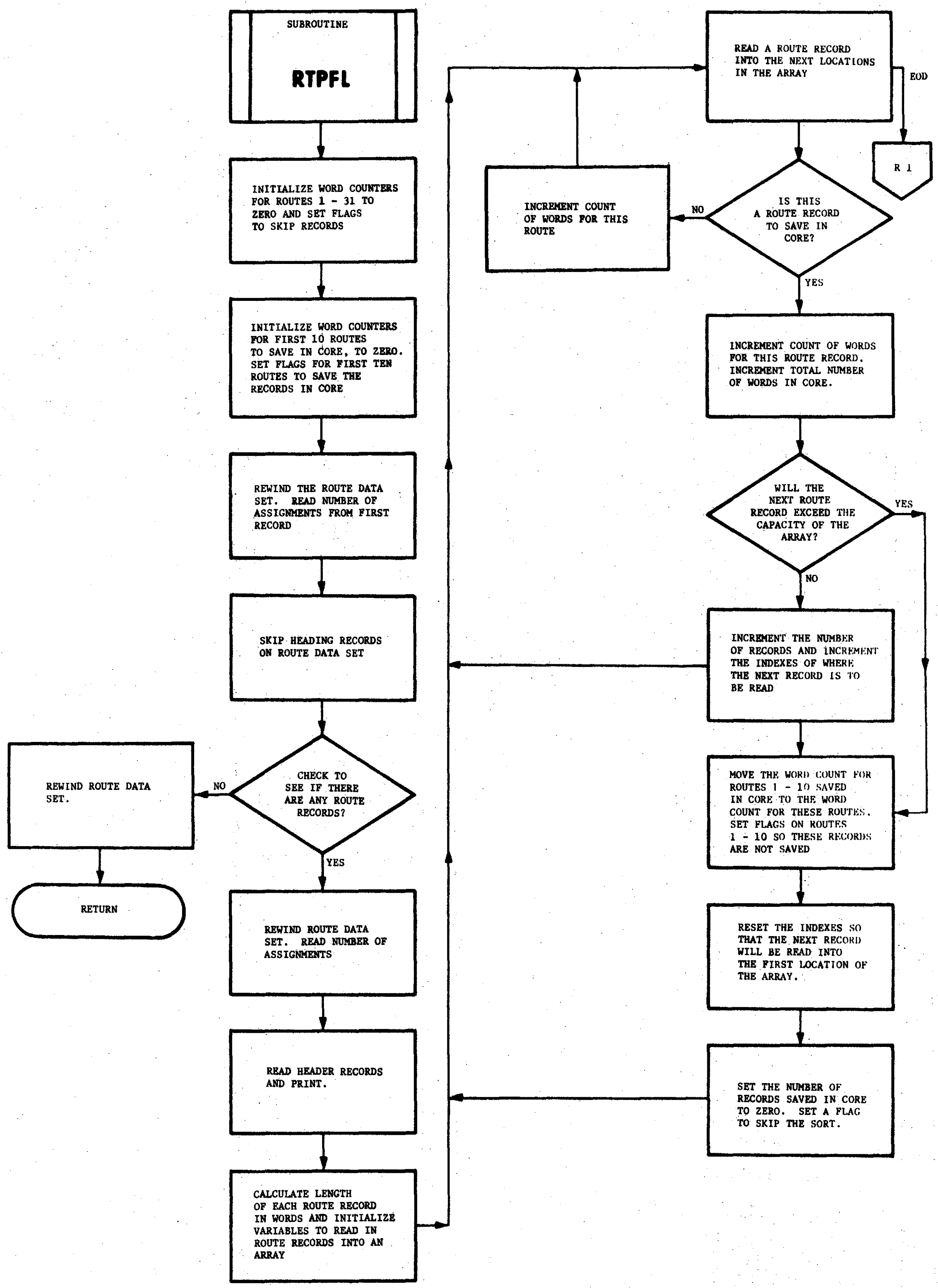
B 2
SAVE NUMBER OF RANGES OF CENTROIDS, KOUNT, FOR SUBNET I
I = I + 1
IS I > NUMBER OF SUBNETS?
NO
B 3
YES
PRINT THE TURN PENALTY AND THE RANGES OF TREES TO BUILD AND WHICH ARE TO BE OUTPUTED
IF THIS AN ASSIGN SELF-BALANCING RUN THEN PRINT WHETHER CAPACITIES OR COUNTS WILL BE USED
ALSO PRINT A MESSAGE 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

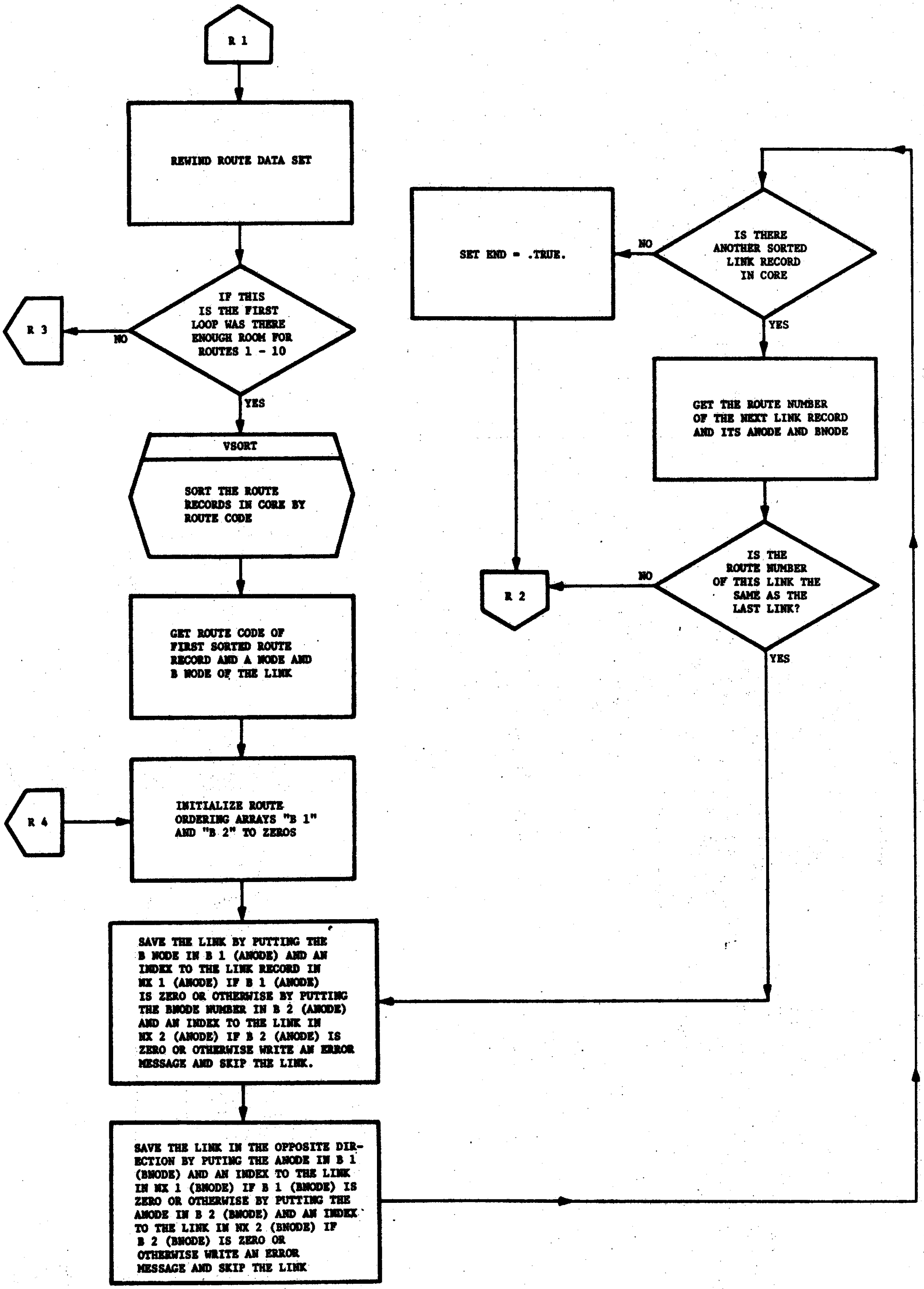

SUBROUTINE
PRPCTV
REWIND UNIT CTVOUT
READ THE PARAMETER CARD. SET IRD = 0
ARE COLUMNS 1 - 5 = *24 HR
YES
IRD = 1
NO
ARE COLUMNS 1 - 5 = *AMPK
YES
IRD = 2
NO
ARE COLUMNS 1 - 5 = *PMPK
YES
IRD = 3
NO
IRD = 1 PRINT A MESSAGE THAT 24 HR FIELD USED.
SET NUMBER OF SUBNETS, NDSUB = 1. SET FIRST CENTROID OF 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

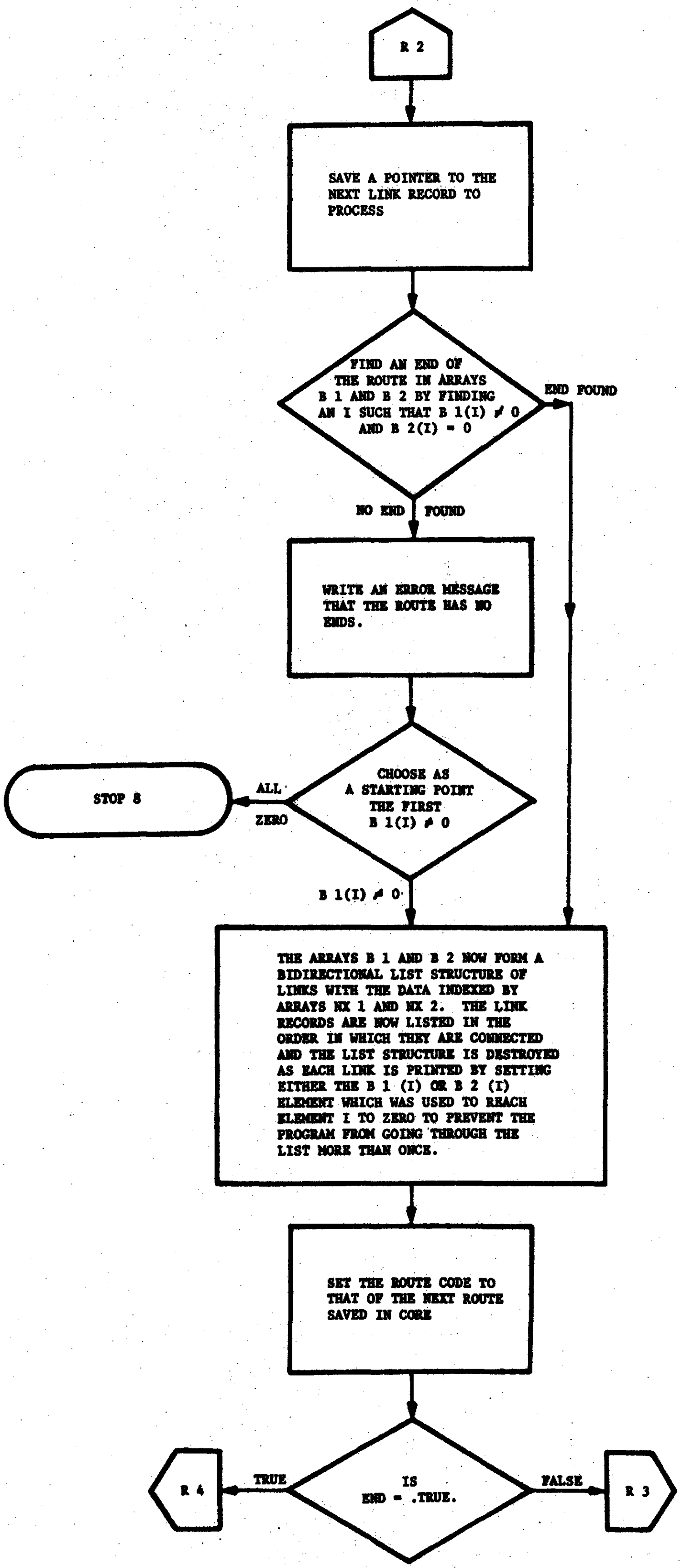

PR 4
PR 2
PR 5
SET A FLAG TO INDICATE THAT THE EOD ON UNIT INCTV HAS BEEN REACHED
EOD
READ A TRIP VOLUME RECORD FROM UNIT INCTV.
KNT = KNT + 1
PUT THE NEW DESTINATION CENTROID AND VOLUME IN THE RECORD
PR 3
YES
IS THE FIRST CHARACTER OF THIS RECORD = 'V'?
YES
IS KNT < 100
NO
NO
SUBFND
FIND WHAT SUBNET 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

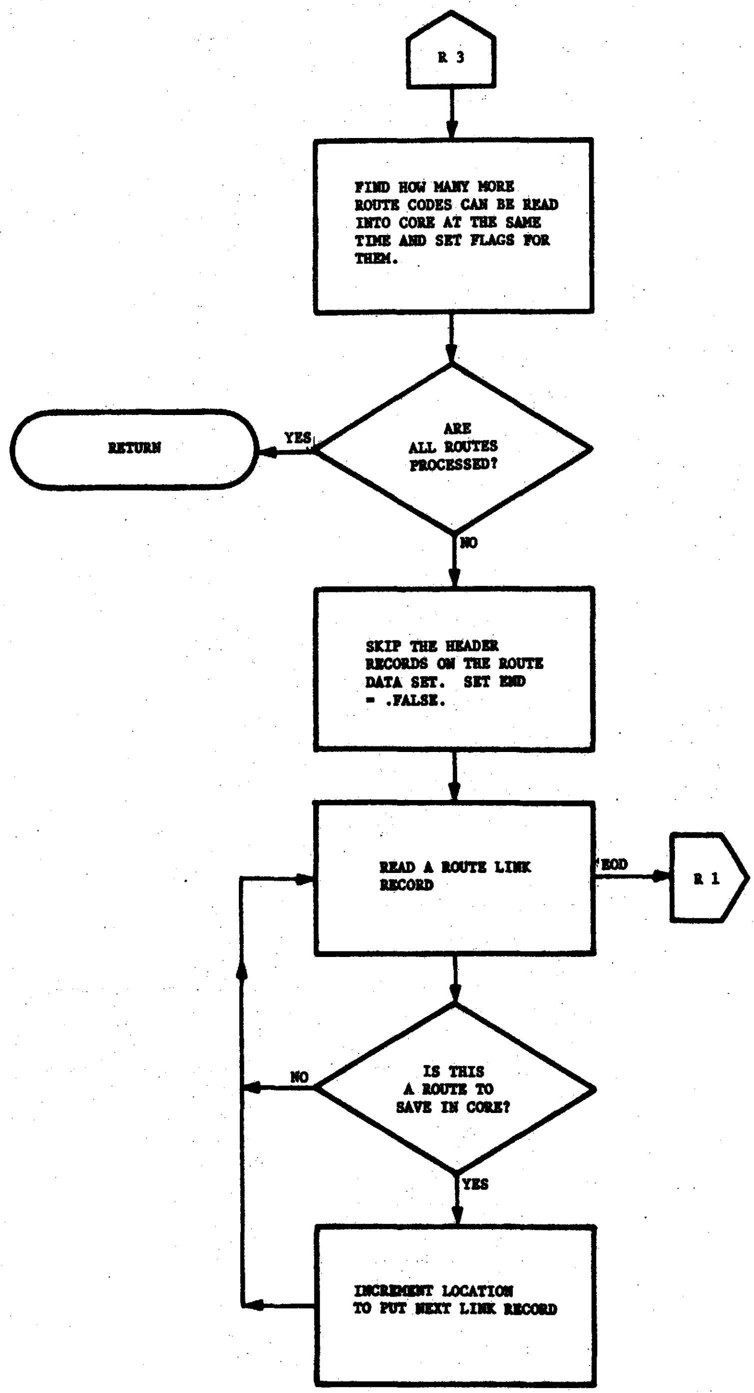

PR 3
IS
KNT = 0?
NO
WRITE A TRIP RECORD WITH KNT DESTINATIONS ON UNIT CTVOUT
YES
KNT = 0
HAS AN EOD BEEN REACHED ON UNIT INCTV?
NO
PR 5
YES
END FILE UNIT CTVOUT, REWIND UNIT CTVOUT
PR 1
RETURN

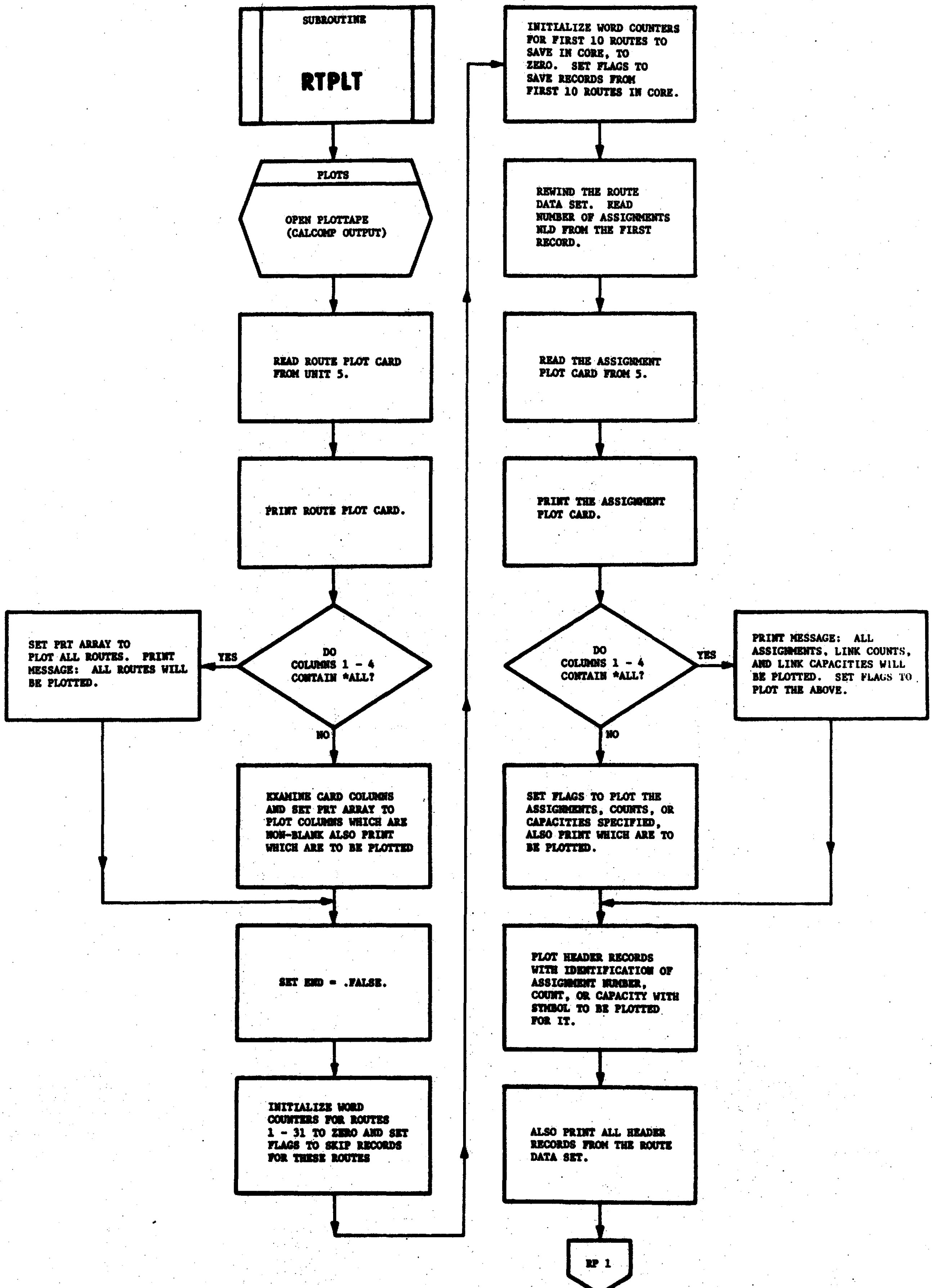

ENTRY
ASMNET
SUBROUTINE
PRPNET
ENTRY
REVNET
FMT =.TRUE.
REV = .FALSE.
FMT = .FALSE.
REV = .FALSE.
FMT = .TRUE.
REV = .TRUE.
NEWNET
READ NETWORK PARAMETER CARD. READ LINK DATA, EDIT AND SORT.
SORTED LINKS (IF L > 5454) UNIT 11
ORTED LINKS F L > 2727) UNIT 3
NODE NAMES (UNIT 4)
OLD FLEXIBLE DATA RECORD (UNIT 12)
VREC
MERGE SORTED LINKS AND CHECK FOR CONNECTION ERROR.
REV?
FALSE
TRUE
MRGREC
MERGE SORTED LINKS AND OLD FLEXIBLE DATA RECORD DELETE OR CHANGE SPECIFIED LINKS.
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

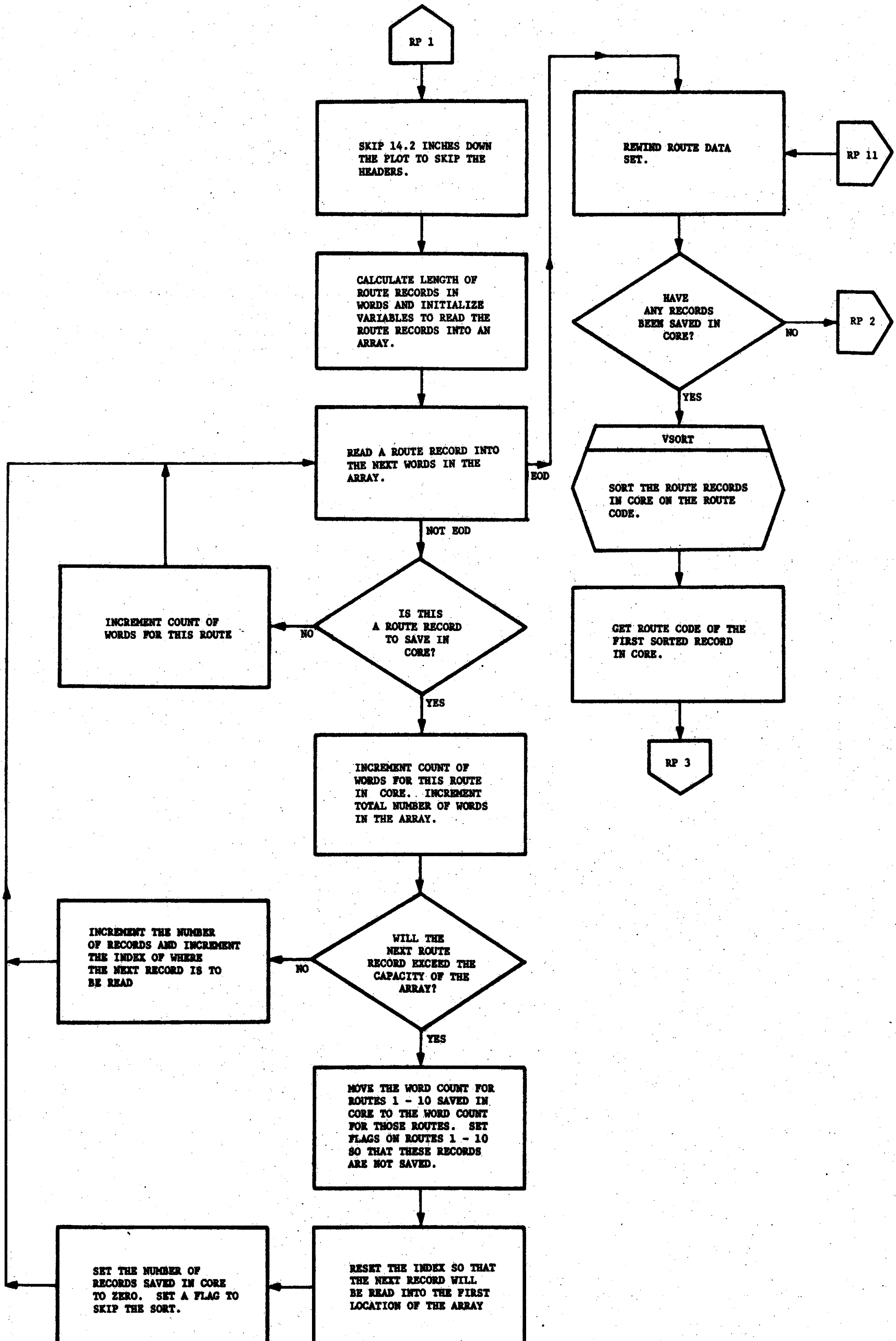

SUBROUTINE
RTPFL
INITIALIZE WORD COUNTERS FOR ROUTES 1 - 31 TO ZERO AND SET FLAGS TO SKIP RECORDS
INITIALIZE WORD COUNTERS FOR FIRST 10 ROUTES TO SAVE IN CORE, TO ZERO. SET FLAGS FOR FIRST TEN ROUTES TO SAVE THE RECORDS IN CORE
REWIND THE ROUTE DATA SET. READ NUMBER OF ASSIGNMENTS FROM FIRST RECORD
SKIP HEADING RECORDS ON ROUTE DATA SET
CHECK TO SEE IF THERE ARE ANY ROUTE RECORDS?
NO
YES
REWIND ROUTE DATA SET.
RETURN
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
INCREMENT COUNT OF WORDS FOR THIS ROUTE
NO
IS THIS A ROUTE RECORD TO SAVE IN CORE?
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.

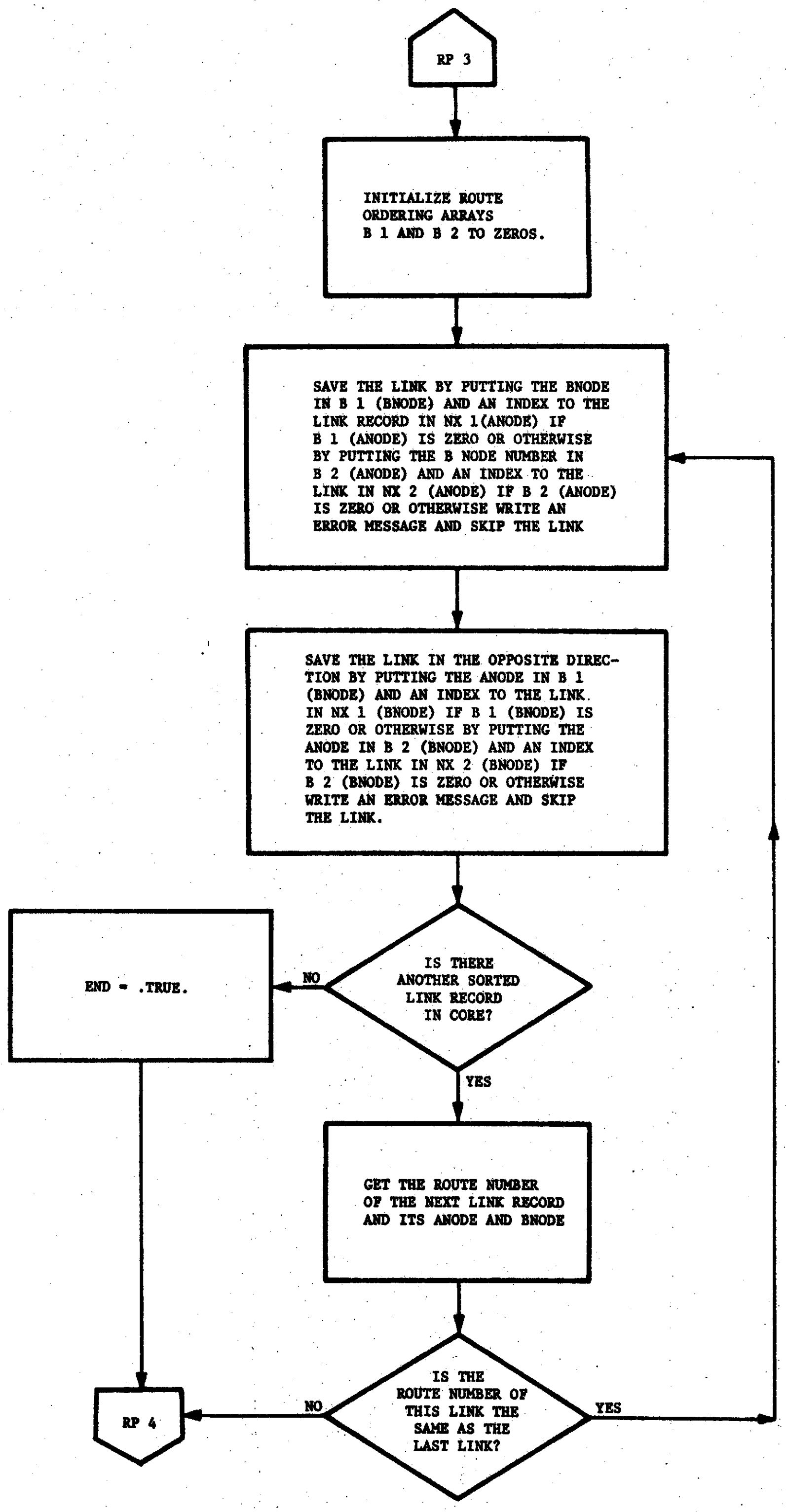

R 1
REWIND ROUTE DATA SET
IF THIS IS THE FIRST LOOP WAS THERE ENOUGH ROOM FOR ROUTES 1 - 10
NO
R 3
YES
VSORT
SORT THE ROUTE RECORDS IN CORE BY ROUTE CODE
GET ROUTE CODE OF FIRST SORTED ROUTE RECORD AND A NODE AND B NODE OF THE LINK
R 4
INITIALIZE ROUTE ORDERING ARRAYS "B 1" AND "B 2" TO ZEROS
SAVE THE 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 PUTING 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
SET END = .TRUE.
IS THERE ANOTHER SORTED LINK RECORD IN CORE
NO
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
R 2
YES

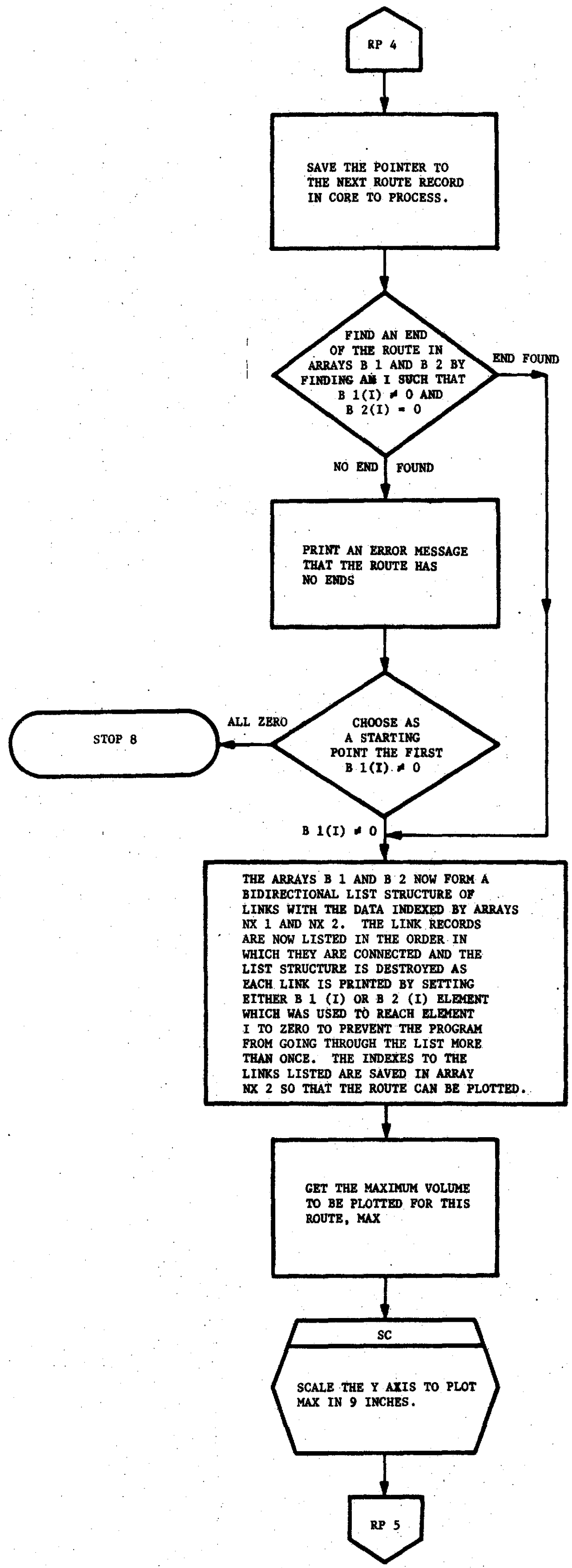

R 2
SAVE A POINTER TO THE NEXT LINK RECORD TO PROCESS
FIND AN END OF THE ROUTE IN ARRAYS B 1 AND B 2 BY FINDING AN I SUCH THAT B 1(I) ≠ 0 AND B 2(I) = 0
END FOUND
NO END FOUND
WRITE AN ERROR MESSAGE THAT THE ROUTE HAS NO ENDS.
CHOOSE AS A STARTING POINT THE FIRST B 1(I) ≠ 0
ALL ZERO
STOP 8
B 1(I) ≠ 0
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
R 4
FALSE
R 3

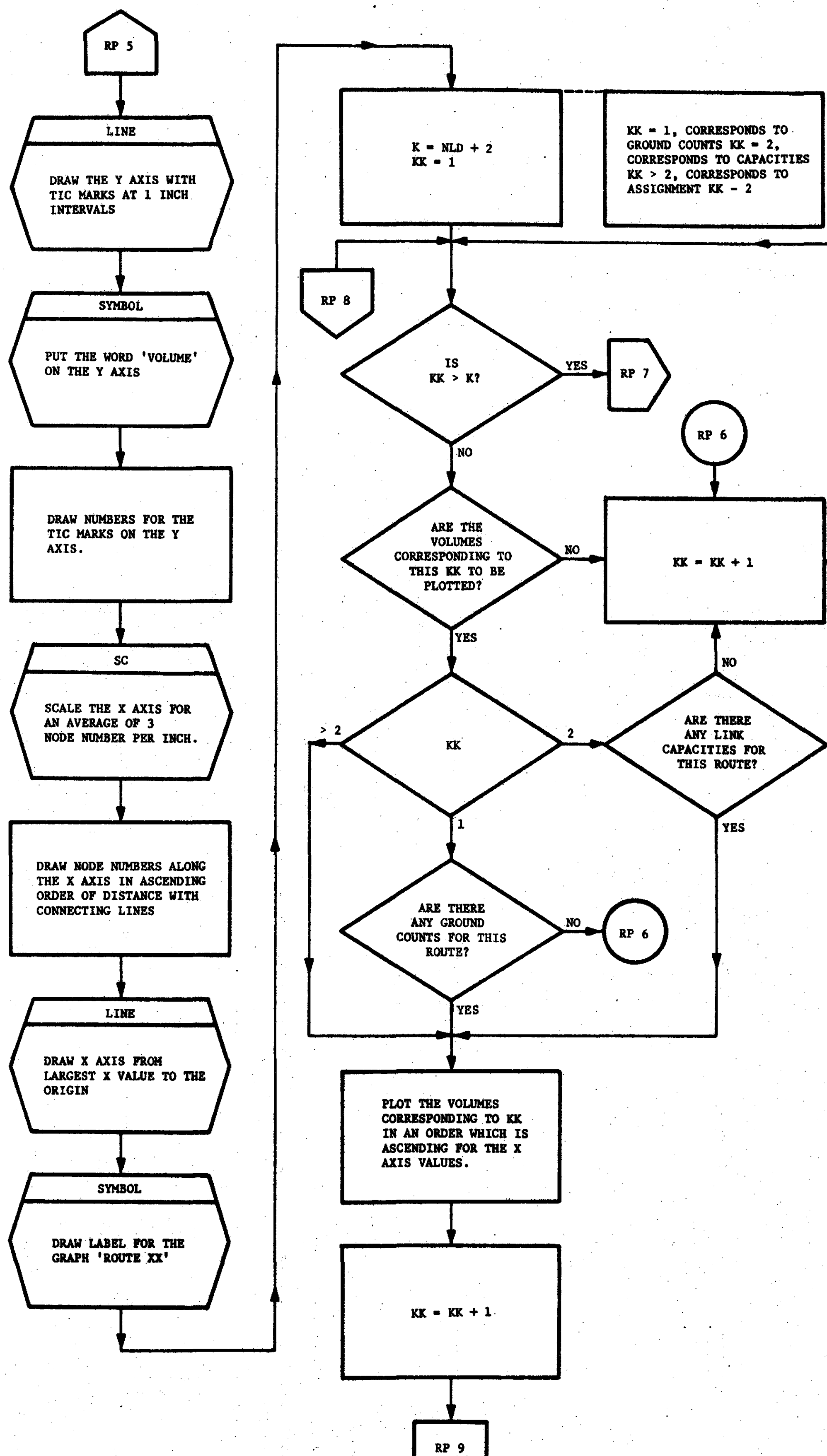

R 3
FIND HOW MANY MORE ROUTE CODES CAN BE READ INTO CORE AT THE SAME TIME AND SET FLAGS FOR THEM.
ARE ALL ROUTES PROCESSED?
YES
RETURN
NO
SKIP THE HEADER RECORDS ON THE ROUTE DATA SET. SET END = .FALSE.
READ A ROUTE LINK RECORD
EOD
R 1
IS THIS A ROUTE TO SAVE IN CORE?
NO
YES
INCREMENT LOCATION TO PUT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

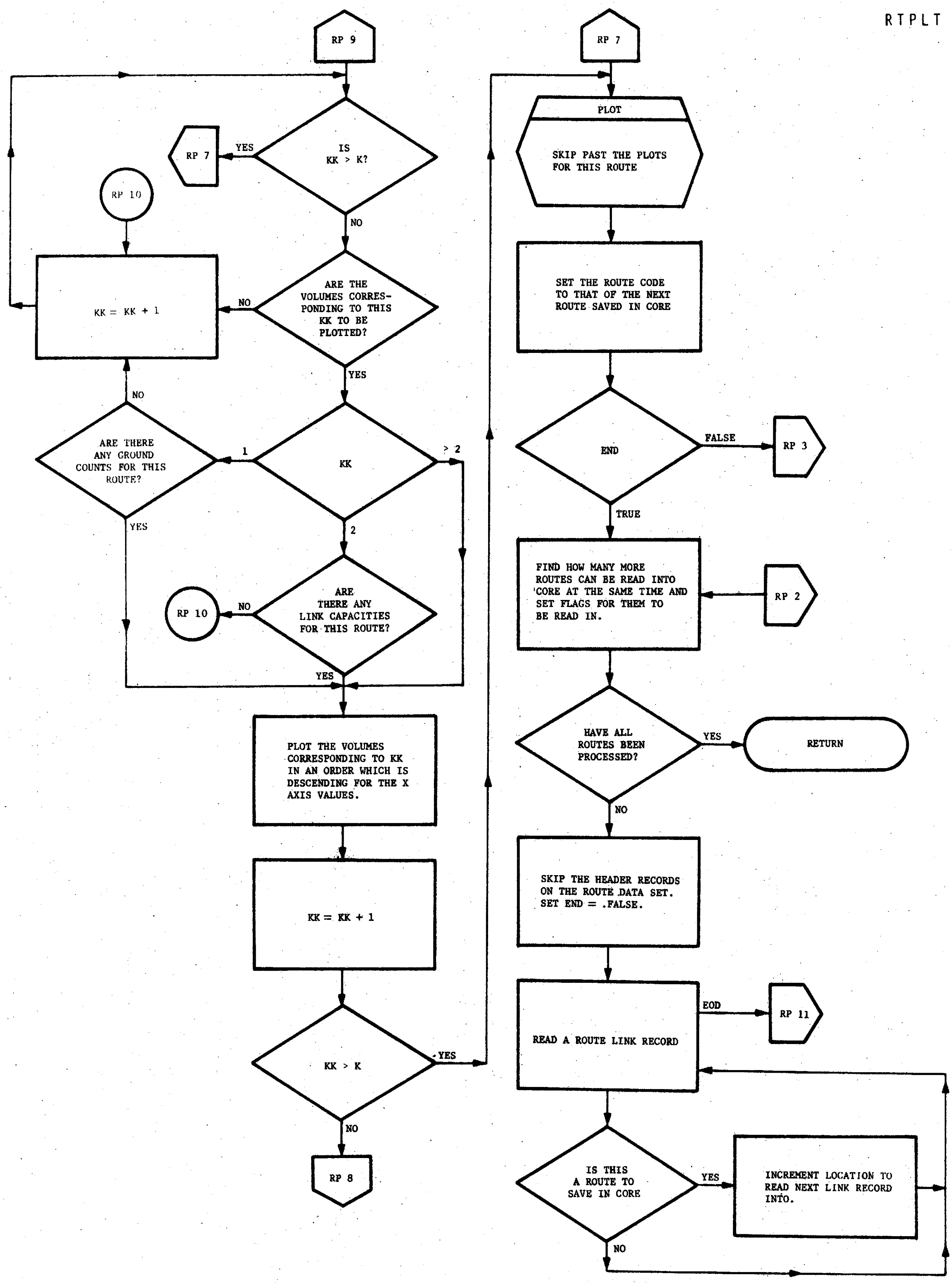

SUBROUTINE
RTPLT
PLOTS
OPEN PLOTTAPE (CALCOMP OUTPUT)
READ ROUTE PLOT CARD FROM UNIT 5.
PRINT ROUTE PLOT CARD.
DO COLUMNS 1 - 4 CONTAIN *ALL?
YES
SET PRT ARRAY TO PLOT ALL ROUTES. PRINT MESSAGE: ALL ROUTES WILL BE PLOTTED.
NO
EXAMINE CARD COLUMNS AND SET PRT ARRAY TO PLOT COLUMNS WHICH ARE NON-BLANK 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

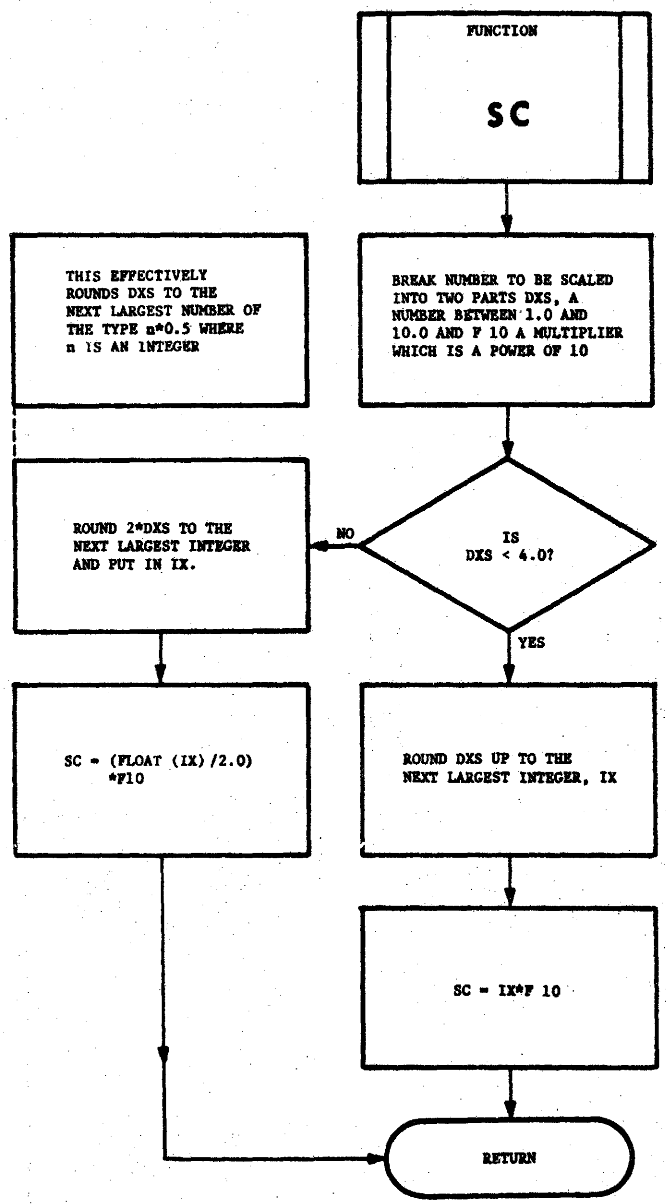

RP 1
SKIP 14.2 INCHES DOWN THE PLOT TO SKIP THE HEADERS.
CALCULATE LENGTH OF ROUTE RECORDS IN WORDS AND INITIALIZE VARIABLES TO READ THE ROUTE RECORDS INTO AN ARRAY.
READ A ROUTE RECORD INTO THE NEXT WORDS IN THE ARRAY.
EOD
NOT EOD
IS THIS A ROUTE RECORD TO SAVE IN CORE?
NO
INCREMENT COUNT OF WORDS FOR 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

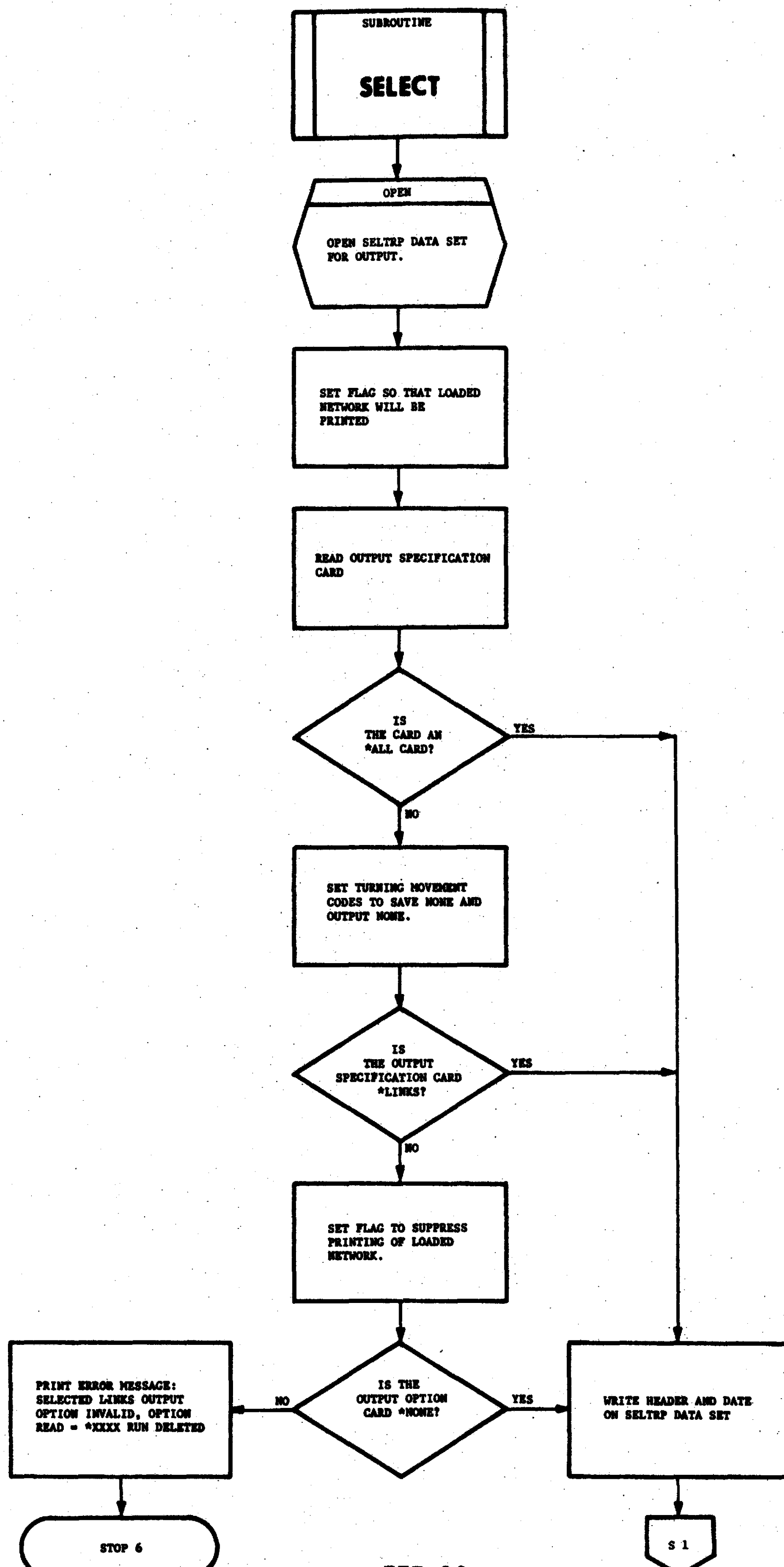

RP 3
INITIALIZE ROUTE ORDERING ARRAYS B 1 AND B 2 TO ZEROS.
SAVE THE LINK BY PUTTING THE BNODE IN B 1 (BNODE) AND AN INDEX TO THE LINK RECORD IN NX 1(ANODE) IF B 1 (ANODE) IS ZERO OR OTHERWISE BY PUTTING THE B NODE NUMBER IN B 2 (ANODE) AND AN INDEX TO THE LINK IN NX 2 (ANODE) IF B 2 (ANODE) IS ZERO 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

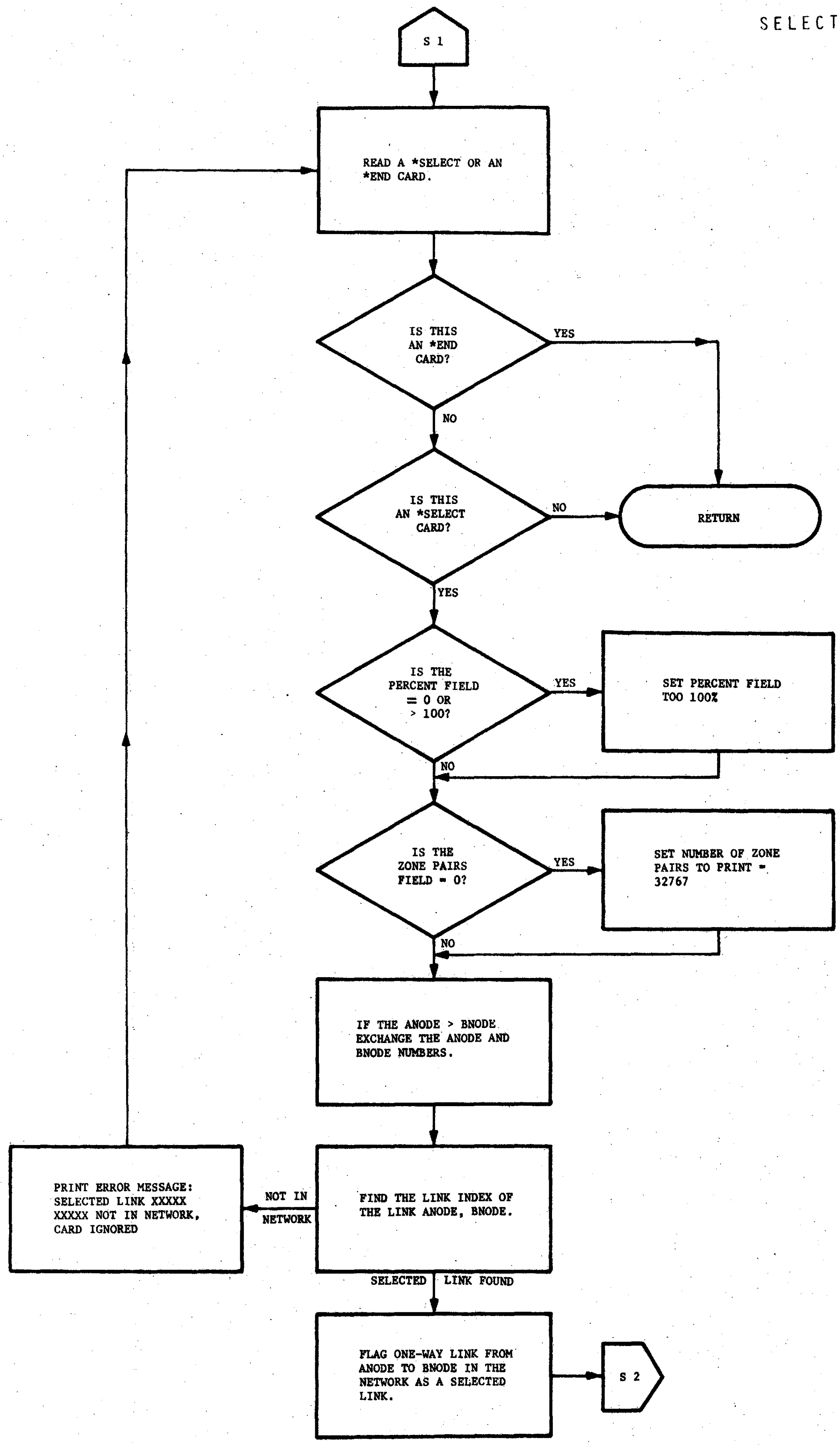

RP 4
SAVE THE POINTER TO THE NEXT ROUTE RECORD IN CORE TO PROCESS.
FIND AN END OF THE ROUTE IN ARRAYS B 1 AND B 2 BY FINDING AN I SUCH THAT B 1(I) ≠ 0 AND B 2(I) = 0
END FOUND
NO END FOUND
PRINT AN ERROR MESSAGE THAT THE ROUTE HAS NO ENDS
CHOOSE AS A STARTING POINT THE FIRST B 1(I) ≠ 0
ALL ZERO
STOP 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

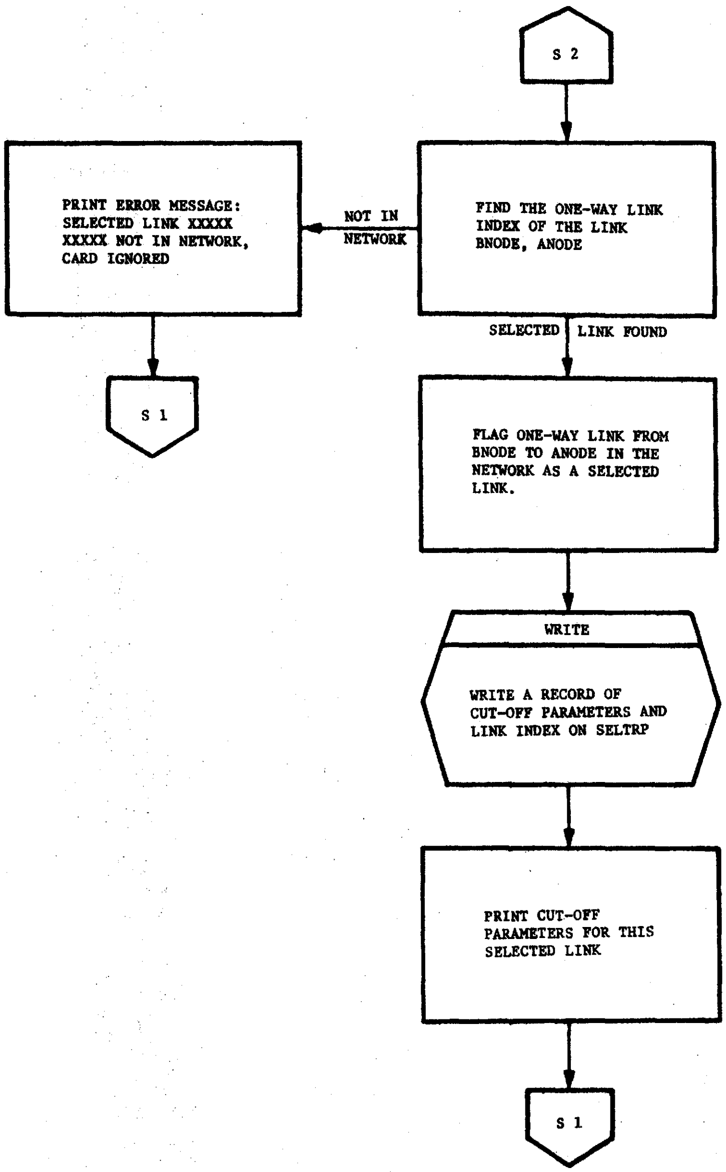

RP 5
LINE
DRAW THE Y AXIS WITH TIC MARKS AT 1 INCH INTERVALS
SYMBOL
PUT THE WORD 'VOLUME' ON THE Y AXIS
DRAW NUMBERS FOR THE TIC MARKS ON THE Y AXIS.
SC
SCALE THE X AXIS FOR AN AVERAGE OF 3 NODE NUMBER PER INCH.
DRAW NODE NUMBERS ALONG THE X AXIS IN ASCENDING ORDER OF DISTANCE WITH CONNECTING LINES
LINE
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
> 2
KK
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

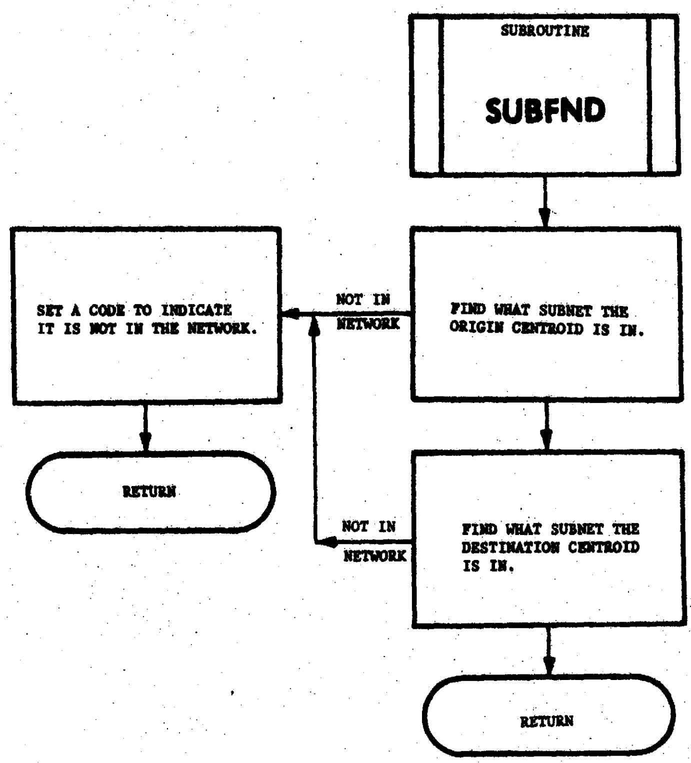

RP 9
IS KK > K?
YES
RP 7
NO
RP 10
ARE THE VOLUMES CORRES-PONDING TO THIS KK TO BE PLOTTED?
NO
KK = KK + 1
YES
KK
1
> 2
2
NO
ARE THERE ANY GROUND COUNTS FOR THIS ROUTE?
YES
ARE THERE ANY LINK CAPACITIES FOR THIS ROUTE?
NO
RP 10
YES
PLOT THE VOLUMES CORRESPONDING TO KK IN AN ORDER WHICH IS DESCENDING FOR 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

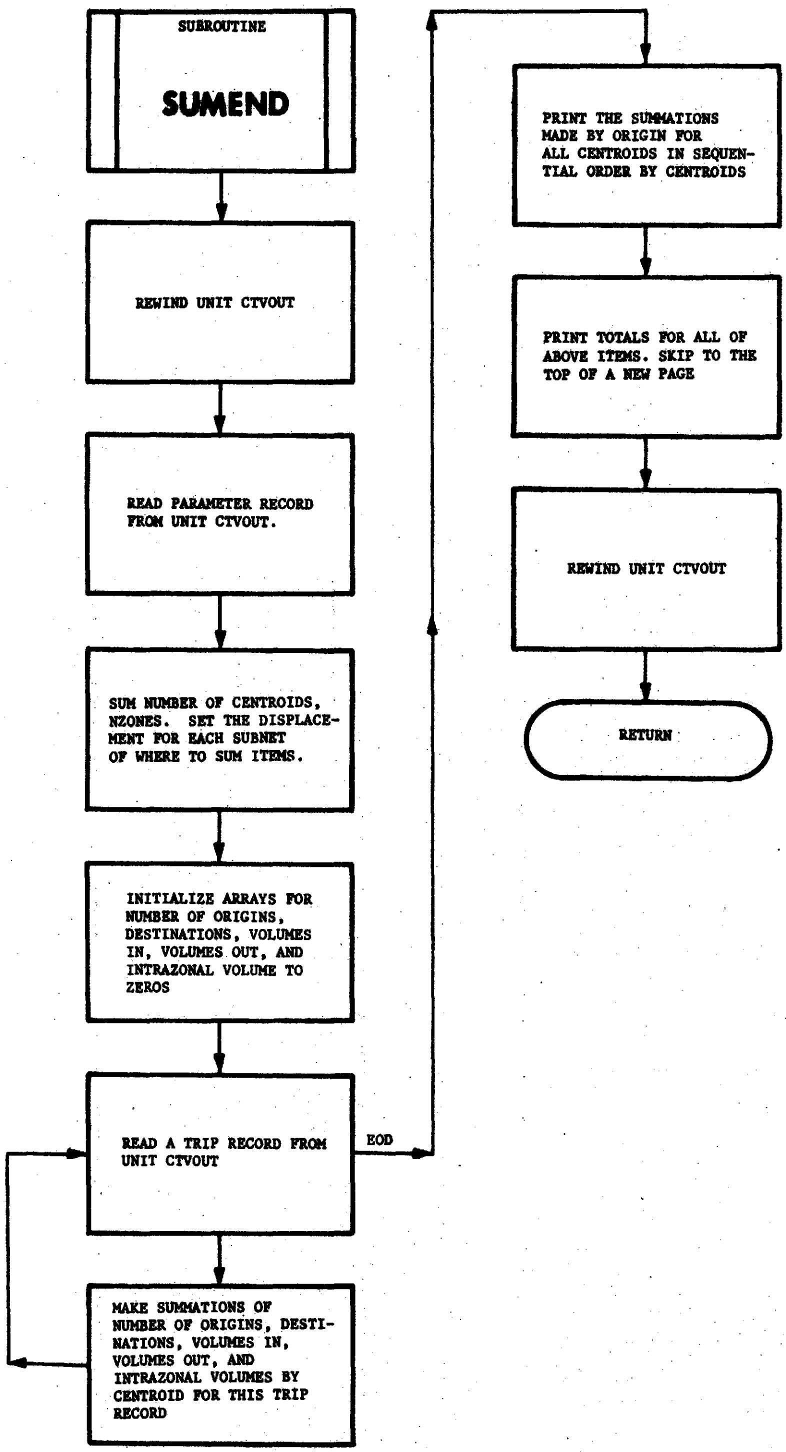

FUNCTION
SC
BREAK NUMBER TO BE SCALED INTO TWO PARTS DXS, A NUMBER BETWEEN 1.0 AND 10.0 AND F 10 A MULTIPLIER WHICH IS A POWER OF 10
THIS EFFECTIVELY ROUNDS DXS TO THE NEXT LARGEST NUMBER OF THE TYPE n*0.5 WHERE n IS AN INTEGER
IS DXS < 4.0?
NO
YES
ROUND 2*DXS TO THE NEXT LARGEST INTEGER AND PUT IN IX.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

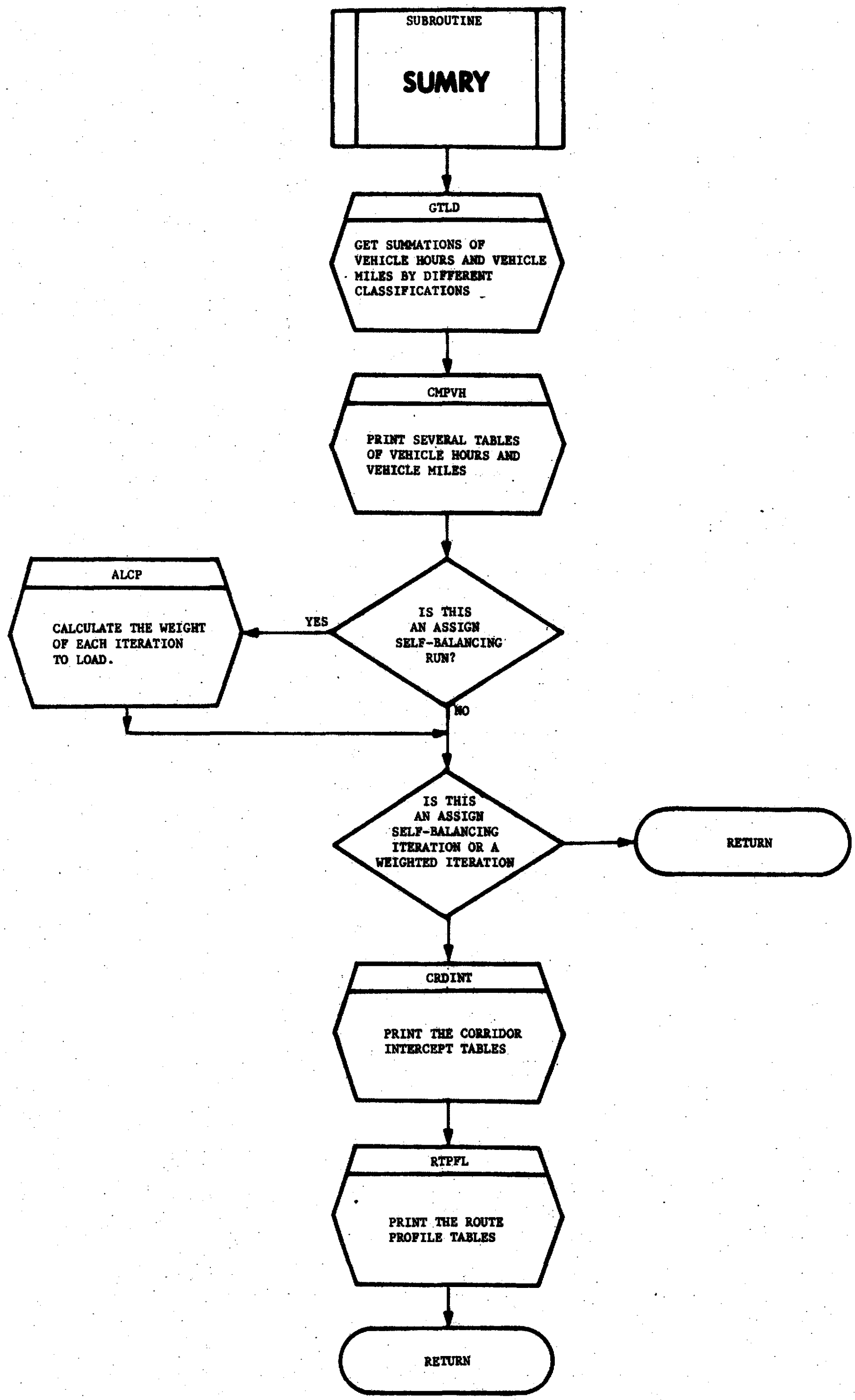

SUBROUTINE
SELECT
OPEN
OPEN SELTRP DATA SET FOR OUTPUT.
SET FLAG SO THAT LOADED NETWORK WILL BE PRINTED
READ OUTPUT SPECIFICATION CARD
IS THE CARD AN *ALL CARD?
YES
NO
SET TURNING MOVEMENT CODES TO SAVE NONE AND OUTPUT NONE.
IS THE OUTPUT SPECIFICATION CARD *LINKS?
YES
NO
SET FLAG TO SUPPRESS PRINTING 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

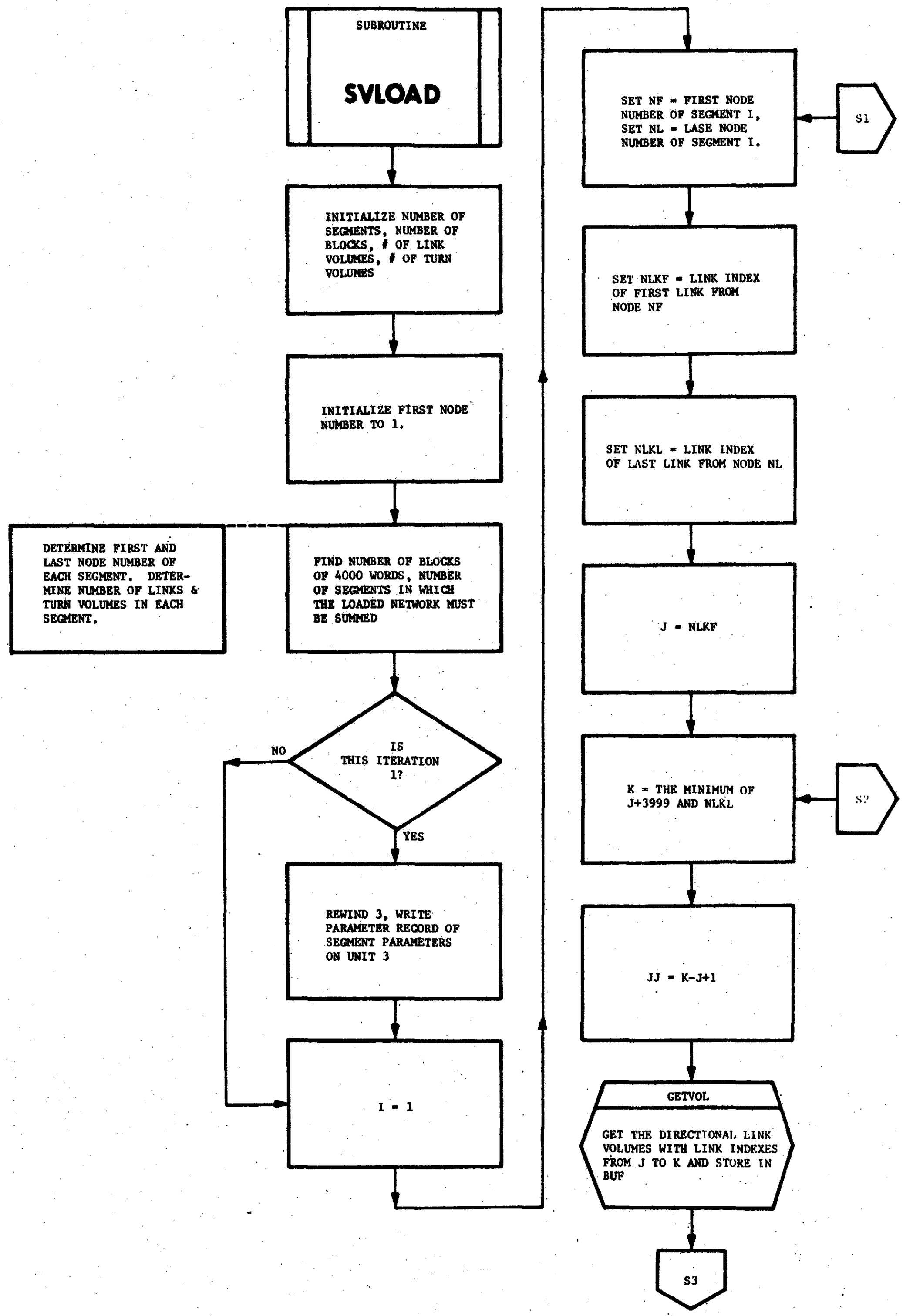

S 1
READ A *SELECT OR AN *END CARD.
IS THIS AN *END CARD?
YES
NO
IS THIS AN *SELECT CARD?
NO
RETURN
YES
IS THE PERCENT FIELD = 0 OR > 100?
YES
SET PERCENT FIELD TOO 100%
NO
IS THE ZONE PAIRS FIELD = 0?
YES
SET NUMBER OF ZONE PAIRS TO PRINT = 32767
NO
IF THE ANODE > BNODE EXCHANGE THE ANODE AND BNODE 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

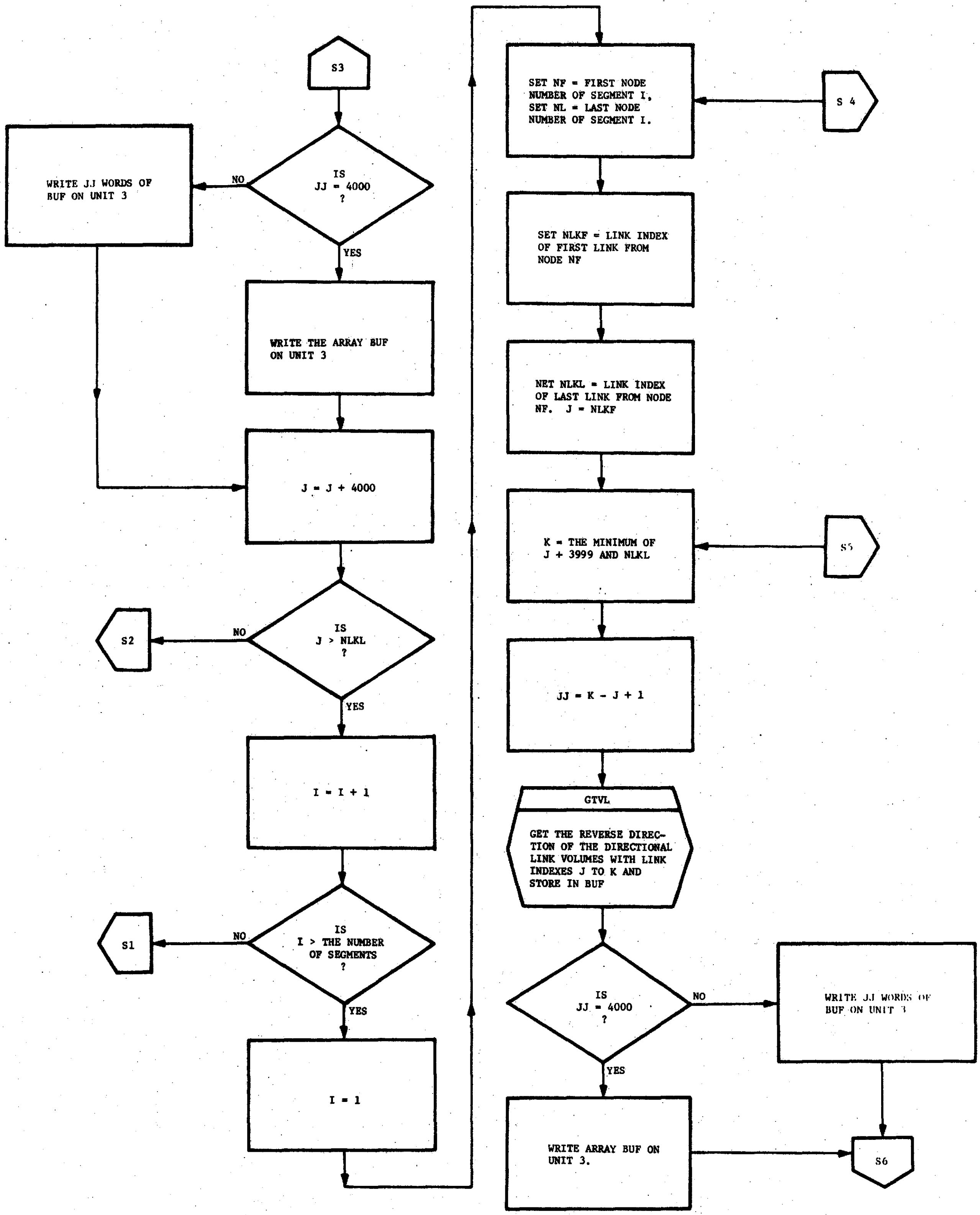

S 2
FIND THE ONE-WAY LINK INDEX OF THE LINK BNODE, ANODE
NOT IN NETWORK
PRINT ERROR MESSAGE: SELECTED LINK XXXXX XXXXX NOT IN NETWORK, CARD IGNORED
S 1
SELECTED LINK FOUND
FLAG ONE-WAY LINK FROM BNODE TO ANODE IN THE NETWORK AS A SELECTED LINK.
WRITE
WRITE A RECORD OF CUT-OFF PARAMETERS AND LINK INDEX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

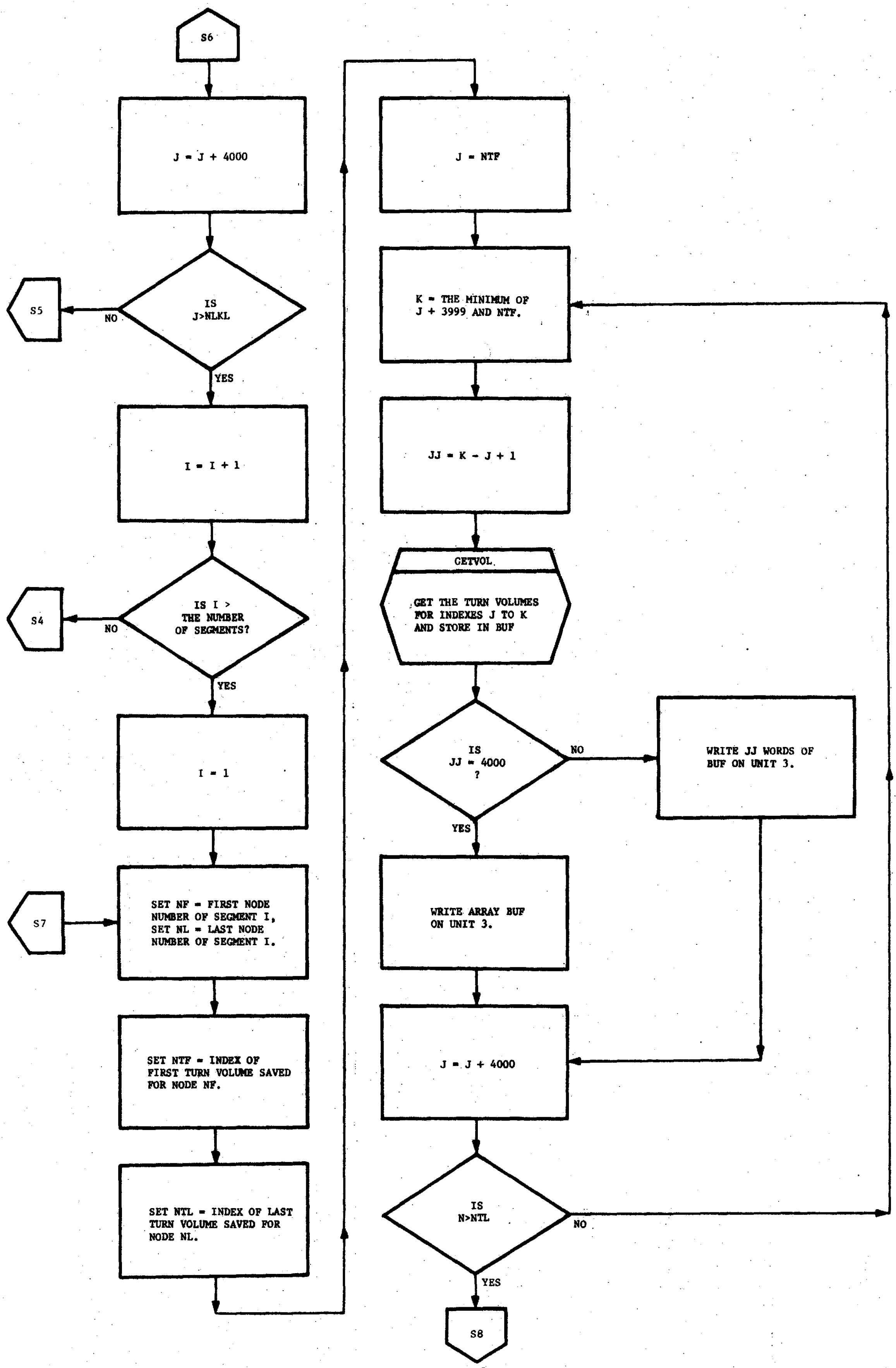

SUBROUTINE
SUBFND
FIND WHAT SUBNET THE ORIGIN CENTROID IS IN.
NOT IN NETWORK
SET A CODE TO INDICATE IT IS NOT IN THE NETWORK.
RETURN
FIND WHAT SUBNET THE DESTINATION CENTROID IS IN.
NOT IN NETWORK
RETURN

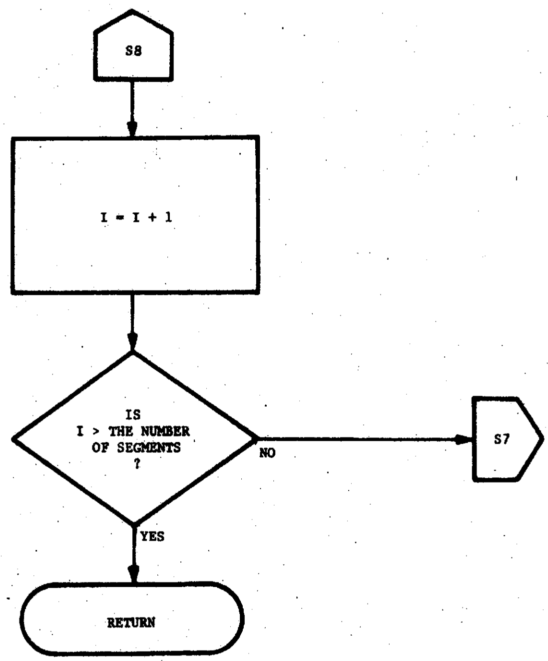

SUBROUTINE
SUMEND
REWIND UNIT CTVOUT
READ PARAMETER RECORD FROM UNIT CTVOUT.
SUM NUMBER OF CENTROIDS, NZONES. SET THE DISPLACEMENT FOR EACH SUBNET OF WHERE TO SUM ITEMS.
INITIALIZE ARRAYS FOR NUMBER OF ORIGINS, DESTINATIONS, VOLUMES IN, VOLUMES OUT, AND INTRAZONAL VOLUME TO ZEROS
READ A TRIP RECORD FROM 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

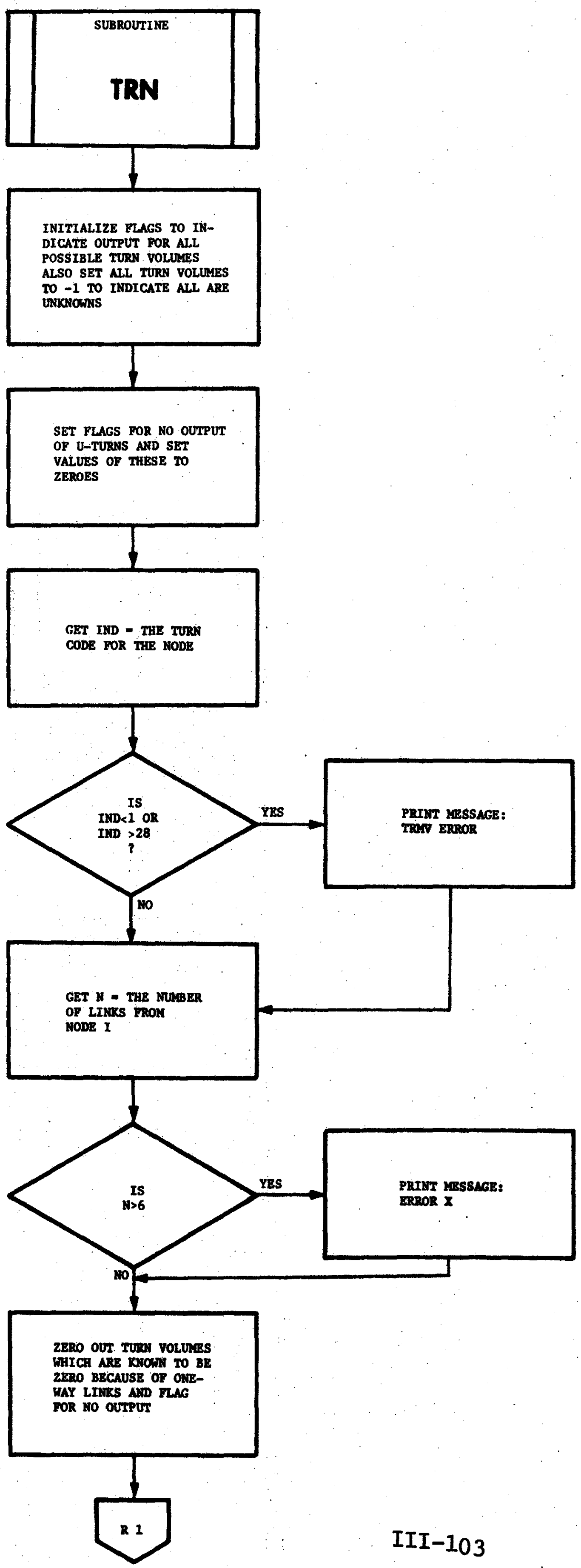

SUBROUTINE
SUMRY
GTLD
GET SUMMATIONS OF VEHICLE HOURS AND VEHICLE MILES BY DIFFERENT CLASSIFICATIONS
CMPVH
PRINT SEVERAL TABLES OF VEHICLE HOURS AND VEHICLE MILES
IS THIS AN ASSIGN SELF-BALANCING RUN?
YES
NO
ALCP
CALCULATE THE WEIGHT OF EACH ITERATION TO LOAD.
IS THIS AN ASSIGN SELF-BALANCING ITERATION OR A WEIGHTED ITERATION
RETURN
CRDINT
PRINT THE CORRIDOR INTERCEPT TABLES
RTPFL
PRINT THE ROUTE PROFILE TABLES
RETURN

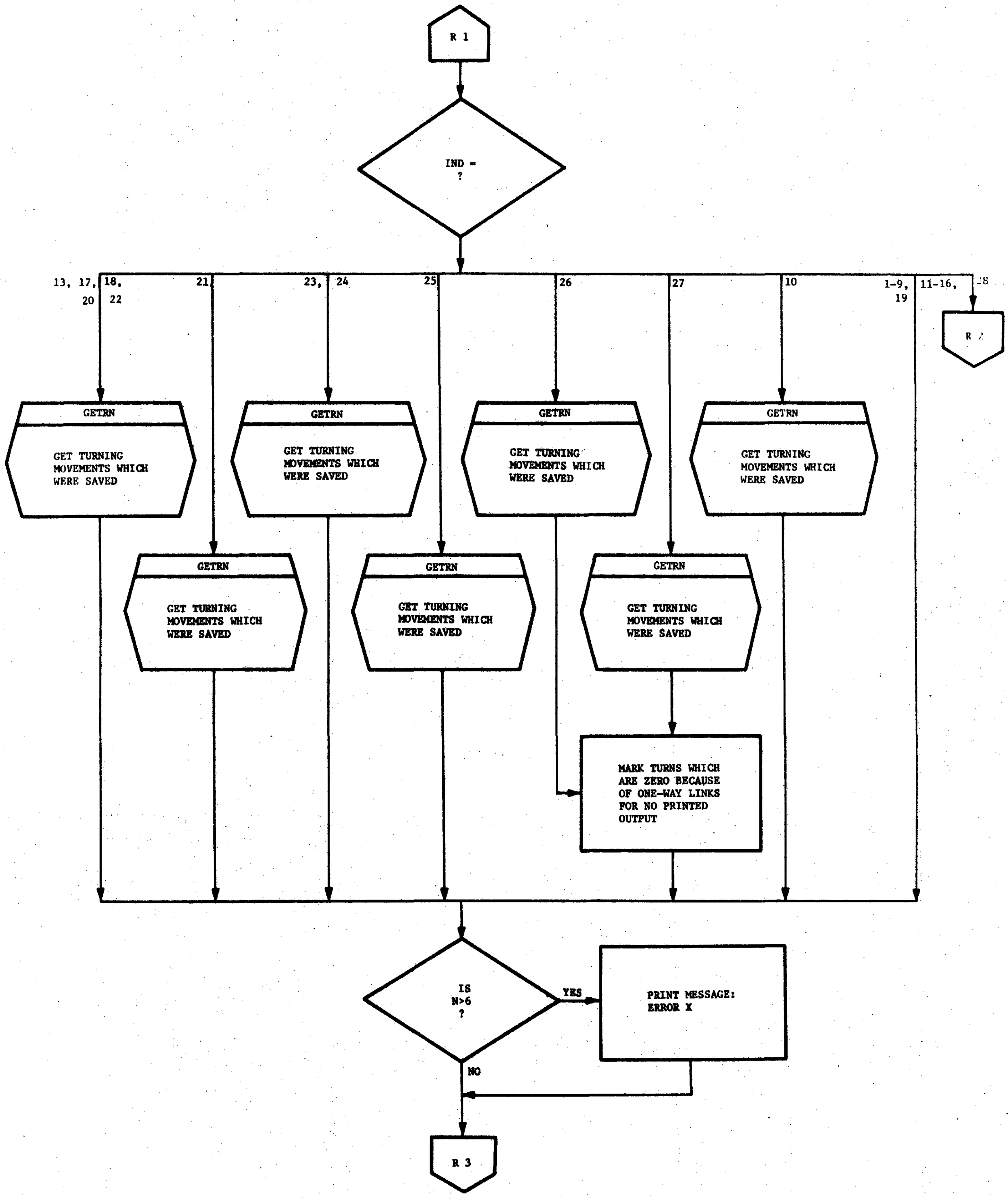

SUBROUTINE
SVLOAD
INITIALIZE NUMBER OF SEGMENTS, NUMBER OF BLOCKS, # OF LINK VOLUMES, # OF TURN VOLUMES
INITIALIZE FIRST NODE NUMBER TO 1.
DETERMINE FIRST AND LAST NODE NUMBER OF EACH SEGMENT. DETERMINE NUMBER OF LINKS & TURN VOLUMES IN EACH SEGMENT.
FIND NUMBER OF BLOCKS OF 4000 WORDS, NUMBER OF SEGMENTS IN WHICH THE LOADED NETWORK MUST BE SUMMED
IS THIS ITERATION 1?
NO
YES
REWIND 3, WRITE PARAMETER RECORD OF SEGMENT PARAMETERS ON UNIT 3
I = 1
SET NF = FIRST NODE NUMBER OF SEGMENT I, SET NL = LASE NODE NUMBER OF SEGMENT I.
S1
SET NLKF = LINK INDEX OF FIRST LINK FROM NODE NF
SET NLKL = LINK INDEX OF LAST LINK FROM NODE NL
J = NLKF
K = THE MINIMUM OF J+3999 AND NLKL
S2
JJ = K-J+1
GETVOL
GET THE DIRECTIONAL LINK VOLUMES WITH LINK INDEXES FROM J TO K AND STORE IN BUF
S3


S3
IS JJ = 4000 ?
NO
WRITE JJ WORDS OF BUF ON UNIT 3
YES
WRITE THE ARRAY BUF ON UNIT 3
J = J + 4000
IS J > NLKL ?
NO
S2
YES
I = I + 1
IS I > THE NUMBER OF SEGMENTS ?
NO
S1
YES
I = 1
SET NF = FIRST NODE NUMBER OF SEGMENT I, SET NL = LAST NODE NUMBER OF SEGMENT I.
S 4
SET NLKF = LINK INDEX OF FIRST LINK FROM NODE NF
NET NLKL = LINK INDEX OF LAST LINK FROM NODE NF. J = NLKF
K = THE MINIMUM OF J + 3999 AND NLKL
S5
JJ = K - J + 1
GTVL
GET THE REVERSE DIRECTION OF THE DIRECTIONAL LINK VOLUMES WITH LINK INDEXES J TO K AND STORE IN BUF
IS JJ = 4000 ?
NO
WRITE JJ WORDS OF BUF ON UNIT 3
YES
WRITE ARRAY BUF ON UNIT 3.
S6

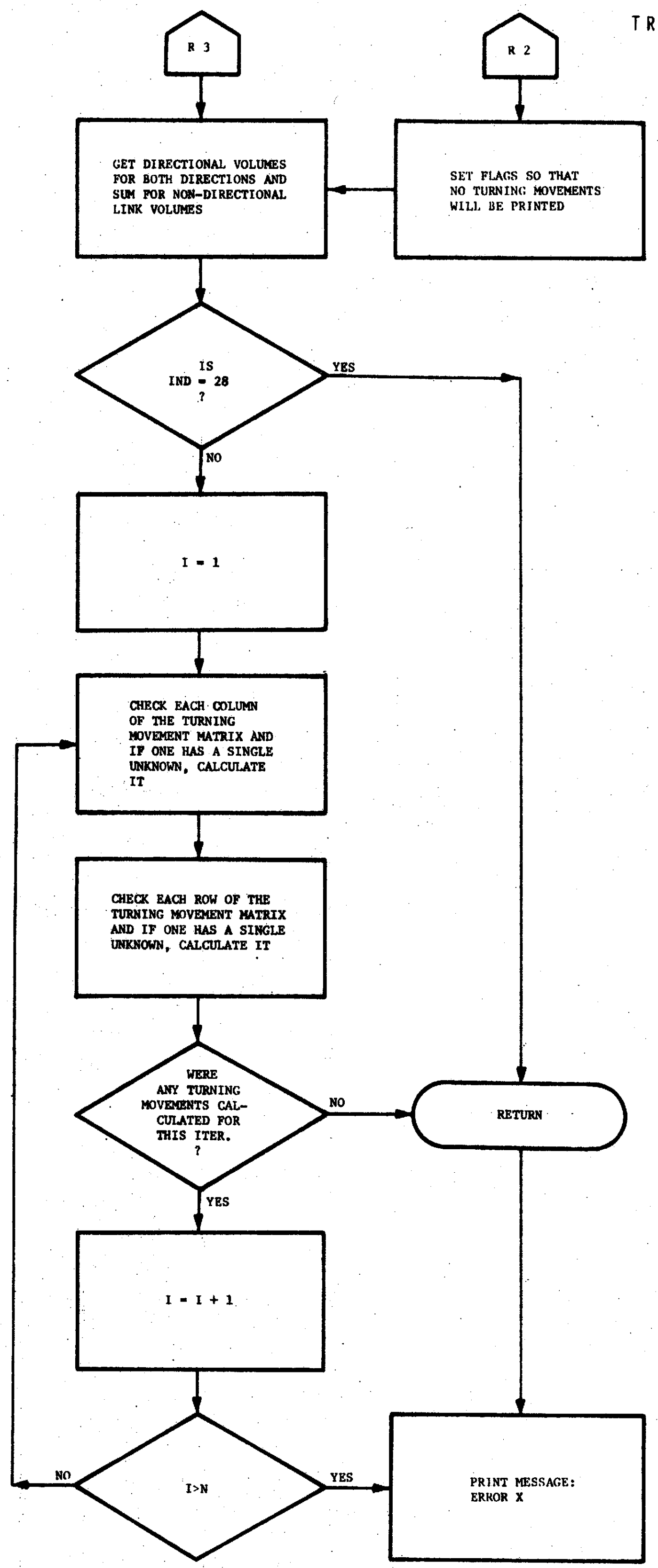

S6
J = J + 4000
IS J>NLKL
NO
S5
YES
I = I + 1
IS I > THE NUMBER OF SEGMENTS?
NO
S4
YES
I = 1
S7
SET NF = FIRST NODE NUMBER OF SEGMENT I, SET NL = LAST NODE NUMBER OF SEGMENT I.
SET NTF = INDEX OF FIRST TURN VOLUME SAVED FOR NODE NF.
SET NTL = INDEX OF LAST TURN VOLUME SAVED FOR NODE NL.
J = NTF
K = THE MINIMUM OF J + 3999 AND NTF.
JJ = K - J + 1
GETVOL
GET THE TURN VOLUMES FOR INDEXES J TO K AND STORE IN BUF
IS JJ = 4000 ?
NO
WRITE JJ WORDS OF BUF ON UNIT 3.
YES
WRITE ARRAY BUF ON UNIT 3.
J = J + 4000
IS N>NTL
NO
YES
S8


S8
I = I + 1
IS
I > THE NUMBER
OF SEGMENTS
?
NO
S7
YES
RETURN

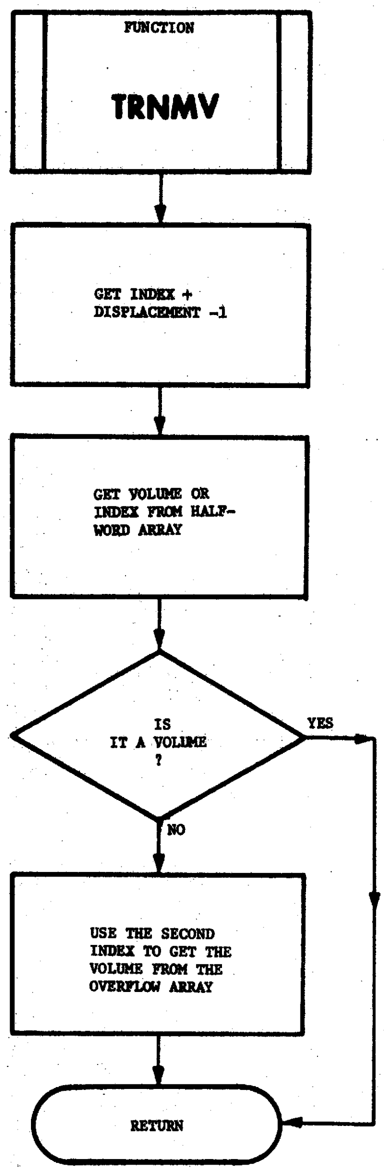

SUBROUTINE
TRN
INITIALIZE FLAGS TO INDICATE OUTPUT FOR ALL POSSIBLE TURN VOLUMES ALSO SET ALL TURN VOLUMES TO -1 TO INDICATE ALL ARE UNKNOWNS
SET FLAGS FOR NO OUTPUT OF U-TURNS AND SET VALUES OF THESE TO ZEROES
GET IND = THE TURN CODE FOR THE NODE
IS IND<1 OR IND >28 ?
YES
PRINT MESSAGE: TRMV ERROR
NO
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

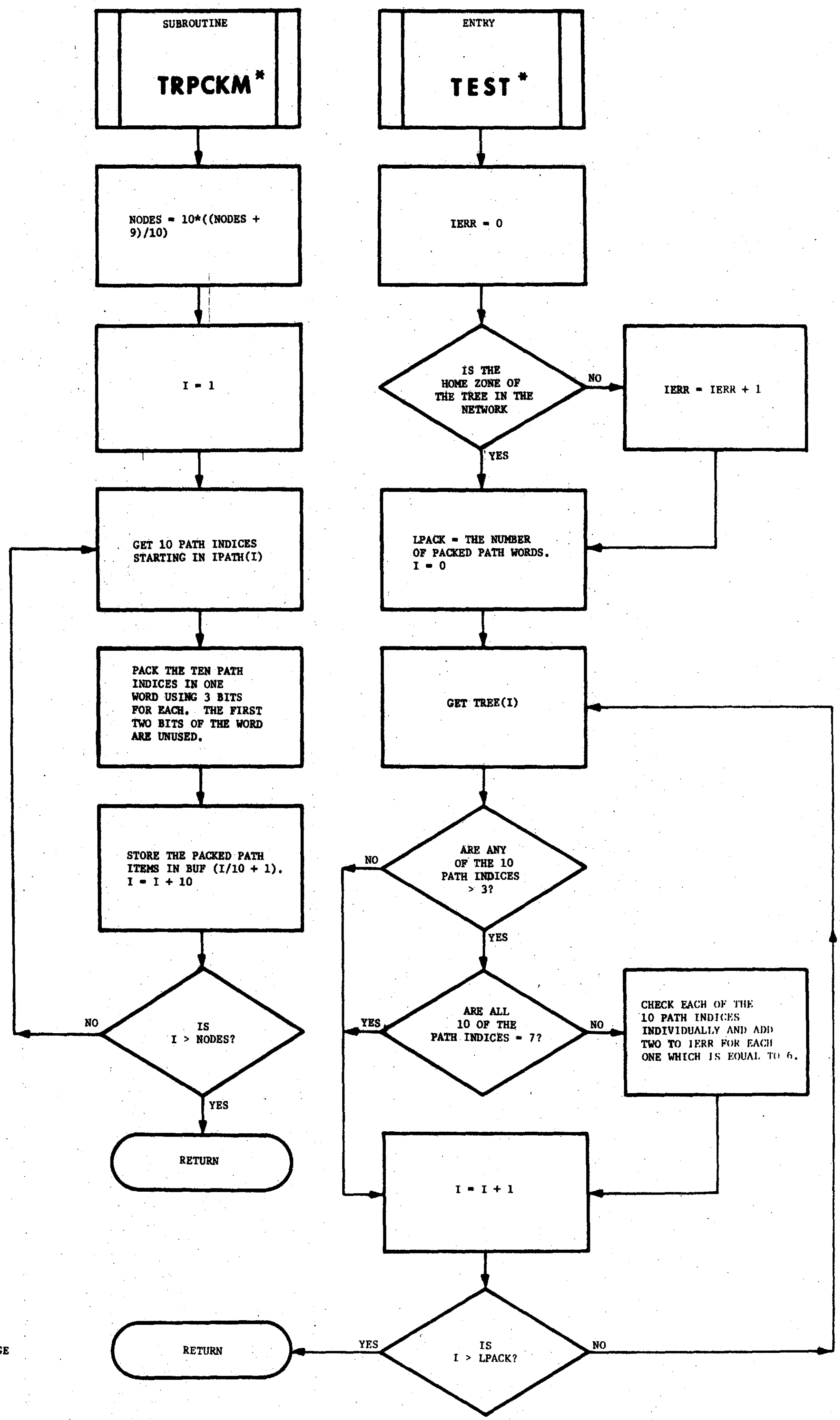

R 1
IND =
?
13, 17,
20
18,
22
21
23,
24
25
26
27
10
1-9,
19
11-16,
28
R 2
GETRN
GET TURNING MOVEMENTS WHICH WERE SAVED
MARK TURNS WHICH ARE ZERO BECAUSE OF ONE-WAY LINKS FOR NO PRINTED OUTPUT
IS N>6 ?
YES
NO
PRINT MESSAGE: ERROR X
R 3

TRN

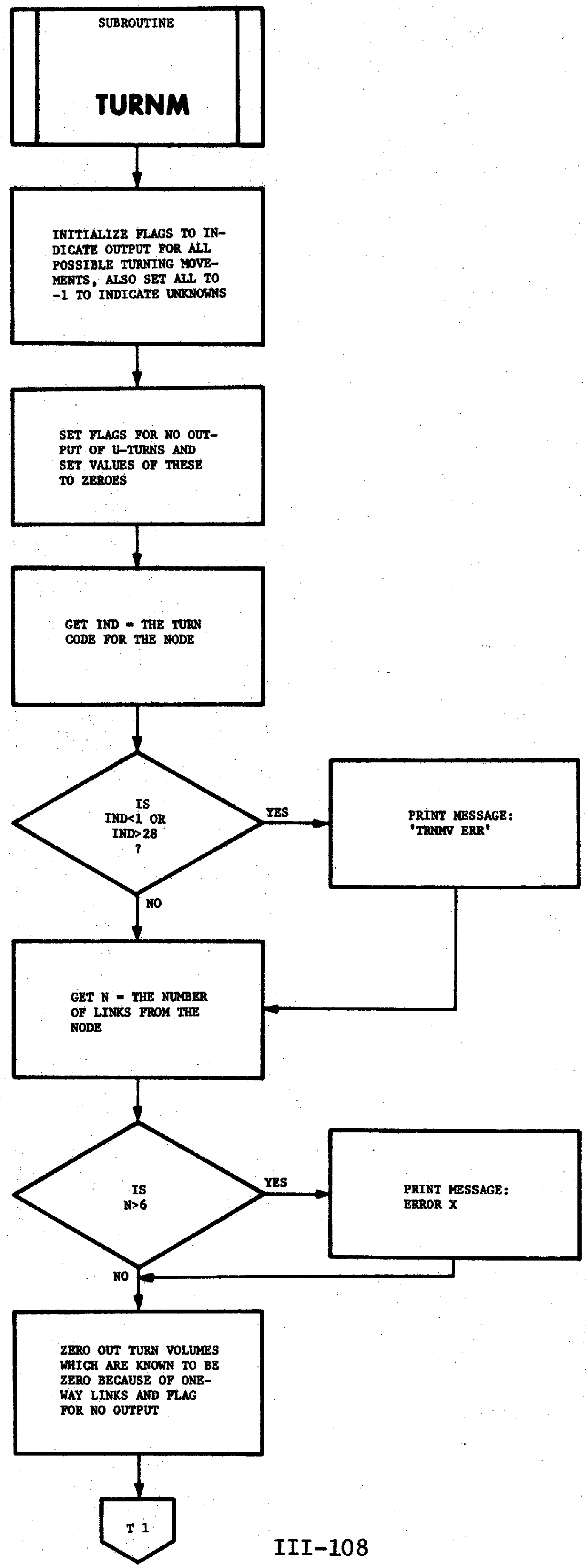


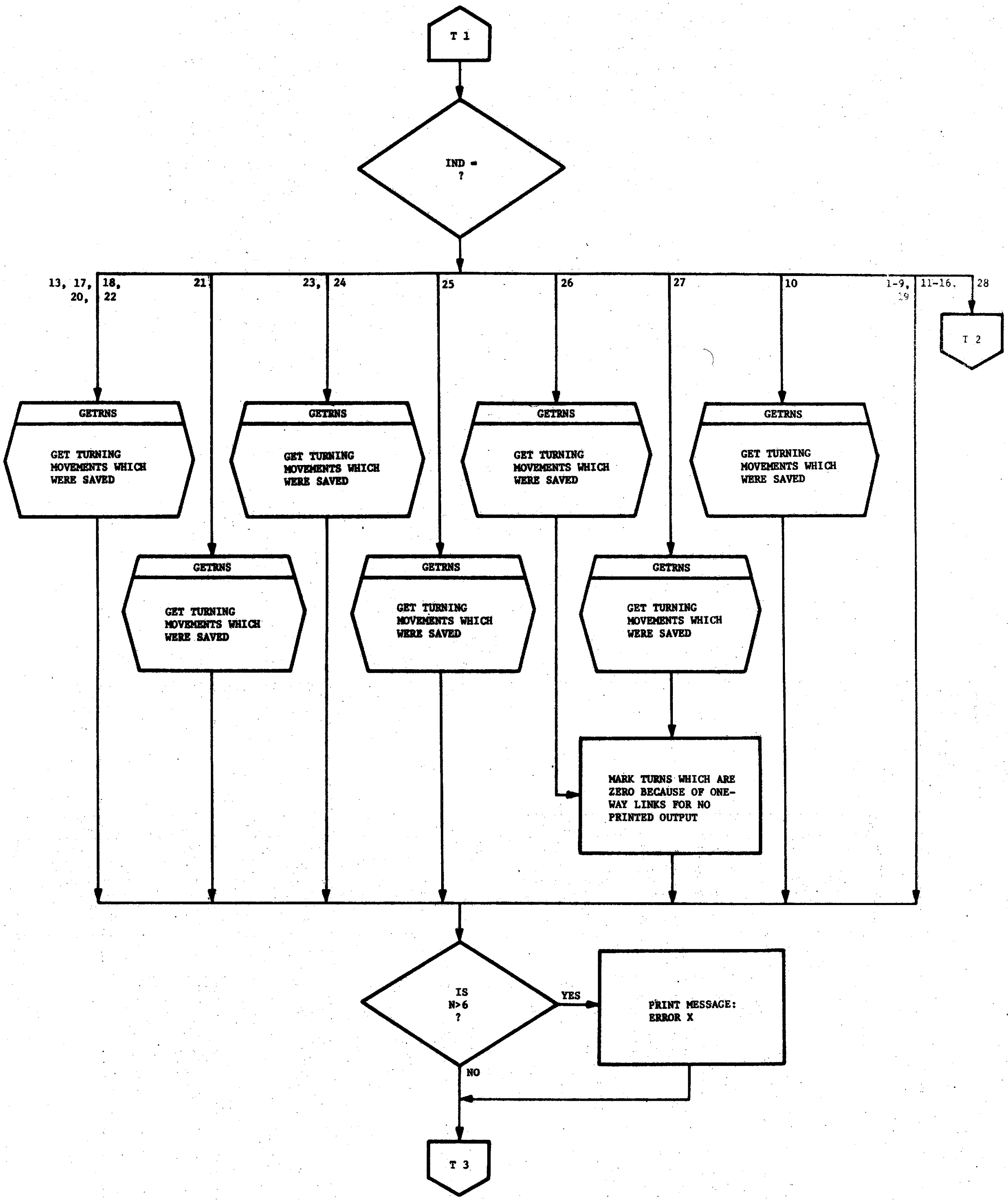
R 3
R 2
GET DIRECTIONAL VOLUMES FOR BOTH DIRECTIONS AND SUM FOR NON-DIRECTIONAL LINK VOLUMES
SET FLAGS SO THAT NO TURNING MOVEMENTS WILL BE PRINTED
IS IND = 28 ?
YES
NO
I = 1
CHECK EACH COLUMN OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
CHECK EACH ROW OF THE TURNING MOVEMENT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

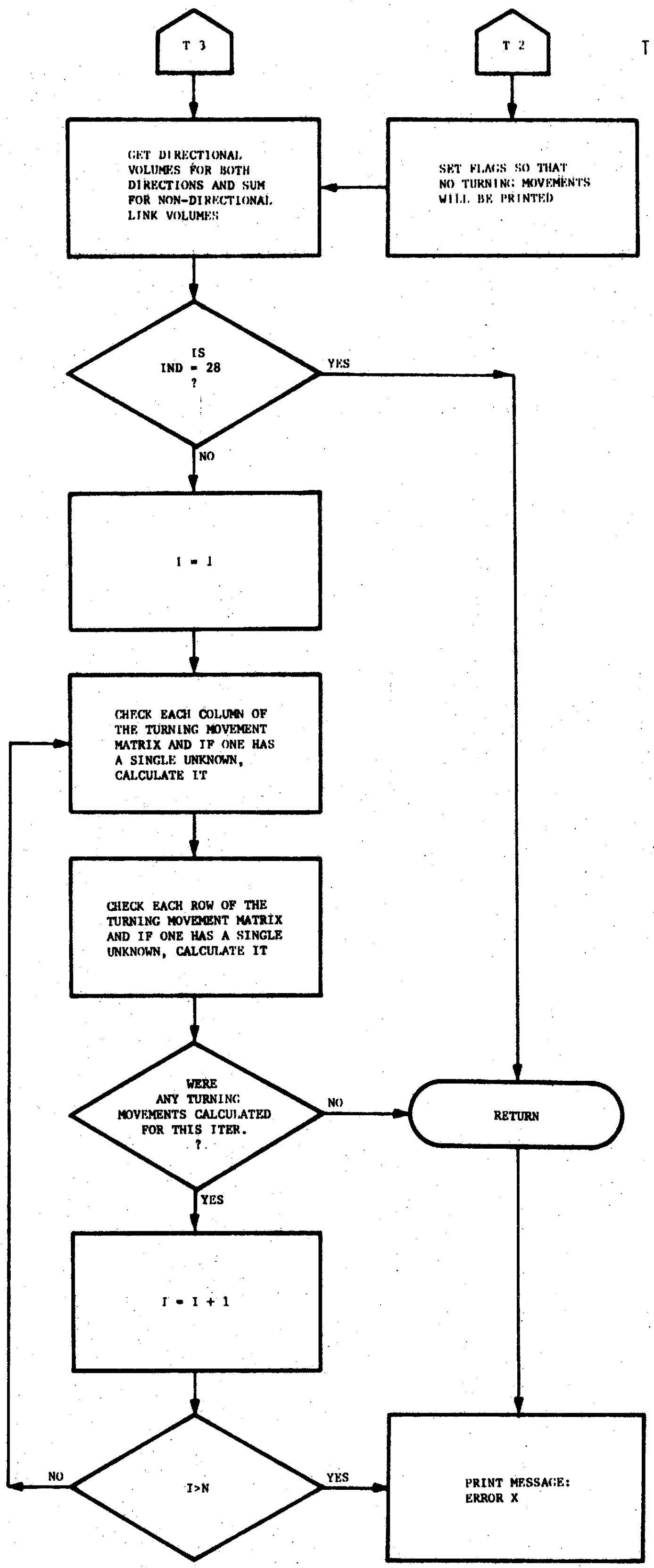

FUNCTION
TRNMV
GET INDEX +
DISPLACEMENT -1
GET VOLUME OR
INDEX FROM HALF-
WORD ARRAY
IS
IT A VOLUME
?
YES
NO
USE THE SECOND
INDEX TO GET THE
VOLUME FROM THE
OVERFLOW ARRAY
RETURN

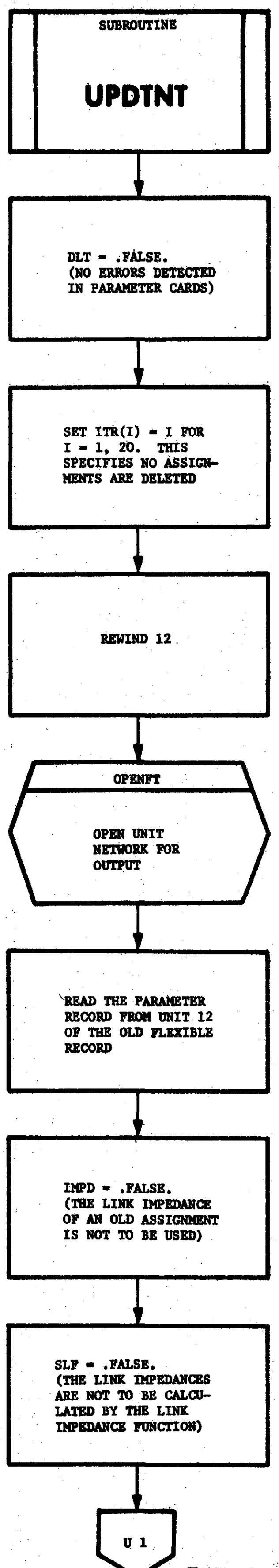


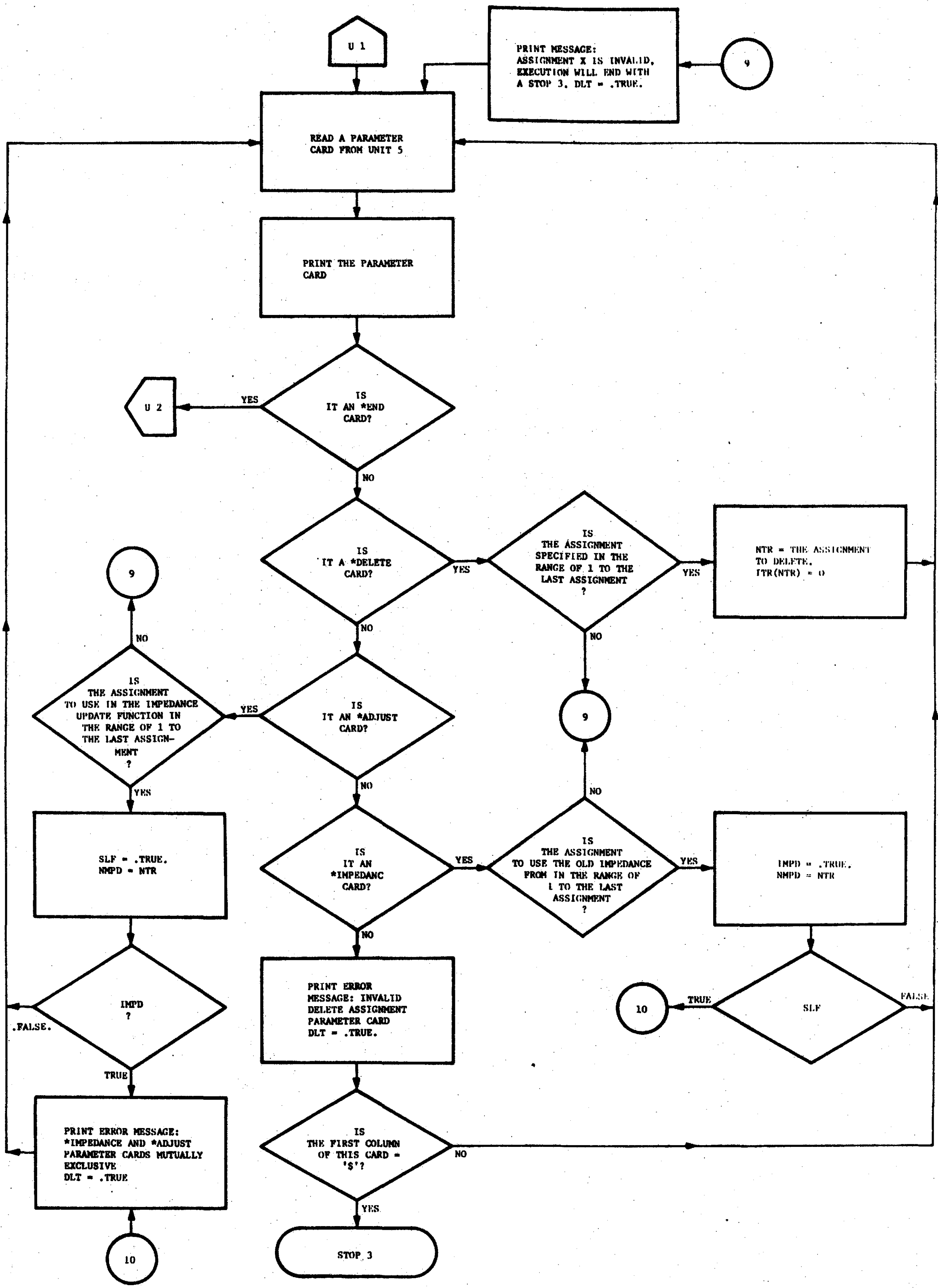
*ASSEMBLY LANGUAGE

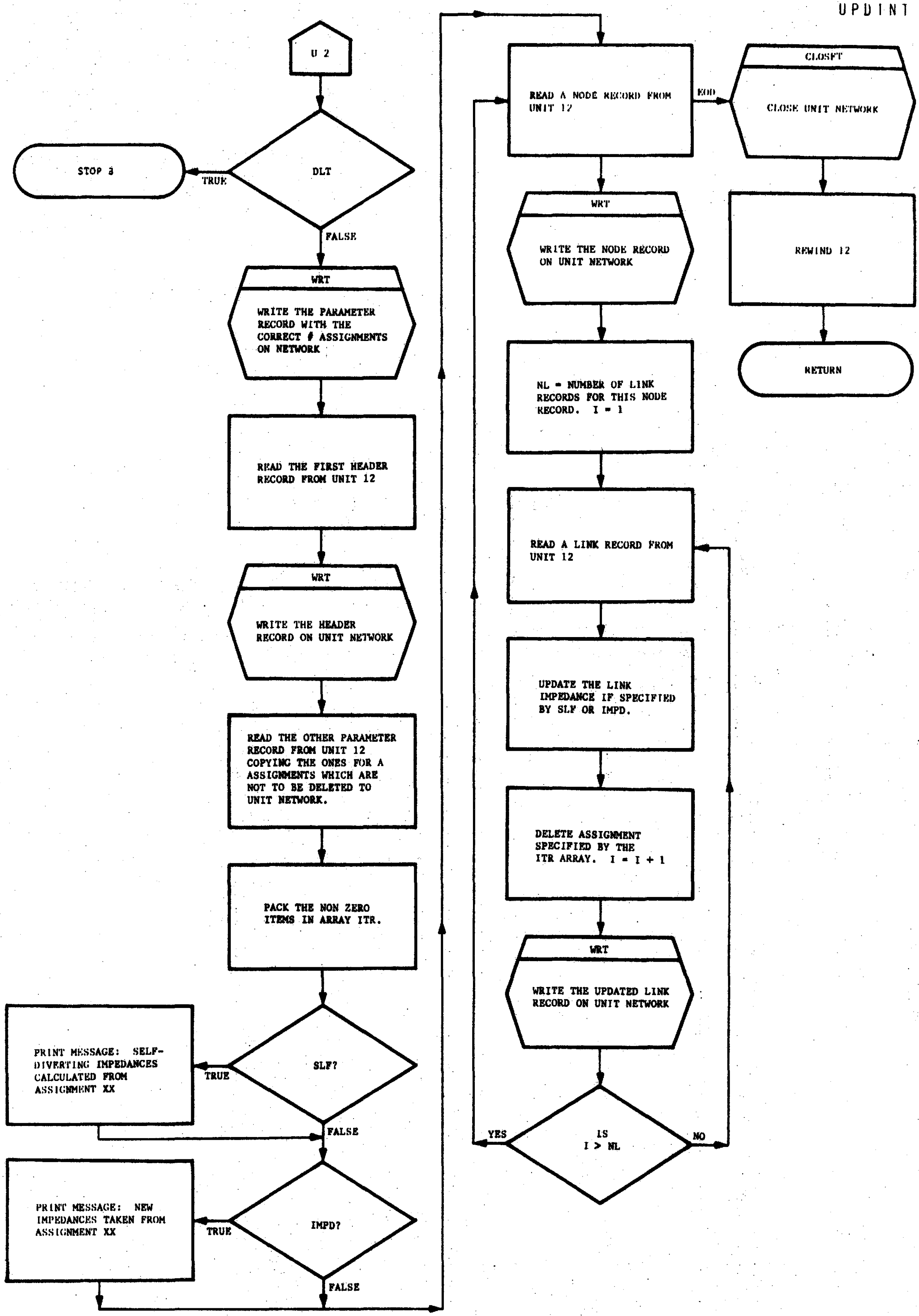

SUBROUTINE
TURNM
INITIALIZE FLAGS TO INDICATE OUTPUT FOR ALL POSSIBLE TURNING MOVEMENTS, ALSO SET ALL TO -1 TO INDICATE UNKNOWNS
SET FLAGS FOR NO OUTPUT OF U-TURNS AND SET VALUES OF THESE TO ZEROES
GET IND = THE TURN CODE FOR THE NODE
IS IND<1 OR IND>28 ?
YES
PRINT MESSAGE: 'TRNMV ERR'
NO
GET N = THE 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

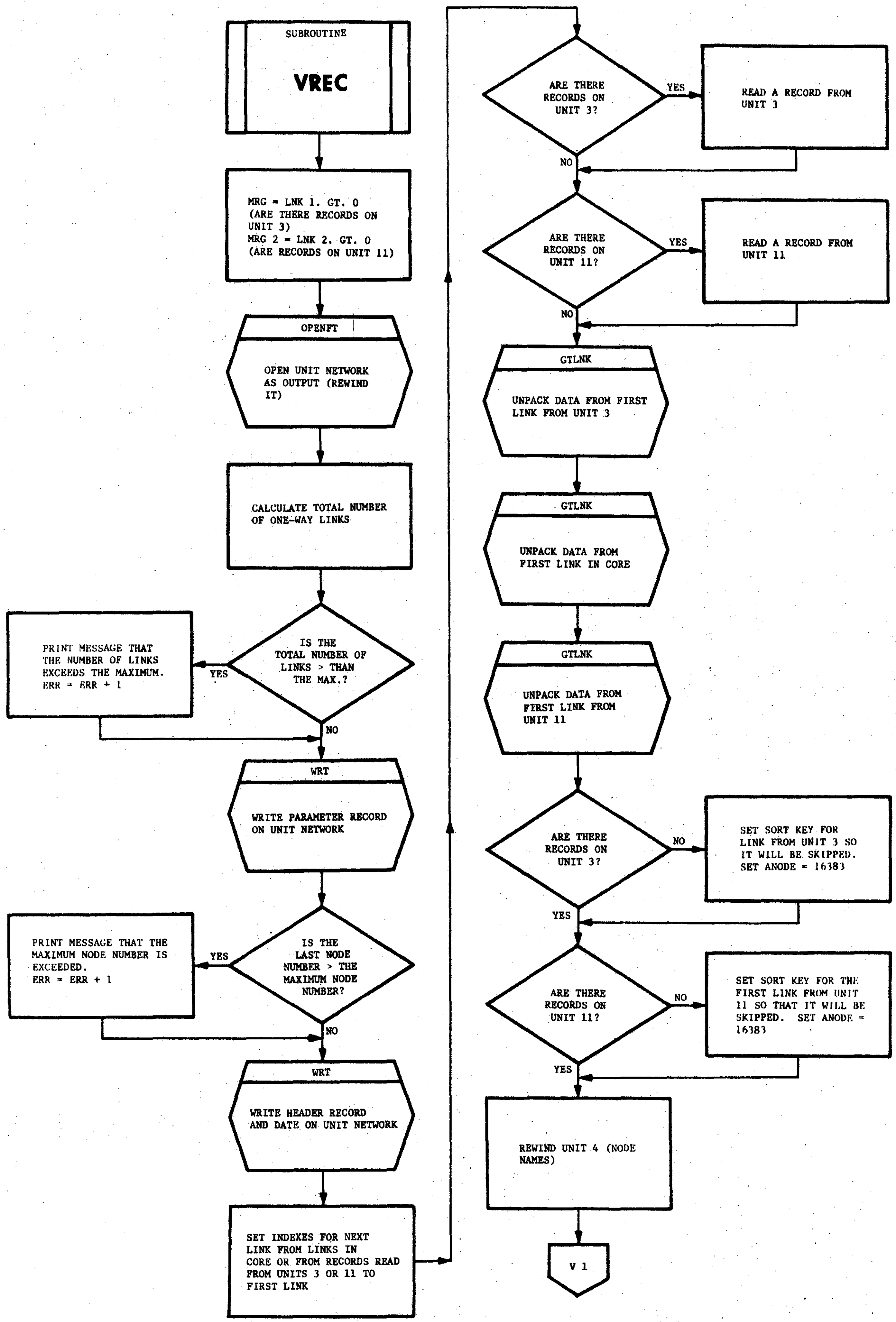

T 1
IND =
?
13, 17, 20,
18, 22
21
23,
24
25
26
27
10
1-9, 19
11-16,
28
T 2
GETRNS
GET TURNING MOVEMENTS WHICH WERE SAVED
MARK TURNS WHICH ARE ZERO BECAUSE OF ONE-WAY LINKS FOR NO PRINTED OUTPUT
IS
N>6
?
YES
PRINT MESSAGE: ERROR X
NO
T 3

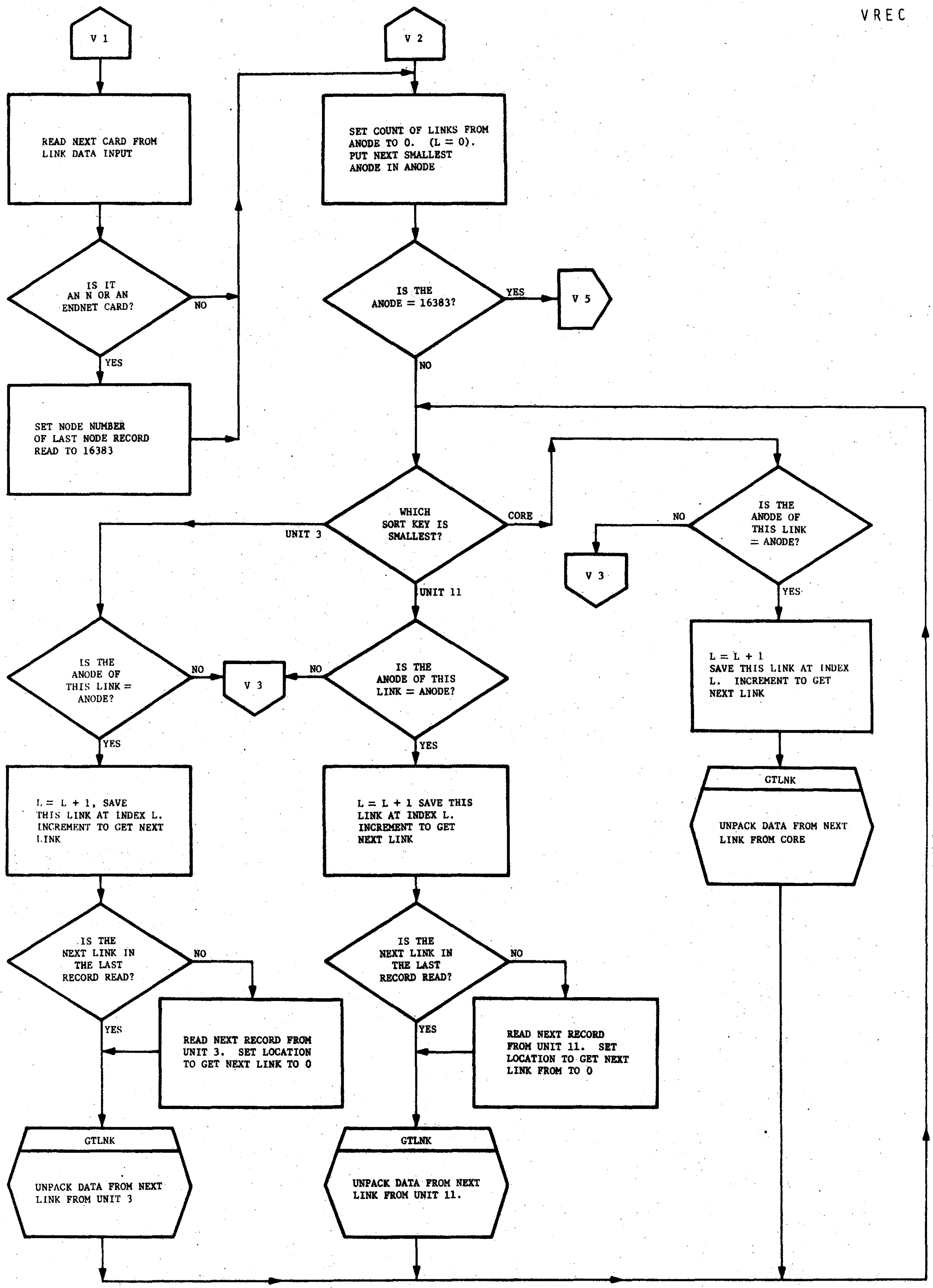

T 3
T 2
GET DIRECTIONAL VOLUMES FOR BOTH DIRECTIONS AND SUM FOR NON-DIRECTIONAL LINK VOLUMES
SET FLAGS SO THAT NO TURNING MOVEMENTS WILL BE PRINTED
IS IND = 28 ?
YES
NO
I = 1
CHECK EACH COLUMN OF THE TURNING MOVEMENT MATRIX AND IF ONE HAS A SINGLE UNKNOWN, CALCULATE IT
CHECK EACH ROW OF THE TURNING MOVEMENT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

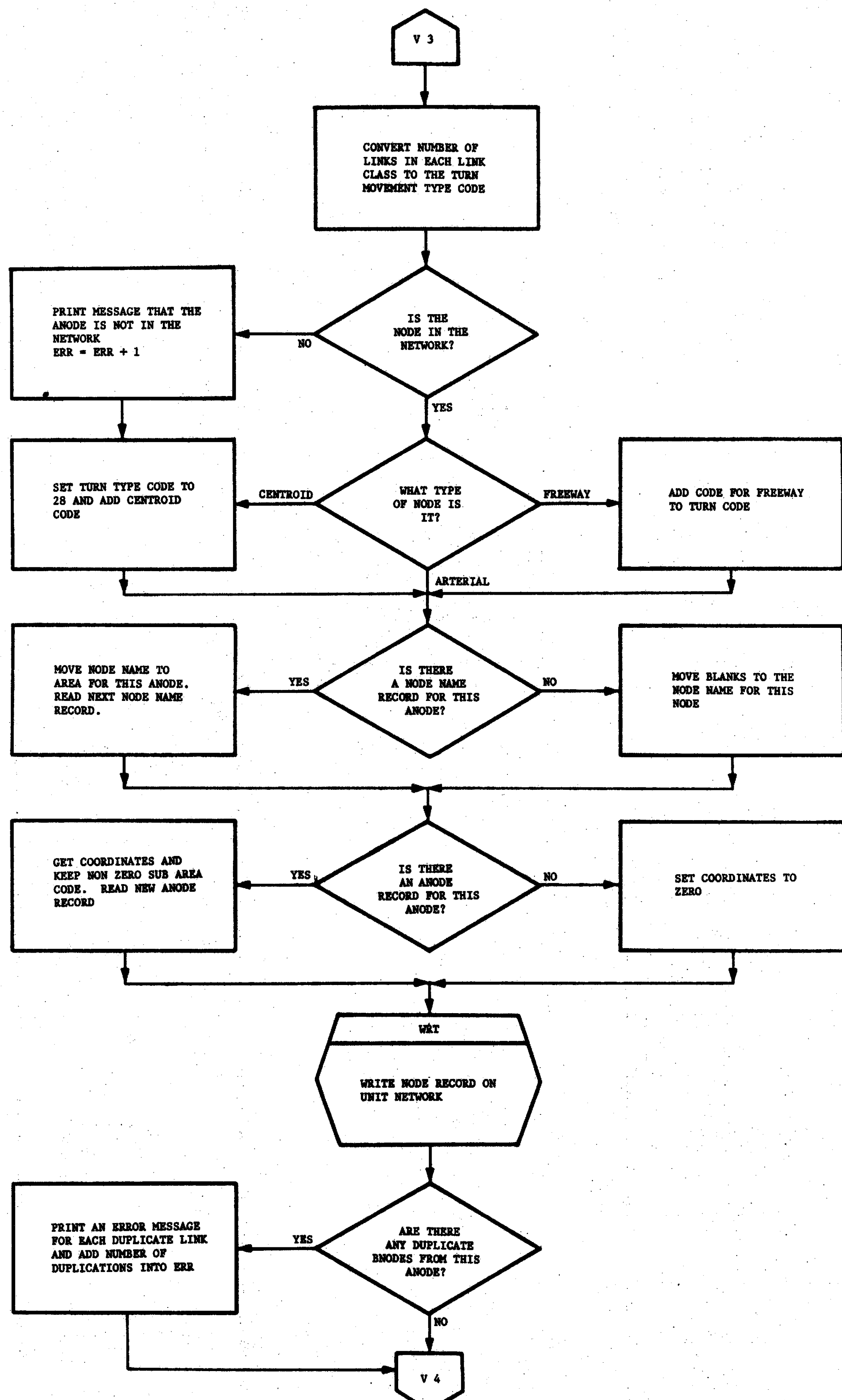

SUBROUTINE
UPDTNT
DLT = .FALSE.
(NO ERRORS DETECTED IN PARAMETER CARDS)
SET ITR(I) = I FOR I = 1, 20. THIS SPECIFIES NO ASSIGN-MENTS ARE DELETED
REWIND 12
OPENFT
OPEN UNIT NETWORK FOR OUTPUT
READ THE PARAMETER RECORD FROM UNIT 12 OF THE OLD FLEXIBLE RECORD
IMPD = .FALSE.
(THE LINK IMPEDANCE OF AN OLD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

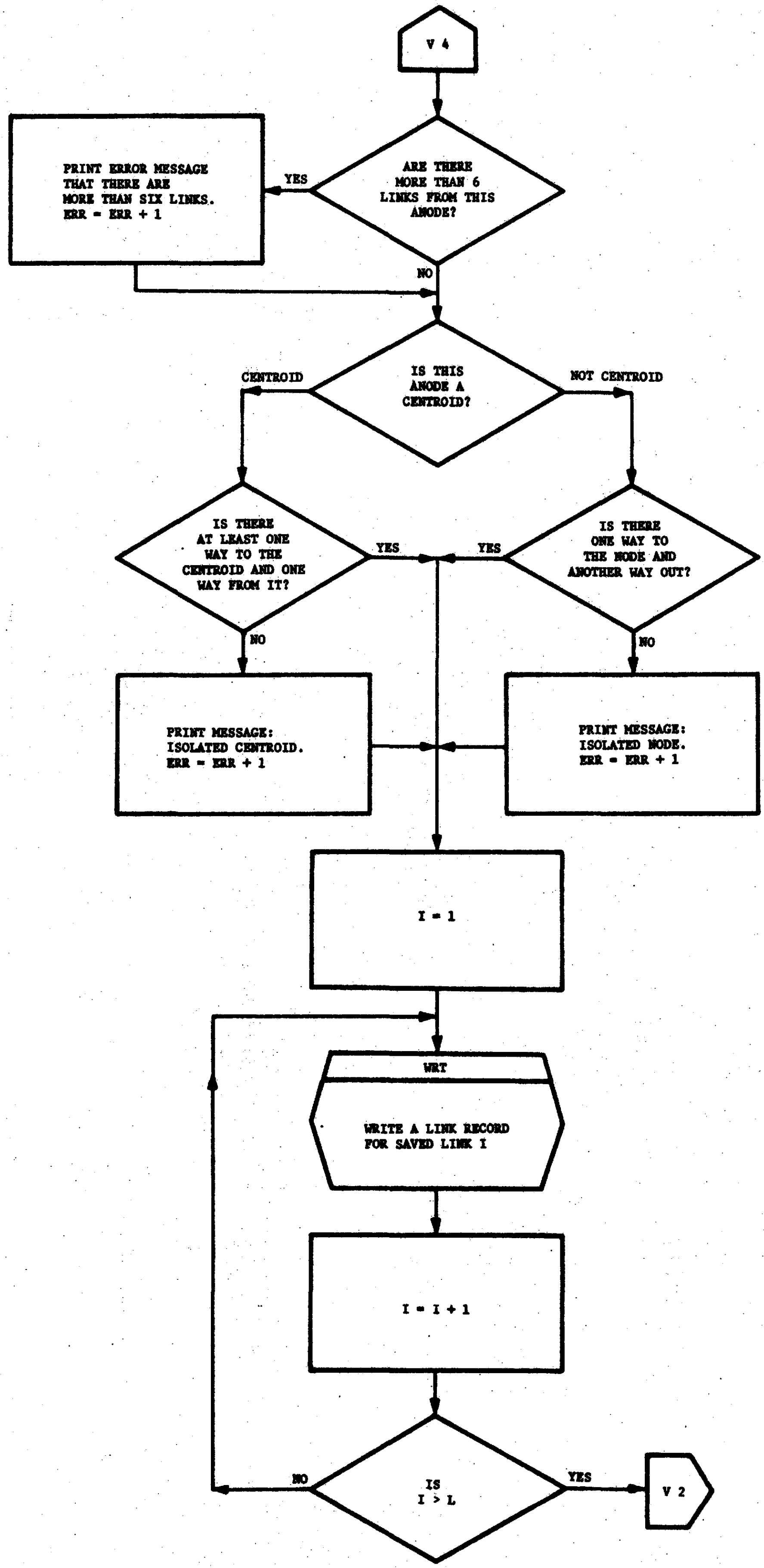

U 1
PRINT MESSAGE: ASSIGNMENT X IS INVALID, EXECUTION WILL END WITH A STOP 3. DLT = .TRUE.
9
READ A PARAMETER CARD FROM UNIT 5
PRINT THE PARAMETER CARD
IS IT AN *END CARD?
YES
U 2
NO
IS IT A *DELETE CARD?
YES
IS THE ASSIGNMENT SPECIFIED IN THE RANGE OF 1 TO THE LAST ASSIGNMENT ?
YES
NTR = THE ASSIGNMENT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

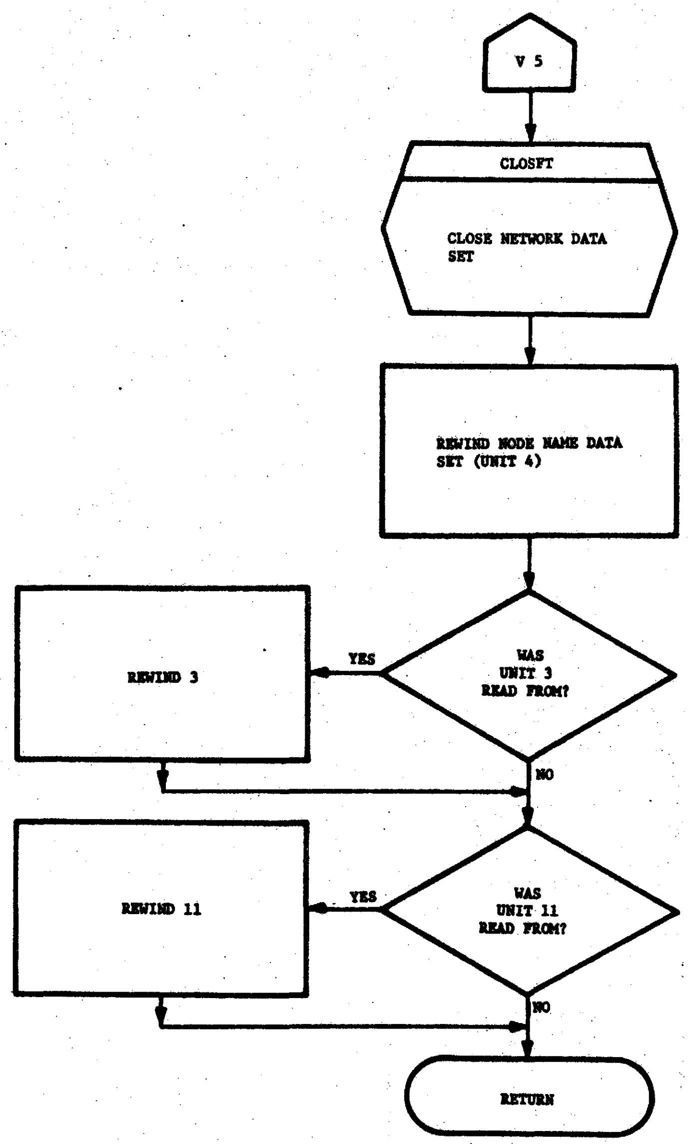

U 2
DLT
TRUE
STOP 3
FALSE
WRT
WRITE THE PARAMETER RECORD WITH THE CORRECT # ASSIGNMENTS ON NETWORK
READ THE FIRST HEADER RECORD FROM UNIT 12
WRT
WRITE THE HEADER RECORD ON UNIT NETWORK
READ THE OTHER PARAMETER RECORD FROM UNIT 12 COPYING THE ONES FOR A ASSIGNMENTS WHICH ARE NOT TO BE DELETED TO UNIT 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

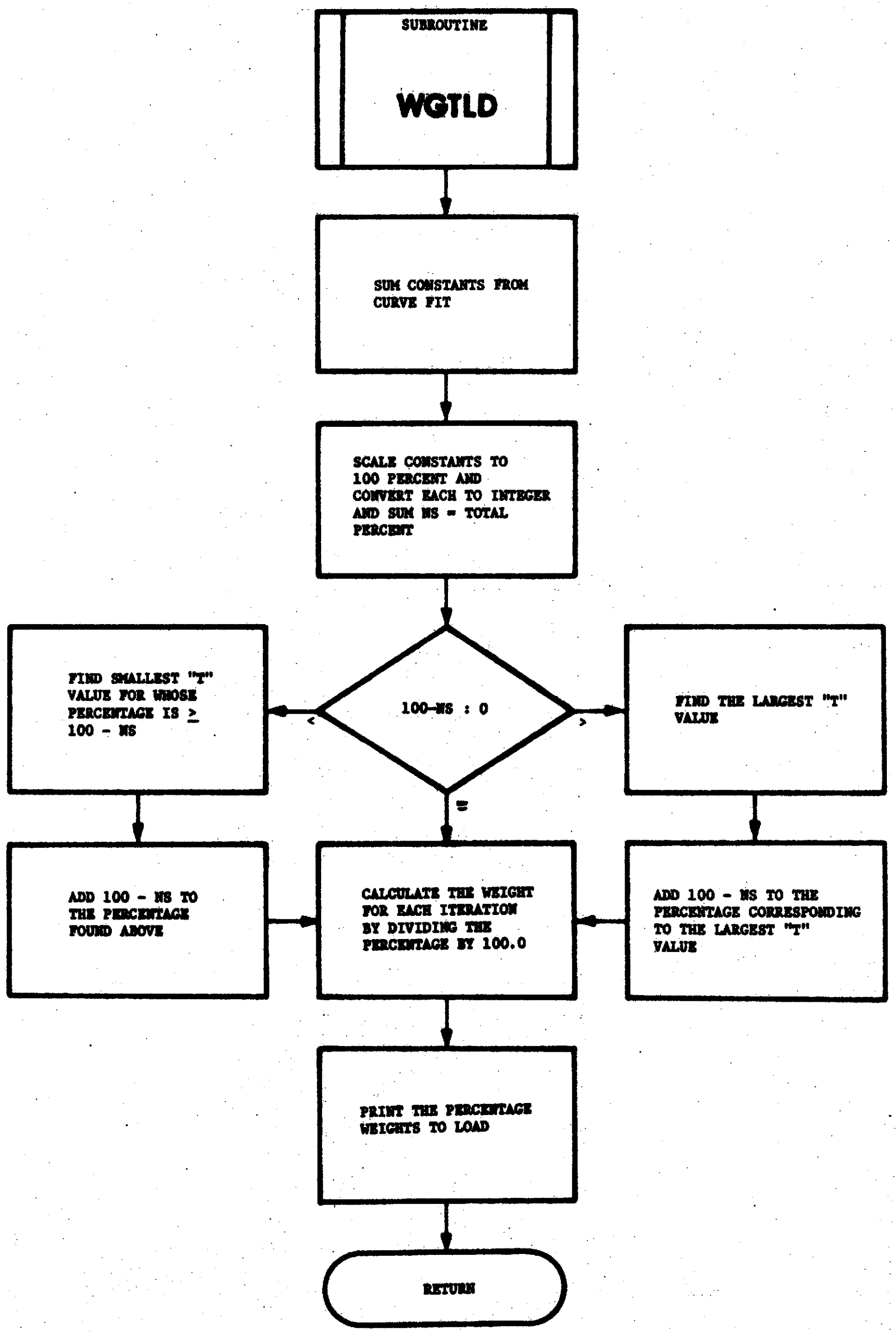

SUBROUTINE
VREC
MRG = LNK 1. GT. 0
(ARE THERE RECORDS ON
UNIT 3)
MRG 2 = LNK 2. GT. 0
(ARE RECORDS ON UNIT 11)
OPENFT
OPEN UNIT NETWORK
AS OUTPUT (REWIND
IT)
CALCULATE TOTAL NUMBER
OF ONE-WAY LINKS
IS THE
TOTAL NUMBER OF
LINKS > THAN
THE MAX.?
YES
PRINT MESSAGE THAT
THE NUMBER OF LINKS
EXCEEDS THE MAXIMUM.
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

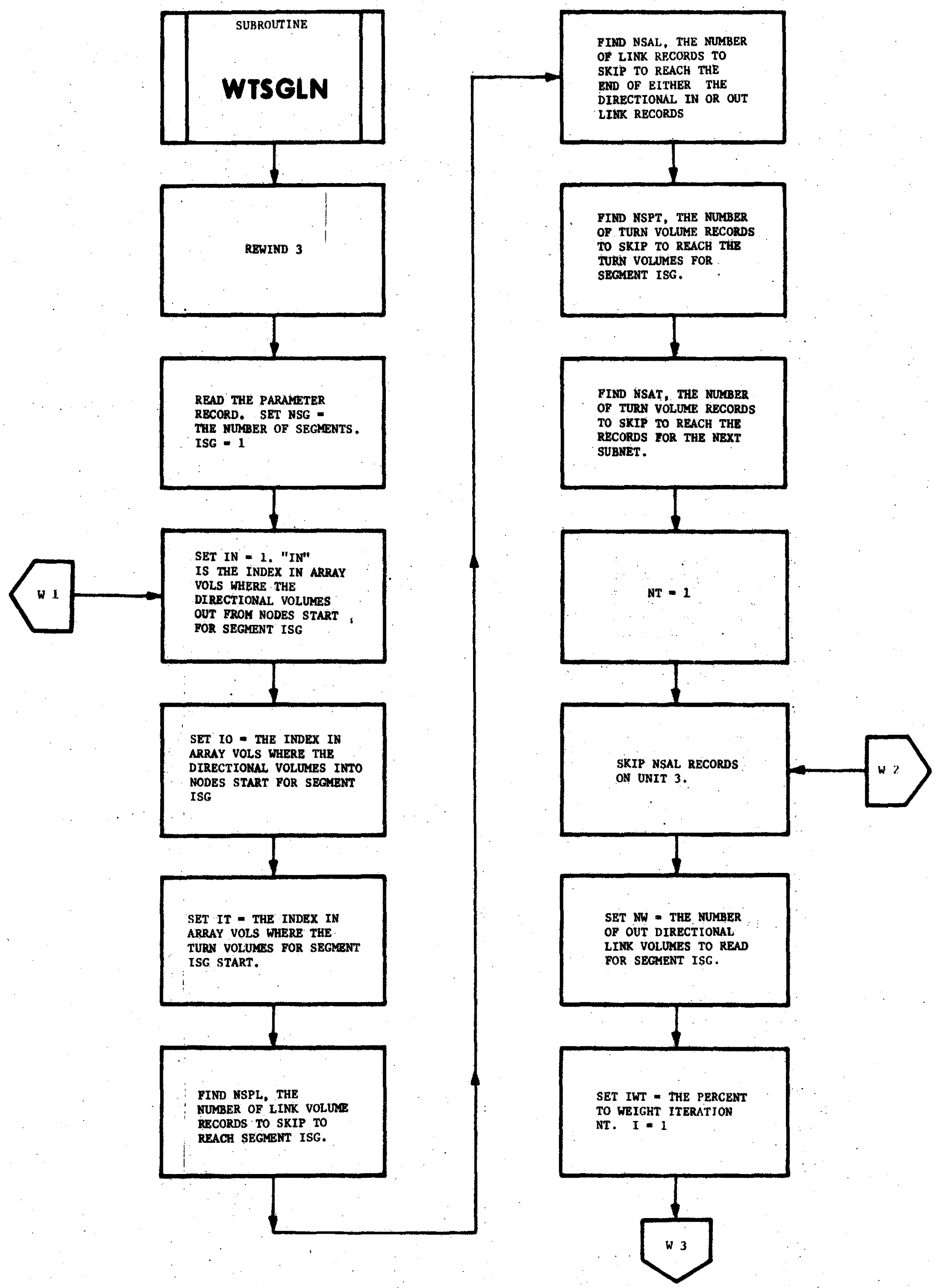

V 1
READ NEXT CARD FROM LINK DATA INPUT
IS IT AN N OR AN ENDNET CARD?
NO
YES
SET NODE NUMBER OF LAST NODE RECORD READ TO 16383
V 2
SET COUNT OF LINKS FROM ANODE TO 0. (L = 0). PUT NEXT SMALLEST ANODE IN ANODE
IS THE ANODE = 16383?
YES
V 5
NO
WHICH SORT KEY IS SMALLEST?
UNIT 3
CORE
UNIT 11
IS THE ANODE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

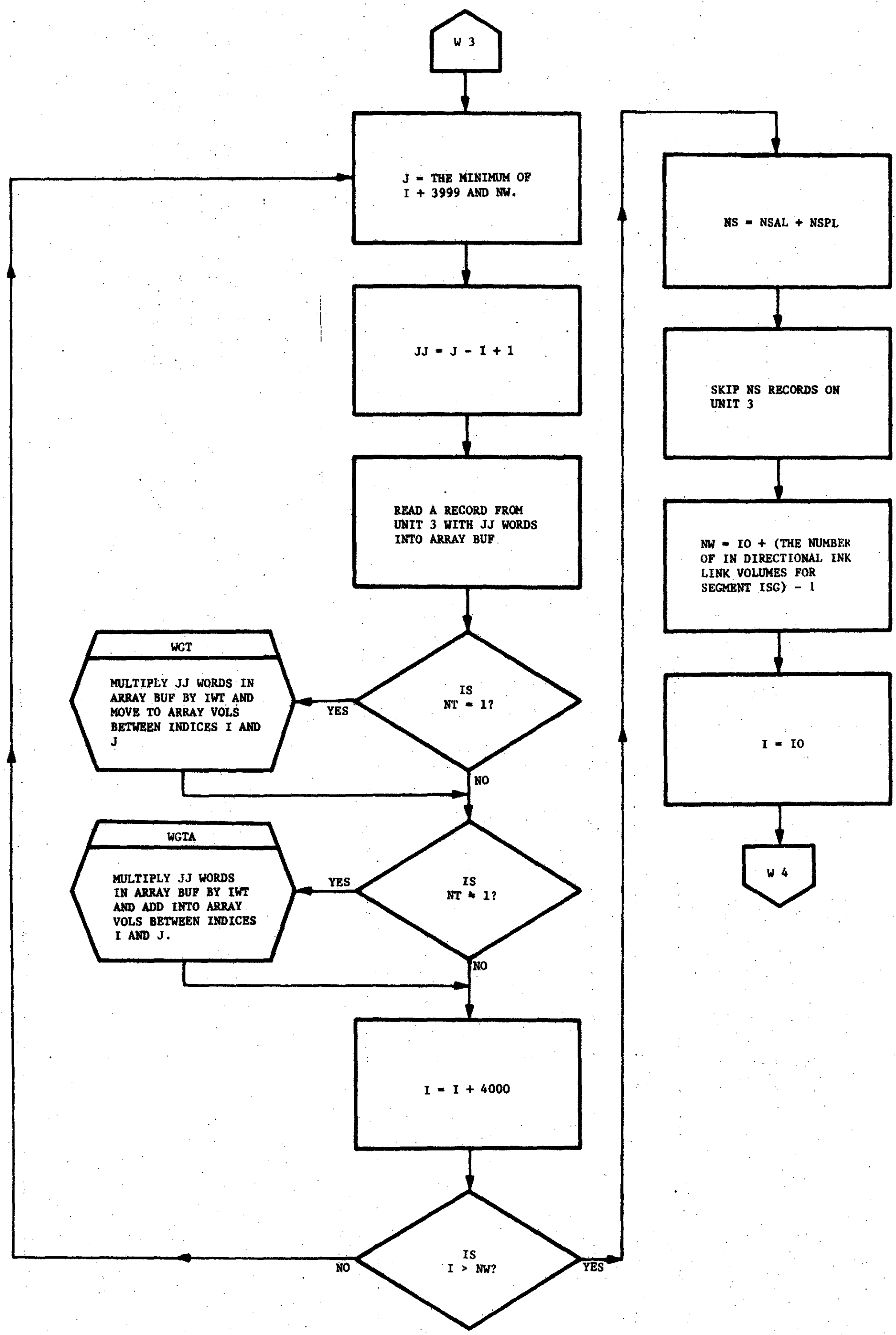

V 3
CONVERT NUMBER OF LINKS IN EACH LINK CLASS TO THE TURN MOVEMENT TYPE CODE
IS THE NODE IN THE NETWORK?
NO
PRINT MESSAGE THAT THE ANODE IS NOT IN THE NETWORK ERR = ERR + 1
YES
WHAT TYPE OF NODE IS IT?
CENTROID
SET TURN TYPE CODE TO 28 AND ADD CENTROID CODE
FREEWAY
ADD CODE FOR FREEWAY TO TURN CODE
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

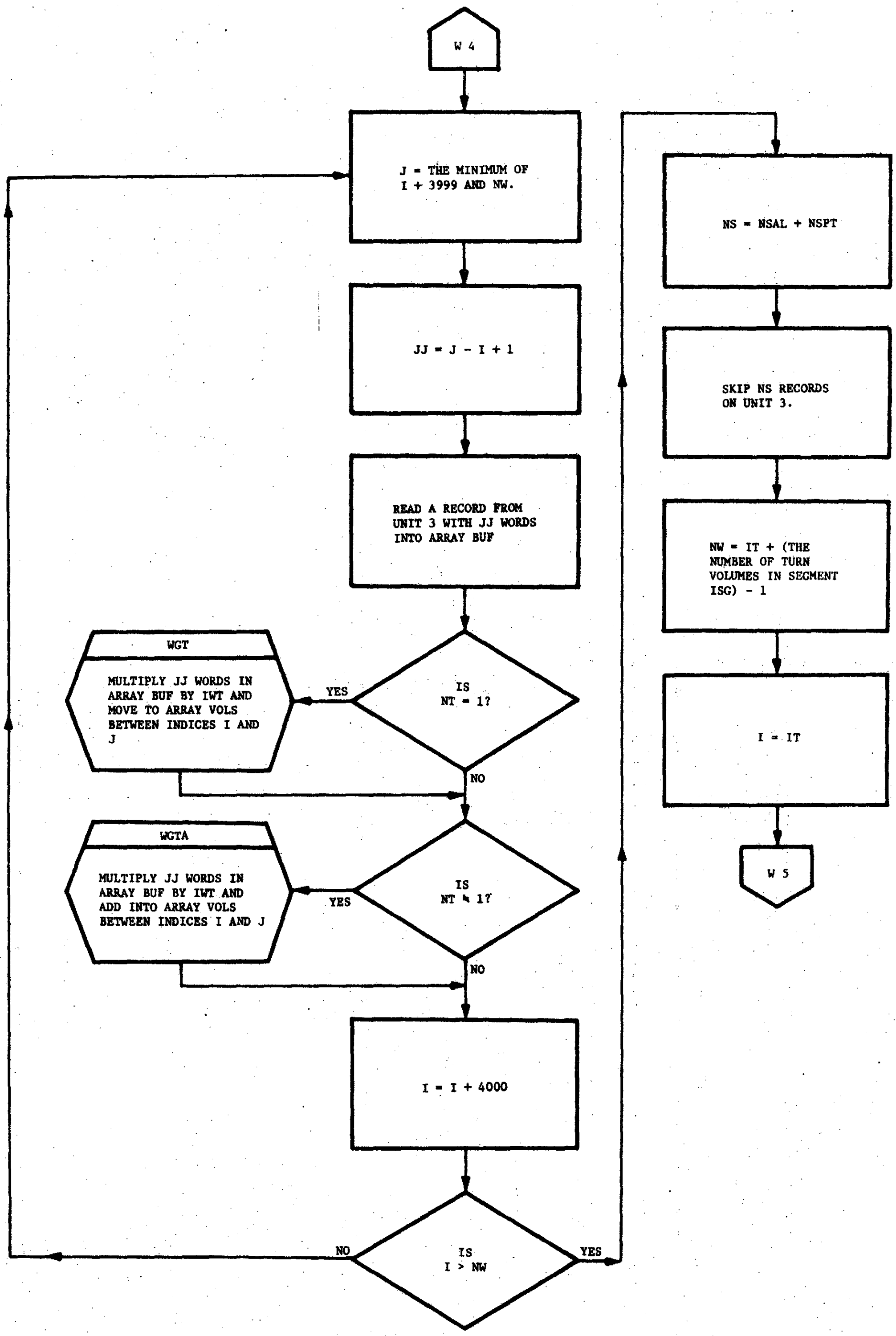

V 4
ARE THERE MORE THAN 6 LINKS FROM THIS ANODE?
YES
PRINT ERROR MESSAGE THAT THERE ARE MORE THAN SIX LINKS. ERR = ERR + 1
NO
IS THIS ANODE A CENTROID?
CENTROID
NOT CENTROID
IS THERE AT LEAST ONE WAY TO THE CENTROID AND ONE WAY FROM IT?
YES
NO
IS THERE ONE WAY TO THE NODE AND ANOTHER WAY OUT?
YES
NO
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

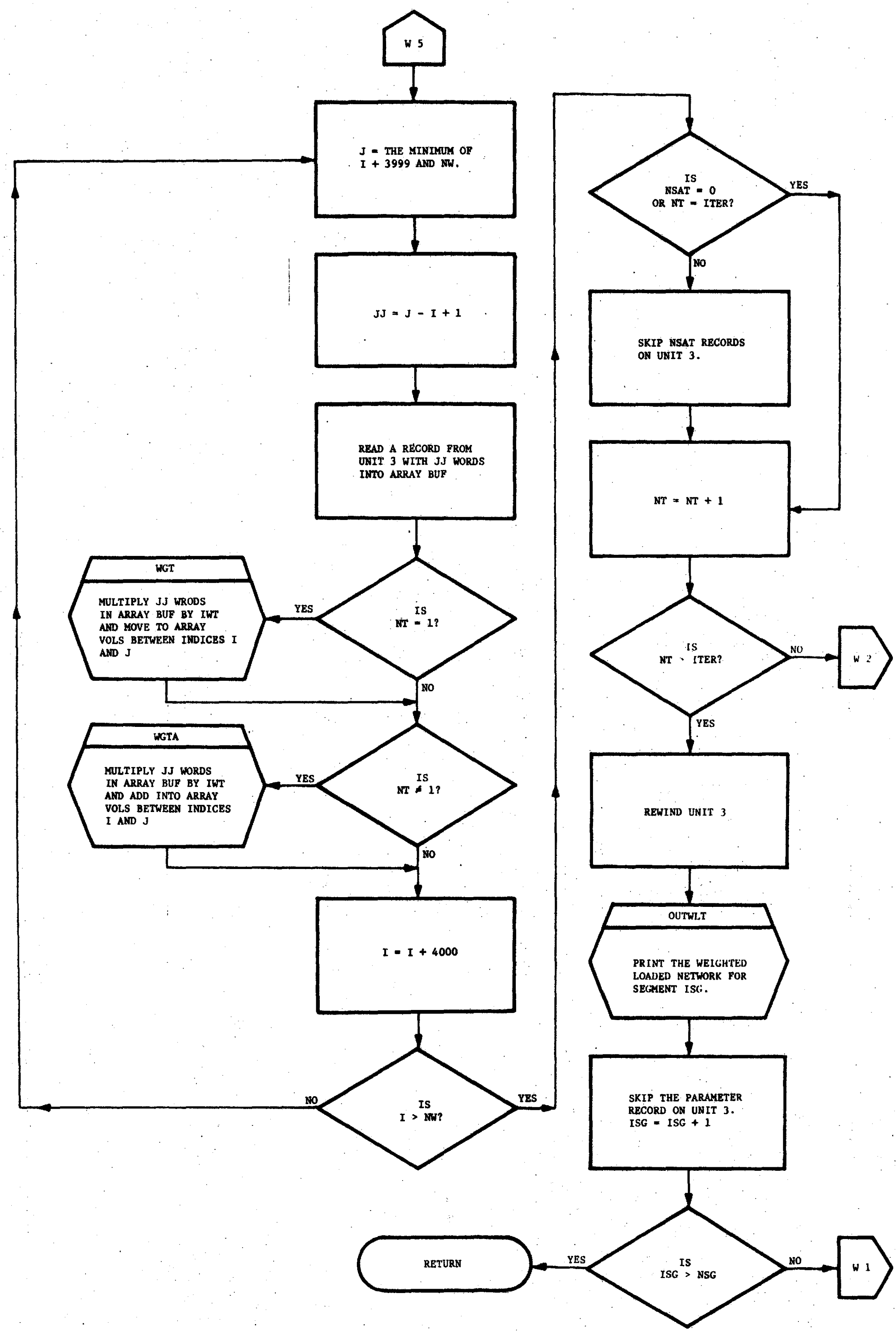

V 5
CLOSFT
CLOSE NETWORK DATA SET
REWIND NODE NAME DATA SET (UNIT 4)
WAS UNIT 3 READ FROM?
YES
REWIND 3
NO
WAS UNIT 11 READ FROM?
YES
REWIND 11
NO
RETURN


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
WTSGLN
REWIND 3
READ THE PARAMETER RECORD. SET NSG = THE NUMBER OF SEGMENTS. ISG = 1
W 1
SET IN = 1. "IN" IS THE INDEX IN ARRAY VOLS WHERE THE DIRECTIONAL VOLUMES OUT FROM NODES START FOR SEGMENT ISG
SET IO = THE INDEX IN ARRAY VOLS WHERE THE DIRECTIONAL VOLUMES INTO NODES START FOR SEGMENT ISG
SET IT = THE INDEX IN ARRAY VOLS WHERE THE TURN VOLUMES FOR SEGMENT ISG START.
FIND NSPL, THE NUMBER OF LINK VOLUME RECORDS TO SKIP TO REACH SEGMENT ISG.
FIND NSAL, THE NUMBER OF LINK RECORDS TO SKIP TO REACH THE END OF EITHER THE DIRECTIONAL IN OR OUT LINK RECORDS
FIND NSPT, THE NUMBER OF TURN VOLUME RECORDS TO SKIP TO REACH THE TURN VOLUMES FOR SEGMENT ISG.
FIND NSAT, THE NUMBER OF TURN VOLUME RECORDS TO SKIP TO REACH THE RECORDS FOR THE NEXT SUBNET.
NT = 1
SKIP NSAL RECORDS ON UNIT 3.
W 2
SET NW = THE NUMBER OF OUT DIRECTIONAL LINK VOLUMES TO READ FOR SEGMENT ISG.
SET IWT = THE PERCENT TO WEIGHT ITERATION NT. I = 1
W 3


W 3
J = THE MINIMUM OF I + 3999 AND NW.
JJ = J - I + 1
READ A RECORD FROM UNIT 3 WITH JJ WORDS INTO ARRAY BUF
IS NT = 1?
YES
WCT
MULTIPLY JJ WORDS IN ARRAY BUF BY IWT AND MOVE TO ARRAY VOLS BETWEEN INDICES I AND J
NO
IS NT ≠ 1?
YES
WGTA
MULTIPLY JJ WORDS IN ARRAY BUF BY IWT AND ADD INTO ARRAY VOLS BETWEEN INDICES I AND J.
NO
I = I + 4000
IS I > NW?
NO
YES
NS = NSAL + NSPL
SKIP NS RECORDS ON UNIT 3
NW = IO + (THE NUMBER OF IN DIRECTIONAL INK LINK VOLUMES FOR SEGMENT ISG) - 1
I = IO
W 4


W 4
J = THE MINIMUM OF I + 3999 AND NW.
JJ = J - I + 1
READ A RECORD FROM UNIT 3 WITH JJ WORDS INTO ARRAY BUF
IS NT = 1?
YES
NO
WGT
MULTIPLY JJ WORDS IN ARRAY BUF BY IWT AND MOVE TO ARRAY VOLS BETWEEN INDICES I AND J
IS NT ≠ 1?
YES
NO
WGTA
MULTIPLY JJ WORDS IN ARRAY BUF BY IWT AND ADD INTO ARRAY VOLS BETWEEN INDICES I AND J
I = I + 4000
IS I > NW
NO
YES
NS = NSAL + NSPT
SKIP NS RECORDS ON UNIT 3.
NW = IT + (THE NUMBER OF TURN VOLUMES IN SEGMENT ISG) - 1
I = IT
W 5


W 5
J = THE MINIMUM OF I + 3999 AND NW.
JJ = J - I + 1
READ A RECORD FROM UNIT 3 WITH JJ WORDS INTO ARRAY BUF
IS NT = 1?
YES
WGT
MULTIPLY JJ WRODS IN ARRAY BUF BY IWT AND MOVE TO ARRAY VOLS BETWEEN INDICES I AND J
NO
IS NT ≠ 1?
YES
WGTA
MULTIPLY JJ WORDS IN ARRAY BUF BY IWT AND ADD INTO ARRAY VOLS BETWEEN INDICES I AND J
NO
I = I + 4000
IS I > NW?
NO
YES
IS NSAT = 0 OR NT = ITER?
YES
NO
SKIP NSAT RECORDS ON UNIT 3.
NT = NT + 1
IS
NT ITER?
NO
W 2
YES
REWIND UNIT 3
OUTWLT
PRINT THE WEIGHTED LOADED NETWORK FOR SEGMENT ISG.
SKIP THE PARAMETER RECORD ON UNIT 3. ISG = ISG + 1
IS ISG > NSG
YES
RETURN
NO
W 1

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

Eleven labeled common control sections are contained in the Texas Large 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 align 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	ARRAYS	CAPRES	CD	DELETE	FILES	HEADR	OUTDCB	SDATE	STOP	VOLTP
ALCP			X				X				
BLOCK DATA			X			X	X		X		
CLOAD			X			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				
OUTNET						X	X				
OUTRIP						X	X				
OUTTRE							X				
OUTWLT			X			X	X				
PATHCL		X	X			X					
PRPBLD			X								
PRPCTV						X					X
RTPFL						X				X	
RTPLT						X					
SELECT							X				
SUBFND											X
SUMEND						X	X				
SUMRY			X	X						X	
UPDTNT			X			X					
VREC					X	X	X				
WGTLD			X				X				
WRT								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 centroid connectors).
M	The number of iterations run in ASSIGN SELF-BALANCING 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 nondirectional assigned link volumes for links with nonzero C fields except for centroid connectors for iterations 1 through M.
SY	The sum of the products of the nondirectional assigned link volumes with the C fields except for centroid connectors for iterations 1 through M.
XX	The sum of the nondirectional assigned link volumes squared for the links with nonzero C fields except for centroid connectors for iterations 1 through M.

SUBROUTINE CLOAD AND LDSEL

Control Variable	Contents	Meaning
SEL	False	This is either a LOAD NETWORK run or an ASSIGN SELF-BALANCING run.
SEL	True	This is a LOAD SELECTED LINKS run.
OUT	False	Don't print loaded network.
OUT	True	Print loaded network.
RES	False	This is either a LOAD NETWORK or a LOAD SELECTED LINKS run.
RES	True	This is an ASSIGN SELF-BALANCING run.

Variable	Contents
NOVER	Number of assigned directional volumes plus turn movements saved which are greater than 32767.
ITRE	Tree data set unit number.
NETD	The unit number NETWORK.

Array	Contents
INDEX(I)	The index into array LINKS where the out direction links for node I start. The values in this half-word array are 16 bit unsigned integers.
LINKS(I)	This is the links array, each half-word is a link which contains three data items. Bit 0 of a link is the last link flag, bit 1 is the selected link flag, and bits 2 thru 15 are the Bnode number of the link.
VOL(I)	VOL(I) is the assigned directional volume for LINKS(I) if the volume is less than 32767. If the volume is greater than 32767 then VOL(I) bits 1 thru 15 are an index into array OVERF where the volume is stored and bit 0 of VOL(I) is a one.
XRTRN(J)	XRTRN(J) is the index into array TRNTB of where the turn volumes saved for node J start. XRTRN(J) is an unsigned 16 bit positive integer.

Array	Contents
TRNTB(J)	TRNTB(J) is either a turn volume or a flag in bit 0 of the half-word of one and an index in bits 1 thru 15 into array OVERF where the turn volume is saved.
PATH	This is the array into which the packed path records are read. The path indices are also unpacked in this array and each index is placed in a separate byte by subroutine UNPKX. In the unpacked format PATH(I) contains path indices 4(I-1)+1 thru 4(I-1)+4.
OVERF	This is a full word array which contains both directional link volumes greater than 32767 and turn volumes saves which are greater than 32767.
BUF	This array is used to read the trip interchange items for one path record. Each interchange item has a volume in bits 0 thru 17 of the word and a destination zone in bits 18 thru 31 of the word.

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 used as the 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 nondirectionallink 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	$LOAD NETWORK control card read.
I	7	$BUILD TREES control card read.
I	8	$STOP control card read.
I	9	$LOAD SELECTED LINKS control card read.
I	10	$FRATAR FORECAST control card read.
I	11	$MERGE 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
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 16 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 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)
AO	Unit CTVOUT
NOSUB	Number of subnets

Array	Contents
TSUM (I,J)	I = subnet number, J = the relative zone in the subnet, TSUM 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 county 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 volumes 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 INITL1

Variable	Contents
NODES	Last node number

Array	Contents
PATH (I)	The turn code is contained in bits 3 thru 7 of PATH (I) for node I.

SUBROUTINE LNKLST

Variable	Contents
NA	The number of iterations, 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

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.

Variable	Contents
IV	Number of volume items in the last trip record read.
IFACT	First zone number minus 1.
NOVER	The number of assigned directional link volumes and saved turn volumes greater than 32767.

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. The indices are unsigned 16 bit binary numbers.
LINKS	This is the links array. Bit 0 is the last link flag, bit 1 is the selected link flag, and bits 2 thru 15 are the Bnode number.
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.
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.

Array	Contents
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 (I)	PATH (I) is the path index for node I. INDEX (I) + PATH (I) is the index into array LINKS for the link whose Bnode is the back node in the path to node I.
OVERF	This is a full word array used to store link volumes greater than 32767 and turn volumes greater than 32767.

SUBROUTINE MAIN

Control Variable	Contents	Meaning
I	Between 1 and 19	Indicates the last control card read. (See variable I in subroutine CRD).
RES	True	An ASSIGN SELF-BALANCING run is in iterations 1 thru 5.
CNVRG	False	If RES is true the "T" value of the last iteration run on ASSIGN SELF-BALANCING is greater than or equal 1.96.
CNVRG	True	If RES is true the "T" value of the last iteration run on ASSIGN SELF-BALANCING is less than 1.96.
CNT	False	The count field is to be used by ASSIGN SELF-BALANCING.
CNT	True	The capacity field is to be used by ASSIGN SELF-BALANCING.
W	False	A weighted assignment using weighted link impedance in ASSIGN SELF-BALANCING is not to be run.
W	True	A weighted assignment using weighted link impedances is to be run in ASSIGN SELF-BALANCING.

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

Variable	Contents
S	The constant 0.75 in the impedance update function.
Q	A constant used in the impedance update function.
J	Is the iteration number in an ASSIGN SELF-BALANCING run.
IMIN	Is the minimum number of iterations to run in ASSIGN SELF-BALANCING.
IMAX	Is the maximum number of iterations to run in ASSIGN SELF-BALANCING.
N	A constant used in the impedance update function.

SUBROUTINE MOORE (Control Section MOOR)

Variable	Contents
H	Home zone number to build a tree from.
NODES	Last node number in the network.

Array	Contents
TP	This array contains 4 words whose contents are 0, turn penalty, turn penalty, 0.
LINK	Each word of this array contains one directional link. The structure of a link word is described in a table which follows.
INDEX (I)	The index into array LINK where the out direction links for node I start. The values in this half word array are used as 16 bit unsigned integers.
BACK (J)	Contains the index of the out link from node J whose Bnode is the previous node in the path to node J.
CUM (I)	Cummulative time to reach node I stored in half words. If node I is not reached the time is set to 327.67 minutes.
PRED	A list of the nodes in the tree in descending time order. This list is part of the sequence structure used in building a tree.
SUCC	A list of the nodes in the tree in ascending time order. This list is part of the sequence structure used in building a tree.
FIRST (I)	For time I, which has been taken modules 1024, FIRST (I) points to array SUCC where the nodes in the sequence table for time I begin.
LAST (I)	For time I, which has been taken modules 1024, LAST (I) points to array PRED where the nodes in the sequence table for time I end.
ARROWA (I)	Each element of this array is one byte long. ARROWA (I) contains the arrow flag of the last link which caused node I to be entered into the sequence table. ARROWA for the home zone contains 12 which keeps a turn penalty from being added to the nodes connected to the home zone.

LINK Word Data Structure

Displacement Bits	Length Bits	Contents
0	1	Last link from Anode Flag (1 if last link or dummy link).
1	1	shaft flag
2	1	arrow flag
3	3	Back index for Bnode (this is the number which must be added to INDEX (Bnode) to index the link which contains the Anode number of this link as its Bnode)
6	1	Bnode Centroid Flag (the Bnode is a centroid if this bit is 1).
7	1	Freeway link Flag.
8	10	Link impedance in units of 1/100 minutes (maximum link impedance = 10.23 minutes).
18	14	Bnode of Link

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 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.

Arrays

Array	Length	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	73343	Contains the sorted packed links array described in NEWNET.

SUBROUTINE 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 Errors detected in subroutines NEWNET, VREC, and MRGREC		

Array LINKS

Array LINKS is the array in which one-way 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	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
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.
OUT	False	Don't print the loaded network.
OUT	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.

Array	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 NODE. The half words in this array are used as 16 bit unsigned integers.
NODE	This array contains a link in each half word, the links are structures which contain 3 data items. Bit 0 is the last link flag, bit 1 is the selected link flag and bits 2 thru 15 are the Bnode.

Array	Contents
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.
VOL(I)	This is a half word array which has the same length as array NODE and element I contains either the assigned directional link volume for link NODE(I) or the index of where it is in array IOVER. 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 IOVER 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 OUTTRE

Variable	Contents
HOMEND	The home zone of the tree.
NODES	The last node number in the network.
LNE	Number of lines of tree data to be printed per page.
LINES	Total number of lines of tree data to print for tree HOMEND.
PAGES	Number of pages to print for tree HOMEND.

Array	Contents
PATH(J)	Contains the index of the out link from node J whose Bnode is the previous node in the path to node J.
LAMBDA(I)	Cumulative time to reach node I stored in half words. If node I is not reached the time is set to 327.67 minutes.
INDEX(I)	The index into array LINKS where the out direction links for node I start. The value in this array are used as 16 bit unsigned integers.
LINKS	Each word of this array contains one directional link. The structure of a link word is described in a table which follows.

Link Word Data Structure

Displacement Bits	Length Bits	Contents
0	1	Last link from Anode flag (1 if last link or dummy link).
1	1	Shaft flag
2	1	Arrow flag
3	3	Back index for Bnode (This is the number which must be added to INDEX (Bnode) to index the link which contains the Anode number of this link as its Bnode).

Link Word Data Structure (continued)

Displacement Bits	Length Bits	Contents
6	1	Bnode Centroid Flag (the Bnode is a centroid if this bit is 1).
7	1	Freeway link Flag.
8	10	Link impedance in units of $\frac{1}{100}$ minutes (maximum link impedance = 10.23 minutes).
18	14	Bnode of link

SUBROUTINE OUTWLT

Variable	Contents
WL	The dimension size of arrays OVOL and IVOL.
WT	The dimension size of array TVOL.
FS	The first node (or zone) of the segment of the loaded weighted network to be printed.
LS	The last node of the segment.
I	The last Anode number read from the NETWORK data set.
NET	The NETWORK unit number.
NNET	The NEWNET unit number.

Array	Contents
OVOL(I)	The out link volume array. The link volumes out from node FS start at index 1. All volumes in this array are multiplied by 100.
IVOL(I)	The in link volume array. The link volumes into node FS start at index 1. All volumes in this array are multiplied by 100.
TVOL(I)	The turn volumes multiplied by 100. The turn volumes for node FS start at index 1.

SUBROUTINE PATHCL

Control Variable	Contents	Meaning
COPY	False	All trees specified on the *TREE cards are to be built.
COPY	True	An old trees data set (unit 49) is to be read and each logical record is to be checked for errors. Trees on unit 49 which have valid data are to be copied and trees specified on the *TREE cards which are missing or have invalid data are to be rebuilt.
OUT(I,J)	False	The trees built with origins between INDX1(I,J) and INDX2(I,J) are not to be printed.
OUT(I,J)	True	The trees built with origins between INDX1(I,J) and INDX2(I,J) are to be printed.

Variable	Contents
NTREE	The unit number (49) from which old packed tree records which are to be copied are read.
ITRE	The unit number (50) on which packed tree records are written.
ISKM	Unit SEPARAT number.
NETD	Unit NETWORK number.
LPACK	The number of words of packed path indices in a packed tree record.

Array	Contents
TRNPTY	This array contains 4 words whose contents are 0, turn penalty, turn penalty, 0.
T(I)	A full work array into which the cumulative time to reach node I in hundredths of a minute are stored to write a separation record from.

Array	Contents
B(I)	B(I) contains the index of the out link from node I whose Bnode is the previous node in the path to node I. This array is equivalenced to array BACK and the array name BACK is actually used in all calling sequences. After subroutine TRPCKM is called the indices are packed into 10 per word (see packed path word Data Structure below).
BACK(J)	This array is equivalenced to array B, but it is a full word array.
CUM(I)	Cumulative time to reach node I stored in half words after subroutine MOORE is called, If node I has not been reached the time is 327.67 minutes.
PRED(I)	A scratch array used by subroutine MOORE.
SUCC(I)	A scratch array used by subroutine MOORE.
ARROW(I)	A scratch array used by subroutine MOORE.
INDEX1(I)	The index into array LINKS1 where the out direction links for node I start. The values in this half word array are used as 16 bit unsigned integers. This allows the array to index up to 65534 one-way links in a network.
LINKS1	Each word of this array contains one directional link (see LINKS1 word Data Structure below).
COUNT(I)	The number of ranges of trees to be built for subnet I.
INDX1(I,J)	The first zone number of the J'th range of trees to build for subnet I.
INDX2(I,J)	The last zone number of the J'th range of trees to build for subnet I.

Packed Path Word Data Structure

Displacement Bits	Length Bits	Contents
0	2	Not used
2	3	Path index for node 10(J-1) + 1
5	3	Path index for node 10(J - 1) + 2
8	3	Path index for node 10(J - 1) + 3

Displacement Bits	Length Bits	Contents
11	3	Path index for node 10(J - 1) + 4
14	3	Path index for node 10(J - 1) + 5
17	3	Path index for node 10(J - 1) + 6
20	3	Path index for node 10(J - 1) + 7
23	3	Path index for node 10(J - 1) + 8
26	3	Path index for node 10(J - 1) + 9
29	3	Path index for node 10(J - 1) + 10

LINKS1 Word Data Structure

Displacement Bits	Length Bits	Contents
0	1	Last Link from Anode Flag (1 if last link or dummy link)
1	1	Shaft flag
2	1	Arrow flag
3	3	Back index for Bnode (This is the number which must be added to INDEX (BNODE) to index the link which contains the Anode number as its BNODE.)
6	1	Bnode Centroid Flag (the Bnode is a centroid if this bit is 1).
7	1	Freeway link Flag.
8	10	Link impedance in units of $\frac{1}{100}$ minutes (maximum link impedance = 10.23 minutes).
18	14	Bnode of Link

SUBROUTINE PRPBLD

Control Variable	Contents	Meaning
RES	False	This is a $BUILD TREES run.
RES	True	This is a $ASSIGN SELF-BALANCING run.
CAPC	False	The Link Count field is to be used in ASSIGN SELF-BALANCING.
W	False	A weighted assignment using weighted link impedances should not be run.
W	True	A weighted assignment should be run from weighted link impedances.
OUT(I,J)	False	The trees built with origins between INDX1(I,J) and INDX2(I,J) are not to be printed.
OUT(I,J)	True	The trees built with origins between INDX1(I,J) and INDX2(I,J) are to be printed.
TYPE	'COPY'	An old path data set is to be read, checked for errors, and paths with no errors are to be copied. Missing paths and those with errors are to be rebuilt.

Variable	Contents
NOSUB	Number of subnets.

Array	Contents
INDX1(I,J)	The first zone number of the J'th range of trees to build for subnet I.
INDX2(I,J)	The last zone number of the J'th range of trees to build for subnet I.
COUNT(I)	The number of ranges of trees to be built for subnet I.
TRNPTY	This array contains 4 words whose contents are 0, turn penalty, turn penalty, 0.

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	13335	This is the maximum number of one-way links in core.
MAXNDS	16000	This is the maximum last node number.
MAXLNK	40000	This is the maximum number of one-way links for the network.
MAXTIM	1023	This is the maximum link time in hundredths of a minute (i.e., 10.23 minutes).

Arrays

Name	Length	Contents
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	73343	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. The indices are used as 16 bit positive integers.
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	3	Back index for Bnode
6	1	Bnode Centroid Flag
7	1	Freeway link flag
8	10	Link impedance in units of 0.01 minutes.
18	14	Bnode of the link

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 SUMRY

Control Variable	Contents	Meaning
RES	False	This is either a LOAD NETWORK run or a LOAD SELECTED LINKS run or the weighted assignment of an ASSIGN SELF-BALANCING run.
RES	True	This is an ASSIGN SELF-BALANCING run in iteration 1 thru 5, call subroutine ALCP
CNVRG	False	The "T" test value for iteration ITER is greater than or equal to 1.96.
CNVRG	True	The "T" test value for iteration ITER is less than 1.96.
SUM	False	A weighted assignment is not to be calculated by subroutine GTLD.
SUM	True	A weighted assignment is to be calculated by subroutine GTLD and written on unit NEWNET.

Variable	Contents
ITER	The iteration number if an ASSIGN SELF-BALANCING is being run, otherwise 1.
LASTJ	The largest jurisdiction code in the network
L	The number of assignments which are on unit NETWORK.

Array	Contents
VMI(J,L)	Vehicle miles cross classified by jurisdiction + 1 used as the first index and three link classes used as the 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.

Array	Contents
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 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 links with a nonzero capacity field cross classified the same as VM.
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 capacity 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 volumes 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;

Array	Contents
	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.
XX(J)	Sum of the nondirectional link volumes multiplied by the count (or capacity) 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 squared count or (squared capacity) fields.
FN2	The number of nonzero count (or capacity) fields for links which are not centroid connectors.

SUBROUTINE SVLOAD

Variable	Contents
NSG	This is the number of segments that the loaded network must be summed in.
NLKF	The index of the first link volume or turn volume of a group of 4000 volumes or less for one record.
ITER	The iteration number.
NODES	The last node number in the network.

Array	Contents
BL(I)	The number of blocks of links (or records) for segment I.
WL(I)	The number of link volumes for segment I.
BT(I)	The number of blocks of turn volumes (or records) for segment I.
WT(I)	The number of turn volumes for segment I.
FS(I)	The first node number of segment I.
LS(I)	The last node number of segment I.
INDEX(I)	The sixteen bit unsigned integer at INDEX(I) is the index into array LINKS where the links out from node I start.
IPATH(I)	The eight bit byte at index I contains in bit 0 a centroid flag, in bit 1 a freeway flag, and in bits 2 thru 7 the turn code for node I.
LINKS(I)	This array contains links. Bit 0 of a link is the last link flag, bit 1 is the selected link flag, and bits 2 thru 15 are the Bnode of the link. The Anode of the link is the index used to index array INDEX.
VOL(I)	VOL(I) is the assigned directional volume for link LINKS(I) if the volume is less than 32767. If the volume is greater than 32767 the VOL(I) bits 1 thru 15 are an index into array IOVER where the volume is stored and bit 0 of VOL(I) is a one.

Array	Contents
XRTRN(J)	XRTRN(J) is the index into array ITR where the turn volumes saved for node J start. XRTRN(J) is an unsigned 16 bit positive integer.
ITR(J)	ITR(J) is either a turn volume or a flag in bit 0 of the half word of 1 and an index in bits 1 thru 15 into array IOVER where the turn volume is saved.
IOVER	This is a full word array which contains both directional link volumes greater than 32767 and turn volumes saved which are greater than 32767.
BUF	This is an array of 4000 words used to group volumes to write in one record.

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
TRNCD	Contains the turn code for NODE.
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.
INDXT	This is the index into array TVOL where the turn volumes for NODE start.

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.
TVOL	This array contains the turn volumes saved, they are indexed by variable INDXT.

Array	Contents
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.

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
ISDP5	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.
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 a path index for node I.

Array	Contents
INDEX(I)	This array contains the FORTRAN type index indicating the location where the links from node I begin in array LINKS. The half words are used as 16 bit unsigned integers.
LINKS	This array contains a link in each half word. Bit 0 is the last link flag, bit 1 is the selected link flag, and bits 2 through 15 are the Bnode number.
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 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.
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.

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 an *END card is read.
IMPD	False	An *IMPEDANCE parameter card has not 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	73343	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 WTSGLN

Variable	Contents
ITER	Number of iteration runs in an ASSIGN SELF-BALANCING run which are to be used to produce a weighted assignment.
ISG	The segment number of the network which is being summed in core.
NSG	The number of segments the network is broken into.
IN	The index into array VOLS where the out link volumes are to be summed.
IO	The index into array VOLS where the in link volumes are to be summed.
IT	The index into array VOLS where the turn volumes are to be summed.
NSPL	The number of link volume records to skip to reach the link volume records for segment ISG for both the out link volumes and the in link volumes.
NSAL	The number of link volume records which must be skipped to reach the end of either type of link volume records after the link volume records for segment ISG have been read.
NSPT	The number of turn volume records which must be skipped to reach the turn volume records for segment ISG.
NSAT	The number of turn volume records to skip after reading the turn volume records for segment ISG to reach the end of the records written for one assignment on unit 3.
IWT	The percent weight for iteration ISG.
NS	The number of records to skip on unit 3 at various places in subroutine WTSGLN.

Array	Contents
WTG(I)	The weight to use for iteration I expressed as a number between 0 and 1.0.
VOLS(I)	The full word array in which all weighted volumes for a segment are summed.

Array	Contents
BL(I)	The number of blocks of links (or records) for segment I.
WL(I)	The number of link volumes for segment I.
BT(I)	The number of blocks of turn volumes (or records) for segment I.
WT(I)	The number of turn volumes for segment I.
FS(I)	The first node number of segment I.
LS(I)	The last node number of segment I.
BUF	This is an array of 4000 words which is used to read link volume records and turn volume records.

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 Large 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 eleven basic formats associated with data sets used by the package. These eleven 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
Paths Data Set	A
Calcomp Plot Tape	P
Route Data Set	R
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	Paths	Paths
(Default) Unit Number	5	10	8	16	*	*	1	25	9	20	3	4	17	11	12	13	**	**	49	50
PREPARE NETWORK	I						0				I/O	I/O		I/O						
ASSEMBLE NETWORK	I						0				I/O	I/O		I/O						
REVISE NETWORK	I						0				I/O	I/O		I/O	I	I/O				
OUTPUT NETWORK							I													
DELETE ASSIGNMENTS							0								I					
PREPARE TRIP VOLUMES		I	0																	
OUTPUT TRIP VOLUMES			I																	
BUILD TREES							I			0									I	0
LOAD NETWORK			I				I	I/O	0											I
ASSIGN SELF-BALANCING			I				I/O	I/O	I/O	0	I/O									I/O
LOAD SELECTED LINKS			I				I	I/O	0								0			I
PLOT ROUTE PROFILES								I										0		
FRATAR FORECAST***			I	I/O									I/O							
SUM TRIP ENDS			I																	
MERGE					0	I														

I = Input Data Set

0 = Output Data Set

* 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	Paths	Paths
(Default) Unit Number	10	8	16	**	**	1	25	9	20	3	4	17	11	12	13	***	***	49	50
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									A	A
LOAD NETWORK		T				F	R	F	I										A
ASSIGN SELF-BALANCING		T				F	R	F	I	Z									A
LOAD SELECTED LINKS		T				F	R	F								L			A
PLOT ROUTE PROFILES							R										P		
FRATAR FORECAST****		T	T									T							
SUM TRIP ENDS		T																	
MERGE				T	T														

** 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 eleven formats is associated with the data set of interest.
- Refer to the detailed description of the format.

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

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

PATHS DATA SET

(Format Type Code: A)

Parameter Record

Displacement Bytes	Length Bytes	Contents
0	4	The number of subnets in the network.
4	4	The last node number in the network.
8	4	The number of words of path indices in a path record.
12	4	The first zone number in subnet 1.*
16	4	The last zone number in subnet 1.*
⋮	⋮	

Path Record

Displacement Bytes	Length Bytes	Contents
0	4	Home zone of the tree.
4	4	Path indices for nodes 1 thru 10.**
8	4	Path indices for nodes 11 thru 20.**
12	4	Path indices for nodes 21 thru 30.**
16	4	Path indices for nodes 31 thru 40.**
⋮	⋮	
4K	4	Path indices for nodes 10(K - 1) + 1 thru 10(K - 1) + 10.**

*These two items are repeated for each subnet.
**The first two bits of each word containing ten path indices are not used.

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 Bytes	Displacement Bits	Length Bytes	Length 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 Bytes	Displacement Bits	Length Bytes	Length Bits	Contents
0	0	0	1	Last Link from Anode flag
0	1	0	1	Shaft flag 0 = one direction 1 = other direction
0	2	0	1	Arrow flag 0 = one direction 1 = other direction

Displacement		Length		
Bytes	Bits	Bytes	Bits	Contents
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 Eliminaotr 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 32,767 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.

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.

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 zone.

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. These 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 Bytes	Displacement Bits	Length Bytes	Length Bits	Contents
12	0	0	10	Link distance in units of 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 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)

Parameter Record

Displacement Bytes	Length Bytes	Contents
0	4	The number of segments that the network is divided into.
4	4	The number of link records for segment 1.
8	4	The number of link volumes for segment 1.
12	4	The number of turn volume records for segment 1.
16	4	The number of turn volumes for segment 1.
20	4	The first node in segment 1.
24	4	The last node in segment 1.
.	.	
.	.	
.	.	

The last six items are repeated in the above order for the other segments.

Volumes Record**

Displacement Bytes	Length Bytes	Contents
0	4	A volume
.	.	.
.	.	.
.	.	.
4 (K-1)	4	A volume

** The volume records contain from one to 4000 volumes.

All of the volume records for one assignment are grouped together. Within the volume records written for one subnet, the records are in the order of out volume records, in volume records, and turn volume records. Each type of volume records is further divided by segment. Each record for each type is 4000 words long unless it must be shorter because of either a segment boundary or the end of the network.

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 E35 exit in the IBM sort using the E35 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

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

Select Record (continued)

Bytes Displacement	Length in Bytes	Contents
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

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 volumes 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	Directonal 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 $LOAD NETWORK AND
$ASSIGN SELF-BALANCING

TURNING MOVEMENTS

PRINTED OUTPUT FROM $LOAD NETWORK 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 $LOAD NETWORK. 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 $LOAD NETWORK

OUTPUT	$ASSIGN SELF-BALANCING						$LOAD NETWORK
	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. Jurisdictional/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 Large 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 ro 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; (8) The matrix is searched by columns and if a column 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 Large 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 these cases with one or more one-way links.

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.

NETWORK SEGMENTS

When the ASSIGN SELF-BALANCING process is run the loaded network is broken into from 1 to 4 segments by the subroutine SVLOAD when it is written on unit 3. The purpose of the segmentation is so that the weighted loaded network may be summed in 75,000 full words. There is no indication of where the segments are in the output of the weighted loaded network.

RECENT CHANGES

AND MODIFICATIONS